conda

Release 26.11.0.dev47

Anaconda, Inc.

Nov 13, 2026

CONTENTS

1 Install 3

2 New to conda? 5

3 Other useful resources 7

4 Contributors welcome 9

Python Module Index 999

Index 1003

Welcome to conda's documentation! Conda provides package, dependency, and environment management for any language. Here, you will find everything you need to get started using conda in your own projects.

CHAPTER

ONE

INSTALL

We recommend the following conda distributions to install conda:

Miniconda Miniconda is an installer by Anaconda that comes preconfigured for use with the Anaconda Repository. See the notes about Anaconda's *Terms of Service*.

Windows x86_64

Windows arm64

macOS arm64 (Apple Silicon)

macOS x86_64 (Intel)

Linux x86_64 (amd64)

Linux aarch64 (arm64)

Or with Homebrew:

```
brew install miniconda
```

Miniforge Miniforge is an installer maintained by the conda-forge community that comes preconfigured for use with the conda-forge channel.

Windows x86_64

Windows arm64

macOS arm64 (Apple Silicon)

macOS x86_64 (Intel)

Linux x86_64 (amd64)

Linux aarch64 (arm64)

Or with Homebrew:

```
brew install miniforge
```

CHAPTER

TWO

NEW TO CONDA?

If you are new to conda, we first recommend the following articles:

Getting started guide Learn the basics of using conda such as creating and adding packages to environments

Getting started with conda Managing environments Learn more about environments and best practices for using them in your projects

Managing environments

See also

Want to get even more in-depth training on how to use conda for free? Check out Anaconda's free course on conda basics.

CHAPTER

THREE

OTHER USEFUL RESOURCES

Command reference Full reference for all standard commands and options

Commands Cheatsheets Get the latest cheatsheet for commonly used commands

Cheatsheet Configuring conda Learn about the various ways conda's behavior can be configured

Using the .condarc conda configuration file Conda Blog Read the latest news, release announcements, and in-depth articles from the conda community

https://conda.org/blog/

CHAPTER

FOUR

CONTRIBUTORS WELCOME

Conda is an open source project and always welcomes new contributions. Please read the following guides to get started developing conda and making your own contributions.

Contributing 101 Learn more about how the conda project is managed and how to contribute

Contributing to conda Development environment Follow this guide to get your own development environment set up

Development Environment

4.1 User guide

4.1.1 Getting started with conda

Conda is a powerful command line tool for package and environment management that runs on Windows, macOS, and Linux.

This guide to getting started with conda goes over the basics of starting up and using conda to create environments and install packages.

Tip

Anaconda Navigator is a graphical desktop application that enables you to use conda without having to run commands at the command line.

See Getting started with Anaconda Navigator to learn more.

Before you start

To bootstrap a `conda` installation, use a minimal installer such as Miniconda or Miniforge.

Conda is also included in the Anaconda Distribution.

Note

Miniconda and Anaconda Distribution come preconfigured to use the Anaconda Repository and installing/using packages from that repository is governed by the Anaconda Terms of Service, which means that it *might* require a commercial fee license. There are exceptions for individuals, universities and companies with fewer than 200 employees (as of September 2024).

Please review the terms of service, Anaconda's most recent Update on Anaconda's Terms of Service for Academia and Research, and the Anaconda Terms of Service FAQ to answer your questions.

Starting conda

Conda is available on Windows, macOS, or Linux and can be used with any terminal application (or shell).

Windows

1. Open either the Anaconda or Miniforge Command Prompt (cmd.exe). A PowerShell prompt is also available with Anaconda Distribution or Miniconda.

macOS

1. Open Launchpad.
2. Open the Other application folder.
3. Open the Terminal application.

Linux

Open a terminal window.

Creating environments

Conda allows you to create separate environments, each containing their own files, packages, and package dependencies. The contents of each environment do not interact with each other.

The most basic way to create a new environment is with the following command:

```
conda create --name <env-name>
```

To add packages while creating an environment, specify them after the environment name:

```
conda create --name myenvironment python numpy pandas
```

For more information on working with environments, see *Managing environments*.

Listing environments

To see a list of all your environments:

```
conda info --envs
```

A list of environments appears, similar to the following:

```
conda environments:

   base           /home/username/Anaconda3
   myenvironment   * /home/username/Anaconda3/envs/myenvironment
```

Tip

The active environment is the one with an asterisk (*).

To change your current environment back to the default one:

```
conda activate
```

> **Tip**
>
> When the environment is deactivated, its name is no longer shown in your prompt, and the asterisk (*) returns to the default env. To verify, you can repeat the `conda info --envs` command.

Installing packages

You can also install packages into a previously created environment. To do this, you can either activate the environment you want to modify or specify the environment name on the command line:

```
# via environment activation
conda activate myenvironment
conda install matplotlib

# via command line option
conda install --name myenvironment matplotlib
```

For more information on searching for and installing packages, see *Managing packages*.

Specifying channels

Channels are locations (on your own computer or elsewhere on the Internet) where packages are stored. By default, conda searches for packages in its *default channels*.

If a package you want is located in another channel, such as conda-forge, you can manually specify the channel when installing the package:

```
conda install conda-forge::numpy
```

You can also override the default channels in your *.condarc* file. For a direct example, see *Channel locations (channels)* or read the entire *Using the .condarc conda configuration file*.

> **Tip**
>
> Find more packages and channels by searching Anaconda.org.

Updating conda

To see your conda version, use the following command:

```
conda --version
```

No matter which environment you run this command in, conda displays its current version:

```
conda 26.11.0.dev47
```

> **Note**
>
> If you get an error message `command not found: conda`, close and reopen your terminal window and verify that you are logged into the same user account that you used to install conda.

First, change your current environment back to the default one:

```
conda activate
```

Then update conda to the latest version:

```
conda update conda
```

Conda compares your version to the latest available version and then displays what is available to install.

Tip

We recommend that you always keep conda updated to the latest version. For conda's official version support policy, see CEP 10.

More information

- *Conda cheat sheet*
- Full documentation
- Free community support

4.1.2 Installing conda

To install conda, you must first pick the right installer for you. The following are the most popular installers currently available:

Miniconda
: Miniconda is a minimal installer provided by Anaconda. Use this installer if you want to install most packages yourself.

Anaconda Distribution
: Anaconda Distribution is a full featured installer that comes with a suite of packages for data science, as well as Anaconda Navigator, a GUI application for working with conda environments.

Miniforge
: Miniforge is an installer maintained by the conda-forge community that comes preconfigured for use with the conda-forge channel. To learn more about conda-forge, visit their website.

Note

Miniconda and Anaconda Distribution come preconfigured to use the Anaconda Repository and installing/using packages from that repository is governed by the Anaconda Terms of Service, which means that it *might* require a commercial fee license. There are exceptions for individuals, universities, and companies with fewer than 200 employees (as of September 2024).

Please review the terms of service, Anaconda's most recent Update on Anaconda's Terms of Service for Academia and Research, and the Anaconda Terms of Service FAQ to answer your questions.

System requirements

- A supported operating systems: Windows, macOS, or Linux
- For Miniconda or Miniforge: 400 MB disk space
- For Anaconda: Minimum 3 GB disk space to download and install

> **Tip**
>
> You do not need administrative or root permissions to install conda if you select a user-writable install location (e.g. `/Users/my-username/conda` or `C:\Users\my-username\conda`).

Regular installation

Follow the instructions for your operating system:

- *Windows*
- *macOS*
- *Linux*

Installing in silent mode

You can use *silent installation* of Miniconda, Anaconda, or Miniforge for deployment or testing or building services, such as GitHub Actions.

Follow the silent-mode instructions for your operating system:

- *Windows*
- *macOS*
- *Linux*

Cryptographic hash verification

SHA-256 checksums are available for Miniconda and Anaconda Distribution. We do not recommend using MD5 verification as SHA-256 is more secure.

Download the installer file and, before installing, verify it as follows:

- Windows:
  - If you have PowerShell V4 or later:

    Open a PowerShell console and verify the file as follows:

    ```
    Get-FileHash filename -Algorithm SHA256
    ```

  - If you don't have PowerShell V4 or later:

    Use the free online verifier tool on the Microsoft website.

    1. Download the file and extract it.
    2. Open a Command Prompt window.
    3. Navigate to the file.
    4. Run the following command:

       ```
       Start-PsFCIV -Path C:\path\to\file.ext -HashAlgorithm SHA256 -Online
       ```

- macOS: In iTerm or a terminal window enter `shasum -a 256 filename`.
- Linux: In a terminal window enter `sha256sum filename`.

Installing on Windows

1. Download the installer:
   - Miniconda installer for Windows
   - Anaconda Distribution installer for Windows
   - Miniforge installer for Windows
2. *Verify your installer hashes*.
3. Double-click the `.exe` file.
4. Follow the instructions on the screen.

   If you are unsure about any setting, accept the defaults. You can change them later.

   When installation is finished, from the **Start** menu, open either the Anaconda Command Prompt (cmd.exe) if using Miniconda or Anaconda Distribution, and the Miniforge Command Prompt if using Miniforge. Powershell prompts are also available.
5. Test your installation. In your terminal window, run the command `conda list`. A list of installed packages appears if it has been installed correctly.

Installing in silent mode

> **Note**
>
> The following instructions are for Miniconda but should also work for the Anaconda Distribution or Miniforge installers.

> **Note**
>
> As of `Anaconda Distribution 2022.05` and `Miniconda 4.12.0`, the option to add Anaconda to the PATH environment variable during an **All Users** installation has been disabled. This was done to address a security exploit. You can still add Anaconda to the PATH environment variable during a **Just Me** installation.

To run the the Windows installer for Miniconda in *silent mode*, use the `/S` argument. The following optional arguments are supported:

- `/InstallationType=[JustMe|AllUsers]`---Default is `JustMe`.
- `/AddToPath=[0|1]`---Default is `0`
- `/RegisterPython=[0|1]`---Make this the system's default Python. `0` indicates Python won't be registered as the system's default. `1` indicates Python will be registered as the system's default.
- `/S`---Install in silent mode.
- `/D=<installation path>`---Destination installation path. Must be the last argument. Do not wrap in quotation marks. Required if you use `/S`.

All arguments are case-sensitive.

Example: The following command installs Miniconda for the current user without registering Python as the system's default:

```
start /wait "" Miniconda3-latest-Windows-x86_64.exe /InstallationType=JustMe /
↪RegisterPython=0 /S /D=%UserProfile%\Miniconda3
```

Updating conda

1. Open Command Prompt or PowerShell from the start menu.
2. Run `conda update conda`.

Uninstalling conda

1. In the Windows Control Panel, click Add or Remove Program.
2. Select Python X.X (Miniconda), where X.X is your version of Python.
3. Click Remove Program.

Note

Removing a program is different in Windows 10.

Installing on macOS

Caution

If you use the `.pkg` installer for Miniconda, beware that those installers may skip the "Destination Select" page which will cause the installation to fail. If the installer skips this page, click "Change Install Location..." on the "Installation Type" page, choose a location for your install, and then click Continue.

1. Download the installer:
   - Miniconda installer for macOS.
   - Anaconda Distribution installer for macOS.
   - Miniforge installer for macOS.
2. *Verify your installer hashes*.
3. Install:
   - Miniconda or Miniforge: in your terminal window, run:

     ```
     bash <conda-installer-name>-latest-MacOSX-x86_64.sh
     ```

   - Anaconda Distribution: double-click the `.pkg` file.
4. Follow the prompts on the installer screens. If you are unsure about any setting, accept the defaults. You can change them later.
5. To make the changes take effect, close and then re-open your terminal window.
6. To verify your installation, in your terminal window, run the command `conda list`. A list of installed packages appears if it has been installed correctly.

Installing in silent mode

> **Note**
>
> The following instructions are for Miniconda but should also work for the Anaconda Distribution or Miniforge installers.

To run the *silent installation* of Miniconda for macOS or Linux, specify the -b and -p arguments of the bash installer. The following arguments are supported:

- `-b`: Batch mode with no PATH modifications to shell scripts. Assumes that you agree to the license agreement. Does not edit shell scripts such as `.bashrc`, `.bash_profile`, `.zshrc`, etc.
- `-p`: Installation prefix/path.
- `-f`: Force installation even if prefix `-p` already exists.

Example

```
wget https://repo.anaconda.com/miniconda/Miniconda3-latest-MacOSX-x86_64.sh -O ~/
↪miniconda.sh
bash ~/miniconda.sh -b -p $HOME/miniconda
```

> **Note**
>
> In order to initialize after the installation process is done, first run `source <path to conda>/bin/activate` and then run `conda init --all`.

Updating Anaconda or Miniconda

1. Open a terminal window.
2. Run `conda update conda`.

Uninstalling Anaconda or Miniconda

1. Open a terminal window.
2. Remove the entire Miniconda install directory with (*this may differ depending on your installation location*)

   ```
   rm -rf ~/miniconda
   ```

3. *Optional*: run `conda init --reverse --all` to undo changes to shell initialization scripts
4. *Optional*: remove the following hidden file and folders that may have been created in the home directory:
   - `.condarc` file
   - `.conda` directory
   - `.continuum` directory

   By running:

   ```
   rm -rf ~/.condarc ~/.conda ~/.continuum
   ```

Installing on Linux

1. Download the installer:
   - Miniconda installer for Linux.
   - Anaconda Distribution installer for Linux.
   - Miniforge installer for Linux.
2. *Verify your installer hashes*.
3. In your terminal window, run:

   ```
   bash <conda-installer-name>-latest-Linux-x86_64.sh
   ```

   `conda-installer-name` will be one of "Miniconda3", "Anaconda", or "Miniforge3".
4. Follow the prompts on the installer screens. If you are unsure about any setting, accept the defaults. You can change them later.
5. To make the changes take effect, close and then re-open your terminal window.
6. Test your installation. In your terminal window, run the command `conda list`. A list of installed packages appears if it has been installed correctly.

Using with fish shell

To use conda with fish shell, run the following in your terminal:

Add conda binary to $PATH, if not yet added:

```
fish_add_path <conda-install-location>/condabin
```

Configure fish-shell:

```
conda init fish
```

Installing in silent mode

See the instructions for *installing in silent mode on macOS*.

Updating conda

1. Open a terminal window.
2. Run `conda update conda`.

Uninstalling conda

1. Open a terminal window.
2. Remove the entire conda install directory with (*this may differ depending on your installation location*)

   ```
   rm -rf ~/conda
   ```
3. *Optional*: run `conda init --reverse --all` to undo changes to shell initialization scripts
4. *Optional*: remove the following hidden file and folders that may have been created in the home directory:
   - `.condarc` file

- `.conda` directory
- `.continuum` directory

By running:

```
rm -rf ~/.condarc ~/.conda ~/.continuum
```

4.1.3 Tasks

Managing conda

Verifying that conda is installed

To verify that conda is installed, in your terminal window, run:

```
conda --version
```

Conda responds with the version number that you have installed, such as `conda 4.12.0`.

If you get an error message, make sure of the following:

- You are logged into the same user account that you used to install Anaconda or Miniconda.
- You are in a directory that Anaconda or Miniconda can find.
- You have closed and re-opened the terminal window after installing conda.

Determining your conda version

In addition to the `conda --version` command explained above, you can determine what conda version is installed by running one of the following commands in your terminal window:

```
conda info
```

OR

```
conda -V
```

Updating conda to the current version

To update conda, in your terminal window, run:

```
conda update --name base conda
```

Conda compares versions and reports what is available to install. It also tells you about other packages that will be automatically updated or changed with the update. If conda reports that a newer version is available, type `y` to update:

```
Proceed ([y]/n)? y
```

Suppressing warning message about updating conda

To suppress the following warning message when you do not want to update conda to the latest version:

```
==> WARNING: A newer version of conda exists. <==
current version: 4.6.13
latest version: 4.8.0
```

Update conda by running: `conda update --name base conda`

Run the following command from your terminal: `conda config --set notify_outdated_conda false`

Or add the following line in your `.condarc` file: `notify_outdated_conda: false`

Managing environments

With conda, you can create, export, list, remove, and update environments that have different versions of Python and/or packages installed in them. Switching or moving between environments is called activating the environment. You can also share an environment file.

There are many options available for the commands described on this page. For a detailed reference on all available commands, see *commands*.

Creating an environment with commands

Use the terminal for the following steps:

1. To create an environment:

   ```
   conda create --name <my-env>
   ```

   Replace <my-env> with the name of your environment.

2. When conda asks you to proceed, type `y`:

   ```
   proceed ([y]/n)?
   ```

   This creates the myenv environment in `/envs/`. No packages will be installed in this environment.

3. To create an environment with a specific version of Python:

   ```
   conda create -n myenv python=3.9
   ```

4. To create an environment with a specific package:

   ```
   conda create -n myenv scipy
   ```

   or:

   ```
   conda create -n myenv python
   conda install -n myenv scipy
   ```

5. To create an environment with a specific version of a package:

   ```
   conda create -n myenv scipy=0.17.3
   ```

   or:

   ```
   conda create -n myenv python
   conda install -n myenv scipy=0.17.3
   ```

6. To create an environment with a specific version of Python and multiple packages:

   ```
   conda create -n myenv python=3.9 scipy=0.17.3 astroid babel
   ```

> **Tip**
>
> Install all the programs that you want in this environment at the same time. Installing one program at a time can lead to dependency conflicts.

To automatically install pip or another program every time a new environment is created, add the default programs to the *create_default_packages* section of your `.condarc` configuration file. The default packages are installed every time you create a new environment. If you do not want the default packages installed in a particular environment, use the `--no-default-packages` flag:

```
conda create --no-default-packages -n myenv python
```

> **Tip**
>
> You can add much more to the `conda create` command. For details, run `conda create --help`.

Creating an environment from an environment.yml file

Before you start, you need to use a text editor to create an environment file that looks like the following:

```
# environment.yml
name: myenv
channels:
  - conda-forge
dependencies:
  - python
  - numpy
```

The file above defines an environment named, "myenv" that uses `conda-forge` as its channel and adds the dependencies `python` and `numpy`.

Now that the `environment.yml` has been created, you can use the terminal for the following steps:

1. Create the environment from the `environment.yml` file:

   ```
   conda create --file environment.yml
   ```

   The first line of the `yml` file sets the new environment's name. For details see *Creating an environment file manually*.

   > **Hint**
   >
   > Creating environments from YAML files was previously done with `conda env create --file <file>`. This still works but is not the preferred way of creating environments. This older invocation will be kept around to ensure backwards compatibility with existing scripts and automations.

2. Activate the new environment: `conda activate myenv`
3. Verify that the new environment was installed correctly:

   ```
   conda env list
   ```

   You can also use `conda info --envs`.

Specifying a different target platform for an environment

By default, `conda` will create environments targeting the platform it's currently running on. You can check which platform you are currently on by running `conda info` and checking the `platform` entry.

However, in some cases you might want to create an environment for a different target platform or architecture. To do so, use the `--platform` flag available in the `conda create` and `conda env create` commands. See `--subdir, --platform` in *conda create* for more information about allowed values.

For example, a user running macOS on the Apple Silicon platform might want to create a `python` environment for Intel processors and emulate the executables with Rosetta. The command would be:

```
conda create --platform osx-64 --name python-x64 python
```

Note

You can't specify the `--platform` flag for existing environments. When created, the environment will be annotated with the custom configuration and subsequent operations on it will remember the target platform.

This flag also allows specifying a different OS (e.g. creating a Linux environment on macOS), but we don't recommend its usage outside of `--dry-run` operations. Common problems with mismatched OSes include:

- The environment cannot be solved because virtual packages are missing. You can workaround this issue by exporting the necessary `CONDA_OVERRIDE_****` environment variables. For example, solving for Linux from macOS, you will probably need `CONDA_OVERRIDE_LINUX=1` and `CONDA_OVERRIDE_GLIBC=2.17`.
- The environment can be solved, but extraction and linking fails due filesystem limitations (case insensitive systems, incompatible paths, etc). The only workaround here is to use `--dry-run --json` to obtain the solution and process the payload into a lockfile that can be shared with the target machine. See *Create explicit lockfiles without creating an environment* for more details.

Specifying a location for an environment

You can control where a conda environment lives by providing a path to a target directory when creating the environment. For example, the following command will create a new environment in a subdirectory of the current working directory called `envs`:

```
conda create --prefix ./envs jupyterlab=3.2 matplotlib=3.5 numpy=1.21
```

You then activate an environment created with a prefix using the same command used to activate environments created by name:

```
conda activate ./envs
```

Specifying a path to a subdirectory of your project directory when creating an environment has the following benefits:

- It makes it easy to tell if your project uses an isolated environment by including the environment as a subdirectory.
- It makes your project more self-contained as everything, including the required software, is contained in a single project directory.

An additional benefit of creating your project's environment inside a subdirectory is that you can then use the same name for all your environments. If you keep all of your environments in your `envs` folder, you'll have to give each environment a different name.

There are a few things to be aware of when placing conda environments outside of the default `envs` folder.

1. Conda can no longer find your environment with the `--name` flag. You'll generally need to pass the `--prefix` flag along with the environment's full path to find the environment.
2. Specifying an install path when creating your conda environments makes it so that your command prompt is now prefixed with the active environment's absolute path rather than the environment's name.

After activating an environment using its prefix, your prompt will look similar to the following:

```
(/absolute/path/to/envs) $
```

This can result in long prefixes:

```
(/Users/USER_NAME/research/data-science/PROJECT_NAME/envs) $
```

To remove this long prefix in your shell prompt, modify the env_prompt setting in your `.condarc` file:

```
conda config --set env_prompt '({name})'
```

This will edit your `.condarc` file if you already have one or create a `.condarc` file if you do not.

Now your command prompt will display the active environment's generic name, which is the name of the environment's root folder:

```
$ cd project-directory
$ conda activate ./env
(env) project-directory $
```

Updating an environment

You may need to update your environment for a variety of reasons. For example, it may be the case that:

- one of your core dependencies just released a new version (dependency version number update).
- you need an additional package for data analysis (add a new dependency).
- you have found a better package and no longer need the older package (add new dependency and remove old dependency).

If any of these occur, all you need to do is update the contents of your `environment.yml` file accordingly and then run the following command:

```
conda env update --file environment.yml  --prune
```

Note

The `--prune` option causes conda to remove any dependencies that are no longer required from the environment.

Freezing or locking an environment

CEP 22 introduced an environment marker file that will instruct `conda` not to allow modifications in the given environment. When attempting to add, update or remove a package, users will receive an error by default:

```
EnvironmentIsFrozenError: Cannot not modify '~/.conda/envs/my-env'.
The environment is marked as frozen. You can ignore this error with
the `--override-frozen` flag, at your own risk.
```

As mentioned above, users can pass the `--override-frozen` flag, but this is really not recommended and should only be done by advanced users who are aware of the risks and would have the knowledge to fix potential complications derived from that operation.

Cloning an environment

Use the terminal for the following steps:

You can make an exact copy of an environment by creating a clone of it:

```
conda create --name myclone --clone myenv
```

> **Note**
>
> Replace `myclone` with the name of the new environment. Replace `myenv` with the name of the existing environment that you want to copy.

To verify that the copy was made:

```
conda info --envs
```

In the environments list that displays, you should see both the source environment and the new copy.

Building identical conda environments

There are two ways to build identical conda environments, using explicit specification files and using lockfiles.

Explicit spec files are usually limited to a single platform, but lockfiles can support multiple platforms.

Multi-platform lockfiles

> **Note**
>
> Lockfile support is available in conda 26.5 and later. Use the following command to update:
>
> ```
> conda install --name base "conda>=26.5"
> ```

A lockfile contains the exact packages, versions, builds, and channels required to recreate an environment exactly. Because the lockfile captures a fully resolved state, conda can skip the solver step and go drectly to downloading and installing packages. Conda supports `conda-lock.yaml` and `pixi.lock` natively, and these lockfile types can be used to exactly recreate environments on Windows, Linux, and macOS (depending on package availability).

Create a lockfile with *conda export*:

```
conda export --name my-env --file conda-lock.yaml
```

Use the lockfile to create an identical environment on another machine:

```
conda create --name my-env --file conda-lock.yaml
```

Create a lockfile for different platforms:

```
conda export --name my-env --file conda-lock.yaml \
--platform linux-64 \
--platform osx-64 \
--platform win-64
```

For more information on conda lockfiles, see the conda-lockfiles GitHub documentation.

Explicit spec files

You can use explicit specification files to build an identical conda environment on the same operating system platform, either on the same machine or on a different machine.

Use the terminal for the following steps:

1. Run `conda list --explicit` to produce a spec list such as:

```
# This file may be used to create an environment using:
# $ conda create --name <env> --file <this file>
# platform: osx-64
@EXPLICIT
https://repo.anaconda.com/pkgs/free/osx-64/mkl-11.3.3-0.tar.bz2
https://repo.anaconda.com/pkgs/free/osx-64/numpy-1.11.1-py35_0.tar.bz2
https://repo.anaconda.com/pkgs/free/osx-64/openssl-1.0.2h-1.tar.bz2
https://repo.anaconda.com/pkgs/free/osx-64/pip-8.1.2-py35_0.tar.bz2
https://repo.anaconda.com/pkgs/free/osx-64/python-3.5.2-0.tar.bz2
https://repo.anaconda.com/pkgs/free/osx-64/readline-6.2-2.tar.bz2
https://repo.anaconda.com/pkgs/free/osx-64/setuptools-25.1.6-py35_0.tar.bz2
https://repo.anaconda.com/pkgs/free/osx-64/sqlite-3.13.0-0.tar.bz2
https://repo.anaconda.com/pkgs/free/osx-64/tk-8.5.18-0.tar.bz2
https://repo.anaconda.com/pkgs/free/osx-64/wheel-0.29.0-py35_0.tar.bz2
https://repo.anaconda.com/pkgs/free/osx-64/xz-5.2.2-0.tar.bz2
https://repo.anaconda.com/pkgs/free/osx-64/zlib-1.2.8-3.tar.bz2
```

2. To create this spec list as a file in the current working directory, run:

```
conda list --explicit > spec-file.txt
```

> **Note**
>
> You can use `spec-file.txt` as the filename or replace it with a filename of your choice.

An explicit spec file is not usually cross platform, and therefore has a comment at the top such as `# platform: osx-64` showing the platform where it was created. This platform is the one where this spec file is known to work. On other platforms, the packages specified might not be available or dependencies might be missing for some of the key packages already in the spec.

To use the spec file to create an identical environment on the same machine or another machine:

```
conda create --name myenv --file spec-file.txt
```

To use the spec file to install its listed packages into an existing environment:

```
conda install --name myenv --file spec-file.txt
```

The `--file` option also supports YAML and other formats via environment specifier plugins. For `conda install` (and `conda update`), only the package list is used; any name or prefix in the file is ignored. Packages are installed into the target environment (-n/-p or the current environment).

Conda does not check architecture or dependencies when installing from a spec file. To ensure that the packages work correctly, make sure that the file was created from a working environment, and use it on the same architecture, operating system, and platform, such as linux-64 or osx-64.

Activating an environment

Activating environments is essential to making the software in the environments work well. Activation entails two primary functions: adding entries to PATH for the environment and running any activation scripts that the environment may contain. These activation scripts are how packages can set arbitrary environment variables that may be necessary for their operation. You can also *use the config API to set environment variables*.

Activation prepends to PATH. This only takes effect when you have the environment active so it is local to a terminal session, not global.

Note

When installing Anaconda, you have the option to "Add Anaconda to my PATH environment variable." *This is not recommended* because it *appends* Anaconda to PATH. When the installer appends to PATH, it does not call the activation scripts.

Note

On Windows, PATH is composed of two parts, the *system* PATH and the *user* PATH. The system PATH always comes first. When you install Anaconda for "Just Me", we add it to the *user* PATH. When you install for "All Users", we add it to the *system* PATH. In the former case, you can end up with system PATH values taking precedence over your entries. In the latter case, you do not. *We do not recommend* multi-user installs.

To activate an environment: `conda activate myenv`

Note

Replace `myenv` with the environment name or directory path.

Conda prepends the path name `myenv` onto your system command.

You may receive a warning message if you have not activated your environment:

```
Warning:
This Python interpreter is in a conda environment, but the environment has
not been activated. Libraries may fail to load. To activate this environment
please see https://conda.io/activation.
```

If you receive this warning, you need to activate your environment. To do so on Windows, run: `c:\Anaconda3\Scripts\activate` in a terminal window.

Windows is extremely sensitive to proper activation. This is because the Windows library loader does not support the concept of libraries and executables that know where to search for their dependencies (RPATH). Instead, Windows relies on a dynamic-link library search order.

If environments are not active, libraries won't be found and there will be lots of errors. HTTP or SSL errors are common errors when the Python in a child environment can't find the necessary OpenSSL library.

Conda itself includes some special workarounds to add its necessary PATH entries. This makes it so that it can be called without activation or with any child environment active. In general, calling any executable in an environment without first activating that environment will likely not work. For the ability to run executables in activated environments, you may be interested in the `conda run` command.

Activating environments in shell scripts

The `conda activate` command is normally made available by shell initialization code added by `conda init`. Non-interactive shell scripts may not read the same shell startup files as an interactive terminal session, so `conda activate` can fail even when it works from the command line.

Initialize conda for the current shell before activating the environment:

Bash

```
eval "$(conda shell.bash hook)"
conda activate myenv
```

PowerShell

```
conda shell.powershell hook | Out-String | Invoke-Expression
conda activate myenv
```

Replace `myenv` with the environment name or directory path. If you only need to run a single command in an environment, you can use `conda run` instead of activating the environment in the script:

```
conda run -n myenv python script.py
```

If you experience errors with PATH, review our *troubleshooting*.

Conda init

Earlier versions of conda introduced scripts to make activation behavior uniform across operating systems. Conda 4.4 allowed `conda activate myenv`. Conda 4.6 added extensive initialization support so that conda works faster and less disruptively on a wide variety of shells (bash, zsh, csh, fish, xonsh, and more). Now these shells can use the `conda activate` command.

Alternatively, `conda init --condabin` will not install a shell function in your profile. Instead, it will only add the `$CONDA_PREFIX/condabin/` directory to `PATH`. This directory only contains the `conda` executable, so it should be minimally invasive.

For more information, read the output from `conda init --help`.

One setting may be useful to you when using `conda init` is:

```
auto_activate: bool
```

This setting controls whether or not conda activates the given default environment when it first starts up. You'll have the `conda` command available either way, but without activating the environment, none of the other programs in the default environment will be available until the environment is activated with `conda activate`.

The environment to be activated by default can be configured with:

```
default_activation_env: str
```

People sometimes choose this setting to speed up the time their shell takes to start up or to keep conda-installed software from automatically hiding their other software.

Nested activation

By default, `conda activate` will deactivate the current environment before activating the new environment and re-activate it when deactivating the new environment. Sometimes you may want to leave the current environment PATH entries in place so that you can continue to easily access command-line programs from the first environment. This is most commonly encountered when common command-line utilities are installed in the default environment. To retain the current environment in the PATH, you can activate the new environment using:

```
conda activate --stack myenv
```

If you wish to always stack when going from the outermost environment, which is typically the default environment, you can set the `auto_stack` configuration option:

```
conda config --set auto_stack 1
```

You may specify a larger number for a deeper level of automatic stacking, but this is not recommended since deeper levels of stacking are more likely to lead to confusion.

Environment variable for DLL loading verification

If you don't want to activate your environment and you want Python to work for DLL loading verification, then follow the *troubleshooting directions*.

Warning

If you choose not to activate your environment, then loading and setting environment variables to activate scripts will not happen. We only support activation.

Deactivating an environment

To deactivate an environment, type: `conda deactivate`

Conda removes the path name for the currently active environment from your system command.

Note

To simply return to the default environment, it's better to call `conda activate` with no environment specified, rather than to try to deactivate. If you run `conda deactivate` from your base environment, you may lose the ability to run conda at all. Don't worry, that's local to this shell - you can start a new one. However, if the environment was activated using `--stack` (or was automatically stacked) then it is better to use `conda deactivate`.

Determining your current environment

Use the terminal for the following steps.

By default, the active environment---the one you are currently using---is shown in parentheses () or brackets [] at the beginning of your command prompt:

```
(myenv) $
```

If you do not see this, run:

```
conda info --envs
```

In the environments list that displays, your current environment is highlighted with an asterisk (*).

By default, the command prompt is set to show the name of the active environment. To disable this option:

```
conda config --set changeps1 false
```

To re-enable this option:

```
conda config --set changeps1 true
```

Viewing a list of your environments

To see a list of all of your environments, in your terminal window, run:

```
conda info --envs
```

OR

```
conda env list
```

A list similar to the following is displayed:

```
conda environments:
myenv                  /home/username/miniconda/envs/myenv
snowflakes             /home/username/miniconda/envs/snowflakes
bunnies                /home/username/miniconda/envs/bunnies
```

If this command is run by an administrator, a list of all environments belonging to all users will be displayed.

The `--json` variants of these commands include per-environment details under `envs_details`, which can help identify environments that are no longer in use:

- `created` — when the environment was created.
- `last_modified` — when packages were last installed, updated, or removed.
- `last_activated` — when the environment was last activated via `conda activate` (including automated activations such as `conda run`).

All timestamps are ISO 8601 UTC and may be `null` when the information is unavailable. In particular, `last_activated` is recorded on a best-effort basis: it is tracked via the modification time of `<prefix>/conda-meta/last_activated`, activations on read-only or frozen environments are silently skipped, and environments that have not been activated since this tracking was introduced report `null`.

Viewing a list of the packages in an environment

To see a list of all packages installed in a specific environment:

- If the environment is not activated, in your terminal window, run:

  ```
  conda list -n myenv
  ```

- If the environment is activated, in your terminal window, run:

```
conda list
```

- To see if a specific package is installed in an environment, in your terminal window, run:

```
conda list -n myenv scipy
```

Using pip in an environment

To use pip in your environment, in your terminal window, run:

```
conda install -n myenv pip
conda activate myenv
pip <pip_subcommand>
```

Issues may arise when using pip and conda together. When combining conda and pip, it is best to use an isolated conda environment. Only after conda has been used to install as many packages as possible should pip be used to install any remaining software. If modifications are needed to the environment, it is best to create a new environment rather than running conda after pip. When appropriate, conda and pip requirements should be stored in text files.

We recommend that you:

Use pip only after conda

- Install as many requirements as possible with conda then use pip.
- Pip should be run with `--upgrade-strategy only-if-needed` (the default).
- Do not use pip with the `--user` argument, avoid all users installs.

Use conda environments for isolation

- Create a conda environment to isolate any changes pip makes.
- Environments take up little space thanks to hard links.
- Care should be taken to avoid running pip in the root environment.

Recreate the environment if changes are needed

- Once pip has been used, conda will be unaware of the changes.
- To install additional conda packages, it is best to recreate the environment.

Store conda and pip requirements in text files

- Package requirements can be passed to conda via the `--file` argument.
- Pip accepts a list of Python packages with `-r` or `--requirements`.
- Conda env will export or create environments based on a file with conda and pip requirements.

Setting environment variables

If you want to associate environment variables with an environment, you can use the config API. This is recommended as an alternative to using activate and deactivate scripts since those are an execution of arbitrary code that may not be safe.

First, create your environment and activate it:

```
conda create -n test-env
conda activate test-env
```

To list any variables you may have, run `conda env config vars list`.

To set environment variables, run `conda env config vars set my_var=value`.

Once you have set an environment variable, you have to reactivate your environment: `conda activate test-env`.

To check if the environment variable has been set, run `echo $my_var` (`echo %my_var%` on Windows) or `conda env config vars list`.

When you deactivate your environment, you can use those same commands to see that the environment variable goes away.

You can specify the environment you want to affect using the `-n` and `-p` flags. The `-n` flag allows you to name the environment and `-p` allows you to specify the path to the environment.

To unset the environment variable, run `conda env config vars unset my_var -n test-env`.

When you deactivate your environment, you can see that environment variable goes away by rerunning `echo my_var` or `conda env config vars list` to show that the variable name is no longer present.

Environment variables set using `conda env config vars` will be retained in the output of `conda env export`. Further, you can declare environment variables in the environment.yml file as shown here:

```
name: env-name
channels:
  - conda-forge
  - defaults
dependencies:
  - python=3.7
  - codecov
variables:
  VAR1: valueA
  VAR2: valueB
```

Saving environment variables

Conda environments can include saved environment variables.

Suppose you want an environment "analytics" to store both a secret key needed to log in to a server and a path to a configuration file. The sections below explain how to write a script named `env_vars` to do this on *Windows* and *macOS or Linux*.

This type of script file can be part of a conda package, in which case these environment variables become active when an environment containing that package is activated.

You can name these scripts anything you like. However, multiple packages may create script files, so be sure to use descriptive names that are not used by other packages. One popular option is to give the script a name in the form `packagename-scriptname.sh`, or on Windows, `packagename-scriptname.bat`.

Windows

1. Locate the directory for the conda environment in your terminal window by running in the command shell `%CONDA_PREFIX%`.
2. Enter that directory and create these subdirectories and files:

```
cd %CONDA_PREFIX%
mkdir .\etc\conda\activate.d
mkdir .\etc\conda\deactivate.d
type NUL > .\etc\conda\activate.d\env_vars.bat
type NUL > .\etc\conda\deactivate.d\env_vars.bat
```

3. Edit `.\etc\conda\activate.d\env_vars.bat` as follows:

```
set MY_KEY='secret-key-value'
set MY_FILE=C:\path\to\my\file
```

4. Edit `.\etc\conda\deactivate.d\env_vars.bat` as follows:

```
set MY_KEY=
set MY_FILE=
```

When you run `conda activate analytics`, the environment variables `MY_KEY` and `MY_FILE` are set to the values you wrote into the file. When you run `conda deactivate`, those variables are erased.

macOS and Linux

1. Locate the directory for the conda environment in your terminal window by running in the terminal `echo $CONDA_PREFIX`.
2. Enter that directory and create these subdirectories and files:

```
cd $CONDA_PREFIX
mkdir -p ./etc/conda/activate.d
mkdir -p ./etc/conda/deactivate.d
touch ./etc/conda/activate.d/env_vars.sh
touch ./etc/conda/deactivate.d/env_vars.sh
```

3. Edit `./etc/conda/activate.d/env_vars.sh` as follows:

```
#!/bin/sh

export MY_KEY='secret-key-value'
export MY_FILE=/path/to/my/file/
```

4. Edit `./etc/conda/deactivate.d/env_vars.sh` as follows:

```
#!/bin/sh

unset MY_KEY
unset MY_FILE
```

When you run `conda activate analytics`, the environment variables `MY_KEY` and `MY_FILE` are set to the values you wrote into the file. When you run `conda deactivate`, those variables are erased.

Sharing an environment

You may want to share your environment with someone else---for example, so they can re-create a test that you have done. To allow them to quickly reproduce your environment, with all of its packages and versions, give them a copy of your environment file.

The recommended way to export conda environments is using the `conda export` command, which supports multiple formats and provides enhanced functionality:

Exporting environments with conda export

The `conda export` command provides a modern, plugin-based approach to environment export with support for multiple output formats.

> **Note**
>
> The `conda export` command has been significantly enhanced in conda 25.7.x with a new plugin-based architecture, multiple export formats, and improved functionality.

Available Export Formats:

- `environment-yaml` - Cross-platform YAML format (default)
- `environment-json` - Cross-platform JSON format
- `explicit` - Platform-specific explicit URLs (CEP 23 compliant)
- `requirements` - Platform-specific requirements format

Basic Usage

Export the current environment to YAML format (default):

```
conda export > environment.yaml
```

Export a specific environment by name:

```
conda export --name myenv --format=environment-yaml
```

Export using automatic format detection:

```
conda export --file=environment.yaml    # Auto-detects YAML format
conda export --file=environment.json    # Auto-detects JSON format
conda export --file=explicit.txt        # Auto-detects explicit format
conda export --file=requirements.txt    # Auto-detects requirements format
```

Export Formats in Detail

Environment YAML Format (Recommended for sharing)

Best for cross-platform environment sharing:

```
conda export --format=environment-yaml --file=environment.yaml
```

Creates a file like:

```
name: myenv
channels:
  - conda-forge
  - defaults
dependencies:
```

(continues on next page)

(continued from previous page)

```
  - python=3.9
  - numpy=1.21.0
  - pandas=1.3.0
```

Environment JSON Format

For programmatic processing or integration with tools:

```
conda export --format=environment-json --file=environment.json
```

Explicit Format

For exact environment reproduction on the same platform:

```
conda export --format=explicit --file=explicit.txt
```

Creates a file with full package URLs:

```
@EXPLICIT
https://repo.anaconda.com/pkgs/main/osx-64/python-3.9.7-h88f2d9e_0.tar.bz2
https://repo.anaconda.com/pkgs/main/osx-64/numpy-1.21.0-py39h2e5f516_0.tar.bz2
```

Requirements Format

For conda-compatible requirements files:

```
conda export --format=requirements --file=requirements.txt
```

Creates a file like:

```
python=3.9.7
numpy=1.21.0
pandas=1.3.0
```

Cross-Platform Compatibility

For maximum cross-platform compatibility, use the `--from-history` flag:

```
conda export --from-history --format=environment-yaml
```

This exports only explicitly installed packages, excluding platform-specific dependencies.

> **Note**
>
> The `--from-history` flag only works with structured formats (`environment-yaml`, `environment-json`). It's not compatible with text formats (`explicit`, `requirements`).

Lockfiles also support multiple platforms:

```
conda export --name my-env --file conda-lock.yaml \
--platform linux-64 \
--platform osx-64 \
--platform win-64
```

For more information on conda lockfiles, see the conda-lockfiles GitHub documentation.

Alternative: Traditional conda env export

You can also use the traditional `conda env export` command for YAML-only export:

> **Note**
>
> If you already have an `environment.yml` file in your current directory, it will be overwritten during this task.

1. Activate the environment to export: `conda activate myenv`

   > **Note**
   >
   > Replace `myenv` with the name of the environment.

2. Export your active environment to a new file:

   ```
   conda env export > environment.yml
   ```

   > **Note**
   >
   > This file handles both the environment's pip packages and conda packages.

3. Email or copy the exported `environment.yml` file to the other person.

Exporting an environment file across platforms

If you want to make your environment file work across platforms, you can use the `--from-history` flag with either command:

```
# Enhanced export command (recommended)
conda export --from-history --format=environment-yaml

# Traditional command
conda env export --from-history
```

For example, if you create an environment and install Python and a package:

```
conda install python=3.7 codecov
```

This will download and install numerous additional packages to solve for dependencies. This will introduce packages that may not be compatible across platforms.

Both export commands with `--from-history` will export only the packages you specifically chose:

```
name: env-name
channels:
  - conda-forge
  - defaults
dependencies:
  - python=3.7
  - codecov
prefix: /Users/username/anaconda3/envs/env-name
```

> **Note**
>
> If you installed Anaconda 2019.10 on macOS, your prefix may be `/Users/username/opt/envs/env-name`.

Creating an environment file manually

You can create an environment file (`environment.yml`) manually to share with others.

EXAMPLE: A simple environment file:

```
name: stats
dependencies:
  - numpy
  - pandas
```

EXAMPLE: A more complex environment file:

```
name: stats2
channels:
  - javascript
dependencies:
  - python=3.9
  - bokeh=2.4.2
  - conda-forge::numpy=1.21.*
  - nodejs=16.13.*
  - flask
  - pip
  - pip:
    - Flask-Testing
```

> **Note**
>
> **Using wildcards**
>
> Note the use of the wildcard * when defining a few of the versions in the complex environment file. Keeping the major and minor versions fixed while allowing the patch to be any number allows you to use your environment file to get any bug fixes while still maintaining consistency in your environment. For more information on package installation values, see *Package search and install specifications*.
>
> **Specifying channels outside of "channels"**
>
> You may occasionally want to specify which channel conda will use to install a specific package. To accomplish this, use the *channel::package* syntax in *dependencies:*, as demonstrated above with *conda-forge::numpy* (version numbers optional). The specified channel does not need to be present in the *channels:* list, which is useful if you want some—but not *all*—packages installed from a community channel such as *conda-forge*.

You can exclude the default channels by adding `nodefaults` to the channels list.

```
channels:
  - javascript
  - nodefaults
```

This is equivalent to passing the `--override-channels` option to most `conda` commands.

Adding `nodefaults` to the channels list in `environment.yml` is similar to removing `defaults` from the *channels list* in the `.condarc` file. However, changing `environment.yml` affects only one of your conda environments while changing `.condarc` affects them all.

For details on creating an environment from this `environment.yml` file, see *Creating an environment from an environment.yml file*.

Restoring an environment

Conda keeps a history of all the changes made to your environment, so you can easily "roll back" to a previous version. To list the history of each change to the current environment: `conda list --revisions`

To restore environment to a previous revision: `conda install --revision=REVNUM` or `conda install --rev REVNUM`.

> **Note**
>
> Replace REVNUM with the revision number.

Example: If you want to restore your environment to revision 8, run `conda install --rev 8`.

Removing an environment

To remove an environment, in your terminal window, run:

```
conda remove --name myenv --all
```

You may instead use `conda env remove --name myenv`.

To verify that the environment was removed, in your terminal window, run:

```
conda info --envs
```

The environments list that displays should not show the removed environment.

Create explicit lockfiles without creating an environment

`@EXPLICIT` lockfiles allow you to (re)create environments without invoking the solver. They consist of an `@EXPLICIT` header plus a list of conda package URLs, optionally followed by their MD5 or SHA256 hash.

They can be obtained from existing environments via `conda list --explicit`, as seen in *Building identical conda environments*.

But what if you only need the lockfile? Would you need create to a temporary environment first just to delete it later? Fortunately, there's a way: you can invoke `conda` in JSON mode and then process the output with `jq`.

> **Tip**
>
> You'll need `jq` in your system. If you don't have it yet, you can install it via `conda` (e.g. `conda create -n jq jq`) or via your system package manager.

The command looks like this for Linux and macOS (replace `MATCHSPECS_GO_HERE` with the relevant packages you want):

```
echo "@EXPLICIT" > explicit.txt
CONDA_PKGS_DIRS=$(mktemp -d) conda create --dry-run MATCHSPECS_GO_HERE --json | jq -r '.
↪actions.FETCH[] | .url + "#" + .md5' >> explicit.txt
```

The syntax in Windows only needs some small changes:

```
echo "@EXPLICIT" > explicit.txt
set "CONDA_PKGS_DIRS=%TMP%\conda-%RANDOM%"
conda create --dry-run MATCHSPECS_GO_HERE --json | jq -r '.actions.FETCH[] | .url + "#"␣
↪+ .md5' >> explicit.txt
set "CONDA_PKGS_DIRS="
```

The resulting `explicit.txt` can be used to create a new environment with:

```
conda create -n new-environment --file explicit.txt
```

Managing channels

Conda channels are the locations where packages are stored. They serve as the base for hosting and managing packages. Conda packages are downloaded from remote channels, which are URLs to directories containing conda packages. The conda command searches a default set of channels and packages are automatically downloaded and updated from the default channel. Read more about *conda channels* and the various terms of service for their use.

For private or protected channels that require credentials, see *Using authenticated channels*.

Different channels can have the same package, so conda must handle these channel collisions.

There will be no channel collisions if you use only the defaults channel. There will also be no channel collisions if all of the channels you use only contain packages that do not exist in any of the other channels in your list. The way conda resolves these collisions matters only when you have multiple channels in your channel list that host the same package.

By default, conda prefers packages from a higher priority channel over any version from a lower priority channel. Therefore, you can put channels at the bottom of your channel list to provide additional packages that are not in the default channels. However, it is not recommended to mix channels in a single environment.

Conda collects all of the packages with the same name across all listed channels and processes them as follows:

1. Sorts packages from highest to lowest channel priority.
2. Sorts tied packages---packages with the same channel priority---from highest to lowest version number. For example, if channelA contains NumPy 1.12.0 and 1.13.1, NumPy 1.13.1 will be sorted higher.
3. Sorts still-tied packages---packages with the same channel priority and same version---from highest to lowest build number. For example, if channelA contains both NumPy 1.12.0 build 1 and build 2, build 2 is sorted first. Any packages in channelB would be sorted below those in channelA.
4. Installs the first package on the sorted list that satisfies the installation specifications.

Essentially, the order goes: channelA::numpy-1.13_1 > channelA::numpy-1.12.1_1 > channelA::numpy-1.12.1_0 > channelB::numpy-1.13_1

> **Note**
>
> If strict channel priority is turned on then channelB::numpy-1.13_1 isn't included in the list at all.

To make conda install the newest version of a package in any listed channel:

- Add `channel_priority: disabled` to your `.condarc` file.

  OR

- Run the equivalent command:

```
conda config --set channel_priority disabled
```

Conda then sorts as follows:

1. Sorts the package list from highest to lowest version number.
2. Sorts tied packages from highest to lowest channel priority.
3. Sorts tied packages from highest to lowest build number.

Because build numbers from different channels are not comparable, build number still comes after channel priority.

The following command adds the channel "new_channel" to the top of the channel list, making it the highest priority:

```
conda config --add channels new_channel
```

Conda has an equivalent command:

```
conda config --prepend channels new_channel
```

Conda also has a command that adds the new channel to the bottom of the channel list, making it the lowest priority:

```
conda config --append channels new_channel
```

Channel configuration best practices

To avoid channel conflicts and ensure environment reproducibility, we recommend the following approach:

1. **Use a single channel as your default.** Configure either `defaults` or `conda-forge` in your `.condarc`, but not both. Mixing these channels in your global configuration can result in mixed-channel environments with incompatible packages and can introduce hard-to-debug dependency conflicts.
2. **Install from a specific channel with a single command when needed.** If you need a package from a channel that isn't in your global config, you can target that channel with a single command using the options below. Note that mixing packages from different channels in this way might result in dependency issues if the channels are not fully compatible.

   - The `--channel` flag installs the named package and all its dependencies from the specified channel. Add `--override-channels` to ignore any channels configured in your `.condarc`, using only the specified channel.

     ```
     conda install --channel conda-forge --override-channels numpy
     ```

   - The `<channel>::<package>` syntax installs only the named package from the specified channel, with dependencies resolved from your configured channels.

     Warning

     If the channel you specify and your configured channels are not fully compatible, this approach can result in binary incompatibilities that can cause hard-to-debug runtime errors, also known as "ABI incompatibility".

```
conda install conda-forge::numpy
```

3. **Use strict channel priority only if you must mix channels.** If your workflow requires both `defaults` and `conda-forge` in your channel list, set your channel priority to `strict` to reduce the risk of incompatible mixed-channel environments:

```
conda config --set channel_priority strict
```

> **Warning**
>
> Strict channel priority can make some environments unsatisfiable. It also prevents fallback to lower-priority channels when a package with the same name exists in a higher-priority channel. This includes channels that supply packages not available as conda packages.

Using authenticated channels

Some conda channels require authentication before packages can be downloaded. For example, a private channel may require an access token, or a company-hosted channel may require HTTP basic authentication.

Conda supports authenticated channels through authentication handler plugins. For common authenticated-channel workflows, use the conda-auth plugin.

Install conda-auth

Install `conda-auth` into your base environment:

```
conda self install conda-auth
```

Because `conda-auth` is a conda plugin, it must be installed in the environment where conda itself is installed, usually `base`.

Log in with HTTP basic authentication

Use HTTP basic authentication for channels that require a username and password:

```
conda auth login https://example.com/my-protected-channel --basic
```

The command prompts for your credentials. After you log in, use the channel as you normally would:

```
conda install --channel https://example.com/my-protected-channel package-name
```

Log in with token authentication

Use token authentication for channels that require an access token:

```
conda auth login my-private-channel --token
```

When you provide a bare channel name such as `my-private-channel`, `conda-auth` uses the corresponding channel on `anaconda.org` by default.

The command prompts for your token. After you log in, use the channel as you normally would:

```
conda install --channel my-private-channel package-name
```

Log out of an authenticated channel

To remove stored credentials for a channel, log out:

```
conda auth logout https://example.com/my-protected-channel
```

Security notes

The `conda auth login` command prompts for secrets by default. This is recommended for interactive use because it avoids placing passwords or tokens directly in your shell history.

Passing passwords or tokens directly on the command line may be useful for non-interactive automation, but it can expose secrets in shell history, process listings, or CI logs. Follow your organization's secret management practices when using authenticated channels in automation.

Advanced authentication configuration

The `channel_settings` configuration option can associate a channel with a specific authentication handler. This is mainly useful for advanced workflows or custom authentication plugins.

For more information, see the conda-auth Getting Started guide, *channel_settings*, and the developer documentation for *Auth Handlers*.

Managing packages

> **Note**
>
> There are many options available for the commands described on this page. For details, see *commands*.

Searching for packages

Use the terminal for the following steps.

To see if a specific package, such as SciPy, is available for installation:

```
conda search scipy
```

To see if a specific package, such as SciPy, is available for installation from Anaconda.org:

```
conda search --override-channels --channel defaults scipy
```

To see if a specific package, such as iminuit, exists in a specific channel, such as http://conda.anaconda.org/mutirri, and is available for installation:

```
conda search --override-channels --channel http://conda.anaconda.org/mutirri iminuit
```

Installing packages

Use the terminal for the following steps.

To install a specific package such as SciPy into an existing environment "myenv":

```
conda install --name myenv scipy
```

If you do not specify the environment name, which in this example is done by `--name myenv`, the package installs into the current environment:

```
conda install scipy
```

To install a specific version of a package such as SciPy:

```
conda install scipy=0.15.0
```

To install multiple packages at once, such as SciPy and cURL:

```
conda install scipy curl
```

> **Note**
>
> It is best to install all packages at once, so that all of the dependencies are installed at the same time.

To install multiple packages at once and specify the version of the package:

```
conda install scipy=0.15.0 curl=7.26.0
```

To install a package for a specific Python version:

```
conda install scipy=0.15.0 curl=7.26.0 -n py34_env
```

If you want to use a specific Python version, it is best to use an environment with that version. For more information, see *Troubleshooting*.

Installing packages with a timestamp cutoff

Added in version 26.9.0.

Use `--exclude-newer` to ignore package records with timestamps after a configured cutoff for one `conda create`, `conda install`, or `conda update` command.

Conda prefers the channel-provided `indexed_timestamp`, which is intended to record when the package first became available in the channel index. If it is absent or zero, conda falls back to the builder-controlled `timestamp`, which records build time rather than publication time. Records without a usable timestamp remain eligible.

For artifacts predating CEP 47, channels may seed `indexed_timestamp` from build timestamps or other historical signals. These values may not reflect the exact time a package became available.

The fallback can still reduce exposure to newly built malicious packages when build timestamps are accurate, giving time for detection and removal. This is a best-effort security benefit, not a reliable full cooldown after publication. An old build uploaded today can pass the cutoff, and a malicious publisher can backdate the build timestamp. For example, with a 7-day cutoff, a package built 6 days before publication has only about 1 day of delay after publication. A cutoff can also delay legitimate security fixes.

Timestamp filtering does not guarantee package safety. It can support reproducible resolution, but does not by itself guarantee an exact reconstruction of a channel's past state.

To install a package while ignoring package records whose effective timestamps fall within the last 7 days:

```
conda install --exclude-newer 7d scipy
```

To update packages while applying the same cutoff:

```
conda update --exclude-newer 7d --all
```

To create an environment while excluding package records with effective timestamps after a specific date:

```
conda create --name myenv --exclude-newer 2026-04-01 python
```

Date-only cutoffs are interpreted as the start of the next day in UTC. To apply this policy persistently, or to set channel-specific or package-specific overrides, configure `exclude_newer` in your `.condarc` file.

Installing similar packages

Installing packages that have similar filenames and serve similar purposes may return unexpected results. The package last installed will likely determine the outcome, which may be undesirable. If the two packages have different names, or if you're building variants of packages and need to line up other software in the stack, we recommend using *Mutex metapackages*.

Installing packages from Anaconda.org

Packages that are not available using `conda install` can be obtained from Anaconda.org, a package management service for both public and private package repositories. Anaconda.org is an Anaconda product, just like Anaconda and Miniconda.

To install a package from Anaconda.org:

1. In a browser, go to http://anaconda.org.
2. To find the package named bottleneck, type `bottleneck` in the top-left box named Search Packages.
3. Find the package that you want and click it to go to the detail page.

   The detail page displays the name of the channel. In this example it is the "pandas" channel.
4. Now that you know the channel name, use the `conda install` command to install the package. In your terminal window, run:

   ```
   conda install -c pandas bottleneck
   ```

   This command tells conda to install the bottleneck package from the pandas channel on Anaconda.org.
5. To check that the package is installed, in your terminal window, run:

   ```
   conda list
   ```

   A list of packages appears, including bottleneck.

> **Note**
>
> For information on installing packages from multiple channels, see *Managing channels*.

Installing non-conda packages

If a package is not available from any of the popular conda channels, there are a few options depending on the package type.

For packages with supported pure Python wheels, the recommended path is to install them directly with `conda install` using the `conda-pypi` channel, which indexes pure Python wheels from the public PyPI index and resolves

them alongside conda packages in a single operation. See *Installing packages from PyPI with conda* for setup instructions and supported workflows.

For packages not covered by the `conda-pypi` workflow, you may be able to use pip as a fallback. The differences between packages installed with pip and conda packages cause certain unavoidable limits in compatibility but conda works hard to be as compatible with pip as possible.

Note

Both pip and conda are included in Anaconda and Miniconda, so you do not need to install them separately.

Conda environments replace virtualenv, so there is no need to activate a virtualenv before using pip.

It is possible to have pip installed outside a conda environment or inside a conda environment.

To gain the benefits of conda integration, be sure to install pip inside the currently active conda environment and then install packages with that instance of pip. The command `conda list` shows packages installed this way, with a label showing that they were installed with pip.

You can install pip in the current conda environment with the command `conda install pip`, as discussed in *Using pip in an environment*.

If there are instances of pip installed both inside and outside the current conda environment, the instance of pip installed inside the current conda environment is used.

To install a non-conda package:

1. Activate the environment where you want to put the program:
   - In your terminal window, run `conda activate myenv`.
2. To use pip to install a program such as See, in your terminal window, run:

   ```
   pip install see
   ```

3. To verify the package was installed, in your terminal window, run:

   ```
   conda list
   ```

   If the package is not shown, install pip as described in *Using pip in an environment* and try these commands again.

Installing commercial packages

Installing a commercial package such as IOPro is the same as installing any other package. In your terminal window, run:

```
conda install --name myenv iopro
```

This command installs a free trial of one of Anaconda's commercial packages called IOPro, which can speed up your Python processing. Except for academic use, this free trial expires after 30 days.

Viewing a list of installed packages

Use the terminal for the following steps.

To list all of the packages in the active environment:

```
conda list
```

To list all of the packages in a deactivated environment:

```
conda list -n myenv
```

Listing package dependencies

To find what packages are depending on a specific package in your environment, there is not one specific conda command. It requires a series of steps:

1. List the dependencies that a specific package requires to run: `conda search package_name --info`
2. Find your installation's package cache directory: `conda info`
3. Find package dependencies. By default, Anaconda/Miniconda stores packages in ~/anaconda/pkgs/ (or ~/opt/pkgs/ on macOS Catalina). Each package has an index.json file which lists the package's dependencies. This file resides in ~anaconda/pkgs/package_name/info/index.json.
4. Now you can find what packages depend on a specific package. Use grep to search all index.json files as follows: `grep package_name ~/anaconda/pkgs/*/info/index.json`

The result will be the full package path and version of anything containing the <package_name>.

Example: `grep numpy ~/anaconda3/pkgs/*/info/index.json`

Output from the above command:

```
/Users/testuser/anaconda3/pkgs/anaconda-4.3.0-np111py36_0/info/index.json: numpy 1.11.3␣
↪py36_0
/Users/testuser/anaconda3/pkgs/anaconda-4.3.0-np111py36_0/info/index.json: numpydoc 0.6.
↪0 py36_0
/Users/testuser/anaconda3/pkgs/anaconda-4.3.0-np111py36_0/info/index.json: numpy 1.11.3␣
↪py36_0
```

Note this also returned "numpydoc" as it contains the string "numpy". To get a more specific result set you can add < and >.

Updating packages

Use `conda update` command to check to see if a new update is available. If conda tells you an update is available, you can then choose whether or not to install it.

Use the terminal for the following steps.

- To update a specific package:

  ```
  conda update biopython
  ```

- To update Python:

  ```
  conda update python
  ```

- To update conda itself:

  ```
  conda update conda
  ```

> **Note**
>
> Conda updates to the highest version in its series, so when updating Python, it updates to the highest available in the 3.x series.

To update the Anaconda metapackage:

```
conda update conda
conda update anaconda
```

Regardless of what package you are updating, conda compares versions and then reports what is available to install. If no updates are available, conda reports "All requested packages are already installed."

If a newer version of your package is available and you wish to update it, type `y` to update:

```
Proceed ([y]/n)? y
```

Preventing packages from updating (pinning)

Pinning a package specification in an environment prevents packages listed in the `pinned` file from being updated.

In the environment's `conda-meta` directory, add a file named `pinned` that includes a list of the packages that you do not want updated.

EXAMPLE: The file below forces NumPy to stay on the 1.7 series, which is any version that starts with 1.7. This also forces SciPy to stay at exactly version 0.14.2:

```
numpy 1.7.*
scipy ==0.14.2
```

With this `pinned` file, `conda update numpy` keeps NumPy at 1.7.1, and `conda install scipy=0.15.0` causes an error.

Use the `--no-pin` flag to override the update restriction on a package. In the terminal, run:

```
conda update numpy --no-pin
```

Because the `pinned` specs are included with each conda install, subsequent `conda update` commands without `--no-pin` will revert NumPy back to the 1.7 series.

Adding default packages to new environments automatically

To automatically add default packages to each new environment that you create:

1. Open a terminal window and run: `conda config --add create_default_packages PACKAGENAME1 PACKAGENAME2`
2. Now, you can create new environments and the default packages will be installed in all of them.

You can also *edit the .condarc file* with a list of packages to create by default.

You can override this option at the command prompt with the `--no-default-packages` flag.

Removing packages

Use the terminal for the following steps.

- To remove a package such as SciPy in an environment such as myenv:

```
conda remove -n myenv scipy
```

- To remove a package such as SciPy in the current environment:

```
conda remove scipy
```

- To remove multiple packages at once, such as SciPy and cURL:

```
conda remove scipy curl
```

- To confirm that a package has been removed:

```
conda list
```

Installing packages from PyPI with conda

Note

The `conda-pypi` channel is free to use for all users. This channel is not subject to the licensing requirements or payment obligations described in Section 1 of the Anaconda Terms of Service.

In conda version 26.9 and later, you can use the `conda-pypi` channel to install supported pure Python wheels from the Python Package Index (PyPI) alongside conda packages directly with `conda install`.

What does the `conda-pypi` channel do?

The `conda-pypi` channel, a channel maintained by Anaconda and hosted for free on anaconda.org, indexes all pure Python wheels from the public PyPI index. The channel repodata is generated from package metadata obtained from PyPI and converted into a format conda understands, so the reliability and security of packages available through `conda-pypi` matches that of PyPI directly. You can browse the conda-pypi channel on Anaconda.org to explore available packages and versions.

When you add this channel to your conda configuration, conda can resolve and install those packages alongside packages from conda channels like `conda-forge` or `defaults` in a single operation. The channel repodata tells conda what packages are available and at what versions. When you install a package, conda fetches the wheel directly from PyPI at install time. This is by design and keeps the channel lightweight, but it means you need a PyPI connection at install time.

One important thing to note is that **this is not a pip replacement**. Conda still manages the environment, the solver still handles the full dependency graph, and the result is a reproducible, exportable conda environment. Using `pip install` directly inside conda environments can work initially, but conda loses track of pip-installed packages, making future environment updates and installs unreliable. For more information on using pip with conda, see our blog Conda and pip are two ecosystems, not just tools.

Prerequisites

The `conda-pypi` workflow requires the following:

conda 26.9 or later

Check your version:

```
conda --version
```

If necessary, update conda:

```
conda self update --all
```

Rattler solver

This workflow requires `conda-rattler-solver`. Check your configured solver with `conda config --show solver`. If it isn't `rattler`, select it explicitly:

```
conda config --set solver rattler
```

For more information, see *Solvers*.

Setting up the conda-pypi channel

To make supported packages from PyPI available to conda, add the `conda-pypi` channel to your conda configuration (`.condarc`) file. Append it after your existing channels to give it a lower channel priority:

```
conda config --append channels conda-pypi
```

Your channel order in `~/.condarc` should look similar to this:

```
channels:
  - conda-forge
  - conda-pypi
```

Channel priority and ABI compatibility

Normally, when mixing channels, strict channel priority is recommended to avoid ABI incompatibilities between compiled packages built in different environments. The `conda-pypi` channel contains only pure Python wheels, so there are no compiled binaries and, therefore, no ABI risk.

Flexible priority (the default) also works correctly when pairing `conda-pypi` with `conda-forge` or `defaults`. For more on channel priority, see *Channel configuration best practices*.

Install packages

Before installing, you can check what is available in the `conda-pypi` channel using `conda search`. Once the channel is configured, `conda install` resolves the full dependency graph across all configured channels, so packages from PyPI and conda channels can be installed together in a single command.

Find available packages

You can browse the conda-pypi channel on Anaconda.org to explore available packages, or search directly from the command line: :

```
conda search conda-pypi::<package-name>
```

Install packages from PyPI and conda channels together

```
conda install <conda-forge-package> <conda-pypi-package>
```

When a package is available from both `conda-forge` and `conda-pypi`, conda selects a build according to your channel priority setting and the environment's dependency requirements.

You can also use `conda create` with packages from PyPI:

```
conda create --name myenv python=3.12 <conda-forge-package> <conda-pypi-package>
```

Disable the conda-pypi workflow

If you want to stop using the `conda-pypi` channel, remove it from your configuration:

```
conda config --remove channels conda-pypi
```

Suppress the conda-pypi suggestion tip

To suppress the suggestion about the `conda-pypi` channel shown when conda newly installs pip into an environment, set the following configuration option:

```
conda config --set plugins.conda_pypi_pip_warning false
```

Limitations and security considerations

The security posture of this workflow is equivalent to using pip or other PyPA tools to install from the public PyPI index directly. Packages are fetched from PyPI at install time. Conda does not re-sign or additionally vet the wheels. Keep the following limitations in mind:

- **Public PyPI only.** The `conda-pypi` channel indexes packages from PyPI. It does not include packages from private or alternative package indexes (such as a corporate artifact repository).
- **No additional vetting beyond PyPI.** The `conda-pypi` channel does not perform additional security scanning beyond what PyPI provides. The *--exclude-newer flag* can reduce exposure to recent malicious packages when usable, accurate timestamps are available, but is not a substitute for security scanning. A reliable publication cooldown requires channel-provided `indexed_timestamp` values. The fallback `timestamp` is builder-controlled, and records without a usable timestamp remain eligible.
- **conda client only.** Supported in the conda CLI. Support in other clients (mamba, micromamba, and so on) is under active CEP discussion.
- **Pure Python wheels only.** Packages with compiled extensions must come from conda channels.
- **rattler solver required.** The classic conda solver is not supported for this workflow.
- **PyPI connection required at install time.** The channel stores metadata only. Wheels are fetched from PyPI during installation.

Troubleshooting

Packages not found in `conda-pypi`

If `conda install` or `conda create` fails with a `PackagesNotFoundInChannelsError` for a package you expect to be available through `conda-pypi`, the most likely cause is an outdated version of conda or the rattler solver.

For example:

```
PackagesNotFoundInChannelsError: The following packages are not available
from current channels:
  - flask

Current channels:
  - defaults
  - https://conda.anaconda.org/conda-pypi
```

To diagnose, check your conda and rattler-solver versions:

```
conda --version
```

```
conda list | grep rattler
```

If either is below the minimum required version (26.9 for `conda` and 0.1.1 for `conda-rattler-solver`), update them:

```
conda self update
```

```
conda install --name base conda-rattler-solver
```

Additional resources

- conda-pypi plugin documentation — advanced configuration and workflows
- *Managing channels* — channel priority and configuration reference

Managing Python

Conda treats Python the same as any other package, so it is easy to manage and update multiple installations.

Conda itself requires Python 3.11+ to run. This is the version of Python conda is installed with, not a limit on the environments conda can create and manage — those can use any available Python version, including versions older than conda's own minimum, as shown below.

Viewing a list of available Python versions

To list the versions of Python that are available to install, in your terminal window, run:

```
conda search python
```

This lists all packages whose names contain the text `python`.

To list only the packages whose full name is exactly `python`, add the `--full-name` option. In your terminal window, run:

```
conda search --full-name python
```

Installing a different version of Python

To install a different version of Python without overwriting the current version, create a new environment and install the second Python version into it:

1. Create the new environment:
    - To create the new environment for Python 3.9, in your terminal window run:

```
conda create -n py39 python=3.9
```

> **Note**
>
> Replace `py39` with the name of the environment you want to create. `python=3.9` is the package and version you want to install in this new environment. This could be any package, such as `numpy=1.19`, or *multiple packages*.

2. *Activate the new environment*.
3. Verify that the new environment is your *current environment*.
4. To verify that the current environment uses the new Python version, in your terminal window, run:

```
python --version
```

Installing PyPy

To use the PyPy builds you can do the following:

```
conda config --add channels conda-forge
conda config --set channel_priority strict
conda create -n pypy pypy
conda activate pypy
```

> **Warning**
>
> Using strict channel priority can make some environments unsatisfiable because it prevents fallback to lower-priority channels when a package with the same name exists in a higher-priority channel. To learn more about strict channel priority, see *channel configuration best practices*.

Using a different version of Python

To switch to an environment that has different version of Python, *activate the environment*.

Updating Python

To update Python to the latest version in your environment, run:

```
conda update python
```

This command will update you to the latest major release (e.g. from `python=3.10` to `python=3.12`).

If you would like to remain on a minor release, use the `conda install` command instead:

```
conda install python=3.10
```

Managing virtual packages

"Virtual" packages are injected into the conda solver to allow real packages to depend on features present on the system that cannot be managed directly by conda, like system driver versions or CPU features. Virtual packages are not real packages and not displayed by `conda list`. Instead `conda` runs a small bit of code to detect the presence or absence of the system feature that corresponds to the package. The currently supported list of virtual packages includes:

- `__cuda`: Maximum version of CUDA supported by the display driver.
- `__osx`: OSX version if applicable.
- `__glibc`: Version of glibc supported by the OS.
- `__linux`: Available when running on Linux.
- `__unix`: Available when running on OSX or Linux.
- `__win`: Available when running on Win.
- `__conda`: Version of conda that is being used for solving.

Other virtual packages will be added in future conda releases. These are denoted by a leading double-underscore in the package name.

> **Note**
>
> Note that as of version `22.11.0`, *virtual packages* are implemented as *conda plugins*.

Listing detected virtual packages

Use the terminal for the following steps.

To see the list of detected virtual packages, run:

```
conda info
```

If a package is detected, you will see it listed in the `virtual packages` section, as shown in this example:

```
     active environment : base
    active env location : /Users/demo/dev/conda/devenv
            shell level : 1
       user config file : /Users/demo/.condarc
populated config files : /Users/demo/.condarc
         conda version : 4.6.3.post8+8f640d35a
   conda-build version : 3.17.8
        python version : 3.7.2.final.0
      virtual packages : __cuda=10.0
      base environment : /Users/demo/dev/conda/devenv (writable)
          channel URLs : https://repo.anaconda.com/pkgs/main/osx-64
                         https://repo.anaconda.com/pkgs/main/noarch
                         https://repo.anaconda.com/pkgs/free/osx-64
                         https://repo.anaconda.com/pkgs/free/noarch
                         https://repo.anaconda.com/pkgs/r/osx-64
                         https://repo.anaconda.com/pkgs/r/noarch
         package cache : /Users/demo/dev/conda/devenv/pkgs
                         /Users/demo/.conda/pkgs
      envs directories : /Users/demo/dev/conda/devenv/envs
```

(continues on next page)

(continued from previous page)

```
                        /Users/demo/.conda/envs
             platform : osx-64
            user-agent : conda/4.6.3.post8+8f640d35a requests/2.21.0 CPython/3.7.2␣
↪Darwin/17.7.0 OSX/10.13.6
                UID:GID : 502:20
            netrc file : None
          offline mode : False
```

Overriding detected packages

For troubleshooting, it is possible to override virtual package detection using an environment variable or a setting in your `.condarc` file.

Environment variables

You can set environment variables when using a `conda install` command. This way of overriding has the highest priority, **but the override variable must be set every time a package requiring that override is installed**.

Example:

```
CONDA_OVERRIDE_CUDA=12.8 conda install pytorch
```

Supported variables include:

Variable Name	Override Entity	Example
CONDA_OVERRIDE_ARCHSPEC	Build Number	x86_64
CONDA_OVERRIDE_CUDA	Version Number	12.8
CONDA_OVERRIDE_GLIBC	Version Number	2.17
CONDA_OVERRIDE_LINUX	Version Number	5.15.0
CONDA_OVERRIDE_OSX	Version Number	11.0
CONDA_OVERRIDE_WIN	Version Number	10.0.22631

Note

For any custom virtual packages, the variable name is `CONDA_OVERRIDE_<NAME>` and the override entity is the `override_entity` variable value, either "version" or "build".

`.condarc` setting

To avoid having to set any override variables every time you install a package, conda also offers the ability to set these overrides in your `.condarc` file.

```
override_virtual_packages:
  archspec: "x86_64"
  cuda: "12.8"
  glibc: "2.17"
  osx: "11.0"
  mycustompackage: "1.2.3"
```

See *override-virtual-packages* for more information on the `.condarc` setting.

Creating custom channels

In this tutorial, we walk through how to create your own channel that can either be accessed via the local or network file system or served from a webserver.

To create a custom channel:

1. You will need to install conda-build to complete this tutorial. If you do not already have it, you can install it with the following command:

```
conda install conda-build
```

2. Organize all the packages in subdirectories for the platforms you wish to serve. Below is an example of what this may look like:

```
channel
├── linux-64
│   └── package-1.0-0.tar.bz2
├── osx-64
│   └── package-1.0-0.tar.bz2
└── win-64
    └── package-1.0-0.tar.bz2
```

3. Run `conda index` on the channel root directory:

```
conda index channel/
```

The conda index command generates a file `repodata.json`, saved to each repository directory, which conda uses to get the metadata for the packages in the channel.

> **Note**
>
> Each time you add or modify a package in the channel, you must rerun `conda index` for conda to see the update.

4. To test custom channels, serve the custom channel using a web server or using a `file://` URL to the channel directory. Test by sending a search command to the custom channel.

Example: if you want a file in the custom channel location `/opt/channel/linux-64/`, search for files in that location:

```
conda search -c file:///opt/channel/ --override-channels
```

> **Note**
>
> - The channel URL does not include the platform, as conda automatically detects and adds the platform.
> - The option `--override-channels` ensures that conda searches only your specified channel and no other channels, such as default channels or any other channels you may have listed in your `.condarc` file.

If you have set up your private repository correctly, you get the following output:

```
Fetching package metadata: . . . .
```

This is followed by a list of the conda packages found. This verifies that you have set up and indexed your private repository successfully.

Creating projects

In this tutorial, we will walk through how to set up a new Python project in conda using an `environment.yml` file. This file will help you keep track of your dependencies and share your project with others. We cover how to create your project, add a simple Python program and update it with new dependencies.

Requirements

To follow along, you will need a working conda installation. Please head over to our *installation guide* for instructions on how to get conda installed if you do not have it.

This tutorial relies heavily on using your computer's terminal (Command Prompt or PowerShell on Windows), so it is also important to have a working familiarity with using basic commands such as `cd` and `ls`.

Creating the project's files

To start off, we will need a directory that will contain the files for our project. This can be created with the following command:

```
mkdir my-project
```

In this directory, we will now create a new `environment.yml` file, which will hold the dependencies for our Python project. In your text editor (e.g. VSCode, PyCharm, vim, etc.), create this file and add the following:

```
name: my-project
channels:
  - defaults
dependencies:
  - python
```

Let's briefly go over what each part of this file means.

Name
: The name of your environment. Here, we have chosen the name "my-project", but this can be anything you want.

Channels
: Channels specify where you want conda to search for packages. We have chosen the `defaults` channel, but others such as `conda-forge` or `bioconda` are also possible to list here.

Dependencies
: All the dependencies that you need for your project. So far, we have just added `python` because we know it will be a Python project. We will add more later.

Creating our environment

Now that we have written a basic `environment.yml` file, we can create and activate an environment from it. To do so, run the following commands:

```
conda create --file environment.yml
# or: conda env create --file environment.yml
conda activate my-project
```

Creating our Python application

With our new environment with Python installed, we can create a simple Python program. In your project folder, create a `main.py` file and add the following:

```
def main():
    print("Hello, conda!")


if __name__ == "__main__":
    main()
```

We can run our simple Python program by running the following command:

```
python main.py
Hello, conda!
```

Updating our project with new dependencies

If you want your project to do more than the simple example above, you can use one of the thousands of available packages on conda channels. To demonstrate this, we will add a new dependency so that we can pull in some data from the internet and perform a basic analysis.

To perform the data analysis, we will be relying on the Pandas package. To add this to our project, we will need to update our `environment.yml` file:

```
name: my-project
channels:
  - defaults
dependencies:
  - python
  - pandas  # <-- This is our new dependency
```

Once we have done that, we can run the `conda env update` command to install the new package:

```
conda env update --file environment.yml
```

Now that our dependencies are installed, we will download some data to use for our analysis. For this, we will use the U.S. Environmental Protection Agency's Walkability Index dataset available on data.gov. You can download this with the following command:

```
curl -O https://edg.epa.gov/EPADataCommons/public/OA/EPA_SmartLocationDatabase_V3_Jan_
↪2021_Final.csv
```

Tip

If you do not have `curl`, you can visit the above link with a web browser to download it.

For our analysis, we are interested in knowing what percentage of U.S. residents live in highly walkable areas. This is a question that we can easily answer using the `pandas` library. Below is an example of how you might go about doing that:

```
import pandas as pd


def main():
    """
    Answers the question:

    What percentage of U.S. residents live highly walkable neighborhoods?

    "15.26" is the threshold on the index for a highly walkable area.
    """
    csv_file = "./EPA_SmartLocationDatabase_V3_Jan_2021_Final.csv"
    highly_walkable = 15.26

    df = pd.read_csv(csv_file)

    total_population = df["TotPop"].sum()
    highly_walkable_pop = df[df["NatWalkInd"] >= highly_walkable]["TotPop"].sum()

    percentage = (highly_walkable_pop / total_population) * 100.0

    print(
        f"{percentage:.2f}% of U.S. residents live in highly" "walkable neighborhoods."
    )


if __name__ == "__main__":
    main()
```

Update your `main.py` file with the code above and run it. You should get the following answer:

```
python main.py
10.69% of Americans live in highly walkable neighborhoods
```

Conclusion

You have just been introduced to creating your own data analysis project by using the `environment.yml` file in conda. As the project grows, you may wish to add more dependencies and also better organize the Python code into separate files and modules.

For even more information about working with environments and `environment.yml` files, please see *Managing Environments*.

Viewing command-line help

To see a list of supported conda commands, in your terminal window, run:

```
conda --help
```

or

```
conda -h
```

To get help for a specific command, type the command name followed by `--help`.

Example

To see help for the `create` command, in your terminal window, run:

```
conda create -h
```

Note

You can see the same command help in *commands*.

The tasks section is organized into various pages which cover nearly everything you can do with conda.

Common Tasks

Managing conda
Everything necessary to know about managing your installation of conda

Managing environments
Various operations involved with creating, updating, exporting, and removing environments, plus more

Managing channels
Information about channels and how they are searched through when installing packages

Using authenticated channels
Learn how to log in to private or protected channels that require HTTP basic authentication or token authentication

Managing packages
Details related to how to find, install, remove, and update packages in a given environment

Installing packages from PyPI
How to install packages from PyPI with conda, including extras support and environment management

Managing python
Supported versions of Python and tips for updating and using multiple Python versions

Managing virtual packages
Learn what virtual packages are and conda uses them

View command line help
Get help on the command line for any conda command

Tutorials

Creating custom channels
Tutorial walking you through how to create a custom channel and serve it from your local computer

Creating projects with conda
Learn how to start a new project with conda using a `environment.yml` file to manage your dependencies

4.1.4 Configuration

Using the .condarc conda configuration file

Overview

The conda configuration file, `.condarc`, is an optional runtime configuration file that allows advanced users to configure various aspects of conda, such as which channels it searches for packages, proxy settings, and environment directories. For all of the conda configuration options, see the *configuration page*.

> **Note**
>
> A `.condarc` file can also be used in an administrator-controlled installation to override the users' configuration. See *Administering a multi-user conda installation*.

The `.condarc` file can change many parameters, including:

- Where conda looks for packages.
- If and how conda uses a proxy server.
- Where conda lists known environments.
- Whether to update the Bash prompt with the currently activated environment name.
- Whether user-built packages should be uploaded to Anaconda.org.
- What default packages or features to include in new environments.

Creating and editing

The `.condarc` file is not included by default, but it is automatically created in your home directory the first time you run the `conda config` command. To create or modify a `.condarc` file, open a terminal and enter the `conda config` command.

The `.condarc` configuration file follows simple YAML syntax.

Example:

```
conda config --add channels conda-forge
```

Alternatively, you can open a text editor such as Notepad on Windows, TextEdit on macOS, or VS Code. Name the new file `.condarc` and save it to your user home directory or root directory. To edit the `.condarc` file, open it from your home or root directory and make edits in the same way you would with any other text file. If the `.condarc` file is in the root environment, it will override any in the home directory.

You can find information about your `.condarc` file by typing `conda info` in your terminal. This will give you information about your `.condarc` file, including where it is located.

You can also download a *sample .condarc file* to edit in your editor and save to your user home directory or root directory.

To set configuration options, edit the `.condarc` file directly or use the `conda config --set` command.

Example:

To set the `auto_update_conda option` to `False`, run:

```
conda config --set auto_update_conda False
```

For a complete list of conda config commands, see the *command reference*. The same list is available at the terminal by running `conda config --help`. You can also see the conda channel configuration for more information.

Conda supports a wide range of configuration options. This page gives a non-exhaustive list of the most frequently used options and their usage. For a complete list of all available options for your version of conda, use the `conda config --describe` command.

Searching for .condarc

Conda looks in the following locations for a `.condarc` file:

```
if on_win:
    SEARCH_PATH = (
        "C:/ProgramData/conda/.condarc",
        "C:/ProgramData/conda/condarc",
        "C:/ProgramData/conda/condarc.d",
    )
else:
    SEARCH_PATH = (
        "/etc/conda/.condarc",
        "/etc/conda/condarc",
        "/etc/conda/condarc.d/",
        "/var/lib/conda/.condarc",
        "/var/lib/conda/condarc",
        "/var/lib/conda/condarc.d/",
    )

SEARCH_PATH += (
    "$CONDA_ROOT/.condarc",
    "$CONDA_ROOT/condarc",
    "$CONDA_ROOT/condarc.d/",
    "$XDG_CONFIG_HOME/conda/.condarc",
    "$XDG_CONFIG_HOME/conda/condarc",
    "$XDG_CONFIG_HOME/conda/condarc.d/",
    "~/.config/conda/.condarc",
    "~/.config/conda/condarc",
    "~/.config/conda/condarc.d/",
    "~/.conda/.condarc",
    "~/.conda/condarc",
    "~/.conda/condarc.d/",
    "~/.condarc",
    "$CONDA_PREFIX/.condarc",
    "$CONDA_PREFIX/condarc",
    "$CONDA_PREFIX/condarc.d/",
    "$CONDARC",
)
```

`XDG_CONFIG_HOME` is the path to where user-specific configuration files should be stored defined following The XDG Base Directory Specification (XDGBDS). Default to $HOME/.config should be used. `CONDA_ROOT` is the path for your base conda install. `CONDA_PREFIX` is the path to the current active environment. `CONDARC` must be a path to a file named `.condarc`, `condarc`, or end with a YAML suffix (`.yml` or `.yaml`).

> **Note**
>
> Any condarc files that exist in any of these special search path directories need to end in a valid yaml extension (".yml" or ".yaml").

Conflict merging strategy

When conflicts between configurations arise, the following strategies are employed:

- Lists - merge
- Dictionaries - merge
- Primitive - clobber

Precedence

The precedence by which the conda configuration is built out is shown below. Each new arrow takes precedence over the ones before it. For example, config files (by parse order) will be superseded by any of the other configuration options. Configuration environment variables (formatted like `CONDA_<CONFIG NAME>`) will always take precedence over all config files. Command line parameters have the highest precedence.



Obtaining information from the .condarc file

You can use the following commands to get the effective settings for conda. The effective settings are those that have merged settings from all the sources mentioned above.

To get all keys and their values:

```
conda config --get
```

To get the value of a specific key, such as channels:

```
conda config --get channels
```

To show all the configuration file sources and their contents:

```
conda config --show-sources
```

Saving settings to your .condarc file

The `.condarc` file can also be modified via conda commands. Below are several examples of how to do this.

To add a new value, such as http://conda.anaconda.org/mutirri, to a specific key, such as channels:

```
conda config --add channels http://conda.anaconda.org/mutirri
```

To remove an existing value, such as http://conda.anaconda.org/mutirri from a specific key, such as channels:

```
conda config --remove channels http://conda.anaconda.org/mutirri
```

To remove a key, such as channels, and all of its values:

```
conda config --remove-key channels
```

To configure channels and their priority for a single environment, make a `.condarc` file in the *root directory of that environment*.

Sample .condarc file

Because the `.condarc` file is just a YAML file, it means that it can be edited directly. Below is an example `.condarc` file:

```yaml
# This is a sample .condarc file.
# It adds the r Anaconda.org channel and enables
# the show_channel_urls option.

# channel locations. These override conda defaults, i.e., conda will
# search *only* the channels listed here, in the order given.
# Use "defaults" to automatically include all default channels.
# Non-url channels will be interpreted as Anaconda.org usernames
# (this can be changed by modifying the channel_alias key; see below).
# The default is just 'defaults'.
channels:
  - r
  - defaults

# Show channel URLs when displaying what is going to be downloaded
# and in 'conda list'. The default is False.
show_channel_urls: True

# Development environment settings (used by dev/start scripts)
# Custom path for conda development environments
devenv: ~/my-conda-devenv

# Preferred installer type for conda development environments
# Options: miniconda, miniforge
installer_type: miniforge

# For more information about this file see:
# https://conda.io/docs/user-guide/configuration/use-condarc.html
```

Settings

This page contains an overview of many important settings available in conda with examples where possible.

General configuration

`channels`: Channel locations

Listing channel locations in the `.condarc` file overrides conda defaults, causing conda to search only the channels listed there in the order given.

Use `defaults` to automatically include all default channels. Non-URL channels are interpreted as Anaconda.org user or organization names. You can change this by modifying the `channel_alias` as described in *channel_alias: Set a channel alias*. The default is just `defaults`.

Example:

```
channels:
  - <anaconda_dot_org_username>
  - http://some.custom/channel
  - file:///some/local/directory
  - defaults
```

To select channels for a single environment, put a `.condarc` file in the root directory of that environment (or use the `--env` option when using `conda config`).

Example: If you have installed Miniconda with Python 3 in your home directory and the environment is named "flowers", the path may be:

```
~/miniconda3/envs/flowers/.condarc
```

`default_channels`: Default channels

Normally, the defaults channel points to several channels at the repo.anaconda.com repository, but if `default_channels` is defined, it sets the new list of default channels. This is especially useful for airgapped and enterprise installations.

To ensure that all users only pull packages from an on-premises repository, an administrator can set both *channel alias* and `default_channels`.

```
default_channels:
  - http://some.custom/channel
  - file:///some/local/directory
```

See also

custom_multichannels can also be used to override the `defaults` multichannel.

`custom_multichannels`: Define custom multichannels

A multichannel is a metachannel composed of multiple channels. The only reserved multichannel is `local`, which is a list of `file://` channel locations where conda-build stashes successfully-built packages and cannot be overridden.

Other multichannels, including `defaults`, can be defined or customized with `custom_multichannels`, where the key is the multichannel name and the value is a list of channel names and/or channel URLs.

Example:

```
custom_multichannels:
  my-multichannel:
    - http://some.custom/channel
    - conda-forge
```

Changed in version 26.1.0: `custom_multichannels` can now be used to define the `defaults` multichannel, providing an alternative to the *default_channels* setting. The `default_channels` setting exists for historical reasons when `defaults` was a reserved multichannel name.

For example, to point the `defaults` multichannel to a custom mirror:

```
custom_multichannels:
  defaults:
    - https://my-mirror.com/pkgs/main
    - https://my-mirror.com/pkgs/r
```

This is equivalent to setting `default_channels` to the same list of channels. If both `default_channels` and `custom_multichannels.defaults` are defined, `custom_multichannels.defaults` takes precedence.

`channel_settings`: Extra settings for individual channels

Added in version 23.3.0.

Added in version 26.9.0: Support for channel-specific `exclude_newer` settings.

With `channel_settings`, it is possible to add extra configuration options for individual channels. This is used to register additional authentication handlers for conda via the *Auth Handlers* plugin hook and to set channel-specific `exclude_newer` policies.

Here is an example of how it may be defined provided there was an available authentication handler called, "test-auth-handler" registered via the aforementioned plugin hook:

```
channel_settings:
   - channel: https://some.custom/channel
     auth: test-auth-handler
     user: my-user-account
   - channel: https://some.base-url-prefix/*
     auth: another-auth-handler
   - channel: conda-forge
     exclude_newer: 7d
   - channel: defaults
     exclude_newer: 14d
```

Note

Each entry in `channel_settings` needs to define the `channel` attribute so that the configuration knows which channel these settings are associated with. The `channel` attribute may be a channel name, a multichannel name such as `defaults`, a channel URL, or a glob-like URL pattern. Multichannel settings apply to their resolved member channels. For glob-like URL patterns, the HTTP schema must match exactly to the channel URL, so a pattern like `*` is not valid.

`allowlist_channels` and `denylist_channels`: Allow or deny specific channels

Added in version 24.9.0: The `denylist_channels` setting was introduced in conda 24.9.0 complementing the existing `allowlist_channels` setting.

With `allowlist_channels` and `denylist_channels`, you can allow or deny specific channels from being used in conda operations. This is useful for restricting the channels that conda can access, especially in enterprise or multi-user environments.

The denylist takes precedence over the allowlist. If a channel is in both lists, it is denied.

Examples:

An example which allows the `defaults` and `conda-forge` channels with the `allowlist_channels` setting is:

```
allowlist_channels:
  - defaults
  - conda-forge
```

An example which denies the `conda-forge` channel with the `denylist_channels` setting is:

```
denylist_channels:
  - conda-forge
```

An example which explicitly allows the `defaults` channel but denies the `conda-forge` channel by using both the `allowlist_channels` and `denylist_channels` settings is:

```
allowlist_channels:
  - defaults
denylist_channels:
  - conda-forge
```

An example to show that channels are automatically normalized based on their base URLs, so you can use either the full channel URL or just the base URL:

```
allowlist_channels:
  - defaults
denylist_channels:
  - https://conda.anaconda.org/conda-forge/linux-64
```

An example that denies using `defaults` (which maps to the *default_channels*) configuration option:

```
denylist_channels:
  - defaults
```

Note

The *defaults channel* points to a list of channels at the repo.anaconda.com repository by default.

An example to explicitly deny the channels that are hosted on `repo.anaconda.com`:

```
denylist_channels:
  - https://repo.anaconda.com/pkgs/main
  - https://repo.anaconda.com/pkgs/r
  - https://repo.anaconda.com/pkgs/msys2
```

`auto_update_conda`: Update conda automatically

When `True`, conda updates itself any time a user updates or installs a package in the root environment. When `False`, conda updates itself only if the user manually issues a `conda update` command. The default is `True`.

Example:

```
auto_update_conda: False
```

always_yes: Always yes

Choose the `yes` option whenever asked to proceed, such as when installing. Same as using the `--yes` flag at the command line. The default is `False`.

Example:

```
always_yes: True
```

show_channel_urls: Show channel URLs

Show channel URLs in `conda list` and when displaying what is going to be downloaded. The default is `False`.

Example:

```
show_channel_urls: True
```

changeps1: Change command prompt

When using `conda activate`, change the command prompt from `$PS1` to include the activated environment. The default is `True`.

Example:

```
changeps1: False
```

> **Note**
>
> In the Fish shell, the command prompt is not constructed from an environment variable but is generated by shell functions. These prompt functions are redefined by `conda init` regardless of the value of `changeps1`. The redefinition of the Fish prompt functions can be disabled by setting the environment variable `CONDA_DISABLE_FISH_PROMPT` to any value.

add_pip_as_python_dependency: Add pip as Python dependency

Add `pip` as a dependency of `python`. This ensures that `pip` is always installed any time `python` is installed. The default is `True`.

A future conda release may change that default to `False` (see https://github.com/conda/conda/issues/16404). The setting itself will remain supported. If you still want `pip` after the default changes, either install it explicitly:

```
conda create --name myenv python pip
```

or keep the current automatic behavior by setting `add_pip_as_python_dependency` to `True`:

```
conda config --set add_pip_as_python_dependency True
```

Example:

```
add_pip_as_python_dependency: False
```

use_pip: Use pip

Use pip when listing packages with `conda list`. This does not affect any conda command or functionality other than the output of the command `conda list`. The default is `True`.

Example:

```
use_pip: False
```

proxy_servers: Configure conda for use behind a proxy server

By default, proxy settings are pulled from the HTTP_PROXY and HTTPS_PROXY environment variables or the system. Setting them here overrides that default:

```
proxy_servers:
    http: http://user:pass@corp.com:8080
    https: http://user:pass@corp.com:8080
```

Mixing HTTPS and HTTP

The protocol in the URL (either `http://` or `https://`) should match the actual protocol of your proxy server. The keys `http` and `https` in the above example merely indicate the type of traffic to route, not the protocol of the proxy server itself. Ensure that both keys use the correct protocol based on your proxy server's configuration.

To give a proxy for a specific scheme and host, use the `scheme://hostname` form for the key. This matches for any request to the given scheme and exact host name:

```
proxy_servers:
  'http://10.20.1.128': 'http://10.10.1.10:5323'
```

If you do not include the username and password or if authentication fails, conda prompts for a username and password.

If your password contains special characters, you need to escape them as described in Percent-encoding reserved characters on Wikipedia.

Be careful not to use `http` when you mean `https` or `https` when you mean `http`.

ssl_verify: SSL verification

If you are behind a proxy that does SSL inspection, such as a Cisco IronPort Web Security Appliance (WSA), you may need to use `ssl_verify` to override the SSL verification settings.

By default, this variable is `True`, which means that SSL verification is used and conda verifies certificates for SSL connections. Setting this variable to `False` disables the connection's normal security and is not recommended:

```
ssl_verify: False
```

Added in version 23.9.0: The `ssl_verify: truststore` setting is only available with conda 23.9.0 or later.

If the certificate authority is already trusted by the operating system, for instance because it was installed by a system administrator, you can tell conda to use the operating system certificate store by setting `ssl_verify` to "truststore":

```
ssl_verify: truststore
```

You can also set `ssl_verify` to a string path to a certificate, which can be used to verify SSL connections:

```
ssl_verify: corp.crt
```

offline: Offline mode only

Filters out all channel URLs that do not use the `file://` protocol. The default is `False`.

Example:

```
offline: True
```

Advanced configuration

allow_softlinks: Allow soft-linking

When `allow_softlinks` is `True`, conda uses hard links when possible and soft links (symlinks) when hard links are not possible, such as when installing on a different file system than the one that the package cache is on.

When `allow_softlinks` is `False`, conda still uses hard links when possible, but when it is not possible, conda copies files. Individual packages can override this option, specifying that certain files should never be soft linked.

The default is `False`.

Example:

```
allow_softlinks: False
```

channel_alias: Set a channel alias

Whenever you use the `-c` or `--channel` flag to give conda a channel name that is not a URL, conda prepends the `channel_alias` to the name that it was given. The default `channel_alias` is https://conda.anaconda.org.

If `channel_alias` is set to `https://my.anaconda.repo:8080/conda/`, then a user who runs the command `conda install -c conda-forge some-package` will install the package some-package from `https://my.anaconda.repo:8080/conda/conda-forge`.

For example, the command:

```
conda install --channel asmeurer <package>
```

is the same as:

```
conda install --channel https://conda.anaconda.org/asmeurer <package>
```

You can set `channel_alias` to your own repository.

Example: To set `channel_alias` to your repository at https://your.repo.com:

```
channel_alias: https://your.repo/
```

On Windows, you must include a slash ("/") at the end of the URL:

Example: https://your.repo/conda/

When `channel_alias` set to your repository at https://your.repo.com:

```
conda install --channel jsmith <package>
```

is the same as:

```
conda install --channel https://your.repo.com/jsmith <package>
```

create_default_packages: Always add packages by default

When creating new environments, add the specified packages by default. The default packages are installed in every environment you create. You can override this option at the command prompt with the `--no-default-packages` flag. The default is to not include any packages.

Example:

```
create_default_packages:
  - pip
  - ipython
  - scipy=0.15.0
```

track_features: Track features

Enable certain features to be tracked by default. The default is to not track any features. This is similar to adding MKL to the `create_default_packages` list.

Example:

```
track_features:
  - mkl
```

update_dependencies: Disable updating of dependencies

By default, `conda install` updates the given package to the latest version and installs any dependencies necessary for that package. However, if dependencies that satisfy the package's requirements are already installed, conda will not update those packages to the latest version.

In this case, if you would prefer that conda update all dependencies to the latest version that is compatible with the environment, set `update_dependencies` to `True`.

The default is `False`.

Example:

```
update_dependencies: True
```

> **Note**
>
> Conda still ensures that dependency specifications are satisfied. Thus, some dependencies may still be updated or, conversely, this may prevent packages given at the command line from being updated to their latest versions. You can always specify versions at the command line to force conda to install a given version, such as `conda install numpy=1.9.3`.

To avoid updating only specific packages in an environment, a better option may be to pin them. For more information, see *Preventing packages from updating (pinning)*.

`exclude_newer`: Exclude newer packages

Added in version 26.9.0.

Exclude package records with timestamps after the configured cutoff. For a one-off solve, the same value can be provided with the `--exclude-newer` option for `conda create`, `conda install`, or `conda update`. A configured `exclude_newer` policy also filters `conda search` results.

Values may be compact durations such as `7d`, `3d12h`, or `1w`, ISO 8601 durations such as `P7D`, RFC 3339 timestamps such as `2026-04-01T12:00:00Z`, ISO 8601 dates such as `2026-04-01`, or a plain number of seconds. Date-only values are interpreted as the start of the next day in UTC. Set this value to `0` for no delay, using the current time as the cutoff. Leave it empty to disable the policy, which is the default.

Conda prefers the channel-provided `indexed_timestamp` and falls back to `timestamp` if `indexed_timestamp` is absent or zero. The fallback is a builder-controlled build time, not a publication time. Accurate build timestamps can still delay newly built malicious packages, but cannot enforce a reliable full cooldown after publication. Records without a usable timestamp remain eligible. See *Installing packages with a timestamp cutoff* for security benefits, tradeoffs, and limitations.

Example:

```
exclude_newer: 7d
```

Channel-specific cutoffs can be set with `exclude_newer` entries in *channel_settings*, and per-package overrides can be set with *exclude_newer_package*. Package-specific overrides take precedence over channel-specific cutoffs, and channel-specific cutoffs take precedence over the global `exclude_newer` value. Channel and package-specific overrides are configured in `.condarc` only; the `--exclude-newer` command option sets a global cutoff for that command.

`exclude_newer_package`: Exclude newer packages by package name

Added in version 26.9.0.

Override the `exclude_newer` policy for individual package names. Values may use the same duration, timestamp, date, or seconds formats as `exclude_newer`. Set a package value to `false` to exempt that package from the policy.

Example:

```
exclude_newer_package:
  openssl: false
  numpy: 30d
```

`disallow`: Disallow installation of specific packages

Disallow the installation of certain packages. The default is to allow installation of all packages.

Example:

```
disallow:
  - anaconda
```

`add_anaconda_token`: Add Anaconda.org token to automatically see private packages

When the channel alias is Anaconda.org or an Anaconda Server GUI, you can set the system configuration so that users automatically see private packages. Anaconda.org was formerly known as binstar.org. This uses the Anaconda command-line client, which you can install with `conda install anaconda-client`, to automatically add the token to the channel URLs.

The default is `True`.

Example:

```
add_anaconda_token: False
```

> **Note**
>
> Even when set to `True`, this setting is enabled only if the Anaconda command-line client is installed and you are logged in with the `anaconda login` command.

`envs_dirs`: Specify environment directories

Specify directories in which environments are located. If this key is set, the root prefix `envs_dir` is not used unless explicitly included. This key also determines where the package caches are located.

For each envs here, `envs/pkgs` is used as the pkgs cache, except for the standard `envs` directory in the root directory, for which the normal `root_dir/pkgs` is used.

Example:

```
envs_dirs:
  - ~/my-envs
  - /opt/anaconda/envs
```

The `CONDA_ENVS_PATH` environment variable overwrites the `envs_dirs` setting:

- For macOS and Linux: `CONDA_ENVS_PATH=~/my-envs:/opt/anaconda/envs`
- For Windows: `set CONDA_ENVS_PATH=C:\Users\joe\envs;C:\Anaconda\envs`

`pkgs_dirs`: Specify package directories

Specify directories in which packages are located. If this key is set, the root prefix `pkgs_dirs` is not used unless explicitly included.

If the `pkgs_dirs` key is not set, then `envs/pkgs` is used as the pkgs cache, except for the standard `envs` directory in the root directory, for which the normal `root_dir/pkgs` is used.

Example:

```
pkgs_dirs:
  - /opt/anaconda/pkgs
```

The `CONDA_PKGS_DIRS` environment variable overwrites the `pkgs_dirs` setting:

- For macOS and Linux: `CONDA_PKGS_DIRS=/opt/anaconda/pkgs`
- For Windows: `set CONDA_PKGS_DIRS=C:\Anaconda\pkgs`

`use_only_tar_bz2`: Force conda to download only .tar.bz2 packages

Conda 4.7 introduced a new `.conda` package file format. `.conda` is a more compact and faster alternative to `.tar.bz2` packages. It's thus the preferred file format to use where available.

Nevertheless, it's possible to force conda to only download `.tar.bz2` packages by setting the `use_only_tar_bz2` boolean to `True`.

The default is `False`.

Example:

```
use_only_tar_bz2: True
```

> **Note**
>
> This is forced to `True` if conda-build is installed and older than 3.18.3, because older versions of conda break when conda feeds it the new file format.

`console`: Configure display type

Added in version 24.11.0: the `console` setting is only available after this version.

The `console` setting allows you to modify the way output is rendered for conda commands. This setting is primarily used as a way to select new reporter backends made available by plugins.

For example, a plugin may create a new reporter backend called "colors". As a user, you would configure it in your `.condarc` file as shown below:

```
console: colors
```

or specify it on the command line with the `--console` option

```
conda info --console=colors
```

`override_virtual_packages`: Set version or build variables for virtual packages

> **Note**
>
> You can also use `virtual_packages` for this setting.

The `override_virtual_packages` setting is a dictionary that allows you set specific version or build overrides for virtual package installs.

Conda sets virtual package version and/or build numbers in three ways:

1. By passing an enviroment variable with the `conda install` command (highest priority)

   For example: `CONDA_OVERRIDE_CUDA="12.0" conda install -c conda-forge tensorflow`

1. The `override_virtual_packages` dictionary in the `.condarc` file

1. The variables set up in the virtual package's code (lowest priority)

For more information on virtual packages, see Managing virtual packages.

Your `override_virtual_packages` dictionary could look like the following:

```
override_virtual_packages:
  cuda: "12.8"
  glbc: "2.17"
  osx: "11.0"
  archspec: "x86_64"
```

repodata_use_shards: Disable sharded repodata

Added in version 26.5.0.

Sharded repodata splits a channel's package metadata into multiple small files, one per package name, which dramatically speeds up conda operations. This happens because conda only fetches the metadata that a solve needs instead of the whole `repodata.json` file. This makes environment creation and updates noticeably faster on channels that provide shards. See CEP 16 for the format and the *developer guide* for how conda uses it.

Sharded repodata is used by default whenever a channel provides it, and channels without shards fall back to `repodata.json` on their own. There is usually no reason to turn it off. However, if you need to, set `repodata_use_shards` to `false` to always fetch the full `repodata.json`:

```
repodata_use_shards: false
```

For a single command, pass `--no-repodata-use-shards` to any command that accepts channel options, such as `conda create`, `conda install`, or `conda search`. The matching `--repodata-use-shards` option turns it back on for that command.

Conda-build configuration

root-dir: Specify conda-build output root directory

Build output root directory. You can also set this with the `CONDA_BLD_PATH` environment variable. The default is `<CONDA_PREFIX>/conda-bld/`. If you do not have write permissions to `<CONDA_PREFIX>/conda-bld/`, the default is `~/conda-bld/`.

Example:

```
conda-build:
    root-dir: ~/conda-builds
```

output_folder: Specify conda-build build folder (conda-build 3.16.3+)

Folder to dump output package to. Packages are moved here if build or test succeeds. If unset, the output folder corresponds to the same directory as `root-dir`: the root build directory. .. code-block:: yaml

conda-build:
output_folder: conda-bld

pkg_version: Specify conda-build package version

Conda package version to create. Use `2` for `.conda` packages. If not set, conda-build defaults to `.tar.bz2`.

```
conda-build:
   pkg_format: 2
```

anaconda_upload: Automatically upload conda-build packages to Anaconda.org

Automatically upload packages built with conda-build to Anaconda.org. The default is `False`.

Example:

```
anaconda_upload: True
```

`anaconda_token`: Token to be used for Anaconda.org uploads (conda-build 3.0+)

Tokens are a means of authenticating with Anaconda.org without logging in. You can pass your token to conda-build with this `.condarc` setting, or with a CLI argument. This is unset by default. Setting it implicitly enables `anaconda_upload`.

```
conda-build:
    anaconda_token: gobbledygook
```

`quiet`: Limit build output verbosity (conda-build 3.0+)

Conda-build's output verbosity can be reduced with the `quiet` setting. For more verbosity, use the CLI flag `--debug`.

```
conda-build:
    quiet: true
```

`filename_hashing`: Disable filename hashing (conda-build 3.0+)

Conda-build 3 adds hashes to filenames to allow greater customization of dependency versions. If you find this disruptive, you can disable the hashing with the following config entry:

```
conda-build:
    filename_hashing: false
```

> **Warning**
>
> Conda-build does not check when clobbering packages. If you utilize conda-build 3's build matrices with a build configuration that is not reflected in the build string, packages will be missing due to clobbering.

`no_verify`: Disable recipe and package verification (conda-build 3.0+)

By default, conda-build enables optional recipe and package verification. You can disable it:

```
conda-build:
    no_verify: true
```

`set_build_id`: Disable per-build folder creation (conda-build 3.0+)

By default, conda-build creates a new folder for each build, named for the package name plus a timestamp. This allows you to do multiple builds at once. If you have issues with long paths, you may need to disable this behavior. You should first try to change the build output root directory with the `root-dir` setting described above, but fall back to this as necessary:

```
conda-build:
    set_build_id: false
```

`skip_existing`: Skip building packages that already exist (conda-build 3.0+)

By default, conda-build builds all recipes that you specify. You can instead skip recipes that are already built. A recipe is skipped if and only if *all* of its outputs are available on your currently configured channels.

```
conda-build:
    skip_existing: true
```

include_recipe: Omit recipe from package (conda-build 3.0+)

By default, conda-build includes the recipe that was used to build the package. If this contains sensitive or proprietary information, you can omit the recipe.

```
conda-build:
    include_recipe: false
```

> **Note**
>
> If you do not include the recipe, you cannot use conda-build to test the package after the build completes. This means that you cannot split your build and test steps across two distinct CLI commands (`conda build --notest recipe` and `conda build -t recipe`). If you need to omit the recipe and split your steps, your only option is to remove the recipe files from the tarball artifacts after your test step. Conda-build does not provide tools for doing that.

activate: Disable activation of environments during build/test (conda-build 3.0+)

By default, conda-build activates the build and test environments prior to executing the build or test scripts. This adds necessary PATH entries, and also runs any activate.d scripts you may have. If you disable activation, the PATH will still be modified, but the activate.d scripts will not run. This is not recommended, but some people prefer this.

```
conda-build:
    activate: false
```

long_test_prefix: Disable long prefix during test (conda-build 3.16.3+)

By default, conda-build uses a long prefix for the test prefix. If you have recipes that fail in long prefixes but would still like to test them in short prefixes, you can disable the long test prefix. This is not recommended.

```
conda-build:
    long_test_prefix: false
```

The default is `true`.

pypirc: PyPI upload settings (conda-build 3.0+)

Unset by default. If you have wheel outputs in your recipe, conda-build will try to upload them to the PyPI repository specified by the `pypi_repository` setting using credentials from this file path.

```
conda-build:
    pypirc: ~/.pypirc
```

pypi_repository: PyPI repository to upload to (conda-build 3.0+)

Unset by default. If you have wheel outputs in your recipe, conda-build will try to upload them to this PyPI repository using credentials from the file specified by the `pypirc` setting.

```
conda-build:
    pypi_repository: pypi
```

Expansion of environment variables

Conda expands environment variables in a subset of configuration settings. These are:

- channel
- channel_alias
- channels
- client_cert_key
- client_cert
- custom_channels
- custom_multichannels
- default_channels
- envs_dirs
- envs_path
- migrated_custom_channels
- pkgs_dirs
- proxy_servers
- verify_ssl
- allowlist_channels
- denylist_channels

This allows you to store the credentials of a private repository in an environment variable, like so:

```
channels:
  - https://${USERNAME}:${PASSWORD}@my.private.conda.channel
```

Configuring number of threads

You can use your `.condarc` file or environment variables to add configuration to control the number of threads. You may want to do this to tweak conda to better utilize your system. If you have a very fast SSD, you might increase the number of threads to shorten the time it takes for conda to create environments and install/remove packages.

repodata_threads

- Default number of threads: None
- Threads used when downloading, parsing, and creating repodata structures from `repodata.json` files. Multiple downloads from different channels may occur simultaneously. This speeds up the time it takes to start solving.

verify_threads

- Default number of threads: 1
- Threads used when verifying the integrity of packages and files to be installed in your environment. Defaults to 1, as using multiple threads here can run into problems with slower hard drives.

execute_threads

- Default number of threads: 1
- Threads used to unlink, remove, link, or copy files into your environment. Defaults to 1, as using multiple threads here can run into problems with slower hard drives.

default_threads

- Default number of threads: None
- When set, this value is used for all of the above thread settings. With its default setting (None), it does not affect the other settings.

Setting any of the above can be done in `.condarc` or with conda config:

At your terminal:

```
conda config --set repodata_threads 2
```

In `.condarc`:

```
verify_threads: 4
```

Administering a multi-user conda installation

By default, conda and all of the packages it installs are installed locally with a user-specific configuration. Administrative privileges are not required, and no upstream files or other users are affected by the installation.

You can make conda and any number of packages available to a group of one or more users, while preventing these users from installing unwanted packages with conda:

1. Install conda and the allowed packages, if any, in a location that is under administrator control and accessible to users.
2. Create a *.condarc system configuration file* in the root directory of the installation. This system-level configuration file will override any user-level configuration files installed by the user.

Each user accesses the central conda installation, which reads settings from the user `.condarc` configuration file located in their home directory. The path to the user file is the same as the root environment prefix displayed by `conda info`, as shown in *User configuration file* below. The user `.condarc` file is limited by the system `.condarc` file.

System configuration settings are commonly used in a system `.condarc` file but may also be used in a user `.condarc` file. All user configuration settings may also be used in a system `.condarc` file.

For information about settings in the `.condarc` file, see *Using the .condarc conda configuration file*.

Example administrator-controlled installation

The following example describes how to view the system configuration file, review the settings, compare it to a user's configuration file, and determine what happens when the user attempts to access a file from a blocked channel. It then describes how the user must modify their configuration file to access the channels allowed by the administrator.

System configuration file

1. The system configuration file must be in the top-level conda installation directory. Check the path where `conda` is located, e.g. in a miniconda installation

```
$ which conda
/tmp/miniconda/bin/conda
```

2. View the contents of the `.condarc` file in the administrator's directory:

```
cat /tmp/miniconda/.condarc
```

   The following administrative `.condarc` file uses the `#!final` flag to specify the channels, default channels, and channel_alias available to the user.

```
$ cat /tmp/miniconda/.condarc

channels:                                    #!final
  - admin

channel_alias: https://conda.anaconda.org/  #!final
```

The `#!final` flag is very similar to the `!important` rule in CSS; any parameter within the `.condarc` that is trailed by the `#!final` cannot be overwritten by any other `.condarc` source. For more information on this flag, see the Anaconda Blog on the subject.

Because the `#!final` flag has been used and the channel defaults are not explicitly specified, users are disallowed from downloading packages from the default channels. You can check this in the next procedure.

User configuration file

1. Check the location of the user's conda installation:

```
$ conda info
Current conda install:
. . .
       channel URLs : https://repo.anaconda.com/pkgs/free/osx-64/
                      https://repo.anaconda.com/pkgs/pro/osx-64/
       config file : /Users/username/.condarc
```

   The `conda info` command shows that conda is using the user's `.condarc` file, located at `/Users/username/.condarc` and that the default channels such as `repo.anaconda.com` are listed as channel URLs.

2. View the contents of the administrative `.condarc` file in the directory that was located in step 1:

```
$ cat ~/.condarc
channels:
  - defaults
```

   This user's `.condarc` file specifies only the default channels, but the administrator config file has blocked default channels by specifying that only `admin` is allowed. If this user attempts to search for a package in the default channels, they get a message telling them what channels are allowed:

```
$ conda search flask
Fetching package metadata:
Error: URL 'http://repo.anaconda.com/pkgs/pro/osx-64/' not
```

(continues on next page)

(continued from previous page)

```
in allowed channels.
Allowed channels are:
 - https://conda.anaconda.org/admin/osx-64/
```

This error message tells the user to add the `admin` channel to their configuration file.

3. The user must edit their local `.condarc` configuration file to access the package through the admin channel:

```
channels:
  - admin
```

The user can now search for packages in the allowed `admin` channel.

Shared conda installations

For clusters, HPC systems, or organizations, you can provide a centralized conda installation that multiple users can access.

Basic setup

1. Install conda to a shared location:

```
bash Miniconda3-latest-Linux-x86_64.sh -b -p /opt/conda
chmod -R a+rX,go-w /opt/conda
```

2. Users activate conda by sourcing the initialization script:

```
source /opt/conda/etc/profile.d/conda.sh
```

Shared package cache and environments

Create a system `.condarc` at `/opt/conda/.condarc` to configure shared locations:

```
pkgs_dirs:
  - /shared/conda/pkgs    # Shared cache
  - ~/.conda/pkgs          # User fallback

envs_dirs:
  - /shared/conda/envs     # Shared environments
  - ~/.conda/envs          # User fallback
```

Set up the shared directories:

```
mkdir -p /shared/conda/pkgs /shared/conda/envs
chmod -R a+rX,go-w /shared/conda/pkgs  # Read-only cache
```

Best practices

- Keep the conda installation read-only; only admins should update it
- Use shared package cache to reduce redundant downloads
- Implement disk quotas on user home directories
- For HPC systems, integrate with module systems (e.g., Lmod)

- Document for users: how to activate conda, where to create environments, and support contacts

Mirroring channels

The conda configuration system has several keys that can be used to set up a mirrored context.

The default setup

By default, `conda` can serve packages from two main locations:

- `repo.anaconda.com`: this is where `defaults` points to by default. The default value has been marked for pending deprecation as of conda 24.9.0, but it is still the default for older versions. In conda 25.3.0, this default will be removed.

  This base location is hardcoded in the default value of `default_channels`:

  - `https://repo.anaconda.com/pkgs/main`
  - `https://repo.anaconda.com/pkgs/r`
  - `https://repo.anaconda.com/pkgs/msys2`

- `conda.anaconda.org`: this is where conda clients look up community channels like `conda-forge` or `bioconda`. This base location can be configured via `channel_alias`.

So, when it comes to mirroring these channels, you have to account for those two locations.

Mirror `defaults`

Use `default_channels` to overwrite the *default configuration*. For example:

```
default_channels:
    - https://my-mirror.com/pkgs/main
    - https://my-mirror.com/pkgs/r
    - https://my-mirror.com/pkgs/msys2
```

Mirror all community channels

Redefine `channel_alias` to point to your mirror. For example:

```
channel_alias: https://my-mirror.com
```

This will make `conda` look for all community channels at `https://my-mirror.com/conda-forge`, `https://my-mirror.com/bioconda`, etc.

Mirror only some community channels

If you want to mirror only some community channels, you must use `custom_channels`. This takes precedence over `channel_alias`. For example:

```
custom_channels:
    conda-forge: https://my-mirror.com/conda-forge
```

With this configuration, conda-forge will be looked up at `https://my-mirror.com/conda-forge`. All other community channels will be looked up at `https://conda.anaconda.org`.

> **Note**
>
> Feel free to explore all the available options in *Configuration*.

Disabling SSL verification

Using conda with SSL is strongly recommended, but it is possible to disable SSL and it may be necessary to disable SSL in certain cases.

Some corporate environments use proxy services that use Man-In-The-Middle (MITM) attacks to sniff encrypted traffic. These services can interfere with SSL connections such as those used by conda and pip to download packages from repositories such as PyPI.

If you encounter this interference, you should set up the proxy service's certificates so that the `requests` package used by conda can recognize and use the certificates.

For cases where this is not possible, conda-build versions 3.0.31 and higher have an option that disables SSL certificate verification and allows this traffic to continue.

`conda skeleton pypi` can disable SSL verification when pulling packages from a PyPI server over HTTPS.

> **Warning**
>
> This option causes your computer to download and execute arbitrary code over a connection that it cannot verify as secure. This is not recommended and should only be used if necessary. Use this option at your own risk.

To disable SSL verification when using `conda skeleton pypi`, set the `SSL_NO_VERIFY` environment variable to either `1` or `True` (case insensitive).

On *nix systems:

```
SSL_NO_VERIFY=1 conda skeleton pypi a_package
```

And on Windows systems:

```
set SSL_NO_VERIFY=1
conda skeleton pypi a_package
set SSL_NO_VERIFY=
```

We recommend that you unset this environment variable immediately after use. If it is not unset, some other tools may recognize it and incorrectly use unverified SSL connections.

Using this option will cause `requests` to emit warnings to STDERR about insecure settings. If you know that what you're doing is safe, or have been advised by your IT department that what you're doing is safe, you may ignore these warnings.

Disabling SSL verification via conda settings

In addition to disabling SSL via environment variables, you can disable it by setting *ssl_verify* to *false* in your config files. To do so, run the following commands to disable and enable it:

```
conda config --set ssl_verify False
# Run conda commands with SSL disabled
conda config --set ssl_verify True
```

Using non-standard certificates

Using conda behind a firewall may require using a non-standard set of certificates, which requires custom settings.

If you are using a non-standard set of certificates, then the requests package requires the setting of `REQUESTS_CA_BUNDLE`. If you receive an error with self-signed certifications, you may consider unsetting `REQUESTS_CA_BUNDLE` as well as `CURL_CA_BUNDLE` and disabling SSL verification to create a conda environment over HTTP.

You may need to set the conda environment to use the root certificate provided by your company rather than conda's generic ones.

One workflow to resolve this on macOS is:

- Open Chrome, got to any website, click on the lock icon on the left of the URL. Click on «Certificate» on the dropdown. In the next window you see a stack of certificates. The uppermost (aka top line in window) is the root certificate (e.g. Zscaler Root CA).
- Open macOS keychain, click on «Certificates» and choose among the many certificates the root certificate that you just identified. Export this to any folder of your choosing.
- Convert this certificate with OpenSSL: `openssl x509 -inform der -in /path/to/your/certificate.cer -out /path/to/converted/certificate.pem`
- For a quick check, set your shell to acknowledge the certificate: `export REQUESTS_CA_BUNDLE=/path/to/converted/certificate.pem`
- To set this permanently, open your shell profile (e.g. `.bashrc` or `.zshrc`) and add this line: `export REQUESTS_CA_BUNDLE=/path/to/converted/certificate.pem`. Now exit your terminal/shell and reopen. Check again.

Using Custom Locations for Environment and Package Cache

For any given conda installation, the two largest folders in terms of disk space are often the `envs` and `pkgs` folders that store created environments and downloaded packages, respectively. If the location where conda is installed has limited disk space and another location with more disk space is available on the same computer, we can change where conda saves its environments and packages with the settings `envs_dirs` and `pkgs_dirs`, respectively.

Assuming conda is installed in the user's home directory and the the folder `/nfs/volume/user` with more disk space is writable, the best way to configure this is by adding the following entries to the `.condarc` file in the user's home directory:

```
envs_dirs:
  - /nfs/volume/user/conda_envs
pkgs_dirs:
  - /nfs/volume/user/conda_pkgs
```

In the example above, we tell conda to use the folder `/nfs/volume/user/conda_envs` to store all of the environments we create, and we tell conda to use `/nfs/volume/user/conda_pkgs` to store all of the packages that we download.

To save even more space, the contents of `/nfs/volume/user/conda_pkgs` will be hard linked to the environments in `/nfs/volume/user/conda_envs` when possible. This means that `pkgs_dirs` will normally take up the most space for a conda installation. But, when hard linking is not possible, the files will be copied over to the environment which means each new environment increases the amount of disk space taken. To ensure this hard linking works properly, we recommend to always store the `envs_dirs` and `pkgs_dirs` on the same mounted volume.

Configuring Temporary File Locations

Conda creates various temporary files during its operations. You can control where these temporary files are stored using standard operating system environment variables.

Why configure temporary file locations?

You may need to configure temporary file locations when:

- Default temporary directories have limited disk space
- Default locations are read-only (common in containers, HPC environments, or restrictive permissions)
- Compliance requirements mandate temporary files be stored in specific locations
- Performance considerations (e.g., using faster storage for temporary operations)
- Working in multi-user or shared computing environments

Environment variables

Conda respects the standard operating system environment variables for temporary file locations. Python's `tempfile` module (specifically `tempfile.gettempdir()`), which conda uses internally, checks these variables in the following order on **all platforms**:

1. `TMPDIR`
2. `TEMP`
3. `TMP`

If none of these environment variables are set, Python falls back to platform-specific default locations:

- **Windows**: `C:\TEMP`, `C:\TMP`, `\TEMP`, `\TMP` (in that order)
- **Unix/Linux/macOS**: `/tmp`, `/var/tmp`, `/usr/tmp` (in that order)
- **All platforms**: As a last resort, the current working directory

Setting temporary directories

Unix/Linux/macOS:

```
# For current session
export TMPDIR=/path/to/writable/tmp
mkdir -p $TMPDIR

# To make permanent, add to ~/.bashrc or equivalent shell profile file
echo 'export TMPDIR=/path/to/writable/tmp' >> ~/.bashrc

# For a single command
TMPDIR=/path/to/writable/tmp conda install package_name
```

Windows:

```
REM For current session (Command Prompt)
set TEMP=C:\path\to\writable\tmp
md %TEMP%
```

```
# For current session (PowerShell)
$env:TEMP = "C:\path\to\writable\tmp"
New-Item -ItemType Directory -Path $env:TEMP -Force

# To make permanent, use the System Properties dialog:
# - Edit the system environment variables or
# - Edit environment variables for your account
```

Container/Docker environments:

In your `Dockerfile` or `docker-compose.yml`:

```
FROM continuumio/miniconda3

# Set temporary directory to a writable location with more space
ENV TMPDIR=/opt/conda/tmp
RUN mkdir -p /opt/conda/tmp
```

Verifying temporary directory location

To check where conda will create temporary files, use `conda info`:

```
conda info
```

Look for the `temporary directory` line in the output, which shows the currently configured temporary directory location.

Alternatively, you can check directly with Python:

```
python -c "import tempfile; print(tempfile.gettempdir())"
```

This will show you the directory that Python (and therefore conda) will use for temporary files.

See also

- *Temporary file errors during conda operations* - Troubleshooting temporary file errors (permission denied, read-only filesystem, no space left on device)
- *Using Custom Locations for Environment and Package Cache* - Configure locations for environments and package cache
- *Settings* - Complete list of conda configuration settings

Improving interoperability with pip

> **Note**
>
> Conda 4.6.0 added improved support for interoperability between conda and pip. This involved the now deprecated `prefix_data_interoperability` configuration setting. For more information on this deprecated feature, see older versions of these docs.

Conda can install supported packages from PyPI using the `conda-pypi` channel. For setup instructions and supported workflows, see *Installing packages from PyPI with conda*.

The following pages have information on how conda can be customized further through configuration.

Using the .condarc conda configuration file
: Learn how to use a settings file (`.condarc`) to override defaults and maintain settings across shell sessions

Settings
: View a list and definition of all the configuration settings that can be used within conda

Administering a multi-user conda installation
: How to set up conda as a system administrator for use by multiple users

Mirroring channels
: Explore how to configure your own channel server mirror with conda

Disable SSL Verification
: Disabling SSL may be necessary in very limited circumstances; learn how here

Using non-standard certificates
: Install and configure non-standard certifications for use with conda

Using Custom Locations for Environment and Package Cache
: How to configure different locations for storing environments and package cache

Configuring Temporary File Locations
: Control where conda creates temporary files using `TMPDIR`/`TEMP`/`TMP` environment variables

Pip interoperability (experimental)
: An experimental feature that makes conda operate better with pip (no longer supported)

4.1.5 Concepts

Commands

The `conda` command is the primary command-line interface for managing conda environments and packages. You can use conda commands to create environments, install packages, update packages, remove packages, search for packages, and inspect your current conda configuration.

Common commands

Some of the most commonly used conda commands include:

`conda create`
: Creates a new conda environment.

```
$ conda create --name myenv python=3.12
```

`conda install`
: Installs packages into the active environment or into a specified environment.

```
$ conda install numpy pandas
```

`conda update`
: Updates packages in an environment.

```
$ conda update numpy
```

`conda remove`
: Removes packages from an environment.

```
$ conda remove numpy
```

Other commands are available for managing channels, listing environments, searching for packages, cleaning package caches, and viewing conda configuration.

> **Tip**
>
> You can abbreviate many frequently used command options that are preceded by two dashes (`--`) to one dash and the first letter of the option. For example, `--name` and `--envs` can be written as `-n` and `-e`.

Getting help for commands

Use `--help` or `-h` to see help for conda itself or for an individual command.

To see top-level conda help:

```
$ conda --help
```

To see help for a specific command:

```
$ conda install --help
```

or:

```
$ conda install -h
```

For full usage of each command, including abbreviations, see *commands*. You can see the same information at the command line by *viewing the command-line help*.

Plugin commands

Conda plugins can add their own commands or subcommands to the conda command-line interface. For example, a plugin may add a command for a custom workflow, package source, or environment management task.

Plugins that add conda commands must be installed in the environment running conda, usually `base`. Starting with conda 26.11.0, use `conda plugins install`, `conda plugins update`, and `conda plugins remove` to manage their conda packages. Select the target environment with `--name` or `--prefix`.

Use `conda plugins list` to see installed plugins and `conda plugins info NAME` to inspect a plugin's metadata and registered hooks. These commands report on the environment running conda and also support `--json` output.

Installing a plugin into another environment does not make its command available to the main `conda` command. That environment must contain conda, and you must run its Python interpreter to load its plugins. See *Plugins* for installation examples and *conda plugins* for the command reference.

Packages

What is a package?

A package is a compressed tarball file (`.tar.bz2`) or `.conda` file that contains:

- system-level libraries.
- Python or other modules.
- executable programs and other components.
- metadata under the `info/` directory.
- a collection of files that are installed directly into an `install` prefix.

Conda keeps track of the dependencies between packages and platforms. The conda package format is identical across platforms and operating systems.

Only files, including symbolic links, are part of a conda package. Directories are not included. Directories are created and removed as needed, but you cannot create an empty directory from the tar archive directly.

.conda file format

The .conda file format was introduced in conda 4.7 as a more compact, and thus faster, alternative to a tarball.

The .conda file format consists of an outer, uncompressed ZIP-format container, with 2 inner compressed .tar files.

For the .conda format's initial internal compression format support, we chose Zstandard (zstd). The actual compression format used does not matter, as long as the format is supported by libarchive. The compression format may change in the future as more advanced compression algorithms are developed and no change to the .conda format is necessary. Only an updated libarchive would be required to add a new compression format to .conda files.

These compressed files can be significantly smaller than their bzip2 equivalents. In addition, they decompress much more quickly. .conda is the preferred file format to use where available, although we continue to provide .tar.bz2 files in tandem.

Read more about the introduction of the .conda file format.

Note

In conda 4.7 and later, you cannot use package names that end in ".conda" as they conflict with the .conda file format for packages.

Using packages

- You may search for packages

```
conda search scipy
```

- You may install a package

```
conda install scipy
```

- You may build a package after installing conda-build

```
conda build my_fun_package
```

Package structure

```
.
├── bin
│   └── pyflakes
├── info
│   ├── LICENSE.txt
│   ├── files
│   ├── index.json
│   ├── paths.json
│   └── recipe
└── lib
    └── python3.5
```

- bin contains relevant binaries for the package.
- lib contains the relevant library files (eg. the .py files).
- info contains package metadata.

Metapackages

When a conda package is used for metadata alone and does not contain any files, it is referred to as a metapackage. The metapackage may contain dependencies to several core, low-level libraries and can contain links to software files that are automatically downloaded when executed. Metapackages are used to capture metadata and make complicated package specifications simpler.

An example of a metapackage is "anaconda," which collects together all the packages in the Anaconda Distribution installer. The command `conda create -n envname anaconda` creates an environment that exactly matches what would be created from the Anaconda Distribution installer. You can create metapackages with the `conda metapackage` command. Include the name and version in the command.

Anaconda metapackage

The Anaconda metapackage is used in the creation of the Anaconda Distribution installers so that they have a set of packages associated with them. Each installer release has a version number, which corresponds to a particular collection of packages at specific versions. That collection of packages at specific versions is encapsulated in the Anaconda metapackage.

The Anaconda metapackage contains several core, low-level libraries, including compression, encryption, linear algebra, and some GUI libraries.

Read more about the Anaconda metapackage and Anaconda Distribution.

Mutex metapackages

A mutex metapackage is a very simple package that has a name. It need not have any dependencies or build steps. Mutex metapackages are frequently an "output" in a recipe that builds some variant of another package. Mutex metapackages function as a tool to help achieve mutual exclusivity among packages with different names.

Let's look at some examples for how to use mutex metapackages to build NumPy against different BLAS implementations.

Building NumPy with BLAS variants

If you build NumPy with MKL, you also need to build SciPy, scikit-learn, and anything else using BLAS also with MKL. It is important to ensure that these "variants" (packages built with a particular set of options) are installed together and never with an alternate BLAS implementation. This is to avoid crashes, slowness, or numerical problems. Lining up these libraries is both a build-time and an install-time concern. We'll show how to use metapackages to achieve this need.

Let's start with the metapackage `blas=1.0=mkl`: https://github.com/AnacondaRecipes/intel_repack-feedstock/blob/e699b12/recipe/meta.yaml#L108-L112

Note that `mkl` is a string of `blas`.

That metapackage is automatically added as a dependency using `run_exports` when someone uses the mkl-devel package as a build-time dependency: https://github.com/AnacondaRecipes/intel_repack-feedstock/blob/e699b12/recipe/meta.yaml#L124

By the same token, here's the metapackage for OpenBLAS: https://github.com/AnacondaRecipes/openblas-feedstock/blob/ae5af5e/recipe/meta.yaml#L127-L131

And the `run_exports` for OpenBLAS, as part of openblas-devel: https://github.com/AnacondaRecipes/openblas-feedstock/blob/ae5af5e/recipe/meta.yaml#L100

Fundamentally, conda's model of mutual exclusivity relies on the package name. OpenBLAS and MKL are obviously not the same package name, and thus are not mutually exclusive. There's nothing stopping conda from installing both at once. There's nothing stopping conda from installing NumPy with MKL and SciPy with OpenBLAS. The metapackage is what allows us to achieve the mutual exclusivity. It unifies the options on a single package name, but with a different build string. Automating the addition of the metapackage with `run_exports` helps ensure the library consumers (package builders who depend on libraries) will have correct dependency information to achieve the unified runtime library collection.

Installing NumPy with BLAS variants

To specify which variant of NumPy that you want, you could potentially specify the BLAS library you want:

```
conda install numpy mkl
```

However, that doesn't actually preclude OpenBLAS from being chosen. Neither MKL nor its dependencies are mutually exclusive (meaning they do not have similar names and different version/build-string).

This pathway may lead to some ambiguity and solutions with mixed BLAS, so using the metapackage is recommended. To specify MKL-powered NumPy in a non-ambiguous way, you can specify the mutex package (either directly or indirectly):

```
conda install numpy "blas=*=mkl"
```

There is a simpler way to address this, however. For example, you may want to try another package that has the desired mutex package as a dependency.

OpenBLAS has this with its "nomkl" package: https://github.com/AnacondaRecipes/openblas-feedstock/blob/ae5af5e/recipe/meta.yaml#L133-L147

Nothing should use "nomkl" as a dependency. It is strictly a utility for users to facilitate switching from MKL (which is the default) to OpenBLAS.

How did MKL become the default? The solver needs a way to prioritize some packages over others. We achieve that with an older conda feature called track_features that originally served a different purpose.

Track_features

One of conda's optimization goals is to minimize the number of track_features needed to specify the desired specs. By adding track_features to one or more of the options, conda will de-prioritize it or "weigh it down." The lowest priority package is the one that would cause the most track_features to be activated in the environment. The default package among many variants is the one that would cause the least track_features to be activated.

There is a catch, though: any track_features must be unique. No two packages can provide the same track_feature. For this reason, our standard practice is to attach track_features to the metapackage associated with what we want to be non-default.

Take another look at the OpenBLAS recipe: https://github.com/AnacondaRecipes/openblas-feedstock/blob/ae5af5e/recipe/meta.yaml#L127-L137

This attached track_features entry is why MKL is chosen over OpenBLAS. MKL does not have any track_features associated with it. If there are 3 options, you would attach 0 track_features to the default, then 1 track_features for the next preferred option, and finally 2 for the least preferred option. However, since you generally only care about the one default, it is usually sufficient to add 1 track_feature to all options other than the default option.

More info

For reference, the Visual Studio version alignment on Windows also uses mutex metapackages. https://github.com/AnacondaRecipes/aggregate/blob/9635228/vs2017/meta.yaml#L24

Noarch packages

Noarch packages are packages that are not architecture specific and therefore only have to be built once. Noarch packages are either generic or Python. Noarch generic packages allow users to distribute docs, datasets, and source code in conda packages. Noarch Python packages are described below.

Declaring these packages as `noarch` in the `build` section of the `meta.yaml` reduces shared CI resources. Therefore, all packages that qualify to be noarch packages should be declared as such.

Noarch Python

The `noarch: python` directive in the build section makes pure-Python packages that only need to be built once.

Noarch Python packages cut down on the overhead of building multiple different pure Python packages on different architectures and Python versions by sorting out platform and Python version-specific differences at install time.

In order to qualify as a noarch Python package, all of the following criteria must be fulfilled:

- No compiled extensions.
- No post-link, pre-link, or pre-unlink scripts.
- No OS-specific build scripts.
- No Python version-specific requirements.
- No skips except for Python version. If the recipe is py3 only, remove skip statement and add version constraint on Python in host and run section.
- 2to3 is not used.
- Scripts argument in setup.py is not used.
- If `console_script` entrypoints are in setup.py, they are listed in `meta.yaml`.
- No activate scripts.
- Not a dependency of conda.

Note

While `noarch: python` does not work with selectors, it does work with version constraints. `skip: True # [py2k]` can sometimes be replaced with a constrained Python version in the host and run subsections, for example: `python >=3` instead of just `python`.

Note

Only `console_script` entry points have to be listed in `meta.yaml`. Other entry points do not conflict with `noarch` and therefore do not require extra treatment.

Read more about conda's noarch packages.

Link and unlink scripts

You may optionally execute scripts before and after the link and unlink steps. For more information, see Adding pre-link, post-link, and pre-unlink scripts.

More information

For more information, go for a deeper dive in our *managing packages guide*. Learn more about package metadata, repository structure and index, and package match specifications at *Package specifications*.

Package specification

Package metadata

The `info/` directory contains all metadata about a package. Files in this location are not installed under the install prefix. Although you are free to add any file to this directory, conda only inspects the content of the files discussed below.

Info

- files
  - a list of all the files in the package (not included in `info/`)
- `index.json`
  - metadata about the package including platform, version, dependencies, and build info

```
{
  "arch": "x86_64",
  "build": "py37hfa4b5c9_1",
  "build_number": 1,
  "depends": [
    "depend > 1.1.1"
  ],
  "license": "BSD 3-Clause",
  "name": "fun-packge",
  "platform": "linux",
  "subdir": "linux-64",
  "timestamp": 1535416612069,
  "version": "0.0.0"
}
```

- `paths.json`
  - a list of files in the package, along with their associated SHA-256, size in bytes, and the type of path (eg. hardlink vs. softlink)

```
{
  "paths": [
    {
      "_path": "lib/python3.7/site-packages/fun-packge/__init__.py",
      "path_type": "hardlink",
      "sha256": "76f3b6e34feeb651aff33ca59e0279c4eadce5a50c6ad93b961c846f7ba717e9",
      "size_in_bytes": 2067
    },
```

(continues on next page)

(continued from previous page)

```
    {
      "_path": "lib/python3.7/site-packages/fun-packge/__config__.py",
      "path_type": "hardlink",
      "sha256": "348e3602616c1fe4c84502b1d8cf97c740d886002c78edab176759610d287f06",
      "size_in_bytes": 87519
    },
    ...
}
```

info/index.json

This file contains basic information about the package, such as name, version, build string, and dependencies. The content of this file is stored in `repodata.json`, which is the repository index file, hence the name `index.json`. The JSON object is a dictionary containing the keys shown below. The filename of the conda package is composed of the first 3 values, as in: `<name>-<version>-<build>.tar.bz2`.

Key	Type	Description
name	string	The lowercase name of the package. May contain the "-" character.
version	string	The package version. May not contain "-". Conda acknowledges PEP 440.
build	string	The build string. May not contain "-". Differentiates builds of packages with otherwise identical names and versions, such as: • A build with other dependencies, such as Python 3.4 instead of Python 2.7. • A bug fix in the build process. • Some different optional dependencies, such as MKL versus ATLAS linkage. Nothing in conda actually inspects the build string. Strings such as `np18py34_1` are designed only for human readability and conda never parses them.
build_number	integer	A non-negative integer representing the build number of the package. Unlike the build string, the build_number is inspected by conda. Conda uses it to sort packages that have otherwise identical names and versions to determine the latest one. This is important because new builds that contain bug fixes for the way a package is built may be added to a repository.
depends	list of strings	A list of dependency specifications, where each element is a string, as outlined in *Package match specifications*.
arch	string	Optional. The architecture the package is built for. EXAMPLE: `x86_64` Conda currently does not use this key.
platform	string	Optional. The OS that the package is built for. EXAMPLE: `osx` Conda currently does not use this key. Packages for a specific architecture and platform are usually distinguished by the repository subdirectory that contains them---see *Repository structure and index*.

info/files

Lists all files that are part of the package itself, 1 per line. All of these files need to get linked into the environment. Any files in the package that are not listed in this file are not linked when the package is installed. The directory delimiter for the files in `info/files` should always be "/", even on Windows. This matches the directory delimiter used in the

tarball.

info/has_prefix

Optional file. Lists all files that contain a hard-coded build prefix or placeholder prefix, which needs to be replaced by the install prefix at installation time.

> **Note**
>
> Due to the way the binary replacement works, the placeholder prefix must be longer than the install prefix.

Each line of this file should be either a path, in which case it is considered a text file with the default placeholder `/opt/anaconda1anaconda2anaconda3`, or a space-separated list of placeholder, mode, and path, where:

- Placeholder is the build or placeholder prefix.
- Mode is either `text` or `binary`.
- Path is the relative path of the file to be updated.

EXAMPLE: On Windows:

```
"Scripts/script1.py"
"C:\Users\username\anaconda\envs\_build" text "Scripts/script2.bat"
"C:/Users/username/anaconda/envs/_build" binary "Scripts/binary"
```

EXAMPLE: On macOS or Linux:

```
bin/script.sh
/Users/username/anaconda/envs/_build binary bin/binary
/Users/username/anaconda/envs/_build text share/text
```

> **Note**
>
> The directory delimiter for the relative path must always be "/", even on Windows. The placeholder may contain either "\" or "/" on Windows, but the replacement prefix will match the delimiter used in the placeholder. The default placeholder `/opt/anaconda1anaconda2anaconda3` is an exception, being replaced with the install prefix using the native path delimiter. On Windows, the placeholder and path always appear in quotes to support paths with spaces.

info/license.txt

Optional file. The software license for the package.

info/no_link

Optional file. Lists all files that cannot be linked - either soft-linked or hard-linked - into environments and are copied instead.

info/about.json

Optional file. Contains the entries in the about section of the `meta.yaml` file. The following keys are added to `info/about.json` if present in the build recipe:

- home
- dev_url
- doc_url
- license_url
- license
- summary
- description
- license_family

info/recipe

A directory containing the full contents of the build recipe.

meta.yaml.rendered

The fully rendered build recipe. See conda render.

This directory is present only when the the include_recipe flag is `True` in the build section.

Repository structure and index

A conda repository - or channel - is a directory tree, usually served over HTTPS, which has platform subdirectories, each of which contain conda packages and a repository index. The index file `repodata.json` lists all conda packages in the platform subdirectory. Use `conda index` to create such an index from the conda packages within a directory. It is simple mapping of the full conda package filename to the dictionary object in `info/index.json` described in link scripts.

In the following example, a repository provides the conda package `misc-1.0-np17py27_0.tar.bz2` on 64-bit Linux and 32-bit Windows:

```
<some path>/linux-64/repodata.json
                    repodata.json.bz2
                    misc-1.0-np17py27_0.tar.bz2
          /win-32/repodata.json
                  repodata.json.bz2
                  misc-1.0-np17py27_0.tar.bz2
```

> **Note**
>
> Both conda packages have identical filenames and are distinguished only by the repository subdirectory that contains them.

Package match specifications

Conda accepts package specifications on the command line with commands such as `conda install` and `conda create`. Common command-line forms are described in *Command-line package specifications*.

Internally, conda translates command-line package specifications into canonical match specifications.

EXAMPLE: python=3.9 is translated to python 3.9.*.

Command-line package specifications

The `package_spec` arguments accepted by commands such as `conda install` and `conda create` are package match specifications written in a command-line friendly form. They can be as simple as a package name or can include version, build, channel, and subdir constraints.

Common forms include:

Form	Meaning
`numpy`	Match any available `numpy` package that is compatible with the environment and configured channels.
`numpy=1.11`	Match the `1.11` release series. This is a fuzzy version match and is translated internally to a version pattern such as `numpy 1.11.*`.
`numpy==1.11`	**Exact match for version 1.11. This also matches 1.11.0 and 1.11.0.0, but not 1.11.1.** Unlike numpy=1.11, it does not match the whole 1.11 series.
`numpy>=1.8`	Match versions greater than or equal to `1.8`.
`numpy>=1.8,<2`	Match versions greater than or equal to `1.8` and less than `2`.
`numpy=1.11.2=*nomkl*`	Match `numpy` version `1.11.2` with a build string matching `*nomkl*`.
`conda-forge::numpy`	Match `numpy` from the `conda-forge` channel.
`conda-forge/linux-64::numpy`	Match `numpy` from the `linux-64` subdir of the `conda-forge` channel.

Supported version operators include <, >, <=, >=, ==, and !=. Commas separate constraints that must all be true (AND), while | separates alternatives (OR). Wildcards such as * can be used in version and build string constraints.

When entering package specifications in a shell, quote specifications that contain characters interpreted by the shell, such as <, >, *, or |. For example:

```
conda install "numpy>=1.8,<2"
conda create -n science-env "python>=3.11" scipy
```

If no version constraint is provided, conda attempts to install a package that satisfies the requested name, the configured channels, the platform, and the other constraints in the environment.

Canonical match specifications

CEP 29 describes the canonical package match specification grammar used by conda.

Package dependencies are specified using a match specification. A match specification is a space-separated string of 1, 2, or 3 parts:

- The first part is always the exact name of the package.
- The second part refers to the version and may contain special characters:
  - | means OR.

    EXAMPLE: `1.0|1.2` matches version 1.0 or 1.2

  - * at the end of a version is a fuzzy match. `1.4*` means the same as `1.4.*`: any version whose leading segments are `1.4`. Conda compares whole segments here, so `1.4*` does not match `1.40` or `14.0`. Use the `.*` form where you can, since it shows where the segments end.

    EXAMPLE: `1.0|1.4.*` matches 1.0, 1.4, 1.4.1b2, and 1.4rc1, but not 1.2, 1.40, or 14.0.

  - * anywhere else in a version is a glob. Conda replaces each * with `.*` and matches the result against the whole version string as a regular expression.

    EXAMPLE: `1.*.3` matches 1.2.3 and 1.22.3, but not 1.3 or 1.2.30.

  - <, >, <=, >=, == and != are relational operators on versions, which are compared using PEP-440. For example, <=`1.0` matches `0.9`, `0.9.1`, and `1.0`, but not `1.0.1`. == and != are exact equality.

    Pre-release versioning is also supported such that >`1.0b4` will match `1.0b5` and `1.0rc1` but not `1.0b4` or `1.0a5`.

    EXAMPLE: <=1.0 matches 0.9, 0.9.1, and 1.0, but not 1.0.1.

  - , means AND.

    EXAMPLE: >=2,<3 matches all packages in the 2 series. 2.0, 2.1 and 2.9 all match, but 3.0 and 1.0 do not.

  - , has higher precedence than |, so >=1,<2|>3 means greater than or equal to 1 AND less than 2 or greater than 3, which matches 1, 1.3 and 3.0, but not 2.2.

  Conda parses the version by splitting it into parts separated by |. If the part begins with <, >, =, or !, it is parsed as a relational operator. Otherwise, it is parsed as a version, possibly containing the "*" operator.
- The third part is always the exact build string. When there are 3 parts, the second part must be the exact version.

Remember that the version specification cannot contain spaces, as spaces are used to delimit the package, version, and build string in the whole match specification. `python >= 2.7` is an invalid match specification. Furthermore, `python>=2.7` is matched as any version of a package named `python>=2.7`.

When using the command line, put double quotes around any package version specification that contains the space character or any of the following characters: <, >, *, or |.

EXAMPLE:

```
conda install numpy=1.11
conda install numpy==1.11
conda install "numpy>1.11"
conda install "numpy=1.11.1|1.11.3"
conda install "numpy>=1.8,<2"
```

Examples

The OR constraint "numpy=1.11.1|1.11.3" matches with 1.11.1 or 1.11.3.

The AND constraint "numpy>=1.8,<2" matches with 1.8 and 1.9 but not 2.0.

The fuzzy constraint numpy=1.11 matches 1.11, 1.11.0, 1.11.1, 1.11.2, 1.11.18, and so on.

The exact constraint numpy==1.11 matches 1.11, 1.11.0, 1.11.0.0, and so on.

The build string constraint "numpy=1.11.2=*nomkl*" matches the NumPy 1.11.2 packages without MKL but not the normal MKL NumPy 1.11.2 packages.

The build string constraint "numpy=1.11.1|1.11.3=py36_0" matches NumPy 1.11.1 or 1.11.3 built for Python 3.6 but not any versions of NumPy built for Python 3.5 or Python 2.7.

The following are all valid match specifications for numpy-1.8.1-py27_0:

- numpy
- numpy 1.8.*
- numpy 1.8.1
- numpy >=1.8
- numpy ==1.8.1
- numpy 1.8|1.8.*
- numpy >=1.8,<2
- numpy >=1.8,<2|1.9
- numpy 1.8.1 py27_0
- numpy=1.8.1=py27_0

Version ordering

The `class VersionOrder(object)` implements an order relation between version strings.

Version strings can contain the usual alphanumeric characters (A-Za-z0-9), separated into components by dots and underscores. Empty segments (i.e. two consecutive dots, a leading/trailing underscore) are not permitted. An optional epoch number - an integer followed by `!` - can precede the actual version string (this is useful to indicate a change in the versioning scheme itself). Version comparison is case-insensitive.

Supported version strings

Conda supports six types of version strings:

- Release versions contain only integers, e.g. `1.0`, `2.3.5`.
- Pre-release versions use additional letters such as `a` or `rc`, for example `1.0a1`, `1.2.beta3`, `2.3.5rc3`.
- Development versions are indicated by the string `dev`, for example `1.0dev42`, `2.3.5.dev12`.
- Post-release versions are indicated by the string `post`, for example `1.0post1`, `2.3.5.post2`.
- Tagged versions have a suffix that specifies a particular property of interest, e.g. `1.1.parallel`. Tags can be added to any of the preceding 4 types. As far as sorting is concerned, tags are treated like strings in pre-release versions.
- An optional local version string separated by + can be appended to the main (upstream) version string. It is only considered in comparisons when the main versions are equal, but otherwise handled in exactly the same manner.

Predictable version ordering

To obtain a predictable version ordering, it is crucial to keep the version number scheme of a given package consistent over time. Conda considers prerelease versions as less than release versions.

- Version strings should always have the same number of components (except for an optional tag suffix or local version string).

- Letters/Strings indicating non-release versions should always occur at the same position.

Before comparison, version strings are parsed as follows:

- They are first split into epoch, version number, and local version number at `!` and + respectively. If there is no `!`, the epoch is set to 0. If there is no +, the local version is empty.
- The version part is then split into components at `.` and `_`.
- Each component is split again into runs of numerals and non-numerals
- Subcomponents containing only numerals are converted to integers.
- Strings are converted to lowercase, with special treatment for `dev` and `post`.
- When a component starts with a letter, the fillvalue 0 is inserted to keep numbers and strings in phase, resulting in `1.1.a1' == 1.1.0a1'`.
- The same is repeated for the local version part.

Examples:

```
1.2g.beta15.rc => [[0], [1], [2, 'g'], [0, 'beta', 15], [0, 'rc']]
```

```
1!2.15.1_ALPHA => [[1], [2], [15], [1, '_alpha']]
```

The resulting lists are compared lexicographically, where the following rules are applied to each pair of corresponding subcomponents:

- Integers are compared numerically.
- Strings are compared lexicographically, case-insensitive.
- Strings are smaller than integers, except
  - `dev` versions are smaller than all corresponding versions of other types.
  - `post` versions are greater than all corresponding versions of other types.
- If a subcomponent has no correspondent, the missing correspondent is treated as integer 0 to ensure `'1.1' == 1.1.0'`.

The resulting order is:

```
  0.4
== 0.4.0
< 0.4.1.rc
== 0.4.1.RC   # case-insensitive comparison
< 0.4.1
< 0.5a1
< 0.5b3
< 0.5C1      # case-insensitive comparison
< 0.5
< 0.9.6
< 0.960923
< 1.0
< 1.1dev1    # special case ``dev``
< 1.1a1
< 1.1.0dev1  # special case ``dev``
== 1.1.dev1   # 0 is inserted before string
< 1.1.a1
< 1.1.0rc1
< 1.1.0
```

(continues on next page)

(continued from previous page)

```
== 1.1
< 1.1.0post1 # special case ``post``
== 1.1.post1  # 0 is inserted before string
< 1.1post1   # special case ``post``
< 1996.07.12
< 1!0.4.1    # epoch increased
< 1!3.1.1.6
< 2!0.4.1    # epoch increased again
```

Some packages (most notably OpenSSL) have incompatible version conventions. In particular, OpenSSL interprets letters as version counters rather than pre-release identifiers. For OpenSSL, the relation `1.0.1 < 1.0.1a => True # for OpenSSL` holds, whereas conda packages use the opposite ordering. You can work around this problem by appending a dash to plain version numbers:

```
1.0.1a => 1.0.1post.a # ensure correct ordering for OpenSSL
```

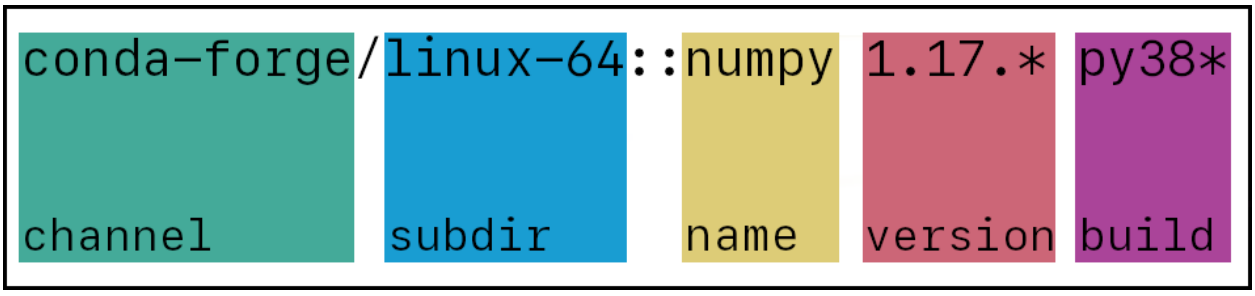
Package search and install specifications

Conda supports the following specifications for `conda search` and `conda install`.

Package search

`conda search` for a specific package or set of packages can be accomplished in several ways. This section includes information on the standard specification and the use of key-value pairs.

Standard specification



channel
: (Optional) Can either be a channel name or URL. Channel names may include letters, numbers, dashes, and underscores.

subdir
: (Optional) A subdirectory of a channel. Many subdirs are used for architectures, but this is not required. Must have a channel and backslash preceding it. For example: `main/noarch`

name
: (Required) Package name. May include the * wildcard. For example, `*py*` returns all packages that have "py" in their names, such as "numpy", "pytorch", "python", etc.

version
: (Optional) Package version. May include the * wildcard or a version range(s) in single quotes. For example: `numpy=1.17.*` returns all numpy packages with a version containing "1.17." and `numpy>1.17,<1.19.2` returns all numpy packages with versions greater than 1.17 and less than 1.19.2.

build
: (Optional) Package build name. May include the * wildcard. For example, `numpy 1.17.3 py38*` returns all version 1.17.3 numpy packages with a build name that contains the text "py38".

Key-value pairs

Package searches can also be performed using what is called "key-value pair notation", which has different rules than the *Standard specification* example image. The search below will return the same list of packages as the standard specification.

```
$ conda search "numpy[channel=conda-forge, subdir=linux-64, version=1.17.*, build=py38*]"
```

This notation supports the following key-value pairs:

```
- build                # validated via GlobStrMatch
- build_number         # validated via BuildNumberMatch
- channel              # validated via ChannelMatch
- extras               # validated via ListOfStrMatch
- features             # validated via FeatureMatch
- flags                # validated via ListOfStrMatch
- fn                   # validated via ExactStrMatch
- license              # validated via CaseInsensitiveStrMatch
- license_family       # validated via CaseInsensitiveStrMatch
- md5                  # validated via ExactStrMatch
- name                 # validated via GlobLowerStrMatch
- sha256               # validated via ExactStrMatch
- subdir               # validated via ExactStrMatch
- track_features       # validated via FeatureMatch
- url                  # validated via ExactStrMatch
- version              # validated via VersionSpec
- when                 # validated via CaseInsensitiveStrMatch
```

Key-value pair notation can be used at the same time as standard notation.

```
$ conda search "conda-forge::numpy=1.17.3[subdir=linux-64, build=py38*]"
```

Warning

Any search values using the key-value pair notation will override values in the rest of the search string. For example, `conda search numpy 1.17.3[version=1.19.2]` will return packages with the version `1.19.2`.

Package installation

When you're installing packages, conda recommends being as concrete as possible. Using * wildcards and version ranges during an install will most likely cause a conflict.

However, * wildcards can still be helpful in an install command when used sparingly.

Installing with wildcards

Let's say you are working on a project that requires version 2.3 of a package. If you upgrade to 2.4 or 3.0, your project will break. You're also using an environment file to create your environment.

In the version `2.3.1`, `2` is the major version, `3` is the minor version, and `1` is the patch. Patches typically contain bug fixes, so if you want to keep version `2.3` in your environment without updating to `2.4` or `3.0`, but want to take advantage of any bug fixes, using `2.3.*` in your environment file would be helpful to you.

Concrete install example

Let's take the search from the *Package search* section.

```
$ conda search "conda-forge/linux-64::numpy 1.17.* py38*"
```

This returns the following:

```
Loading channels: done
# Name                       Version           Build  Channel
numpy                         1.17.3  py38h95a1406_0  conda-forge
numpy                         1.17.5  py38h18fd61f_1  conda-forge
numpy                         1.17.5  py38h95a1406_0  conda-forge
```

You can then choose a specific version and build, if necessary, and edit your `conda install` command accordingly.

```
$ conda install "conda-forge/linux-64::numpy 1.17.5 py38h95a1406_0"
```

Solvers

When you install, update, or remove packages, conda must determine which package versions can work together in the same environment. The component that does this is called a **solver**. This page explains what solver plugins are, which solvers conda supports, and how to check and switch between them.

What are solver plugins?

Conda does not have a single, built-in solver. Instead, solvers are provided through a plugin system. Each solver is a separate package that registers itself with conda, which means you can:

- Use a different solver without changing conda itself.
- Install more than one solver side by side.
- Configure one solver globally and switch to another for a single command.

Because solvers are independent packages, each one has its own documentation, release schedule, and issue tracker. See the links in each section for solver-specific details.

Available solvers

The following table summarizes the solvers that work with conda:

Solver	Package	Status
rattler	`conda-rattler-solver`	Default since conda 26.10.0
libmamba	`conda-libmamba-solver`	Available alternative solver
pycosat (classic)	`conda-pycosat-solver`	Available alternative solver

> **Note**
>
> The default solver can change between conda releases. An explicit solver setting continues to take precedence when you upgrade conda. To see which solver your installation uses, follow the steps in *Check which solver is activated*.

rattler

The `rattler` solver plugin is provided by the `conda-rattler-solver` package. It uses the Resolvo dependency resolution engine through rattler, a collection of Rust libraries for working with conda packages.

The plugin solves environments faster than libmamba, with the largest improvements on large or heavily constrained environments. Starting with conda 26.10.0, `rattler` is the default solver.

conda-rattler-solver documentation

libmamba

The libmamba solver is provided by the `conda-libmamba-solver` package. It uses the solving engine from the mamba project and is available as an alternative solver in conda. This solver delegates to the C/C++ `libsolv` library, which gives it a low memory footprint, fast resolve times, and clear error messages that identify conflicting constraints directly.

conda-libmamba-solver documentation

pycosat

The pycosat solver is provided by the `conda-pycosat-solver` package. This solver has powered conda since conda's first release and is based on the `pycosat` Python bindings for the PicoSAT library.

> **Note**
>
> `classic` is a supported alias for `pycosat`. Both names select the same solver, so existing `solver: classic` settings and `--solver=classic` commands continue to work.

Choose the pycosat solver when backwards compatibility with existing long-lived environments is important and you need solve results that match the original conda behavior exactly.

conda-pycosat-solver documentation

Checking which solver is activated

To see which solver conda currently has configured, run:

```
conda config --show solver
```

The command prints the name of the configured solver, such as `rattler`. To see which configuration file the setting comes from, run:

```
conda config --show-sources
```

Changing your solver

To change the solver conda uses for all commands, set the `solver` configuration option:

```
conda config --set solver rattler
```

Replace `rattler` with `libmamba` or `pycosat` to use a different solver. Conda saves the setting in your `.condarc` file. For more information on your `.condarc` file, see *Using the .condarc conda configuration file*.

Note

The solver plugin must be installed in your `base` environment before you can select it. For example:

```
conda install --name base conda-rattler-solver
```

To return to the default solver, remove the custom solver setting:

```
conda config --remove-key solver
```

Use a solver for a single command

To try a different solver without changing your configuration, use the `--solver` flag with `conda install`, `conda create`, or `conda update`:

```
conda create --name test-env --solver rattler numpy
```

This approach is useful for comparing solvers or working around a problem with your default solver while you troubleshoot.

Report issues

Where to report a problem depends on where it occurs. Solver plugins are developed separately from conda, so report issues in the repository for the solver you are using:

- **rattler:** conda-rattler-solver issues
- **libmamba:** conda-libmamba-solver issues
- **pycosat:** conda-pycosat-solver issues

If you aren't sure whether a problem is caused by the solver or by conda itself, try the same command with a different solver using the `--solver` option (see *Use a solver for a single command*).

If the result changes, the problem is likely in the solver. Otherwise, report it in the conda issue tracker.

Channels

What is a "channel"?

Channels are locations where conda packages are stored. A channel can be a public package repository, a private or internal package repository, or a local directory on your computer.

Conda *packages* are downloaded from channels, which are URLs or channel names that point to directories containing conda packages. The `conda` command searches a configured set of channels when you install, update, or search for packages.

By default, packages are automatically downloaded and updated from the default channel, which may require a paid license, as described in the repository terms of service. You can modify which channels conda searches. For details, see how to *modify your channel lists*.

Common public channels

Some commonly used public channels and multichannels include:

defaults
: A built-in multichannel that includes Anaconda-hosted channels such as `main`, `r`, and `msys2` under `repo.anaconda.com`.

main
: The primary Anaconda-hosted channel under `repo.anaconda.com`. It contains many commonly used packages distributed by Anaconda.

conda-forge
: A community-led channel with packages maintained by thousands of contributors. The conda-forge project provides a large collection of packages across many ecosystems.

bioconda
: A community channel focused on bioinformatics packages. The Bioconda project is commonly used together with `conda-forge`.

You can browse public channels and search for packages on Anaconda.org.

Specifying channels when installing packages

From the command line, use `--channel` to search a specific channel:

```
$ conda install scipy --channel conda-forge
```

You may specify multiple channels by passing the argument multiple times:

```
$ conda install scipy --channel conda-forge --channel bioconda
```

Priority decreases from left to right: the first argument has higher priority than the second.

Use `--override-channels` to search only the specified channel or channels instead of any channels configured in `.condarc`. This also ignores conda's default channels.

```
$ conda search scipy --channel conda-forge --override-channels
```

In `.condarc`, use the `channels` key to configure the list of channels conda searches for packages.

Learn more about *managing channels*.

Local channels

A local channel is a channel stored on your own computer or on a shared filesystem. Local channels are useful when you are building your own packages, testing packages before publishing them, or maintaining packages for an internal workflow.

Local channels use the same channel layout as remote channels. They contain platform subdirectories such as `linux-64`, `osx-64`, `win-64`, and `noarch`. Each subdirectory contains package files and repository metadata.

To use a local channel, provide a `file://` URL or path to the channel root. For example:

```
$ conda search --channel file:///opt/conda-channel --override-channels
```

For a step-by-step example of creating and indexing a local channel, see *creating custom channels*.

Environments

An environment is a directory that contains a specific collection of packages that you have installed. For example, you may have one environment with NumPy 1.7 and its dependencies, and another environment with NumPy 1.6 for legacy testing. If you change one environment, your other environments are not affected. You can easily activate or deactivate environments, which is how you switch between them. You can also share your environment with someone by giving them a copy of your `environment.yaml` file. For more information, see *Managing environments*.

Conda directory structure

ROOT_DIR

The directory where the conda distribution was installed into.

EXAMPLES:

```
/opt/Anaconda  #Linux
C:\Anaconda    #Windows
```

/pkgs

Also referred to as PKGS_DIR. This directory contains decompressed packages, ready to be linked in conda environments. Each package resides in a subdirectory corresponding to its canonical name.

/envs

The system location for additional conda environments to be created.

The following subdirectories comprise the default Anaconda environment:

```
/bin
/include
/lib
/share
```

Other conda environments usually contain the same subdirectories as the default environment.

Virtual environments

A virtual environment is a tool that helps to keep dependencies required by different projects separate by creating isolated spaces for them that contain per-project dependencies for them.

Users can create virtual environments using one of several tools such as Pipenv or Poetry, or a conda virtual environment. Pipenv and Poetry are based around Python's built-in venv library, whereas conda has its own notion of virtual environments that is lower-level (Python itself is a dependency provided in conda environments).

Scroll to the right in the table below.

Some other traits are:

	Python virtual environment	Conda virtual environment
Libraries	Statically link, vendor libraries in wheels, or use apt/yum/brew/etc.	Install system-level libraries as conda dependencies.
System	Depend on base system install of Python.	Python is independent from system.
Extending environment	Extend environment with pip.	Extended environment with conda or pip.
Non-Python dependencies		Manages non-Python dependencies (R, Perl, arbitrary executables).
Tracking dependencies		Tracks binary dependencies explicitly.

Why use venv-based virtual environments

- You prefer their workflow or spec formats.
- You prefer to use the system Python and libraries.
- Your project maintainers only publish to PyPI, and you prefer packages that come more directly from the project maintainers, rather than someone else providing builds based on the same code.

Why use conda virtual environments?

- You want control over binary compatibility choices.
- You want to utilize newer language standards, such as C++ 17.
- You need libraries beyond what the system Python offers.
- You want to manage packages from languages other than Python in the same space.

Workflow differentiators

Some questions to consider as you determine your preferred workflow and virtual environment:

- Is your environment shared across multiple code projects?
- Does your environment live alongside your code or in a separate place?
- Do your install steps involve installing any external libraries?
- Do you want to ship your environment as an archive of some sort containing the actual files of the environment?

Package system differentiators

There are potential benefits for choosing PyPI or conda.

PyPI has one global namespace and distributed ownership of that namespace. Because of this, it is easier within PyPI to have single sources for a package directly from package maintainers.

Conda has unlimited namespaces (channels) and distributed ownership of a given channel. As such, it is easier to ensure binary compatibility within a channel using conda.

Installing with conda

Conda packages can be installed by running the following command:

```
conda install <package>
```

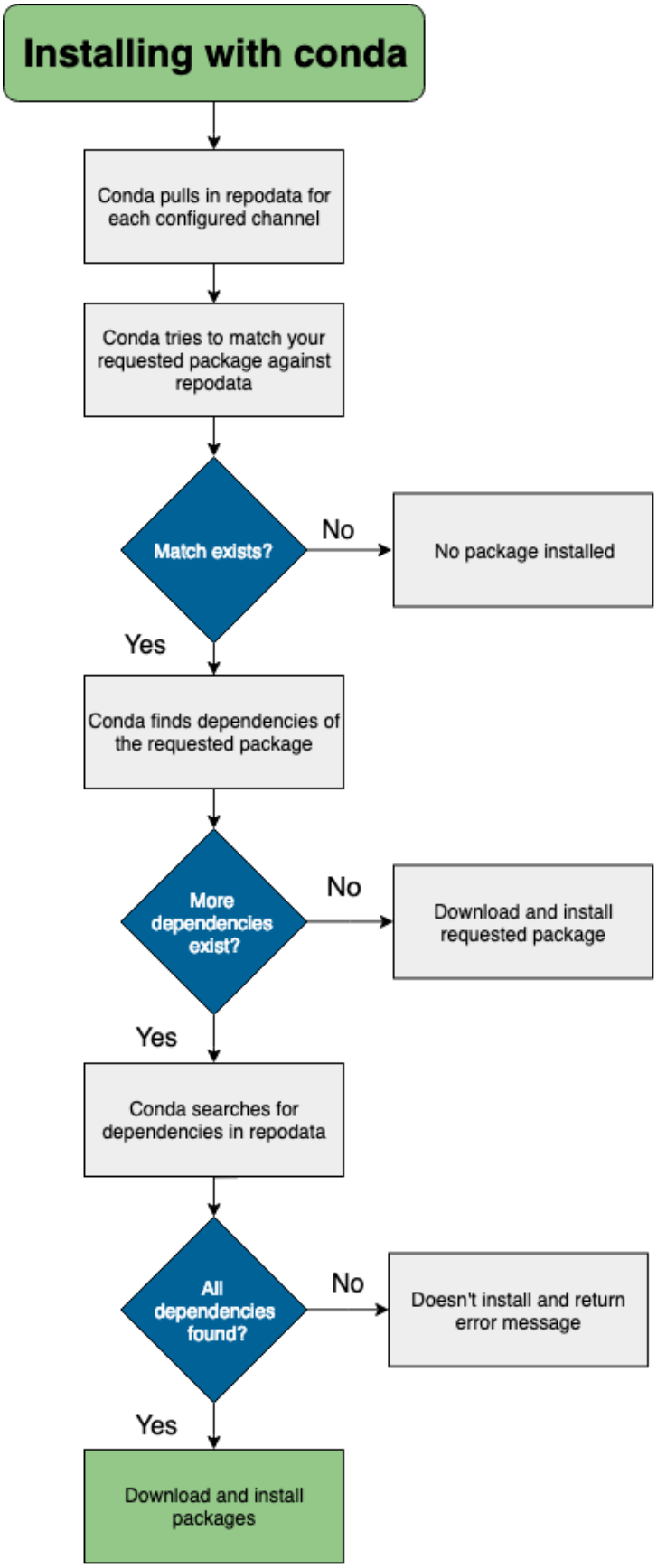
When conda installs a package, it is automatically added to your active environment. These packages are collections of files and directories that make up everything you need to use that particular library or software. For Python packages, these are primarily Python files that can be imported into other Python applications, but for compiled software packages, such as `ffmpeg`, these are typically binary executables you use directly on your computer.

> **Note**
>
> If you would like to learn more about how environments are structured, head over to *conda environments*.

Below is a more precise overview of everything that happens during the installation process for a single package:

- Currently configured channels (e.g. `defaults` or `conda-forge`) are read in order of priority
- Repodata for these configured channels is downloaded and read
- The repodata is searched for the package, starting with the highest priority channel first
- Once the package is found, conda makes a separate download request and then installs it
- This process then repeats for each of the package's dependencies, if there are any

A graphic illustration of this process is shown below:


Installing with conda
Conda pulls in repodata for each configured channel
Conda tries to match your requested package against repodata
Match exists?
No
No package installed
Yes
Conda finds dependencies of the requested package
More dependencies exist?
No
Download and install requested package
Yes
Conda searches for dependencies in repodata
All dependencies found?
No
Doesn't install and return error message
Yes
Download and install packages

Conda update versus conda install

`conda update` updates packages to the latest compatible version. `conda install` can be used to install any version.

Example:

- If Python 2.7.0 is currently installed, and the latest version of Python 2 is 2.7.5, then `conda update python` installs Python 2.7.5. It does not install Python 3.
- If Python 3.7.0 is currently installed, and the latest version of Python is 3.9.0, then `conda install python=3` installs Python 3.9.0.

Conda uses the same rules for other packages. `conda update` always installs the highest version with the same major version number, whereas `conda install` always installs the highest version.

Installing conda packages offline

To install conda packages offline, run: `conda install /path-to-package/package-filename.tar.bz2/`

If you prefer, you can create a /tar/ archive file containing many conda packages and install them all with one command: `conda install /packages-path/packages-filename.tar`

> **Note**
>
> If an installed package does not work, it may be missing dependencies that need to be resolved manually.

Installing packages directly from the file does not resolve dependencies.

Installing conda packages with a specific build number

If you want to install conda packages with the correct package specification, try `pkg_name=version=build_string`. Read more about build strings and package naming conventions. Learn more about package specifications and metadata.

For example, if you want to install llvmlite 0.31.0dev0 on Python 3.7.8, you would enter:

```
conda install  -c numba/label/dev llvmlite=0.31.0dev0=py37_8
```

Performance

Conda's performance can be affected by a variety of things. Unlike many package managers, Anaconda's repositories generally don't filter or remove old packages from the index. This allows old environments to be easily recreated. However, it does mean that the index metadata is always growing, and thus conda becomes slower as the number of packages increases.

How a package is installed

While you are waiting, conda is doing a lot of work installing the packages. At any point along these steps, performance issues may arise.

Conda follows these steps when installing a package:

1. Downloading and processing index metadata.
2. Reducing the index.
3. Expressing the package data and constraints as a SAT problem.
4. Running the solver.

5. Downloading and extracting packages.
6. Verifying package contents.
7. Linking packages from package cache into environments.

Therefore, if you're experiencing a slowdown, evaluate the following questions to identify potential causes:

- Are you creating a new environment or installing into an existing one?
- Does your environment have pip-installed dependencies in it?
- What channels are you using?
- What packages are you installing?
- Is the channel metadata sane?
- Are channels interacting in bad ways?

Improving conda performance

This section goes over some of the best practices we recommend for addressing performance challenges.

1. **Make your package specifications more narrow.**

   For example, instead of `numpy`, we recommend `numpy=1.15` or, even better, `numpy=1.15.4`.

2. **Use the rattler solver plugin.** The plugin is the default solver starting with conda 26.10.0 and improves performance for large environments. If you previously selected another solver, that setting is preserved when you upgrade conda.

   > **Note**
   >
   > The classic solver, now named *pycosat* (*classic* remains a supported alias), is provided by the standalone conda-pycosat-solver plugin rather than being registered by conda itself.

   To check which solver you have, run the following command:

   ```
   conda config --show solver
   ```

   If rattler is set as your solver, you will see the following:

   ```
   solver: rattler
   ```

   If you don't have rattler set as your solver, follow these steps to enable it:

   1. Make sure `conda-rattler-solver` is installed in your base environment:

      ```
      conda install --name base conda-rattler-solver
      ```

   2. Set rattler as your solver:

      ```
      conda config --set solver rattler
      ```

   > **Tip**
   >
   > You can also use the rattler solver temporarily when installing a package:

```
conda install --solver=rattler package_name
```

For more information on conda's solver options, see *Solvers*.

3. **Configure channels to minimize conflicts.**

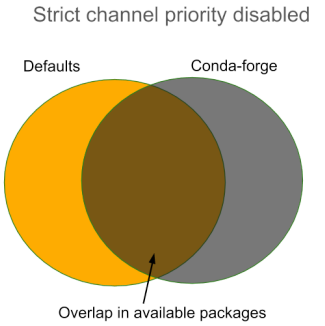
   Channel configuration has a significant impact on solver performance and environment reproducibility. We recommend using a single channel as your default, such as `conda-forge` or `defaults`, but not both. Mixing channels in your global configuration increases solver search space and the risk of incompatible packages.

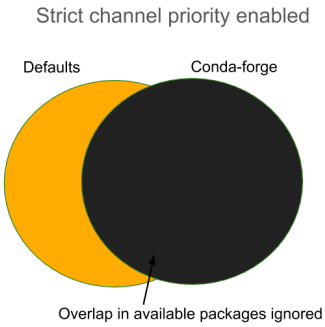
   For more information on channel configuration, see *Channel configuration best practices*.

   If you must use both `defaults` and `conda-forge` in your channel list, setting strict channel priority reduces solver search space and helps prevent mixed-channel environments:

```
conda config --set channel_priority strict
```

> **Warning**
>
> Strict channel priority can make some environments unsatisfiable and blocks fallback to lower-priority channels when a package with the same name exists in a higher-priority channel. To learn more about strict channel priority and channel configuration, see *channel configuration best practices*.





4. **Use sharded repodata.** This splits your repodata into multiple small files and fetches only what is needed, which dramatically speeds up environment creation and updates. Learn more in CEP 16.

> **Note**
>
> This option is currently available for conda-forge and prefix.dev for all channels as of March 2026.

With the rattler and libmamba solvers, sharded repodata is enabled by default through conda's `repodata_use_shards` setting. Channels that do not publish sharded repodata are fetched normally.

To disable sharded repodata, run:

```
conda config --set repodata_use_shards false
```

Conda for data scientists

Conda is useful for any packaging process but it stands out from other package and environment management systems through its utility for data science.

Conda's benefits include:

- Providing prebuilt packages which avoid the need to deal with compilers or figuring out how to set up a specific tool.
- Managing one-step installation of tools that are more challenging to install (such as TensorFlow or IRAF).
- Allowing you to provide your environment to other people across different platforms, which supports the reproducibility of research workflows.
- Allowing the use of other package management tools, such as pip, inside conda environments where a library or tools are not already packaged for conda.
- Providing commonly used data science libraries and tools, such as R, NumPy, SciPy, and TensorFlow. These are built using optimized, hardware-specific libraries (such as Intel's MKL or NVIDIA's CUDA) which speed up performance without code changes.

Read more about how conda supports data scientists.

Plugins

In order to enable customization and extra features that are compatible with and discoverable by conda (but do not necessarily ship as a default part of the conda codebase), an official conda plugin mechanism has been implemented as of version `22.11.0`.

Managing plugin packages

Added in version 26.11.0: The `conda plugins install`, `conda plugins remove`, and `conda plugins update` commands.

These commands manage conda packages that provide plugins. Use `conda plugins list` and `conda plugins info` to inspect plugins loaded by the running conda installation. See *the command reference* for all subcommands and options.

Installing plugins

Use `conda plugins install` to install a conda plugin package into an environment. Select the environment containing the conda installation you want to extend with `--name` or `--prefix`. Choose plugin packages from channels compatible with the packages already installed there.

> **Warning**
>
> Do not install conda-forge plugins into an Anaconda Distribution or Miniconda `base` environment using `defaults`. Adding `--channel conda-forge` or `--override-channels` does not make mixing those packages safe. Use a separate conda-forge environment instead. See *Channel configuration best practices*.

Choose the example matching the package source of your conda installation:

Anaconda Distribution / Miniconda

Keep `base` on Anaconda's `defaults` channels. To try the conda-forge plugin `conda-tree`, create a separate environment containing conda:

```
conda create --name conda-forge-plugins --override-channels --channel conda-forge conda
conda plugins install --name conda-forge-plugins --override-channels --channel conda-
↪forge conda-tree
```

These commands leave the packages in `base` unchanged. Run the separate environment's conda to use the plugin:

```
conda run --name conda-forge-plugins python -m conda tree --help
```

Miniforge

Miniforge uses conda-forge packages in `base`. Install `conda-tree` there, explicitly restricting package sources to conda-forge:

```
conda plugins install --name base --override-channels --channel conda-forge conda-tree
```

This example assumes `base` still contains only compatible conda-forge packages. Do not use it for an Anaconda `defaults` environment.

`--channel` (`-c`) selects a package source for this command. Repeat it to search multiple compatible sources in the specified order. `--override-channels` excludes the configured channels from the search. It does not isolate packages already installed in the target environment or migrate them to another channel. See *Channel configuration best practices*.

These options do not create or modify channel configuration and can be used before a `.condarc` exists. Use `--override-channels --channel conda-forge` again for subsequent installs and updates in `conda-forge-plugins`. Without `--override-channels`, conda also searches the configured channels.

Channel aliases, authentication, network settings, and channel restrictions are handled by conda in the same way as `conda install`. Plugin packages are checked for a conda entry point before the installation transaction is confirmed. This check identifies plugin packages; it does not verify that their code is safe. Only install plugins from sources you trust.

Conda loads plugins from the environment containing the Python interpreter that runs it. `conda plugins list` and `conda plugins info` report plugins from that same environment. Activating another environment does not change the conda installation used by the shell's `conda` command.

To load a plugin installed in another environment, run that environment's Python. The environment must also contain conda, as in the example above:

```
conda run --name conda-forge-plugins python -m conda tree --help
conda run --name conda-forge-plugins python -m conda plugins list
conda run --name conda-forge-plugins python -m conda plugins info conda-tree
```

Inspecting plugins

List the plugins available to the running conda installation:

```
conda plugins list
```

The table shows each plugin's distribution name, version, status, and registered hooks. Hooks identify the capabilities the plugin provides, such as subcommands or solvers. To inspect one plugin, replace `NAME` with a distribution name from the table:

```
conda plugins info NAME
```

`info` shows the plugin's canonical name, summary, license, homepage, and project URLs in addition to its version, status, and hooks. It also accepts the canonical plugin name. Metadata fields may be empty when the plugin does not provide them.

Both commands support JSON output for scripts:

```
conda plugins list --json
conda plugins info NAME --json
```

Unlike install, update, and remove, these commands do not accept `--name` or `--prefix` to select another environment. Use that environment's Python as shown above. `list` and `info` were available before conda 26.11.0.

Updating plugins

Update installed plugins using the same target environment and package sources used to install them. Do not switch an Anaconda `base` environment to conda-forge to update a plugin.

Anaconda Distribution / Miniconda

Update plugin packages in your `defaults`-based `base` environment using only `defaults`:

```
conda plugins update --name base --override-channels --channel defaults --all
```

For `conda-tree`, use the separate conda-forge environment created in the installation example. Do not target `base`:

```
conda plugins update --name conda-forge-plugins --override-channels --channel conda-
↪forge conda-tree
```

To update all plugins in that separate environment:

```
conda plugins update --name conda-forge-plugins --override-channels --channel conda-
↪forge --all
```

Miniforge

Update `conda-tree` in your conda-forge-based `base` environment:

```
conda plugins update --name base --override-channels --channel conda-forge conda-tree
```

To update all plugin packages in that environment:

```
conda plugins update --name base --override-channels --channel conda-forge --all
```

`--all` selects plugin packages rather than every package in the environment. Plugin updates disable automatic conda upgrades, but the solver may still change conda or other packages to satisfy plugin dependencies. Review the transaction plan before confirming. `--all` cannot be combined with package names or `--file`.

Use `--dry-run` with install or update to preview the package plan without installing packages. Dry runs do not download new packages to check their entry points, so a successful dry run does not guarantee that those packages provide conda plugins.

Removing plugins

Remove a plugin by its conda package name from the same target environment:

Anaconda Distribution / Miniconda

Remove `conda-tree` from the separate conda-forge environment:

```
conda plugins remove --name conda-forge-plugins --override-channels --channel conda-
↪forge conda-tree
```

Miniforge

Remove `conda-tree` from your conda-forge-based `base` environment:

```
conda plugins remove --name base --override-channels --channel conda-forge conda-tree
```

Removal can also remove packages that depend on the plugin. Review the package plan before confirming, particularly when removing a solver or another plugin used by your configuration.

Both remove and named update require the requested packages to be installed as conda plugin packages in the target environment. They do not manage plugins installed with pip. Package names can differ from the plugin names shown by `conda plugins list`; use `conda list --name base` to check package names.

Implementation

Plugins in conda integrate the "hook + entry point" structure by utilizing the Pluggy Python framework. This implementation can be broken down via the following two steps:

- Define the hook(s) to be registered
- Register the plugin under the conda entrypoint namespace

Hook

Below is an example of a very basic plugin "hook":

Listing 1: my_plugin.py

```
import conda.plugins


@conda.plugins.hookimpl
def conda_subcommands(): ...
```

Packaging using a pyproject.toml file

Below is an example that configures `setuptools` using a `pyproject.toml` file (note that the `setup.py` file is optional if a `pyproject.toml` file is defined, and thus will not be discussed here):

Listing 2: pyproject.toml

```toml
[build-system]
requires = ["setuptools", "setuptools-scm"]
build-backend = "setuptools.build_meta"

[project]
name = "my-conda-plugin"
version = "1.0.0"
description = "My conda plugin"
requires-python = ">=3.11"
dependencies = ["conda"]

[project.entry-points."conda"]
my-conda-plugin = "my_plugin"
```

Conda plugins use cases

The new conda plugin API ecosystem brings about many possibilities, including but not limited to:

- *Custom subcommands*
- Support for packaging-related topics (*e.g.*, *virtual packages*)
- Development environment integrations (*e.g.*, shells)
- Alternative dependency solver backends
- Network adapters
- Build system integrations
- Non-Python language support (*e.g.*, C, Rust)
- *Environment export formats*
- Experimental features that are not currently covered by conda

Benefits of conda plugins

A conda plugin ecosystem enables contributors across the conda community to develop and share new features, thus bringing about more functionality and focus on the user experience. Though the list below is by no means exhaustive, some of the benefits of conda plugins include:

- Support for a better distribution of maintenance in the conda community
- Enabling third party contributors to use official APIs instead of having to divert to workarounds and wrappers
- The ability to extend conda internals via official APIs
- Lowering the barrier for contributions from other stakeholders in the conda ecosystem
- ... and much more!

In this section, we provide you with detailed information about the fundamental concepts in conda, including information about packages, channels, environments, and plugins, among others.

Commands
: Conda commands are your interface for interacting with everything

Packages
: Learn about the different forms a package can take

Package specification
: Learn about exactly what belongs in a package and what the different metadata files mean

Package search and install
: The search specifications for a package (for the `conda install` and `conda search` commands)

Solvers
: Learn about the different solvers available in conda and how to use them

Channels
: Learn about channels and how they host packages

Environments
: Learn about how environments work and how they differ from Python's virtual environments

Installing with conda
: Take a deep dive into exactly what happens during an installation with conda

Performance
: Understand what impacts the performance and speed of conda and how to improve it

Conda for data scientists
: See why conda is a such a valuable tool for data scientists

Plugins
: The behavior of conda can be extended via plugins; learn more here

4.1.6 Troubleshooting

Recovering from plugin errors

To disable all external plugins for one invocation, use `--no-plugins` or set `CONDA_NO_PLUGINS=true`. Built-in plugins remain enabled:

```
conda --no-plugins info
```

Commands that solve an environment still need a solver. Keep an installed solver plugin enabled with `--enable-plugins`, and select its solver explicitly:

```
conda --no-plugins --enable-plugins conda-libmamba-solver install --solver=libmamba numpy
```

If `conda-pycosat-solver` is installed, it can be selected instead:

```
conda --no-plugins --enable-plugins conda-pycosat-solver install --solver=pycosat numpy
```

The same override works when `CONDA_NO_PLUGINS=true` is set. To disable only selected plugins, use `--disable-plugins NAME[,NAME...]`:

```
conda --disable-plugins plugin-a --disable-plugins=plugin-b info
```

`--disable-plugins` and `--enable-plugins` accept distribution names, entry-point names, or canonical registered names. Both accept whitespace or = before the value, comma-separated lists, and repeated options. `--no-plugins` remains a boolean flag and does not take plugin names.

Explicit `--enable-plugins` selections override disabling regardless of option order. Without disabling, `--enable-plugins` leaves other plugins enabled. An unknown or unsuccessfully loaded enable target is an error.

Place plugin selection options before the conda subcommand. Arguments passed to an executable through `conda run` are not plugin selection options for conda.

These options disable plugin hooks after entry points are imported. They do not prevent import-time errors or side effects, and enabling a plugin enables all of its hooks, not just its solver.

Using conda in Windows Batch script exits early

In conda 4.6+, the way that you interact with conda goes through a batch script (`%PREFIX%\condabin\conda.bat`). Unfortunately, this means it's a little complicated to use conda from other batch scripts. When using batch scripts from within batch scripts, you must prefix your command with `CALL`. If you do not do this, your batch script that calls conda will exit immediately after the conda usage. In other words, if you write this in a .bat file:

```
conda create myenv python
conda activate myenv
echo test
```

Neither the activation, nor the echo will happen. You must write this in your batch script:

```
CALL conda create myenv python
CALL conda activate myenv
echo test
```

This is known behavior with cmd.exe, and we have not found any way to change it. https://stackoverflow.com/questions/4798879/how-do-i-run-a-batch-script-from-within-a-batch-script/4798965

NumPy MKL library load failed

Error messages like

```
Intel MKL FATAL ERROR: Cannot load mkl_intel_thread.dll
```

or

```
The ordinal 241 could not be located in the the dynamic link library
```

Cause

NumPy is unable to load the correct MKL or Intel OpenMP runtime libraries. This is almost always caused by one of two things:

1. The environment with NumPy has not been activated.
2. Another software vendor has installed MKL or Intel OpenMP (libiomp5md.dll) files into the C:\Windows\System32 folder. These files are being loaded before Anaconda's and they're not compatible.

Solution

If you are not activating your environments, start with doing that. There's more info at *Activating environments*. If you are still stuck, you may need to consider more drastic measures.

1. Remove any MKL-related files from C:\Windows\System32. We recommend renaming them to add .bak to the filename to effectively hide them. Observe if any other software breaks. Try moving the

DLL files alongside the .exe of the software that broke. If it works again, you can keep things in the moved state - Anaconda doesn't need MKL in System32, and no other software should need it either. If you identify software that is installing software here, please contact the creators of that software. Inform them that their practice of installing MKL to a global location is fragile and is breaking other people's software and wasting a lot of time. See the list of guilty parties below.

2. You may try a special DLL loading mode that Anaconda builds into Python. This changes the DLL search path from System32 first to System32 as another entry on PATH, allowing libraries in your conda environment to be found before the libraries in System32. Control of this feature is done with environment variables. Only Python builds beyond these builds will react to these environment variables:

   - Python 2.7.15 build 14
   - Python 3.6.8 build 7
   - Python 3.7.2 build 8

   To update Python from the defaults channel:

```
conda update -c defaults python
```

> **Note**
>
> Anaconda has built special patches into its builds of Python to enable this functionality. If you get your Python package from somewhere else (e.g. conda-forge), these flags may not do anything.

   Control environment variables:

   - `CONDA_DLL_SEARCH_MODIFICATION_ENABLE`
   - `CONDA_DLL_SEARCH_MODIFICATION_DEBUG`
   - `CONDA_DLL_SEARCH_MODIFICATION_NEVER_ADD_WINDOWS_DIRECTORY`
   - `CONDA_DLL_SEARCH_MODIFICATION_NEVER_ADD_CWD`

   To set variables on Windows, you may use either the CLI or a Windows GUI.

   - CLI: https://superuser.com/questions/79612/setting-and-getting-windows-environment-variables-from-the-command-p 79614
   - GUI: http://www.dowdandassociates.com/blog/content/howto-set-an-environment-variable-in-windows-gui/

   These should be set to a value of `1` to enable them. For example, in a terminal:

```
set CONDA_DLL_SEARCH_MODIFICATION_ENABLE=1
```

> **Note**
>
> Only `CONDA_DLL_SEARCH_MODIFICATION_ENABLE` should be set finally.

List of known software that installs Intel libraries to C:\Windows\System32:

- Amplitube, by IK Multimedia
- ASIO4ALL, by Michael Tippach

If you find others, please let us know. If you're on this list and you want to fix things, let us know. In either case, the conda issue tracker at https://github.com/conda/conda/issues is the best way to reach us.

SSL connection errors

This is a broad umbrella of errors with many causes. Here are some we've seen.

CondaHTTPError: HTTP 000 CONNECTION FAILED

If you're on Windows and you see this error, look a little further down in the error text. Do you see something like this?:

```
SSLError(MaxRetryError('HTTPSConnectionPool(host=\'repo.anaconda.com\', port=443): Max␣
↪retries exceeded with url: /pkgs/r/win-32/repodata.json.bz2 (Caused by SSLError("Can\
↪'t connect to HTTPS URL because the SSL module is not available."))'))
```

The key part there is the last bit:

```
Caused by SSLError("Can\'t connect to HTTPS URL because the SSL module is not available.
↪")
```

Conda is having problems because it can't find the OpenSSL libraries that it needs.

Cause

You may observe this error cropping up after a conda update. More recent versions of conda and more recent builds of Python are more strict about requiring activation of environments. We're working on better error messages for them, but here's the story for now. Windows relies on the PATH environment variable as the way to locate libraries that are not in the immediate folder, and also not in the C:\Windows\System32 folder. Searching for libraries in the PATH folders goes from left to right. If you choose to put Anaconda's folders on PATH, there are several of them:

- (install root)
- (install root)/Library/(MSYS2 env)/bin ## dependent on MSYS2 packages
- (install root)/Library/mingw-w64/bin
- (install root)/Library/usr/bin
- (install root)/Library/bin
- (install root)/Scripts
- (install root)/bin
- (install root)/condabin

Early installers for Anaconda put these on PATH. That was ultimately fragile because Anaconda isn't the only software on the system. If other software had similarly named executables or libraries, and came earlier on PATH, Anaconda could break. On the flip side, Anaconda could break other software if Anaconda were earlier in the PATH order and shadowed any other executables or libraries. To make this easier, we began recommending "activation" instead of modifying PATH. Activation is a tool where conda sets your PATH, and also runs any custom package scripts which are often used to set additional environment variables that are necessary for software to run (e.g. `JAVA_HOME`). Because activation runs only in a local terminal session (as opposed to the permanent PATH entry), it is safe to put Anaconda's PATH entries first. That means that Anaconda's libraries get higher priority when you're running Anaconda but Anaconda doesn't interfere with other software when you're not running Anaconda.

Anaconda's Python interpreter included a patch for a long time that added the (install root)/Library/bin folder to that Python's PATH. Unfortunately, this interfered with reasoning about PATH at all when using that Python interpreter. We removed that patch in Python 3.7.0, and we regret that this has caused problems for people who are not activating

their environments and who otherwise do not have the proper entries on PATH. We're experimenting with approaches that will allow our executables to be less dependent on PATH and more self-aware of their needed library load paths. For now, though, the only solutions to this problem are to manage PATH properly.

Our humble opinion is that activation is the easiest way to ensure that things work. See more information on activation in *Activating environments*.

Solution

Use shells opened from Anaconda Navigator. If you use a GUI IDE and you see this error, ask the developers of your IDE to add activation for conda environments.

SSL certificate errors

Cause

Installing packages may produce a "connection failed" error if you do not have the certificates for a secure connection to the package repository.

Solution

Pip can use the `--use-feature=truststore` option to use the operating system certificate store. This may be of help in typically corporate environments with https traffic inspection, where the corporate CA is installed in the operating system certificate store:

```
pip install --use-feature=truststore
```

Conda has a similar option:

```
conda config --set ssl_verify truststore
```

Alternatively, pip can use the `--trusted-host` option to indicate that the URL of the repository is trusted:

```
pip install --trusted-host pypi.org
```

Conda has three similar options.

1. The option `--insecure` or `-k` ignores certificate validation errors for all hosts.

   Running `conda create --help` shows:

   ```
   Networking Options:
     -k, --insecure        Allow conda to perform "insecure" SSL connections and
                           transfers. Equivalent to setting 'ssl_verify' to
                           'False'.
   ```

2. The configuration option `ssl_verify` can be set to `False`.

   Running `conda config --describe ssl_verify` shows:

   ```
   # # ssl_verify (bool, str)
   # #   aliases: verify_ssl
   # #   conda verifies SSL certificates for HTTPS requests, just like a web
   # #   browser. By default, SSL verification is enabled and conda operations
   # #   will fail if a required URL's certificate cannot be verified. Setting
   # #   ssl_verify to False disables certification verification. The value for
   ```

(continues on next page)

(continued from previous page)

```
# #   ssl_verify can also be (1) a path to a CA bundle file, (2) a path to a
# #   directory containing certificates of trusted CA, or (3) 'truststore'
# #   to use the operating system certificate store.
# #
# ssl_verify: true
```

Running `conda config --set ssl_verify false` modifies `~/.condarc` and sets the `-k` flag for all future conda operations performed by that user. Running `conda config --help` shows other configuration scope options.

When using `conda config`, the user's conda configuration file at `~/.condarc` is used by default. The flag `--system` will instead write to the system configuration file for all users at `<CONDA_BASE_ENV>/.condarc`. The flag `--env` will instead write to the active conda environment's configuration file at `<PATH_TO_ACTIVE_CONDA_ENV>/.condarc`. If `--env` is used and no environment is active, the user configuration file is used.

3. The configuration option `ssl_verify` can be used to install new certificates.

   Running `conda config --describe ssl_verify` shows:

```
# # ssl_verify (bool, str)
# #   aliases: verify_ssl
# #   conda verifies SSL certificates for HTTPS requests, just like a web
# #   browser. By default, SSL verification is enabled, and conda operations
# #   will fail if a required URL's certificate cannot be verified. Setting
# #   ssl_verify to False disables certification verification. The value for
# #   ssl_verify can also be (1) a path to a CA bundle file, (2) a path to a
# #   directory containing certificates of trusted CA, or (3) 'truststore'
# #   to use the operating system certificate store.
# #
# ssl_verify: true
```

   Your network administrator can give you a certificate bundle for your network's firewall. Then `ssl_verify` can be set to the path of that certificate authority (CA) bundle and package installation operations will complete without connection errors.

   When using `conda config`, the user's conda configuration file at `~/.condarc` is used by default. The flag `--system` will instead write to the system configuration file for all users at `<CONDA_BASE_ENV>/.condarc`. The flag `--env` will instead write to the active conda environment's configuration file at `<PATH_TO_ACTIVE_CONDA_ENV>/.condarc`. If `--env` is used and no environment is active, the user configuration file is used.

SSL verification errors

Cause

This error may be caused by lack of activation on Windows or expired certifications:

```
SSL verification error: [SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed (_ssl.
↪c:590)
```

Solution

Make sure your conda is up-to-date: `conda --version`

If not, run: `conda update conda`

Try using the operating system certificate store. Set you `ssl_verify` variable to `truststore` using the following command:

```
conda config --set ssl_verify truststore
```

If using the operating system certificate store does not solve your issue, temporarily set your `ssl_verify` variable to false, upgrade the requests package, and then set `ssl_verify` back to `true` using the following commands:

```
conda config --set ssl_verify false
conda update requests
conda config --set ssl_verify true
```

You can also set `ssl_verify` to a string path to a certificate, which can be used to verify SSL connections. Modify your `.condarc` and include the following:

```
ssl_verify: path-to-cert/chain/filename.ext
```

If the repository uses a self-signed certificate, use the actual path to the certificate. If the repository is signed by a private certificate authority (CA), the file needs to include the root certificate and any intermediate certificates.

Temporary file errors during conda operations

Conda creates temporary files during various operations including environment activation, package installation, and running commands. These errors typically occur when the temporary directory is read-only, has insufficient space, or has permission issues.

You may see errors like:

```
PermissionError: [Errno 13] Permission denied: '/tmp/...'
OSError: [Errno 30] Read-only file system: '/tmp/...'
OSError: [Errno 28] No space left on device: '/tmp/...'
```

Cause

- The default temporary directory is read-only
- Insufficient disk space in the temporary directory
- Restrictive permissions on the temporary directory
- Running in containers or HPC environments with limited or read-only temporary directories

Solution

Configure conda to use a different temporary directory by setting the `TMPDIR`, `TEMP`, or `TMP` environment variables.

For detailed instructions, platform-specific examples, and information on how Python's `tempfile` module determines temporary directories, see *Configuring Temporary File Locations*.

Permission denied errors during installation

Cause

The `umask` command determines the mask settings that control how file permissions are set for newly created files. If you have a very restrictive `umask`, such as `077`, you get "permission denied" errors.

Solution

Set a less restrictive `umask` before calling conda commands. Conda was intended as a user space tool, but often users need to use it in a global environment. One place this can go awry is with restrictive file permissions. Conda creates links when you install files that have to be read by others on the system.

To give yourself full permissions for files and directories but prevent the group and other users from having access:

1. Before installing, set the `umask` to `007`.
2. Install conda.
3. Return the `umask` to the original setting:

```
umask 007
conda install
umask 077
```

For more information on `umask`, see http://en.wikipedia.org/wiki/Umask.

Permission denied errors after using sudo conda command

Solution

Once you run conda with sudo, you must use sudo forever. We recommend that you NEVER run conda with sudo.

Already installed error message

Cause

If you are trying to fix conda problems without removing the current installation and you try to reinstall Miniconda or Anaconda to fix it, you get an error message that Miniconda or Anaconda is already installed and you cannot continue.

Solution

Install using the `--force` option.

Download and install the appropriate Miniconda for your operating system from the Miniconda download page using the force option `--force` or `-f`:

```
bash Miniconda3-latest-MacOSX-x86_64.sh -f
```

> **Note**
>
> Substitute the appropriate filename and version for your operating system.

> **Note**
>
> Be sure that you install to the same location as your existing install so it overwrites the core conda files and does not install a duplicate in a new folder.

Conda reports that a package is installed, but it appears not to be

Sometimes conda claims that a package is already installed but it does not appear to be, for example, a Python package that gives ImportError.

There are several possible causes for this problem, each with its own solution.

Cause

You are not in the same conda environment as your package.

Solution

1. Make sure that you are in the same conda environment as your package. The `conda info` command tells you what environment is currently active under `default environment`.
2. Verify that you are using the Python from the correct environment by running:

```
import sys

print(sys.prefix)
```

Cause

For Python packages, you have set the `PYTHONPATH` or `PYTHONHOME` variable. These environment variables cause Python to load files from locations other than the standard ones. Conda works best when these environment variables are not set, as their typical use cases are obviated by conda environments and a common issue is that they cause Python to pick up the wrong or broken versions of a library.

Solution

For Python packages, make sure you have not set the `PYTHONPATH` or `PYTHONHOME` variables. The command `conda info -a` displays the values of these environment variables.

- To unset these environment variables temporarily for the current terminal session, run `unset PYTHONPATH`.
- To unset them permanently, check for lines in the files:
  - If you use bash---`~/.bashrc`, `~/.bash_profile`, `~/.profile`.
  - If you use zsh---`~/.zshrc`.
  - If you use PowerShell on Windows, the file output by `$PROFILE`.

Cause

You have site-specific directories or, for Python, you have so-called site-specific files. These are typically located in `~/.local` on macOS and Linux. For a full description of the locations of site-specific packages, see PEP 370. As with `PYTHONPATH`, Python may try importing packages from this directory, which can cause issues.

Solution

For Python packages, remove site-specific directories and site-specific files.

Cause

For C libraries, the following environment variables have been set:

- macOS---`DYLD_LIBRARY_PATH`.
- Linux---`LD_LIBRARY_PATH`.

These act similarly to `PYTHONPATH` for Python. If they are set, they can cause libraries to be loaded from locations other than the conda environment. Conda environments obviate most use cases for these variables. The command `conda info -a` shows what these are set to.

Solution

Unset `DYLD_LIBRARY_PATH` or `LD_LIBRARY_PATH`.

Cause

Occasionally, an installed package becomes corrupted. Conda works by unpacking the packages in the `pkgs` directory and then hard-linking them to the environment. Sometimes these get corrupted, breaking all environments that use them. They also break any additional environments since the same files are hard-linked each time.

Solution

Run the command `conda install -f` to unarchive the package again and relink it. It also does an MD5 verification on the package. Usually if this is different it is because your channels have changed and there is a different package with the same name, version, and build number.

> **Note**
>
> This breaks the links to any other environments that already had this package installed, so you have to reinstall it there, too. It also means that running `conda install -f` a lot can use up significant disk space if you have many environments.

> **Note**
>
> The `-f` flag to `conda install` (`--force`) implies `--no-deps`, so `conda install -f package` does not reinstall any of the dependencies of `package`.

pkg_resources.DistributionNotFound: conda==3.6.1-6-gb31b0d4-dirty

Cause

The local version of conda needs updating.

Solution

Force reinstall conda. A useful way to work off the development version of conda is to run `python setup.py develop` on a checkout of the conda GitHub repository. However, if you are not regularly running `git pull`, it is a good idea to un-develop, as you will otherwise not get any regular updates to conda. The normal way to do this is to run `python setup.py develop -u`.

However, this command does not replace the `conda` script itself. With other packages, this is not an issue, as you can just reinstall them with `conda`, but conda cannot be used if conda is installed.

The fix is to use the `./bin/conda` executable in the conda git repository to force reinstall conda. That is, run `./bin/conda install -f conda`. You can then verify with `conda info` that you have the latest version of conda, and not a git checkout. The version should not include any hashes.

macOS error "ValueError unknown locale: UTF-8"

Cause

This is a bug in the macOS Terminal app that shows up only in certain locales. Locales are country-language combinations.

Solution

1. Open Terminal in `/Applications/Utilities`
2. Clear the Set locale environment variables on startup checkbox.

This sets your `LANG` environment variable to be empty. This may cause Terminal to use incorrect settings for your locale. The `locale` command in Terminal tells you what settings are used.

To use the correct language, add a line to your bash profile, which is typically `~/.profile`:

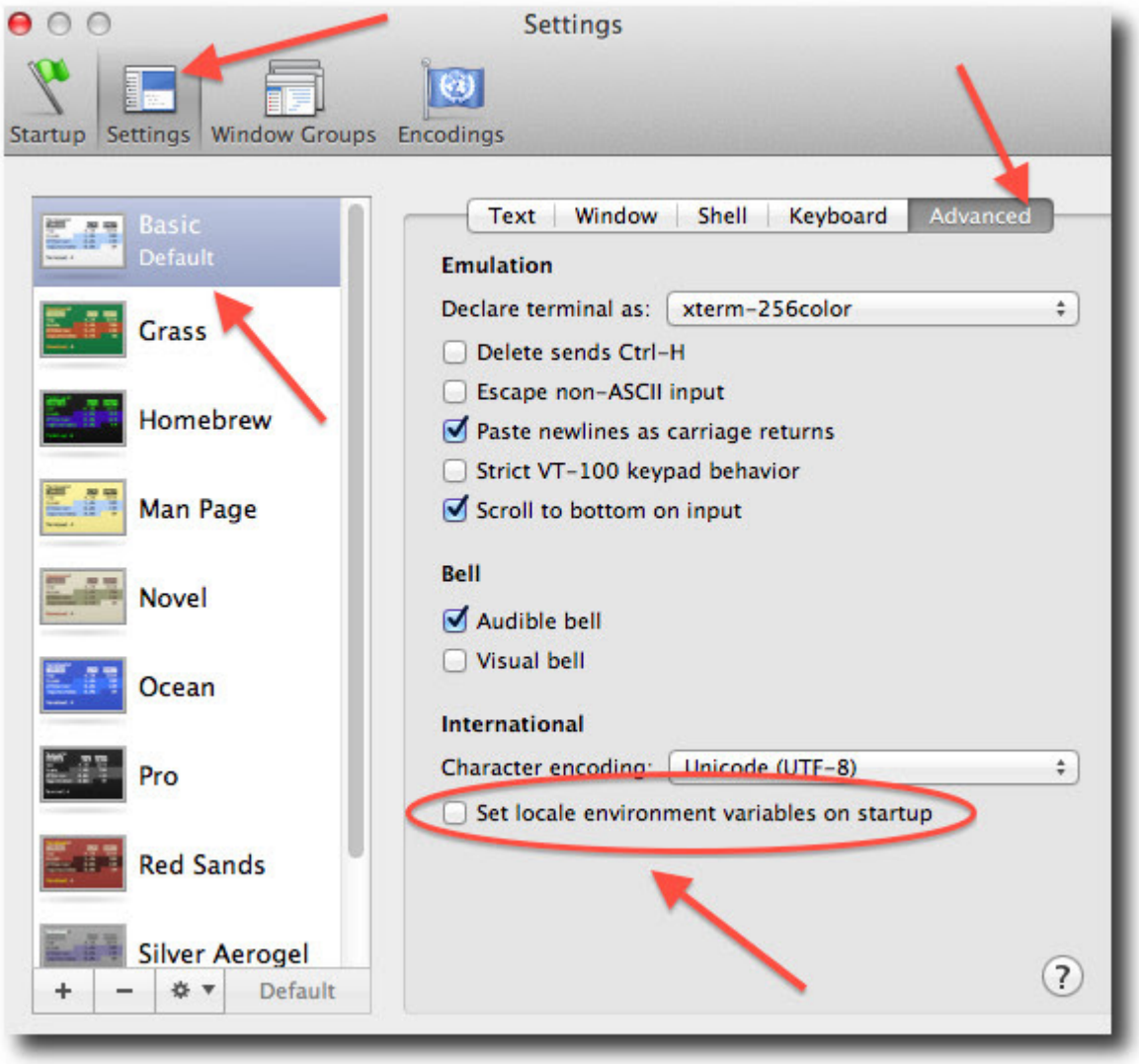
```
export LANG=your-lang
```

> **Note**
>
> Replace `your-lang` with the correct locale specifier for your language.

The command `locale -a` displays all the specifiers. For example, the language code for US English is `en_US.UTF-8`. The locale affects what translations are used when they are available and also how dates, currencies, and decimals are formatted.

AttributeError or missing getproxies

When running a command such as `conda update ipython`, you may get an `AttributeError: 'module' object has no attribute 'getproxies'`.


Settings
Startup
Settings
Window Groups
Encodings
Basic
Default
Grass
Homebrew
Man Page
Novel
Ocean
Pro
Red Sands
Silver Aerogel
Default
Text
Window
Shell
Keyboard
Advanced
Emulation
Declare terminal as:
xterm-256color
Delete sends Ctrl-H
Escape non-ASCII input
Paste newlines as carriage returns
Strict VT-100 keypad behavior
Scroll to bottom on input
Bell
Audible bell
Visual bell
International
Character encoding:
Unicode (UTF-8)
Set locale environment variables on startup

Cause

This can be caused by an old version of `requests` or by having the `PYTHONPATH` environment variable set.

Solution

Update `requests` and be sure `PYTHONPATH` is not set:

1. Run `conda info -a` to show the `requests` version and various environment variables such as `PYTHONPATH`.
2. Update the `requests` version with `pip install -U requests`.
3. Clear `PYTHONPATH`:
    - On Windows, clear it the environment variable settings.
    - On macOS and Linux, clear it by removing it from the bash profile and restarting the shell.

Shell commands open from the wrong location

When you run a command within a conda environment, conda does not access the correct package executable.

Cause

In both bash and zsh, when you enter a command, the shell searches the paths in PATH one by one until it finds the command. The shell then caches the location, which is called hashing in shell terminology. When you run command again, the shell does not have to search the PATH again.

The problem is that before you installed the program, you ran a command which loaded and hashed another version of that program in some other location on the PATH, such as `/usr/bin`. Then you installed the program using `conda install`, but the shell still had the old instance hashed.

Solution

Reactivate the environment or run `hash -r` (in bash) or `rehash` (in zsh).

When you run `conda activate`, conda automatically runs `hash -r` in bash and `rehash` in zsh to clear the hashed commands, so conda finds things in the new path on the PATH. But there is no way to do this when `conda install` is run because the command must be run inside the shell itself, meaning either you have to run the command yourself or used a source file that contains the command.

This is a relatively rare problem, since this happens only in the following circumstances:

1. You activate an environment or use the default environment, and then run a command from somewhere else.
2. Then you `conda install` a program, and then try to run the program again without running `activate` or `deactivate`.

The command `type command_name` always tells you exactly what is being run. This is better than `which command_name`, which ignores hashed commands and searches the PATH directly. The hash is reset by `conda activate` or by `hash -r` in bash or `rehash` in zsh.

Programs fail due to invoking conda Python instead of system Python

Cause

After installing Anaconda or Miniconda, programs that run `python` switch from invoking the system Python to invoking the Python in the root conda environment. If these programs rely on the system Python to have certain configurations or dependencies that are not in the root conda environment Python, the programs may crash. For example, some users of the Cinnamon desktop environment on Linux Mint have reported these crashes.

Solution

Edit your `.bash_profile` and `.bashrc` files so that the conda binary directory, such as `~/miniconda3/bin`, is no longer added to the PATH environment variable. You can still run `conda activate` and `conda deactivate` by using their full path names, such as `~/miniconda3/bin/conda`.

You may also create a folder with symbolic links to `conda activate` and `conda deactivate` and then edit your `.bash_profile` or `.bashrc` file to add this folder to your PATH. If you do this, running `python` will invoke the system Python, but running `conda` commands, `conda activate MyEnv`, `conda activate root`, or `conda deactivate` will work normally.

After running `conda activate` to activate any environment, including after running `conda activate root`, running `python` will invoke the Python in the active conda environment.

UnsatisfiableSpecifications error

Cause

Some conda package installation specifications are impossible to satisfy. For example, `conda create -n tmp python=3 wxpython=3` produces an "Unsatisfiable Specifications" error because wxPython 3 depends on Python 2.7, so the specification to install Python 3 conflicts with the specification to install wxPython 3.

When an unsatisfiable request is made to conda, conda shows a message such as this one:

```
The following specifications were found to be in conflict:
- python 3*
- wxpython 3* -> python 2.7*
Use ``conda search <package> --info`` to see the dependencies
for each package.
```

This indicates that the specification to install wxpython 3 depends on installing Python 2.7, which conflicts with the specification to install Python 3.

Solution

Use `conda search wxpython --info` or `conda search 'wxpython=3' --info` to show information about this package and its dependencies:

```
wxpython 3.0 py27_0
--------------------
file name   : wxpython-3.0-py27_0.tar.bz2
name        : wxpython
version     : 3.0
build number: 0
build string: py27_0
channel     : defaults
size        : 34.1 MB
date        : 2014-01-10
fn          : wxpython-3.0-py27_0.tar.bz2
license_family: Other
md5         : adc6285edfd29a28224c410a39d4bdad
priority    : 2
schannel    : defaults
url         : https://repo.continuum.io/pkgs/free/osx-64/wxpython-3.0-py27_0.tar.bz2
dependencies:
```

(continues on next page)

(continued from previous page)

```
    python 2.7*
    python.app
```

By examining the dependencies of each package, you should be able to determine why the installation request produced a conflict and modify the request so it can be satisfied without conflicts. In this example, you could install wxPython with Python 2.7:

```
conda create -n tmp python=2.7 wxpython=3
```

Package installation fails from a specific channel

Cause

Sometimes it is necessary to install a specific version from a specific channel because that version is not available from the default channel.

Solution

The following example describes the problem in detail and its solution.

Suppose you have a specific need to install the Python `cx_freeze` module with Python 3.4. A first step is to create a Python 3.4 environment:

```
conda create -n py34 python=3.4
```

Using this environment you should first attempt:

```
conda install -n py34 cx_freeze
```

However, when you do this you get the following error:

```
Using Anaconda Cloud api site https://api.anaconda.org
Fetching package metadata .........
Solving package specifications: .
Error: Package missing in current osx-64 channels:
- cx_freeze

You can search for packages on anaconda.org with

  anaconda search -t conda cx_freeze
```

The message indicates that `cx_freeze` cannot be found in the default package channels. However, there may be a community-created version available and you can search for it by running the following command:

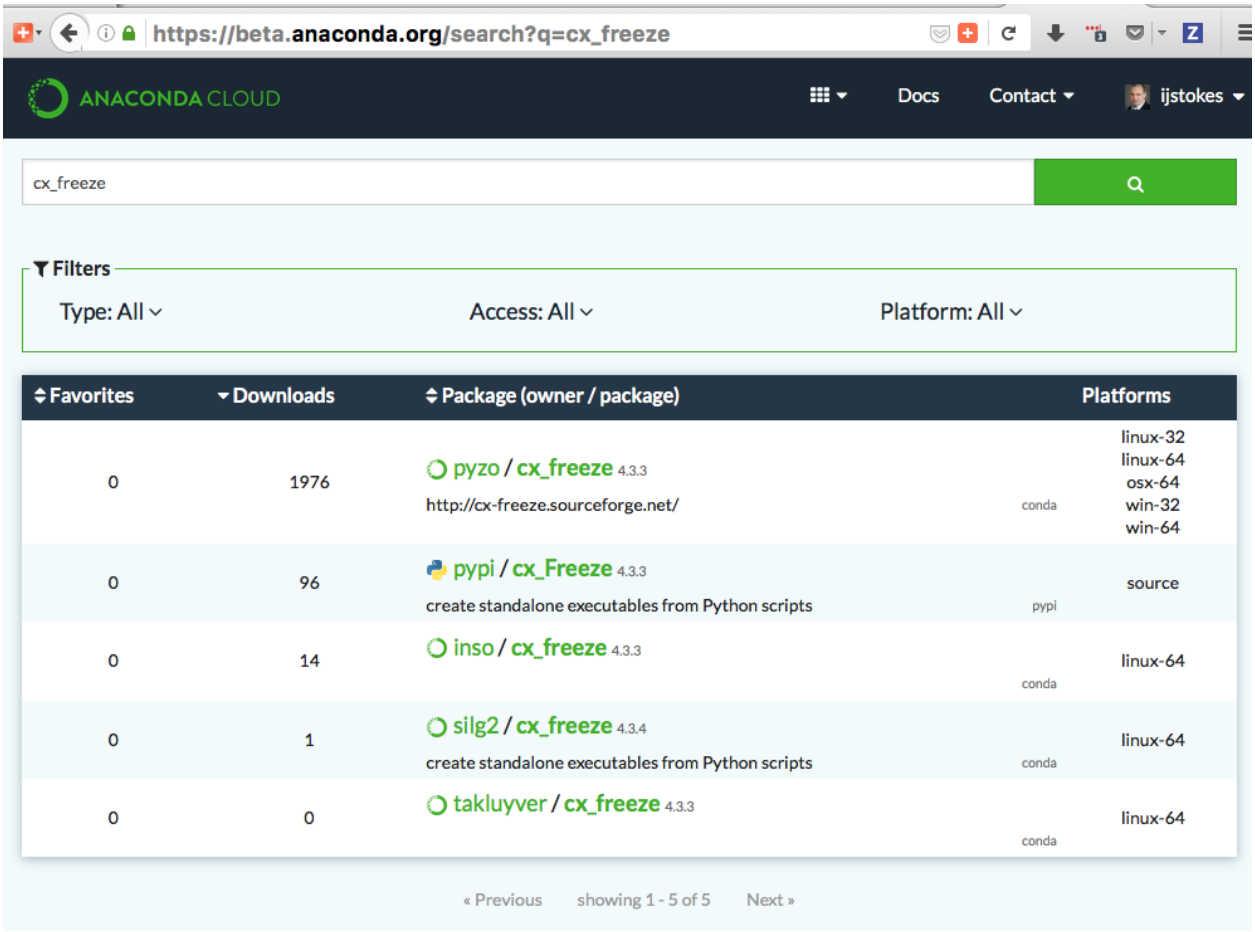
```
$ anaconda search -t conda cx_freeze
Using Anaconda Cloud api site https://api.anaconda.org
Run 'anaconda show <USER/PACKAGE>' to get more details:
Packages:
     Name                      |  Version | Package Types   | Platforms
     ------------------------- |   ------ | --------------- | ---------------
     inso/cx_freeze            |    4.3.3 | conda           | linux-64
     pyzo/cx_freeze            |    4.3.3 | conda           | linux-64, win-32, win-64,␣
↪linux-32, osx-64
```

(continues on next page)

(continued from previous page)

```
                                        : http://cx-freeze.sourceforge.net/
    silg2/cx_freeze           |    4.3.4 | conda           | linux-64
                                        : create standalone executables from Python␣
↪scripts
    takluyver/cx_freeze       |    4.3.3 | conda           | linux-64
Found 4 packages
```

In this example, there are 4 different places that you could try to get the package. None of them are officially supported or endorsed by Anaconda, but members of the conda community have provided many valuable packages. If you want to go with public opinion, then the web interface provides more information:



Notice that the `pyzo` organization has by far the most downloads, so you might choose to use their package. If so, you can add their organization's channel by specifying it on the command line:

```
$ conda create -c pyzo -n cxfreeze_py34 cx_freeze python=3.4
Using Anaconda Cloud api site https://api.anaconda.org
Fetching package metadata: ..........
Solving package specifications: .........
```

(continues on next page)

(continued from previous page)

```
Package plan for installation in environment /Users/username/anaconda/envs/cxfreeze_py34:

The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    cx_freeze-4.3.3            |           py34_4         1.8 MB
    setuptools-20.7.0          |           py34_0         459 KB
    ------------------------------------------------------------
                                           Total:         2.3 MB

The following NEW packages will be INSTALLED:

    cx_freeze:  4.3.3-py34_4
    openssl:    1.0.2h-0
    pip:        8.1.1-py34_1
    python:     3.4.4-0
    readline:   6.2-2
    setuptools: 20.7.0-py34_0
    sqlite:     3.9.2-0
    tk:         8.5.18-0
    wheel:      0.29.0-py34_0
    xz:         5.0.5-1
    zlib:       1.2.8-0
```

Now you have a software environment sandbox created with Python 3.4 and `cx_freeze`.

Conda automatically upgrades to unwanted version

When making a Python package for an app, you create an environment for the app from a file `req.txt` that sets a certain version, such as `python=2.7.9`. However, when you `conda install` your package, it automatically upgrades to a later version, such as `2.7.10`.

Cause

If you make a conda package for the app using conda-build, you can set dependencies with specific version numbers. The requirements lines that say `- python` could be `- python ==2.7.9` instead. It is important to have 1 space before the == operator and no space after.

Solution

Exercise caution when coding version requirements.

Conda upgrade error

Cause

Downgrading conda from 4.6.1 to 4.5.x and then trying to `conda install conda` or `conda upgrade conda` will produce a solving and upgrade error similar to the following:

```
Solving environment: failed
CondaUpgradeError: This environment has previously been operated on by a conda version␣
```

(continues on next page)

(continued from previous page)

```
↪that's newer than the conda currently being used. A newer version of conda is required.
target environment location: /opt/conda
current conda version: 4.5.9
minimum conda version: 4.6
```

Solution

Change the `.condarc` file. Set the parameter by editing the `.condarc` file directly: `allow_conda_downgrades: true` in conda version 4.5.12. This will then let you upgrade. If you have something older than 4.5.12, install conda 4.6.1 again from the package cache.

EXAMPLE: If my conda info says package cache : /opt/conda/pkgs and my Python version is 3.7, then on the command line, type `conda install /opt/conda/pkgs/conda-4.6.1-py37_0.tar.bz2` to resolve the issue.

ValidationError: Invalid value for timestamp

Cause

This happens when certain packages are installed with conda 4.3.28, and then conda is downgraded to 4.3.27 or earlier.

Solution

See https://github.com/conda/conda/issues/6096.

Unicode error after installing Python 2

Example: UnicodeDecodeError: 'ascii' codec can't decode byte 0xd3 in position 1: ordinal not in range(128)

Cause

Python 2 is incapable of handling unicode properly, especially on Windows. In this case, if any character in your PATH env. var contains anything that is not ASCII then you see this exception.

Solution

Remove all non-ASCII from PATH or switch to Python 3.

Windows environment has not been activated

Cause

You may receive a warning message if you have not activated your environment:

```
Warning:
This Python interpreter is in a conda environment, but the environment has
not been activated. Libraries may fail to load. To activate this environment
please see https://conda.io/activation
```

Solution

If you receive this warning, you need to activate your environment. To do so on Windows, on a terminal via PowerShell or the Command Prompt, run: `call <your anaconda/miniconda install location>\Scripts\activate`.

The system cannot find the path specified on Windows

Cause

PATH does not contain entries for all of the necessary conda directories. PATH may have too many entries from 3rd party software adding itself to PATH at install time, despite the user not needing to run the software via PATH lookup.

Solution

Strip PATH to have fewer entries and activate your environment.

If there's some software that needs to be found on PATH (you run it via the CLI), we recommend that you create your own batch files to set PATH dynamically within a console session, rather than permanently modifying PATH in the system settings.

For example, a new conda prompt batch file that first strips PATH, then calls the correct activation procedure could look like:

```
set
PATH="%SystemRoot%\system32;%SystemRoot%;%SystemRoot%\System32\Wbem;%SYSTEMROOT%\
↪System32\WindowsPowerShell\v1.0\;<3rd-party-entries>"
call "<miniconda/anaconda root>\Scripts\activate"
```

If you need to run 3rd party software (software other than Windows built-ins and Anaconda) from this custom conda prompt, then you should add those entries (and only those strictly necessary) to the set PATH entry above. Note that only the quotes wrapping the entire expression should be there. That is how variables are properly set in batch scripts, and these account for any spaces in any entries in PATH. No additional quotes should be within the value assigned to PATH.

To make 3rd party software take precedence over the same-named programs as supplied by conda, add it to PATH after activating conda:

```
set
"PATH=%SystemRoot%\system32;%SystemRoot%;%SystemRoot%\System32\Wbem;%SYSTEMROOT%\
↪System32\WindowsPowerShell\v1.0\"
call "<miniconda/anaconda root>\Scripts\activate"
set "PATH=<3rd-party-entries>;%PATH%"
```

To make conda software take precedence, call the activation script last. Because activation prepends the conda environment PATH entries, they have priority.

```
set
PATH="%SystemRoot%\system32;%SystemRoot%;%SystemRoot%\System32\Wbem;%SYSTEMROOT%\
↪System32\WindowsPowerShell\v1.0\;<3rd-party-entries>"
call "<miniconda/anaconda root>\Scripts\activate"
```

4.1.7 Cheatsheet

The conda cheatsheet contains the most important information about using conda, such as basic commands for creating and managing environments, installing packages, and importing and exporting environments.

See the `conda cheatsheet` for a PDF version.

Versions

- `conda 25.3.1` (latest)
- `conda 24.4.0`

- `conda 4.14.x`
- `conda 4.12.x`
- `conda 4.6.x`

Quickstart

> **Tip**
>
> It is recommended to create a new environment for any new project or workflow.

verify conda install and check version	`conda info`
update conda in base environment	`conda update --name base conda`
install latest anaconda distribution (see release notes)	`conda install anaconda`
create a new environment (tip: name environment descriptively)	`conda create --name ENVNAME`
activate environment (do this before installing packages)	`conda activate ENVNAME`

Channels and Packages

> **Tip**
>
> Package dependencies and platform specifics are automatically resolved when using conda.

list installed packages	`conda list`
list installed packages with source info	`conda list --show-channel-urls`
update all packages	`conda update --all`
install a package from specific channel	`conda install --channel CHANNELNAME PKGNAME` `conda install CHANNELNAME::PKGNAME`
install specific version of package	`conda install PKGNAME=3.1.4`
install package with AND logic	`conda install "PKGNAME>2.5, <3.2"`
install package with OR logic	`conda install "PKGNAME [version='2.5|3.2']"`
uninstall package	`conda uninstall PKGNAME`
view channel sources	`conda config --show-sources`
add channel	`conda config --add channels CHANNELNAME`

Working with Conda Environments

Tip

List environments at the beginning of your session. Environments with an asterisk are active.

list all environments and locations	`conda info --envs`
list all packages + source channels	`conda list --name ENVNAME --show-channel-urls`
install packages in environment	`conda install --name ENVNAME PKGNAME1 PKGNAME2`
remove package from environment	`conda uninstall --name ENVNAME PKGNAME`
update all packages in environment	`conda update --all --name ENVNAME`

Environment Management

Tip

Specifying the environment name confines conda commands to that environment.

create environment with Python version	`conda create --name ENVNAME python=3.10`
clone environment	`conda create --clone ENVNAME --name NEWENV`
rename environment	`conda rename --name ENVNAME NEWENVNAME`
delete environment by name	`conda remove --name ENVNAME --all`
list revisions made to environment	`conda list --name ENVNAME --revisions`
restore environment to a revision	`conda install --name ENVNAME --revision NUMBER`
uninstall package from specific channel	`conda remove --name ENVNAME --channel CHANNELNAME PKGNAME`

Exporting Environments

Tip

Name your export file after your environment to preserve your environment name.

cross-platform compatible (YAML)	`conda export --format=environment-yaml`
cross-platform compatible (JSON)	`conda export --format=environment-json`
cross-platform from history	`conda export --from-history`
platform + package specific (explicit)	`conda export --format=explicit`
platform + package specific (requirements)	`conda export --format=requirements`
auto-detect format by filename	`conda export --file=environment.yaml`
traditional env export	`conda env export > environment.yml`

Importing Environments

> **Tip**
>
> When importing an environment, conda resolves platform and package specifics.

from a .yml file	`conda create --file ENV.yml`
from a .txt file	`conda create --name ENVNAME --file ENV.txt`

Additional Hints

get help for any command	`conda COMMAND --help`
get info for any package	`conda search PKGNAME --info`
run commands w/o user prompt, e.g., installing	`conda COMMAND ARG --yes`
multiple packages	`conda install PKGNAME1 PKGNAME2 --yes`
remove all unused files	`conda clean --all`
examine conda configuration	`conda config --show`

In this guide, you will learn the about the common tasks involved with using the conda package manager.

4.1.8 First steps

If you are brand new to conda, then these are guides that you will want to start with first:

Installing conda
: Follow these instructions to get a working installation of conda on your computer

Getting started
: Learn the essential commands you need in your day-to-day usage of conda

Using conda for your project
: A tutorial explaining how to use conda in your projects

> **See also**
>
> Check out Anaconda's free course on conda basics to learn even more.

4.1.9 Learn more

Configuring conda
: Reference and explanation for all the ways you can configure conda

Working with packages
: Learn how to search for and use conda packages

Working with environments
: Learn how to create, update, remove, and export your conda environments

4.1.10 Additional resources

Cheat sheet
: Commonly used commands organized into a PDF

Troubleshooting
: Various solutions to commonly encountered problems

4.2 New features to try

This page contains recent additions to `conda` worth your attention, including features still in beta that we'd love your feedback on. For the full list of changes across releases, see the *Release notes*.

Stage legend

Stable — Recommended for production use. Some features require configuration.

Beta — Opt in to test, not recommended for production. We want your feedback!

4.2.1 Manage conda plugins

Available in conda 26.11.0 or later Stable

Use `conda plugins list` to see installed plugins and `conda plugins info NAME` to inspect their metadata and registered hooks. Conda 26.11.0 adds `conda plugins install`, `conda plugins update`, and `conda plugins remove` to manage their packages. Conda checks requested packages for conda entry points before confirming installation or updates. This identifies plugin packages, but does not verify that their code is safe.

How to manage a plugin

Choose the example for your distribution. Do not install conda-forge packages into an Anaconda Distribution or Miniconda `base` environment using `defaults`.

Anaconda Distribution / Miniconda

Keep `base` unchanged and create a separate conda-forge environment for `conda-tree`:

```
conda create --name conda-forge-plugins --override-channels --channel conda-forge conda
conda plugins install --name conda-forge-plugins --override-channels --channel conda-
↪forge conda-tree
conda run --name conda-forge-plugins python -m conda plugins list
conda run --name conda-forge-plugins python -m conda plugins info conda-tree
conda plugins update --name conda-forge-plugins --override-channels --channel conda-
↪forge conda-tree
conda plugins remove --name conda-forge-plugins --override-channels --channel conda-
↪forge conda-tree
```

Miniforge

For a Miniforge `base` environment containing compatible conda-forge packages:

```
conda plugins install --name base --override-channels --channel conda-forge conda-tree
conda plugins list
conda plugins info conda-tree
conda plugins update --name base --override-channels --channel conda-forge conda-tree
conda plugins remove --name base --override-channels --channel conda-forge conda-tree
```

Replace the package name in the update command with `--all` to update all conda plugin packages in the same environment, keeping the target and channel options. This disables automatic conda upgrades, but dependency changes may still update conda or other packages. Review the package plan before confirming.

Plugins must be installed in the environment running conda. Choose channels compatible with that environment's existing packages; the guide below explains how to use a separate environment when needed.

See *Managing plugin packages* for more examples, or *the command reference* for all options.

4.2.2 Faster solves: conda-rattler-solver is now conda's default solver

Default in conda 26.10 or later Stable

Starting with **conda 26.10**, **conda-rattler-solver** replaces conda-libmamba-solver as **conda's default solver**, delivering faster dependency resolution, especially for medium and large environments.

`conda-rattler-solver` also natively supports newer conda capabilities, including *Installing packages from PyPI with conda*. This transition also paves the way for removing libmamba and classic solvers from the default conda installation, reducing its size and dependencies.

How to manage your solver

Check your configured solver:

```
conda config --show solver
```

To change your solver configuration:

```
conda config --set solver libmamba
```

Remove the override to use conda's default solver:

```
conda config --remove-key solver
```

For details about available solvers and configuration options, see ***Solvers***.

If you encounter problems, please **report an issue on GitHub**.

4.2.3 Native Windows ARM64 support

Available in conda 26.9 or later Stable

Conda has introduced native support for `win-arm64`, the Windows on ARM platform, and can now resolve and install packages natively for the win-arm64 architecture. This means you no longer need to rely on workarounds like WSL2, x86-64 emulation, or manual compilation to run your Python data science toolchain on ARM hardware.

How to use it

Specify the `win-arm64` target platform when creating an environment:

```
conda create --platform win-arm64 --name myenv-arm64 python
```

Select `win-arm64` as your configured target platform:

```
conda config --set subdir win-arm64
```

Restore automatic platform detection:

```
conda config --remove-key subdir
```

4.2.4 Exclude newer packages from solves and searches

Available in conda 26.9 or later Stable

The `--exclude-newer` flag excludes package records with timestamps after a configured cutoff from environment solves. A configured `exclude_newer` policy also filters `conda search` results. This setting can be used to:

- avoid pulling in brand-new package releases before regressions or malicious versions have had time to surface and be pulled
- support reproducible resolution with time-based cutoffs
- reduce the number of packages the solver considers while creating your environment, causing faster solve times

Usage and limitations

Usage

Use `--exclude-newer` with `conda create`, `conda install`, or `conda update`:

```
conda install --exclude-newer 2026-08-01 scipy
```

You can also configure a global cutoff, per-channel overrides, and per-package exceptions in your `.condarc`:

```
# Global cutoff
exclude_newer: 7d

# Per-channel overrides
channel_settings:
  - channel: conda-forge
    exclude_newer: 3d

# Per-package overrides
exclude_newer_package:
  openssl: false
  ca-certificates: false
  numpy: 30d
```

Package overrides take precedence over channel overrides, which take precedence over the global cutoff.

Accepted values include durations (`7d`, `3d12h`, `1w`), ISO 8601 durations (`P7D`), RFC 3339 timestamps, and dates (`2026-04-01`).

Timestamp behavior and limitations

Conda prefers the channel-provided `indexed_timestamp`, which is intended to record when a package first became available in the channel index. If it is absent or zero, conda falls back to the builder-controlled `timestamp`, which records build time rather than publication time. If a package does not have a useable timestamp, conda cannot compare the package against the exclusion cutoff, so it remains eligible for installation.

For artifacts predating CEP 47, channels may seed `indexed_timestamp` from build timestamps or other historical signals. These values may not reflect the exact time a package became available.

This fallback can still reduce exposure to newly built malicious packages when build timestamps are accurate, allowing time for detection and removal. However, it does not provide a reliable cooldown period after publication:

- An old build uploaded today may pass the cutoff.
- A malicious publisher can backdate the build timestamp.
- A cutoff may delay legitimate security fixes.

Timestamp filtering does not guarantee package safety. It can support reproducible resolution, but does not guarantee an exact reconstruction of a channel's historical state.

For more information, see ***Installing packages with a timestamp cutoff***.

4.2.5 Install packages from PyPI with `conda install`

Available in conda 26.9 or later Stable

The new conda-pypi plugin lets you install supported pure Python wheels from PyPI natively with the `conda install` command. Conda resolves dependencies together, without a separate `pip install` step that can introduce conflicts and make environments harder to reproduce. Once installed, these packages are fully managed by conda: they appear in `conda list`, are captured by `conda export`, and can be removed with `conda remove`.

PyPI packages are made available through the free `conda-pypi` channel, which you can now also **browse on Anaconda.org** to discover available packages.

How to try it

The `conda-rattler-solver` plugin is required. Check your configured solver with `conda config --show solver`. If it isn't `rattler`, select it explicitly:

```
conda config --set solver rattler
```

Add the `conda-pypi` channel after your conda channels:

```
conda config --append channels conda-pypi
```

Search for a package in the channel, then install it:

```
conda search conda-pypi::<package-name>
conda install <package-name>
```

Note: Only pure Python wheels from public PyPI are supported in this workflow. Wheels come directly from PyPI and aren't reviewed before they're added to the channel.

For more details and limitations, see ***Installing packages from PyPI with conda***.

4.2.6 Reproducible environments with multi-platform lockfiles

Available in conda 26.5 or later Stable

Conda can export environments to lockfiles and create environments from them. A lockfile pins exact package versions and builds, so conda recreates the environment without solving again. A single multi-platform lockfile can contain resolved packages for Linux, macOS, and Windows.

The `conda-lock.yaml` and `pixi.lock` formats are supported natively. No additional tools or plugins are required.

How to use it

Export a lockfile for an existing environment:

```
conda export --name myenv --file conda-lock.yaml
```

Create an environment from the lockfile:

```
conda create --name myenv-copy --file conda-lock.yaml
```

Export a lockfile that covers multiple platforms:

```
conda export --name myenv --file conda-lock.yaml \
  --platform linux-64 \
  --platform osx-arm64 \
  --platform win-64
```

For more details, see **Multi-platform lockfiles**.

4.3 Configuration

```
# ##################################################
# ##              Channel Configuration           ##
# ##################################################

# # channels (sequence: primitive)
# #   aliases: channel
# #   env var string delimiter: ','
# #   The list of conda channels to include for relevant operations.
# #
# channels: []

# # channel_alias (str)
# #   The prepended url location to associate with channel names.
# #
# channel_alias: https://conda.anaconda.org

# # channel_settings (sequence: map)
# #   env var string delimiter: ','
# #   A list of mappings that allows overriding certain settings for a
# #   single channel. Each list item should include at least the "channel"
# #   key and the setting you would like to override. The "channel" value
# #   may be a channel name, multichannel name, channel URL, or glob-like
# #   URL pattern. Supported settings include auth-related plugin settings
# #   and exclude_newer.
# #
# channel_settings: []

# # default_channels (sequence: primitive)
# #   env var string delimiter: ','
# #   The list of channel names and/or urls used for the 'defaults'
# #   multichannel. Can be overridden by 'custom_multichannels.defaults'.
# #
# default_channels:
#   - https://repo.anaconda.com/pkgs/main
#   - https://repo.anaconda.com/pkgs/r

# # override_channels_enabled (bool)
# #   Permit use of the --override-channels command-line flag.
# #
# override_channels_enabled: true

# # allowlist_channels (sequence: primitive)
# #   aliases: whitelist_channels
# #   env var string delimiter: ','
# #   The exclusive list of channels allowed to be used on the system. Use
# #   of any other channels will result in an error. If conda-build channels
# #   are to be allowed, along with the --use-local command line flag, be
# #   sure to include the 'local' channel in the list. If the list is empty
```

(continues on next page)

(continued from previous page)

```
# #   or left undefined, no channel exclusions will be enforced.
# #
# allowlist_channels: []

# # denylist_channels (sequence: primitive)
# #   env var string delimiter: ','
# #   The list of channels that are denied to be used on the system. Use of
# #   any of these channels will result in an error. If conda-build channels
# #   are to be allowed, along with the --use-local command line flag, be
# #   sure to not include the 'local' channel in the list. If the list is
# #   empty or left undefined, no channel exclusions will be enforced.
# #
# denylist_channels: []

# # custom_channels (map: primitive)
# #   A map of key-value pairs where the key is a channel name and the value
# #   is a channel location. Channels defined here override the default
# #   'channel_alias' value. The channel name (key) is not included in the
# #   channel location (value).  For example, to override the location of
# #   the 'conda-forge' channel where the url to repodata is
# #   https://anaconda-repo.dev/packages/conda-forge/linux-64/repodata.json,
# #   add an entry 'conda-forge: https://anaconda-repo.dev/packages'.
# #
# custom_channels:
#   pkgs/pro: https://repo.anaconda.com

# # custom_multichannels (map: sequence)
# #   A multichannel is a metachannel composed of multiple channels. The
# #   only reserved multichannel is 'local', which is a list of file://
# #   channel locations where conda-build stashes successfully-built
# #   packages and cannot be overridden. Other multichannels, including
# #   'defaults', can be defined or customized with custom_multichannels,
# #   where the key is the multichannel name and the value is a list of
# #   channel names and/or channel urls. The 'defaults' multichannel can
# #   also be customized using the 'default_channels' parameter (a
# #   historical setting from when 'defaults' was reserved). If both are
# #   defined, 'custom_multichannels.defaults' takes precedence.
# #
# custom_multichannels: {}

# # migrated_channel_aliases (sequence: primitive)
# #   env var string delimiter: ','
# #   A list of previously-used channel_alias values. Useful when switching
# #   between different Anaconda Repository instances.
# #
# migrated_channel_aliases: []

# # migrated_custom_channels (map: primitive)
# #   A map of key-value pairs where the key is a channel name and the value
# #   is the previous location of the channel.
# #
# migrated_custom_channels: {}
```

(continues on next page)

(continued from previous page)

```
# # add_anaconda_token (bool)
# #   aliases: add_binstar_token
# #   In conjunction with the anaconda command-line client (installed with
# #   `conda install anaconda-client`), and following logging into an
# #   Anaconda Server API site using `anaconda login`, automatically apply a
# #   matching private token to enable access to private packages and
# #   channels.
# #
# add_anaconda_token: true

# # allow_non_channel_urls (bool)
# #   Warn, but do not fail, when conda detects a channel url is not a valid
# #   channel.
# #
# allow_non_channel_urls: false

# # repodata_fns (sequence: primitive)
# #   env var string delimiter: ','
# #   Specify filenames for repodata fetching. The default is
# #   ('current_repodata.json', 'repodata.json'), which tries a subset of
# #   the full index containing only the latest version for each package,
# #   then falls back to repodata.json.  You may want to specify something
# #   else to use an alternate index that has been reduced somehow.
# #
# repodata_fns:
#   - current_repodata.json
#   - repodata.json

# # use_only_tar_bz2 (NoneType, bool)
# #   A boolean indicating that only .tar.bz2 conda packages should be
# #   downloaded. This is forced to True if conda-build is installed and
# #   older than 3.18.3, because older versions of conda break when conda
# #   feeds it the new file format.
# #
# use_only_tar_bz2:

# # repodata_threads (int)
# #   Threads to use when downloading and reading repodata.  When not set,
# #   defaults to None, which uses the default ThreadPoolExecutor behavior.
# #
# repodata_threads: 0

# # fetch_threads (int)
# #   Threads to use when downloading packages.  When not set, defaults to
# #   None, which uses the default ThreadPoolExecutor behavior.
# #
# fetch_threads: 0

# # experimental (sequence: primitive)
# #   env var string delimiter: ','
# #   List of experimental features to enable.
```

(continues on next page)

(continued from previous page)

```
# #
# experimental: []

# # no_lock (bool)
# #   Disable index cache lock (defaults to enabled).
# #
# no_lock: false

# # repodata_use_zst (bool)
# #   Use `repodata.json.zst` if available.
# #
# repodata_use_zst: true

# # repodata_use_shards (bool)
# #   Use sharded repodata if available.
# #
# repodata_use_shards: true


# ##########################################################
# ##            Basic Conda Configuration             ##
# ##########################################################

# # envs_dirs (sequence: primitive)
# #   aliases: envs_path
# #   env var string delimiter: ':'
# #   The list of directories to search for named environments. When
# #   creating a new named environment, the environment will be placed in
# #   the first writable location.
# #
# envs_dirs: []

# # pkgs_dirs (sequence: primitive)
# #   env var string delimiter: ','
# #   The list of directories where locally-available packages are linked
# #   from at install time. Packages not locally available are downloaded
# #   and extracted into the first writable directory.
# #
# pkgs_dirs: []

# # default_threads (int)
# #   Threads to use by default for parallel operations.  Default is None,
# #   which allows operations to choose themselves.  For more specific
# #   control, see the other *_threads parameters:     * repodata_threads -
# #   for fetching/loading repodata     * verify_threads - for verifying
# #   package contents in transactions     * execute_threads - for carrying
# #   out the unlinking and linking steps
# #
# default_threads: 0

# # preview (sequence: primitive)
# #   env var string delimiter: ','
```

(continues on next page)

(continued from previous page)

```
# #   List of preview features to opt into.
# #
# preview: []


# ######################################################
# ##              Network Configuration               ##
# ######################################################

# # client_ssl_cert (NoneType, str)
# #   aliases: client_cert
# #   A path to a single file containing a private key and certificate (e.g.
# #   .pem file). Alternately, use client_ssl_cert_key in conjunction with
# #   client_ssl_cert for individual files.
# #
# client_ssl_cert:

# # client_ssl_cert_key (NoneType, str)
# #   aliases: client_cert_key
# #   Used in conjunction with client_ssl_cert for a matching key file.
# #
# client_ssl_cert_key:

# # local_repodata_ttl (bool, int)
# #   For a value of False or 0, always fetch remote repodata (HTTP 304
# #   responses respected). For a value of True or 1, respect the HTTP
# #   Cache-Control max-age header. Any other positive integer values is the
# #   number of seconds to locally cache repodata before checking the remote
# #   server for an update.
# #
# local_repodata_ttl: 1

# # offline (bool)
# #   Restrict conda to cached download content and file:// based urls.
# #
# offline: false

# # proxy_servers (map: primitive)
# #   A mapping to enable proxy settings. Keys can be either (1) a
# #   scheme://hostname form, which will match any request to the given
# #   scheme and exact hostname, or (2) just a scheme, which will match
# #   requests to that scheme. Values are are the actual proxy server, and
# #   are of the form 'scheme://[user:password@]host[:port]'. The optional
# #   'user:password' inclusion enables HTTP Basic Auth with your proxy.
# #
# proxy_servers: {}

# # remote_connect_timeout_secs (float)
# #   The number seconds conda will wait for your client to establish a
# #   connection to a remote url resource.
# #
# remote_connect_timeout_secs: 9.15
```

(continues on next page)

(continued from previous page)

```
# # remote_max_retries (int)
# #   The maximum number of retries each HTTP connection should attempt.
# #
# remote_max_retries: 3

# # remote_backoff_factor (int)
# #   The factor determines the time HTTP connection should wait for
# #   attempt.
# #
# remote_backoff_factor: 1

# # remote_read_timeout_secs (float)
# #   Once conda has connected to a remote resource and sent an HTTP
# #   request, the read timeout is the number of seconds conda will wait for
# #   the server to send a response.
# #
# remote_read_timeout_secs: 60.0

# # ssl_verify (bool, str)
# #   aliases: verify_ssl
# #   Conda verifies SSL certificates for HTTPS requests, just like a web
# #   browser. By default, SSL verification is enabled, and conda operations
# #   will fail if a required url's certificate cannot be verified. Setting
# #   ssl_verify to False disables certification verification. The value for
# #   ssl_verify can also be (1) a path to a CA bundle file, (2) a path to a
# #   directory containing certificates of trusted CA, or (3) 'truststore'
# #   to use the operating system certificate store.
# #
# ssl_verify: true


# ######################################################
# ##               Solver Configuration               ##
# ######################################################

# # add_pip_as_python_dependency (bool)
# #   Add pip as a dependency of python. This ensures pip will always be
# #   installed any time python is installed.
# #
# add_pip_as_python_dependency: true

# # aggressive_update_packages (sequence: primitive)
# #   env var string delimiter: ','
# #   A list of packages that, if installed, are always updated to the
# #   latest possible version.
# #
# aggressive_update_packages:
#   - ca-certificates
#   - certifi
#   - openssl
```

(continues on next page)

(continued from previous page)

```
# # auto_update_conda (bool)
# #   aliases: self_update
# #   Automatically update conda when a newer or higher priority version is
# #   detected.
# #
# auto_update_conda: true

# # channel_priority (ChannelPriority)
# #   Accepts values of 'strict', 'flexible', and 'disabled'. The default
# #   value is 'flexible'. With strict channel priority, packages in lower
# #   priority channels are not considered if a package with the same name
# #   appears in a higher priority channel. With flexible channel priority,
# #   the solver may reach into lower priority channels to fulfill
# #   dependencies, rather than raising an unsatisfiable error. With channel
# #   priority disabled, package version takes precedence, and the
# #   configured priority of channels is used only to break ties. In
# #   previous versions of conda, this parameter was configured as either
# #   True or False. True is now an alias to 'flexible'.
# #
# channel_priority: flexible

# # exclude_newer (str)
# #   aliases: cooldown
# #   Exclude packages published more recently than the given threshold.
# #   Accepts durations (7d, 3d12h, 1w, P7D), ISO 8601 dates (2026-04-01),
# #   RFC 3339 timestamps (2026-04-01T12:00:00Z), or a plain number of
# #   seconds. Date-only values are interpreted as the start of the next day
# #   in UTC. Set to 0 for no delay, using the current time as the cutoff.
# #   Leave empty to disable (the default). Packages without an
# #   indexed_timestamp or timestamp are included for compatibility.
# #   Channel-specific cutoffs can be set with exclude_newer entries in
# #   channel_settings, and per-package overrides can be set with
# #   exclude_newer_package.
# #
# exclude_newer: ''

# # exclude_newer_package (map: primitive)
# #   aliases: cooldown_exclude
# #   Per-package overrides for the exclude_newer policy. Maps package names
# #   to a duration string (e.g. "30d"), a timestamp, or false to exempt the
# #   package entirely. For example:   exclude_newer_package:     openssl:
# #   false     numpy: 30d
# #
# exclude_newer_package: {}

# # create_default_packages (sequence: primitive)
# #   env var string delimiter: ','
# #   Packages that are by default added to a newly created environments.
# #
# create_default_packages: []

# # disallowed_packages (sequence: primitive)
```

(continues on next page)

(continued from previous page)

```
# #   aliases: disallow
# #   env var string delimiter: '&'
# #   Package specifications to disallow installing. The default is to allow
# #   all packages.
# #
# disallowed_packages: []

# # force_reinstall (bool)
# #   Ensure that any user-requested package for the current operation is
# #   uninstalled and reinstalled, even if that package already exists in
# #   the environment.
# #
# force_reinstall: false

# # pinned_packages (sequence: primitive)
# #   env var string delimiter: '&'
# #   A list of package specs to pin for every environment resolution. This
# #   parameter is in BETA, and its behavior may change in a future release.
# #
# pinned_packages: []

# # prefix_data_interoperability (bool)
# #   aliases: pip_interop_enabled
# #   Enable plugins to allow conda to interact with non-conda-installed
# #   packages.
# #
# prefix_data_interoperability: false

# # track_features (sequence: primitive)
# #   env var string delimiter: ','
# #   A list of features that are tracked by default. An entry here is
# #   similar to adding an entry to the create_default_packages list.
# #
# track_features: []

# # solver (str)
# #   aliases: experimental_solver
# #   A string to choose between the different solver logics implemented in
# #   conda. A solver logic takes care of turning your requested packages
# #   into a list of specs to add and/or remove from a given environment,
# #   based on their dependencies and specified constraints.
# #
# solver: rattler


# ######################################################
# ##  Package Linking and Install-time Configuration  ##
# ######################################################

# # allow_softlinks (bool)
# #   When allow_softlinks is True, conda uses hard-links when possible, and
# #   soft-links (symlinks) when hard-links are not possible, such as when
```

(continues on next page)

(continued from previous page)

```
# #   installing on a different filesystem than the one that the package
# #   cache is on. When allow_softlinks is False, conda still uses hard-
# #   links when possible, but when it is not possible, conda copies files.
# #   Individual packages can override this setting, specifying that certain
# #   files should never be soft-linked (see the no_link option in the build
# #   recipe documentation).
# #
# allow_softlinks: false

# # always_copy (bool)
# #   aliases: copy
# #   Register a preference that files be copied into a prefix during
# #   install rather than hard-linked.
# #
# always_copy: false

# # always_softlink (bool)
# #   aliases: softlink
# #   Register a preference that files be soft-linked (symlinked) into a
# #   prefix during install rather than hard-linked. The link source is the
# #   'pkgs_dir' package cache from where the package is being linked.
# #   WARNING: Using this option can result in corruption of long-lived
# #   conda environments. Package caches are *caches*, which means there is
# #   some churn and invalidation. With this option, the contents of
# #   environments can be switched out (or erased) via operations on other
# #   environments.
# #
# always_softlink: false

# # path_conflict (PathConflict)
# #   The method by which conda handle's conflicting/overlapping paths
# #   during a create, install, or update operation. The value must be one
# #   of 'clobber', 'warn', or 'prevent'. The '--clobber' command-line flag
# #   or clobber configuration parameter overrides path_conflict set to
# #   'prevent'.
# #
# path_conflict: clobber

# # rollback_enabled (bool)
# #   Should any error occur during an unlink/link transaction, revert any
# #   disk mutations made to that point in the transaction.
# #
# rollback_enabled: true

# # safety_checks (SafetyChecks)
# #   Enforce available safety guarantees during package installation. The
# #   value must be one of 'enabled', 'warn', or 'disabled'.
# #
# safety_checks: warn

# # extra_safety_checks (bool)
# #   Spend extra time validating package contents.  Currently, runs sha256
```

(continues on next page)

```
# #   verification on every file within each package during installation.
# #
# extra_safety_checks: false

# # signing_metadata_url_base (NoneType, str)
# #   Base URL for obtaining trust metadata updates (i.e., the `*.root.json`
# #   and `key_mgr.json` files) used to verify metadata and (eventually)
# #   package signatures.
# #
# signing_metadata_url_base:

# # shortcuts (bool)
# #   Allow packages to create OS-specific shortcuts (e.g. in the Windows
# #   Start Menu) at install time.
# #
# shortcuts: true

# # shortcuts_only (sequence: primitive)
# #   env var string delimiter: ','
# #   Create shortcuts only for the specified package names.
# #
# shortcuts_only: []

# # non_admin_enabled (bool)
# #   Allows completion of conda's create, install, update, and remove
# #   operations, for non-privileged (non-root or non-administrator) users.
# #
# non_admin_enabled: true

# # separate_format_cache (bool)
# #   Treat .tar.bz2 files as different from .conda packages when filenames
# #   are otherwise similar. This defaults to False, so that your package
# #   cache doesn't churn when rolling out the new package format. If you'd
# #   rather not assume that a .tar.bz2 and .conda from the same place
# #   represent the same content, set this to True.
# #
# separate_format_cache: false

# # verify_threads (int)
# #   Threads to use when performing the transaction verification step.
# #   When not set, defaults to 1.
# #
# verify_threads: 0

# # execute_threads (int)
# #   Threads to use when performing the unlink/link transaction.  When not
# #   set, defaults to 1.  This step is pretty strongly I/O limited, and you
# #   may not see much benefit here.
# #
# execute_threads: 0


# ######################################################
```

(continues on next page)

(continued from previous page)

```
# ##            Conda-build Configuration             ##
# ##########################################################

# # bld_path (str)
# #   The location where conda-build will put built packages. Same as
# #   'croot', but 'croot' takes precedence when both are defined. Also used
# #   in construction of the 'local' multichannel.
# #
# bld_path: ''

# # croot (str)
# #   The location where conda-build will put built packages. Same as
# #   'bld_path', but 'croot' takes precedence when both are defined. Also
# #   used in construction of the 'local' multichannel.
# #
# croot: ''

# # anaconda_upload (NoneType, bool)
# #   aliases: binstar_upload
# #   Automatically upload packages built with conda build to anaconda.org.
# #
# anaconda_upload:

# # conda_build (map: primitive)
# #   aliases: conda-build
# #   General configuration parameters for conda-build.
# #
# conda_build: {}


# ##########################################################
# ##  Output, Prompt, and Flow Control Configuration  ##
# ##########################################################

# # always_yes (NoneType, bool)
# #   aliases: yes
# #   Automatically choose the 'yes' option whenever asked to proceed with a
# #   conda operation, such as when running `conda install`.
# #
# always_yes:

# # auto_activate (bool)
# #   aliases: auto_activate_base
# #   Automatically activate the environment given at
# #   'default_activation_env' during shell initialization.
# #
# auto_activate: true

# # default_activation_env (str)
# #   The environment to be automatically activated on startup if
# #   'auto_activate' is True. Also sets the default environment to activate
# #   when 'conda activate' receives no arguments.
```

(continues on next page)

(continued from previous page)

```
# #
# default_activation_env: base

# # auto_stack (int)
# #   Implicitly use --stack when using activate if current level of nesting
# #   (as indicated by CONDA_SHLVL environment variable) is less than or
# #   equal to specified value. 0 or false disables automatic stacking, 1 or
# #   true enables it for one level.
# #
# auto_stack: 0

# # changeps1 (bool)
# #   When using activate, change the command prompt ($PS1) to include the
# #   activated environment.
# #
# changeps1: true

# # env_prompt (str)
# #   Template for prompt modification based on the active environment.
# #   Currently supported template variables are '{prefix}', '{name}', and
# #   '{default_env}'. '{prefix}' is the absolute path to the active
# #   environment. '{name}' is the basename of the active environment
# #   prefix. '{default_env}' holds the value of '{name}' if the active
# #   environment is a conda named environment ('-n' flag), or otherwise
# #   holds the value of '{prefix}'. Templating uses python's str.format()
# #   method.
# #
# env_prompt: '({default_env}) '

# # json (bool)
# #   Ensure all output written to stdout is structured json.
# #
# json: false

# # console (str)
# #   Configure different backends to be used while rendering normal console
# #   output. Defaults to "classic".
# #
# console: classic

# # notify_outdated_conda (bool)
# #   Notify if a newer version of conda is detected during a create,
# #   install, update, or remove operation.
# #
# notify_outdated_conda: true

# # quiet (bool)
# #   Disable progress bar display and other output.
# #
# quiet: false

# # report_errors (NoneType, bool)
```

(continues on next page)

(continued from previous page)

```
# #   Opt in, or opt out, of automatic error reporting to core maintainers.
# #   Error reports are anonymous, with only the error stack trace and
# #   information given by `conda info` being sent.
# #
# report_errors:

# # show_channel_urls (NoneType, bool)
# #   Show channel URLs when displaying what is going to be downloaded.
# #
# show_channel_urls:

# # list_fields (sequence: primitive)
# #   env var string delimiter: ','
# #   Default fields to report as columns in the output of `conda list`.
# #
# list_fields:
#   - name
#   - version
#   - build
#   - channel_name

# # verbosity (int)
# #   aliases: verbose
# #   Sets output log level. 0 is warn. 1 is info. 2 is debug. 3 is trace.
# #
# verbosity: 0

# # unsatisfiable_hints (bool)
# #   A boolean to determine if conda should find conflicting packages in
# #   the case of a failed install.
# #
# unsatisfiable_hints: true

# # unsatisfiable_hints_check_depth (int)
# #   An integer that specifies how many levels deep to search for
# #   unsatisfiable dependencies. If this number is 1 it will complete the
# #   unsatisfiable hints fastest (but perhaps not the most complete). The
# #   higher this number, the longer the generation of the unsat hint will
# #   take. Defaults to 3.
# #
# unsatisfiable_hints_check_depth: 2

# # number_channel_notices (int)
# #   Sets the number of channel notices to be displayed when running
# #   commands the "install", "create", "update", "env create", and "env
# #   update" . Defaults to 5. In order to completely suppress channel
# #   notices, set this to 0.
# #
# number_channel_notices: 5

# # envvars_force_uppercase (bool)
# #   Force uppercase for new environment variable names. Defaults to True.
```

(continues on next page)

(continued from previous page)

```
# #
# envvars_force_uppercase: true

# # export_platforms (sequence: primitive)
# #   aliases: extra_platforms
# #   env var string delimiter: ','
# #   Additional platform(s)/subdir(s) for export (e.g., linux-64, osx-64,
# #   win-64), current platform is always included.
# #
# export_platforms: []

# # override_virtual_packages (map: primitive)
# #   aliases: virtual_packages
# #   Set override values for virtual packages.
# #
# override_virtual_packages: {}


# ##########################################################
# ##               Plugin Configuration               ##
# ##########################################################

# # no_plugins (bool)
# #   Disable all currently-registered plugins, except built-in conda
# #   plugins.
# #
# no_plugins: false


# ##########################################################
# ##                    Experimental                    ##
# ##########################################################

# # environment_specifier (NoneType, str)
# #   aliases: env_spec
# #   **EXPERIMENTAL** While experimental, expect both major and minor
# #   changes across minor releases.  The name of the environment specifier
# #   plugin that should be used for this context. If not specified, the
# #   plugin manager will try to detect the plugin to use.
# #
# environment_specifier:
```

4.4 Commands

Conda provides many commands for managing packages and environments. The links on this page provide help for each command. You can also access help from the command line with the `--help` flag:

```
conda install --help
```

The following commands are part of conda:

4.4.1 conda activate

```
usage: conda activate [-h]
```

4.4.2 conda clean

Remove unused packages and caches.

```
usage: conda clean [-h] [-a] [-i] [-p] [-t] [-f] [-c [TEMPFILES ...]] [-l]
                   [--json] [--console {classic,json}] [-v] [-q] [-d] [-y]
```

Removal Targets

-a, --all Remove index cache, unused cache packages, tarballs, tempfiles, and logfiles.

-i, --index-cache Remove index cache.

-p, --packages Remove unused packages from writable package caches. WARNING: Environments not listed by *conda info --envs* are not checked for symlinks back to the package cache.

-t, --tarballs Remove cached package tarballs.

-f, --force-pkgs-dirs Remove *all* writable package caches. This option is not included with the --all flag. WARNING: This will break environments with packages installed using symlinks back to the package cache.

-c, --tempfiles Remove temporary files that could not be deleted earlier due to being in-use. The argument for the --tempfiles flag is a path (or list of paths) to the environment(s) where the tempfiles should be found and removed.

-l, --logfiles Remove log files.

Output, Prompt, and Flow Control Options

--json Report all output as json. Suitable for using conda programmatically.

--console Possible choices: classic, json

Select the backend to use for normal output rendering.

-v, --verbose Can be used multiple times. Once for detailed output, twice for INFO logging, thrice for DEBUG logging, four times for TRACE logging.

-q, --quiet Do not display progress bar.

-d, --dry-run Only display what would have been done.

-y, --yes Sets any confirmation values to 'yes' automatically. Users will not be asked to confirm any adding, deleting, backups, etc.

Examples:

conda clean --tarballs

4.4.3 conda compare

Compare packages between conda environments.

```
usage: conda compare [-h] [--json] [--console {classic,json}] [-v] [-q]
                     [-n ENVIRONMENT | -p PATH]
                     file
```

Positional Arguments

file Path to the environment file that is to be compared against.

Output, Prompt, and Flow Control Options

--json Report all output as json. Suitable for using conda programmatically.

--console Possible choices: classic, json

Select the backend to use for normal output rendering.

-v, --verbose Can be used multiple times. Once for detailed output, twice for INFO logging, thrice for DEBUG logging, four times for TRACE logging.

-q, --quiet Do not display progress bar.

Target Environment Specification

-n, --name Name of environment.

-p, --prefix Full path to environment location (i.e. prefix).

Examples:

Compare packages in the current environment with respect to 'environment.yml' located in the current working directory:

conda compare environment.yml

Compare packages installed into the environment 'myenv' with respect to 'environment.yml' in a different directory:

conda compare -n myenv path/to/file/environment.yml

4.4.4 conda config

Modify configuration values in .condarc.

This is modeled after the git config command. Writes to the user .condarc file (/home/docs/.condarc) by default. Use the --show-sources flag to display all identified configuration locations on your computer.

```
usage: conda config [-h] [--json] [--console {classic,json}] [-v] [-q]
                    [--system | --env | --file FILE | -n ENVIRONMENT |
                    -p PATH] [--show [SHOW ...] | --show-sources |
                    --validate | --describe [DESCRIBE ...] | --write-default]
                    [--get [KEY ...] | --append KEY VALUE |
                    --prepend KEY VALUE | --set KEY VALUE |
```

(continues on next page)

(continued from previous page)

```
                    --remove KEY VALUE | --clear KEY | --remove-key KEY |
                    --stdin]
```

Output, Prompt, and Flow Control Options

--json Report all output as json. Suitable for using conda programmatically.

--console Possible choices: classic, json

Select the backend to use for normal output rendering.

-v, --verbose Can be used multiple times. Once for detailed output, twice for INFO logging, thrice for DEBUG logging, four times for TRACE logging.

-q, --quiet Do not display progress bar.

Config File Location Selection

Without one of these flags, the user config file at '/home/docs/.condarc' is used.

--system Write to the system .condarc file at '/home/docs/checkouts/readthedocs.org/user_builds/continuumio-conda/conda/latest/.condarc'.

--env Write to the active conda environment .condarc file (<no active environment>). If no environment is active, write to the user config file (/home/docs/.condarc).

--file Write to the given file.

-n, --name Name of environment.

-p, --prefix Full path to environment location (i.e. prefix).

Config Subcommands

--show Display configuration values as calculated and compiled. If no arguments given, show information for all configuration values.

--show-sources Display all identified configuration sources.

--validate Validate all configuration sources. Iterates over all .condarc files and checks for parsing errors.

--describe Describe given configuration parameters. If no arguments given, show information for all configuration parameters.

--write-default Write the default configuration to a file. Equivalent to *conda config --describe > ~/.condarc*.

Config Modifiers

--get Get a configuration value.

--append Add one configuration value to the end of a list key.

--prepend, --add Add one configuration value to the beginning of a list key.

--set Set a boolean or string key.

--remove **Remove a configuration value from a list key.**
This removes all instances of the value.

--clear Clear all values from a list key.

--remove-key	Remove a configuration key (and all its values).
--stdin	Apply configuration information given in yaml format piped through stdin.

See *conda config --describe* or https://conda.io/docs/config.html for details on all the options that can go in .condarc.

Examples:

Display all configuration values as calculated and compiled:

conda config --show

Display all identified configuration sources:

conda config --show-sources

Print the descriptions of all available configuration options to your command line:

conda config --describe

Print the description for the "channel_priority" configuration option to your command line:

conda config --describe channel_priority

Add the conda-canary channel:

conda config --add channels conda-canary

Set the output verbosity to level 3 (highest) for the current activate environment:

conda config --set verbosity 3 --env

Add the 'conda-forge' channel as a backup to 'defaults':

conda config --append channels conda-forge

4.4.5 conda create

Create a new conda environment from a list of specified packages.

Environments can be created from package specs on the command line, from an input file whose format is detected from its name or contents, or as a clone of an existing environment. See the epilog for the input formats available in your installation.

To use the newly-created environment, use 'conda activate envname'. This command requires either the -n NAME or -p PREFIX option unless --dry-run or --download-only is specified.

```
usage: conda create [-h] [--clone ENV] [-n ENVIRONMENT | -p PATH] [-c CHANNEL]
                    [--use-local] [-0] [--repodata-fn REPODATA_FNS]
                    [--experimental {jlap,lock}] [--no-lock]
                    [--repodata-use-zst | --no-repodata-use-zst]
                    [--repodata-use-shards | --no-repodata-use-shards]
                    [--strict-channel-priority] [--no-channel-priority]
                    [--no-deps | --only-deps] [--no-pin]
                    [--exclude-newer DURATION_OR_DATE] [--copy]
                    [--no-shortcuts] [--shortcuts-only SHORTCUTS_ONLY]
                    [--clobber] [-C] [-k] [--offline] [--json]
                    [--console {classic,json}] [-v] [-q] [-d] [-y]
                    [--download-only] [--show-channel-urls] [-f FILE]
                    [--environment-specifier {cep-24,environment-yaml,env.yml,conda-lock-
```

(continues on next page)

(continued from previous page)

```
↪v1,conda-lock,environment.yml,explicit,rattler-lock-v6,pixi,pixi-lock-v6,requirements.
↪txt,requirements,reqs}]
                    [--format FORMAT] [--no-default-packages]
                    [--subdir SUBDIR]
                    [--solver {classic,libmamba,pycosat,rattler}] [--dev]
                    [package_spec ...]
```

Positional Arguments

package_spec List of packages to install or update in the conda environment.

Named Arguments

--clone Create a new environment as a copy of an existing local environment.

-f, --file Read environment or package specs from a file. The format is detected from the filename or contents. Which formats are supported depends on the installed plugins (see the epilog for the list available here). Custom filenames require --format. May be repeated (e.g. --file=file1 --file=file2).

--environment-specifier, --env-spec Possible choices: cep-24, environment-yaml, env.yml, conda-lock-v1, conda-lock, environment.yml, explicit, rattler-lock-v6, pixi, pixi-lock-v6, requirements.txt, requirements, reqs

--env-spec is deprecated and will be removed in 27.3. Use the *--format* flag instead.

--format Possible choices: cep-24, environment-yaml, env.yml, conda-lock-v1, conda-lock, environment.yml, explicit, rattler-lock-v6, pixi, pixi-lock-v6, requirements.txt, requirements, reqs

Override auto-detection of the input file's format. See *conda export --help* for the formats available in your installation. Aliases are interchangeable with canonical names.

--dev *--dev* is pending deprecation and will be removed in 27.9. Set *PYTHONPATH* to the conda source root instead.

Target Environment Specification

-n, --name Name of environment.

-p, --prefix Full path to environment location (i.e. prefix).

Channel Customization

-c, --channel Additional channel to search for packages. These are URLs searched in the order they are given (including local directories using the 'file://' syntax or simply a path like '/home/conda/mychan' or '../mychan'). Then, the defaults or channels from .condarc are searched (unless --override-channels is given). You can use 'defaults' to get the default packages for conda. You can also use any name and the .condarc channel_alias value will be prepended. The default channel_alias is https://conda.anaconda.org/.

--use-local Use locally built packages. Identical to '-c local'.

-O, --override-channels Do not search default or .condarc channels. Requires --channel.

--repodata-fn Specify file name of repodata on the remote server where your channels are configured or within local backups. Conda will try whatever you specify, but will ultimately fall back to repodata.json if your specs are not satisfiable with what you specify here. This is used to employ repodata that is smaller and reduced in time scope. You may pass this flag more than once. Leftmost entries are tried first, and the fallback to repodata.json is added for you automatically. For more information, see conda config --describe repodata_fns.

--experimental Possible choices: jlap, lock

--experimental is deprecated and will be removed in 27.3. Deprecated: jlap and lock no longer supported.

--no-lock Disable locking when reading, updating index (repodata.json) cache.

--repodata-use-zst, --no-repodata-use-zst Check for/do not check for repodata.json.zst. Enabled by default.

--repodata-use-shards, --no-repodata-use-shards Use sharded repodata if available. Enabled by default.

--subdir, --platform Possible choices: emscripten-wasm32, freebsd-64, linux-32, linux-64, linux-aarch64, linux-armv6l, linux-armv7l, linux-ppc64, linux-ppc64le, linux-riscv64, linux-s390x, osx-64, osx-arm64, wasi-wasm32, win-32, win-64, win-arm64, zos-z

Use packages built for this platform. The new environment will be configured to remember this choice. Should be formatted like 'osx-64', 'linux-32', 'win-64', and so on. Defaults to the current (native) platform.

Solver Mode Modifiers

--strict-channel-priority Packages in lower priority channels are not considered if a package with the same name appears in a higher priority channel.

--no-channel-priority Package version takes precedence over channel priority. Overrides the value given by *conda config --show channel_priority*.

--no-deps Do not install, update, remove, or change dependencies. This WILL lead to broken environments and inconsistent behavior. Use at your own risk.

--only-deps Only install dependencies.

--no-pin Ignore pinned file.

--exclude-newer Exclude packages published more recently than the given duration (e.g. 7d, 3d12h, 1w) or date (e.g. 2026-04-01, 2026-04-01T12:00:00Z). Date-only values use the start of the next UTC day. Supply 0 for no delay, using the current time as the cutoff. Channel and per-package overrides can be set via channel_settings and exclude_newer_package in .condarc.

--no-default-packages Ignore create_default_packages in the .condarc file.

--solver Possible choices: classic, libmamba, pycosat, rattler

Choose which solver backend to use.

Package Linking and Install-time Options

--copy Install all packages using copies instead of hard- or soft-linking.

--no-shortcuts Don't install start menu shortcuts

--shortcuts-only Install shortcuts only for this package name. Can be used several times.

--clobber Allow clobbering (i.e. overwriting) of overlapping file paths within packages and suppress related warnings.

Networking Options

-C, --use-index-cache Use cache of channel index files, even if it has expired. This is useful if you don't want conda to check whether a new version of the repodata file exists, which will save bandwidth.

-k, --insecure Allow conda to perform "insecure" SSL connections and transfers. Equivalent to setting 'ssl_verify' to 'false'.

--offline Offline mode. Don't connect to the Internet.

Output, Prompt, and Flow Control Options

--json Report all output as json. Suitable for using conda programmatically.

--console Possible choices: classic, json

Select the backend to use for normal output rendering.

-v, --verbose Can be used multiple times. Once for detailed output, twice for INFO logging, thrice for DEBUG logging, four times for TRACE logging.

-q, --quiet Do not display progress bar.

-d, --dry-run Only display what would have been done.

-y, --yes Sets any confirmation values to 'yes' automatically. Users will not be asked to confirm any adding, deleting, backups, etc.

--download-only Solve an environment and ensure package caches are populated, but exit prior to unlinking and linking packages into the prefix.

--show-channel-urls Show channel urls. Overrides the value given by *conda config --show show_channel_urls*.

Examples:

Create from package specs:
conda create -n myenv python=3.12 numpy

Create from an environment spec (solved at install time):
conda create -n myenv --file environment.yml

Create from a lockfile (no solve, exact reproduction):
conda create -n myenv --file explicit.txt

Clone an existing environment:
conda create -n env2 --clone env1

Available input formats:

Environment specs:

- cep-24 (aliases: environment-yaml, env.yml): environment.yml, environment.yaml
- environment.yml: environment.yml, environment.yaml
- requirements.txt (aliases: requirements, reqs): requirements.txt, spec.txt

Lockfiles:

- conda-lock-v1 (aliases: conda-lock): conda-lock.yml, conda-lock.yaml
- explicit: explicit.txt
- rattler-lock-v6 (aliases: pixi, pixi-lock-v6): pixi.lock

4.4.6 conda deactivate

```
usage: conda deactivate [-h]
```

4.4.7 conda doctor

Display a health report for your environment by running registered health checks.

> **Tip**
>
> `conda check` is an alias for `conda doctor`. Use whichever you prefer!

Fixing Issues

Use `conda doctor --fix` to automatically fix issues that have available fixes. When a health check detects a problem and has an associated fix, the `--fix` flag will attempt to repair it.

By default, fixes will prompt for confirmation before making changes. Use `--yes` to skip confirmation prompts, or `--dry-run` to see what would be changed without actually making modifications.

Example:

```
# Check environment health
conda doctor

# Fix issues automatically (with confirmation prompts)
conda doctor --fix

# Fix issues without prompts
conda doctor --fix --yes

# Preview fixes without applying them
conda doctor --fix --dry-run
```

Built-in Health Checks

Use `conda doctor --list` to see all available health checks and their fix capabilities. Conda includes several built-in health checks:

- `missing-files` - Detects packages with missing files and can reinstall them
- `altered-files` - Detects packages with modified files (checksum mismatch) and can reinstall them
- `consistency` - Checks for missing or inconsistent dependencies

- `environment-txt` - Verifies the environment is registered in `environments.txt`
- `pinned` - Validates the format of the `pinned` file
- `file-locking` - Reports if file locking is supported
- `requests-ca-bundle` - Validates the SSL certificate bundle configuration

You can run specific health checks by name:

```
conda doctor missing-files altered-files
```

Display a health report for your environment.

```
usage: conda doctor [-h] [-n ENVIRONMENT | -p PATH] [--override-frozen]
                    [--json] [--console {classic,json}] [-v] [-q] [-d] [-y]
                    [--list] [--fix]
                    [NAME ...]
```

Positional Arguments

NAME Health check names to run (default: all). Use --list to see available checks.

Named Arguments

--override-frozen DANGEROUS. Use at your own risk. Ignore protections if the environment is frozen.

--list List all available health checks and their fix capabilities.

--fix, --heal Fix issues found by health checks.

Target Environment Specification

-n, --name Name of environment.

-p, --prefix Full path to environment location (i.e. prefix).

Output, Prompt, and Flow Control Options

--json Report all output as json. Suitable for using conda programmatically.

--console Possible choices: classic, json

Select the backend to use for normal output rendering.

-v, --verbose Can be used multiple times. Once for detailed output, twice for INFO logging, thrice for DEBUG logging, four times for TRACE logging.

-q, --quiet Do not display progress bar.

-d, --dry-run Only display what would have been done.

-y, --yes Sets any confirmation values to 'yes' automatically. Users will not be asked to confirm any adding, deleting, backups, etc.

4.4.8 conda env

```
usage: conda env [-h] command ...
```

Positional Arguments

command Possible choices: config, create, export, list, remove, update

conda env config

Configure a conda environment.

```
usage: conda env config [-h] {vars} ...
```

Examples:

conda env config vars list -n my_env conda env config vars set MY_VAR=foo

conda env config vars

Interact with environment variables associated with Conda environments.

```
usage: conda env config vars [-h] {list,set,unset} ...
```

Examples:

conda env config vars list -n my_env conda env config vars set MY_VAR=foo conda env config vars unset MY_VAR

conda env config vars list

List environment variables for a conda environment.

```
usage: conda env config vars list [-h] [-n ENVIRONMENT | -p PATH] [--json]
                                  [--console {classic,json}] [-v] [-q]
```

Target Environment Specification

-n, --name Name of environment.

-p, --prefix Full path to environment location (i.e. prefix).

Output, Prompt, and Flow Control Options

--json Report all output as json. Suitable for using conda programmatically.

--console Possible choices: classic, json

Select the backend to use for normal output rendering.

-v, --verbose	Can be used multiple times. Once for detailed output, twice for INFO logging, thrice for DEBUG logging, four times for TRACE logging.
-q, --quiet	Do not display progress bar.

Example:

conda env config vars list -n my_env

conda env config vars set

Set environment variables for a conda environment.

```
usage: conda env config vars set [-h] [-n ENVIRONMENT | -p PATH] [vars ...]
```

Positional Arguments

vars	Environment variables to set in the form <KEY>=<VALUE> separated by spaces

Target Environment Specification

-n, --name	Name of environment.
-p, --prefix	Full path to environment location (i.e. prefix).

Example:

conda env config vars set MY_VAR=foo

conda env config vars unset

Unset environment variables for a conda environment.

```
usage: conda env config vars unset [-h] [-n ENVIRONMENT | -p PATH] [vars ...]
```

Positional Arguments

vars	Environment variables to unset in the form <KEY> separated by spaces

Target Environment Specification

-n, --name	Name of environment.
-p, --prefix	Full path to environment location (i.e. prefix).

Example:

conda env config vars unset MY_VAR

conda env create

Create an environment based on an environment definition file.

The file format is detected from the filename or contents. Which formats are supported depends on the plugins installed in your environment. See the epilog for the list of formats available here.

If the file declares a name in its contents (for instance as the first line of an environment.yml file), that name is used unless overridden on the CLI with -n/--name.

Unless you are in the directory containing the environment definition file, use -f to specify the file path of the environment definition file you want to use.

```
usage: conda env create [-h] [-f [FILE ...]] [-n ENVIRONMENT | -p PATH] [-C]
                        [-k] [--offline]
                        [--environment-specifier {cep-24,environment-yaml,env.yml,conda-
↪lock-v1,conda-lock,environment.yml,explicit,rattler-lock-v6,pixi,pixi-lock-v6,
↪requirements.txt,requirements,reqs}]
                        [--format FORMAT] [--no-default-packages] [--json]
                        [--console {classic,json}] [-v] [-q] [-d] [-y]
                        [--solver {classic,libmamba,pycosat,rattler}]
                        [--subdir SUBDIR]
```

Named Arguments

-f, --file Environment definition file (default: environment.yml). Standard filenames registered by the installed format plugins are auto-detected. Custom filenames require --format.

--environment-specifier, --env-spec Possible choices: cep-24, environment-yaml, env.yml, conda-lock-v1, conda-lock, environment.yml, explicit, rattler-lock-v6, pixi, pixi-lock-v6, requirements.txt, requirements, reqs

--env-spec is deprecated and will be removed in 27.3. Use the *--format* flag instead.

--format Possible choices: cep-24, environment-yaml, env.yml, conda-lock-v1, conda-lock, environment.yml, explicit, rattler-lock-v6, pixi, pixi-lock-v6, requirements.txt, requirements, reqs

Override auto-detection of the input file's format. See *conda export --help* for the formats available in your installation. Aliases are interchangeable with canonical names.

--no-default-packages Ignore create_default_packages in the .condarc file.

--solver Possible choices: classic, libmamba, pycosat, rattler

Choose which solver backend to use.

--subdir, --platform Possible choices: emscripten-wasm32, freebsd-64, linux-32, linux-64, linux-aarch64, linux-armv6l, linux-armv7l, linux-ppc64, linux-ppc64le, linux-riscv64, linux-s390x, osx-64, osx-arm64, wasi-wasm32, win-32, win-64, win-arm64, zos-z

Use packages built for this platform. The new environment will be configured to remember this choice. Should be formatted like 'osx-64', 'linux-32', 'win-64', and so on. Defaults to the current (native) platform.

Target Environment Specification

-n, --name Name of environment.

-p, --prefix Full path to environment location (i.e. prefix).

Networking Options

-C, --use-index-cache Use cache of channel index files, even if it has expired. This is useful if you don't want conda to check whether a new version of the repodata file exists, which will save bandwidth.

-k, --insecure Allow conda to perform "insecure" SSL connections and transfers. Equivalent to setting 'ssl_verify' to 'false'.

--offline Offline mode. Don't connect to the Internet.

Output, Prompt, and Flow Control Options

--json Report all output as json. Suitable for using conda programmatically.

--console Possible choices: classic, json

Select the backend to use for normal output rendering.

-v, --verbose Can be used multiple times. Once for detailed output, twice for INFO logging, thrice for DEBUG logging, four times for TRACE logging.

-q, --quiet Do not display progress bar.

-d, --dry-run Only display what would have been done.

-y, --yes Sets any confirmation values to 'yes' automatically. Users will not be asked to confirm any adding, deleting, backups, etc.

Examples:

Create from an environment spec (solved at install time):
conda env create -f /path/to/environment.yml

Create from a lockfile (no solve, exact reproduction):
conda env create -f explicit.txt

Use the default file in the current directory:
conda env create conda env create -n envname

Available input formats:

Environment specs:

- cep-24 (aliases: environment-yaml, env.yml): environment.yml, environment.yaml
- environment.yml: environment.yml, environment.yaml
- requirements.txt (aliases: requirements, reqs): requirements.txt, spec.txt

Lockfiles:

- conda-lock-v1 (aliases: conda-lock): conda-lock.yml, conda-lock.yaml
- explicit: explicit.txt
- rattler-lock-v6 (aliases: pixi, pixi-lock-v6): pixi.lock

conda env export

> **Note**
>
> A newer `conda export` command is now available with enhanced functionality, including multiple export formats and a plugin-based architecture. See *conda export* for the modern approach to environment export.

Export a conda environment to a file. The set of supported formats depends on the plugins installed in your environment. Both portable environment specs and reproducible lockfiles may be available. See the epilog for the list of formats available here.

```
usage: conda env export [-h] [-c CHANNEL] [-O] [--platform EXPORT_PLATFORMS]
                        [--override-platforms] [-n ENVIRONMENT | -p PATH]
                        [-f FILE] [--format FORMAT] [--no-builds]
                        [--ignore-channels] [--json]
                        [--console {classic,json}] [-v] [-q] [--from-history]
```

Named Arguments

-c, --channel Additional channel to include in the export

-O, --override-channels Do not include .condarc channels

--platform, --subdir Target platform(s)/subdir(s) for export (e.g. linux-64, osx-arm64, win-64). For formats that support multi-platform output, repeat the flag to produce a single file covering every platform.

--override-platforms Override the platforms specified in the condarc

-f, --file File name or path for the exported environment. Standard filenames registered by the installed format plugins are auto-detected. Custom filenames require --format. Existing files are overwritten silently.

--format Possible choices: conda-lock, conda-lock-v1, env.yml, environment-json, environment-yaml, explicit, json, pixi, pixi-lock-v6, rattler-lock-v6, reqs, requirements, txt, yaml, yml

Override the output format. When omitted, the format is detected from --file. See the epilog for the formats available in your installation.

--no-builds Remove build specification from dependencies

--ignore-channels Do not include channel names with package names.

--from-history Build environment spec from explicit specs in history

Target Environment Specification

-n, --name Name of environment.

-p, --prefix Full path to environment location (i.e. prefix).

Output, Prompt, and Flow Control Options

--json	Report all output as json. Suitable for using conda programmatically.
--console	Possible choices: classic, json Select the backend to use for normal output rendering.
-v, --verbose	Can be used multiple times. Once for detailed output, twice for INFO logging, thrice for DEBUG logging, four times for TRACE logging.
-q, --quiet	Do not display progress bar.

Examples:

Export an environment spec:
conda export --from-history > environment.yml

Export a lockfile for the same platform:
conda export --file explicit.txt

Export a lockfile for multiple platforms:
conda export --file conda-lock.yml --platform linux-64 --platform osx-arm64

Available output formats:

Lockfiles:

- conda-lock-v1 (aliases: conda-lock): conda-lock.yml, conda-lock.yaml
- explicit: explicit.txt
- rattler-lock-v6 (aliases: pixi, pixi-lock-v6): pixi.lock

Environment specs:

- environment-json (aliases: json): environment.json
- environment-yaml (aliases: yaml, yml, env.yml): environment.yaml, environment.yml
- requirements (aliases: reqs, txt): requirements.txt, spec.txt

Overview

The `conda env export` command exports conda environments to YAML format. This command continues to be fully supported.

For new projects, consider *conda export*, which provides additional features:

- Multiple export formats (YAML, JSON, explicit, requirements)
- Automatic format detection based on filename
- Plugin-based architecture for custom formats
- Enhanced cross-platform compatibility options

Alternative: Enhanced Export Command

If you're interested in additional export formats and functionality, the `conda export` command offers enhanced capabilities:

Traditional (conda env export):

```
conda env export > environment.yml
conda env export --from-history > environment.yml
```

Enhanced (conda export):

```
conda export --format=environment-yaml > environment.yaml
conda export --from-history --format=environment-yaml > environment.yaml
```

The enhanced command provides additional features:

```
# Auto-detect format from filename
conda export --file=environment.yaml

# Export to different formats
conda export --format=environment-json --file=environment.json
conda export --format=explicit --file=explicit.txt
```

See Also

- *conda export* - Enhanced environment export command with multiple formats
- *Managing environments* - Environment management guide

conda env list

An alias for *conda info --envs*. Lists all conda environments.

```
usage: conda env list [-h] [--json] [--console {classic,json}] [-v] [-q]
                      [--size]
```

Named Arguments

--size Show conda-managed disk usage for each environment (excludes untracked files created after installation).

Output, Prompt, and Flow Control Options

--json Report all output as json. Suitable for using conda programmatically.

--console Possible choices: classic, json

Select the backend to use for normal output rendering.

-v, --verbose Can be used multiple times. Once for detailed output, twice for INFO logging, thrice for DEBUG logging, four times for TRACE logging.

-q, --quiet Do not display progress bar.

Examples:

conda env list conda env list --json conda env list --size

conda env remove

Remove an environment.

Removes a provided environment. You must deactivate the existing environment before you can remove it.

```
usage: conda env remove [-h] [--override-frozen] [-n ENVIRONMENT | -p PATH]
                        [--solver {classic,libmamba,pycosat,rattler}] [--json]
                        [--console {classic,json}] [-v] [-q] [-d] [-y]
```

Named Arguments

--override-frozen DANGEROUS. Use at your own risk. Ignore protections if the environment is frozen.

--solver Possible choices: classic, libmamba, pycosat, rattler

Choose which solver backend to use.

Target Environment Specification

-n, --name Name of environment.

-p, --prefix Full path to environment location (i.e. prefix).

Output, Prompt, and Flow Control Options

--json Report all output as json. Suitable for using conda programmatically.

--console Possible choices: classic, json

Select the backend to use for normal output rendering.

-v, --verbose Can be used multiple times. Once for detailed output, twice for INFO logging, thrice for DEBUG logging, four times for TRACE logging.

-q, --quiet Do not display progress bar.

-d, --dry-run Only display what would have been done.

-y, --yes Sets any confirmation values to 'yes' automatically. Users will not be asked to confirm any adding, deleting, backups, etc.

Examples:

conda env remove --name FOO conda env remove -n FOO

conda env update

Update the current environment based on environment file.

```
usage: conda env update [-h]
                        [--environment-specifier {cep-24,environment-yaml,env.yml,conda-
↪lock-v1,conda-lock,environment.yml,explicit,rattler-lock-v6,pixi,pixi-lock-v6,
↪requirements.txt,requirements,reqs}]
```

(continues on next page)

(continued from previous page)

```
                        [--format FORMAT] [--override-frozen]
                        [-n ENVIRONMENT | -p PATH] [-f FILE] [--prune]
                        [--json] [--console {classic,json}] [-v] [-q]
                        [--solver {classic,libmamba,pycosat,rattler}]
```

Named Arguments

--environment-specifier, --env-spec Possible choices: cep-24, environment-yaml, env.yml, conda-lock-v1, conda-lock, environment.yml, explicit, rattler-lock-v6, pixi, pixi-lock-v6, requirements.txt, requirements, reqs

--env-spec is deprecated and will be removed in 27.3. Use the *--format* flag instead.

--format Possible choices: cep-24, environment-yaml, env.yml, conda-lock-v1, conda-lock, environment.yml, explicit, rattler-lock-v6, pixi, pixi-lock-v6, requirements.txt, requirements, reqs

Override auto-detection of the input file's format. See *conda export --help* for the formats available in your installation. Aliases are interchangeable with canonical names.

--override-frozen DANGEROUS. Use at your own risk. Ignore protections if the environment is frozen.

-f, --file environment definition (default: environment.yml)

--prune remove installed packages not defined in environment.yml

--solver Possible choices: classic, libmamba, pycosat, rattler

Choose which solver backend to use.

Target Environment Specification

-n, --name Name of environment.

-p, --prefix Full path to environment location (i.e. prefix).

Output, Prompt, and Flow Control Options

--json Report all output as json. Suitable for using conda programmatically.

--console Possible choices: classic, json

Select the backend to use for normal output rendering.

-v, --verbose Can be used multiple times. Once for detailed output, twice for INFO logging, thrice for DEBUG logging, four times for TRACE logging.

-q, --quiet Do not display progress bar.

Examples:

conda env update conda env update -n=foo conda env update -f=/path/to/environment.yml conda env update --name=foo --file=environment.yml conda env update vader/deathstar

4.4.9 conda export

The `conda export` command allows you to export conda environments to various file formats. This command supports an enhanced plugin-based architecture that supports multiple export formats for different use cases.

Export a conda environment to a file. The set of supported formats depends on the plugins installed in your environment. Both portable environment specs and reproducible lockfiles may be available. See the epilog for the list of formats available here.

```
usage: conda export [-h] [-c CHANNEL] [-O] [--platform EXPORT_PLATFORMS]
                    [--override-platforms] [-n ENVIRONMENT | -p PATH]
                    [-f FILE] [--format FORMAT] [--no-builds]
                    [--ignore-channels] [--json] [--console {classic,json}]
                    [-v] [-q] [--from-history]
```

Named Arguments

-c, --channel Additional channel to include in the export

-O, --override-channels Do not include .condarc channels

--platform, --subdir Target platform(s)/subdir(s) for export (e.g. linux-64, osx-arm64, win-64). For formats that support multi-platform output, repeat the flag to produce a single file covering every platform.

--override-platforms Override the platforms specified in the condarc

-f, --file File name or path for the exported environment. Standard filenames registered by the installed format plugins are auto-detected. Custom filenames require --format. Existing files are overwritten silently.

--format Possible choices: conda-lock, conda-lock-v1, env.yml, environment-json, environment-yaml, explicit, json, pixi, pixi-lock-v6, rattler-lock-v6, reqs, requirements, txt, yaml, yml

Override the output format. When omitted, the format is detected from --file. See the epilog for the formats available in your installation.

--no-builds Remove build specification from dependencies

--ignore-channels Do not include channel names with package names.

--from-history Build environment spec from explicit specs in history

Target Environment Specification

-n, --name Name of environment.

-p, --prefix Full path to environment location (i.e. prefix).

Output, Prompt, and Flow Control Options

--json Report all output as json. Suitable for using conda programmatically.

--console Possible choices: classic, json

Select the backend to use for normal output rendering.

-v, --verbose Can be used multiple times. Once for detailed output, twice for INFO logging, thrice for DEBUG logging, four times for TRACE logging.

-q, --quiet Do not display progress bar.

Examples:

Export an environment spec:
conda export --from-history > environment.yml

Export a lockfile for the same platform:
conda export --file explicit.txt

Export a lockfile for multiple platforms:
conda export --file conda-lock.yml --platform linux-64 --platform osx-arm64

Available output formats:

Lockfiles:

- conda-lock-v1 (aliases: conda-lock): conda-lock.yml, conda-lock.yaml
- explicit: explicit.txt
- rattler-lock-v6 (aliases: pixi, pixi-lock-v6): pixi.lock

Environment specs:

- environment-json (aliases: json): environment.json
- environment-yaml (aliases: yaml, yml, env.yml): environment.yaml, environment.yml
- requirements (aliases: reqs, txt): requirements.txt, spec.txt

Overview

The export command creates portable representations of conda environments that can be shared with others or used to recreate environments on different machines. The command supports multiple output formats through a plugin architecture:

Structured Formats (cross-platform friendly):

- `environment-yaml` - YAML format (default) [aliases: `yaml`, `yml`]
- `environment-json` - JSON format [aliases: `json`]

Text Formats (platform/architecture specific):

- `explicit` - Explicit URLs with `@EXPLICIT` header (CEP 23 compliant)
- `requirements` - Requirements format with MatchSpec strings [aliases: `reqs`, `txt`]

Format Selection

You can specify the export format in three ways:

1. **Explicit format specification** (recommended):

```
conda export --format=environment-yaml
```

2. **File extension detection**:

```
conda export --file=environment.yaml
```

3. **Default behavior** (outputs YAML to stdout):

```
conda export
```

Supported Output Formats

Environment YAML Format

The default format that creates cross-platform compatible environment files.

Format name	`environment-yaml`
Aliases	`yaml`, `yml`
Auto-detected filenames	`environment.yaml`, `environment.yml`

```
# Export to stdout
conda export --format=environment-yaml

# Export to file
conda export --file=environment.yaml

# Alternative using aliases
conda export --format=yaml
conda export --format=yml
```

Example output:

```
name: myenv
channels:
  - conda-forge
  - defaults
dependencies:
  - numpy=2.3.1
  - pandas=2.3.1
  - python=3.13.5
```

Environment JSON Format

JSON representation of the environment for programmatic processing.

Format name	`environment-json`
Aliases	`json`
Auto-detected filenames	`environment.json`

```
# Export to stdout
conda export --format=environment-json

# Export to file
conda export --file=environment.json

# Alternative using alias
conda export --format=json
```

Example output:

```
{
  "name": "myenv",
  "channels": [
    "conda-forge",
    "defaults"
  ],
  "dependencies": [
    "numpy=2.3.1",
    "pandas=2.3.1",
    "python=3.13.5"
  ]
}
```

Explicit Format (CEP 23)

Creates explicit package URLs for exact environment reproduction.

Format name	`explicit`
Aliases	None
Auto-detected filenames	`explicit.txt`

```
# Export to stdout
conda export --format=explicit

# Export to file
conda export --file=explicit.txt
```

Example output:

```
# This file may be used to create an environment using:
# $ conda create --name <env> --file <this file>
# platform: osx-arm64
@EXPLICIT

https://repo.anaconda.com/pkgs/main/noarch/tzdata-2025b-h04d1e81_0.conda
https://repo.anaconda.com/pkgs/main/osx-arm64/libffi-3.4.4-hca03da5_1.conda
https://repo.anaconda.com/pkgs/main/osx-arm64/libcxx-17.0.6-he5c5206_4.conda
https://repo.anaconda.com/pkgs/main/noarch/pip-25.1-pyhc872135_2.conda
https://repo.anaconda.com/pkgs/main/osx-arm64/pytz-2025.2-py313hca03da5_0.conda
https://repo.anaconda.com/pkgs/main/osx-arm64/python-3.13.5-h2eb94d5_100_cp313.conda
https://repo.anaconda.com/pkgs/main/osx-arm64/ca-certificates-2025.2.25-hca03da5_0.conda
https://repo.anaconda.com/pkgs/main/osx-arm64/numexpr-2.11.0-py313hf08b1b5_0.conda
https://repo.anaconda.com/pkgs/main/osx-arm64/python_abi-3.13-0_cp313.conda
https://repo.anaconda.com/pkgs/main/osx-arm64/bzip2-1.0.8-h80987f9_6.conda
https://repo.anaconda.com/pkgs/main/osx-arm64/xz-5.6.4-h80987f9_1.conda
https://repo.anaconda.com/pkgs/main/osx-arm64/libgfortran5-11.3.0-h009349e_28.conda
https://repo.anaconda.com/pkgs/main/osx-arm64/six-1.17.0-py313hca03da5_0.conda
https://repo.anaconda.com/pkgs/main/osx-arm64/blas-1.0-openblas.conda
https://repo.anaconda.com/pkgs/main/osx-arm64/libgfortran-5.0.0-11_3_0_hca03da5_28.conda
https://repo.anaconda.com/pkgs/main/osx-arm64/openssl-3.0.17-h4ee41c1_0.conda
https://repo.anaconda.com/pkgs/main/osx-arm64/wheel-0.45.1-py313hca03da5_0.conda
```

(continues on next page)

(continued from previous page)

```
https://repo.anaconda.com/pkgs/main/osx-arm64/setuptools-72.1.0-py313hca03da5_0.conda
https://repo.anaconda.com/pkgs/main/osx-arm64/numpy-base-2.3.1-py313h2506b34_0.conda
https://repo.anaconda.com/pkgs/main/noarch/python-tzdata-2025.2-pyhd3eb1b0_0.conda
https://repo.anaconda.com/pkgs/main/osx-arm64/llvm-openmp-17.0.6-h2f17746_0.conda
https://repo.anaconda.com/pkgs/main/osx-arm64/numpy-2.3.1-py313h50dd0cd_0.conda
https://repo.anaconda.com/pkgs/main/osx-arm64/bottleneck-1.4.2-py313ha35b7ea_0.conda
https://repo.anaconda.com/pkgs/main/osx-arm64/ncurses-6.4-h313beb8_0.conda
https://repo.anaconda.com/pkgs/main/osx-arm64/tk-8.6.14-h6ba3021_1.conda
https://repo.anaconda.com/pkgs/main/osx-arm64/sqlite-3.50.2-h79febb2_1.conda
https://repo.anaconda.com/pkgs/main/osx-arm64/libmpdec-4.0.0-h80987f9_0.conda
https://repo.anaconda.com/pkgs/main/osx-arm64/expat-2.7.1-h313beb8_0.conda
https://repo.anaconda.com/pkgs/main/osx-arm64/readline-8.2-h1a28f6b_0.conda
https://repo.anaconda.com/pkgs/main/osx-arm64/zlib-1.2.13-h18a0788_1.conda
https://repo.anaconda.com/pkgs/main/osx-arm64/pandas-2.3.1-py313h17050e6_0.conda
https://repo.anaconda.com/pkgs/main/osx-arm64/libopenblas-0.3.29-hea593b9_0.conda
https://repo.anaconda.com/pkgs/main/osx-arm64/python-dateutil-2.9.0post0-py313hca03da5_2.
↪conda
```

Requirements Format

Creates a requirements file with MatchSpec strings.

> **Warning**
>
> This format produces conda MatchSpec strings, which differ from `conda list --export` output. The requirements format includes full package specifications with channels and build strings, while `conda list --export` produces simpler package=version format suitable for basic reproduction.

Format name	`requirements`
Aliases	`reqs`, `txt`
Auto-detected filenames	`requirements.txt`, `spec.txt`

```
# Export to stdout
conda export --format=requirements

# Export to file
conda export --file=requirements.txt

# Alternative using aliases
conda export --format=reqs
conda export --format=txt
```

Example output:

```
# This file may be used to create an environment using:
# $ conda create --name <env> --file <this file>
# platform: osx-arm64
# Note: This is a conda requirements file (MatchSpec format)
# Contains conda package specifications, not pip requirements
```

(continues on next page)

(continued from previous page)

```
pkgs/main::blas==1.0=openblas
pkgs/main::bottleneck==1.4.2=py313ha35b7ea_0
pkgs/main::bzip2==1.0.8=h80987f9_6
pkgs/main::ca-certificates==2025.2.25=hca03da5_0
pkgs/main::expat==2.7.1=h313beb8_0
pkgs/main::libcxx==17.0.6=he5c5206_4
pkgs/main::libffi==3.4.4=hca03da5_1
pkgs/main::libgfortran==5.0.0=11_3_0_hca03da5_28
pkgs/main::libgfortran5==11.3.0=h009349e_28
pkgs/main::libmpdec==4.0.0=h80987f9_0
pkgs/main::libopenblas==0.3.29=hea593b9_0
pkgs/main::llvm-openmp==17.0.6=h2f17746_0
pkgs/main::ncurses==6.4=h313beb8_0
pkgs/main::numexpr==2.11.0=py313hf08b1b5_0
pkgs/main::numpy==2.3.1=py313h50dd0cd_0
pkgs/main::numpy-base==2.3.1=py313h2506b34_0
pkgs/main::openssl==3.0.17=h4ee41c1_0
pkgs/main::pandas==2.3.1=py313h17050e6_0
pkgs/main::pip==25.1=pyhc872135_2
pkgs/main::python==3.13.5=h2eb94d5_100_cp313
pkgs/main::python-dateutil==2.9.0post0=py313hca03da5_2
pkgs/main::python-tzdata==2025.2=pyhd3eb1b0_0
pkgs/main::python_abi==3.13=0_cp313
pkgs/main::pytz==2025.2=py313hca03da5_0
pkgs/main::readline==8.2=h1a28f6b_0
pkgs/main::setuptools==72.1.0=py313hca03da5_0
pkgs/main::six==1.17.0=py313hca03da5_0
pkgs/main::sqlite==3.50.2=h79febb2_1
pkgs/main::tk==8.6.14=h6ba3021_1
pkgs/main::tzdata==2025b=h04d1e81_0
pkgs/main::wheel==0.45.1=py313hca03da5_0
pkgs/main::xz==5.6.4=h80987f9_1
pkgs/main::zlib==1.2.13=h18a0788_1
```

Common Options

Export Current Environment

Export the currently active environment:

```
conda export
```

Export Specific Environment

Export a named environment:

```
conda export --name myenv
```

Export Environment by Path

Export an environment by its path:

```
conda export --prefix /path/to/env
```

Cross-Platform Compatibility

For cross-platform sharing, use the `--from-history` flag with structured formats:

```
# Export only explicitly installed packages (cross-platform friendly)
conda export --from-history --format=environment-yaml

# This excludes dependency packages that might be platform-specific
```

When not to use `--from-history`:

- With `explicit` format (always uses all packages)
- With `requirements` format (always uses all packages)
- When you need exact dependency reproduction

Platform-Specific Exports

You can specify target platforms for export using the `--platform` option or by configuring `export_platforms` in your condarc file:

```
# Export for specific platforms
conda export --platform linux-64 --platform osx-64 --format=environment-yaml
# or
conda export --subdir linux-64 --subdir osx-64 --format=environment-yaml

# Export using condarc configuration
conda export --format=environment-yaml
```

With condarc configuration:

```
export_platforms:
  - linux-64
  - osx-64
  - win-64
```

File Format Detection

The command automatically detects the export format based on filename patterns:

Table 1: File Detection Patterns and Aliases

Filename	Detected Format	Format Aliases	Description
environment.yaml	environment-yaml	yaml, yml	YAML environment file
environment.yml	environment-yaml	yaml, yml	YAML environment file (alternative extension)
environment.json	environment-json	json	JSON environment file
explicit.txt	explicit	None	Explicit URL format
requirements.txt	requirements	reqs, txt	Requirements format
spec.txt	requirements	reqs, txt	Requirements format

You can use either the full format name or any of its aliases:

```
# These are all equivalent
conda export --format=environment-yaml
conda export --format=yaml
conda export --format=yml

# These are all equivalent
conda export --format=requirements
conda export --format=reqs
conda export --format=txt
```

Examples

Basic Usage

```
# Export current environment to YAML (default)
conda export > environment.yaml

# Export specific environment
conda export --name myenv --format=environment-yaml

# Export with cross-platform compatibility
conda export --from-history --file=environment.yaml
```

Advanced Usage

```
# Export to explicit format for exact reproduction
conda export --format=explicit --file=explicit.txt

# Export to requirements format using aliases
conda export --format=reqs > requirements.txt
conda export --format=txt --file=spec.txt

# Export YAML using short alias
```

(continues on next page)

(continued from previous page)

```
conda export --format=yml --file=environment.yml

# Export JSON for programmatic processing
conda export --format=json --file=environment.json

# Export with custom channels
conda export --channel conda-forge --format=environment-yaml

# Export for multiple platforms
conda export --platform linux-64 --platform osx-64 --format=environment-yaml
conda export --subdir linux-64 --subdir osx-64 --format=environment-yaml

# Export for specific platform only
conda export --platform win-64 --format=explicit --file=explicit-win-64.txt
conda export --subdir win-64 --format=explicit --file=explicit-win-64.txt
```

Error Handling

The export command will fail in these cases:

- **Empty environments with text formats**: The `explicit` and `requirements` formats require installed packages
- **Unrecognized filenames**: Files that don't match supported patterns
- **Invalid format names**: Format names that don't exist

For unrecognized filenames, specify the format explicitly:

```
# This will fail
conda export --file=my-custom-file.xyz

# This will work
conda export --file=my-custom-file.xyz --format=environment-yaml
```

Plugin Architecture

The export functionality is built on a plugin architecture that allows extending conda with custom export formats. For information on creating custom exporters, see *Environment Exporters*.

See Also

- *conda env export* - Traditional environment export command
- *Managing environments* - Environment management guide
- *Environment Exporters* - Plugin development guide

4.4.10 conda info

Display information about current conda install.

```
usage: conda info [-h] [--json] [--console {classic,json}] [-v] [-q] [-a]
                  [--base] [-e] [-s] [--size] [--unsafe-channels]
```

Named Arguments

-a, --all	Show all information.
--base	Display base environment path.
-e, --envs	List all known conda environments. Combine with *--json* to obtain more details.
-s, --system	List environment variables.
--size	Show conda-managed disk usage for each environment (excludes untracked files created after installation).
--unsafe-channels	Display list of channels with tokens exposed.

Output, Prompt, and Flow Control Options

--json	Report all output as json. Suitable for using conda programmatically.
--console	Possible choices: classic, json Select the backend to use for normal output rendering.
-v, --verbose	Can be used multiple times. Once for detailed output, twice for INFO logging, thrice for DEBUG logging, four times for TRACE logging.
-q, --quiet	Do not display progress bar.

4.4.11 conda init

Initialize conda for shell interaction.

```
usage: conda init [-h] [--all] [--condabin] [--user] [--no-user] [--system]
                  [--reverse] [--json] [--console {classic,json}] [-v] [-q]
                  [-d]
                  [SHELLS ...]
```

Positional Arguments

SHELLS	Possible choices: bash, fish, tcsh, xonsh, zsh, powershell One or more shells to be initialized. If not given, the default value is 'bash' on unix and 'cmd.exe' & 'powershell' on Windows. Use the '--all' flag to initialize all shells. Available shells: ['bash', 'fish', 'powershell', 'tcsh', 'xonsh', 'zsh']

Named Arguments

--all	Initialize all currently available shells.
-d, --dry-run	Only display what would have been done.

setup type

--condabin	Add 'condabin/' directory to PATH only. Does not install the shell function.
--user	Initialize conda for the current user (default).
--no-user	Don't initialize conda for the current user.
--system	Initialize conda for all users on the system.
--reverse	Undo effects of last conda init.

Output, Prompt, and Flow Control Options

--json	Report all output as json. Suitable for using conda programmatically.
--console	Possible choices: classic, json Select the backend to use for normal output rendering.
-v, --verbose	Can be used multiple times. Once for detailed output, twice for INFO logging, thrice for DEBUG logging, four times for TRACE logging.
-q, --quiet	Do not display progress bar.

Key parts of conda's functionality require that it interact directly with the shell within which conda is being invoked. The *conda activate* and *conda deactivate* commands specifically are shell-level commands. That is, they affect the state (e.g. environment variables) of the shell context being interacted with. Other core commands, like *conda create* and *conda install*, also necessarily interact with the shell environment. They're therefore implemented in ways specific to each shell. Each shell must be configured to make use of them.

The --condabin option adds the "$CONDA_PREFIX/condabin" directory to the PATH environment variable. This directory only contains the conda executable and does not contain other executables from other packages installed in the base environment. On most shells, a small snippet is added to the shell profile. For CMD, the PATH environment variable is modified directly in the registry.

This command makes changes to your system that are specific and customized for each shell. To see the specific files and locations on your system that will be affected before, use the '--dry-run' flag. To see the exact changes that are being or will be made to each location, use the '--verbose' flag.

IMPORTANT: After running *conda init*, most shells will need to be closed and restarted for changes to take effect.

4.4.12 conda install

Install a list of packages into a specified conda environment.

This command accepts a list of package specifications (e.g, bitarray=0.8) and installs a set of packages consistent with those specifications and compatible with the underlying environment. If full compatibility cannot be assured, an error is reported and the environment is not changed.

Conda attempts to install the newest versions of the requested packages. To accomplish this, it may update some packages that are already installed, or install additional packages. To prevent existing packages from updating, use the --freeze-installed option. This may force conda to install older versions of the requested packages, and it does not prevent additional dependency packages from being installed.

If you wish to skip dependency checking altogether, use the '--no-deps' option. This may result in an environment with incompatible packages, so this option must be used with great caution.

conda can also be called with a list of explicit conda package filenames (e.g. ./lxml-3.2.0-py27_0.tar.bz2). Using conda in this mode implies the --no-deps option, and should likewise be used with great caution. Explicit filenames and package specifications cannot be mixed in a single command.

When using --file, only the package list from the file is used. Any name or prefix in the file (e.g. in environment.yml) is ignored; packages are installed into the target environment (-n/-p or the current environment).

```
usage: conda install [-h] [--revision REVISION] [--override-frozen]
                     [-n ENVIRONMENT | -p PATH] [-c CHANNEL] [--use-local]
                     [-O] [--repodata-fn REPODATA_FNS]
                     [--experimental {jlap,lock}] [--no-lock]
```

(continues on next page)

(continued from previous page)

```
                    [--repodata-use-zst | --no-repodata-use-zst]
                    [--repodata-use-shards | --no-repodata-use-shards]
                    [--strict-channel-priority] [--no-channel-priority]
                    [--no-deps | --only-deps] [--no-pin]
                    [--exclude-newer DURATION_OR_DATE] [--copy]
                    [--no-shortcuts] [--shortcuts-only SHORTCUTS_ONLY]
                    [--clobber] [-C] [-k] [--offline] [--json]
                    [--console {classic,json}] [-v] [-q] [-d] [-y]
                    [--download-only] [--show-channel-urls] [-f FILE]
                    [--environment-specifier {cep-24,environment-yaml,env.yml,conda-
↪lock-v1,conda-lock,environment.yml,explicit,rattler-lock-v6,pixi,pixi-lock-v6,
↪requirements.txt,requirements,reqs}]
                    [--format FORMAT]
                    [--solver {classic,libmamba,pycosat,rattler}]
                    [--force-reinstall] [--freeze-installed | --update-deps |
                    -S | --update-all | --update-specs] [--dev]
                    [package_spec ...]
```

Positional Arguments

package_spec List of packages to install or update in the conda environment.

Named Arguments

--revision Revert to the specified REVISION.

--override-frozen DANGEROUS. Use at your own risk. Ignore protections if the environment is frozen.

-f, --file Read environment or package specs from a file. The format is detected from the filename or contents. Which formats are supported depends on the installed plugins (see the epilog for the list available here). Custom filenames require --format. May be repeated (e.g. --file=file1 --file=file2).

--environment-specifier, --env-spec Possible choices: cep-24, environment-yaml, env.yml, conda-lock-v1, conda-lock, environment.yml, explicit, rattler-lock-v6, pixi, pixi-lock-v6, requirements.txt, requirements, reqs

--env-spec is deprecated and will be removed in 27.3. Use the *--format* flag instead.

--format Possible choices: cep-24, environment-yaml, env.yml, conda-lock-v1, conda-lock, environment.yml, explicit, rattler-lock-v6, pixi, pixi-lock-v6, requirements.txt, requirements, reqs

Override auto-detection of the input file's format. See *conda export --help* for the formats available in your installation. Aliases are interchangeable with canonical names.

--dev *--dev* is pending deprecation and will be removed in 27.9. Set *PYTHONPATH* to the conda source root instead.

Target Environment Specification

-n, --name Name of environment.

-p, --prefix Full path to environment location (i.e. prefix).

Channel Customization

-c, --channel Additional channel to search for packages. These are URLs searched in the order they are given (including local directories using the 'file://' syntax or simply a path like '/home/conda/mychan' or '../mychan'). Then, the defaults or channels from .condarc are searched (unless --override-channels is given). You can use 'defaults' to get the default packages for conda. You can also use any name and the .condarc channel_alias value will be prepended. The default channel_alias is https://conda.anaconda.org/.

--use-local Use locally built packages. Identical to '-c local'.

-O, --override-channels Do not search default or .condarc channels. Requires --channel.

--repodata-fn Specify file name of repodata on the remote server where your channels are configured or within local backups. Conda will try whatever you specify, but will ultimately fall back to repodata.json if your specs are not satisfiable with what you specify here. This is used to employ repodata that is smaller and reduced in time scope. You may pass this flag more than once. Leftmost entries are tried first, and the fallback to repodata.json is added for you automatically. For more information, see conda config --describe repodata_fns.

--experimental Possible choices: jlap, lock

--experimental is deprecated and will be removed in 27.3. Deprecated: jlap and lock no longer supported.

--no-lock Disable locking when reading, updating index (repodata.json) cache.

--repodata-use-zst, --no-repodata-use-zst Check for/do not check for repodata.json.zst. Enabled by default.

--repodata-use-shards, --no-repodata-use-shards Use sharded repodata if available. Enabled by default.

Solver Mode Modifiers

--strict-channel-priority Packages in lower priority channels are not considered if a package with the same name appears in a higher priority channel.

--no-channel-priority Package version takes precedence over channel priority. Overrides the value given by *conda config --show channel_priority*.

--no-deps Do not install, update, remove, or change dependencies. This WILL lead to broken environments and inconsistent behavior. Use at your own risk.

--only-deps Only install dependencies.

--no-pin Ignore pinned file.

--exclude-newer Exclude packages published more recently than the given duration (e.g. 7d, 3d12h, 1w) or date (e.g. 2026-04-01, 2026-04-01T12:00:00Z). Date-only values use the start of the next UTC day. Supply 0 for no delay, using the current time as the cutoff. Channel and per-package overrides can be set via channel_settings and exclude_newer_package in .condarc.

--solver Possible choices: classic, libmamba, pycosat, rattler

Choose which solver backend to use.

--force-reinstall Ensure that any user-requested package for the current operation is uninstalled and reinstalled, even if that package already exists in the environment.

--freeze-installed, --no-update-deps Do not update or change already-installed dependencies.

--update-deps Update dependencies that have available updates.

-S, --satisfied-skip-solve Exit early and do not run the solver if the requested specs are satisfied. Also skips aggressive updates as configured by the 'aggressive_update_packages' config setting. Use 'conda config --describe aggressive_update_packages' to view your setting. --satisfied-skip-solve is similar to the default behavior of 'pip install'.

--update-all, --all Update all installed packages in the environment.

--update-specs Update based on provided specifications.

Package Linking and Install-time Options

--copy Install all packages using copies instead of hard- or soft-linking.

--no-shortcuts Don't install start menu shortcuts

--shortcuts-only Install shortcuts only for this package name. Can be used several times.

--clobber Allow clobbering (i.e. overwriting) of overlapping file paths within packages and suppress related warnings.

Networking Options

-C, --use-index-cache Use cache of channel index files, even if it has expired. This is useful if you don't want conda to check whether a new version of the repodata file exists, which will save bandwidth.

-k, --insecure Allow conda to perform "insecure" SSL connections and transfers. Equivalent to setting 'ssl_verify' to 'false'.

--offline Offline mode. Don't connect to the Internet.

Output, Prompt, and Flow Control Options

--json Report all output as json. Suitable for using conda programmatically.

--console Possible choices: classic, json

Select the backend to use for normal output rendering.

-v, --verbose Can be used multiple times. Once for detailed output, twice for INFO logging, thrice for DEBUG logging, four times for TRACE logging.

-q, --quiet Do not display progress bar.

-d, --dry-run Only display what would have been done.

-y, --yes Sets any confirmation values to 'yes' automatically. Users will not be asked to confirm any adding, deleting, backups, etc.

--download-only Solve an environment and ensure package caches are populated, but exit prior to unlinking and linking packages into the prefix.

--show-channel-urls Show channel urls. Overrides the value given by *conda config --show show_channel_urls*.

Examples:

Install the package 'scipy' into the currently-active environment:

    conda install scipy

Install a list of packages into an environment, myenv:

    conda install -n myenv scipy curl wheel

Install a specific version of 'python' into an environment, myenv:

    conda install -p path/to/myenv python=3.11

4.4.13 conda list

List installed packages in a conda environment.

```
usage: conda list [-h] [-n ENVIRONMENT | -p PATH] [--json]
                  [--console {classic,json}] [-v] [-q] [--show-channel-urls]
                  [--fields LIST_FIELDS] [--reverse] [-c] [-f] [--explicit]
                  [--md5] [--sha256] [-e] [-r] [--size] [--no-pip] [--auth]
                  [regex]
```

Positional Arguments

regex List only packages matching this regular expression.

Named Arguments

--show-channel-urls Show channel urls. Overrides the value given by *conda config --show show_channel_urls*.

--fields Comma-separated list of fields to print. Valid values: arch,build,build_number,channel,channel_name,constrains,depends,dist_str,features,fn,license,license_fan

--reverse List installed packages in reverse order.

-c, --canonical Output canonical names of packages only.

-f, --full-name Only search for full names, i.e., ^<regex>$. --full-name NAME is identical to regex '^NAME$'.

--explicit List explicitly all installed conda packages with URL (output may be used by conda create --file).

--md5 Add MD5 hashsum when using --explicit.

--sha256 Add SHA256 hashsum when using --explicit.

-e, --export Output explicit, machine-readable requirement strings instead of human-readable lists of packages. This output may be used by conda create --file.

-r, --revisions List the revision history.

--size Show package and environment sizes.

--no-pip Do not include pip-only installed packages.

--auth In explicit mode, leave authentication details in package URLs. They are removed by default otherwise.

Target Environment Specification

-n, --name Name of environment.

-p, --prefix Full path to environment location (i.e. prefix).

Output, Prompt, and Flow Control Options

--json Report all output as json. Suitable for using conda programmatically.

--console Possible choices: classic, json

Select the backend to use for normal output rendering.

-v, --verbose Can be used multiple times. Once for detailed output, twice for INFO logging, thrice for DEBUG logging, four times for TRACE logging.

-q, --quiet Do not display progress bar.

Examples:

List all packages in the current environment:

```
conda list
```

List all packages in reverse order:

```
conda list --reverse
```

List all packages installed into the environment 'myenv':

```
conda list -n myenv
```

List all packages that begin with the letters "py", using regex:

```
conda list ^py
```

List name and version only:

```
conda list --fields name,version
```

Save packages for future use:

```
conda list --export > package-list.txt
```

Reinstall packages from an export file:

```
conda create -n myenv --file package-list.txt
```

4.4.14 conda notices

Retrieve latest channel notifications.

Conda channel maintainers have the option of setting messages that users will see intermittently. Some of these notices are informational while others are messages concerning the stability of the channel.

```
usage: conda notices [-h] [-c CHANNEL] [--use-local] [-O]
                     [--repodata-fn REPODATA_FNS] [--experimental {jlap,lock}]
                     [--no-lock] [--repodata-use-zst | --no-repodata-use-zst]
                     [--repodata-use-shards | --no-repodata-use-shards]
                     [--json] [--console {classic,json}] [-v] [-q]
```

Channel Customization

-c, --channel Additional channel to search for packages. These are URLs searched in the order they are given (including local directories using the 'file://' syntax or simply a path like '/home/conda/mychan' or '../mychan'). Then, the defaults or channels from .condarc are searched (unless --override-channels is given). You can use 'defaults' to get the default packages for conda. You can also use any name and the .condarc channel_alias value will be prepended. The default channel_alias is https://conda.anaconda.org/.

--use-local Use locally built packages. Identical to '-c local'.

-O, --override-channels Do not search default or .condarc channels. Requires --channel.

--repodata-fn Specify file name of repodata on the remote server where your channels are configured or within local backups. Conda will try whatever you specify, but will ultimately fall back to repodata.json if your specs are not satisfiable with what you specify here. This is used to employ repodata that is smaller and reduced in time scope. You may pass this flag more than once. Leftmost entries are tried first, and the fallback to repodata.json is added for you automatically. For more information, see conda config --describe repodata_fns.

--experimental Possible choices: jlap, lock

--experimental is deprecated and will be removed in 27.3. Deprecated: jlap and lock no longer supported.

--no-lock Disable locking when reading, updating index (repodata.json) cache.

--repodata-use-zst, --no-repodata-use-zst Check for/do not check for repodata.json.zst. Enabled by default.

--repodata-use-shards, --no-repodata-use-shards Use sharded repodata if available. Enabled by default.

Output, Prompt, and Flow Control Options

--json Report all output as json. Suitable for using conda programmatically.

--console Possible choices: classic, json

Select the backend to use for normal output rendering.

-v, --verbose Can be used multiple times. Once for detailed output, twice for INFO logging, thrice for DEBUG logging, four times for TRACE logging.

-q, --quiet Do not display progress bar.

Examples:

conda notices

conda notices -c defaults

4.4.15 conda package

Create low-level conda packages. (EXPERIMENTAL)

```
usage: conda package [-h] [-n ENVIRONMENT | -p PATH] [-w PATH [PATH ...]] [-r]
                     [-u] [--pkg-name PKG_NAME] [--pkg-version PKG_VERSION]
                     [--pkg-build PKG_BUILD]
```

Named Arguments

-w, --which	Given some file's PATH, print which conda package the file came from.
-r, --reset	Remove all untracked files and exit.
-u, --untracked	Display all untracked files and exit.
--pkg-name	Designate package name of the package being created.
--pkg-version	Designate package version of the package being created.
--pkg-build	Designate package build number of the package being created.

Target Environment Specification

-n, --name	Name of environment.
-p, --prefix	Full path to environment location (i.e. prefix).

4.4.16 conda plugins

Inspect plugins loaded by the running conda installation and manage conda plugin packages in a selected environment. See *Plugins* for installation, update, and removal examples and guidance on choosing the environment that runs your plugins.

Added in version 26.11.0: The `install`, `remove`, and `update` subcommands.

Manage conda plugins.

```
usage: conda plugins [-h] {info,install,list,remove,update} ...
```

subcommands

subcommand	Possible choices: info, install, list, remove, update

Sub-commands

info

Show detailed information about an installed conda plugin.

```
conda plugins info [-h] [--json] [--console {classic,json}] [-v] [-q] NAME
```

Positional Arguments

NAME Installed plugin distribution name or canonical plugin name.

Output, Prompt, and Flow Control Options

--json Report all output as json. Suitable for using conda programmatically.

--console Possible choices: classic, json

Select the backend to use for normal output rendering.

-v, --verbose Can be used multiple times. Once for detailed output, twice for INFO logging, thrice for DEBUG logging, four times for TRACE logging.

-q, --quiet Do not display progress bar.

install

Install conda plugin packages into an environment.

```
conda plugins install [-h] [--override-frozen] [-n ENVIRONMENT | -p PATH]
                      [-c CHANNEL] [--use-local] [-O]
                      [--repodata-fn REPODATA_FNS]
                      [--experimental {jlap,lock}] [--no-lock]
                      [--repodata-use-zst | --no-repodata-use-zst]
                      [--repodata-use-shards | --no-repodata-use-shards]
                      [--strict-channel-priority] [--no-channel-priority]
                      [--no-deps] [--no-pin]
                      [--exclude-newer DURATION_OR_DATE] [--copy]
                      [--no-shortcuts] [--shortcuts-only SHORTCUTS_ONLY]
                      [--clobber] [-C] [-k] [--offline] [--json]
                      [--console {classic,json}] [-v] [-q] [-d] [-y]
                      [--download-only] [--show-channel-urls] [-f FILE]
                      [--environment-specifier {cep-24,environment-yaml,env.yml,conda-
↪lock-v1,conda-lock,environment.yml,explicit,rattler-lock-v6,pixi,pixi-lock-v6,
↪requirements.txt,requirements,reqs}]
                      [--format FORMAT]
                      [--solver {classic,libmamba,pycosat,rattler}]
                      [--force-reinstall] [--freeze-installed |
                      --update-deps | -S | --update-specs]
                      [package_spec ...]
```

Positional Arguments

package_spec List of packages to install or update in the conda environment.

Named Arguments

--override-frozen DANGEROUS. Use at your own risk. Ignore protections if the environment is frozen.

-f, --file Read environment or package specs from a file. The format is detected from the filename or contents. Which formats are supported depends on the installed plugins (see the epilog for the list available here). Custom filenames require --format. May be repeated (e.g. --file=file1 --file=file2).

--environment-specifier, --env-spec Possible choices: cep-24, environment-yaml, env.yml, conda-lock-v1, conda-lock, environment.yml, explicit, rattler-lock-v6, pixi, pixi-lock-v6, requirements.txt, requirements, reqs

--env-spec is deprecated and will be removed in 27.3. Use the *--format* flag instead.

--format Possible choices: cep-24, environment-yaml, env.yml, conda-lock-v1, conda-lock, environment.yml, explicit, rattler-lock-v6, pixi, pixi-lock-v6, requirements.txt, requirements, reqs

Override auto-detection of the input file's format. See *conda export --help* for the formats available in your installation. Aliases are interchangeable with canonical names.

Target Environment Specification

-n, --name Name of environment.

-p, --prefix Full path to environment location (i.e. prefix).

Channel Customization

-c, --channel Additional channel to search for packages. These are URLs searched in the order they are given (including local directories using the 'file://' syntax or simply a path like '/home/conda/mychan' or '../mychan'). Then, the defaults or channels from .condarc are searched (unless --override-channels is given). You can use 'defaults' to get the default packages for conda. You can also use any name and the .condarc channel_alias value will be prepended. The default channel_alias is https://conda.anaconda.org/.

--use-local Use locally built packages. Identical to '-c local'.

-O, --override-channels Do not search default or .condarc channels. Requires --channel.

--repodata-fn Specify file name of repodata on the remote server where your channels are configured or within local backups. Conda will try whatever you specify, but will ultimately fall back to repodata.json if your specs are not satisfiable with what you specify here. This is used to employ repodata that is smaller and reduced in time scope. You may pass this flag more than once. Leftmost entries are tried first, and the fallback to repodata.json is added for you automatically. For more information, see conda config --describe repodata_fns.

--experimental Possible choices: jlap, lock

--experimental is deprecated and will be removed in 27.3. Deprecated: jlap and lock no longer supported.

--no-lock Disable locking when reading, updating index (repodata.json) cache.

--repodata-use-zst, --no-repodata-use-zst Check for/do not check for repodata.json.zst. Enabled by default.

--repodata-use-shards, --no-repodata-use-shards Use sharded repodata if available. Enabled by default.

Solver Mode Modifiers

--strict-channel-priority Packages in lower priority channels are not considered if a package with the same name appears in a higher priority channel.

--no-channel-priority Package version takes precedence over channel priority. Overrides the value given by *conda config --show channel_priority*.

--no-deps Do not install, update, remove, or change dependencies. This WILL lead to broken environments and inconsistent behavior. Use at your own risk.

--no-pin Ignore pinned file.

--exclude-newer Exclude packages published more recently than the given duration (e.g. 7d, 3d12h, 1w) or date (e.g. 2026-04-01, 2026-04-01T12:00:00Z). Date-only values use the start of the next UTC day. Supply 0 for no delay, using the current time as the cutoff. Channel and per-package overrides can be set via channel_settings and exclude_newer_package in .condarc.

--solver Possible choices: classic, libmamba, pycosat, rattler

Choose which solver backend to use.

--force-reinstall Ensure that any requested plugin package is uninstalled and reinstalled, even if it already exists in the environment.

--freeze-installed, --no-update-deps Do not update or change already-installed dependencies.

--update-deps Update dependencies that have available updates.

-S, --satisfied-skip-solve Exit early and do not run the solver if the requested specs are satisfied. Also skips aggressive updates as configured by the 'aggressive_update_packages' config setting. Use 'conda config --describe aggressive_update_packages' to view your setting. --satisfied-skip-solve is similar to the default behavior of 'pip install'.

--update-specs Update based on provided specifications.

Package Linking and Install-time Options

--copy Install all packages using copies instead of hard- or soft-linking.

--no-shortcuts Don't install start menu shortcuts

--shortcuts-only Install shortcuts only for this package name. Can be used several times.

--clobber Allow clobbering (i.e. overwriting) of overlapping file paths within packages and suppress related warnings.

Networking Options

-C, --use-index-cache Use cache of channel index files, even if it has expired. This is useful if you don't want conda to check whether a new version of the repodata file exists, which will save bandwidth.

-k, --insecure Allow conda to perform "insecure" SSL connections and transfers. Equivalent to setting 'ssl_verify' to 'false'.

--offline Offline mode. Don't connect to the Internet.

Output, Prompt, and Flow Control Options

--json Report all output as json. Suitable for using conda programmatically.

--console Possible choices: classic, json

Select the backend to use for normal output rendering.

-v, --verbose Can be used multiple times. Once for detailed output, twice for INFO logging, thrice for DEBUG logging, four times for TRACE logging.

-q, --quiet Do not display progress bar.

-d, --dry-run Only display what would have been done.

-y, --yes Sets any confirmation values to 'yes' automatically. Users will not be asked to confirm any adding, deleting, backups, etc.

--download-only Solve an environment and ensure package caches are populated, but exit prior to unlinking and linking packages into the prefix.

--show-channel-urls Show channel urls. Overrides the value given by *conda config --show show_channel_urls*.

list

List installed conda plugins.

```
conda plugins list [-h] [--json] [--console {classic,json}] [-v] [-q]
```

Output, Prompt, and Flow Control Options

--json Report all output as json. Suitable for using conda programmatically.

--console Possible choices: classic, json

Select the backend to use for normal output rendering.

-v, --verbose Can be used multiple times. Once for detailed output, twice for INFO logging, thrice for DEBUG logging, four times for TRACE logging.

-q, --quiet Do not display progress bar.

remove

Remove conda plugin packages from an environment.

```
conda plugins remove [-h] [--override-frozen] [-n ENVIRONMENT | -p PATH]
                     [-c CHANNEL] [--use-local] [-0]
                     [--repodata-fn REPODATA_FNS] [--experimental {jlap,lock}]
                     [--no-lock] [--repodata-use-zst | --no-repodata-use-zst]
                     [--repodata-use-shards | --no-repodata-use-shards]
                     [--force-remove] [--no-pin]
                     [--solver {classic,libmamba,pycosat,rattler}] [-C] [-k]
                     [--offline] [--json] [--console {classic,json}] [-v] [-q]
                     [-d] [-y]
                     [package_name ...]
```

Positional Arguments

package_name Plugin package names to remove from the environment.

Named Arguments

--override-frozen DANGEROUS. Use at your own risk. Ignore protections if the environment is frozen.

Target Environment Specification

-n, --name Name of environment.

-p, --prefix Full path to environment location (i.e. prefix).

Channel Customization

-c, --channel Additional channel to search for packages. These are URLs searched in the order they are given (including local directories using the 'file://' syntax or simply a path like '/home/conda/mychan' or '../mychan'). Then, the defaults or channels from .condarc are searched (unless --override-channels is given). You can use 'defaults' to get the default packages for conda. You can also use any name and the .condarc channel_alias value will be prepended. The default channel_alias is https://conda.anaconda.org/.

--use-local Use locally built packages. Identical to '-c local'.

-O, --override-channels Do not search default or .condarc channels. Requires --channel.

--repodata-fn Specify file name of repodata on the remote server where your channels are configured or within local backups. Conda will try whatever you specify, but will ultimately fall back to repodata.json if your specs are not satisfiable with what you specify here. This is used to employ repodata that is smaller and reduced in time scope. You may pass this flag more than once. Leftmost entries are tried first, and the fallback to repodata.json is added for you automatically. For more information, see conda config --describe repodata_fns.

--experimental Possible choices: jlap, lock

--experimental is deprecated and will be removed in 27.3. Deprecated: jlap and lock no longer supported.

--no-lock Disable locking when reading, updating index (repodata.json) cache.

--repodata-use-zst, --no-repodata-use-zst Check for/do not check for repodata.json.zst. Enabled by default.

--repodata-use-shards, --no-repodata-use-shards Use sharded repodata if available. Enabled by default.

Solver Mode Modifiers

--force-remove, --force Remove a plugin package without removing packages that depend on it. Using this option will usually leave your environment broken.

--no-pin Ignore pinned package(s) that apply to the current operation.

--solver Possible choices: classic, libmamba, pycosat, rattler

Choose which solver backend to use.

Networking Options

-C, --use-index-cache Use cache of channel index files, even if it has expired. This is useful if you don't want conda to check whether a new version of the repodata file exists, which will save bandwidth.

-k, --insecure Allow conda to perform "insecure" SSL connections and transfers. Equivalent to setting 'ssl_verify' to 'false'.

--offline Offline mode. Don't connect to the Internet.

Output, Prompt, and Flow Control Options

--json Report all output as json. Suitable for using conda programmatically.

--console Possible choices: classic, json

Select the backend to use for normal output rendering.

-v, --verbose Can be used multiple times. Once for detailed output, twice for INFO logging, thrice for DEBUG logging, four times for TRACE logging.

-q, --quiet Do not display progress bar.

-d, --dry-run Only display what would have been done.

-y, --yes Sets any confirmation values to 'yes' automatically. Users will not be asked to confirm any adding, deleting, backups, etc.

update

Update conda plugin packages in an environment.

```
conda plugins update [-h] [--override-frozen] [-n ENVIRONMENT | -p PATH]
                     [-c CHANNEL] [--use-local] [-O]
                     [--repodata-fn REPODATA_FNS] [--experimental {jlap,lock}]
                     [--no-lock] [--repodata-use-zst | --no-repodata-use-zst]
                     [--repodata-use-shards | --no-repodata-use-shards]
                     [--strict-channel-priority] [--no-channel-priority]
                     [--no-deps] [--no-pin] [--exclude-newer DURATION_OR_DATE]
                     [--copy] [--no-shortcuts]
                     [--shortcuts-only SHORTCUTS_ONLY] [--clobber] [-C] [-k]
                     [--offline] [--json] [--console {classic,json}] [-v] [-q]
                     [-d] [-y] [--download-only] [--show-channel-urls]
                     [-f FILE]
                     [--environment-specifier {cep-24,environment-yaml,env.yml,conda-
↪lock-v1,conda-lock,environment.yml,explicit,rattler-lock-v6,pixi,pixi-lock-v6,
↪requirements.txt,requirements,reqs}]
                     [--format FORMAT]
                     [--solver {classic,libmamba,pycosat,rattler}]
                     [--force-reinstall] [--freeze-installed | --update-deps |
                     -S | --update-specs] [--update-all]
                     [package_spec ...]
```

Positional Arguments

package_spec List of packages to install or update in the conda environment.

Named Arguments

--override-frozen DANGEROUS. Use at your own risk. Ignore protections if the environment is frozen.

-f, --file Read environment or package specs from a file. The format is detected from the filename or contents. Which formats are supported depends on the installed plugins (see the epilog for the list available here). Custom filenames require --format. May be repeated (e.g. --file=file1 --file=file2).

--environment-specifier, --env-spec Possible choices: cep-24, environment-yaml, env.yml, conda-lock-v1, conda-lock, environment.yml, explicit, rattler-lock-v6, pixi, pixi-lock-v6, requirements.txt, requirements, reqs

--env-spec is deprecated and will be removed in 27.3. Use the *--format* flag instead.

--format Possible choices: cep-24, environment-yaml, env.yml, conda-lock-v1, conda-lock, environment.yml, explicit, rattler-lock-v6, pixi, pixi-lock-v6, requirements.txt, requirements, reqs

Override auto-detection of the input file's format. See *conda export --help* for the formats available in your installation. Aliases are interchangeable with canonical names.

Target Environment Specification

-n, --name Name of environment.

-p, --prefix Full path to environment location (i.e. prefix).

Channel Customization

-c, --channel Additional channel to search for packages. These are URLs searched in the order they are given (including local directories using the 'file://' syntax or simply a path like '/home/conda/mychan' or '../mychan'). Then, the defaults or channels from .condarc are searched (unless --override-channels is given). You can use 'defaults' to get the default packages for conda. You can also use any name and the .condarc channel_alias value will be prepended. The default channel_alias is https://conda.anaconda.org/.

--use-local Use locally built packages. Identical to '-c local'.

-O, --override-channels Do not search default or .condarc channels. Requires --channel.

--repodata-fn Specify file name of repodata on the remote server where your channels are configured or within local backups. Conda will try whatever you specify, but will ultimately fall back to repodata.json if your specs are not satisfiable with what you specify here. This is used to employ repodata that is smaller and reduced in time scope. You may pass this flag more than once. Leftmost entries are tried first, and the fallback to repodata.json is added for you automatically. For more information, see conda config --describe repodata_fns.

--experimental Possible choices: jlap, lock

--experimental is deprecated and will be removed in 27.3. Deprecated: jlap and lock no longer supported.

--no-lock Disable locking when reading, updating index (repodata.json) cache.

--repodata-use-zst, --no-repodata-use-zst Check for/do not check for repodata.json.zst. Enabled by default.

--repodata-use-shards, --no-repodata-use-shards Use sharded repodata if available. Enabled by default.

Solver Mode Modifiers

--strict-channel-priority Packages in lower priority channels are not considered if a package with the same name appears in a higher priority channel.

--no-channel-priority Package version takes precedence over channel priority. Overrides the value given by *conda config --show channel_priority*.

--no-deps Do not install, update, remove, or change dependencies. This WILL lead to broken environments and inconsistent behavior. Use at your own risk.

--no-pin Ignore pinned file.

--exclude-newer Exclude packages published more recently than the given duration (e.g. 7d, 3d12h, 1w) or date (e.g. 2026-04-01, 2026-04-01T12:00:00Z). Date-only values use the start of the next UTC day. Supply 0 for no delay, using the current time as the cutoff. Channel and per-package overrides can be set via channel_settings and exclude_newer_package in .condarc.

--solver Possible choices: classic, libmamba, pycosat, rattler

Choose which solver backend to use.

--force-reinstall Ensure that any requested plugin package is uninstalled and reinstalled, even if it already exists in the environment.

--freeze-installed, --no-update-deps Do not update or change already-installed dependencies.

--update-deps Update dependencies that have available updates.

-S, --satisfied-skip-solve Exit early and do not run the solver if the requested specs are satisfied. Also skips aggressive updates as configured by the 'aggressive_update_packages' config setting. Use 'conda config --describe aggressive_update_packages' to view your setting. --satisfied-skip-solve is similar to the default behavior of 'pip install'.

--update-specs Update based on provided specifications.

--update-all, --all Update all installed conda plugin packages without updating other packages directly.

Package Linking and Install-time Options

--copy Install all packages using copies instead of hard- or soft-linking.

--no-shortcuts Don't install start menu shortcuts

--shortcuts-only Install shortcuts only for this package name. Can be used several times.

--clobber Allow clobbering (i.e. overwriting) of overlapping file paths within packages and suppress related warnings.

Networking Options

-C, --use-index-cache Use cache of channel index files, even if it has expired. This is useful if you don't want conda to check whether a new version of the repodata file exists, which will save bandwidth.

-k, --insecure Allow conda to perform "insecure" SSL connections and transfers. Equivalent to setting 'ssl_verify' to 'false'.

--offline Offline mode. Don't connect to the Internet.

Output, Prompt, and Flow Control Options

--json Report all output as json. Suitable for using conda programmatically.

--console Possible choices: classic, json

Select the backend to use for normal output rendering.

-v, --verbose Can be used multiple times. Once for detailed output, twice for INFO logging, thrice for DEBUG logging, four times for TRACE logging.

-q, --quiet Do not display progress bar.

-d, --dry-run Only display what would have been done.

-y, --yes Sets any confirmation values to 'yes' automatically. Users will not be asked to confirm any adding, deleting, backups, etc.

--download-only Solve an environment and ensure package caches are populated, but exit prior to unlinking and linking packages into the prefix.

--show-channel-urls Show channel urls. Overrides the value given by *conda config --show show_channel_urls*.

4.4.17 `conda repoquery`

Note

The `conda repoquery` command is provided by the `conda-libmamba-solver`. For more information on the subcommands provided, see Subcommands.

For more information on the libmamba solver, see *Solvers*.

Advanced search for repodata.

```
usage: conda repoquery [-h] {whoneeds,depends,search} ...
```

Positional Arguments

subcmd Possible choices: whoneeds, depends, search

conda repoquery depends

> **Note**
>
> The `conda repoquery depends` command is provided by the `conda-libmamba-solver`. For more information on the subcommands provided, see Subcommands.
>
> For more information on the libmamba solver, see *Solvers*.

```
usage: conda repoquery depends [-p PLATFORM] [--no-installed] [--pretty] [-a]
                               [--use-cache-only] [-t] [--recursive] [-h]
                               [-c CHANNEL] [--use-local] [-O]
                               [--repodata-fn REPODATA_FNS]
                               [--experimental {jlap,lock}] [--no-lock]
                               [--repodata-use-zst | --no-repodata-use-zst]
                               [--repodata-use-shards | --no-repodata-use-shards]
                               [-C] [-k] [--offline] [--json]
                               [--console {classic,json}] [-v] [-q]
                               specs [specs ...]
```

Positional Arguments

specs The target package(s).

Subcommand options

-p, --platform Platform/subdir to search packages for. Defaults to current platform.

--no-installed Do not search currently installed packages.

--pretty Prettier output with more details.

-a, --all-channels Look at all channels (for depends / whoneeds).

--use-cache-only Search in pkgs_dirs too

Dependency options

-t, --tree Show dependencies in a tree-like format.

--recursive Show dependencies recursively.

Channel Customization

-c, --channel Additional channel to search for packages. These are URLs searched in the order they are given (including local directories using the 'file://' syntax or simply a path like '/home/conda/mychan' or '../mychan'). Then, the defaults or channels from .condarc are searched (unless --override-channels is given). You can use 'defaults' to get the default packages for conda. You can also use any name and the .condarc channel_alias value will be prepended. The default channel_alias is https://conda.anaconda.org/.

--use-local Use locally built packages. Identical to '-c local'.

-O, --override-channels Do not search default or .condarc channels. Requires --channel.

--repodata-fn Specify file name of repodata on the remote server where your channels are configured or within local backups. Conda will try whatever you specify, but will ultimately fall back to repodata.json if your specs are not satisfiable with what you specify here. This is used to employ repodata that is smaller and reduced in time scope. You may pass this flag more than once. Leftmost entries are tried first, and the fallback to repodata.json is added for you automatically. For more information, see conda config --describe repodata_fns.

--experimental Possible choices: jlap, lock

--experimental is deprecated and will be removed in 27.3. Deprecated: jlap and lock no longer supported.

--no-lock Disable locking when reading, updating index (repodata.json) cache.

--repodata-use-zst, --no-repodata-use-zst Check for/do not check for repodata.json.zst. Enabled by default.

--repodata-use-shards, --no-repodata-use-shards Use sharded repodata if available. Enabled by default.

Networking Options

-C, --use-index-cache Use cache of channel index files, even if it has expired. This is useful if you don't want conda to check whether a new version of the repodata file exists, which will save bandwidth.

-k, --insecure Allow conda to perform "insecure" SSL connections and transfers. Equivalent to setting 'ssl_verify' to 'false'.

--offline Offline mode. Don't connect to the Internet.

Output, Prompt, and Flow Control Options

--json Report all output as json. Suitable for using conda programmatically.

--console Possible choices: classic, json

Select the backend to use for normal output rendering.

-v, --verbose Can be used multiple times. Once for detailed output, twice for INFO logging, thrice for DEBUG logging, four times for TRACE logging.

-q, --quiet Do not display progress bar.

`conda repoquery search`

Note

The `conda repoquery search` command is provided by the `conda-libmamba-solver`. For more information on the subcommands provided, see Subcommands.

For more information on the libmamba solver, see *Solvers*.

```
usage: conda repoquery search [-p PLATFORM] [--no-installed] [--pretty] [-a]
                              [--use-cache-only] [-h] [-c CHANNEL]
                              [--use-local] [-O] [--repodata-fn REPODATA_FNS]
                              [--experimental {jlap,lock}] [--no-lock]
```

(continues on next page)

(continued from previous page)

```
                                [--repodata-use-zst | --no-repodata-use-zst]
                                [--repodata-use-shards | --no-repodata-use-shards]
                                [-C] [-k] [--offline] [--json]
                                [--console {classic,json}] [-v] [-q]
                                specs [specs ...]
```

Positional Arguments

specs The target package(s).

Subcommand options

-p, --platform Platform/subdir to search packages for. Defaults to current platform.

--no-installed Do not search currently installed packages.

--pretty Prettier output with more details.

-a, --all-channels Look at all channels (for depends / whoneeds).

--use-cache-only Search in pkgs_dirs too

Channel Customization

-c, --channel Additional channel to search for packages. These are URLs searched in the order they are given (including local directories using the 'file://' syntax or simply a path like '/home/conda/mychan' or '../mychan'). Then, the defaults or channels from .condarc are searched (unless --override-channels is given). You can use 'defaults' to get the default packages for conda. You can also use any name and the .condarc channel_alias value will be prepended. The default channel_alias is https://conda.anaconda.org/.

--use-local Use locally built packages. Identical to '-c local'.

-O, --override-channels Do not search default or .condarc channels. Requires --channel.

--repodata-fn Specify file name of repodata on the remote server where your channels are configured or within local backups. Conda will try whatever you specify, but will ultimately fall back to repodata.json if your specs are not satisfiable with what you specify here. This is used to employ repodata that is smaller and reduced in time scope. You may pass this flag more than once. Leftmost entries are tried first, and the fallback to repodata.json is added for you automatically. For more information, see conda config --describe repodata_fns.

--experimental Possible choices: jlap, lock

--experimental is deprecated and will be removed in 27.3. Deprecated: jlap and lock no longer supported.

--no-lock Disable locking when reading, updating index (repodata.json) cache.

--repodata-use-zst, --no-repodata-use-zst Check for/do not check for repodata.json.zst. Enabled by default.

--repodata-use-shards, --no-repodata-use-shards Use sharded repodata if available. Enabled by default.

Networking Options

-C, --use-index-cache Use cache of channel index files, even if it has expired. This is useful if you don't want conda to check whether a new version of the repodata file exists, which will save bandwidth.

-k, --insecure Allow conda to perform "insecure" SSL connections and transfers. Equivalent to setting 'ssl_verify' to 'false'.

--offline Offline mode. Don't connect to the Internet.

Output, Prompt, and Flow Control Options

--json Report all output as json. Suitable for using conda programmatically.

--console Possible choices: classic, json

Select the backend to use for normal output rendering.

-v, --verbose Can be used multiple times. Once for detailed output, twice for INFO logging, thrice for DEBUG logging, four times for TRACE logging.

-q, --quiet Do not display progress bar.

`conda repoquery whoneeds`

> **Note**
>
> The `conda repoquery whoneeds` command is provided by the `conda-libmamba-solver`. For more information on the subcommands provided, see Subcommands.
>
> For more information on the libmamba solver, see *Solvers*.

```
usage: conda repoquery whoneeds [-p PLATFORM] [--no-installed] [--pretty] [-a]
                                [--use-cache-only] [-t] [--recursive] [-h]
                                [-c CHANNEL] [--use-local] [-0]
                                [--repodata-fn REPODATA_FNS]
                                [--experimental {jlap,lock}] [--no-lock]
                                [--repodata-use-zst | --no-repodata-use-zst]
                                [--repodata-use-shards | --no-repodata-use-shards]
                                [-C] [-k] [--offline] [--json]
                                [--console {classic,json}] [-v] [-q]
                                specs [specs ...]
```

Positional Arguments

specs The target package(s).

Subcommand options

-p, --platform Platform/subdir to search packages for. Defaults to current platform.

--no-installed Do not search currently installed packages.

--pretty Prettier output with more details.

-a, --all-channels Look at all channels (for depends / whoneeds).

--use-cache-only Search in pkgs_dirs too

Dependency options

-t, --tree Show dependencies in a tree-like format.

--recursive Show dependencies recursively.

Channel Customization

-c, --channel Additional channel to search for packages. These are URLs searched in the order they are given (including local directories using the 'file://' syntax or simply a path like '/home/conda/mychan' or '../mychan'). Then, the defaults or channels from .condarc are searched (unless --override-channels is given). You can use 'defaults' to get the default packages for conda. You can also use any name and the .condarc channel_alias value will be prepended. The default channel_alias is https://conda.anaconda.org/.

--use-local Use locally built packages. Identical to '-c local'.

-O, --override-channels Do not search default or .condarc channels. Requires --channel.

--repodata-fn Specify file name of repodata on the remote server where your channels are configured or within local backups. Conda will try whatever you specify, but will ultimately fall back to repodata.json if your specs are not satisfiable with what you specify here. This is used to employ repodata that is smaller and reduced in time scope. You may pass this flag more than once. Leftmost entries are tried first, and the fallback to repodata.json is added for you automatically. For more information, see conda config --describe repodata_fns.

--experimental Possible choices: jlap, lock

--experimental is deprecated and will be removed in 27.3. Deprecated: jlap and lock no longer supported.

--no-lock Disable locking when reading, updating index (repodata.json) cache.

--repodata-use-zst, --no-repodata-use-zst Check for/do not check for repodata.json.zst. Enabled by default.

--repodata-use-shards, --no-repodata-use-shards Use sharded repodata if available. Enabled by default.

Networking Options

-C, --use-index-cache Use cache of channel index files, even if it has expired. This is useful if you don't want conda to check whether a new version of the repodata file exists, which will save bandwidth.

-k, --insecure Allow conda to perform "insecure" SSL connections and transfers. Equivalent to setting 'ssl_verify' to 'false'.

--offline Offline mode. Don't connect to the Internet.

Output, Prompt, and Flow Control Options

--json Report all output as json. Suitable for using conda programmatically.

--console Possible choices: classic, json

Select the backend to use for normal output rendering.

-v, --verbose Can be used multiple times. Once for detailed output, twice for INFO logging, thrice for DEBUG logging, four times for TRACE logging.

-q, --quiet Do not display progress bar.

4.4.18 conda remove

Remove a list of packages from a specified conda environment.

Use *--all* flag to remove all packages and the environment itself.

This command will also remove any package that depends on any of the specified packages as well---unless a replacement can be found without that dependency. If you wish to skip this dependency checking and remove just the requested packages, add the '--force' option. Note however that this may result in a broken environment, so use this with caution.

```
usage: conda remove [-h] [--override-frozen] [-n ENVIRONMENT | -p PATH]
                    [-c CHANNEL] [--use-local] [-0]
                    [--repodata-fn REPODATA_FNS] [--experimental {jlap,lock}]
                    [--no-lock] [--repodata-use-zst | --no-repodata-use-zst]
                    [--repodata-use-shards | --no-repodata-use-shards]
                    [--features] [--force-remove] [--no-pin]
                    [--solver {classic,libmamba,pycosat,rattler}] [-C] [-k]
                    [--offline] [--json] [--console {classic,json}] [-v] [-q]
                    [-d] [-y] [--all] [--keep-env] [--dev]
                    [package_name ...]
```

Positional Arguments

package_name Package names to remove from the environment.

Named Arguments

--override-frozen DANGEROUS. Use at your own risk. Ignore protections if the environment is frozen.

--all Remove all packages, i.e., the entire environment.

--keep-env Used with *--all*, delete all packages but keep the environment.

--dev *--dev* is pending deprecation and will be removed in 27.9. Set *PYTHONPATH* to the conda source root instead.

Target Environment Specification

-n, --name Name of environment.

-p, --prefix Full path to environment location (i.e. prefix).

Channel Customization

-c, --channel Additional channel to search for packages. These are URLs searched in the order they are given (including local directories using the 'file://' syntax or simply a path like '/home/conda/mychan' or '../mychan'). Then, the defaults or channels from .condarc are searched (unless --override-channels is given). You can use 'defaults' to get the default packages for conda. You can also use any name and the .condarc channel_alias value will be prepended. The default channel_alias is https://conda.anaconda.org/.

--use-local Use locally built packages. Identical to '-c local'.

-O, --override-channels Do not search default or .condarc channels. Requires --channel.

--repodata-fn Specify file name of repodata on the remote server where your channels are configured or within local backups. Conda will try whatever you specify, but will ultimately fall back to repodata.json if your specs are not satisfiable with what you specify here. This is used to employ repodata that is smaller and reduced in time scope. You may pass this flag more than once. Leftmost entries are tried first, and the fallback to repodata.json is added for you automatically. For more information, see conda config --describe repodata_fns.

--experimental Possible choices: jlap, lock

--experimental is deprecated and will be removed in 27.3. Deprecated: jlap and lock no longer supported.

--no-lock Disable locking when reading, updating index (repodata.json) cache.

--repodata-use-zst, --no-repodata-use-zst Check for/do not check for repodata.json.zst. Enabled by default.

--repodata-use-shards, --no-repodata-use-shards Use sharded repodata if available. Enabled by default.

Solver Mode Modifiers

--features Remove features (instead of packages).

--force-remove, --force Forces removal of a package without removing packages that depend on it. Using this option will usually leave your environment in a broken and inconsistent state.

--no-pin Ignore pinned package(s) that apply to the current operation. These pinned packages might come from a .condarc file or a file in <TARGET_ENVIRONMENT>/conda-meta/pinned.

--solver Possible choices: classic, libmamba, pycosat, rattler

Choose which solver backend to use.

Networking Options

-C, --use-index-cache Use cache of channel index files, even if it has expired. This is useful if you don't want conda to check whether a new version of the repodata file exists, which will save bandwidth.

-k, --insecure Allow conda to perform "insecure" SSL connections and transfers. Equivalent to setting 'ssl_verify' to 'false'.

--offline Offline mode. Don't connect to the Internet.

Output, Prompt, and Flow Control Options

--json Report all output as json. Suitable for using conda programmatically.

--console Possible choices: classic, json

Select the backend to use for normal output rendering.

-v, --verbose Can be used multiple times. Once for detailed output, twice for INFO logging, thrice for DEBUG logging, four times for TRACE logging.

-q, --quiet	Do not display progress bar.
-d, --dry-run	Only display what would have been done.
-y, --yes	Sets any confirmation values to 'yes' automatically. Users will not be asked to confirm any adding, deleting, backups, etc.

Examples:

Remove the package 'scipy' from the currently-active environment:

```
conda remove scipy
```

Remove a list of packages from an environment 'myenv':

```
conda remove -n myenv scipy curl wheel
```

Remove all packages from environment *myenv* and the environment itself:

```
conda remove -n myenv --all
```

Remove all packages from the environment *myenv* but retain the environment:

```
conda remove -n myenv --all --keep-env
```

4.4.19 `conda rename`

Rename an existing environment.

This command renames a conda environment via its name (-n/--name) or its prefix (-p/--prefix).

The base environment and the currently-active environment cannot be renamed.

```
usage: conda rename [-h] [-n ENVIRONMENT | -p PATH] [--json]
                    [--console {classic,json}] [-v] [-q] [-d] [-y]
                    destination
```

Positional Arguments

destination	New name for the conda environment.

Target Environment Specification

-n, --name	Name of environment.
-p, --prefix	Full path to environment location (i.e. prefix).

Output, Prompt, and Flow Control Options

--json	Report all output as json. Suitable for using conda programmatically.
--console	Possible choices: classic, json Select the backend to use for normal output rendering.
-v, --verbose	Can be used multiple times. Once for detailed output, twice for INFO logging, thrice for DEBUG logging, four times for TRACE logging.
-q, --quiet	Do not display progress bar.
-d, --dry-run	Only display what would have been done.

-y, --yes Sets any confirmation values to 'yes' automatically. Users will not be asked to confirm any adding, deleting, backups, etc.

Examples:

conda rename -n test123 test321

conda rename --name test123 test321

conda rename -p path/to/test123 test321

conda rename --prefix path/to/test123 test321

4.4.20 conda run

Run an executable in a conda environment.

```
usage: conda run [-h] [-n ENVIRONMENT | -p PATH] [-v] [--dev]
                 [--debug-wrapper-scripts] [--cwd CWD] [-s]
                 ...
```

Positional Arguments

executable_call Executable name, with additional arguments to be passed to the executable on invocation. Use '--' to separate conda options from executable options.

Named Arguments

-v, --verbose Can be used multiple times. Once for detailed output, twice for INFO logging, thrice for DEBUG logging, four times for TRACE logging.

--dev *--dev* is deprecated and will be removed in 27.3. Set *PYTHONPATH* to the conda source root instead.

--debug-wrapper-scripts When this is set, where implemented, the shell wrapper scriptswill use the echo command to print debugging information to stderr (standard error).

--cwd Current working directory for command to run in. Defaults to the user's current working directory if no directory is specified.

-s, --no-capture-output, --live-stream Don't capture stdout/stderr (standard out/standard error).

Target Environment Specification

-n, --name Name of environment.

-p, --prefix Full path to environment location (i.e. prefix).

Example:

$ conda create -y -n my-python-env python=3 $ conda run -n my-python-env python --version

Use '--' to separate *conda run*'s options from the executable's options:

$ conda run -n my-python-env -- python -v $ conda run -v -n my-python-env -- tar -tvf file.tar

4.4.21 conda search

Search for packages and display associated information using the MatchSpec format.

MatchSpec is a query language for conda packages.

```
usage: conda search [-h] [--envs] [-i] [--subdir SUBDIR]
                    [--skip-flexible-search] [-c CHANNEL] [--use-local] [-O]
                    [--repodata-fn REPODATA_FNS] [--experimental {jlap,lock}]
                    [--no-lock] [--repodata-use-zst | --no-repodata-use-zst]
                    [--repodata-use-shards | --no-repodata-use-shards] [-C]
                    [-k] [--offline] [--json] [--console {classic,json}] [-v]
                    [-q]
```

Named Arguments

--envs Search all of the current user's environments. If run as Administrator (on Windows) or UID 0 (on unix), search all known environments on the system.

-i, --info Provide detailed information about each package.

--subdir, --platform Specify which platform to search. Should be formatted like 'osx-64', 'linux-32', 'win-64', and so on. Defaults to the current (native) platform.

--skip-flexible-search Do not perform flexible search if initial search fails.

Channel Customization

-c, --channel Additional channel to search for packages. These are URLs searched in the order they are given (including local directories using the 'file://' syntax or simply a path like '/home/conda/mychan' or '../mychan'). Then, the defaults or channels from .condarc are searched (unless --override-channels is given). You can use 'defaults' to get the default packages for conda. You can also use any name and the .condarc channel_alias value will be prepended. The default channel_alias is https://conda.anaconda.org/.

--use-local Use locally built packages. Identical to '-c local'.

-O, --override-channels Do not search default or .condarc channels. Requires --channel.

--repodata-fn Specify file name of repodata on the remote server where your channels are configured or within local backups. Conda will try whatever you specify, but will ultimately fall back to repodata.json if your specs are not satisfiable with what you specify here. This is used to employ repodata that is smaller and reduced in time scope. You may pass this flag more than once. Leftmost entries are tried first, and the fallback to repodata.json is added for you automatically. For more information, see conda config --describe repodata_fns.

--experimental Possible choices: jlap, lock

--experimental is deprecated and will be removed in 27.3. Deprecated: jlap and lock no longer supported.

--no-lock Disable locking when reading, updating index (repodata.json) cache.

--repodata-use-zst, --no-repodata-use-zst Check for/do not check for repodata.json.zst. Enabled by default.

--repodata-use-shards, --no-repodata-use-shards Use sharded repodata if available. Enabled by default.

Networking Options

-C, --use-index-cache Use cache of channel index files, even if it has expired. This is useful if you don't want conda to check whether a new version of the repodata file exists, which will save bandwidth.

-k, --insecure Allow conda to perform "insecure" SSL connections and transfers. Equivalent to setting 'ssl_verify' to 'false'.

--offline Offline mode. Don't connect to the Internet.

Output, Prompt, and Flow Control Options

--json Report all output as json. Suitable for using conda programmatically.

--console Possible choices: classic, json

Select the backend to use for normal output rendering.

-v, --verbose Can be used multiple times. Once for detailed output, twice for INFO logging, thrice for DEBUG logging, four times for TRACE logging.

-q, --quiet Do not display progress bar.

Examples:

Search for a specific package named 'scikit-learn':

conda search scikit-learn

Search for packages containing 'scikit' in the package name:

conda search *scikit*

Note that your shell may expand '*' before handing the command over to conda. Therefore, it is sometimes necessary to use single or double quotes around the query:

conda search '*scikit' conda search "*scikit*"

Search for packages for 64-bit Linux (by default, packages for your current platform are shown):

conda search numpy[subdir=linux-64]

Search for a specific version of a package:

conda search 'numpy>=1.12'

Search for a package on a specific channel:

conda search conda-forge::numpy

Search for a package on a specific channel and platform:

conda search 'numpy[channel=conda-forge, subdir=osx-64]'

Search for a package with a specific software version and python version:

conda search 'numpy-base=2.4.2=py313*' conda search 'numpy-base[version="2.4.2",build=py313*]'

Search for a package with a specific software version and build string:

conda search 'r-shiny=1.12.0=r45*' conda search 'r-shiny[version="1.12.0",build=r45*]'

4.4.22 conda update

Update conda packages to the latest compatible version.

This command accepts a list of package names and updates them to the latest versions that are compatible with all other packages in the environment.

Conda attempts to install the newest versions of the requested packages. To accomplish this, it may update some packages that are already installed, or install additional packages. To prevent existing packages from updating, use the --no-update-deps option. This may force conda to install older versions of the requested packages, and it does not prevent additional dependency packages from being installed.

When using --file, only the package list from the file is used. Any name or prefix in the file is ignored; packages are updated in the target environment (-n/-p or the current environment).

```
usage: conda update [-h] [--override-frozen] [-n ENVIRONMENT | -p PATH]
                    [-c CHANNEL] [--use-local] [-O]
                    [--repodata-fn REPODATA_FNS] [--experimental {jlap,lock}]
                    [--no-lock] [--repodata-use-zst | --no-repodata-use-zst]
                    [--repodata-use-shards | --no-repodata-use-shards]
                    [--strict-channel-priority] [--no-channel-priority]
                    [--no-deps | --only-deps] [--no-pin]
                    [--exclude-newer DURATION_OR_DATE] [--copy]
                    [--no-shortcuts] [--shortcuts-only SHORTCUTS_ONLY]
                    [--clobber] [-C] [-k] [--offline] [--json]
                    [--console {classic,json}] [-v] [-q] [-d] [-y]
                    [--download-only] [--show-channel-urls] [-f FILE]
                    [--environment-specifier {cep-24,environment-yaml,env.yml,conda-lock-
↪v1,conda-lock,environment.yml,explicit,rattler-lock-v6,pixi,pixi-lock-v6,requirements.
↪txt,requirements,reqs}]
                    [--format FORMAT]
                    [--solver {classic,libmamba,pycosat,rattler}]
                    [--force-reinstall] [--freeze-installed | --update-deps |
                    -S | --update-all | --update-specs]
                    [package_spec ...]
```

Positional Arguments

package_spec List of packages to install or update in the conda environment.

Named Arguments

--override-frozen DANGEROUS. Use at your own risk. Ignore protections if the environment is frozen.

-f, --file Read environment or package specs from a file. The format is detected from the filename or contents. Which formats are supported depends on the installed plugins (see the epilog for the list available here). Custom filenames require --format. May be repeated (e.g. --file=file1 --file=file2).

--environment-specifier, --env-spec Possible choices: cep-24, environment-yaml, env.yml, conda-lock-v1, conda-lock, environment.yml, explicit, rattler-lock-v6, pixi, pixi-lock-v6, requirements.txt, requirements, reqs

--env-spec is deprecated and will be removed in 27.3. Use the *--format* flag instead.

--format Possible choices: cep-24, environment-yaml, env.yml, conda-lock-v1, conda-lock, environment.yml, explicit, rattler-lock-v6, pixi, pixi-lock-v6, requirements.txt, requirements, reqs

Override auto-detection of the input file's format. See *conda export --help* for the formats available in your installation. Aliases are interchangeable with canonical names.

Target Environment Specification

-n, --name Name of environment.

-p, --prefix Full path to environment location (i.e. prefix).

Channel Customization

-c, --channel Additional channel to search for packages. These are URLs searched in the order they are given (including local directories using the 'file://' syntax or simply a path like '/home/conda/mychan' or '../mychan'). Then, the defaults or channels from .condarc are searched (unless --override-channels is given). You can use 'defaults' to get the default packages for conda. You can also use any name and the .condarc channel_alias value will be prepended. The default channel_alias is https://conda.anaconda.org/.

--use-local Use locally built packages. Identical to '-c local'.

-O, --override-channels Do not search default or .condarc channels. Requires --channel.

--repodata-fn Specify file name of repodata on the remote server where your channels are configured or within local backups. Conda will try whatever you specify, but will ultimately fall back to repodata.json if your specs are not satisfiable with what you specify here. This is used to employ repodata that is smaller and reduced in time scope. You may pass this flag more than once. Leftmost entries are tried first, and the fallback to repodata.json is added for you automatically. For more information, see conda config --describe repodata_fns.

--experimental Possible choices: jlap, lock

--experimental is deprecated and will be removed in 27.3. Deprecated: jlap and lock no longer supported.

--no-lock Disable locking when reading, updating index (repodata.json) cache.

--repodata-use-zst, --no-repodata-use-zst Check for/do not check for repodata.json.zst. Enabled by default.

--repodata-use-shards, --no-repodata-use-shards Use sharded repodata if available. Enabled by default.

Solver Mode Modifiers

--strict-channel-priority Packages in lower priority channels are not considered if a package with the same name appears in a higher priority channel.

--no-channel-priority Package version takes precedence over channel priority. Overrides the value given by *conda config --show channel_priority*.

--no-deps Do not install, update, remove, or change dependencies. This WILL lead to broken environments and inconsistent behavior. Use at your own risk.

--only-deps Only install dependencies.

--no-pin Ignore pinned file.

--exclude-newer Exclude packages published more recently than the given duration (e.g. 7d, 3d12h, 1w) or date (e.g. 2026-04-01, 2026-04-01T12:00:00Z). Date-only values use the start of the next UTC day. Supply 0 for no delay, using the current time as the cutoff. Channel and per-package overrides can be set via channel_settings and exclude_newer_package in .condarc.

--solver Possible choices: classic, libmamba, pycosat, rattler

Choose which solver backend to use.

--force-reinstall Ensure that any user-requested package for the current operation is uninstalled and reinstalled, even if that package already exists in the environment.

--freeze-installed, --no-update-deps Do not update or change already-installed dependencies.

--update-deps Update dependencies that have available updates.

-S, --satisfied-skip-solve Exit early and do not run the solver if the requested specs are satisfied. Also skips aggressive updates as configured by the 'aggressive_update_packages' config setting. Use 'conda config --describe aggressive_update_packages' to view your setting. --satisfied-skip-solve is similar to the default behavior of 'pip install'.

--update-all, --all Update all installed packages in the environment.

--update-specs Update based on provided specifications.

Package Linking and Install-time Options

--copy Install all packages using copies instead of hard- or soft-linking.

--no-shortcuts Don't install start menu shortcuts

--shortcuts-only Install shortcuts only for this package name. Can be used several times.

--clobber Allow clobbering (i.e. overwriting) of overlapping file paths within packages and suppress related warnings.

Networking Options

-C, --use-index-cache Use cache of channel index files, even if it has expired. This is useful if you don't want conda to check whether a new version of the repodata file exists, which will save bandwidth.

-k, --insecure Allow conda to perform "insecure" SSL connections and transfers. Equivalent to setting 'ssl_verify' to 'false'.

--offline Offline mode. Don't connect to the Internet.

Output, Prompt, and Flow Control Options

--json Report all output as json. Suitable for using conda programmatically.

--console Possible choices: classic, json

Select the backend to use for normal output rendering.

-v, --verbose Can be used multiple times. Once for detailed output, twice for INFO logging, thrice for DEBUG logging, four times for TRACE logging.

-q, --quiet Do not display progress bar.

-d, --dry-run Only display what would have been done.

-y, --yes	Sets any confirmation values to 'yes' automatically. Users will not be asked to confirm any adding, deleting, backups, etc.
--download-only	Solve an environment and ensure package caches are populated, but exit prior to unlinking and linking packages into the prefix.
--show-channel-urls	Show channel urls. Overrides the value given by *conda config --show show_channel_urls*.

Examples:

conda update -n myenv scipy

4.4.23 conda commands

```
usage: conda commands [-h]
```

4.4.24 Conda vs. pip vs. virtualenv commands

If you have used pip and virtualenv in the past, you can use conda to perform all of the same operations. Pip is a package manager and virtualenv is an environment manager. conda is both.

Scroll to the right to see the entire table.

Task	Conda package and environment manager command	Pip package manager command	Virtualenv environment manager command
Install a package	`conda install $PACKAGE_NAME`	`pip install $PACKAGE_NAME`	X
Update a package	`conda update --name $ENVIRONMENT_NAME $PACKAGE_NAME`	`pip install --upgrade $PACKAGE_NAME`	X
Update package manager	`conda update conda`	Linux/macOS: `pip install -U pip` Win: `python -m pip install -U pip`	X
Uninstall a package	`conda remove --name $ENVIRONMENT_NAME $PACKAGE_NAME`	`pip uninstall $PACKAGE_NAME`	X
Create an environment	`conda create --name $ENVIRONMENT_NAME python`	X	`cd $ENV_BASE_DIR; virtualenv $ENVIRONMENT_NAME`
Activate an environment	`conda activate $ENVIRONMENT_NAME*`	X	`source $ENV_BASE_DIR/ $ENVIRONMENT_NAME/bin/ activate`
Deactivate an environment	`conda deactivate`	X	`deactivate`
Search available packages	`conda search $SEARCH_TERM`	`pip search $SEARCH_TERM`	X
Install package from specific source	`conda install --channel $URL $PACKAGE_NAME`	`pip install --index-url $URL $PACKAGE_NAME`	X
List installed packages	`conda list --name $ENVIRONMENT_NAME`	`pip list`	X
Create requirements file	`conda export --format=requirements` or `conda list --export`	`pip freeze`	X
List all environments	`conda info --envs`	X	Install virtualenv wrapper, then `lsvirtualenv`
Install other package manager	`conda install pip`	`pip install conda`	X
Install Python	`conda install python=x.x`	X	X
Update Python	`conda update python*`	X	X

* `conda activate` only works on conda 4.6 and later versions. For conda versions prior to 4.6, type:

  - Windows: `activate`
  - Linux and macOS: `source activate`

* `conda update python` updates to the most recent in the series, so any Python 2.x would update to the latest 2.x and any Python 3.x to the latest 3.x.

4.5 Release notes

This information is drawn from the GitHub conda project changelog: https://github.com/conda/conda/blob/main/CHANGELOG.md

4.5.1 26.10.0 (2026-11-06)

Special announcement

This is an announcement about an important change in conda's functionality:

With this 26.10.0 release we are changing the default solver of conda to `conda-rattler-solver`!

The previous default, the `libmamba` solver, remains available as an alternative, as does the `pycosat` solver (also known as `classic`). The `classic` solver now ships as the separate `conda-pycosat-solver` plugin, which is bundled with conda. Existing `solver: classic` settings and `--solver=classic` commands keep working, and the plugin also accepts `--solver=pycosat`.

Why are we switching the solver?

In short: to make conda faster and more accurate.

Rattler is a set of Rust libraries for working with conda packages. The `conda-rattler-solver` plugin uses rattler's dependency solver and optimized sparse repodata parsing, which reduces overhead when resolving complex dependency trees. It solves environments faster than libmamba, with the largest improvements on large or heavily constrained environments.

Conda 26.10.0 switches to rattler for fresh installations and upgrades that do not have an explicit solver setting. If you have already selected a solver, conda keeps that choice.

What can I do if this update doesn't work for me?

If the new solver is not working as you expect:

- Check if the behavior you are observing is a known limitation or a deliberate change.
- If that's not the case, please submit a bug report or feature request in the conda-rattler-solver repository.
- If necessary, you can go back to using the `libmamba` solver without modifying your conda installation:
  - When possible, pass the command line option `--solver=libmamba` to your `conda` calls.
  - Otherwise (e.g. for `conda build ...` or `constructor ...`), set the environment variable `CONDA_SOLVER=libmamba`.
  - For permanent changes, use the conda configuration system: `conda config --set solver libmamba`.
- The `pycosat` solver (also known as `classic`) remains available with `--solver=pycosat` or `--solver=classic`.

Where can I learn more about conda-rattler-solver?

The documentation of the `conda-rattler-solver` plugin can be found at conda.github.io/conda-rattler-solver, including the known limitations and differences, and in the conda solvers guide.

Enhancements

- IMPORTANT: Set `solver: rattler` as the new default solver, including after upgrades when no solver is explicitly configured. Require `conda-rattler-solver >=26.10.2`. (#16853 via #16944)
- Track the last time an environment was activated via user invocation. Activation modifies the mtime of `<prefix>/conda-meta/last_activated` (best-effort; failures are ignored). The timestamp is exposed as `last_activated` in JSON output. (#14308 via #16808)
- Add an `is_active` property to `PrefixData`. (#14308 via #16808)
- Add `--disable-plugins` to selectively disable plugins at runtime using comma-separated names or by repeating the option. Keep `--no-plugins` as a boolean flag to disable all external plugins. (#15914 via #15915)
- Add `--enable-plugins` to keep selected plugins enabled when using `--no-plugins` or `CONDA_NO_PLUGINS`, including solver plugins needed for recovery. (#15914 via #15915)
- Add a `root_extras` option to `conda._private.shards.subset.build_repodata_subset` (and the underlying `RepodataSubset.reachable*` methods) so callers can request that CEP 44 `extra_depends` groups be fetched only for specific root/user-requested packages, without recursing into extras of their dependencies. (#16724 via #16742)
- Add `LinkSourceNotFoundError` and `LinkScriptError` (a `LinkError` subclass) so callers can detect a missing package cache source and a failed pre-/post-link script with `isinstance` and inspect `src`, `action`, `prec`, and `path` instead of parsing error messages. Error text is unchanged, and `LinkScriptError` remains catchable as `LinkError`, but code that matched the missing-source case against a bare `CondaError` now needs `LinkSourceNotFoundError`. (#16785 via #16747)
- Follow CEP 44 extras explicitly requested by dependency specs (e.g. `pkg[extras=more]`) when fetching sharded repodata, and revisit a package when another dependency activates an additional extras group. Groups that were not requested stay inactive. Previously only extras requested for root packages via `root_extras` were followed. (#16891 via #16908)
- Parse `MatchSpec` strings with the v3 parser for all solvers, not just for `rattler`. This keeps complete `extras` and `flags` lists (e.g. `httpx2[extras=[cli,http2]]` or `httpx2[extras=cli,http2]`) and accepts nested `when` expressions. (#16724 via #16906)

Bug fixes

- Preserve platform-named parent directories when parsing channel and package URLs, fixing conda-build failures with paths such as `osx-arm64/conda-bld`. (conda/conda-libmamba-solver#1108 via #16926)
- Preserve nested shard URLs when adding stored Anaconda tokens. (#16946 via #16948)
- Avoid `FileNotFoundError` on Windows when `rmtree()` deletes a directory that `RD /S /Q` already removed (for example during `conda package` temp cleanup). (#16539 via #16803)
- Write configuration files atomically when directory permissions allow, preserve macOS ACLs, and reject writes if the file changed in the meantime. Keep writable files editable in read-only directories, with best-effort recovery after write failures. (#16694 via #16695)
- Ask once before `conda env remove` and `conda remove --all` delete an environment. Those commands previously confirmed the package unlink and then, after the packages were already uninstalled, asked again before deleting the rest of the prefix, including non-conda files. Declining that second prompt left a half-deleted environment. The package plan and the prefix-deletion warning are now shown together before anything is removed, and declining leaves the environment unchanged. (#16753 via #16852)

- Fix `conda config --get` writing to the config file when no key is given. (#16825 via #16831)
- Fix `iter_records_v3()` not converting raw-bytes hashes (`sha256`, `md5`, `legacy_bz2_md5`, `attestations_sha256`) to hex strings, as written by e.g. `rattler-index`. (#16866)
- Retry remote requests that fail with HTTP 502 (Bad Gateway) or 504 (Gateway Timeout), which are usually transient server or CDN errors. (#16848)
- Fix `ObjectParameter` configuration types not expanding environment variables when `expandvars=True`. (#16897)
- Fix `conda.deprecations.deprecated` and `conda.deprecations.deprecated.argument` to preserve generator functions when decorating, instead of wrapping them in a plain function and losing generator semantics (e.g. `pytest` yield-fixtures now warn and yield correctly when stacked under `@pytest.fixture`). (#16900)
- Use a fresh YAML parser for every read and write to prevent concurrent calls and failed operations from corrupting later results. (#16943 via #16949)
- Show environment names in PowerShell prompts for canary packages built with `changeps1: false`. (#16858)

Deprecations

- Mark `conda.plugins.solvers`, `conda.resolve`, `conda.common.logic`, and `conda.common._logic` as pending deprecation, to be removed in 27.9. Use `conda-pycosat-solver` instead. (#14170 via #16520)
- Mark `conda.models.prefix_graph.GeneralGraph` and its `breadth_first_search_by_name` method as pending deprecation, to be removed in 27.9. (#16802)
- Mark `conda.testing.cases` as pending deprecation, to be removed in 27.9. Use native `pytest` fixtures instead. (#16900)
- Mark `conda.testing.helpers.assert_equals`, `assert_in`, and `assert_not_in` as pending deprecation, to be removed in 27.9. Use a plain `assert` instead. (#16900)
- Mark `conda.testing.helpers.supplement_index_with_repodata` as pending deprecation, to be removed in 27.9. Use `conda.models.records.PackageRecord.from_objects` instead. (#16900)
- Mark `conda.testing.helpers.add_feature_records_legacy` as pending deprecation, to be removed in 27.9. (#16900)
- Mark `conda.testing.integration.Commands` as pending deprecation, to be removed in 27.9. Use the `conda_cli` fixture instead. (#16900)
- Mark `conda.testing.integration.escape_for_winpath` as pending deprecation, to be removed in 27.9. Use `re.escape` instead. (#16900)
- Mark the `conda.testing.integration` constants `UNICODE_CHARACTERS`, `UNICODE_CHARACTERS_RESTRICTED`, `SPACER_CHARACTER`, and `cp_or_copy` as pending deprecation, to be removed in 27.9. (#16900)
- Mark `conda.testing.solver_helpers.temp_simple_env` as pending deprecation, to be removed in 27.9. Use `conda.testing.solver_helpers.SimpleEnvironment` instead. (#16900)
- Mark `conda.models.match_spec._parse_spec_str` as pending deprecation, to be removed in 27.9. Use `conda.models.match_spec._parse_spec_str_v3` instead. (#16724 via #16906)
- Mark the unused `conda.core.subdir_data.SubdirData` pickle cache helpers and the `cache_path_pickle` property as pending deprecation, to be removed in 27.9. (#16928)

Docs

- Document the `last_activated` timestamp in the JSON output. (#14308 via #16808)
- Add a "Rendering output" deep dive documenting the reporter backend as the output system for the conda CLI, with logging and error handling guidelines. (#16700 via #16843)
- Document logging guidelines for library code in the logging deep dive. (#16701 via #16912)
- Add a guide to choosing a solver to the documentation, with information about the available solvers and how to change your solver. (#16844)

Other

- Move the classic solver out of conda core and into the standalone `conda-pycosat-solver` plugin, bundled with conda by default. Both `--solver=pycosat` and `--solver=classic` continue to work unchanged. (#14170 via #16520)
- Keep PowerShell CI installations separate from PowerShell's runtime cache. (#16762 via #16907)
- Add `tmp_repodata_channel` for constructing local test channels from in-memory package metadata. (#16807)
- Replace production-channel inputs in `test_neutering_of_historic_specs` with a small local channel while preserving historical-spec regression coverage. (#16767 via #16807)
- Speed up `test_cross_channel_incompatibility` by using two small local channels instead of remote conda-forge/defaults metadata, preserving optional-constraint regression coverage. (#16768 via #16796)
- Raise the minimum Python version required to run conda to 3.11, following conda-build#6176. Conda can still create and manage environments containing Python 3.10 and older; this only affects the interpreter used to run conda itself. (#16795 via #16801)
- Do not allow local condarc files to populate the `context_testdata` fixture. (#16797 via #16809)
- Warn when a `conda doctor` reinstall attempt fails due to malformed `conda-meta` data. (#16636 via #16834)
- Convert raw-bytes hash digests to hex strings in memory when sharded repodata is decoded, rather than re-converting records on each read. (#16868)

Contributors

- @aditya0929 made their first contribution in https://github.com/conda/conda/pull/16831
- @conda-bot
- @danyeaw
- @dependabot[bot]
- @dholth
- @ForgottenProgramme
- @gavinw2143 made their first contribution in https://github.com/conda/conda/pull/16796
- @jezdez
- @kathatherine
- @kenodegard
- @larsoner made their first contribution in https://github.com/conda/conda/pull/16848
- @pre-commit-ci[bot]
- @r3wretrhy made their first contribution in https://github.com/conda/conda/pull/16803

- @ryanskeith
- @soapy1
- @travishathaway
- @VINAYRAO0406 made their first contribution in https://github.com/conda/conda/pull/16814

4.5.2 26.9.4 (2026-10-21)

Bug fixes

- Require `certifi >=2026.10.19` and `ca-certificates >=2026.10.19` to update the packaged CA bundles following the eMudhra intermediate CA compromise. (#16880 via #16881)

Contributors

- @jezdez

4.5.3 26.9.3 (2026-10-14)

Bug fixes

- Preserve installed file metadata when safety checks are disabled so Windows launcher lookup can use the checksums supplied by the package. (#16856 via #16855)

Contributors

- @jezdez

4.5.4 26.9.2 (2026-10-08)

Bug fixes

- Fix `conda run` selecting the active environment instead of the requested target when conda-build is installed. (#16811 via #16821)
- Fix `AttributeError: 'PackageRecord' object has no attribute 'extras'` when solving from channels whose repodata uses extra_depends by adding `extra_depends` to `PackageRecord` and ignoring the `extras` selector during record matching. (#16822 via #16823)

Contributors

- @danyeaw
- @jezdez
- @kenodegard

4.5.5 26.9.1 (2026-10-02)

Bug fixes

- Keep cached package archive paths consistent with their recorded filenames and checksums when both archive formats are present. Restore the fix from conda 26.7.3 that was missing in 26.9.0. (#9674 via #16793)

Contributors

- @jezdez

4.5.6 26.9.0 (2026-09-29)

Enhancements

- Clarify the classic solver's repodata fallback message so it no longer describes the fallback as an unsuccessful attempt. (#12008 via #16599)
- Add `--exclude-newer` and `exclude_newer` configuration to exclude package records newer than a timestamp cutoff, with channel overrides and `exclude_newer_package` settings. Configured cutoffs also filter `conda search` results. The configured solver must support the requested cutoffs. Accurate timestamps can give security vendors time to flag newly built malicious packages before installation. Conda prefers channel-provided `indexed_timestamp` values and otherwise uses build timestamps. Records without usable timestamps remain eligible, so filtering does not guarantee a full delay after publication. (#15759)
- Leave cached repodata unfiltered when applying `exclude_newer`, and validate solver selections before installation. (#15759)
- Load subcommand parsers only when needed, reducing the number of modules imported at startup and improving conda startup time. (#15867 via #15868)
- Prefer extracted packages cached on the target environment's filesystem during transaction setup so they can be hard linked instead of copied. (#15969)
- Use `conda-launchers` for Windows entry points, including native ARM64 launchers, and verify their SHA-256 hashes before installation. Use SHA-256 instead of MD5 for `conda init` launcher comparisons. Retain bundled launchers for upgrade compatibility and Windows 32-bit fallback. (#16293)
- Add `conda plugins list` and `conda plugins info` to list installed plugins and inspect their metadata, including disabled plugins. (#16354, #16356)
- Show Constructor installer metadata in `conda info`. (#16432)
- Show guidance for enabling sharded repodata when a channel provides only shards and sharded repodata is disabled. (#16453)
- Show `add_pip_as_python_dependency` under Solver Configuration in `conda config --describe`. (#16545)
- Add `is_stdin_tty()`, `is_stderr_tty()`, and `is_interactive_tty()` helpers to `conda.common.terminal` for checking stdin, stderr, and both stdin and stdout, respectively. Keep `is_tty()` as a stdout-only check. Use these helpers for stream checks and interactive prompts instead of handwritten `isatty()` checks. (#16594)
- Avoid loading unrelated package records when copying indexes or preparing classic-solver solves, while preserving installed records and cached packages with their dependencies. (#16638)
- Speed up wildcard searches of sharded channels by matching package names before constructing records, and show progress when searching for similar package names. (#16745 via #16757)
- Add a `conda config --clear KEY` option to explicitly set sequence configuration parameters to an empty list. (#7617 via #16499)

Bug fixes

- Do not force `context.dev` off when `--dev` is absent from activation or hook commands. Preserve dev mode configured through `CONDA_DEV` or `dev: true`, keeping `CONDA_EXE` intact when stacking or reactivating environments. (#14142 via #16571, #15696 via #16571)

- Require confirmation through `--yes` or `always_yes` to replace an existing environment with `conda env create`. (#15071)
- Validate `conda install --revision` values during command parsing and report non-integer values as usage errors. (#16536)
- Filter duplicate `.tar.bz2` records from v3 sharded repodata when a `.conda` package with the same filename stem exists. (#16310)
- Install pip dependencies from multiple environment files in one pip invocation so pip resolves them together. (#16359)
- Cache the absence of classic `repodata.json` after an HTTP 404 when shards are present, avoiding repeated fallback requests. (#16452)
- Limit cached channel notice responses to 24 hours so newly published notices can be fetched. (#16502)
- Fix `channel_settings` lookups for channel names and URLs. (#16510)
- Include support for individual sharded-repodata package shards up to 64 MiB after decompression, first released in conda 26.7.2. (#16575)
- Fix false reports of altered files from `conda doctor` after binary prefix replacement on macOS ARM64 by recording `sha256_in_prefix` after code signing. (#16590)
- Fix `conda doctor --fix` reinstalls for missing or altered files by building `name=version=build` specs from conda-meta records and passing `force_reinstall` through to the solver. (#16591 via #16617)
- Keep extracted cache packages referenced by soft-linked environments when running `conda clean --packages` or `conda clean --all`. (#16614)
- Wrap channel notice text to fit the terminal. (#16598 via #16629)
- Invalidate context-derived caches when entering and exiting `Context._override()`, including `memoizedproperty` values such as `custom_multichannels` and `custom_channels`, and the `Channel.from_value` cache. Prevent stale values from masking temporary overrides or persisting after they end. (#16631 via #16651)
- Fix `Channel.url(with_credentials=True)` mangling authenticated sharded-repodata URLs by recognizing `.msgpack.zst` as a repodata extension. (#16637 via #16634)
- Send YAML environment-file warnings to stderr, including ignored sections and `pip:` dependencies without an explicit conda `pip` package. (#16543 via #16641)
- Make `conda env config` command help consistent and remove the invalid `conda env config --append` example. (#16642 via #16643)
- Fix the classic solver raising a `TypeError` instead of a package-not-found error for missing packages. (#16659)
- Find v3 package records when searching channels without sharded repodata. (#16677 via #16731)
- Ensure v3 packages are available when mixing sharded and unsharded channels. (#16676 via #16688)
- Allow configuration map entries containing periods to be set, read, and removed without losing nested-key access. (#16691 via #16693)
- Fix `get_session` raising a `TypeError` when `channel_settings` identifies a channel with a `file://` URL. (#16698 via #16732)
- Parse unquoted comma-separated MatchSpec extras as a list (e.g. `pkg[extras=http2,cli]`). (#16722 via #16730)
- Preserve `exclude_newer` capabilities when wrapping solver backends for sharded repodata. (#16752)

- Fix `IsADirectoryError` when installing packages with absolute directory symlinks, and remove directory symlinks correctly during package removal and rollback. (#16759)
- Report an actionable error without a traceback when activating with a positive `CONDA_SHLVL` and an unset or empty `CONDA_PREFIX`. (#15864, #16764 via #16064)
- Fix native Windows ARM64 upgrades from defaults conda 26.7.2 that leave Python entry points missing by preserving the launcher needed by the running installer. (#16776 via #16777)
- Report Python entry-point creation failures through the existing transaction error handling and roll back failed transactions when rollback is enabled. (#16776 via #16777)

Deprecations

- Mark `conda activate --dev`, `conda create --dev`, `conda install --dev`, `conda remove --dev`, `conda.base.context.Context.dev`, and `conda.utils.wrap_subprocess_call(dev_mode)` as pending deprecation, to be removed in 27.9. Outside dev mode, activation unsets `_CE_M` and `_CE_CONDA`, while shell wrappers continue expanding them. Set `PYTHONPATH` to the conda source root instead. (#14142 via #16571)
- Mark the `conda.cli.conda_argparse.configure_parser_*` re-exported functions as pending deprecation, to be removed in 27.9. Import `configure_parser` directly from the relevant `conda.cli.main_*` module instead. (#15867 via #15868)
- Mark `conda.cli.conda_argparse.user_rc_path`, `sys_rc_path`, `escaped_user_rc_path`, and `escaped_sys_rc_path` as pending deprecation, to be removed in 27.9. Use `conda.base.context.user_rc_path` or `sys_rc_path` instead, applying `.replace("%", "%%")` locally for the escaped variants. (#15867 via #15868)
- Remove `conda.gateways.disk.create.create_application_entry_point`, `conda.gateways.disk.create.ProgressFileWrapper`, and `conda.gateways.disk.create.create_fake_executable_softlink`, deprecated in 26.3. (#16314)
- Remove `conda.cli.common.arg2spec`. Use `str(MatchSpec(arg))` with `conda.models.match_spec.MatchSpec` instead, keeping error handling and name-only validation for update commands in the caller. (#16314)
- Remove the deprecated `json` keyword argument from `conda.cli.common.specs_from_args` and `specs_from_url`. Stop passing this argument. (#16314)
- Remove `conda.cli.main_compare.get_packages`. Use `conda.core.prefix_data.PrefixData(prefix, interoperability=True).map_records()`, which returns a mapping of package names to records instead of a sorted list. (#16314)
- Remove `json.JSONEncoder` monkey patching for `frozendict` and `obj.to_json()`. Use `conda.common.serialize.json.dumps()` or `CondaJSONEncoder` to encode these objects. (#16314)
- Remove `conda.auxlib.entity.EntityEncoder`, `conda.common.serialize.EntityEncoder`, and `conda.exports.EntityEncoder`. Use `conda.common.serialize.json.CondaJSONEncoder` instead. Remove `conda.common.serialize.json_load` and `json_dump`. Use `loads()` and `dumps(..., sort_keys=True)` from `conda.common.serialize.json`, respectively. (#16314)
- Remove `conda.auxlib.logz.DumpEncoder`, `_DUMPS`, and `jsondumps`. Use `CondaJSONEncoder` or `dumps(..., ensure_ascii=False, sort_keys=True)` from `conda.common.serialize.json` instead. (#16314)
- Remove `conda.plugins.hookspec.spec_name`. Use `conda.base.constants.APP_NAME`. (#16314)
- Remove type re-exports from `conda.plugins`. Import them from `conda.plugins.types` instead. (#16314)
- Mark implicit installation of pip as a Python dependency as pending deprecation. Deprecate this behavior in conda 27.3 and change the default of `add_pip_as_python_dependency` to `false` in conda 27.9. To keep installing pip automatically with Python, run `conda config --set add_pip_as_python_dependency true`.

To request pip explicitly, include it in the package specs, for example `conda create -n myenv python pip`. To opt out early, run `conda config --set add_pip_as_python_dependency false`. (#16404)

- Mark `conda.cli.install.get_revision` as pending deprecation, to be removed in 27.9. Parse revision numbers with `int()` and handle invalid values in the caller. (#16536)
- Mark passing `caused_by` as a positional argument to `CondaError` as pending deprecation, to be removed in 27.9. Use the `caused_by` keyword argument instead. (#16537)
- Remove `conda.env.env.Environment`. Use `conda.env.env.EnvironmentYaml` instead. (#16314)
- Remove `conda.env.specs.yaml_file.YamlFileSpec.environment`. Use `YamlFileSpec.env`, which returns `conda.models.environment.Environment` instead of `EnvironmentYaml`. (#16314)
- Remove the CEP-24 compliance warning when loading non-compliant environment files. (#16314)
- Remove automatic fallback to the `environment.yml` environment specifier when auto-detection fails. Pass `--format=environment.yml` to select this reader explicitly. (#16314)
- Remove `conda.auxlib.compat.isiterable`. Use `conda.common.compat.isiterable` instead. (#16564)
- Remove `conda.core.index.calculate_channel_urls`. Use `conda.models.channel.all_channel_urls(context.channels)` for configured channels, preserving any custom channels, platform, or `use_local` handling in the caller. (#16564)
- Mark `conda.base.context.Context.experimental` as pending deprecation, becoming deprecated in 27.3 and removed in 27.9. (#16018 via #16685)
- Mark `conda.common.configuration.Configuration.register_reset_callaback` as pending deprecation, to be removed in 27.9. Use the correctly spelled `register_reset_callback` instead. (#16631)

Docs

- Note that `_CE_M` and `_CE_CONDA` in activation examples are empty no-ops. Use `PYTHONPATH` and `dev/start` for local conda development. (#14142 via #16571)
- Correct the cheatsheet page's stale PDF note — the linked file is 64 KB and two pages, not 3 MB and single-page. (#15122)
- Update the commands concept page with common conda commands, command-line help examples, and a note about plugin-provided commands. (#16023 via #16485)
- Update the channels concept page with examples of common public channels, guidance for finding channels on Anaconda.org, and a section explaining local channels. (#16026 via #16484)
- Add a user guide task page for using authenticated channels with the `conda-auth` plugin. (#16231 via #16483)
- Clarify how command-line package specifications are interpreted when installing packages with conda. (#10491 via #16465)
- Document conda environment activation in Bash and PowerShell scripts using `conda shell.<shell> hook`, and `conda run` as an alternative for executing a single command in an environment. (#7980 via #16478)
- Install PlantUML from conda-forge instead of downloading the jar from SourceForge during docs builds. (#16492)
- Fix Miniconda download and information links. (#16529)
- Document `add_pip_as_python_dependency`, including the planned default change in #16404. (#16545)
- Add a developer guide to planning, announcing, and shipping breaking changes beyond public API removals, including configuration defaults, solver defaults, and CLI behavior. Link it from the Deprecations page and developer guide index. (#16570)
- Document running and writing benchmarks, and link to Bencher results. (#16586 via #16639)

- Clarify wildcard matching in version specs: a trailing * matches version segments, so `1.4*` matches `1.4.1` but not `1.40`. An embedded * uses glob matching. (#16619 via #16655)
- Add a guide to installing supported pure Python wheels from PyPI with `conda-pypi`, including workflows, additional commands, security considerations, and limitations. (#16475 via #16625, #16756)
- Update the new features page for stable `conda-pypi` support and multi-platform lockfiles, and add a card for `--exclude-newer`. (#16626)
- Add a card to the new features page for native Windows ARM64 support, including choosing between `win-64` and `win-arm64` environments. (#16736)
- Clarify the `exclude_newer` build-timestamp fallback and its publication-cooldown limitations. (#16739 via #16741)

Other

- Build Windows ARM64 canaries with native Python 3.14 and exercise their Windows entry points. (#15448 via #16719)
- Require `packaging` 26.3 or newer. (#16354)
- Speed up `test_dont_remove_conda` with local `conda` and `pycosat` test packages. (#16426, #16427 via #16577)
- Add Windows ARM64 GitHub-hosted runner coverage using win-64 emulation. (#16466)
- Run the Windows test suite natively on ARM64 with Python 3.14 and conda-forge packages. (#16540)
- Raise the `ruamel.yaml` upper version bound from <`0.19` to <`0.20`. (#16576)
- Fix always-true skip condition so `test_dont_remove_conda_3` runs with solvers shipped with conda. (#16579)
- Report benchmark results for the measured commit and compare matching testbeds. (#16586 via #16639)
- Speed up `test_clone_env_with_conda` by installing the local test-recipes `conda` package instead of a full `conda-forge::conda` stack before cloning. (#16604)
- Speed up `test_update_env_no_action_json_output` by using `tmp_env` with `ca-certificates` instead of creating an environment with `pip` and PyPI `click`. (#16605)
- Speed up `test_json_create_install_update_remove` by using `test_recipes_channel` with `pycosat` and `versioned` instead of remote `zlib` and `ca-certificates`. (#16606)
- Speed up `test_dont_remove_conda_3` by checking transaction verification directly instead of installing conda and converting the checkout. (#16669 via #16751)
- Speed up `test_update_env_json_output` by using `test_recipes_channel` with `small-executable` and `dependency` instead of remote `ca-certificates` and `zlib`. (#16670 via #16744)
- Speed up `test_update` by using `test_recipes_channel` with `small-executable` and `dependency` instead of remote `ca-certificates` and `zlib`. (#16671 via #16743)
- Record informational pull request benchmark comparisons and runner diagnostics on Ubuntu 24.04. (#16681)
- Fix context cleanup in the shard connection test. (#16702)
- Add `conda-rattler-solver` to the plugins bundled with conda and update plugin minimums to `conda-libmamba-solver` >=`26.7.0`, `conda-lockfiles` >=`0.2.2`, `conda-pypi` >=`0.13.0`, `conda-rattler-solver` >=`0.2.0`, and `conda-self` >=`0.3.0`. (#16491, #16784)

Contributors

- @agriyakhetarpal
- @andreruizloera made their first contribution in https://github.com/conda/conda/pull/16436
- @anki-code
- @carterbox
- @codewithdaniel1 made their first contribution in https://github.com/conda/conda/pull/16465
- @conda-bot
- @danyeaw
- @dependabot[bot]
- @ForgottenProgramme
- @GruffElixir made their first contribution in https://github.com/conda/conda/pull/16668
- @isuruf
- @jaimergp
- @jezdez
- @jjerphan made their first contribution in https://github.com/conda/conda/pull/16495
- @kathatherine
- @kenodegard
- @lrandersson
- @mwtoews made their first contribution in https://github.com/conda/conda/pull/16455
- @paperbenni made their first contribution in https://github.com/conda/conda/pull/16635
- @ryanskeith
- @soapy1
- @travishathaway

4.5.7 26.7.3 (2026-09-29)

Bug fixes

- Ensure v3 packages are available when mixing sharded and unsharded repos. (#16676 via #16688)
- Keep cached package archive paths consistent with their recorded filenames and checksums when both archive formats are present. (#9674)

Contributors

- @jezdez

4.5.8 26.7.2 (2026-09-03)

Bug fixes

- Allow individual sharded-repodata package shards up to 64 MiB after decompression. (#16575)

Contributors

- @jezdez

4.5.9 26.7.1 (2026-08-18)

Bug fixes

- Fix `DeprecationHandler` version fallback when `packaging` raises `InvalidVersion` for unparseable versions (e.g. `None` under packaging >=26.3). (#16495)
- Refresh cached empty channel notice responses and parse CEP 6 `expires_at` timestamps. (#16500, #16501)
- Fall back to threaded package extraction when process synchronization primitives are unavailable. (#16512)
- Fix `CondaHttpAuth.add_binstar_token` mangling URLs that end in `notices.json` (or any filename without a platform subdir) by correcting `Channel.url()` to preserve `package_filename` when `platform` is `None`. (#16516)
- Fix `AttributeError` when accessing `CondaError.guidance`, `__str__`, or `dump_map` on subclasses that skip `CondaError.__init__`. (#16534 via #16535)
- Report extraction errors from process workers when they cannot cross the process boundary, and include process-pool failure causes in debug logs. (#16552)

Contributors

- @jezdez
- @kenodegard
- @travishathaway

4.5.10 26.7.0 (2026-07-27)

Enhancements

- Replace runtime dependency on `zstandard` with `compression.zstd` (Python >=3.14) and `backports.zstd` (Python 3.10–3.13) for repodata and sharded repodata decompression. (#15486 via #16191)
- Defer `concurrent.futures` and `threading` imports in `common/io.py` so they only load when executor classes (`DummyExecutor`, `ThreadLimitedThreadPoolExecutor`) are first accessed. (#15867 via #15881)
- Defer heavy imports in `exceptions.py` (`models.channel`, `auxlib.logz`, `common.io`, `common.iterators`, `serialize.json`) to function level so they only load on error paths. (#15867 via #15880)
- Defer importing `conda.plugins.types` until it is imported explicitly or one of its deprecated re-exports is accessed. (#15867 via #15893)
- Defer `ruamel.yaml` import in `common/serialize/yaml.py` so it only loads when YAML parsing/writing is first needed. (#15867 via #15882)
- Cache `context.root_writable` for the lifetime of the `Context` instance to avoid opening a file on every access. (#15867 via #15887)
- Skip plugin loading for shell activation commands (`conda shell.* activate | deactivate | reactivate | hook`), avoiding ~430 imports when the plugin manager has not already been initialized. Plugin authors should not rely on `pre_commands` / `post_commands` hooks running for these commands. (#15867 via #15877)
- Add `conda.common.terminal` module centralizing TTY detection and terminal output-mode logic with helpers `is_tty()`, `no_color()`, `force_color()`, and `should_use_color()` that respect the `NO_COLOR`, `FORCE_COLOR`, and `TERM=dumb`/`TERM=unknown` environment variables. (#15943 via #15982)

- Suppress progress-bar and spinner animations in the console reporter backend when the output is not a TTY or `TERM=dumb`/`TERM=unknown` is set. (#15943 via #15982)
- Replace the quadratic position scan in `diff_for_unlink_link_precs()` with linear order restoration. A focused synthetic benchmark with a 50,000-record prefix improved this function by about 780x. (#15969 via #15970)
- Build a per-name candidate index in `PrefixGraph.__init__()` to accelerate graph construction for large prefixes. (#15969 via #15971)
- Extract built-in `.conda` and `.tar.bz2` package archives with a process pool so zstd decompression can run outside the GIL, while keeping package-cache metadata updates in the parent process. (#15969 via #15974)
- Batch per-file `codesign` invocations on osx-arm64 into one end-of-transaction call, reducing process launches for packages with many prefix-replaced binaries. This does not affect other platforms. (#15969 via #15975)
- Add structured error guidance to `CondaError`, including actionable terminal hints, a `guidance` field in `--json` output, and initial guidance for `RemoveError`, `PackagesNotFoundError`, and `UnsatisfiableError`. (#16036 via #16160)
- Add search support for sharded repodata while disallowing `conda search *` to avoid downloading every shard. (#16134 via #16149)
- Improve conflicting plugin errors to list each provider by distribution name and version when available. (#16166)
- Add support for aliases on plugin-provided subcommands. (#16187 via #16188)
- Make `conda env create` an alias of `conda create`. (#15576 via #16210)
- Support displaying plugin parameter values with `conda config --show-sources`. (#16238 via #16246)
- Rework `InvalidInstaller` as a `CondaError` with structured guidance, removing redundant CLI try/except wrappers. (#16276)
- Add a `conda_error_hints` plugin hook for appending structured guidance hints to `CondaError` output. (#16290 via #16305)
- Introduce `conda.core.solve.BaseSolver` as the solver plugin base class. (#16320)
- Add channel notices cache directory to `conda info` output. (#16374)
- Suggest `conda self update` when the `conda-self` plugin is installed. (#16319 via #16414)
- Validate `--console` values against available reporter backends and reject unknown values with a clear error. (#16418 via #16420)

Bug fixes

- Fix `MatchSpec` incorrectly collapsing URL-form channel specs through `channel_alias`, causing `file://path/to/distr::pkg` to resolve to the remote `distr` channel instead of the local one, and `https://repo.anaconda.com/pkgs/free::numpy` to lose its explicit host. URL-form specs now round-trip through `str(MatchSpec(...))` with their identity preserved. (#16175 via #16176)
- Fix `ChannelMatch` incorrectly matching URL-form channel specs against channels at a different host with the same path (e.g., `https://software.repos.intel.com/python/conda` no longer matches records from `https://other.example.com/python/conda`). (#16176)
- Fix `conda.common.io.timeout()` leaving the `SIGALRM` alarm armed (and the previous signal handler unrestored) when the wrapped callable raised an unexpected exception, which could cause the alarm to fire later during unrelated code. The alarm is now always cancelled and the previous handler restored via a `finally` block. (#15702 via #16437)

- Compare channels by `canonical_name` when merging prefix and repodata records, so a prefix record built from a bare URL (`platform=None`) matches a repodata record built from a subdir URL. Fixes doubled or platform-mismatched subdirs in `PackageRecord.dist_str()`. (#15933 via #15934)
- Avoid a `TypeError` in `CondaHTTPError` and `UnavailableInvalidChannel` by only reading the RFC 9457 `detail` field from `application/problem+json` responses. (#15999)
- Fix `conda clean --all` help text to remove obsolete lock-file mention and list tempfiles among the removed targets. (#16043 via #16156)
- Remove the environment directory created by a failed `conda create` when the transaction fails to verify (e.g. a `ClobberError` under `CONDA_PATH_CONFLICT=prevent`), instead of leaving a partial/empty environment folder behind. Prefixes that already existed before the transaction are never removed. (#16076 via #16421)
- Raise `NoChannelsConfiguredError` from `conda search` when no channels are configured, matching the behavior of `conda install`. (#16180 via #16181)
- Prevent `conda env create` from creating environments inside existing prefixes. (#16210)
- Make `conda env create --dry-run` and `conda create --dry-run` outputs consistent. (#16210)
- Forward the `truststore` SSL context to proxy connections so that `ssl_verify: truststore` is honored when a proxy is configured, not just on direct connections. (#16252 via #16253)
- Avoid a crash when trying to format a `CondaError` with a message that has already been formatted. (#16263 via #16266)
- Fix conda-meta JSON paths for wheel package records by deriving them from package record name, version, and build instead of the package filename or extracted package directory. (#16235, #16265 via #16387)
- Include pip actions in the JSON output for `conda create` commands. (#16269)
- Ensure auth headers are set when fetching sharded repodata. (#16250 via #16273)
- Fix `conda env update` reporting a literal `{0}` placeholder when an external installer is unavailable. (#16276)
- Exclude test files from Hatch build to prevent them from being packaged. (#15513 via #16289)
- Fix an `AttributeError` when moving files to the trash on Windows. (#15760 via #16315)
- Preserve relocatable Python shebangs without depending on the build tool's host prefix name. (#16327 via #16328)
- Allow `conda doctor --fix --override-frozen` to run fixers against frozen environments. (#16336 via #16337)
- Fix parsing of entry point definitions surrounded by quotes. (#16339 via #16340)
- Increase the decompressed size limit of `repodata_shards.msgpack.zst` to allow for channels with a very large number of packages. (#16285 via #16383)
- Restore imports of deprecated `conda.common.io.IS_INTERACTIVE` so downstream plugins continue to load. (#16401 via #16406)
- Fix platform detection when emulating win-64 on win-arm64, which previously caused conda to select win-arm64 packages for the win-64 base environment. (#16416 via #16417)
- Fix a `NameError` in `conda.auxlib.logz.stringify` when logging an HTTP response with `Content-Type: application/json`. This regression was introduced by #15926. (#16446)
- Skip CUDA detection on osx-arm64. Warn and assume CUDA is unsupported when detection raises `PermissionError`, such as when conda runs in a sandbox. (#16388)
- Add `Shards.iter_records_v3()` to yield records from classic and v3 repodata with unique `(key, section_name)` keys. (#16099)

- Include the Python entry point validation first released in conda 26.5.2 to address GHSA-9m8m-c4j3-rj2c. (#16168)
- Render nested structured error guidance when printing a `CondaMultiError` (e.g. `conda remove conda`). Apply `conda_error_hints` to each nested leaf error rather than the multi-error container. (#16462)
- Include v3 records in `Shards.iter_records()` again so solvers that still call this API can install from shards-only channels such as `conda-pypi`. Prefer `Shards.iter_records_v3()` for section-aware iteration. (#16469)
- Fix environment history being deleted when rolling back a failed install into an existing environment. (#16473 via #16474)
- Remove the target prefix when `conda create` fails and that path did not exist before the create. (#16474)

Deprecations

- Mark `conda.common.serialize.yaml.CondaYAMLRepresenter` as pending deprecation, to be removed in 27.9. Use `conda.common.serialize.yaml.write()` and `conda.common.serialize.yaml.read()` for serialization, or subclass `ruamel.yaml.representer.RoundTripRepresenter` directly. (#15867 via #15882)
- Mark `conda.common.io.IS_INTERACTIVE` as pending deprecation, to be removed in 27.9. Use `conda.common.terminal.is_tty()` instead. (#15943 via #15982)
- Mark `conda.gateways.subprocess.subprocess_call_with_clean_env` as pending deprecation, to be removed in 27.9. Use `conda.gateways.subprocess.subprocess_call()` with a cleaned copy of `os.environ` instead. (#16312)
- Mark `conda.exports.Solver` as pending deprecation, to be removed in 27.9. Use `conda.core.solve.Solver` instead. (#16320)

Docs

- Add warning to new features page about removing environments with `conda-pypi` packages before uninstalling Miniconda or Anaconda Distribution. (#16225, #16242)
- Update guidance on strict channel priority and add a channel configuration best-practices section. (#15996 via #16247)
- Include shards documentation from `conda-libmamba-solver` in `conda` to accompany the ported implementation. (#16267, #16277)
- Update Miniconda links on the platform installation pages to point directly to the Anaconda download page. (#16281 via #16284)

Other

- Add the private `conda/_preview/` namespace for opt-in preview features and scaffold the non-operational `env-setup` preview. (#16011 via #16014)
- Replace the deprecated `JasonEtco/create-an-issue` GitHub Action with native GitHub CLI commands. (#16233)
- Re-add `conda-pypi` to the conda recipe and require `conda-pypi >=0.10.1`. (#16210, #16222)
- Enable infrastructure-managed Dependabot configuration via conda/infrastructure templates. (#16251, #16271)
- Speed up `test_create_install_update_remove_smoketest` by serving local test-recipes over HTTP instead of remote python/flask solves. (#16009 via #16287, #16307)

- Speed up `test_dont_remove_conda` by merging duplicate setup from `test_dont_remove_conda_1` and `test_dont_remove_conda_2`, parametrizing remove order, and sharing one session-scoped conda install across variants. (#16299 via #16311)

Contributors

- @Adelagric made their first contribution in https://github.com/conda/conda/pull/16420
- @agriyakhetarpal
- @bdice made their first contribution in https://github.com/conda/conda/pull/16328
- @conda-bot
- @danyeaw
- @carterbox made their first contribution in https://github.com/conda/conda/pull/16446
- @dholth
- @duncanmmacleod
- @isuruf
- @jaimergp
- @jsmolic
- @jezdez
- @kathatherine
- @kenodegard
- @ForgottenProgramme
- @mcg1969
- @ryanskeith
- @soapy1
- @travishathaway
- @Functionhx made their first contribution in https://github.com/conda/conda/pull/16391
- @btraven00 made their first contribution in https://github.com/conda/conda/pull/16266
- @dependabot[bot]
- @eeshsaxena made their first contribution in https://github.com/conda/conda/pull/16338
- @pre-commit-ci[bot]

4.5.11 26.5.3 (2026-06-16)

Bug fixes

- If shards were not found on a channel, conda would cache the not-found response, which can break shards-only channels. Instead, always check for sharded repodata if classic repodata has not been cached. (#16183)

Other

- Fix `get_shortcut_dir()` test helper to use the supplied prefix when detecting `.nonadmin` on Windows, aligning with menuinst 2.5 shortcut path behavior. (#16207)

Contributors

- @danyeaw
- @jezdez

4.5.12 26.5.2 (2026-06-01)

Bug fixes

- Validate Python entry point definitions before generating entry point scripts. (#16168)

Contributors

- @jaimergp

4.5.13 26.5.1 (2026-05-26)

Bug fixes

- Fix channel notices display failing with `ImportError` when a decorated command replaces base `python` while conda is still running on the previous interpreter. Pre-import `conda.notices.views` before the command so post-command display does not load modules from rewritten `site-packages`. (#16126 via #16142)
- Fix crash in `UnavailableInvalidChannel` and `CondaHTTPError` when `simplejson` is installed and a non-JSON HTTP error response is received. (#16136)

Contributors

- @jezdez
- @kenodegard

4.5.14 26.5.0 (2026-05-15)

Enhancements

- Add parsing and storage support for conditional dependencies, optional dependency groups, and variant flags in `MatchSpec` expressions (introduced in conda/ceps#164, conda/ceps#165, and conda/ceps#166). (#15443)
- Add `--clobber` flag to `conda create`, matching existing support in `conda install` and `conda update`. (#15584 via #15801)
- Display a Warning when exporting lockfiles from environments containing packages installed with pip, uv, or other 3rd party Python package managers. (#15581 via #15838)
- Speed up condarc search path expansion on every `reset_context()` by ~5x (~1.35 ms/call to ~0.27 ms/call on a 27-entry path), via cheaper `custom_expandvars()` and `os.scandir()`-based directory traversal in `Configuration._expand_search_path()`. (#15867)
- Drop the `ChannelType` metaclass from `conda.models.channel.Channel`. Single-argument `Channel(value)` construction and `MultiChannel` dispatch have moved to `Channel.__new__`, and the per-value cache is now `@functools.cache` on `Channel.from_value`. (#15867)
- Remove a redundant `context.__init__()` call in `main_subshell()`. The first initialization with `pre_args` was immediately repeated after the `no_plugins` check without any change in arguments. (#15867)

- `ContextStack` now allocates a single slot on construction instead of three. Slots are still doubled on demand, so deep nesting works identically. Eliminates two unnecessary `ContextStackObject` allocations at import time. (#15867)
- `Resolve._get_reduced_index()` no longer uses `copy.deepcopy()` to snapshot `specs_by_name`. A dict comprehension with slice copies (`v[:]`) is sufficient because `MatchSpec` objects are immutable. Removes the `import copy` from `resolve.py`. (#15867)
- Defer heavy imports in `notices/core.py` and `cli/main_env.py` to reduce the module cost of loading individual subcommand modules. (#15867)
- Add fast path for `conda --version` / `conda -V` that prints the version and exits without loading the argument parser, context, or plugin system. (#15867)
- Replace f-string log calls with %-style lazy formatting across the codebase. String interpolation is now deferred until the log record is actually emitted. Enable the ruff `G004` rule to enforce this going forward. (#15867)
- `context.category_map` is now a `@memoizedproperty`. The large static dict was rebuilt on every access; it is now constructed once and cached for the lifetime of the `Context` instance. (#15867)
- Hot-path regexes in `conda/models/match_spec.py` and `conda/activate.py` are now compiled once at module load time instead of on every call. Removes repeated `re.compile()` overhead from the `MatchSpec` parser and the shell activation path resolver. (#15867)
- `PrefixData._load_single_record()` now reads the conda-meta JSON file with `Path.read_bytes()` + `json.loads()` instead of opening a text file handle and calling `json.load()`. For the small files typical of conda-meta records, a single read syscall followed by an in-memory parse is faster than the incremental read performed by `json.load()`. (#15867)
- Add Ruff `TID253` rule to ban module-level imports of `requests` on the startup path, preventing accidental reintroduction of heavy import chains (e.g. the A1 regression fixed in #15866). Files in `conda/gateways/` and other non-startup-path locations that legitimately import `requests` are annotated with `# noqa: TID253`.
- Port sharded repodata implementation from conda-libmamba-solver. (#15906)
- Add `conda_exception_observers` plugin hook for observational callbacks on any exception conda encounters (filterable per observer via `watch_for`). (#15910)
- Bump `conda-incubator/setup-miniconda` to v4.0.0 across `.github/workflows/tests.yml`. v4.0.0 ships the perf work tested in #15741, including direct `.condarc` YAML writes, local `isDefaultEnvironment` resolution, parallelized Windows `takeown`, bulk-move pkg cleanup, and dropped HTML index scraping; the Node.js 24 runtime is the breaking change motivating the major bump. (#15923)
- Add default-implemented `available_platforms` and `env_for(platform)` on `EnvironmentSpecBase` (**EXPERIMENTAL**) for multi-platform environment specifiers (`conda-lock.yml`, `pixi.lock`). (#15927)
- `conda env create` / `conda env update` check `context.subdir in spec.available_platforms` before calling `spec.env_for(context.subdir)` and raise `PlatformMismatchError` with the platforms the file does cover and a `conda export` command that includes the current platform when the check fails. (#15927)
- `Environment.from_cli` pre-flights every `-f` / `--file` argument and reports all files that do not cover `context.subdir` in a single `PlatformMismatchError` instead of stopping at the first mismatch. (#15927)
- Add `PlatformMismatchError` (subclass of `CondaValueError`) so the wording stays consistent across `conda env create`, `conda env update`, and `Environment.from_cli`. (#15927)
- Rewrite the help text for `conda create`, `conda export`, `conda env create`, and `conda env export` (already an alias of `conda export`) to make lockfile workflows discoverable. The epilog of each command now lists the available input/output formats grouped by category, driven dynamically by the installed environment specifier and exporter plugins. (#15960)

- Add `context.preview` configuration parameter (via `CONDA_PREVIEW` env var or `preview:` in `.condarc`) and `context.preview_enabled(value)` method for opt-in feature previews. (#16010)
- Sharded repodata: extract dependency names and package records from v3 repodata groups in `shard_mentioned_packages` and `iter_records` for compatible solvers. (#16067 via #16078)
- Add startup performance benchmarks (`tests/cli/test_startup_benchmarks.py`) to the CodSpeed benchmark suite, covering import cost, context initialization, parser generation, and module-count guardrails. (#15838)
- Canonicalize deferred deprecation shims in `conda.auxlib.logz` and `conda.common.serialize` to use `deprecated.constant(..., factory=...)`, avoiding eager imports of `conda.common.serialize.json` on cold paths. (#15926)
- Add `factory=` kwarg to `conda.deprecations.deprecated.constant` as a mutually-exclusive alternative to positional `value`. When given, the callable is invoked (and the result cached) only on first access of the deprecated symbol. Enables deprecating symbols whose materialization would otherwise force heavy imports at registration time, without giving up the canonical `deprecated.constant` API.
- Improve initialization time by using stdlib `json` instead of `requests.compat.json` in `conda.common.serialize.json`, avoiding many transitive imports. (#15838)

Bug fixes

- Don't raise `RemoveError` if a missing conda dependency is not being installed or uninstalled by the current transaction; can happen when a conda dependency was installed with pip. (#14050)
- Fix v3 MatchSpec parser regression where PyPI-style extras in `constrains` (e.g. `dask[array]`) crash the classic solver due to a regex anchor bug and a test-only dispatcher override. (#15443 via #16095)
- Run PowerShell reactivate after `conda install`, `update`, `upgrade`, `remove`, and `uninstall` so that `activate.ps1` runs and environment variables set by packages (e.g. proj, GDAL) are available in the same session without manually deactivating and reactivating. (#15643 via #15710)
- Make `conda.notices.cache.clear_cache()` robust against transient Windows file locks. It now rewinds `notices.cache`'s mtime instead of deleting the file (which could silently fail with `PermissionError`), and ignores `OSError` when removing per-channel response caches. This fixes flaky behavior of `tests/cli/test_main_notices.py::test_notices_shown_after_previous_command_error` on Windows. (#15796, #15817, #15839)
- Require `pluggy >=1.6.0` to ensure the `HookImpl.wrapper` attribute is available; older versions caused an `AttributeError` at runtime in the plugin manager. (#15862)
- Fix `Environment.extrapolate` so the solver receives virtual packages for the target platform (e.g. `__glibc` on `linux-*`) instead of the host's. This unblocks multi-platform `conda export` of lockfiles when specs like `python=3.13` are pinned. (#15919)
- Fix `Context._override` to correctly restore `ParameterLoader` (and other non-data descriptor) attributes on exit. The previous implementation shadowed the descriptor in `__dict__` permanently, causing later `reset_context()` calls to ignore env-var changes for the overridden parameter.
- Update the `conda activate`/`conda deactivate` commands return a 0 exit code when run with the `-h`/`--help` flag. (#15826 via #15958)
- Make the windows (bat activator) recognize the `-h`/`--help` flags. (#15826 via #15958)
- Make the windows powershell activator recognize the `-h`/`--help` flags. (#15978)
- When invalidating the channel notices cache, rewind `notices.cache`'s mtime in integer nanoseconds (`stat.st_mtime_ns + os.utime(ns=...)`) and check expiry against `time.time_ns()` so the `NOTICES_DECORATOR_DISPLAY_INTERVAL` boundary check is not subject to float/FILETIME rounding on Windows. (#16003)

- Fix `test_get_index_no_platform_with_offline_cache` failing on conda-forge CI by warming the repodata cache before testing offline reads. (#16041)
- Ensure that reported conda anchor files report case matched paths on Windows. (#16052 via #15983)
- Fix conda doctor --fix package reinstall step (#15740 via #16054)
- Remove `conda-pypi` dependency from the conda recipe to prevent downstream CI breakages. (#16082)
- Validate `MatchSpec` package names against CEP-26 rules, rejecting names with invalid characters or structure. (#16087)
- Fix wheel package extension not found when conda-pypi URLs have a trailing sha256 hash. (#16088 via #16089)
- Update `tests/cli/test_cli_install.py::test_emscripten_forge` to use `https://repo.prefix.dev/emscripten-forge-4x`, the channel URL recommended by the emscripten-forge installation docs. The legacy `https://repo.mamba.pm/emscripten-forge` host has been unreachable, causing the integration test to fail on every CI run across all platforms.

Deprecations

- Accidentally allowed and silently ignored `package[version<1]`-like `MatchSpecs` will be removed in `27.3`. The square bracket syntax must always include `key=value` pairs. (#15443)
- Mark `--env-spec`/`--environment-specifier` as pending deprecation in favor of `--format` (#15794 via #15834)
- Deprecate `conda.models.channel.ChannelType`. `Channel` is now a regular class; use the builtin `type(Channel)` if you need to inspect the metaclass. (#15867)
- Mark the `conda.instructions` module as pending deprecation for removal in 27.3, as a follow-up to the removal of `conda.plan`. (#15874)
- Mark `conda.gateways.logging.initialize_std_loggers` as pending deprecation to be removed in 27.3. (#15876)
- Mark `conda.stdout`, `conda.stderr`, `conda.stdoutlog`, `conda.stderrlog`, and `conda.stdout.verbose` loggers as pending deprecation to be removed in 27.3. (#15876)
- Define private symbols, modules (i.e. starts-with-underscore _) as not requiring deprecation. Library users should avoid these symbols. See deprecations.md. (#15977)

Docs

- Start dev-oriented documentation for type hints and how we use them. (#15768)
- Add tutorial and API reference for the new exception observer plugin hook. (#15910)
- Document `available_platforms` and `env_for` on `EnvironmentSpecBase`. (#15927)
- Add information on conda-lockfiles to Managing environments page. (#16086)
- Document `factory=` kwarg on `deprecated.constant` in the deprecations dev-guide.

Other

- Add type hints to `conda.common.io`. (#15773)
- Provide better error messages for when an environment file is not detected. (#15718 via #15793)
- Provide better error messages for when too many plugins can read an environment files. (#15718 via #15793)
- Require `conda-libmamba-solver >=26.4.0`. (#15950)
- Require `conda-pypi >=0.8.0` in the conda recipe. (#15987)

- Bump `conda-libmamba-solver` minimum version to `>=26.4.1`. (#16043)
- Make the Linux integration jobs resilient to Azure-hosted Ubuntu apt mirror outages by switching to the canonical `archive.ubuntu.com` mirror and enabling apt retries (`Acquire::Retries=3`). Also install shells (`ash`, `csh`, `fish`, `tcsh`, `zsh`) non-interactively with `--no-install-recommends` to shrink the download footprint.

Contributors

- @agriyakhetarpal
- @conda-bot
- @costajohnt made their first contribution in https://github.com/conda/conda/pull/15801
- @dashagurova made their first contribution in https://github.com/conda/conda/pull/16051
- @danyeaw
- @dholth
- @jaimergp
- @jezdez
- @jsmolic made their first contribution in https://github.com/conda/conda/pull/16082
- @kathatherine
- @kenodegard
- @prady0t made their first contribution in https://github.com/conda/conda/pull/15950
- @smartcoder0777 made their first contribution in https://github.com/conda/conda/pull/15710
- @soapy1
- @travishathaway
- @VedantMadane made their first contribution in https://github.com/conda/conda/pull/15773

4.5.15 26.3.2 (2026-04-10)

Bug fixes

- Revert inline environment activation for and subsequent optimizations to `conda run` (#15534, #15672), due to regressions with post-link scripts. (#15896)

Contributors

- @agriyakhetarpal

4.5.16 26.3.1 (2026-04-03)

Enhancements

- Record the originating hook for each plugin, enabling `conda info` and diagnostics to show where plugins were registered from. (#15840)

Deprecations

- No longer search for `conda-*` executables on `PATH` when building the CLI parser. (#15840)
- Mark `conda.cli.find_commands` as pending deprecation for removal in 27.3. (#15840)
- Mark `conda.utils.sys_prefix_unfollowed` as pending deprecation for removal in 27.3. (#15840)

- Mark `conda.exceptions.CommandNotFoundError` as pending deprecation for removal in 27.3. (#15840)

Contributors

- @kenodegard

4.5.17 26.3.0 (2026-03-30)

Enhancements

- Improve handling of Windows CMD.EXE activation for environment paths that contain the `^` (caret) character. (#12558 via #15628)
- Add support for `PackageRecord.requested_specs`. This will store all the MatchSpec objects that were involved in requesting a package (instead of a single, pre-merged object). To that end, two new attributes in `conda.core.solve.Solver`, `unmerged_specs_to_add` and `unmerged_specs_to_remove`, store the list of input specs as received, instead of always attempting to merge them. (#15111 via #15110)
- Pass unmerged MatchSpec objects to `pre-solve` plugins (instead of pre-merged). (#15110)
- Add option to disable redefining Fish shell prompt functions in conda init. (#15545 via #15549)
- `conda create --file <filename>` now supports environment.yaml files as well as requirements.txt and explicit environment exports, and detects name and prefix automatically when `--name` / `--prefix` are omitted. (#15573 via #15770)
- Add `description` and `environment_format` metadata fields to environment plugin types (`CondaEnvironmentSpecifier` and `CondaEnvironmentExporter`). (#15577 via #15763)
- Optimize S3 downloads to avoid an extra file copy through a temporary buffer. (#15636)
- Add path customization parameters to `tmp_env` fixture (`name`, `path_prefix`, `path_infix`, `path_suffix`) that pass through to `path_factory`. Add "parts mode" to `path_factory` where specifying `prefix`, `infix`, or `suffix` triggers UUID defaults for unspecified parts, enabling easier special character path testing. (#15650)
- Add `PackagesNotFoundInChannelsError` and `PackagesNotFoundInPrefixError` as specialized cases of `PackagesNotFoundError`, to distinguish between packages missing from configured channels and packages missing from a specific environment prefix. (#12197 via #15653)
- Always include conda-meta manifest file size in `PrefixRecord.package_size()`, so packages with no files (such as metapackages, mutex packages) display a non-zero size instead of "0 B" in `conda list --size`. (#15681 via #15656)
- Speed up `conda run` by dropping the shell hook subprocess and relying on inline activators. (#14993 via #15672)
- `conda remove <spec>` with the classic solver now exits early if the spec to be removed does not exist in the environment. (#15697)
- `conda search --envs <package>` now raises an error when the package is not found in any environment. (#15698)
- Add support for defining aliases of environment spec plugins. (#15701)
- Add pattern matching support to environment specifier and exporter plugins via `default_filenames` field. Plugins can now use wildcards (e.g., `*.conda-lock.yml`) for filename-based detection. (#15717 via #15719)
- `conda create|install -f` is now recognized as an alias to `conda create|install --file`. (#15790)
- Improve log messages when conda auto-detects the applicable environment file format. (#15736)

Bug fixes

- Fix `conda config --show-sources` displaying `frozendict.frozendict(...)` for `channel_settings` entries instead of readable dict format. (#15505 via #15789)
- Do not include pip-installed packages in the list of explicit packages when building a `conda.models.Environment` with `conda.models.Environment.from_prefix`. (#15619)
- Use bundled Windows stub executables (`conda/shell/cli-32.exe`, `conda/shell/cli-64.exe`) for creating entry points instead of relying on `Scripts/conda.exe` so that entry point creation works when conda is not installed as a conda package (e.g. tests via `python -m pytest`). Select the correct exe for the target env via a platform-to-stub mapping; for win-arm64 use `cli-64.exe` (relying on Windows x64-on-ARM64 emulation). Use target platform for `conda init` as well, fixing cross-architecture installs (e.g. win-32 host creating a win-64 env). (#15678)
- Use case-insensitive matching when finding Python metadata files on Windows to make sure they are correctly identified as conda packages. (#15153 via #15725)
- Always include the `dependencies` field when exporting an environment.yaml-style environment, including when it is empty, for CEP-24 compliance. (#15753)
- Fix a regression first introduced in #14661 that adversely affected package resolution for custom multichannels. (#15750 via #15776)
- Allow `conda search --offline` to use the local channel cache. (#15781 via #15795)
- Fix flaky `test_token_not_present_in_conda_create` test by removing unreliable assertion that expected `/t/<TOKEN>/` in output. The test now focuses on the critical security guarantee (raw tokens must not leak) while new unit tests verify the `TokenURLFilter` integration. (#13939)
- Fix `win-arm64` subdir detection on native Windows ARM64. `platform.machine()` returns uppercase `ARM64` on Windows, which did not match the lowercase entries in `non_x86_machines`, causing the platform to incorrectly resolve to `win-64`. (#15716)
- Ignore JSON metadata paths beginning with `._` (e.g. AppleDouble files) when loading prefix data, so spurious files from macOS or external drives do not break metadata loading. (#15652 via #15664)

Deprecations

- Mark `conda.common.io.env_var`, `conda.common.io.env_vars`, and `conda.common.io.env_unmodified` as pending deprecation to be removed in 27.3. In tests, prefer `monkeypatch.setenv` / `monkeypatch.delenv` instead. (#14095 via #15728)
- Mark `conda run --dev` as pending deprecation, to be removed in 27.3. Set `PYTHONPATH` to the conda source root instead. (#15641)
- `-f` is no longer an alias for `--force`. (#15790)
- Remove `conda info --root`. Use `conda info --base` instead. (#15791)
- Remove `conda info -l/--license`. (#15791)
- Remove `conda_env` module. (#15791, #15816)
- Remove `conda._vendor` module. Conda no longer vendors dependencies. (#15791)
- Remove `conda.base.context._first_writable_envs_dir`. Use `conda.gateways.disk.create.first_writable_envs_dir` instead. (#15791)
- Remove `conda.base.context.add_plugin_setting`. Use `conda.plugins.config.PluginConfig.add_plugin_setting` instead. (#15791)
- Remove `conda.base.context.Context.auto_activate_base`. Use `conda.base.context.Context.auto_activate` instead. (#15791)

- Remove `conda.base.context.Context.conda_exe`. Use `conda.base.context.Context.conda_exe_vars_dict` instead. (#15791)
- Remove `conda.base.context.Context.pip_interop_enabled`. Use `conda.base.context.Context.prefix_data_interoperability` instead. (#15791)
- Remove `conda.base.context.get_plugin_config_data`. Use `conda.base.context.context.plugins.raw_data` instead. (#15791)
- Remove `conda.base.context.remove_all_plugin_settings`. Use `conda.plugins.config.PluginConfig.remove_all_plugin_settings` instead. (#15791)
- Remove `conda.base.context.validate_prefix_name`. Use `conda.core.prefix_data.PrefixData.{validate_name,from_name}` instead. (#15791)
- Remove `conda.cli.common.validate_prefix_is_writable`. Use `conda.core.prefix_data.PrefixData.assert_writable` instead. (#15791)
- Remove `conda.cli.common.validate_prefix`. Use `conda.core.prefix_data.PrefixData.assert_environment` instead. (#15791)
- Remove `conda.cli.install.check_prefix`. Use `conda.core.prefix_data.PrefixData.{exists,validate_path,validate_name}` instead. (#15791)
- Remove `conda.cli.install.print_activate`. Use `conda.cli.common.print_activate` instead. (#15791)
- Remove `conda.cli.install.validate_new_prefix`. Use `conda.core.prefix_data.PrefixData.{exists,validate_path}` instead. (#15791)
- Remove `conda.cli.install.validate_prefix_exists`. Use `conda.core.prefix_data.PrefixData.exists` instead. (#15791)
- Remove `conda.cli.main_info.get_info_components`. Use `conda.cli.main_info.iter_info_components` instead. (#15791)
- Remove `conda.cli.main_rename.check_protected_dirs`. Use `conda.core.prefix_data.PrefixData.validate_path` instead. (#15791)
- Remove `conda.cli.main_rename.validate_src`. Use `conda.core.prefix_data.PrefixData.{validate_path,validate_name}` instead. (#15791)
- Remove `conda.common.io._logger_lock`. Use `logging._lock` instead. (#15791)
- Remove `conda.common.pkg_formats` module. Use `conda.plugins.prefix_data_loaders.pypi.pkg_format` instead. (#15791)
- Remove `conda.common.url.hex_octal_to_int`. Use `int(..., 16)` instead. (#15791)
- Remove `conda.core.index._supplement_index_with_system`. Use `conda.core.Index.reload` instead. (#15791)
- Remove `conda.core.index.check_allowlist`. Use `conda.base.context.validate_channels` instead. (#15791)
- Remove `conda.core.path_actions._Action`. Use `conda.core.path_actions.Action` instead. (#15791)
- Remove `conda.core.prefix_data.get_python_version_for_prefix`. Use `conda.core.prefix_data.PrefixData.get("python").version` instead. (#15791)
- Remove `conda.core.prefix_data.PrefixData._load_site_packages`. Use `conda.plugins.prefix_data_loaders.pypi.load_site_packages` instead. (#15791)

- Remove `conda.core.prefix_data.PrefixData._pip_interop_enabled`. Use `conda.core.prefix_data.PrefixData.interoperability` instead. (#15791)
- Remove `conda.core.prefix_data.PrefixData._python_pkg_record`. Use `conda.core.prefix_data.PrefixData.get("python")` instead. (#15791)
- Remove `conda.core.prefix_data.python_record_for_prefix`. Use `conda.core.prefix_data.PrefixData.get("python")` instead. (#15791)
- Remove `conda.env.env.get_filename`. (#15791)
- Remove `conda.env.specs.binstar` module. (#15791)
- Remove `conda.env.specs.detect(name,directory)`. Specify the full path in filename instead. (#15791)
- Remove `conda.env.specs.get_spec_class_from_file`. Use `conda.base.context.plugin_manager.detect_environment_specifier` instead. (#15791)
- Remove `conda.env.specs.requirements.RequirementsSpec._valid_file`. (#15791)
- Remove `conda.env.specs.requirements.RequirementsSpec._valid_name`. (#15791)
- Remove `conda.env.specs.requirements.RequirementsSpec.environment`. Use `conda.env.specs.requirements.RequirementsSpec.env` instead. (#15791)
- Remove `conda.env.specs.requirements.RequirementsSpec.name`. (#15791)
- Remove `conda.env.specs.requirements.RequirementsSpec(name)`. (#15791)
- Remove `conda.gateways.disk.read.read_python_record`. Use `conda.plugins.prefix_data_loaders.pypi.pkg_format.read_python_record` instead. (#15791)
- Remove `conda.gateways.disk.test.is_conda_environment`. Use `conda.core.prefix_data.PrefixData.is_environment` instead. (#15791)
- Remove `conda.misc.touch_nonadmin`. Use `conda.core.prefix_data.PrefixData.set_nonadmin` instead. (#15791)
- Remove `conda.models.records.PackageRecord.schannel`. Use `conda.models.records.PackageRecord.channel_name` instead. (#15791)
- Remove `conda.plan` module. (#15791)
- Remove `conda.testing.fixtures.session_capsys`. (#15791)
- Remove `conda.utils.unix_path_to_win`. Use `conda.common.path.unix_path_to_win` instead. (#15791)
- Remove loading conda subcommands via executables. Use the plugin system instead. (#15791)
- Rename `conda.activate._Activator.hook(auto_activate_base)` to `auto_activate`. (#15791)
- Rename `conda.core.prefix_data.PrefixData(pip_interop_enabled)` to `interoperability`. (#15791)
- Remove `conda.trust` module. The implementation has been moved to `conda-content-trust`. (#15649)
- Remove built-in `signature-verification` post-solve plugin. Signature verification is now provided by the `conda-content-trust` plugin when installed. (#15649)
- Rename `conda.core.PrefixActions` to `conda.core.PrefixActionGroup`. (#15824)

Docs

- Update documentation for `conda create --file` env specifier support in several places. (#15573 via #15770)
- Remove documentation for RPM and DEB packages, which are no longer supported. (#13896 and #14826 via #15694)
- Update advice on improving conda performance in the Concepts section of the conda User Guide. (#15780)
- Update pip interoperability documentation page with historical note and link to conda-pypi documentation. (#15815)
- Fix broken links on contributing documentation pages. (#15729)
- Remove Discourse logo from documentation build. (#15806)
- Expand `conda search` documentation to include more search examples. (#15774)

Other

- Add comprehensive test cases to characterize environment name validation behavior for Windows special characters (`!`, `=`, `^`, `%`, `(`, `)`). Tests confirm that `!` now works (previously fixed in #14607) and identify `^` as broken. (#12558 via #15628)
- Remove JLAP (JSON Line-oriented Patch) experimental support. Zstd-compressed repodata (`.json.zst`) functionality is preserved and continues to work as before. (#15233 via #15693)
- Remove file extension validation from environment spec plugins. Requirements plugin now validates that all lines are valid matchspecs to avoid collisions with other plugins (e.g., environment.yaml). (#15714)
- Remove internal use of `boltons.setutils.IndexedSet()`. It has been replaced with either sequences or sets. (#15742 via #15744)
- Improve the error message when an environment spec plugin cannot handle a file. (#15718 via #15792)
- Add `pre-commit` updates to Dependabot and keep automated dependency updates managed via `.github/dependabot.yml`. (#15814)

Contributors

- @agriyakhetarpal
- @conda-bot
- @danyeaw
- @ELundby45 made their first contribution in https://github.com/conda/conda/pull/15716
- @jaimergp
- @jezdez
- @kathatherine
- @kenodegard
- @ForgottenProgramme
- @ryanskeith
- @schuylermartin45
- @soapy1
- @crowecawcaw made their first contribution in https://github.com/conda/conda/pull/15636
- @travishathaway

- @dependabot[bot]
- @lsoksane made their first contribution in https://github.com/conda/conda/pull/15549
- @pre-commit-ci[bot]
- @shenhaofang made their first contribution in https://github.com/conda/conda/pull/15664

4.5.18 26.1.1 (2026-02-17)

Bug fixes

- Fix `conda doctor --fix` failing with `install() missing 1 required positional argument: 'parser'` when attempting to fix packages with missing files. (#15669)

Contributors

- @jezdez

4.5.19 26.1.0 (2026-01-29)

Enhancements

- Allow `custom_multichannels.defaults` to override `default_channels`. Continuation of effort to devendor conda and remove hardcoded "defaults" channel. (#14661)
- Defer `PrefixData` record instantiation until they are accessed for faster query performance. (#14811)
- Add support for Python 3.14. (#15080, #15618)
- Speed up the context initialization process by caching condarc file reads. (#15150)
- Raise a `NoChannelsConfiguredError` when attempting to solve an environment with no channels configured, and provide a helpful message to guide users in adding channels. (#15262 via #15345)
- Add `conda.cli.condarc.ConfigurationFile` class to expose configuration file manipulation interface programmatically. This allows other Python interfaces to read and modify conda configuration files without shelling out to `conda config` commands. (#15238 via #15376)
- Add `-s` as a shorthand alias for `conda run --no-capture-output`. (#15426)
- `conda run` now deactivates the environment on exiting the command. (#15427)
- Rename `conda.models.enums.PathType` enum to `conda.models.enums.PathEnum` to resolve a naming conflict with the `PathType` type annotation. This change follows modern Python conventions and establishes the `*Enum` suffix as the pattern for enums in the codebase. (#15428)
- Validate plugin-provided HTTP headers and raise `PluginError` if forbidden headers are detected, preventing plugins from compromising conda's network stack. (#15405 via #15436)
- Add `conda_package_extractors` plugin hook to allow plugins to register custom package archive extractors. (#15524 via #15441)
- Add `-O` as a flag alias for `--override-channels`. (#15499)
- Add reusable `http_test_server` fixture to `conda.testing` module for serving mock conda channels and remote files in tests. Fixture uses standard `@pytest.mark.parametrize()` with `indirect=True` for configuring directories, supporting both pre-existing directories and dynamic content generation. Provides clean API with `get_url()` helper method and `directory` attribute. (#15528 via #15529)
- Speed up `conda run` by inlining environment activation instead of using a subprocess. (#15534)
- Add a `--size` flag to the `conda info --envs` and `conda env list` commands to display disk usage for each environment. (#8040 and #15544)

- Add `PrefixRecord.package_size()` and `PrefixData.size` for programmatic access to conda-managed disk usage. (#15544)
- Add `--size` flag to `conda list` to show environment size and package sizes. (#15544)
- Improve error messages for HTTP 403 (Forbidden) responses with context-aware guidance on authentication and permissions. (#15594)
- Improve error message when `conda activate` fails on Windows due to unwritable `%TEMP%` directory. (#15601 via #15602)
- Package extension detection is now dynamic based on registered `conda_package_extractors` plugins. (#15629)
- Allow `conda create --download-only` to run without specifying `-n/--name` or `-p/--prefix`, similar to `--dry-run`. (#15633)
- Add `--fix` flag to `conda doctor` with health check fix capabilities. (#15530)
- Improve error message when `conda env export` is invoked with an invalid platform. (#15561)

Bug fixes

- Stop `conda run` from adding an additional new line after printing to `stdout` or `stderr`. (#11763 via #15539)
- Non-conda installed Python packages now show up in `conda list` in free-threading environments. (#14674)
- Fix Python packages not being listed as from the `pypi` channel if the `site-packages` directory includes a symlink. (#14674)
- Replace deprecated `codecs.open()` with built-in `open()` for Python 3.14 compatibility. (#15080)
- Fix parallel conda command failures on Windows (e.g., `start /b`) by using GUIDs for unique temp filenames instead of `%RANDOM%`. (#15169 via #15598)
- Sanitize channel config input to remove null values. (#15268 via #15323)
- Fix `conda list --fields dist_str` representation. (#15420)
- Add the `--` separator such that `conda run` can properly separate executable options from conda options when invoking commands. Now, executable options are correctly parsed and handled. (#15432)
- Raise an error when no command is provided to `conda run`, instead of proceeding with an empty command. (#15437)
- Do not expose `__cuda` virtual package when no CUDA version was detected. (#15500)
- Ensure an appropriate error code is returned when `conda run` is invoked with an invalid environment name or prefix. (#15534)
- Record `size_in_bytes` for generated entry points and compiled `.pyc` files in the prefix manifests. (#15544)
- Allow `conda env update` to execute an update on an environment for environment files that do not have conda dependencies. (#15569 via #15570)
- Terminate activation early on Windows when activation scripts fail. (#15616)
- Fix package installation from repodata v3 channels by extracting the correct filename from the URL instead of using the repodata key. This resolves issues when installing packages where the repodata key differs from the actual filename (e.g., wheel packages where the key uses underscores but the filename uses hyphens). (#15620 via #15626)
- Fix incorrect source location in deprecation warnings when multiple constants are deprecated in the same module. (#15623 via #15624)
- Fix `conda clean --tarballs` not cleaning up `.partial` download files. (#15634 via #15635)

- Decode SHA256 fields in Python's `RECORD` metadata files. (#15591)
- Fix `CondaJSONEncoder` to support serializing `frozendicts`. (#15532)
- Fix argument parser aliases incorrectly defined as a string instead of a tuple. (#15418)
- Raise an error when attempting to deactivate an environment that is not currently active. (#15609)

Deprecations

- Mark `conda.exception_handler.ExceptionHandler._upload` as pending deprecation, to be removed in 27.3. (#13900 via #14698)
- Mark `conda.exception_handler.ExceptionHandler._ask_upload` as pending deprecation, to be removed in 27.3. (#13900 via #14698)
- Mark `conda.exception_handler.ExceptionHandler._execute_upload` as pending deprecation, to be removed in 27.3. (#13900 via #14698)
- Mark `conda.exception_handler.ExceptionHandler._post_upload` as pending deprecation, to be removed in 27.3. (#13900 via #14698)
- Mark private helper functions in `conda.cli.main_config` (`_key_exists`, `_get_key`, `_set_key`, `_remove_item`, `_remove_key`, `_read_rc`, `_write_rc`, `_validate_provided_parameters`) as pending deprecation, to be removed in 27.3. Use the new `ConfigurationFile` class instead. (#15376)
- Mark `conda.models.enums.PathType` as pending deprecation, to be removed in 27.3. Use `conda.models.enums.PathEnum` instead. (#15428)
- Mark `conda.utils.massage_arguments` as pending deprecation, to be removed in 27.3. (#15439)
- Mark `conda.gateways.disk.create.extract_tarball` as pending deprecation, to be removed in 27.3. Use `conda.base.context.context.plugin_manager.extract_package` instead. (#15524 via #15441)
- Mark the `use_system_tmp_path` argument of `conda.utils.wrap_subprocess_call(use_system_tmp_path=..)` as pending deprecation, to be removed in 27.3. Use the TMPDIR, TEMP, or TMP environment variables to set the system temporary directory location. (#15440 via #15442)
- Mark private helper functions in `conda.common.serializers` (`_yaml_round_trip`, `_yaml_safe`) as pending deprecation, to be removed in 27.3. Use `conda.common.serialize.yaml._yaml()` instead. (#15503)
- Mark `conda.common.serializers.yaml_round_trip_load` and `conda.common.serializers.yaml_safe_load` as pending deprecation, to be removed in 27.3. Use `conda.common.serialize.yaml.load()` instead. (#15503)
- Mark `conda.common.serializers.yaml_round_trip_dump` and `conda.common.serializers.yaml_safe_dump` as pending deprecation, to be removed in 27.3. Use `conda.common.serialize.yaml.dump()` instead. (#15503)
- Mark `conda.cli.actions.NullCountAction._ensure_value` as pending deprecation, to be removed in 27.3. (#15512)
- Mark `conda.exports.input` as pending deprecation, to be removed in 27.3. Use `builtins.input` instead. (#15523 via #15526)
- Mark `conda.exports.StringIO` as pending deprecation, to be removed in 27.3. Use `io.StringIO` instead. (#15523 via #15526)
- Mark `conda.exports.PY3` as pending deprecation, to be removed in 27.3. Python 2 is no longer supported. (#15523 via #15526)
- Mark `conda.exports.string_types` as pending deprecation, to be removed in 27.3. Use `str` instead. (#15523 via #15526)

- Mark `conda.exports.text_type` as pending deprecation, to be removed in 27.3. Use `str` instead. (#15523 via #15526)
- Mark `conda.exports.DEFAULT_CHANNELS` as pending deprecation, to be removed in 27.3. Use `conda.base.constants.DEFAULT_CHANNELS` instead. (#15523 via #15526)
- Mark `conda.exports.DEFAULT_CHANNELS_UNIX` as pending deprecation, to be removed in 27.3. Use `conda.base.constants.DEFAULT_CHANNELS_UNIX` instead. (#15523 via #15526)
- Mark `conda.exports.DEFAULT_CHANNELS_WIN` as pending deprecation, to be removed in 27.3. Use `conda.base.constants.DEFAULT_CHANNELS_WIN` instead. (#15523 via #15526)
- Mark `conda.exports.PREFIX_PLACEHOLDER` as pending deprecation, to be removed in 27.3. Use `conda.base.constants.PREFIX_PLACEHOLDER` instead. (#15523 via #15526)
- Mark `conda.exports._PREFIX_PLACEHOLDER` as pending deprecation, to be removed in 27.3. Use `conda.base.constants.PREFIX_PLACEHOLDER` instead. (#15523 via #15526)
- Mark `conda.exports.prefix_placeholder` as pending deprecation, to be removed in 27.3. Use `conda.base.constants.PREFIX_PLACEHOLDER` instead. (#15523 via #15526)
- Mark `conda.exports.CondaError` as pending deprecation, to be removed in 27.3. Use `conda.CondaError` instead. (#15523 via #15526)
- Mark `conda.exports.CondaHTTPError` as pending deprecation, to be removed in 27.3. Use `conda.exceptions.CondaHTTPError` instead. (#15523 via #15526)
- Mark `conda.exports.CondaOSError` as pending deprecation, to be removed in 27.3. Use `conda.exceptions.CondaOSError` instead. (#15523 via #15526)
- Mark `conda.exports.LinkError` as pending deprecation, to be removed in 27.3. Use `conda.exceptions.LinkError` instead. (#15523 via #15526)
- Mark `conda.exports.LockError` as pending deprecation, to be removed in 27.3. Use `conda.exceptions.LockError` instead. (#15523 via #15526)
- Mark `conda.exports.PaddingError` as pending deprecation, to be removed in 27.3. Use `conda.exceptions.PaddingError` instead. (#15523 via #15526)
- Mark `conda.exports.PathNotFoundError` as pending deprecation, to be removed in 27.3. Use `conda.exceptions.PathNotFoundError` instead. (#15523 via #15526)
- Mark `conda.exports.CondaFileNotFoundError` as pending deprecation, to be removed in 27.3. Use `conda.exceptions.PathNotFoundError` instead. (#15523 via #15526)
- Mark `conda.exports.UnsatisfiableError` as pending deprecation, to be removed in 27.3. Use `conda.exceptions.UnsatisfiableError` instead. (#15523 via #15526)
- Mark `conda.base.context.report_errors` as pending deprecation, to be removed in 27.3. (#13900 via #15580)
- Mark `conda.base.constants.ERROR_UPLOAD_URL` as pending deprecation, to be removed in 27.3. (#13900 via #15558)
- Mark `conda.base.context.error_upload_url` as pending deprecation, to be removed in 27.3. (#13900 via #15558)
- Mark `conda.exception_handler.ExceptionHandler.error_upload_url` as pending deprecation, to be removed in 27.3. (#13900 via #15558)
- Mark `conda.common.path.is_package_file` as pending deprecation, to be removed in 27.3. Use `conda.base.context.context.plugin_manager.has_package_extension` instead. (#15629)

- Mark `conda.common.path.KNOWN_EXTENSIONS` as pending deprecation, to be removed in 27.3. Use `conda.common.path.strip_pkg_extension` instead. (#15629)
- Mark `conda.misc.url_pat` as pending deprecation, to be removed in 27.3. Use `conda.misc._get_url_pattern()` instead. (#15629)
- Mark `conda.base.constants.CONDA_PACKAGE_EXTENSIONS` as pending deprecation, to be removed in 27.3. Use `conda.base.context.context.plugin_manager.get_package_extractors()` instead. (#15629)
- Mark `conda.base.constants.CONDA_PACKAGE_PARTS` as pending deprecation, to be removed in 27.3. The `.part` suffix has not been used since 2014; use `conda.base.constants.PARTIAL_EXTENSION` (`.partial`) with `str.removesuffix()` instead. (#15634 via #15635)

Docs

- Add documentation for `custom_multichannels` setting, including its ability to override the `defaults` multichannel. (#14661)
- Add documentation for the new `conda_package_extractors` plugin hook. (#15524 via #15441)
- Document how to configure temporary file locations using `TMPDIR`/`TEMP`/`TMP` environment variables. Add comprehensive guide covering container environments, HPC systems, and troubleshooting read-only filesystem errors. (#15440 via #15442)
- Add manual development environment setup instructions. (#15482)
- Add comprehensive guide for using HTTP test server fixture in `docs/source/dev-guide/writing-tests/http-test-server.md`. Includes usage examples, API reference, and troubleshooting tips. (#15528 via #15529)
- Update community support link in getting-started guide. (#15554)
- Fix duplicated API Guide in documentation. (#15555)
- Remove setuptools and wheel from `add_pip_as_python_dependency` configuration description. (#15622)

Other

- Disable uploading error reports. (#13900 via #14698)
- Deprecation warnings now raise errors during testing to better track and address them. (#15525)

Contributors

- @agriyakhetarpal
- @degerahmet made their first contribution in https://github.com/conda/conda/pull/14698
- @Bhanuu01 made their first contribution in https://github.com/conda/conda/pull/15554
- @barabo made their first contribution in https://github.com/conda/conda/pull/15418
- @conda-bot
- @danyeaw
- @dholth
- @gayanMatch made their first contribution in https://github.com/conda/conda/pull/15596
- @giacomo-ciro made their first contribution in https://github.com/conda/conda/pull/15428
- @jaimergp
- @jezdez

- @jjhelmus
- @kenodegard
- @ForgottenProgramme
- @matthewfeickert made their first contribution in https://github.com/conda/conda/pull/15622
- @opoplawski
- @pirzada-ahmadfaraz made their first contribution in https://github.com/conda/conda/pull/15602
- @ryanskeith
- @soapy1
- @travishathaway
- @vshevchenko-anaconda made their first contribution in https://github.com/conda/conda/pull/15376
- @dependabot[bot]
- @pre-commit-ci[bot]

4.5.20 25.11.1 (2025-12-12)

Bug fixes

- Do not expose `__cuda` virtual package when no CUDA version was detected. (#15500)

Contributors

- @jaimergp
- @kenodegard

4.5.21 25.11.0 (2025-11-20)

Enhancements

- Add three new fields to `conda.plugins.types.CondaVirtualPackage` class:
  - `override_entity`: Control which field (version or build number) can be overridden
  - `empty_override`: Specify behavior when override value is empty
  - `version_validation`: Validate override version values These fields abstract and generalize the version/build number override logic into the `CondaVirtualPackages` class, removing the need for individual virtual packages to implement their own override handling. (#15248)
- Add a new `override_virtual_packages` (with alias `virtual_packages`) key to `condarc`. (#15156 via #15318)
- Add `.created` and `.last_modified` properties to `PrefixData`, which provide information about the creation and last modification times of a given environment, respectively. (#14308 via #15329)
- Add a new `envs_details` field to the `conda info --json` output to inspect properties of the registered environments. This is also part of the `conda info --envs --json` and `conda env list --json` outputs. (#15330)
- `conda.plugins.reporter_backends.JSONReporterRenderer.envs_list` now allows `dict[str, dict[str, str | bool]]`, which is passed directly to the JSON dumper. (#15330)

Bug fixes

- Ensure `MatchSpec` serialization does not contain orphan = separator when the `version` field is set to `*`. (#14357 via #15294)
- Require `conda-libmamba-solver >=25.4.0`. (#15289)
- Fix `Channel.urls()` to respect explicit `subdirs` parameter when the channel has a platform defined. (#14258)
- Include `platform` in `Channel` equality and hash checks. (#14259 via #15296)
- Fix where `conda search --json --use-index-cache <package>` could fail to encode message as json, when cache is empty. (#15308)
- Adjust whitespace in `conda info --envs` header. (#15330)
- Fix plugin settings YAML serialization to use valid nested YAML format for plugin settings in `conda config --describe` output. (#15339 via #15341)
- Fix several bugs related to Python 3.14 compatibility. (#15079, #15088 via #15340)
- Ignore and warn user if the `PATH` environment variable is set in the environment config. Conda should not enable modifying the `PATH` to any arbitrary value. (#15342)
- Restrict the JSON output to only the `envs` list when `conda info --envs --json` and `conda env list --json` are called. (#15353)
- Produce a warning when a user tries to add a `PATH` environment variable to their environment configuration. Adding reserved env vars like `PATH` to the environment variables config for an environment may produce unexpected results in conda. (#15292 via #15354)
- Fix zsh parse error on Windows. (#13915, #9922 via #15335)
- Remove `return` from `try-finally` per PEP-765. (#15370 via #15371)

Deprecations

- Mark `conda.plugins.hookspec.spec_name` as pending deprecation, to be removed in 26.9. Use `conda.base.constants.APP_NAME` instead. (#15255)
- Mark `conda.cli.main_compare.get_packages` as pending deprecation, to be removed in 26.9. Use `conda.core.prefix_data.PrefixData.map_records` instead. (#15387)

Docs

- Expand contributor documentation. (#15208)
- Reference `conda.plugins.types.*` instead of re-exports from `conda.plugins`. (#15307)
- Use `--name` more consistently in getting started docs. (#15346)

Other

- Reduce warnings in tests. (#15307)
- Performance improvements for the `conda env list` command, which now defers retrieving environment information until necessary. (#15320)
- Stop using assertions in non-testing code, and enable the corresponding linting rules. (#15357 via #15361)

Contributors

- @agriyakhetarpal
- @conda-bot
- @danyeaw made their first contribution in https://github.com/conda/conda/pull/15208
- @dholth
- @jaimergp
- @jezdez
- @jakirkham
- @kathatherine
- @kenodegard
- @ForgottenProgramme
- @lang-m made their first contribution in https://github.com/conda/conda/pull/13165
- @ryanskeith
- @hoxbro made their first contribution in https://github.com/conda/conda/pull/15325
- @soapy1
- @sumanth-manchala made their first contribution in https://github.com/conda/conda/pull/14179
- @travishathaway
- @dependabot[bot]
- @pre-commit-ci[bot]

4.5.22 25.9.1 (2025-10-10)

Bug fixes

- Require `conda-libmamba-solver >=25.4.0`. (#15289)
- Fix `conda.models.environment.Environment.from_cli`'s spec grouping to correctly categorize `@EXPLICIT` URLs. (#15306 via #15309)

Contributors

- @dholth
- @jakirkham
- @kenodegard

4.5.23 25.9.0 (2025-09-26)

Special Announcement

Following feedback from conda users last year about the pre-configuration of the conda code base to favor channels from Anaconda Inc, we've finished removing hardcoding Anaconda's channels as the default set of channels in the conda source code.

It is now up to providers of the conda distributions, such as miniforge or Anaconda (including miniconda), to pre-configure their preferred channels, e.g. by running the necessary `conda config --set channels` command.

Enhancements

- Add conda doctor check for if the pinned file is ill-formatted. (#7978 via #15022)
- If an HTTP error response includes a field named `detail`, relay that as the error message. Stop uppercasing the error message. (#15021 via #15037)
- Raise exceptions to prevent renaming or removing the environment specified by `default_activation_env`. (#15061)
- Make `conda list REGEX` raise an exception with exit code 1 when the regular expression does not match any packages in the target environment. (#15074 via #15075)
- Add a bytes cache to support sharded repodata. (#15083)
- Add a "file locking" health check to `conda doctor`. (#13921 & #14948 via #15114)
- Add a new environment spec plugin, called `cep-24`, that implements the `environment.yml` specification in CEP 24. This is enabled by default. (#15120)
- Change the `environment.yml` plugin to be disabled by default. Users can explicitly use it by setting the `--environment-specifier` flag. (#15120)
- Add `conda.models.environment.Environment.extrapolate` to extrapolate the environment for a different platform. (#15157 via #15183)
- Add `CondaEnvironmentExporter.multiplatform_export` to support export formats with multiple platforms. (#15159 via #15253, #15261)
- Update `dev/start/dev\start.bat` scripts to allow choosing between `miniconda` and `miniforge` installers via `-i/--installer` flag, `installer_type` in `~/.condarc` configuration, or interactive prompt when not specified. (#15084 via #15163)
- Report environments marked as frozen (CEP 22) with `+` in `conda info --envs` and `conda env list`. (#15213)

Bug fixes

- Slash-normalize `PrefixRecord.files` entries on Windows to emit correct output for `conda list` in environments created by (micro)mamba. (#14523)
- Silence error in `fish` when undefined environment variables are unset. (#14659 via #14660, #15102)
- Fix `validate_subdir_config` cli validation – only search config file context sources for subdir configuration. (#15020 via #15038)
- Fix `conda install/create/update` commands to not ignore CLI arguments if an explicit package is specified to install. (#15028)
- Override `create_default_packages` setting for specs originating from files. (#15041)
- Don't install explicit packages if invoked with `conda update`. (#15044)
- Update `conda create --clone` to disallow users to specify a `--file` CLI argument. (#15702 via #15073)
- Ignore explicit packages specified in `create_default_packages` setting and provide a helpful warning message for the setting. (#15115)
- Avoid triple-equals string exports when the `MatchSpec.version` field starts with a single equals. (#15123 via #15141)
- Respect `CONDA_PROTECT_FROZEN_ENVS` environment variable in the absence of `--override-frozen`. (#15162)

- Indicate that a dry run was performed and no changes were made when running `conda init --dry-run`. (#15097 via #15211)
- Don't load the `environment.yml` environment as part of its `can_handle` function. Just verify if the plugin will be able to interpret the provided file. For the `environment.yml` plugin, it can read any `.yml`/`.yaml` file. (#15237)

Deprecations

- Mark `conda.cli.common.specs_from_url` keyword argument `json` as pending deprecation, to be removed in 26.9. (#15028)
- Mark re-exports from `conda.plugins.types` in `conda.plugins` as pending deprecation, to be removed in 26.9. Import from `conda.plugins.types` instead:
  - `conda.plugins.CondaAuthHandler` → `conda.plugins.types.CondaAuthHandler`
  - `conda.plugins.CondaEnvironmentSpecifier` → `conda.plugins.types.CondaEnvironmentSpecifier`
  - `conda.plugins.CondaHealthCheck` → `conda.plugins.types.CondaHealthCheck`
  - `conda.plugins.CondaPostCommand` → `conda.plugins.types.CondaPostCommand`
  - `conda.plugins.CondaPostSolve` → `conda.plugins.types.CondaPostSolve`
  - `conda.plugins.CondaPostTransactionAction` → `conda.plugins.types.CondaPostTransactionAction`
  - `conda.plugins.CondaPreCommand` → `conda.plugins.types.CondaPreCommand`
  - `conda.plugins.CondaPrefixDataLoader` → `conda.plugins.types.CondaPrefixDataLoader`
  - `conda.plugins.CondaPreSolve` → `conda.plugins.types.CondaPreSolve`
  - `conda.plugins.CondaPreTransactionAction` → `conda.plugins.types.CondaPreTransactionAction`
  - `conda.plugins.CondaReporterBackend` → `conda.plugins.types.CondaReporterBackend`
  - `conda.plugins.CondaRequestHeader` → `conda.plugins.types.CondaRequestHeader`
  - `conda.plugins.CondaSetting` → `conda.plugins.types.CondaSetting`
  - `conda.plugins.CondaSolver` → `conda.plugins.types.CondaSolver`
  - `conda.plugins.CondaSubcommand` → `conda.plugins.types.CondaSubcommand`
  - `conda.plugins.CondaVirtualPackage` → `conda.plugins.types.CondaVirtualPackage`
- Add deprecation warnings for invalid fields in environment.yml files. (#15120)
- Mark `conda.core.index.calculate_channel_urls` as pending deprecation, to be removed in 26.9. Use `conda.models.channel.all_channel_urls(conda.base.context.context.channels)` instead. (#15173)
- Stop implicitly adding the `defaults` channel to the channel list. (#15196)
- Mark `conda.core.index._supplement_index_with_system` as pending deprecation, to be removed in 26.3.
- Postpone `conda.utils.unix_path_to_win` deprecation to `26.3`. Use `conda.common.path.unix_path_to_win` instead. (#15230)
- Constrain minimum compatible conda-build version as >= 25.9 (#15235)

- Mark `conda.auxlib.compat.isiterable` as pending deprecation, to be removed in 26.9. Use `conda.common.compat.isiterable` instead. (#15252)
- Remove `conda.activate._Activator.commands`. Use `conda commands` instead. (#15179)
- Remove `conda.activate._Activator._parse_and_set_arts` argument `arguments`. (#15179)
- Remove `conda.activate.ensure_binary`. Use `conda.common.compat.ensure_binary` instead. (#15179)
- Remove `conda.activate.ensure_fs_path_encoding`. (#15179)
- Remove constants `conda.activate._cygpath`, `conda.activate.native_path_to_unix`, `conda.activate.unix_path_to_native`, `conda.activate.path_identity`. (#15179)
- Remove `conda.cli.common.confirm`. Use `conda.reporters.confirm_yn` instead. (#15180)
- Remove `conda.cli.common.confirm_yn`. Use `conda.reporters.confirm_yn` instead. (#15180)
- Remove `conda.cli.common.print_envs_list`. Use `conda.reporters.render(style='env_list')` instead. (#15180)
- Remove `conda env create remote_definition` argument. (#15181)
- Remove `conda env update remote_definition` argument. (#15181)
- Remove `conda.env.specs.binstar.BinstarSpec` class. (#15181)
- Remove experimental Binstar environment spec plugin. (#15181)
- Remove `conda.cli.main_env_remove.execute`.Use `conda.cli.main_remove.execute` instead. (#15182)
- Remove `conda.cli.install.validate_new_prefix`. Use `conda.cli.install.validate_new_prefix` instead. (#15182)
- Remove constant `conda.exports.win_path_to_unix`. (#15184)
- Remove constant `conda.exports.unix_path_to_win`. (#15184)
- Remove `conda.exports.iteritems`. Use builtin `dict.items()` instead. (#15184)
- Remove unused class `conda.exports.Completer`. (#15184)
- Remove unused class `conda.exports.InstalledPackages`. (#15184)
- Remove constant `conda.exports.move_to_trash`. (#15184)
- Remove constant `conda.exports.KEYS`. (#15184)
- Remove constant `conda.exports.KEYS_DIR`. (#15184)
- Remove unused method `conda.exports.verify`. (#15184)
- Remove `conda.exports.symlink_conda`. Use `conda.activate` instead. (#15184)
- Remove `conda.exports._symlink_conda_help`. Use `conda.activate` instead. (#15184)
- Remove `conda.exports.win_conda_bat_redirect`. Use `conda.activate` instead. (#15184)
- Remove constants `conda.testing.CondaCLIFixture`, `conda.testing.conda_cli`, `conda.testing.PathFactoryFixture`, `conda.testing.path_factory`, `conda.testing.TmpEnvFixture`, `conda.testing.tmp_env`, `conda.testing.TmpChannelFixture`, `conda.testing.tmp_channel`, `conda.testing.context_aware_monkeypatch`, `conda.testing.tmp_pkgs_dir`, `conda.testing.tmp_envs_dir`. (#15185)
- Remove constant `conda.testing.integration.BIN_DIRECTORY`. (#15185)

- Remove `conda.misc.rel_path`. (#15190)
- Remove argument `index_args` `conda.misc.explicit`. (#15190)
- Remove constant `conda.utils.path_identity`. Use `conda.common.path.path_identity` instead. (#15191)
- Remove `conda.utils.win_path_to_cygwin`. Use `conda.common.path.win_path_to_cygwin` instead. (#15191)
- Remove `conda.utils.translate_stream`. (#15191)
- Remove constant `conda.utils._UNIX_SHELL_BASE`. Use `conda.activate` instead. (#15191)
- Remove constant `conda.utils._MSYS2_SHELL_BASE`. Use `conda.activate` instead. (#15191)
- Remove constant `conda.utils._SHELLS`. Use `conda.activate` instead. (#15191)
- Remove free channel from channel configuration. (#15195)
- Remove deprecated property `conda.base.context.restore_free_channel`. (#15195)
- Remove field `conda.base.context._restore_free_channel`. (#15195)
- Remove deprecation warning `conda.base.context._warn_defaults_deprecation`. (#15196)
- Remove deprecated module `conda.cli.python_api`. (#15219)
- Remove deprecated module `conda.common.disk`. (#15219)
- Remove deprecated constant `FILESYSTEM_ENCODING` from `conda.common.compat`. (#15220)
- Remove deprecated function `open` from `conda.common.compat`. Use `conda.common.compat.open_utf8` instead. (#15220)
- Remove deprecated function `six_with_metaclass` from `conda.common.compat`. Use class' `metaclass=` keyword argument instead. (#15220)
- Remove deprecated function `ensure_unicode` from `conda.common.compat`.
- Remove deprecated function `ensure_fs_path_encoding` from `conda.common.compat`.
- Remove class `conda.common.io.Spinner`. Use `conda.reporters.get_spinner` instead. (#15221)
- Remove class `conda.common.io.ProgressBar`. Use `conda.reporters.get_progress_bar` instead. (#15221)
- Remove constant `conda.common.path._VERSION_REGEX`. Use `conda.common.path.python._VERSION_REGEX` instead. (#15222)
- Remove function `conda.common.path.get_bin_directory_short_path`. Use `conda.common.path.BIN_DIRECTORY` instead. (#15222)
- Remove function `conda.common.path.is_private_env_name`. (#15222)
- Remove function `conda.common.path.is_private_env_path`. (#15222)
- Remove constant `conda.common.path.which`.Use builtin `shutil.which` instead. (#15222)
- Remove constant `conda.common.path._cygwin.RE_UNIX`. Use `conda.common.path._cygpath.RE_WIN_DRIVE` instead. (#15222)
- Remove function `conda.common.path._cygwin.translate_unix`. Use `conda.common.path._cygpath._to_unix_drive` instead. (#15222)
- Remove constant `conda.common.path._cygwin.RE_DRIVE`. Use `conda.common.path._cygpath.RE_UNIX_DRIVE` instead. (#15222)

- Remove constant `conda.common.path._cygwin.translation_drive`. Use `conda.common.path._cygpath._to_win_drive` instead. (#15222)
- Remove constant `conda.common.path._cygwin.RE_MOUNT`. Use `conda.common.path._cygpath.RE_UNIX_MOUNT` instead. (#15222)
- Remove constant `conda.common.path._cygwin.translation_mount`. Use `conda.common.path._cygpath._to_win_mount` instead. (#15222)
- Remove constant `conda.common.path._cygwin.RE_ROOT`. Use `conda.common.path._cygpath.RE_UNIX_ROOT` instead. (#15222)
- Remove constant `conda.common.path._cygwin.translation_root`. Use `conda.common.path._cygpath._to_win_root` instead. (#15222)
- Remove `conda.core.index.get_index`. Use `conda.core.Index` instead. (#15223)
- Remove `conda.core.index.fetch_index`. Use `conda.core.Index` instead. (#15223)
- Remove `conda.core.index._supplement_index_with_prefix`. Use `conda.core.Index.reload` instead. (#15223)
- Remove `conda.core.index._supplement_index_with_cache`. Use `conda.core.Index.reload` instead. (#15223)
- Remove `conda.core.index._make_virtual_package`. Use `conda.core.models.records.PackageRecord.virtual_package` instead. (#15223)
- Remove `conda.core.index._supplement_index_with_features`. Use `conda.core.Index.reload(features=True)` instead. (#15223)
- Remove `conda.core.index.get_reduced_index`. Use `conda.core.ReducedIndex` or `conda.core.Index.get_reduced_index` instead. (#15223)
- Remove `conda.core.subdir.make_feature_record`. Use `conda.core.models.records.PackageRecord.feature` instead. (#15223)
- Remove argument `max_retries` for `conda.gateways.disk.delete.rm_rf`. (#15224)
- Remove argument `trash` for `conda.gateways.disk.delete.rm_rf`. (#15224)
- Remove constant `conda.gateways.disk.delete.try_rmdir_all_empty`. (#15224)
- Remove constant `conda.gateways.disk.delete.move_to_trash`. (#15224)
- Remove constant `conda.gateways.disk.delete.move_path_to_trash`. (#15224)
- Remove function `conda.gateways.logging.initialize_root_logger`. (#15224)
- Remove module `conda.models.leased_path_entry`. (#15224)
- Remove function `conda.plugins.manager.get_virtual_packages`. Use `conda.plugins.manager.get_virtual_package_records` instead. (#15224)
- Remove function `conda.env.utils.make_temp_envs_dir`. Use `conda.testing.tmp_envs_dir` instead. (#15224)

Docs

- Update documentation to reflect that `conda` CLI arguments have a higher precedence than environment variables. (#15054)

Other

- Refactor the CLI layer to use the environment model. (#15039)
- Make the `environment.yml` plugin try to handle the environment file, if all other environment spec plugins fail (as a final try). (#15120)
- Update the tests to accept quotes in Python 3.12 argparse errors. (#15172)
- Improve the `EnvironmentIsFrozenError` exception message to indicate how to bypass it with `--override-frozen`. (#15228)
- Check conda-build source code for missing imports in CI. (#15229)
- Make constant for '.condarc' and 'condarc' filenames. (#15232)

Contributors

- @agriyakhetarpal
- @conda-bot
- @dholth
- @zeyugao
- @jaimergp
- @jezdez
- @jcazevedo
- @kenodegard
- @ForgottenProgramme
- @nblair
- @peytondmurray
- @lrandersson
- @ryanskeith
- @soapy1
- @dependabot[bot]
- @pre-commit-ci[bot]

4.5.24 25.7.0 (2025-08-01)

Enhancements

- Ensure conda does not allow for mixing input file types. (#14684 via #14725)
- Add "environment consistency" health check to `conda doctor`. (#14715 via #14799)
- Add support for explicit environment specification files as a supported environment spec according to CEP-23, bringing parity between `conda env` and `conda create/install/update`. (#14820)
- Add an experimental data model to represent an environment. (#14870)
- Add config option `environment_specifier` and CLI argument (`--environment-specifier`, `--env-spec`) to allow users to select which environment specifier plugin to use. (#14877)
- Enhance `conda export` command now supports plugin-based architecture with multiple output formats:

  - `environment-yaml` (cross-platform YAML, default format)
  - `environment-json` (cross-platform JSON for programmatic use)
  - `explicit` (CEP 23 compliant explicit URLs for exact reproduction)
  - `requirements` (MatchSpec-based requirements format) (#14886)
- Add automatic export format detection based on filename patterns (e.g., `environment.yaml`, `explicit.txt`, `requirements.txt`) (#14886)
- Add export format aliases for convenience (`yaml`, `yml`, `json`, `reqs`, `txt`) (#14886)
- Introduce new dataclass `conda.models.environment.EnvironmentConfig` to represent `Environment` settings, a subset of `context` settings that impact environment creation and management. (#14913, #15026)
- Allow env spec plugins to opt-out of auto detection. (#14914)
- Handle exceptions coming from environment spec plugins `can_handle` functions. (#14916)
- The `conda` package is included when cloning an environment. (#14917 via #14919)
- Lazily evaluate CLI options originating from the plugin manager. (#14925)
- Improve the error message that appears when duplicate records are found in the prefix. (#14927)
- Add virtual packages field to the Environment model. (#14979)
- Add ability to create `EnvironmentConfig` instances from a given context. (#14986)
- Add `CondaPlugin` base class with name normalization. (#15002)
- Enhance `LazyChoicesAction` with dynamic choices property and caching for improved CLI argument validation and help text generation. (#15046)

Bug fixes

- Set default value for `version` parameter when `build` is set in `MatchSpec.conda_build_form()`. (#11200 via #15025)
- Improve conda notices cache handling for better user experience. (#14229)
- Improve type hints and error handling in conda installer functions. (#14820)
- Fix `conda export --override-channels` behavior to properly include installed packages' channels (unless `--ignore-channels` is also provided), restoring the original intended functionality and improving environment reproducibility. (#15048 via #14886)
- Fix `tmp_channel` fixture to also include dependencies in the temporary channel. (#14924)
- Fix unsetting environment variables when `envvars_force_uppercase = False`. (#14934 via #14942)
- Respect `--platform` option on subsequent environment operations after initial `conda env create` call. (#14949 via #14956)
- Apply case normalization to all activation environment variables. (#14960)
- Separate `requirements.txt` and `explicit` environment specs. (#14684 via #14963)
- Fix `EnvironmentConfig.aggressive_update_packages` type. It is a list of strings, not a bool. (#14982)
- Fix `EnvironmentConfig.channel_settings` type. It is a list of dicts, not a dict. (#14984)
- Ensure file types can not be mixed in the conda install command. (#14999)
- `ProxyError` now accepts an optional custom message parameter to provide more specific error details. (#14945)

Deprecations

- Mark monkey-patching `json` to support `frozendict` and `obj.to_json()` as pending deprecation, to be removed in 26.9. Use `conda.common.serialize.json.CondaJSONEncoder` instead. (#14709)
- Mark `conda.auxlib.entity.EntityEncoder` as pending deprecation, to be removed in 26.9. Use `conda.common.serialize.json.CondaJSONEncoder` instead. (#14709)
- Mark `conda.auxlib.logz.DumpEncoder` as pending deprecation, to be removed in 26.9. Use `conda.common.serialize.json.CondaJSONEncoder` instead. (#14709)
- Mark `conda.auxlib.logz._DUMPS` as pending deprecation, to be removed in 26.9. Use `conda.common.serialize.json.CondaJSONEncoder(sort_keys=True).encode` instead. (#14709)
- Mark `conda.auxlib.logz.jsondumps` as pending deprecation, to be removed in 26.9. Use `conda.common.serialize.json.CondaJSONEncoder(sort_keys=True).encode` instead. (#14709)
- Mark `conda.common.serialize.EntityEncoder` as pending deprecation, to be removed in 26.9. Use `conda.common.serialize.json.CondaJSONEncoder` instead. (#14709)
- Mark `conda.common.serialize.json_load` as pending deprecation, to be removed in 26.9. Use `conda.common.serialize.json.loads(sort_keys=True)` instead. (#14709)
- Mark `conda.common.serialize.json_dump` as pending deprecation, to be removed in 26.9. Use `conda.common.serialize.json.dumps(sort_keys=True)` instead. (#14709)
- Mark `conda.exports.EntityEncoder` as pending deprecation, to be removed in 26.9. Use `conda.common.serialize.json.CondaJSONEncoder` instead. (#14709)
- Mark `conda.env.specs.binstar.BinstarSpec.environment` as pending deprecation in 26.9. Use `conda.env.specs.binstar.BinstarSpec.env` instead. (#14397)
- Mark `conda.env.specs.requirements.RequirementsSpec.environment` as pending deprecation in 26.9. Use `conda.env.specs.requirements.RequirementsSpec.env` instead. (#14397)
- Mark `conda.env.specs.yaml_file.YamlFileSpec.environment` as pending deprecation in 26.9. Use `conda.env.specs.yaml_file.YamlFileSpec.env` instead. (#14397)
- Mark `conda.env.env.Environment` as pending deprecation in 26.9. Use `conda.env.env.EnvironmentYaml` instead. (#14981)
- Mark `conda.cli.common.arg2spec` as pending deprecation, to be removed in 26.9. (#15028)
- Mark `conda.cli.common.specs_from_args` keyword argument `json` as pending deprecation, to be removed in 26.9. (#15028)
- Mark `conda.gateways.disk.create.create_application_entry_point` as pending deprecation, to be removed in 26.9. (#15062)
- Mark `conda.gateways.disk.create.ProgressFileWrapper` as pending deprecation, to be removed in 26.9. (#15062)
- Mark `conda.gateways.disk.create.create_fake_executable_softlink` as pending deprecation, to be removed in 26.9. (#15062)
- Postpone `conda.env.specs.binstar` deprecation to 26.9. (#15062)

Docs

- Add documentation for `conda export` command with examples and format specifications. (#14886)
- Add environment exporter plugin development guide with examples and best practices. (#14886)
- Update environment management documentation to highlight new export capabilities. (#14886)

- Update cheatsheet with new export formats and commands. (#14886)
- Enhance command comparison table to include new export functionality. (#14886)
- Add glossary entries for CEP 23, environment exporters, explicit format, and requirements format. (#14886)
- Add note about enhanced `conda export` functionality alongside existing `conda env export`. (#14886)
- Add docs about environment spec plugin detection. (#14918)
- No longer claim that `noarch/repodata.json.bz2` is required. Conda has not used `repodata.json.bz2` for years. (#14965)

Other

- Delay environment activation in dev setup to avoid issues when `default_activation_env` is set. (#14910)
- Environment Spec plugins should return an Environment model. (#14937)
- EnvironmentConfig lists of values should be tuples. (#15000)
- Avoid modifying `tests/env/support` in test suite and confusing subsequent test runs. (#15011)
- Remove unused import of deprecated `DumpEncoder`. (#15015)

Contributors

- @conda-bot
- @dholth
- @jaimergp
- @jezdez
- @jjhelmus
- @kathatherine
- @kenodegard
- @ForgottenProgramme
- @mmc1718 made their first contribution in https://github.com/conda/conda/pull/15025
- @soapy1
- @travishathaway
- @dependabot[bot]
- @pre-commit-ci[bot]

4.5.25 25.5.1 (2025-06-05)

Bug fixes

- Allow `conda config` commands to process aliased settings names by forwarding them to the canonical name (e.g. `self_update` -> `auto_update_conda`). (#14899 via #14898)

Docs

- Mark environment spec hooks and types as experimental (#14900)

Contributors

- @jaimergp
- @soapy1
- @travishathaway

4.5.26 25.5.0 (2025-05-21)

Enhancements

- Add the ability to define pre- and post-transaction hooks. (#14754)
- Add support for showing and manipulating settings registered via the settings plugin hook to `conda config`. (#13661 via #14510)
- Raise `conda.exceptions.OfflineError` instead of generic `RuntimeError` when `--offline` is used for better error handling. (#14664 via #14665)
- Add `context.default_activation_env` setting, which allows users to customize which environment should get auto activated on initialization (if `context.auto_activate` is enabled), as well as `conda activate` calls without arguments. Defaults to `base`. (#14666)
- Add `conda init --condabin` to only add the `$CONDA_PREFIX/condabin` directory to the `PATH` environment variable instead of installing a shell function. (#14703)
- Add `environment_specifiers` plugin hooks to allow the creation of plugins to read different file formats and sources. (#14710)
- Extend `PrefixData` API with two constructors (`from_name()`, `from_context()`), a new property (`name`), several boolean-returning methods (`exists()`, `is_environment()`, `is_base()`), and a few exception-raising methods (`assert_exists()`, `assert_environment()`, `assert_writable()`, `validate_path()`, `validate_name()`). An additional method `set_nonadmin()` allows to plant the `.nonadmin` marker on Windows. (#14747 via #14750)
- Add support for frozen environment markers (CEP 22). (#14746 via #14766)
- Add a plugin hook for `PrefixData` loaders and use it to wrap the existing PyPI interoperability features. These are now present in `conda.plugins.prefix_data_loaders.pypi`. (#14780)
- Add a `--fields` flag to `conda list` that allows to customize the content of the `human` output format. (#14781)
- Add debug logs to get visibility into environment spec plugin selection. (#14815)
- Add `conda.models.records.SolvedRecord` class. It is a `PackageRecord` subclass augmented with the `requested_spec` field originally found in `PrefixRecord`. (#14821)

Bug fixes

- Cleanup references to `DepsModifier.UPDATE_SPECS`. (#14807)
- Report package variant installation as "REVISED" as opposed to "DOWNGRADED". (#13797 via #14727)
- Add a validation step to check if the target environment is writable when running the `install` or `update` commands. (#12561 via #14668)
- Fix cache key in `PrefixData`. It will now consider `interoperability`. (#14750)
- Do not cache `PrefixData` records across successive `conda list` invocations. (#14750)
- Fix `environment.yaml` spec plugin to only handle yaml files (files with `.yaml` or `.yml` file extensions). (#14823)
- Fix `conda info` rendering to display components in a consistent order. (#14829 via #14831)

Deprecations

- Mark `conda.base.context.context.auto_activate_base` as pending deprecation, to be removed in 26.3. Use `conda.base.context.context.auto_activate` instead. (#14666)
- Mark `conda.base.context.Context.pip_interop_enabled` as pending deprecation, to be removed in 26.3. Use `conda.base.context.Context.prefix_data_interoperability`. instead. (#14750)
- Mark `conda.base.context._first_writable_envs_dir` as pending deprecation, to be removed in 26.3. Use `conda.gateways.disk.create.first_writable_envs_dir()` and `PrefixData.from_name()` instead. (#14750)
- Mark `conda.base.context.validate_prefix_name` as pending deprecation, to be removed in 26.3. Use `PrefixData.validate_name()` and `PrefixData.from_name()` instead. (#14750)
- Mark `conda.cli.common.validate_prefix` as pending deprecation, to be removed in 26.3. Use `PrefixData.assert_environment()` instead. (#14750)
- Mark `conda.cli.common.validate_prefix_is_writable` as pending deprecation, to be removed in 26.3. Use `PrefixData.assert_writable()` instead. (#14750)
- Mark `conda.cli.install.check_prefix` as pending deprecation, to be removed in 26.3. Use `PrefixData.exists()`, `PrefixData.validate_path()`, `PrefixData.validate_name()` instead. (#14750)
- Mark `conda.cli.install.print_activate` as pending deprecation, to be removed in 26.3. Use `conda.cli.common.print_activate` instead. (#14670)
- Mark `conda.cli.install.validate_new_prefix` as pending deprecation, to be removed in 26.3. Use `PrefixData.exists()` and `PrefixData.validate_path()` instead. (#14750)
- Mark `conda.cli.install.validate_prefix_exists` as pending deprecation, to be removed in 26.3. Use `PrefixData.exists()` instead. (#14750)
- Mark `conda.cli.main_info.get_info_components` as pending deprecation, to be removed in 26.3. Use `conda.cli.main_info.iter_info_components` instead. (#14837)
- Mark `conda.cli.main_rename.check_protected_dirs` as pending deprecation, to be removed in 26.3. Use `PrefixData.validate_path()` instead. (#14750)
- Mark `conda.cli.main_rename.validate_src` as pending deprecation. to be removed in 26.3. Use `PrefixData.validate_path()` and `PrefixData.validate_name()` instead. (#14750)
- Mark `conda.common.pkg_formats.python` module as pending deprecation, to be removed in 26.3. Use `conda.plugins.prefix_data_loaders.pypi.pkg_format` instead. (#14798)
- Mark `conda.common.pkg_formats` subpackage as pending deprecation, to be removed in 26.3. Use the `prefix_data_loaders` plugin hook instead. (#14798)
- Mark `conda.common.url.hex_octal_to_int` as pending deprecation, to be removed in 26.3. Use `int(..., 16)` instead. (#14750)
- Mark `conda.core.link.PrefixActionGroup` as pending deprecation, to be removed in 26.3. Use `conda.core.link.PrefixActions` instead. (#14754)
- Mark `conda.core.link.PrefixActions` as pending deprecated, to be removed in 26.3. Use `conda.core.link.PrefixActionGroup` instead. (#14754)
- Mark `conda.core.path_actions._Action` as pending deprecation, to be removed in 26.3. Use `conda.core.path_actions.Action` instead. (#14754)
- Mark `conda.core.prefix_data.get_python_version_for_prefix()` as pending deprecation, to be removed in 26.3. Use `conda.core.prefix_data.PrefixData(prefix).get("python").version` instead. (#14750)

- Mark `conda.core.prefix_data.PrefixData` keyword argument `pip_interop_enabled` as pending deprecation, to be removed in 26.3. Use the `interoperability` keyword argument instead. (#14750)
- Mark `conda.core.prefix_data.PrefixData._load_site_packages()` as pending deprecation, to be removed in 26.3. Use `conda.plugins.prefix_data_loaders.pypi.load_site_packages()` instead. (#14750)
- Mark `conda.core.prefix_data.PrefixData._python_pkg_record` as pending deprecation, to be removed in 26.3. Use `PrefixData.get("python")` instead. (#14750)
- Mark `conda.core.prefix_data.python_record_for_prefix()` as pending deprecation, to be removed in 26.3. Use `conda.core.prefix_data.PrefixData(prefix).get("python")` instead. (#14750)
- Mark `conda.gateways.disk.read.read_python_record` as pending deprecation, to be removed in 26.3. Use `conda.plugins.prefix_data_loaders.pypi.pkg_format.read_python_record` instead. (#14798)
- Mark `conda.gateways.disk.test.is_conda_environment` as pending deprecation, to be removed in 26.3. Use `PrefixData.is_environment()` instead. (#14750)
- Mark `conda.gateways.disk.test.touch_nonadmin` as pending deprecation, to be removed in 26.3. Use `PrefixData.set_nonadmin()` instead. (#14750)
- Mark `conda.models.records.PackageRecord.schannel` as pending deprecation, to be removed in 26.3. Use `conda.models.records.PackageRecord.channel_name` instead. (#14781)
- Mark `conda.trust.signature_verification` and `conda.trust.constants` as pending deprecation, to be removed in 26.3. The functionality will be moved to `conda-content-trust` and `conda-anaconda-trust-root` packages respectively. (#14849)
- Mark `conda info --license` as pending deprecation, to be removed in 26.3. (#14831)
- Mark `conda info --root` as pending deprecation, to be removed in 26.3. Use `conda info --base` instead. (#14831)

Docs

- Add type hints and docstrings to `conda.core.portability`. (#13820)
- Add type hints and docstrings to `conda.core.subdir_data`. (#13821)
- Add type hints to `conda.base.constants`. (#13480)
- Add type hints to `conda.base.context`. (#14776)
- Add type hints to `conda.__init__`, `conda.exceptions` and `conda.exception_handler`. (#14776)
- Add type hints to `conda.core.prefix_data`. (#14779)
- Add examples to environment specs plugin docs. (#14814)
- Add type hints to `conda.models.channel.Channel`. (#14817)
- Update the conda cheatsheet to 25.3.1 version. (#14830)

Other

- Refactor `--repodata-fn` iteration in `conda.cli.install` and other smaller cleanups. (#14670)
- Refactor command line argument validations from `conda.cli.install.install` to the relevant `cli` modules. (#14742)
- Refactor `conda create --clone` logic into a separate function. (#14743)
- Use `dict.fromkeys` for sequence deduplication, instead of `IndexedSet()`. (#14777)

Contributors

- @conda-bot
- @faithrider made their first contribution in https://github.com/conda/conda/pull/13820
- @jaimergp
- @jezdez
- @kathatherine
- @kenodegard
- @peytondmurray made their first contribution in https://github.com/conda/conda/pull/14754
- @samhaese
- @soapy1
- @travishathaway
- @dependabot[bot]
- @pre-commit-ci[bot]

4.5.27 25.3.1 (2025-04-03)

Bug fixes

- Restore CMD prompt modifier. (#14711 via #14713)

Contributors

- @kenodegard

4.5.28 25.3.0 (2025-03-17)

Enhancements

- Add support for Python 3.13. (#14584)
- Refactor CMD's activation process to use a static shell script instead of a dynamic temporary script. This resolves using `conda activate` with script execution restrictions when Windows AppLocker is enabled. (#13610 via #14607)

Bug fixes

- Fix variable deletion in xonsh. (#14543 via #14550)
- Make `conda.gateways.logging.TokenURLFilter.TOKEN_REPLACE` a staticmethod for future Python compatibility. (#14590)
- Allow user to specify `--name` or `--prefix` as part of `conda config` command. (#12137 via #14648)

Deprecations

- Postpone `conda.base.context.Context.conda_exe` deprecation to `26.3`. (#14647)
- Postpone `conda.base.context.Context.restore_free_channel` deprecation to `25.9`. (#14647)
- Postpone implicit `defaults` multichannel deprecation to `25.9`. Users relying on this behavior are encouraged to run `conda config --add channels defaults`. (#14178 via #14662)
- Postpone loading subcommands from executables deprecation to `conda 26.3`. (#14644)

- Mark `conda {create,install,update} -f` as pending deprecation. Use `conda {create,install,update} --file` instead. (#14446)
- Mark `conda._vendor` as pending deprecation. Conda no longer vendors dependencies. (#14562)
- Mark `conda.common.io._logger_lock` as pending deprecation. Use `logging._lock` instead. (#14584)
- Mark `conda.exports.plan` as pending deprecation. (#14631)
- Mark `conda.plan.Channel` as pending deprecation. Use `conda.models.channel.Channel` instead. (#14603)
- Mark `conda.plan.context` as pending deprecation. Use `conda.base.context.context` instead. (#14603)
- Mark `conda.plan.dashlist` as pending deprecation. Use `conda.common.io.dashlist` instead. (#14603)
- Mark `conda.plan.defaultdict` as pending deprecation. Use builtin `collections.defaultdict` instead. (#14603)
- Mark `conda.plan.DEFAULTS_CHANNEL_NAME` as pending deprecation. Use `conda.base.constants.DEFAULTS_CHANNEL_NAME` instead. (#14603)
- Mark `conda.plan.Dist` as pending deprecation. Use `conda.models.dist.Dist` instead. (#14603)
- Mark `conda.plan.env_vars` as pending deprecation. Use `conda.common.io.env_vars` instead. (#14603)
- Mark `conda.plan.FETCH` as pending deprecation. Use `conda.instructions.FETCH` instead. (#14603)
- Mark `conda.plan.groupby_to_dict` as pending deprecation. Use `conda.common.iterators.groupby_to_dict` instead. (#14603)
- Mark `conda.plan.human_bytes` as pending deprecation. Use `conda.utils.human_bytes` instead. (#14603)
- Mark `conda.plan.IndexedSet` as pending deprecation. Use `boltons.setutils.IndexedSet` instead. (#14603)
- Mark `conda.plan.LAST_CHANNEL_URLS` as pending deprecation. Use `conda.core.index.LAST_CHANNEL_URLS` instead. (#14603)
- Mark `conda.plan.LINK` as pending deprecation. Use `conda.instructions.LINK` instead. (#14603)
- Mark `conda.plan.LinkType` as pending deprecation. Use `conda.models.enums.LinkType` instead. (#14603)
- Mark `conda.plan.log` as pending deprecation. Use builtin `logging` instead. (#14603)
- Mark `conda.plan.MatchSpec` as pending deprecation. Use `conda.models.match_spec.MatchSpec` instead. (#14603)
- Mark `conda.plan.normalized_version` as pending deprecation. Use `conda.models.version.normalized_version` instead. (#14603)
- Mark `conda.plan.PackageRecord` as pending deprecation. Use `conda.models.records.PackageRecord` instead. (#14603)
- Mark `conda.plan.PrefixSetup` as pending deprecation. Use `conda.core.link.PrefixSetup` instead. (#14603)
- Mark `conda.plan.prioritize_channels` as pending deprecation. Use `conda.models.channel.prioritize_channels` instead. (#14603)
- Mark `conda.plan.reset_context` as pending deprecation. Use `conda.base.context.reset_context` instead. (#14603)

- Mark `conda.plan.SYMLINK_CONDA` as pending deprecation. Use `conda.instructions.SYMLINK_CONDA` instead. (#14603)
- Mark `conda.plan.sys` as pending deprecation. Use builtin `sys` instead. (#14603)
- Mark `conda.plan.time_recorder` as pending deprecation. Use `conda.common.io.time_recorder` instead. (#14603)
- Mark `conda.plan.TRACE` as pending deprecation. Use `conda.common.constants.TRACE` instead. (#14603)
- Mark `conda.plan.UNKNOWN_CHANNEL` as pending deprecation. Use `conda.base.constants.UNKNOWN_CHANNEL` instead. (#14603)
- Mark `conda.plan.UNLINK` as pending deprecation. Use `conda.instructions.UNLINK` instead. (#14603)
- Mark `conda.plan.UnlinkLinkTransaction` as pending deprecation. Use `conda.core.link.UnlinkLinkTransaction` instead. (#14603)
- Mark `conda.plan` as pending deprecation. (#14603)
- Mark `conda.testing.fixtures.session_capsys` as pending deprecation. (#14575)
- Mark `conda_env.cli` as pending deprecation. (#14564)
- Mark `conda_env.installers` as pending deprecation. (#14564)
- Mark `conda_env` as pending deprecation. (#14564)
- Remove `conda create --mkdir`. Redundant argument. (#14644)
- Remove `conda install --mkdir`. Use `conda create` instead. (#14644)
- Remove `conda rename --force`. Use `conda rename --yes` instead. (#14644)
- Remove `conda._vendor.frozendict`. Use `frozendict` instead. (#14562, #14582)
- Remove `conda.activate._Activator._get_path_dirs(extra_library_bin)`. (#14563)
- Remove `conda.activate._Activator.add_export_unset_vars`. Use `conda.activate._Activator.get_export_unset_vars` instead. (#14563)
- Remove `conda.activate._Activator.get_scripts_export_unset_vars`. Use `get_scripts_export_unset_vars` helper function in `test_activate.py` instead. (#14563)
- Remove `conda.activate.JSONFormatMixin.get_scripts_export_unset_vars`. Use `conda.activate._Activator.get_export_unset_vars` instead. (#14563)
- Remove `conda.auxlib.collection.make_immutable`. Use `frozendict.deepfreeze` instead. (#14586)
- Remove `conda.cli.main_env.configure_parser(sub_parsers=None)`. (#14600)
- Remove `conda.cli.main_env_list.execute`. Use `conda.cli.main_info.execute(envs=True)` instead. (#14604)
- Remove `conda.cli.main_info.get_info_dict(system)`. (#14604)
- Remove `conda.common.compat.encode_arguments`. (#14604)
- Remove `conda.env.specs.detect(remote_definition)`. (#14604)
- Remove `conda.instructions.PREFIX`. (#14604)
- Remove `conda.plan._get_best_prec_match`. Use `conda.misc._get_best_prec_match` instead. (#14603)
- Remove `conda.plan._handle_menuinst`. (#14603)
- Remove `conda.plan._inject_UNLINKLINKTRANSACTION`. (#14603)

- Remove `conda.plan._plan_from_actions`. (#14603)
- Remove `conda.plan._update_old_plan`. (#14603)
- Remove `conda.plan.add_defaults_to_specs`. (#14603)
- Remove `conda.plan.add_unlink`. (#14603)
- Remove `conda.plan.display_actions`. (#14603)
- Remove `conda.plan.execute_actions`. (#14603)
- Remove `conda.plan.execute_plan`. (#14603)
- Remove `conda.plan.execute_plan`. (#14603)
- Remove `conda.plan.get_blank_actions`. (#14603)
- Remove `conda.plan.install_actions`. (#14603)
- Remove `conda.plan.print_dists`. (#14603)
- Remove `conda.plan.revert_actions`. Use `conda.cli.install.revert_actions` instead. (#14603)
- Remove `conda.plan` as an entrypoint. (#14603)
- Remove `conda.testing.integration._get_temp_prefix`. Use `tmp_path`, `conda.testing.path_factory`, or `conda.testing.tmp_env` instead. (#14645)
- Remove `conda.testing.integration.create_temp_location`. Use `tmp_path` or `conda.testing.fixtures.path_factory` fixtures instead. (#14604)
- Remove `conda.testing.integration.FORCE_temp_prefix`. Use `tmp_path`, `conda.testing.fixtures.path_factory`, or `conda.testing.fixtures.tmp_env` fixtures instead. (#14604)
- Remove `conda.testing.integration.make_temp_channel`. Use `conda.testing.fixtures.tmp_channel` fixture instead. (#14604)
- Remove `conda.testing.integration.make_temp_env`. Use `conda.testing.fixtures.tmp_env` fixture instead. (#14604)
- Remove `conda.testing.integration.make_temp_package_cache`. Use `conda.testing.fixtures.tmp_pkgs_dir` fixture instead. (#14604)
- Remove `conda.testing.integration.make_temp_prefix`. Use `tmp_path`, `conda.testing.path_factory`, or `conda.testing.tmp_env` instead. (#14645)
- Remove `conda.testing.integration.run_command`. Use `conda.testing.conda_cli` instead. (#14645)
- Remove `conda.testing.integration.running_a_python_capable_of_unicode_subprocessing`. (#14669)
- Remove `conda.testing.integration.set_tmpdir` fixture. Use `tmp_path`, `conda.testing.path_factory`, or `conda.testing.tmp_env` instead. (#14645)
- Remove `conda.testing.integration.tempdir`. Use `tmp_path` or `conda.testing.fixtures.path_factory` fixtures instead. (#14604)
- Remove `conda_env.cli.common`. Use `conda.env.env` instead. (#14564)
- Remove `conda_env.cli.main_config`. Use `conda.cli.main_env_config` instead. (#14564)
- Remove `conda_env.cli.main_create`. Use `conda.cli.main_env_create` instead. (#14564)
- Remove `conda_env.cli.main_export`. Use `conda.cli.main_export` instead. (#14564, #14601)
- Remove `conda_env.cli.main_list`. Use `conda.cli.main_env_list` instead. (#14564)

- Remove `conda_env.cli.main_remove`. Use `conda.cli.main_env_remove` instead. (#14564)
- Remove `conda_env.cli.main_update`. Use `conda.cli.main_env_update` instead. (#14564)
- Remove `conda_env.cli.main_vars`. Use `conda.cli.main_env_vars` instead. (#14564)
- Remove `conda_env.env`. Use `conda.env.env` instead. (#14564)
- Remove `conda_env.installers.base`. Use `conda.env.installers.base` instead. (#14564)
- Remove `conda_env.installers.conda`. Use `conda.env.installers.conda` instead. (#14564)
- Remove `conda_env.installers.pip`. Use `conda.env.installers.pip` instead. (#14564)
- Remove `conda_env.pip_util`. Use `conda.env.pip_util` instead. (#14564)
- Remove `conda_env.specs.binstar`. Use `conda.env.specs.binstar` instead. (#14564)
- Remove `conda_env.specs.requirements`. Use `conda.env.specs.requirements` instead. (#14564)
- Remove `conda_env.specs.yaml_file`. Use `conda.env.specs.yaml_file` instead. (#14564)
- Remove `conda_env.specs`. Use `conda.env.specs` instead. (#14564)
- Remove `logging.Logger.trace` monkeypatch. Use `logging.getLogger(__name__)(conda.common.constants.TRACE, ...)` instead. (#14647)

Contributors

- @v2thegreat made their first contribution in https://github.com/conda/conda/pull/14581
- @conda-bot
- @jaimergp
- @jezdez
- @jjhelmus
- @kenodegard
- @soapy1
- @travishathaway
- @dependabot[bot]
- @pre-commit-ci[bot]

4.5.29 25.1.1 (2025-01-28)

Bug fixes

- Fix PowerShell activation/deactivation to properly unset `$env:_CE_M` and `$env:_CE_CONDA` environment variables. (#14292 via #14517)

Contributors

- @travishathaway

4.5.30 25.1.0 (2025-01-17)

Enhancements

- Set `__win` version and enable `CONDA_OVERRIDE_WIN` usage. (#14443 via #14450)

Bug fixes

- Merge overlapping glob build specs instead of raising `Incompatible component merge`. (#11612)
- Fix a bug when invalid values are being passed to `conda shell.posix` command (#14398)
- Sort suggested subcommands when an incorrect subcommand is provided in the CLI. (#13332 via 14402)
- Fix a bug where the setting `denylist_channels` was not being recognized in certain cases. (#14405)
- Do not use native platform version to set `__osx` or `__linux` version if the underlying OS is not macOS or Linux, respectively. (#14448 via #14449)
- Report real macOS version (11+ instead of 10.16) even if the Python interpreter was linked against SDK 10.15 or earlier. This applies to the `__osx` virtual package and the user agent info. (#13178, #13832 via #14449)
- Accept %-encoded URLs as a valid `MatchSpec`. (#14481)
- Retry failed downloads one time on `ChecksumMismatchError` as caused by bad partial downloads. Use `r+b` or `w+b` instead of "append" mode. Improve test coverage. (#13488)

Docs

- Use the correct parameter name, `handler`, in the CondaAuthHandler example (#14428).

Other

- Require `conda-libmamba-solver >=24.11.0` for `libmamba 2.x` compatibility. (#11612)
- Added new `PYTHONPATH` autoused fixture in `conda.testing` to ensure development `conda` is used across all tests. The fixture doesn't apply in `PYTHONPATH` is already set in the environment. (#14475)

Contributors

- @conda-bot
- @dbast
- @dholth
- @jaimergp
- @jezdez
- @jjhelmus
- @kenodegard
- @ForgottenProgramme
- @minrk
- @travishathaway
- @dependabot[bot]
- @pre-commit-ci[bot]

4.5.31 24.11.3 (2025-01-06)

Bug fixes

- Fix recursion error introduced in `__conda_reactivate` deprecation message. (#14468)

Contributors

- @jaimergp

4.5.32 24.11.2 (2024-12-19)

Bug fixes

- Restore `__conda_reactivate` shell command removed in 24.11.0. (#14455)

Deprecations

- Mark `__conda_reactivate` as deprecated. Use `__conda_activate reactivate` instead. (#14455)

Contributors

- @kenodegard

4.5.33 24.11.1 (2024-12-10)

Bug fixes

- Fix a bug where the setting `denylist_channels` was not being recognized in certain cases. (#14405)

Contributors

- @travishathaway

4.5.34 24.11.0 (2024-11-18)

Enhancements

- Adds a new plugin hook for reporter backends. (#13868)
- Add support for CEP-17 that allows specifying the path to the site-packages directory of the the Python package via the `repodata.json`. (#14053 via #14256)
- Adds progress bar support for reporter backends plugin hook. (#14083)
- Adds support for defining spinners for the reporter backends plugin hook. (#14206)
- Adds support for confirmation functions for reporter backends plugin hook. (#14244)
- Add new plugin hooks (`conda_session_headers` and `conda_request_headers`) to add headers to outgoing HTTP requests. (#14325, #14382)

Bug fixes

- Do not retry solves twice in failed `conda env` runs. (#13784)
- Remove CreateNonAdminAction to prevent conda remove from deleting `.nonadmin` files. (#14271)
- Do not map Python distribution names to conda names in `PrefixData(pip_interop_enabled=True)`. (#14310 via #14317)
- Fix output writing for `conda export --json --file`. (#14316 via #14323)
- Update `deprecated.action()` function to account for positional arguments that have no value specified. (#14355 via #14359)
- Fix continuous integration upload of coverage files. (#14375)

Deprecations

- Remove `__conda_reactivate` shell function in favor of `__conda_activate reactivate`. (#14277)
- Mark `conda.misc.rel_path` as pending deprecation. (#14338)
- Require Python 3.9 or greater. (#14201 via #14368)

Contributors

- @beeankha
- @conda-bot
- @dholth
- @duncanmmacleod
- @jaimergp
- @jezdez
- @jjhelmus
- @kathatherine
- @kenodegard
- @zklaus
- @ForgottenProgramme
- @marcoesters
- @muffato made their first contribution in https://github.com/conda/conda/pull/14342
- @nilskch made their first contribution in https://github.com/conda/conda/pull/14214
- @travishathaway
- @dependabot[bot]
- @pre-commit-ci[bot]

4.5.35 24.9.2 (2024-10-16)

Bug fixes

- Refine the deprecation warning to the prepopulation of the channel list with “defaults”. (#14305)
- Add protected directories guardrail to `conda create` command. (#14282, #14315)

Contributors

- @beeankha
- @jezdez
- @travishathaway

4.5.36 24.9.1 (2024-10-01)

Bug fixes

- Tweak pending deprecation warning for the upcoming changes in channel defaults. (#14287 via #14288)

Deprecations

- Remove deprecated testing entrypoint `shell/bin/conda`. Use `dev/start[.bat]` instead. (#14285)

Contributors

- @jezdez
- @kenodegard

4.5.37 24.9.0 (2024-09-26)

Special announcement

This is an announcement about an important and positive **future** change in conda's functionality:

Following feedback from conda users about the pre-configuration of the conda code base to favor channels from Anaconda Inc, we've started the process to deprecate hardcoding Anaconda's channels as the default set of channels in the conda source code, which is a remnant of conda's incubation at the company.

In the future, we will rely on providers of conda distributions, such as miniforge or Anaconda (including miniconda), to pre-configure their preferred channels, e.g. by running the necessary `conda config --set channels` command.

We're also going to continue to work on improving channel management in the forseeable future and would love to get your feedback.

Enhancements

- Add `conda.core.index.Index` as a faster drop-in replacement of the realized dictionary index. Note: The loggers are no longer implicitly initialized when fetching the index. Instead, you must explicitly call `conda.gateways.logging.initialize_logging`. (#13880, #14267)
- Alias `conda env remove` command to `conda remove --all`. (#13977)
- Add a new health check to `conda doctor` that detects if the `REQUESTS_CA_BUNDLE` env var points to a non-existent file. (#12905 via #14037)
- Add `--sha256` flag to `conda list --explicit` so it lists URLs with a SHA256 hash instead of MD5 and make `conda install|create` compatible with these inputs. (#2903, #7882 via #14048)
- Report `conda` version used to generate a `@EXPLICIT` text file. (#14048)
- Update health checks outputs for consistency. (#14049 via #14079)
- Fix Windows to Unix path conversion to handle UNC mounts and root paths. (#14157)
- Add a new `denylist_channels` config option and `CONDA_DENYLIST_CHANNELS` environment variable to explicitly deny using specific channels globally, which is complementing the already existing `allowlist_channels` config option. (#14176 via #14196)
- Add `conda commands` subcommand. (#14215)
- Add `conda.testing.fixtures.session_capsys`. Use this to capture stdout and stderr within module, package, and session scoped fixtures. (#14243)
- Add `conda.testing.fixtures.session_conda_cli`. Use this to invoke conda commands within module, package, and session scoped fixtures. (#14243)
- Add `conda.testing.fixtures.session_tmp_env`. Use this to create a conda environment within module, package, and session scoped fixtures. (#14243)

Bug fixes

- Prevent directories that contain conda environments from being specified as an environment prefix when creating new envs; this provides guardrails against the accidental deletion of environments via commands such as `conda rename`. (#13955)
- Enable `conda doctor` to check whether the environment exists or not before trying to generate a health report. (#14042 via #14112)
- Fix `conda.common.configuration.ObjectParameter`'s ability to appropriately handle defaults. (#14148 via #14149)
- `conda remove [env] --all` command no longer deletes empty parent directories for environments that are removed. (#14173)
- Raise an error when attempting to remove non-existent environment. (#14174 via #14199)
- Add pre-/post-command hooks to activation subcommands (`activate`, `deactivate`, `reactivate`, `hooks`, and `commands`). (#14211 via #14212)
- Replace extensions from end of filename only, not `str.replace()`, in two places. (#14241)
- Fix PowerShell activation/deactivation to properly unset environment variables using `$Env:VARIABLE = $null` instead of `$Env:VARIABLE = ""`. (#14237 via #14246)

Deprecations

- Mark `conda.core.index.get_index` as pending deprecation. Use `conda.core.index.Index` instead. (#13880)
- Mark `conda.core.index.get_reduced_index` as pending deprecation. Use `conda.core.index.ReducedIndex` instead. (#13880)
- Mark `conda.core.index.fetch_index` as pending deprecation. Use `conda.core.index.Index` instead. (#13880)
- Mark `conda.core.index._supplement_index_with_prefix` as pending deprecation. Use `conda.core.index.Index.reload` instead. (#13880)
- Mark `conda.core.index._supplement_index_with_cache` as pending deprecation. Use `conda.core.index.Index.reload` instead. (#13880)
- Mark `conda.core.index._supplement_index_with_features` as pending deprecation. Use `conda.core.index.Index.reload` instead. (#13880)
- Mark `conda.core.index._supplement_index_with_system` as pending deprecation. Use `conda.core.index.Index.reload` instead. (#13880)
- Mark `conda.core.index._make_virtual_package` as pending deprecation. Use `conda.models.records.PackageRecord.virtual_package` instead. (#13880)
- Mark `conda.core.subdir_data.make_feature_record` as pending deprecation. Use `conda.models.records.PackageRecord.feature` instead. (#13880)
- Mark `conda.plugins.manager.CondaPluginManager.get_virtual_packages` as pending deprecation. Use `conda.plugins.manager.CondaPluginManager.get_virtual_package_records` instead. (#13880)
- Mark `conda.misc.explicit(index_args)` as pending deprecation. (#14267)
- Mark `conda.cli.main_rename.validate_destination` as pending deprecation. Use `conda.cli.install.validate_new_prefix` instead. (#13955)

- Mark `conda.cli.main_env_remove.execute` as pending deprecation. Use `conda.cli.main_remove.execute` instead. (#13977)
- Mark `conda.activate.path_identity` as pending deprecation. Use `conda.common.path.path_identity` instead. (#14068)
- Mark `conda.utils.path_identity` as pending deprecation. Use `conda.common.path.path_identity` instead. (#14068)
- Mark `conda.models.leased_path_entry` as pending deprecation. (#14077)
- Postpone removal of `conda.base.context.Context.conda_exe` to `conda 25.3`. (#14077)
- Postpone removal of `conda.cli.python_api` to `conda 25.9`. (#14077)
- Remove `conda._vendor.appdirs`. Use `platformdirs` instead. (#14077)
- Remove `conda._vendor.cpuinfo`. (#14077)
- Remove `conda._vendor.distro`. Use `distro` instead. (#14077)
- Remove `conda.auxlib.collection.call_each`. (#14077)
- Remove `conda.auxlib.collection.firstitem`. (#14077)
- Remove `conda.auxlib.compat.NoneType`. (#14077)
- Remove `conda.auxlib.compat.primitive_types`. (#14077)
- Remove `conda.auxlib.compat.utf8_writer`. (#14077)
- Remove `conda.auxlib.exceptions.AssignmentError`. (#14077)
- Remove `conda.auxlib.exceptions.AuthenticationError`. (#14077)
- Remove `conda.auxlib.exceptions.InitializationError`. (#14077)
- Remove `conda.auxlib.exceptions.NotFoundError`. (#14077)
- Remove `conda.auxlib.exceptions.SenderError`. (#14077)
- Remove `conda.auxlib.type_coercion.boolify_truthy_string_ok`. (#14077)
- Remove `conda.auxlib.type_coercion.listify`. (#14077)
- Remove `conda.base.context.Context.root_dir`. Use `conda.base.context.Context.root_prefix` instead. (#14077)
- Remove `conda.base.exceptions`. (#14077)
- Remove `conda.cli.main.generate_parser`. Use `conda.cli.conda_argparse.generate_parser` instead. (#14077)
- Remove `conda.cli.main.init_loggers(context)` (#14077)
- Remove `conda.cli.main_rename.validate_src(name)`. (#14077)
- Remove `conda.cli.main_rename.validate_src(prefix)`. (#14077)
- Remove `conda.common.configuration.load_file_configs`. (#14077)
- Remove `conda.common.decorators.env_override`. (#14077)
- Remove `conda.common.decorators`. (#14077)
- Remove `conda.common.disk.temporary_content_in_file`. Use `tempfile` instead. (#14077)
- Remove `conda.core.package_cache_data.download`. Use `conda.gateways.connection.download.download` instead. (#14077)

- Remove `conda.core.package_cache_data.rm_fetched`. (#14077)
- Remove `conda.core.package_cache`. Use `conda.core.package_cache_data` instead. (#14077)
- Remove `conda.core.prefix_data.PrefixData._has_python`. (#14077)
- Remove `conda.core.subdir_data.get_cache_control_max_age`. Use `conda.gateways.repodata.get_cache_control_max_age` instead. (#14077)
- Remove `conda.exports.fetch_index`. Use `conda.core.index.fetch_index` instead. (#14077)
- Remove `conda.exports.IndexRecord`. Use `conda.models.records.PackageRecord` instead. (#14077)
- Remove `conda.gateways.anaconda_client.EnvAppDirs`. Use `platformdirs` instead. (#14077)
- Remove `conda.gateways.connection.adapters.ftp.FTPAdapter.stor`. (#14077)
- Remove `conda.gateways.connection.adapters.ftp.parse_multipart_files` (#14077)
- Remove `conda.gateways.logging.set_verbosity`. Use `conda.gateways.logging.set_log_level` instead. (#14077)
- Remove `conda.gateways.logging.VERBOSITY_LEVELS`. (#14077)
- Remove `conda.models.dist.IndexRecord`. Use `conda.models.records.PackageRecord` instead. (#14077)
- Remove `conda.models.enums.LeasedPathType`. (#14077)
- Remove `conda.models.leased_path_entry.LeasedPathEntry`. (#14077)
- Remove `conda.plugins.subcommands.doctor.get_prefix`. Use `conda.base.context.context.target_prefix` instead. (#14077)
- Remove `conda.plugins.subcommands.doctor.health_checks.display_health_checks`. (#14077)
- Remove `conda.plugins.subcommands.doctor.health_checks.display_report_heading`. (#14077)
- Remove `conda.testing.helpers.set_active_prefix`. Use `mocker.patch('conda.base.context.Context.active_prefix')` instead. (#14077)
- Remove `tests.env.utils.run_command`. Use `conda_cli` fixture instead. (#14077)
- Mark `tests.env.utils.make_temp_envs_dir` as pending deprecation. Use `tmp_envs_dir` fixture instead. (#14093)
- Mark `conda.gateways.disk.delete.rm_rf(max_retries)` as pending deprecated. (#14094)
- Mark `conda.gateways.disk.delete.rm_rf(trash)` as pending deprecated. (#14094)
- Mark `conda.gateways.disk.delete.try_rmdir_all_empty` as pending deprecated. Use `conda.gateways.disk.delete.rm_rf` instead. (#14094)
- Mark `conda.gateways.disk.delete.move_to_trash` as pending deprecated. Use `conda.gateways.disk.delete.rm_rf` instead. (#14094)
- Mark `conda.gateways.disk.delete.move_path_to_trash` as pending deprecated. Use `conda.gateways.disk.delete.rm_rf` instead. (#14094)
- Mark `conda.exports.move_to_trash` as pending deprecation. Use `conda.gateways.disk.delete.rm_rf` instead. (#14118)
- Mark `conda.activate.ensure_binary` as pending deprecation. Use `conda.common.compat.ensure_binary` instead. (#14144)
- Mark `conda.activate.ensure_fs_path_encoding` as pending deprecation. (#14144)

- Mark `conda.common.compat.six_with_metaclass` as pending deprecation. Use class' `metaclass=` keyword argument instead. (#14144)
- Mark `conda.common.compat.open` as pending deprecation. Use `conda.common.compat.open_utf8` instead. (#14144, #14169)
- Mark `conda.common.compat.ensure_unicode` as pending deprecation. (#14144)
- Mark `conda.common.compat.ensure_fs_path_encoding` as pending deprecation. (#14144)
- Mark `conda.common.compat.FILESYSTEM_ENCODING` as pending deprecation. (#14144, #14169)
- Mark `conda.activate.native_path_to_unix` as pending deprecation. Use `conda.common.path.win_path_to_unix` instead. (#14157)
- Mark `conda.activate.unix_path_to_native` as pending deprecation. Use `conda.common.path.unix_path_to_win` instead. (#14157)
- Mark `conda.activate._Cygpath` as pending deprecation. Use `conda.common.path._cygpath` instead. (#14157)
- Mark `conda.activate._Cygpath.RE_UNIX` as pending deprecation. Use `conda.common.path._cygpath.RE_WIN_DRIVE` instead. (#14157)
- Mark `conda.activate._Cygpath.translate_unix` as pending deprecation. Use `conda.common.path._cygpath._to_unix_drive` instead. (#14157)
- Mark `conda.activate._Cygpath.RE_DRIVE` as pending deprecation. Use `conda.common.path._cygpath.RE_UNIX_DRIVE` instead. (#14157)
- Mark `conda.activate._Cygpath.translation_drive` as pending deprecation. Use `conda.common.path._cygpath._to_win_drive` instead. (#14157)
- Mark `conda.activate._Cygpath.RE_MOUNT` as pending deprecation. Use `conda.common.path._cygpath.RE_UNIX_MOUNT` instead. (#14157)
- Mark `conda.activate._Cygpath.translation_mount` as pending deprecation. Use `conda.common.path._cygpath._to_win_mount` instead. (#14157)
- Mark `conda.activate._Cygpath.RE_ROOT` as pending deprecation. Use `conda.common.path._cygpath.RE_UNIX_ROOT` instead. (#14157)
- Mark `conda.activate._Cygpath.translation_root` as pending deprecation. Use `conda.common.path._cygpath._to_win_root` instead. (#14157)
- Mark `conda.utils.unix_path_to_win` as pending deprecation. Use `conda.common.path.unix_path_to_win` instead. (#14157)
- Deprecate `conda.env.specs.binstar` module. (#14158 via #14160)
- Deprecate `conda env [create|update] REMOTE_DEFINITION`. Use `conda env [create|update] --file=URL` instead. (#14158 via #14160)
- Mark `conda.testing.integration.BIN_DIRECTORY` as pending deprecation. Use `conda.common.path.BIN_DIRECTORY` instead. (#14188)
- Mark `conda.common.path.get_bin_directory_short_path()` as pending deprecation. Use `conda.common.path.BIN_DIRECTORY` instead. (#14188)
- Mark `conda.common.path.is_private_env_name` as pending deprecation. (#14189)
- Mark `conda.common.path.is_private_env_path` as pending deprecation. (#14189)
- Mark `conda shell.SHELL commands` as pending deprecation. Use `conda commands` instead. (#14215)

- The `defaults` multichannel will stop being the (implicit) default value for `channels`. Users relying on this behavior are encouraged to run `conda config --add channels defaults`. This is pending deprecation, and will be fully deprecated in 25.3. (#14178 via #14227)
- `conda config --add/--append channels ...` will warn when `defaults` is implicitly added. In `conda 25.3`, this behavior will be removed and users should run `conda config --add/--append channels defaults` explicitly if needed. Conda distribution installers like miniforge or miniconda will pre-configure conda channels during installation. (#12356 via #14227)
- Without an explicit `channels` configuration (via `condarc` files, environment variables, or CLI flags), conda will warn about using `defaults` implicitly. In 25.3, an empty list will be used. (#14227)
- Mark the `restore_free_channel` configuration option as pending deprecation. Add `https://repo.anaconda.com/pkgs/free` to your channel list after `defaults` instead. (#14231 via #14269)
- Mark `conda.testing.CondaCLIFixture` as pending deprecation. Use `conda.testing.fixtures.CondaCLIFixture` instead. (#14243)
- Mark `conda.testing.conda_cli` as pending deprecation. Use `conda.testing.fixtures.conda_cli` instead. (#14243)
- Mark `conda.testing.PathFactoryFixture` as pending deprecation. Use `conda.testing.fixtures.PathFactoryFixture` instead. (#14243)
- Mark `conda.testing.path_factory` as pending deprecation. Use `conda.testing.fixtures.path_factory` instead. (#14243)
- Mark `conda.testing.TmpEnvFixture` as pending deprecation. Use `conda.testing.fixtures.TmpEnvFixture` instead. (#14243)
- Mark `conda.testing.tmp_env` as pending deprecation. Use `conda.testing.fixtures.tmp_env` instead. (#14243)
- Mark `conda.testing.TmpChannelFixture` as pending deprecation. Use `conda.testing.fixtures.TmpChannelFixture` instead. (#14243)
- Mark `conda.testing.tmp_channel` as pending deprecation. Use `conda.testing.fixtures.tmp_channel` instead. (#14243)
- Mark `conda.testing.context_aware_monkeypatch` as pending deprecation. Use `conda.testing.fixtures.context_aware_monkeypatch` instead. (#14243)
- Mark `conda.testing.tmp_pkgs_dir` as pending deprecation. Use `conda.testing.fixtures.tmp_pkgs_dir` instead. (#14243)
- Mark `conda.testing.tmp_envs_dir` as pending deprecation. Use `conda.testing.fixtures.tmp_envs_dir` instead. (#14243)

Docs

- Document `--platform` flag for `conda [env] create`. (#14100 via #14181)

Other

- Update `xonsh` support to accomodate deprecated import path. (#14047)

Contributors

- @anki-code made their first contribution in https://github.com/conda/conda/pull/14047
- @beeankha
- @conda-bot

- @dholth
- @jaimergp
- @jezdez
- @kenodegard
- @zklaus
- @ForgottenProgramme
- @scw
- @SylvainCorlay made their first contribution in https://github.com/conda/conda/pull/14226
- @travishathaway
- @dependabot[bot]
- @pre-commit-ci[bot]

4.5.38 24.7.1 (2024-07-24)

Bug fixes

- Revert potential regression introduced in #13975. The previously forbidden characters (`^`, `%`, `!`, `=`, `(`, `)`, `\`) are allowed again until the impact is assessed. (#14065)

Contributors

- @jaimergp

4.5.39 24.7.0 (2024-07-17)

Enhancements

- Add a new `reporters` setting for configuring output. (#13736)
- Report traceback of plugin loading errors with verbosity 2 or higher (`-vv` or more). (#13742 via #13846)
- Skip checking for `.pyc` and `.pyo` files in the `conda doctor` "missing files" health check. (#13370 via #13931)
- **Breaking change** `conda list --explicit` will not print authentication details by default. A new flag `--auth` has been added so folks can opt-in to this behaviour. (#13936)
- Print transaction report for `@EXPLICIT` lockfile installs too. (#13940)
- Do not require `-n/--name` or `-p/--prefix` if `conda create` is invoked with `--dry-run`. (#13941)
- Add an `envvars_force_uppercase` setting which defaults to `True`, uppercasing all environment variables (thereby justifying `conda`'s current behaviour); when `envvars_force_uppercase` is set to `False`, conda will only save preserved-case variable names. (#13713 via #13943)
- Alias `conda env list` command to `conda info --envs`. (#13972)

Bug fixes

- Improve treatment of logger levels. (#13735)
- Mask authentication details in `conda-meta/*.json` metadata. (#13937)
- Mask Anaconda.org tokens in verbose logs. (#13939, #13987)
- Fix parsing error when history file only contains a single commented line. (#13960)

- Add missing `emscripten` and `wasi` entries to the recognized platforms, and `wasm32` to the recognized architectures. (#13095)
- Fix checksum comparisons in `conda.gateways.connection.download.download()` to be case insensitive. (#13969)
- Disallow some more characters in Windows for prefix names (`^`, `%`, `!`, `=`, `(`, `)`, `\`). These characters complicate or prevent environment activation if present. (#12558 via #13975)
- Fix caching when `repodata.json` contains `\r\n` line endings. (#14002 via #14003)
- Fix `conda.core.portability.binary_replace` not matching chunks that end with `\n`. (#14043 via #14044)

Deprecations

- Mark `conda.gateways.logging.initialize_root_logger` as pending deprecation. (#13735, #14046)
- Mark `conda.cli.main_env_list.execute` as pending deprecation. Use `conda.cli.main_info.execute` instead. (#13972)
- Revert `--all` deprecation in `conda info`. (#14004)
- Mark `conda.exports.iteritems` as pending deprecation. Use builtin `dict.items()` instead. (#14034)
- Mark `conda.exports.Completer` as pending deprecation. (#14034)
- Mark `conda.exports.InstalledPackages` as pending deprecation. (#14034)
- Mark `conda.exports.KEYS` as pending deprecation. (#14034)
- Mark `conda.exports.KEYS_DIR` as pending deprecation. (#14034)
- Mark `conda.exports.hash_file` as pending deprecation. (#14034)
- Mark `conda.exports.verify` as pending deprecation. (#14034)
- Mark `conda.exports.symlink_conda` as pending deprecation. Use `conda.activate` instead. (#14034)
- Mark `conda.exports._symlink_conda_hlp` as pending deprecation. Use `conda.activate` instead. (#14034)
- Mark `conda.exports.win_conda_bat_redirect` as pending deprecation. Use `conda.activate` instead. (#14034)
- Mark `conda.utils.win_path_to_cygwin` as pending deprecation. Use `conda.common.path.win_path_to_unix` instead. (#14034)
- Mark `conda.utils.cygwin_path_to_win` as pending deprecation. Use `conda.utils.unix_path_to_win` instead. (#14034)
- Mark `conda.utils.translate_stream` as pending deprecation. (#14034)
- Mark `conda.utils.unix_shell_base` as pending deprecation. Use `conda.activate` instead. (#14034)
- Mark `conda.utils.msys2_shell_base` as pending deprecation. Use `conda.activate` instead. (#14034)
- Mark `conda.utils.shells` as pending deprecation. Use `conda.activate` instead. (#14034)

Docs

- Clarify proxy server configuration in documentation. (#12856)
- Add type hints and doc strings to `conda.core.envs_manager`. (#13817)
- Add logging overview as deep-dive. (#13735)
- Update conda cheatsheet text and add it directly to cheatsheet page. (#13889)

- Add cheatsheet PDF download to cheatsheet page. (#13909)
- Add `ssl_verify: truststore` to the user guide. (#13935)
- Fix the help text of the `satisfied-skip-solve` flag. (#13946)
- Add a section explaining how to correctly raise exceptions from a plugin. (#13741 via #13950)
- Add new article for configuring `envs_dirs` and `pkgs_dirs`. (#13954)

Other

- Replace calls to `logger.warn` with `logger.warning`. (#13963)

Contributors

- @beeankha
- @conda-bot
- @erik-whiting made their first contribution in https://github.com/conda/conda/pull/13877
- @ifitchet
- @jaimergp
- @jezdez
- @kathatherine
- @kelvinou01 made their first contribution in https://github.com/conda/conda/pull/14044
- @kenodegard
- @zklaus
- @ForgottenProgramme
- @Nathann03
- @zeehio
- @skupr-anaconda made their first contribution in https://github.com/conda/conda/pull/13946
- @tl-hbk
- @DerThorsten made their first contribution in https://github.com/conda/conda/pull/13962
- @travishathaway
- @dependabot[bot]
- @padeoe made their first contribution in https://github.com/conda/conda/pull/12856
- @pre-commit-ci[bot]

4.5.40 24.5.0 (2024-05-08)

Enhancements

- Report which `MatchSpec` item caused `Invalid*Spec` exceptions for more informative error messages. (#11203 via #13598)
- MSYS2 packages can now use the upstream installation prefixes. (#13649)
- Add support for CEP-15 `base_url` field in `repodata.json`. (#13137 via #13744)

- In custom channel settings, allow specification of channel URLs using a glob-like wildcard pattern, e.g. for user with auth handler plugins. (#13778 via #13779)

Bug fixes

- Fix `conda notices --json` to correctly output JSON. (#13561)
- Fix prefix replacement for Windows `subdir` on Unix. (#13689)

Deprecations

- Mark `conda.plan._get_best_prec_match` as pending deprecation. Use `conda.misc._get_best_prec_match` instead. (#12421)
- Mark `conda.plan._handle_menuinst` as pending deprecation. (#12421)
- Mark `conda.plan._inject_UNLINKLINKTRANSACTION` as pending deprecation. (#12421)
- Mark `conda.plan._plan_from_actions` as pending deprecation. (#12421)
- Mark `conda.plan.add_defaults_to_specs` as pending deprecation. (#12421)
- Mark `conda.plan.add_unlink` as pending deprecation. (#12421)
- Mark `conda.plan.display_actions` as pending deprecation. (#12421)
- Mark `conda.plan.execute_actions` as pending deprecation. (#12421)
- Mark `conda.plan.get_blank_actions` as pending deprecation. (#12421)
- Mark `conda.plan.install_actions` as pending deprecation. (#12421)
- Mark `conda.plan.print_dists` as pending deprecation. (#12421)
- Mark `conda.plan.revert_actions` as pending deprecation. Use `conda.cli.install.revert_actions` instead. (#12421)
- Mark `conda.plan` as an entrypoint as pending deprecation. (#12421)
- Mark `conda.activate._Activator.add_export_unset_vars` as pending deprecation. Use `conda.activate._Activator.get_export_unset_vars` instead. (#13720)
- Mark `conda.activate._Activator.get_scripts_export_unset_vars` as pending deprecation. Use `get_scripts_export_unset_vars` helper function in `test_activate.py` instead. (#13720)
- Mark `conda.activate._Activator._get_path_dirs(extra_library_bin)` as pending deprecation. (#13720)
- Mark `conda.activate.JSONFormatMixin.get_scripts_export_unset_vars` as pending deprecation. Use `conda.activate._Activator.get_export_unset_vars` instead. (#13720)
- Mark `conda.gateways.logging.trace` as pending deprecation. Use `Logger.log(conda.common.constants.TRACE, msg)` instead. (#13732)
- Mark `conda create --mkdir` as pending deprecation. The argument is redundant and unnecessary. (#13751)
- Mark `conda install --mkdir` as pending deprecation. Use `conda create` instead. (#13751)
- Mark `conda._vendor.frozendict` as pending deprecation. Use `frozendict` instead. (#13767 via #13766)
- Mark `conda.auxlib.collection.make_immutable` as pending deprecation. Use `frozendict.deepfreeze` instead. (#13801)
- Mark `conda.plan.execute_plan` as pending deprecation. (#13869)
- Mark `conda.plan.execute_instructions` as pending deprecation. (#13869)

- Mark `conda.plan._update_old_plan` as pending deprecation. (#13869)

Docs

- Add type hints and doc strings to `conda.core.index`. (#13816)

Other

- Remove `setuptools` remainings (`MANIFEST.in`, `wheel` build dependency) not required since the move to `hatch` in #12509. (#13684)
- Remove and update any imports inside conda that is importing from `conda/exports.py`. (#13869)

Contributors

- @beeankha
- @conda-bot
- @dbast
- @ifitchet made their first contribution in https://github.com/conda/conda/pull/13649
- @isuruf
- @jaimergp
- @jezdez
- @kenodegard
- @zklaus
- @ForgottenProgramme
- @mattkram
- @Nathann03 made their first contribution in https://github.com/conda/conda/pull/13816
- @dwr-psandhu made their first contribution in https://github.com/conda/conda/pull/13770
- @travishathaway
- @dependabot[bot]
- @pre-commit-ci[bot]

4.5.41 24.4.0 (2024-04-24)

Enhancements

- For Windows users, the stub executables used for Python entrypoints in packages are now codesigned. (#13721)

Contributors

- @dholth
- @jezdez
- @Callek made their first contribution in https://github.com/conda/conda/pull/13721

4.5.42 24.3.0 (2024-03-12)

Enhancements

- Show first few characters of undecodeable response if `repodata.json` raises `JSONDecodeError`. (#11804)
- Update `conda.gateways.subprocess.subprocess_call` to use `text=True` to avoid manual encoding/decoding. (#13240)
- Add a new plugin hook giving plugin authors the ability to define new settings. (#13554)
- Optimize module imports to speed up `conda activate`. (#13567 via #13568)
- Move `conda env export` to `conda export` and alias the old command to the new command. (#13577)
- Report progress while running `conda install --revision <idx>`. (#13611)
- Add `conda.testing.tmp_channel` pytest fixture to create a temporary local channel for testing. (#13634)

Bug fixes

- Print traceback on `KeyboardInterrupt` instead of raising another `AttributeError` exception, when conda debugging logs are enabled. (#13531)
- Parse integer channel notice IDs as `str` instead of raising an exception. (#13543)
- Add direct runtime dependency on `zstandard` for use when downloading `repodata.json.zst`. (#13551)
- Fallback to `repodata.json` if `repodata.json.zst` cannot be decompressed as `zstandard`. (#13558)
- `conda rename` command no longer throws an error when conda is not active. (#13565)
- Fallback to `repodata.json` from `repodata.json.zst` on most 4xx error codes. (#13573)
- Fix excess resource usage by log handling when fetching repodata. (#13541 via #13628)
- Re-enable `--subdir` and `--platform` flags to be available for `conda env create` command. (#13632)
- Fix `__archspec` virtual package on Windows to return microarchitecture instead of the default `x86_64`. (#13641)
- Check `Content-Length` is nonzero before calculating progress, avoiding a possible `ZeroDivisionError`. (#13653, #13671)

Deprecations

- Discontinue custom docker images. Use images provided by Anaconda Inc. or conda-forge instead. (#13162)
- Mark `conda.common.compat.encode_arguments` as pending deprecation. (#13240)
- Remove `conda.export.handle_proxy_407`. (#13629)
- Mark `conda.testing.integration.make_temp_channel` as pending deprecation. Use `conda.testing.tmp_channel` fixture instead. (#13634)
- Mark `conda.testing.integration.running_a_python_capable_of_unicode_subprocessing` as pending deprecation. (#13634)
- Mark `conda.testing.integration.set_tmpdir` as pending deprecation. Use `tmp_path`, `conda.testing.path_factory`, or `conda.testing.tmp_env` instead. (#13634)
- Mark `conda.testing.integration._get_temp_prefix` as pending deprecation. Use `tmp_path`, `conda.testing.path_factory`, or `conda.testing.tmp_env` instead. (#13634)
- Mark `conda.testing.integration.make_temp_prefix` as pending deprecation. Use `tmp_path`, `conda.testing.path_factory`, or `conda.testing.tmp_env` instead. (#13634)

- Mark `conda.testing.integration.FORCE_temp_prefix` as pending deprecation. Use `tmp_path`, `conda.testing.path_factory`, or `conda.testing.tmp_env` instead. (#13634)
- Mark `conda.testing.integration.create_temp_location` as pending deprecation. Use `tmp_path` or `conda.testing.path_factory` instead. (#13634)
- Mark `conda.testing.integration.tempdir` as pending deprecation. Use `tmp_path` or `conda.testing.path_factory` instead. (#13634)
- Mark `conda.testing.integration.reload_config` as pending deprecation. Use `conda.base.context.reset_context` instead. (#13634)
- Postpone `conda.base.context.Context.conda_exe` deprecation to `conda 24.9`. (#13634)
- Postpone `conda.testing.integration.run_command` deprecation to `conda 25.3`. (#13634)
- Postpone loading subcommands from executables deprecation to `conda 25.3`. (#13634)
- Remove vendored `conda._vendor.boltons`. Use `boltons` package instead. (#12681 via #13634)
- Remove `conda.auxlib.packaging`. Use a modern build system instead; see https://packaging.python.org/en/latest/tutorials/packaging-projects#creating-pyproject-toml for more details. (#12681 via #13634)
- Remove `conda env create --force`. Use `conda env create --yes` instead. (#12681 via #13634)
- Remove `conda info PACKAGE`. Use `conda search PACKAGE --info` instead. (#12681 via #13634)
- Remove `conda.core.subdir_data.fetch_repodata_remote_request`. Use `conda.core.subdir_data.SubdirData.repo_fetch.fetch_latest_parsed` instead." (#12681 via #13634)
- Remove `conda.exports.memoized`. Use `functools.lru_cache` instead. (#12681 via #13634)
- Remove `conda.gateways.disk.read._digest_path`. Use `conda.gateways.disk.read.compute_sum` instead. (#12681 via #13634)
- Remove `conda.gateways.disk.read.compute_md5sum`. Use `conda.gateways.disk.read.compute_sum(path, "md5")` instead. (#12681 via #13634)
- Remove `conda.gateways.disk.read.compute_sha256sum`. Use `conda.gateways.disk.read.compute_sum(path, "sha256")` instead. (#12681 via #13634)
- Remove `conda.instructions.PREFIX`. (#12681 via #13634)
- Remove `conda.instructions.PREFIX_CMD`. (#12681 via #13634)
- Remove `conda.testing.encode_for_env_var`. (#12681 via #13634)
- Remove `conda.testing.conda_check_versions_aligned`. (#12681 via #13634)
- Remove `conda.testing.helpers.run_inprocess_conda_command`. Use `conda.testing.tmp_env` instead. (#12681 via #13634)
- Remove `conda.testing.helpers.capture_json_with_argv`. (#12681 via #13634)
- Remove `conda.testing.integration.get_conda_list_tuple`. Use `conda.core.prefix_data.PrefixData.get` instead. (#12681 via #13634)
- Remove `conda.utils.md5_file`. Use `conda.gateways.disk.read.compute_sum(path, "md5")` instead. (#12681 via #13634)
- Remove `conda.utils.hashsum_file`. Use `conda.gateways.disk.read.compute_sum` instead. (#12681 via #13634)
- Remove `conda.utils.safe_open`. Use `open` instead. (#12681 via #13634)
- Remove `python -m conda_env`. Use `conda env` or `python -m conda env` instead. (#12681 via #13634)

- Remove `conda_env.env.load_from_directory`. (#12681 via #13634)
- Remove `conda_env.pip_util.get_pip_version`. (#12681 via #13634)
- Remove `conda_env.pip_util.PipPackage`. (#12681 via #13634)
- Remove `conda_env.pip_util.installed`. (#12681 via #13634)
- Remove `conda_env.pip_util._canonicalize_name`. (#12681 via #13634)
- Remove `conda_env.pip_util.add_pip_installed`. (#12681 via #13634)

Docs

- Update the navigation links for Miniconda. (#13572)

Other

- Remove `dev/*` scripts in favor of `conda-incubator/setup-miniconda` GitHub Action in `.github/workflows/tests.yml`. (#13162)
- Stop chaining commands for steps in `.github/workflows/tests.yml`. (#12418 via #13162)
- Modernize tests. (#13547, #13292)
- Run GitHub tests workflow also on `osx-arm64` (aka Apple Silicon) runners. Enable `osx-arm64` canary builds. Fix or disable broken tests. (#13617)
- Upload stable release artifacts to GitHub releases during releases. (#13399)

Contributors

- @beeankha
- @conda-bot
- @dbast
- @dholth
- @FFY00
- @isuruf
- @jaimergp
- @jezdez
- @jjhelmus
- @kenodegard
- @zklaus made their first contribution in https://github.com/conda/conda/pull/13579
- @ForgottenProgramme
- @mbargull
- @travishathaway
- @pre-commit-ci[bot]

4.5.43 24.1.2 (2024-02-15)

Bug fixes

- Fix deprecated `fetch_repodata_remote_request` when `repodata_use_zst` is enabled. (#13595)

Contributors

- @dholth

4.5.44 24.1.1 (2024-02-12)

Bug fixes

- Fallback to `repodata.json` if `repodata.json.zst` cannot be decompressed as zstandard. (#13558)
- Fallback to `repodata.json` from `repodata.json.zst` on most 4xx error codes. (#13573)

Contributors

- @dholth

4.5.45 24.1.0 (2024-01-24)

Special announcement

The `conda_env.*` modules have been merged into the `conda` package!

To improve the integration of the `conda env` subcommand (previously standalone), we've moved its code into the `conda` package, while allowing old `conda env` commands to still work via Python import redirects. This is a first step of many to improving the user experience of the conda command line interface related to environment management. (#13168)

Enhancements

- Verify signatures on to-be-installed packages instead of on all packages. (#11545, #13053)
- Add new `pre-solves` and `post-solves` plugin hooks. (#13053)
- Add support for Python 3.12. (#13072)
- Check `repodata.json.zst` for faster repodata downloads. (#13256)
- Add `--skip-flexible-search` option in `conda search` to skip flexible search. (#13315)
- Provide a more useful warning when attempting to rename a non-existent prefix. (#13387)
- Add a new flag `--keep-env` to be used with `conda remove --all`. It allows users to delete all packages in the environment while retaining the environment itself. (#13419)
- Add a Y/N prompt warning users that `conda env remove` and `conda remove --all` deletes not only the conda packages but the entirety of the specified environment. (#13440)
- Add `--repodata-use-zst/--no-repodata-use-zst` flag to control `repodata.json.zst` check; corresponding `repodata_use_zst: true/false` for `.condarc`. Default is to check for `repodata.json.zst`. Disable if remote returns unparseable `repodata.json.zst` instead of correct data or 404. (#13504)

Bug fixes

- Create the ~/.conda directory before trying to write to the environments.txt file. (#13338)
- Ensure PackageRecord.timestamp is dumped in milliseconds. (#13483)
- Fix error when setting a non-default repodata filename via CONDA_REPODATA_FNS. (#13490)
- Fix the config file location where the integrated Anaconda client gateway loads user configuration from. This is a regression introduced in conda 23.11.0 when the platformdirs library was adopted. (#13517 via #13520)
- Interpret missing Cache-Control header as max-age=0 instead of exception. (#13522)

Deprecations

- Mark conda_env/cli/common as pending deprecation. Use conda.cli.common instead. (#13168)
- Mark conda_env/cli/main_config as pending deprecation. Use conda.cli.main_env_config instead. (#13168)
- Mark conda_env/cli/main_create as pending deprecation. Use conda.cli.main_env_create instead. (#13168)
- Mark conda_env/cli/main_export as pending deprecation. Use conda.cli.main_env_export instead. (#13168)
- Mark conda_env/cli/main_list as pending deprecation. Use conda.cli.main_env_list instead. (#13168)
- Mark conda_env/cli/main_remove as pending deprecation. Use conda.cli.main_env_remove instead. (#13168)
- Mark conda_env/cli/main_update as pending deprecation. Use conda.cli.main_env_update instead. (#13168)
- Mark conda_env/cli/main_vars as pending deprecation. Use conda.cli.main_env_vars instead. (#13168)
- Mark conda_env/env as pending deprecation. Use conda.env.env instead. (#13168)
- Mark conda_env/installers/base as pending deprecation. Use conda.env.installers.base instead. (#13168)
- Mark conda_env/installers/conda as pending deprecation. Use conda.env.installers.conda instead. (#13168)
- Mark conda_env/installers/pip as pending deprecation. Use conda.env.installers.pip instead. (#13168)
- Mark conda_env/pip_util as pending deprecation. Use conda.env.pip_util instead. (#13168)
- Mark conda_env/specs as pending deprecation. Use conda.env.specs instead. (#13168)
- Mark conda_env/specs/binstar as pending deprecation. Use conda.env.specs.binstar instead. (#13168)
- Mark conda_env/specs/requirements as pending deprecation. Use conda.env.specs.requirements instead. (#13168)
- Mark conda_env/specs/yaml_file as pending deprecation. Use conda.env.specs.yaml_file instead. (#13168)
- Mark conda.testing.integration.make_temp_package_cache as pending deprecation. (#13511)

Docs

- Update Getting Started documentation in User Guide. (#13190)
- Add GoatCounter (https://www.goatcounter.com/) as an analytics tool. (#13384)
- Add type hints and doc strings to `conda.cli.main_info`. (#13445)
- Add type hints and doc strings to `conda.cli.main_search`. (#13465)

Other

- Add type hinting for `VersionOrder` class. (#13380)
- Re-enable and apply `pyupgrade` via `ruff`. (#13272, #13433)
- Start tracking performance in continuous integration and automatically report about it in pull requests. (#13460)
- Add `tmp_pkgs_dir` fixture to replace `make_temp_package_cache`. (#13511)
- Improve lock API for the repodata cache. (#13455)

Contributors

- @beeankha
- @conda-bot
- @dbast
- @dholth
- @jaimergp
- @jezdez
- @johnnynunez
- @kathatherine
- @kenodegard
- @ForgottenProgramme
- @marcoesters
- @mfansler
- @schuylermartin45 made their first contribution in https://github.com/conda/conda/pull/13385
- @travishathaway
- @pre-commit-ci[bot]
- @samhaese made their first contribution in https://github.com/conda/conda/pull/13465

4.5.46 23.11.0 (2023-11-30)

Special announcement

New `menuinst` v2 support!

`conda` has supported Start menu items on Windows for a long time. This is what allows users to open up their Miniconda prompt on CMD (Command Prompt) with an initialized `conda` session. This menu item (or shortcut) creation logic is provided by the `menuinst` package.

With the release of 23.11.0, conda now supports menuinst v2, which enables the same experience across Windows, Linux, and macOS. This means package builders will be able to provide desktop icons across all operating systems, which can be especially useful for GUI applications. See the documentation for more details.

If you don't want conda to create shortcuts, you can disable it via:

- shortcuts: false entry in your .condarc configuration
- CONDA_SHORTCUTS=false environment variable
- --no-shortcuts command-line flag

Enhancements

- Add support for menuinst v2, enabling shortcuts across all platforms (Windows, Linux, macOS) using a new JSON schema (see CEP-11). Retain support for old v1-style JSON menus. (#11882)
- Stop using vendored chardet package by requests/pip; explicitly depend on charset_normalizer. (#13171)
- Introduce a new plugin hook, CondaHealthCheck, as part of conda doctor. (#13186)
- Include activate and deactivate in the --help command list. (#13191)
- Prioritize download of larger packages to prevent smaller ones from waiting. (#13248)
- Display the used solver in conda info output for debugging purposes. (#13265)
- Add __conda virtual package. (#13266)
- Switch from appdirs to platformdirs. (#13306)
- Implement resume capability for interrupted package downloads. (#8695)

Bug fixes

- Log expected JLAP range-request errors at info level, occurring when the remote file has rolled over. (#12913)
- Fix a bug causing an error when options like --debug are used without specifying a command. (#13232)
- Improve CTRL-C (cancellation) handling for parallel download threads. (#13234)
- Allow overriding of CONDA_FETCH_THREADS/fetch_threads to set parallel package downloads beyond the default 5. (#13263)
- Require requests >=2.28 for enhanced response.json() exception handling. (#13346)
- Apply callback=reset_context in conda.plan to resolve conda-build + conda-libmamba-solver incompatibilities. (conda-libmamba-solver#393 and conda-libmamba-solver#386 via #13357)

Deprecations

- Deprecate conda.plugins.subcommands.doctor.health_checks.display_health_checks function. (#13186)
- Deprecate conda.plugins.subcommands.doctor.health_checks.display_report_heading function. (#13186)
- Remove ruamel_yaml fallback; use ruamel.yaml exclusively. (#13218)
- Deprecate conda.gateways.anaconda_client.EnvAppDirs in favor of platformdirs. (#13306)
- Mark conda._vendor.cpuinfo for pending deprecation. (#13313)
- Deprecate conda._vendor.distro in favor of the distro package. (#13317)

Docs

- Add the `conda-sphinx-theme` to the conda documentation. (#13298)
- Update specific pages to remove redundant TOC entries. (#13298)
- Include instructions on updating `conda` in the main `README.md`. (#13343)

Other

- Add a lighter weight s3 test; update embedded test package index. (#13085)
- Refactor code to use lazy imports for all relative imports in `conda.cli.main_*`, and separate argparse configuration functions from `conda.cli.conda_argparse` to their respective `conda.cli.main_*` modules. (#13173)
- Move custom `argparse.Actions` to `conda.cli.actions` (e.g., `NullCountAction`), and relocate helper argparse functions to `conda.cli.helpers` (e.g., `add_parser_prefix`). (#13173)
- Update upper bound for `ruamel.yaml` to `<0.19` following the release of `0.18`. (#13258)
- Replace `black` with `ruff format` in pre-commit. (#13272)

Contributors

- @AniketP04 made their first contribution in https://github.com/conda/conda/pull/13224
- @beeankha
- @13rac1 made their first contribution in https://github.com/conda/conda/pull/13191
- @conda-bot
- @dholth
- @eltociear
- @jaimergp
- @jezdez
- @kathatherine
- @kenodegard
- @kennethlaskoski made their first contribution in https://github.com/conda/conda/pull/13322
- @ForgottenProgramme
- @marcoesters
- @opoplawski
- @scruel made their first contribution in https://github.com/conda/conda/pull/13274
- @travishathaway
- @gfggithubleet made their first contribution in https://github.com/conda/conda/pull/13270
- @pre-commit-ci[bot]

4.5.47 23.10.0 (2023-10-30)

Special announcement

This is an announcement about an important change in conda's functionality:

With this 23.10.0 release we are changing the default solver of conda to `conda-libmamba-solver`!

The previously "classic" solver is based on pycosat/Picosat and will remain part of conda for the foreseeable future, a fallback is possible and available.

Why are we switching the solver?

In short: to make conda faster and more accurate.

A "solver" is the core component of most package managers; it calculates which dependencies (and which version of those dependencies) to install when a user requests to install a package from a package repository. To address growth-related challenges within the conda ecosystem, the conda maintainers, alongside partners Anaconda, Quansight and QuantStack, introduced a new conda dependency solver based on the Mamba project in December 2022.

Since July 2023, the `conda-libmamba-solver` plugin has been included in all major conda ecosystem installers (miniforge, miniconda, mambaforge and Anaconda Distribution), but was disabled by default. As soon as these installers are updated to contain conda 23.10.0 or later, they will automatically default to using the conda-libmamba-solver plugin.

What can I do if this update doesn't work for me?

If the new solver is not working as you expect:

- Check if the behavior you are observing is a known issue or a deliberate change.
- If that's not the case, please consider submitting a bug report or feature request in the conda-libmamba-solver repository.
- If necessary, you can go back to using the `classic` solver without modifying your conda installation:
  - When possible, pass the command line option `--solver=classic` to your `conda` calls.
  - Otherwise (e.g. for `conda build ...` or `constructor ...`), set the environment variable `CONDA_SOLVER=classic`.
  - For permanent changes, use the conda configuration system: `conda config --set solver classic`.

Where can I learn more about conda-libmamba-solver?

The documentation of the `conda-libmamba-solver` plugin can be found on conda.github.io/conda-libmamba-solver.

For more information about the `conda-libmamba-solver` rollout plan, please also see our blog post from earlier this year.

Enhancements

- Provide `--platform` and `--subdir` flags to create environments for non-native platforms, remembering that choice in future operations. (#11505 via #11794)
- IMPORTANT: Set `solver: libmamba` as the new default solver. (#12984)

Bug fixes

- Check name of symlink, not its target against valid configuration file names (`condarc`, `.condarc`, or `*.yml`/`yaml`). (#12956)
- Have `conda doctor` ignore blank lines in `~/.conda/environments.txt`. (#12984)

Deprecations

- Mark `conda.cli.main.generate_parser` as pending deprecation. Use `conda.cli.conda_argparse.generate_parser` instead. (#13144)
- Mark `conda.auxlib.collection.firstitem` as pending deprecation. (#13144)
- Mark `conda.auxlib.collection.call_each` as pending deprecation. (#13144)
- Mark `conda.auxlib.compat.NoneType` as pending deprecation. (#13144)
- Mark `conda.auxlib.compat.primative_types` as pending deprecation. (#13144)
- Mark `conda.auxlib.compat.utf8_writer` as pending deprecation. (#13144)
- Mark `conda.auxlib.exceptions.AuthenticationError` as pending deprecation. (#13144)
- Mark `conda.auxlib.exceptions.NotFoundError` as pending deprecation. (#13144)
- Mark `conda.auxlib.exceptions.InitializationError` as pending deprecation. (#13144)
- Mark `conda.auxlib.exceptions.SenderError` as pending deprecation. (#13144)
- Mark `conda.auxlib.exceptions.AssignmentError` as pending deprecation. (#13144)
- Mark `conda.auxlib.type_coercion.boolify_truthy_string_ok` as pending deprecation. (#13144)
- Mark `conda.auxlib.type_coercion.listify` as pending deprecation. (#13144)
- Mark `conda.models.dist.IndexRecord` as pending deprecation for removal in 24.9. (#13193)
- Mark `conda.exports.fetch_index` as pending deprecation for removal in 24.9. Use `conda.core.index.fetch_index` instead. (#13194)

Other

- Constrain minimum conda-build version to >=`3.27`. (#13177)

Contributors

- @conda-bot
- @dholth
- @jaimergp
- @jezdez
- @kenodegard
- @timhoffm
- @pre-commit-ci[bot]

4.5.48 23.9.0 (2023-09-27)

Special announcement

This is an announcement about an important and positive future change in conda's functionality:

> We will change the default solver of conda to `conda-libmamba-solver` in a **special 23.10.0 release** in the near future!

You can already benefit from it *today* by configuring your conda installation to use it (e.g. by running `conda config --set solver libmamba`).

The current "classic" solver is based on pycosat/Picosat and will remain part of conda for the foreseeable future, a fallback is possible and available (see below).

Plan to change the default solver

Here is our updated plan to change the default solver, to better follow CEP 8 and reduce the potential impact on conda users:

- The upcoming, special 23.10.0 release will be dedicated to the switch of the default solver to `libmamba`.
- Users will be able to opt out of the `libmamba` solver and use the `classic` solver instead, by using one of these options:
  - the `--solver=classic` command line option,
  - the `CONDA_SOLVER=classic` environment variable or
  - running `conda config --set solver classic`.
- All development of `conda-libmamba-solver` plugin happens in the conda-libmamba-solver repo, including issue tracking.
- The documentation of the `conda-libmamba-solver` plugin can be found on conda.github.io/conda-libmamba-solver.

For more information about the `conda-libmamba-solver` rollout plan, please also see our blog post from earlier this year.

Context

A "solver" is the core component of most package managers; it calculates which dependencies (and which version of those dependencies) to install when a user requests to install a package from a package repository. To address growth-related challenges within the conda ecosystem, the conda maintainers, alongside partners Anaconda, Quansight and QuantStack, introduced a new conda dependency solver based on the Mamba project in December 2022.

Since July 2023, that `conda-libmamba-solver` plugin has been included in and automatically installed with all major conda ecosystem installers (miniforge, miniconda, mambaforge and Anaconda Distribution), *with the default solver configuration unchanged*.

Enhancements

- Improve speed of `fish` shell initialization. (#12811)
- Directly suppress use of binstar (conda) token when fetching trust metadata. (#12889)
- Add a new "auth handler" plugin hook for conda. (#12911)
- Lock index cache metadata by default. Added `--no-lock` option in case of problems, should not be necessary. Older `--experimental=lock` no longer has an effect. (#12920)
- Add `context.register_envs` option to control whether to register environments in `~/.conda/environments.txt` when they are created. Defaults to true. (#12924)
- Inject a new detailed output verbosity level (i.e., the old debug level `-vv` is now `-vvv`). (#12985, #12977, #12420, #13036)
- Add support for `truststore` to the `ssl_verify` config option, enabling conda to use the operating system certificate store (requires Python 3.10 or later). (#13075 and #13149)
- Add `emscripten-wasm32` and `wasi-wasm32` platforms to known platforms. (#13095)
- Adds the `py.typed` marker file to the `conda` package for compliance with PEP-561. (#13107)

- Import `boto3` only when S3 channels are used, saving startup time. (#12914)

Bug fixes

- When using pip dependencies with `conda env create`, check the directory permissions before writing to disk. (#11610)
- Hide `InsecureRequestWarning` for JLAP when `CONDA_SSL_VERIFY=false`, matching non-JLAP behavior. (#12731)
- Disallow ability to create a conda environment with a colon in the prefix. (#13044)
- Fix `AttributeError` logging response with nonexistent request when using JLAP with `file:///` URIs. (#12966)
- Do not show progress bars in non-interactive runs for cleaner logs. (#12982)
- Fix S3 bucket name. (#12989)
- Default `--json` and `--debug` to `NULL` so as to not override `CONDA_JSON` and `CONDA_DEBUG` environment variables. (#12987)
- `XonshActivator` now uses `source-bash` in non-interactive mode to avoid side-effects from interactively loaded RC files. (#13012)
- Fix `conda remove --all --json` output. (#13019)
- Update test data to stop triggering security scanners' false-positives. (#13034)
- Fix performance regression of basic commands (e.g., `conda info`) on WSL. (#13035)
- Configure conda to ignore "Retry-After" header to avoid the scenarios when this value is very large and causes conda to hang indefinitely. (#13050)
- Treat `JSONDecodeError` on `repodata.info.json` as a warning, equivalent to a missing `repodata.info.json`. (#13056)
- Fix sorting error for `conda config --show-sources --json`. (#13076)
- Catch `OSError` in `find_commands` to account for incorrect `PATH` entries on Windows. (#13125)
- Catch a `NotWritableError` when trying to find the first writable package cache dir. (#9609)
- `conda env update --prune` uses only the specs coming from `environment.yml` file and ignores the history specs. (#9614)

Deprecations

- Removed `conda.another_unicode()`. (#12948)
- Removed `conda._vendor.toolz`. (#12948, #13141)
- Removed `conda._vendor.tqdm`. (#12948)
- Removed `conda.auxlib.decorators.memoized` decorator. (#12948)
- Removed `conda.base.context.Context.experimental_solver`. (#12948)
- Removed `conda.base.context.Context.conda_private`. (#12948)
- Removed `conda.base.context.Context.cuda_version`. (#12948)
- Removed `conda.base.context.get_prefix()`. (#12948)
- Removed `conda.cli.common.ensure_name_or_prefix()`. (#12948)
- Removed `--experimental-solver` command line option. (#12948)

- Removed `conda.common.cuda` module. (#12948)
- Removed `conda.common.path.explode_directories(already_split)`. (#12948)
- Removed `conda.common.url.escape_channel_url()`. (#12948)
- Removed `conda.core.index.check_whitelist()`. (#12948)
- Removed `conda.core.solve._get_solver_class()`. (#12948)
- Removed `conda.core.subdir_data.read_mod_and_etag()`. (#12948)
- Removed `conda.gateways.repodata.RepodataState.load()`. (#12948)
- Removed `conda.gateways.repodata.RepodataState.save()`. (#12948)
- Removed `conda.lock` module. (#12948)
- Removed `conda_env.cli.common.stdout_json()`. (#12948)
- Removed `conda_env.cli.common.get_prefix()`. (#12948)
- Removed `conda_env.cli.common.find_prefix_name()`. (#12948)
- Remove import of deprecated cgi module by deprecating ftp STOR support. (#13013)
- Require `boto3` for S3 support and drop support for the older `boto` as it doesn't support our minimum required version of Python. (#13112)
- Reduce startup delay from deprecations module by using `sys._getframe()` instead of `inspect.stack()`. (#12919)

Other

- Use Ruff linter in pre-commit configuration (#12279)
- Remove unused `cache_path` arguments from `RepoInterface`/`JlapRepoInterface`; replaced by cache object. (#12927)

Contributors

- @beenje
- @beeankha
- @chbrandt
- @chenghlee
- @conda-bot
- @dbast
- @dholth
- @duncanmmacleod
- @gforsyth
- @eltociear
- @jaimergp
- @jezdez
- @jmcarpenter2 made their first contribution in https://github.com/conda/conda/pull/13034
- @kenodegard

- @ForgottenProgramme
- @Mon-ius made their first contribution in https://github.com/conda/conda/pull/12811
- @otaithleigh made their first contribution in https://github.com/conda/conda/pull/13035
- @psteyer made their first contribution in https://github.com/conda/conda/pull/11610
- @tarcisioe made their first contribution in https://github.com/conda/conda/pull/9614
- @travishathaway
- @wolfv made their first contribution in https://github.com/conda/conda/pull/13095
- @zeehio made their first contribution in https://github.com/conda/conda/pull/13075
- @pre-commit-ci[bot]

4.5.49 23.7.4 (2023-09-12)

Enhancements

- Use `os.scandir()` to find conda subcommands without `stat()` overhead. (#13033, #13067)

Bug fixes

- Fix S3 bucket name in test suite. (#12989)
- Fix performance regression of basic commands (e.g., `conda info`) on WSL. (#13035)
- Catch `PermissionError` raised by `conda.cli.find_commands.find_commands` when user's `$PATH` contains restricted paths. (#13062, #13089)
- Fix sorting error for `conda config --show-sources --json`. (#13076)

Contributors

- @beeankha
- @dholth
- @kenodegard
- @otaithleigh made their first contribution in https://github.com/conda/conda/pull/13035

4.5.50 23.7.3 (2023-08-21)

Bug fixes

- Fix regression for supporting conda executable plugins installed into non-base environments. (#13006)

Contributors

- @kenodegard

4.5.51 23.7.2 (2023-07-27)

Bug fixes

- Fix regression in parsing `--json` and `--debug` flags for executable plugins. (#12935, #12936)

Contributors

- @kenodegard

4.5.52 23.7.1 (2023-07-26)

Bug fixes

- Patch parsed args with pre_args to correctly parse `--json` and `--debug` arguments. (#12928, #12929)

Contributors

- @jezdez
- @kenodegard

4.5.53 23.7.0 (2023-07-25)

Enhancements

- Add `conda.deprecations.DeprecationHandler.action` helper to deprecate `argparse.Action`s. (#12493)
- Add support for the FreeBSD operating system and register `freebsd-64` as a known subdirectory for FreeBSD on x86-64. (#12647)
- Do not mock `$CONDA_PREFIX` when `--name` or `--prefix` is provided. (#12696)
- Add support for `sha256` filters in the MatchSpec syntax (e.g. `*[sha256=f453db4ffe2271ec492a2913af4e61d4a6c118201f07`
(#12654 via #12707)
- Add a new health check to `conda doctor` detecting altered packages in an environment by comparing expected and computed `sha256` checksums. (#12757)
- Add new `pre_commands` and `post_commands` plugin hooks allowing plugins to run code before and after `conda` subcommands. (#12712, #12758, #12864)
- Stop using `distutils` directly in favor of the vendored version in `setuptools` 60 and later or standard library equivalents. (#11136)
- Add a `CITATION.cff` file to the root of the repository to make it easier for users to cite conda. (#12781)
- Add optional `CondaSubcommand.configure_parser` allowing third-party plugins to hook into conda's argument parser. (#12814)
- Only display third-party subcommands in `conda --help` and not for every other subcommand. (#12814, #12740)
- Add a new config option, `no_plugins`, a `--no-plugins` command line flag, and a `CONDA_NO_PLUGINS` environment variable that disables external plugins for built-in conda commands. (#12748)
- Register plugins using their canonical/fully-qualified name instead of the easily spoofable entry point name. (#12869)
- De-duplicate plugin and legacy subcommands in `conda --help`. (#12893)
- Implement a 2-phase parser to better handle plugin disabling (via `--no-plugins`). (#12910)
- Refactor subcommand parsing to use a greedy parser since `argparse.REMAINDER` has known issues. (#12910)

Bug fixes

- Use `requests.exceptions.JSONDecodeError` for ensuring compatibility with different `json` implementations used by requests. This fixes a bug that caused only the first of multiple given source URLs to be tried. This also raises the minimum required requests version to 2.27.0. (#12683)
- Don't export `__osx` virtual package when `CONDA_OVERRIDE_OSX=""`. (#12715)
- Fix erroneous `conda deactivate` behavior of unsetting preexisting environment variables that are identical to those set during `conda activate`. (#12769)
- Correct third-party subcommands to receive *remaining* arguments instead of a blanket `sys.argv[2:]` which broke `conda_cli` testing. (#12814, #12910)

Deprecations

- Mark `conda.base.context.context.root_dir` as pending deprecation. Use `conda.base.context.context.root_prefix` instead. (#12701)
- Mark `conda.plugins.subcommands.doctor.cli.get_prefix` as pending deprecation. Use `conda.base.context.context.target_prefix` instead. (#12725)
- Mark `conda.models.leased_path_entry.LeasedPathEntry` as pending deprecation. (#12735)
- Mark `conda.models.enums.LeasedPathType` as pending deprecation. (#12735)
- Mark `conda.common.temporary_content_in_file` as pending deprecation. Use `tempfile` instead. (#12795)
- Mark `conda.cli.python_api` as pending deprecation. Use `conda.testing.conda_cli` fixture instead. (#12796)

Docs

- Document how to use the new `pre_commands` and `post_commands` plugin hooks. (#12712, #12758)
- Add docstrings to all public modules. (#12792)
- Auto-generate API docs using `sphinx-autoapi`. (#12798)
- Convert all manual redirects into config using `sphinx-reredirects`. (#12798)
- Revise the plugins index page to make it easier to understand how to create a conda plugin. (#12802)
- Add missing `conda env` CLI docs. (#12841)

Other

- Update `tests/cli/test_main_rename.py` to use latest fixtures. (#12517)
- Update `tests/test_activate.py` to test the new behavior. (#12769)
- Re-enable all `conda_env` tests and remove irrelevant tests. (#12813)
- Convert all `unittest`-style tests to `pytest`-style. (#12819)
- Convert `tests/test-recipes` into local noarch packages instead of relying on conda-test channel and local builds. (#12879)

Contributors

- @beeankha
- @conda-bot
- @dariocurr
- @jaimergp
- @jezdez
- @johanneskoester made their first contribution in https://github.com/conda/conda/pull/12683
- @jjhelmus
- @kalawac made their first contribution in https://github.com/conda/conda/pull/12738
- @kenodegard
- @schackartk made their first contribution in https://github.com/conda/conda/pull/12781
- @lesteve made their first contribution in https://github.com/conda/conda/pull/12715
- @ForgottenProgramme
- @marcoesters made their first contribution in https://github.com/conda/conda/pull/12863
- @mpotane made their first contribution in https://github.com/conda/conda/pull/11740
- @mattkram made their first contribution in https://github.com/conda/conda/pull/12730
- @morremeyer made their first contribution in https://github.com/conda/conda/pull/12871
- @mcg1969
- @travishathaway
- @pre-commit-ci[bot]

4.5.54 23.5.2 (2023-07-13)

Bug fixes

- Correct `native_path_to_unix` failure to handle no paths (e.g., an empty string or an empty iterable). (#12880)

Contributors

- @kenodegard

4.5.55 23.5.1 (2023-07-12)

Bug fixes

- Add (back) the `cygpath` fallback logic since `cygpath` is not always available on Windows. (#12873)

Contributors

- @kenodegard

4.5.56 23.5.0 (2023-05-17)

Enhancements

- Add `conda doctor` subcommand plugin. (#474)
- Add Python 3.11 support. (#12256)
- Add `conda list --reverse` to return a reversed list of installed packages. (#11954)
- Switch from `setup.py` to `pyproject.toml` and use Hatchling for our build system. (#12509)
- Optimize which Python modules get imported during `conda activate` calls to make it faster. (#12550)
- Add `conda_cli` fixture to replace `conda.testing.helpers.run_inprocess_conda_command` and `conda.testing.integration.run_command`. (#12592)
- Add `tmp_env` fixture to replace `conda.testing.integration.make_temp_env`. (#12592)
- Add `path_factory` fixture to replace custom prefix logic like `conda.testing.integration._get_temp_prefix` and `conda.testing.integration.make_temp_prefix`. (#12592)
- Refactor the way that the `Activator` classes are defined in `conda/activate.py`. (#12627)
- Warn about misconfiguration when signature verification is enabled. (#12639)

Bug fixes

- `conda clean` no longer fails if we failed to get the file stats. (#12536)
- Provide fallback version if `conda.deprecations.DeprecationHandler` receives a bad version. (#12541)
- Ensure the default value for `defaults` includes `msys2` when `context.subdir` is `win-*` on non-Windows platforms. (#12555)
- Avoid `TypeError` when non-string types are written to the index cache metadata. (#12562)
- `conda.core.package_cache_data.UrlsData.get_url` no longer fails when `package_path` has `.conda` extension. (#12516)
- Stop pre-converting paths to Unix style on Windows in `conda.sh`, so that they are prefix replaceable upon installation, which got broken by #12509. It also relies on `cygpath` at runtime, which all `msys2/cygwin` bash versions on Windows should have available. (#12627)

Deprecations

- Mark `conda_env.pip_util.get_pip_version` as pending deprecation. (#12492)
- Mark `conda_env.pip_util.PipPackage` as pending deprecation. (#12492)
- Mark `conda_env.pip_util.installed` as pending deprecation. (#12492)
- Mark `conda_env.pip_util._canonicalize_name` as pending deprecation. (#12492)
- Mark `conda_env.pip_util.add_pip_installed` as pending deprecation. (#12492)
- Mark `conda_env.env.load_from_directory` as pending deprecation. (#12492)
- Mark `python -m conda_env.cli.main` as pending deprecation. Use `conda env` instead. (#12492)
- Mark `python -m conda_env` as pending deprecation. Use `conda env` instead. (#12492)
- Mark `conda.auxlib.packaging` for deprecation in 24.3.0. (#12509)
- Rename index cache metadata file `.state.json` to `.info.json` to track draft CEP. (#12669)

- Mark `conda.testing.integration.get_conda_list_tuple` as pending deprecation. Use `conda.core.prefix_data.PrefixData().get()` instead. (#12676)
- Mark `conda.testing.encode_for_env_var` as pending deprecation. (#12677)
- Mark `conda.testing.integration.temp_chdir` as pending deprecation. Use `monkeypatch.chdir` instead. (#12678)

Docs

- Change the README example from IPython Notebook and NumPy to PyTorch. (#12579)
- Discuss options available to properly configure mirrored channels. (#12583, #12641)
- Add `flake8-docstrings` to `pre-commit`. (#12620)

Other

- Update retry language in flexible solve and `repodata` logs to be less ominous. (#12612)
- Improve `repodata` / `subdir_data` programming interface (#12521). Index cache metadata has changed to `.info.json` to better align with the draft CEP. Improve cache locking when using `jlap`. Improve `jlap` logging. (#12572)
- Format with `black` and replaced `pre-commit`'s `darker` hook with `black`. (#12554)
- Format with `isort` and add `pre-commit` `isort` hook. (#12554)
- Add functional tests around conda's content trust code. (#11805)
- Enable `flake8` checks that are now handled by `black`. (#12620)

Contributors

- @beeankha
- @chbrandt made their first contribution in https://github.com/conda/conda/pull/12419
- @chenghlee
- @conda-bot
- @dholth
- @THEdavehogue made their first contribution in https://github.com/conda/conda/pull/12612
- @HeavenEvolved made their first contribution in https://github.com/conda/conda/pull/12496
- @eltociear
- @jaimergp
- @jezdez
- @johnnynunez made their first contribution in https://github.com/conda/conda/pull/12256
- @kenodegard
- @ForgottenProgramme
- @pkmooreanaconda
- @tl-hbk made their first contribution in https://github.com/conda/conda/pull/12604
- @vic-ma made their first contribution in https://github.com/conda/conda/pull/12579
- @pre-commit-ci[bot]

- @sausagenoods made their first contribution in https://github.com/conda/conda/pull/12631

4.5.57 23.3.1 (2023-03-28)

Enhancements

- Fix and re-enable binstar tests. Replace custom property caching with `functools.cached_property`. (#12495)

Bug fixes

- Restore default argument for `SubdirData` method used by `conda-index`. (#12513)
- Include `conda.gateways.repodata.jlap` submodule in package. (#12545)

Other

- Add linux-s390x to multi-arch ci/dev container. (#12498)
- Expose a `MINIO_RELEASE` environment variable to provide a way to pin `minio` versions in CI setup scripts. (#12525)
- Add `jsonpatch` dependency to support `--experimental=jlap` feature. (#12528)

Contributors

- @conda-bot
- @dbast
- @dholth
- @jaimergp
- @kenodegard
- @ForgottenProgramme

4.5.58 23.3.0 (2023-03-14)

Enhancements

- Allow the use of environment variables for channel urls in `environment.yaml`. (#10018)
- Improved error message for `conda env create` if the environment file is missing. (#11883)
- Stop using `toolz.dicttoolz.merge` and `toolz.dicttoolz.merge_with`. (#12039)
- Add support for incremental `repodata.json` updates with `--experimental=jlap` on the command line or `experimental: ["jlap"]` in `.condarc` (#12090). Note: switching between "use jlap" and "don't use jlap" invalidates the cache.
- Added a new `conda.deprecations` module for easier & standardized deprecation. Includes decorators to mark functions, modules, classes, and arguments for deprecation and functions to mark modules, constants, and topics for deprecation. (#12125)
- Adds a new `channel_settings` configuration parameter that will be used to override arbitrary settings on per-channel basis. (#12239)
- Improve speed of `repodata.json` parsing by deferring creation of individual `PackageRecord` objects. (#8500)
- Refactor subcommand argument parsing to make it easier to understand. This calls the plugin before invoking the default argument parsing. (#12285)

- Handle I/O errors raised while retrieving channel notices. (#12312)
- Add support for the 64-bit RISC-V architecture on Linux. (#12319)
- Update vendored version of `py-cpuinfo` to 0.9.0. (#12319)
- Improved code coverage. (#12346, #12457, #12469)
- Add a note about `use_only_tar_bz2` being enabled on `PackagesNotFoundError` exceptions. (#12353)
- Added to conda CLI help that `conda remove -n <myenv> --all` can be used to delete environments. (#12378)
- Handle Python import errors gracefully when loading conda plugins from entrypoints. (#12460)

Bug fixes

- Fixed errors when renaming without an active environment. (#11915)
- Prevent double solve attempt if `PackagesNotFoundError` is raised. (#12201)
- Virtual packages follow `context.subdir` instead of `platform.system()` to enable cross-platform installations. (#12219)
- Don't export `__glibc` virtual package when `CONDA_OVERRIDE_GLIBC=""`. (#12267)
- Fix `arg_parse` pass-through for `--version` and `--help` in `conda.xsh`. (#12344)
- Filter out `None` path values from `pwd.getpwall()` on Unix systems, for users without home directories, when running as root. (#12063)
- Catch `ChunkedEncodingError` exceptions to prevent network error tracebacks hitting the output. (#12196 via #12487)
- Fix race conditions in `mkdir_p_sudo_safe`. (#12490)

Deprecations

- Drop `toolz.itertoolz.unique` in favor of custom `conda.common.iterators.unique` implementation. (#12252)
- Stop using `OrderedDict/odict` since `dict` preserves insert order since Python 3.7. (#12254)
- Mark `conda._vendor.boltons` for deprecation in 23.9.0. (#12272, #12482)
- Mark `conda_exe` in `context.py` and a topic in `print_package_info` `cli/main_info.py` for official deprecation. (#12398)
- Remove unused `chain`, `methodcaller`, `mkdtemp`, `StringIO` imports in `conda.common.compat`; apply other fixes from `ruff --fix .` in the test suite. (#12294)
- Remove unused optimization for searching packages based on `*[track_features=<feature name>]`. (#12314)
- Remove Notebook spec support from `conda env`; this was deprecated already and scheduled to be remove in version 4.5. (#12307)
- Mark `conda_exe` in `context.py` and a topic in `print_package_info` `cli/main_info.py` for official deprecation. (#12276)
- Marking `conda.utils.hashsum_file` as pending deprecation. Use `conda.gateways.disk.read.compute_sum` instead. (#12414)
- Marking `conda.utils.md5_file` as pending deprecation. Use `conda.gateways.disk.read.compute_sum(path, "md5")` instead. (#12414)

- Marking `conda.gateways.disk.read.compute_md5sum` as pending deprecation. Use `conda.gateways.disk.read.compute_sum(path, "md5")` instead. (#12414)
- Marking `conda.gateways.disk.read.compute_sha256sum` as pending deprecation. Use `conda.gateways.disk.read.compute_sum(path, "sha256")` instead. (#12414)
- Drop Python 3.7 support. (#12436)

Docs

- Added docs for `conda.deprecations`. (#12452)
- Updated some instances of "Anaconda Cloud" to be "Anaconda.org". (#12238)
- Added documentation on the specifications for `conda search` and `conda install`. (#12304)
- Mark `conda.utils.safe_open` for deprecation. Use builtin `open` instead. (#12415)

Other

- Update `<cache key>.json.state` `repodata.json` cache format; check `mtime` against cached `repodata.json`. (#12090)
- Skip redundant `tar --no-same-owner` when running as root on Linux, since newer `conda-package-handling` avoids setting ownership from the archive. (#12231)
- Add additional extensions to `conda.common.path` for future use. (#12261)
- Pass `--cov` in test runner scripts but not in `setup.cfg` defaults, for easier debugging. (#12268)
- Constrain conda-build to at least >=3.18.3, released 2019-06-20. (#12309)
- Improve `start.bat` Windows development script. (#12311)
- Provide conda-forge-based Docker images and fix the bundled `minio` binary. (#12335)
- Add support for conda-forge-based CI runtimes. On Linux (all architectures), unit & integration tests will use Python 3.10. On Windows, Python 3.8. On macOS, only the unit tests are run with conda-forge (*instead* of `defaults`!), using Python 3.9. (#12350, #12447 via #12448)
- Fix testing data issue where the `subdir` entry in some files was mismatched. (#12389)
- Initialize conda after installing test requirements during CI. (#12446)
- Speedup pre-commit by a factor of 15 by removing ignored hooks (`pylint/bandit`). This locally reduces the pre-commit runtime from ~43sec to 2.9sec and thus makes it possible to run pre-commit in a loop during development to constantly provide feedback and style the code. (#12466)

Contributors

- @AdrianFreundQC made their first contribution in https://github.com/conda/conda/pull/11883
- @sanzoghenzo made their first contribution in https://github.com/conda/conda/pull/12074
- @beeankha
- @conda-bot
- @dbast
- @dholth
- @FelisNivalis made their first contribution in https://github.com/conda/conda/pull/11915
- @gforsyth made their first contribution in https://github.com/conda/conda/pull/12344

- @eltociear made their first contribution in https://github.com/conda/conda/pull/12377
- @jaimergp
- @jezdez
- @jjhelmus
- @kannanjayachandran made their first contribution in https://github.com/conda/conda/pull/12363
- @kathatherine
- @kenodegard
- @ForgottenProgramme
- @ryanskeith made their first contribution in https://github.com/conda/conda/pull/12439
- @31Sanskrati made their first contribution in https://github.com/conda/conda/pull/12371
- @travishathaway
- @pre-commit-ci[bot]

4.5.59 23.1.0 (2023-01-17)

Bug fixes

- Detect CUDA driver version in subprocess. (#11667)
- Fixes the behavior of the `--no-user` flag in `conda init` so that a user's `.bashrc`, etc. remains unaltered, as expected. (#11949)
- Fix several more user facing `MatchSpec` crashes that were identified by fuzzing efforts. (#12099)
- Lock `sys.stdout` to avoid corrupted `--json` multithreaded download progress. (#12231)

Docs

- Optional Bash completion support has been removed starting in v4.4.0, and not just deprecated. (#11171)
- Documented optional `channel::package` syntax for specifying dependencies in `environment.yml` files. (#11890)

Other

- Refactor `repodata.json` fetching; update on-disk cache format. Based on work by @FFY00. (#11600)
- Environment variable overwriting WARNING is printed only if the env vars are different from those specified in the OS. (#12128)
- Added `conda-libmamba-solver` run constraint. (#12156)
- Updated `ruamel.yaml` version. (#12156)
- Added `tqdm` dependency. (#12191)
- Use `itertools.chain.from_iterable` instead of equivalent `tlz.concat`. (#12165)
- Use `toolz.unique` instead of vendored copy. (#12165)
- Use `itertools.islice` instead of `toolz.take`. (#12165)
- Update CI test workflow to only run test suite when code changes occur. (#12180)
- Added Python 3.10 canary builds. (#12184)

Contributors

- @beeankha
- @dholth
- @dariocurr made their first contribution in https://github.com/conda/conda/pull/12128
- @FFY00 made their first contribution in https://github.com/conda/conda/pull/11600
- @jezdez
- @jay-tau made their first contribution in https://github.com/conda/conda/pull/11738
- @kenodegard
- @pkmooreanaconda
- @sven6002 made their first contribution in https://github.com/conda/conda/pull/12162
- @ReveStobinson made their first contribution in https://github.com/conda/conda/pull/12213
- @travishathaway
- @XuehaiPan made their first contribution in https://github.com/conda/conda/pull/11667
- @xylar made their first contribution in https://github.com/conda/conda/pull/11949
- @pre-commit-ci[bot]

4.5.60 22.11.1 (2022-12-06)

Bug fixes

- Restore default virtual package specs as in 22.9.0 (#12148)
  - re-add `__unix`/`__win` packages
  - restore `__archspec` version/build string composition

Other

- Skip test suite for non-code changes. (#12141)

Contributors

- @LtDan33
- @jezdez
- @kenodegard
- @mbargull
- @travishathaway

4.5.61 22.11.0 (2022-11-23)

Enhancements

- Add LD_PRELOAD to env variable list. (#10665)
- Improve CLI warning about updating conda. (#11300)
- Conda's initialize block in the user's profiles will check whether the conda executable exists before calling the conda hook. (#11374)

- Switch to `tqdm` as a real dependency. (#12005)
- Add a new plugin mechanism. (#11435)
- Add an informative message if explicit install fails due to requested packages not being in the cache. (#11591)
- Download and extract packages in parallel. Greatly speeds up package downloads when latency is high. Controlled by the new `fetch_threads` config parameter, defaulting to 5 parallel downloads. Thanks @shuges-uk for reporting. (#11841)
- Add a new plugin hook for virtual packages and convert existing code for virtual packages (from index.py) into plugins. (#11854)
- Require `ruamel.yaml`. (#11868, #11837)
- Stop using `toolz.accumulate`. (#12020)
- Stop using `toolz.groupby`. (#11913)
- Remove vendored `six` package. (#11979)
- Add the ability to extend the solver backends with the `conda_solvers` plugin hook. (#11993)
- Stop using `toolz.functoolz.excepts`. (#12016)
- Stop using `toolz.itertoolz.concatv`. (#12020)
- Also try UTF16 and UTF32 encodings when replacing the prefix. (#9946)

Bug fixes

- `conda env update` would ask for user input and hang when working with pip installs of git repos and the repo was previously checked out. Tell pip not to ask for user input for that case. (#11818)
- Fix for `conda update` and `conda install` issues related to channel notices. (#11852)
- Signature verification printed `None` when disabled, changes default `metadata_signature_status` to an empty string instead. (#11944)
- Fix importlib warnings when importing conda.cli.python_api on python=3.10. (#11975)
- Several user facing MatchSpec crashes were identified by fuzzing efforts. (#11999)
- Apply minimal fixes to deal with these (and similar) crashes. (#12014)
- Prevent `conda` from using `/bin/sh` + `exec` trick for its own entry point, which drops `$PS1` on some shells (#11885, #11893 via #12043).
- Handle `CTRL+C` during package downloading more gracefully. (#12056)
- Prefer the outer name when a MatchSpec specifies a package's name twice package-name[name=package-name] (#12062)

Deprecations

- Add a pending deprecation warning for when importing `tqdm` from `conda._vendor`. (#12005)
- Drop `ruamel_yaml` and `ruamel_yaml_conda` in favor of `ruamel.yaml`. (#11837)
- `context.experimental_solver` is now marked for pending deprecation. It is replaced by `context.solver`. The same applies to the `--experimental-solver` flag, the `CONDA_EXPERIMENTAL_SOLVER` environment variable, and the `ExperimentalSolverChoice` enum, which will be replaced by `--solver`, `EXPERIMENTAL_SOLVER` and `SolverChoice`, respectively. (#11889)
- Mark `context.conda_private` as pending deprecation. (#12101)

Docs

- Add corresponding documentation for the new plugins mechanism. (#11435)
- Update conda cheatsheet for the 4.14.0 release. The cheatsheet now includes an example for `conda rename`. (#11768)
- Document conda-build package format v2, also known as the `.conda`-format. (#11881)
- Remove `allow_other_channels` config option from documentation, as the option no longer exists. (#11866)
- Fix bad URL to "Introduction to conda for Data Scientists" course in conda docs. (#9782)

Other

- Add a comment to the code that explains why .bashrc is modified on Linux and .bash_profile is modified on Windows/macOS when executing `conda init`. (#11849)
- Add `--mach` and `--arch` options to `dev/start`. (#11851)
- Remove encoding pragma in file headers, as it's not needed in Python 3 anymore. (#11880)
- Refactor `conda init SHELLS` as argparse choices. (#11897)
- Drop pragma fixes from pre-commit checks. (#11909)
- Add pyupgrade to pre-commit checks. This change affects many files. Existing pull requests may need to be updated, rebased, or merged to address conflicts. (#11909)
- Add aarch64 and ppc64le as additional CI platforms for smoke testing. (#11911)
- Serve package files needed for testing using local server. (#12024)
- Update canary builds to guarantee builds for the commits that trigger workflow. (#12040)

Contributors

- @arq0017 made their first contribution in https://github.com/conda/conda/pull/11810
- @beeankha
- @conda-bot
- @dbast
- @dholth
- @erykoff
- @considerRatio made their first contribution in https://github.com/conda/conda/pull/12028
- @jaimergp
- @jezdez
- @kathatherine
- @kenodegard
- @ForgottenProgramme made their first contribution in https://github.com/conda/conda/pull/11926
- @hmaarrfk made their first contribution in https://github.com/conda/conda/pull/9946
- @NikhilRaverkar made their first contribution in https://github.com/conda/conda/pull/11842
- @pavelzw made their first contribution in https://github.com/conda/conda/pull/11849
- @pkmooreanaconda made their first contribution in https://github.com/conda/conda/pull/12014

- @fragmede made their first contribution in https://github.com/conda/conda/pull/11818
- @SatyamVyas04 made their first contribution in https://github.com/conda/conda/pull/11870
- @timhoffm
- @travishathaway
- @dependabot made their first contribution in https://github.com/conda/conda/pull/11965
- @pre-commit-ci[bot]
- @wulmer

4.5.62 22.9.0 (2022-09-14)

Special announcement

If you have been following the conda project previously, you will notice a change in our version number for this release. We have officially switched to the CalVer versioning system as agreed upon in CEP 8 (Conda Enhancement Proposal).

Please read that CEP for more information, but here is a quick synopsis. We hope that this versioning system and our release schedule will help make our releases more predictable and transparent to the community going forward. We are now committed to making at least one release every two months, but keep in mind that we can (and most likely will) be making minor version releases within this window.

Enhancements

- Replace vendored toolz with toolz dependency. (#11589, #11700)
- Update bundled Python launchers for Windows (`conda/shell/cli-*.exe`) to match the ones found in conda-build. (#11676)
- Add `win-arm64` as a known platform (subdir). (#11778)

Bug fixes

- Remove extra prefix injection related to the shell interface breaking `conda run`. (#11666)
- Better support for shebang instructions in prefixes with spaces. (#11676)
- Fix `noarch` entry points in Unicode-containing prefixes on Windows. (#11694)
- Ensure that exceptions that are raised show up properly instead of resulting in a blank `[y/N]` prompt. (#11746)

Deprecations

- Mark `conda._vendor.toolz` as pending deprecation. (#11704)
- Removes vendored version of urllib3. (#11705)

Docs

- Added conda capitalization standards to CONTRIBUTING file. (#11712)

Other

- Add arm64 support to development script `. ./dev/start`. (#11752)
- Update canary-release version to resolve canary build issue. (#11761)
- Renamed canary recipe from `conda.recipe` to `recipe`. (#11774)

Contributors

- @beeankha
- @chenghlee
- @conda-bot
- @dholth
- @isuruf
- @jaimergp
- @jezdez
- @razzlestorm made their first contribution in https://github.com/conda/conda/pull/11736
- @jakirkham
- @kathatherine
- @kenodegard
- @scdub made their first contribution in https://github.com/conda/conda/pull/11816
- @travishathaway
- @pre-commit-ci[bot]

4.5.63 4.14.0 (2022-08-02)

Enhancements

- Only star activated environment in `conda info --envs`/`conda env list`. (#10764)
- Adds new sub-command, `conda notices`, for retrieving channel notices. (#11462)
- Notices will be intermittently shown after running, `install`, `create`, `update`, `env create` or `env update`. New notices will only be shown once. (#11462)
- Implementation of a new `rename` subcommand. (#11496)
- Split `SSLError` from `HTTPError` to help resolve HTTP 000 errors. (#11564)
- Include the invalid package name in the error message. (#11601)
- Bump requests version (`>=2.20.1`) and drop monkeypatching. (#11643)
- Rename `whitelist_channels` to `allowlist_channels`. (#11647)
- Always mention channel when notifying about a new conda update. (#11671)

Bug fixes

- Correct a misleading `conda --help` error message. (#11625)
- Fix support for CUDA version detection on WSL2. (#11626)
- Fixed the bug when providing empty environment.yml to `conda env create` command. (#11556, #11630)
- Fix MD5 hash generation for FIPS-enabled systems. (#11658)
- Fixed `TypeError` encountered when logging is set to DEBUG and the package's JSON cannot be read. (#11679)

Deprecations

- `conda.cli.common.ensure_name_or_prefix` is pending deprecation in a future release. (#11490)
- Mark `conda.lock` as pending deprecation. (#11571)
- Remove lgtm.com config. (#11572)
- Remove Python 2.7 `conda.common.url.is_ipv6_address_win_py27` implementation. (#11573)
- Remove redundant `conda.resolve.dashlist` definition. (#11578)
- Mark `conda_env.cli.common.get_prefix` and `conda.base.context.get_prefix` as pending deprecation in favor of `conda.base.context.determine_target_prefix`. (#11594)
- Mark `conda_env.cli.common.stdout_json` as pending deprecation in favor of `conda.cli.common.stdout_json`. (#11595)
- Mark `conda_env.cli.common.find_prefix_name` as pending deprecation. (#11596)
- Mark `conda.auxlib.decorators.memoize` as pending deprecation in favor of `functools.lru_cache`. (#11597)
- Mark `conda.exports.memoized` as pending deprecation in favor of `functools.lru_cache`. (#11597)
- Mark `conda.exports.handle_proxy_407` as pending deprecation. (#11597)
- Refactor `conda.activate._Activator.get_export_unset_vars` to use `**kwargs` instead of `OrderedDict`. (#11649)
- Mark `conda.another_to_unicode` as pending deprecation. (#11678)

Docs

- Corresponding documentation of `notices` subcommand. (#11462)
- Corresponding documentation of `rename` subcommand. (#11496)
- Correct docs URL to https://docs.conda.io. (#11508)
- Updated the list of environment variables that can now expand in the Use Condarc section. (#11514)
- Include notice that the "All Users" installation option in the Anaconda Installer is no longer available due to security concerns. (#11528)
- Update `conda-zsh-completeion` link. (#11541)
- Missing `pip` as a dependency when including a pip-installed dependency. (#11543)
- Convert `README.rst` to `README.md`. (#11544)
- Updated docs and CLI help to include information on `conda init` arguments. (#11574)
- Added docs for writing integration tests. (#11604)
- Updated `conda env create` CLI documentation description and examples to be more helpful. (#11611)

Other

- Display tests summary in CI. (#11558)
- Update `Dockerfile` and `ci-images.yml` flow to build multi arch images. (#11560)
- Rename master branch to main. (#11570)

Contributors

- @drewja made their first contribution in #11614
- @beeankha
- @topherocity made their first contribution in #11658
- @conda-bot
- @dandv made their first contribution in #11636
- @dbast
- @dholth
- @deepyaman made their first contribution in #11598
- @dogukanteber made their first contribution in #11556/#11630
- @jaimergp
- @kathatherine
- @kenodegard
- @nps1ngh made their first contribution in #10764
- @pseudoyim made their first contribution in #11528
- @SamStudio8 made their first contribution in #11679
- @SamuelWN made their first contribution in #11543
- @spencermathews made their first contribution in #11508
- @timgates42
- @timhoffm made their first contribution in #11601
- @travishathaway
- @esc
- @pre-commit-ci[bot]

4.5.64 4.13.0 (2022-05-19)

Enhancements

- Introducing `conda clean --logfiles` to remove logfiles generated by conda-libmamba-solver. (#11387)
- Add the solver name and version to the user-agent. (#11415, #11455)
- Attempt parsing HTTP errors as a JSON and extract error details. If present, prefer these details instead of those hard-coded. (#11440)

Bug fixes

- Fix inconsistencies with `conda clean --dryrun` (#11385)
- Standardize tarball & package finding in `conda clean` (#11386, #11391)
- Fix `escape_channel_url` logic on Windows (#11416)
- Use 'Accept' header, not 'Content-Type' in GET header (#11446)
- Allow extended user-agent collection to fail but log the exception (#11455)

Deprecations

- Removing deprecated `conda.cli.activate`. Originally deprecated in conda 4.6.0 in May 2018. (#11309)
- Removing deprecated `conda.compat`. Originally deprecated in conda 4.6.0 in May 2018. (#11322)
- Removing deprecated `conda.install`. Originally deprecated in conda 4.6.0 in May 2018. (#11323)
- Removing deprecated `conda.cli.main_help`. Originally deprecated in conda 4.6.0 in May 2018. (#11325)
- Removed unused `conda.auxlib.configuration`. (#11349)
- Removed unused `conda.auxlib.crypt`. (#11349)
- Removed unused `conda.auxlib.deprecation`. (#11349)
- Removed unused `conda.auxlib.factory`. (#11349)
- Removed minimally used `conda.auxlib.path`. (#11349)
- Removed `conda.exports.CrossPlatformStLink`, a Windows Python <3.2 fix for `os.lstat.st_nlink`. (#11351)
- Remove Python 2.7 and other legacy code (#11364)
- `conda run --live-stream` aliases `conda run --no-capture-output`. (#11422)
- Removes unused exceptions. (#11424)
- Combines conda_env.exceptions with conda.exceptions. (#11425)
- Drop Python 3.6 support. (#11453)
- Remove outdated test `test_init_dev_and_NoBaseEnvironmentError` (#11469)

Docs

- Initial implementation of deep dive docs (#11059)
- Correction of RegisterPython description in Windows Installer arguments. (#11312)
- Added autodoc documentation for `conda compare`. (#11336)
- Remove duplicated instruction in manage-python.rst (#11381)
- Updated conda cheatsheet. (#11443)
- Fix typos throughout the codebase (#11448)
- Fix `conda activate` example (#11471)
- Updated `conda` 4.12 cheatsheet with new anaconda distribution version (#11479)

Other

- Add Python 3.10 as a test target. (#10992)
- Replace custom `conda._vendor` with vendoring (#11290)
- Replace `conda.auxlib.collection.frozendict` with vendored `frozendict` (#11398)
- Reorganize and new tests for `conda.cli.main_clean` (#11360)
- Removing vendored usage of urllib3 and instead implementing our own wrapper around std. lib urllib (#11373)
- Bump vendored `py-cpuinfo` version 4.0.0 → 8.0.0. (#11393)
- Add informational Codecov status checks (#11400)

Contributors

- @beeankha made their first contribution in #11469
- @ChrisPanopoulos made their first contribution in #11312
- @conda-bot
- @dholth
- @jaimergp
- @jezdez
- @kathatherine made their first contribution in #11443
- @kenodegard
- @kianmeng made their first contribution in #11448
- @simon9500 made their first contribution in #11381
- @thomasrockhu-codecov made their first contribution in #11400
- @travishathaway made their first contribution in #11373
- @pre-commit-ci[bot]

4.5.65 4.12.0 (2022-03-08)

Enhancements

- Add support for libmamba integrations. (#11193)

  This is a new **experimental and opt-in** feature that allows use of the new conda-libmamba-solver for an improved user experience, based on the libmamba community project - the library version of the mamba package manager.

  Please follow these steps to try out the new libmamba solver integration:

  1. Make sure you have conda-libmamba-solver installed in your conda base environment.
  2. Try out the solver using the `--experimental-solver=libmamba` command line option.

     E.g. with a dry-run to install the `scipy` package:

     ```
     conda create -n demo scipy --dry-run --experimental-solver=libmamba
     ```

     Or install in an activated conda environment:

     ```
     conda activate my-environment
     conda install scipy --experimental-solver=libmamba
     ```

- Make sure that `conda env update -f` sets env vars from the referenced yaml file. (#10652)
- Improve command line argument quoting, especially for `conda run`. (#11189)
- Allow `conda run` to work in read-only environments. (#11215)
- Add support for prelink_message. (#11123)
- Added `conda.CONDA_SOURCE_ROOT`. (#11182)

Bug fixes

- Refactored `conda.utils.ensure_comspec_set` into `conda.utils.get_comspec`. (#11168)
- Refactored `conda.cli.common.is_valid_prefix` into `conda.cli.common.validate_prefix`. (#11172)
- Instantiate separate S3 session for thread-safety. (#11038)
- Change overly verbose info log to debug. (#11260)
- Remove five.py and update metaclass definitions. (#11267)
- Remove unnecessary conditional in setup.py (#11013)

Docs

- Clarify on AIE messaging in download.rst. (#11221)
- Fix conda environment variable echo, update example versions. (#11237)
- Fixed link in docs. (#11268)
- Update profile examples. (#11278)
- Fix typos. (#11070)
- Document conda run command. (#11299)

Other

- Added macOS to continuous integration. (#10875)
- Added ability to build per-pullrequest review builds. (#11135)
- Improved subprocess handling on Windows. (#11179)
- Add `CONDA_SOURCE_ROOT` env var. (#11182)
- Automatically check copyright/license disclaimer & encoding pragma. (#11183)
- Development environment per Python version. (#11233)
- Add concurrency group to cancel GHA runs on repeated pushes to branch/PR. (#11258)
- Only run GHAs on non-forks. (#11265)

Contributors

- @opoplawski
- @FaustinCarter
- @jaimergp
- @rhoule-anaconda
- @jezdez
- @hajapy
- @erykoff
- @uwuvalon
- @kenodegard
- @manics
- @NaincyKumariKnoldus

- @autotmp
- @yuvipanda
- @astrojuanlu
- @marcelotrevisani

4.5.66 4.11.0 (2021-11-22)

Enhancements

- Allow channel_alias to interpolate environment variables.
- Support running conda with PyPy on Windows.
- Add ability to add, append and prepend to sequence values when using the conda config subcommand.
- Support Python 3.10 in version parser.
- Add `XDG_CONFIG_HOME` to the conda search path following the XDG Base Directory Specification (XDGBDS).

Bug fixes

- Fix the PowerShell activator to not show an error when unsetting environment variables.
- Remove superfluous `eval` statements in fish shell integration.
- Indent the conda fish integration file using fish_indent.
- Fix handling of environment variables containing equal signs (=).
- Handle permission errors when listing all known prefixes.
- Catch Unicode decoding errors when parsing conda-meta files.
- Fix handling write errors when trying to create package cache or env directories.

Docs

- Update path of conda repo in RHEL based systems to `/etc/yum.repos.d/conda.repo`.
- Fix the advanced pip example to stop using the now invalid `file:` prefix.
- Minor docs cleanup and adding Code of Conduct.
- Add auto-built architecture documentation for conda based on the C4 Model. See the conda documentation for more information.
- Expand the contributing documentation with a section about static code analysis and code linting.
- Add developer guide section to the documentation, including a conda architecture overview.
- Stop referring to updating anaconda when `conda update` fails with an error.

Other

- Build Docker images periodically on GitHub Actions for the continuous integration testing on Linux, storing them on GitHub Packages's registry for reduced latency and cost when using Docker Hub.
- Simplify the Linux GitHub actions workflows by combining used shell scripts.
- Add periodic GitHub Actions workflow to review old issues in the conda issue tracker and mark them as stale if no feedback is provided in a sensible amount of time, eventually closing them.

- Add periodic GitHub Actions workflow to lock the comment threads of old issues and pull requests in the conda GitHub repository to surface regressions with new issues instead.
- Refactor test suite to use more GitHub Actions runners in parallel, reducing total run time by 50%.
- Switched the issue tracker to use forms with additional questions for bug reporters to help in ticket triage.
- Add and automatically run pre-commit as part of the CI system to improve the code quality continuously and raise issues in contributed patches early on.

  The used code linters are: flake8, pylint and bandit.

  The Python code formatter black is used as well but is only enforced on changed code in a commit and not to the whole code base at once.
- Automatically build the conda package upon the successful merge into the master branch and upload it to the conda-canary channel on anaconda.org.

  To try conda out simply run:

```
conda install -c conda-canary/label/dev conda
```

- Automate adding new issues to public GitHub project board to facilitate issue triage.
- Update GitHub issue and pull request labels to be more consistent.
- Start using rever for release management.
- (preview) Enable one-click gitpod and GitHub Codespaces setup for Linux development.

Contributors

- Benjamin Bertrand
- Chawye Hsu
- Cheng H. Lee
- Dan Meador
- Daniel Bast
- Daniel Holth
- Gregor Kržmanc
- Hsin-Hsiang Peng
- Ilan Cosman
- Isuru Fernando
- Jaime Rodríguez-Guerra
- Jan-Benedikt Jagusch
- Jannis Leidel
- John Flavin
- Jonas Haag
- Ken Odegard
- Kfir Zvi
- Mervin Fansler

- bfis
- mkincaid
- pre-commit CI

4.5.67 4.10.3 (2021-06-29)

Bug fixes

- Reverts "Don't create an unused S3 client at import time (#10516)" in 4.10.2 that introduced a regression for users using S3 based channels. (#10756)

4.5.68 4.10.2 (2021-06-25)

Enhancements

- Add `--dry-run` option to `conda env create` (#10635)
- Print warning about pip-installed dependencies only once (#10638)
- Explicit install now respects `--download-only` flag (#10688)
- Bump vendored tqdm version (#10721)

Bug fixes

- Fix `changeps1` handling for PowerShell (#10624)
- Handle unbound `$PS1` so sh activation does not fail with `set -u` (#10701)
- Fix sh activation so `$PATH` is properly restored on errors (#10631)
- Fix `-c` option handling so defaults channel is not always re-added (#10735)
- Fix artifact verification-related warnings and errors (#10627, #10677)
- Fix log level used in `conda/core/prefix_data.py` (#9998)
- Fix log level used when fetching artifact verification metadata (#10621)
- Don't create an unused S3 client at import time (#10516)
- Don't load `binstar_client` until needed (#10692)
- Reflect dropping of older Python versions in `setup.py` (#10642)

Docs

- Merge release notes and changelog to reduce maintenance burden (#10745)
- Add mentions to PyPy, Anaconda terms of service (#10329, #10712)
- Update Python versions in examples (#10329, #10744)
- Update install macOS instructions (#10728)

Contributors

- @AlbertDeFusco
- @awwad
- @casperdcl
- @cgranade

- @chenghlee
- @ColemanTom
- @dan-hook
- @dbast
- @ericpre
- @HedgehogCode
- @jamesp
- @jezdez
- @johnhany97
- @lightmare
- @mattip
- @maxerbubba
- @mrakitin
- @stinos
- @thermokarst

4.5.69 4.10.1 (2021-04-12)

Bug fixes

- Fix version detection for `__linux` virtual package (#10599)
- Fix import from `conda_content_trust` (#10589)
- Fix how URL for verification metadata files are constructed (#10617)
- Partially fix profile `$PATH` setup on MSYS2 (#10459)
- Remove `.empty` directory even when `rsync` is not installed (#10331)

Contributors

- @awwad
- @chenghlee
- @codepage949
- @niklasholm

4.5.70 4.10.0 (2021-03-30)

NOTE: This release formally drops support for Python 2.7 and Python < 3.6.

Enhancements

- Add pilot support for metadata signatures and verification (#10578)
- Add `__linux` virtual package (#10552, #10561)
- Support nested keys when using `conda config --get` (#10447, #10572)
- Support installing default packages when using `conda env create` (#10530)

- Support HTTP sources for `conda env update -f` (#10536)
- Make macOS code signing operations less verbose (#10372)

Bug fixes

- Fix `conda search` crashing on Python 3.9 (#10542)
- Allow `{channel}::pip` to satisfy pip requirements (#10550)
- Support `{host}:{port}` specifications in environment YAML files (#10417)
- Fall back to system `.condarc` if user `.condarc` is absent (#10479)
- Try UTF-16 if UTF-8 fails when reading environment YAML files (#10356)
- Properly parse Python version >= 3.10 (#10478)
- Fix zsh initialization when `$ZDOTDIR` is defined (#10413)
- Fix path handling for csh (#10410)
- Fix `setup.py` requirement for vendored `ruamel_yaml_conda` (#10441)
- Fix errors when pickling vendored `auxlib` objects (#10386)

Docs

- Document the `__unix` and `__windows` virtual packages (#10511)
- Update list of supported and default versions of Python (#10531)
- Favor using `pip` instead of `setup.py` when setting up CI (#10308)

Miscellaneous

- CI: drop Python 2.7 and add Python 3.9 (#10548)

Contributors

- @awwad
- @BastianZim
- @beenje
- @bgobbi
- @blubs
- @chenghlee
- @cjmartian
- @ericpre
- @erykoff
- @felker
- @giladmaya
- @jamesmyatt
- @mingwandroid
- @opoplawski

- @saadparwaiz1
- @saucoide

4.5.71 4.9.2 (2020-11-10)

Enhancements

- Use vendored `tqdm` in `conda.resolve` for better consistency (#10337)

Bug fixes

- Revert to previous naming scheme for repodata cache files when `use_only_tar_bz2` config option is false (#10350)

Docs

- Fix missing release notes (#10342)
- Fix permission errors when configuring deb repositories (#10347)

Contributors

- @chenghlee
- @csoja
- @dylanmorroll
- @sscherfke

4.5.72 4.9.1 (2020-10-26)

Enhancements

- Respect PEP 440 ~= "compatible release" clause (#10313)

Bug fixes

- Remove `preload_openssl` for Win32 (#10298)
- Add `if exist` to Windows registry hook (#10305)

Contributors

- @mingwandroid

4.5.73 4.9.0 (2020-10-19)

Enhancements

- Add `osx-arm64` as a recognized platform (#10128, #10134, #10137)
- Resign files modified during installation on ARM64 macOS (#10260)
- Add `__archspec` virtual package to identify CPU microarchitecture (#9930)
- Add `__unix` and `__win` virtual packages (#10214)
- Add `--no-capture--output` option to `conda run` (#9646)
- Add `--live-stream` option to `conda run` (#10270)

- Export and import environment variables set using `conda env config` (#10169)
- Cache repodata from `file://` channels (#9730)
- Do not relink already-installed packages (#10208)
- Speed up JSON formatting in logz module (#10189)

Bug fixes:

- Stop `env remove --dry-run` from actually removing environments (#10261)
- Virtual package requirements are now considered by the solver (#10057)
- Fix cached filename processing when using only tar.bz2 (#10193)
- Stop showing solver hints about CUDA when it is not a dependency (#10275)
- Ignore virtual packages when checking environment consistency (#10196)
- Fix `config --json` output errors in certain circumstances (#10194)
- More consistent error handling by `conda shell` (#10238)
- Bump vendored version of tqdm to fix various threading and I/O bugs (#10266)

Docs

- Correctly state default `/AddToPath` option in Windows installer (#10179)
- Fix typos in `--repodata-fn` help text (#10279)

Miscellaneous

- Update CI infrastructure to use GitHub Actions (#10176, #10186, #10234)
- Update README badge to show GitHub Actions status (#10254)

Contributors

- @AlbertDeFusco
- @angloyna
- @bbodenmiller
- @casperdcl
- @chenghlee
- @chrisburr
- @cjmartian
- @dhirschfeld
- @ericpre
- @gabrielcnr
- @InfiniteChai
- @isuruf
- @jjhelmus
- @LorcanHamill

- @maresb
- @mingwandroid
- @mlline00
- @xhochy
- @ydmytryk

4.5.74 4.8.5 (2020-09-14)

Enhancements

- Add `osx-arm64` as a recognized platform (#10128, #10134)

Contributors

- @isuruf
- @jjhelmus

4.5.75 4.8.4 (2020-08-06)

Enhancements

- Add `linux-ppc64` as a recognized platform (#9797, #9877)
- Add `linux-s390x` as a recognized platform (#9933, #10051)
- Add spinner to pip installer (#10032)
- Add support for running conda in PyPy (#9764)
- Support creating conda environments using remote specification files (#9835)
- Allow request retries on various HTTP errors (#9919)
- Add `compare` command for environments against a specification file (#10022)
- Add (preliminary) support for JSON-format activation (#8727)
- Properly handle the `CURL_CA_BUNDLE` environment variable (#10078)
- More uniformly handle `$CONDA_PREFIX` when exporting environments (#10092)
- Enable trailing _ to anchor OpenSSL-like versions (#9859)
- Replace `listdir` and `glob` with `scandir` (#9889)
- Ignore virtual packages when searching for constrained packages (#10117)
- Add virtual packages to be considered in the solver (#10057)

Bug fixes:

- Prevent `remove --all` from deleting non-environment directories (#10086)
- Prevent `create --dry-run --yes` from deleting existing environments (#10090)
- Remove extra newline from environment export file (#9649)
- Print help on incomplete `conda env config` command rather than crashing (#9660)
- Correctly set exit code/errorlevel when `conda run` exits (#9665)
- Send “inconsistent environment” warnings to stderr to avoid breaking JSON output (#9738)

- Fix output formatting from post-link scripts (#9841)
- Fix URL parsing for channel subdirs (#9844)
- Fix `conda env export -f` sometimes producing empty output files (#9909)
- Fix handling of Python releases with two-digit minor versions (#9999)
- Do not use gid to determine if user is an admin on *nix platforms (#10002)
- Suppress spurious xonsh activation warnings (#10005)
- Fix crash when running `conda update --all` on a nonexistent environment (#10028)
- Fix collections import for Python 3.8 (#10093)
- Fix regex-related deprecation warnings (#10093, #10096)
- Fix logic error when running under Python 2.7 on 64-bit platforms (#10108)
- Fix Python 3.8 leaked semaphore issue (#10115)

Docs

- Fix formatting and typos (#9623, #9689, #9898, #10042)
- Correct location for yum repository configuration files (#9988)
- Clarify usage for the `--channel` option (#10054)
- Clarify Python is not installed by default into new environments (#10089)

Miscellaneous

- Fixes to tests and CI pipelines (#9842, #9863, #9938, #9960, #10010)
- Remove conda-forge dependencies for developing conda (#9857, #9871)
- Audit YAML usage for `safe_load` vs `round_trip_load` (#9902)

Contributors

- @alanhdu
- @angloyna
- @Anthchirp
- @Arrowbox
- @bbodenmiller
- @beenje
- @bernardoduarte
- @birdsarah
- @bnemanich
- @chenghlee
- @ChihweiLHBird
- @cjmartian
- @ericpre
- @error404-beep

- @esc
- @hartb
- @hugobuddel
- @isuruf
- @jjhelmus
- @kalefranz
- @mingwandroid
- @mlline00
- @mparry
- @mrocklin
- @necaris
- @pdnm
- @pradghos
- @ravigumm
- @Reissner
- @scopatz
- @sidhant007
- @songmeixu
- @speleo3
- @tomsaleeba
- @WinstonPais

4.5.76 4.8.3 (2020-03-13)

Docs

- Add release notes for 4.8.2 to docs (#9632)
- Fix typos in docs (#9637, #9643)
- Grammatical and formatting changes (#9647)

Bug fixes:

- Account for channel is specs (#9748)

Contributors

- @bernardoduarte
- @forrestwaters
- @jjhelmus
- @msarahan
- @rrigdon

- @timgates42

4.5.77 4.8.2 (2020-01-24)

Enhancements

- Solver messaging improvements (#9560)

Docs

- Added precedence and conflict info (#9565)
- Added how to set env variables with config API (#9536)
- Updated user guide, deleted Overview, minor clean up (#9581)
- Add code of conduct (#9601, #9602, #9603, #9603, #9604 #9605)

Bug fixes:

- change fish prompt only if changeps1 is true (#7000)
- make frozendict JSON serializable (#9539)
- Conda env create empty dir (#9543)

Contributors

- @msarahan
- @jjhelmus
- @rrigdon
- @soapy1
- @teake
- @csoja
- @kfranz

4.5.78 4.8.1 (2019-12-19)

Enhancements

- improve performance for conda run by avoiding Popen.communicate (#9381)
- Put conda keyring in /usr/share/keyrings on Debian (#9424)
- refactor common.logic to fix some bugs and prepare for better modularity (#9427)
- Support nested configuration (#9449)
- Support Object configuration parameters (#9465)
- Use freeze_installed to speed up conda env update (#9511)
- add networking args to conda env create (#9525)

Docs

- fix string concatenation running words together regarding CONDA_EXE (#9411)
- Fix typo (“list” -> “info”) (#9433)
- typo in condarc key envs_dirs (#9478)
- Clarify channel priority and package sorting (#9492)
- improve description of DLL loading verification and activating environments (#9453)
- Installing with specific build number (#9534)

Bug fixes:

- Fix calling python api run_command with list and string arguments (#9331)
- revert init bash completion (#9421)
- set tmp to shortened path that excludes spaces (#9409)
- avoid function redefinition upon resourcing conda.fish (#9444)
- propagate pip error level when creating envs with conda env (#9460)
- fix incorrect chown call (#9464)
- Add subdir to PackageRecord dist_str (#9418)
- Fix running conda activate in multiple processes on windows (#9477)
- Don’t check in pkgs for trash (#9472)
- remove setuptools from run_constrained in recipe (#9485)
- Fix `__conda_activate` function to correctly return exit code (#9532)
- fix overly greedy capture done by subprocess for conda run (#9537)

Contributors

- @AntoinePrv
- @brettcannon
- @bwildenhain
- @cjmartian
- @felker
- @forrestwaters
- @gilescope
- @isuruf
- @jeremyjliu
- @jjhelmus
- @jhultman
- @marcuscaisey
- @mbargull
- @mingwandroid

- @msarahan
- @okhoma
- @osamoylenko
- @rrigdon
- @ruleofthehuns
- @soapy1
- @tartansandal

4.5.79 4.8.0 (2019-11-04)

Enhancements

- retry downloads if they fail, controlled by `remote_max_retries` and `remote_backoff_factor` configuration values (#9318)
- redact authentication information in some URLs (#9341)
- add osx version virtual package , `__osx` (#9349)
- add glibc virtual package, `__glibc` (#9358)

Docs

- removeed references to MD5s from docs (#9247)
- Add docs on `CONDA_DLL_SEARCH_MODIFICATION_ENABLED` (#9286)
- document threads, spec history and configuration (#9327)
- more documentation on channels (#9335)
- document the .condarc search order (#9369)
- various minor documentation fixes (#9238, #9248, #9267, #9334, #9351, #9372, #9378, #9388, #9391, #9393)

Bug fixes

- fix issues with xonsh activation on Windows (#8246)
- remove unsupported --lock argument from conda clean (#8310)
- do not add `sys_prefix_path` to failed activation or deactivation (#9282)
- fix csh setenv command (#9284)
- do not memorize `PackageRecord.combined_depends` (#9289)
- use `CONDA_INTERNAL_OLDPATH` rather than `OLDPATH` in activation script (#9303)
- fixes xonsh activation and tab completion (#9305)
- fix what channels are queried when context.offline is True (#9385)

Contributors

- @analog-cbarber
- @andreasg123
- @beckermr

- @bryant1410
- @colinbrislawn
- @felker
- @forrestwaters
- @gabrielcnr
- @isuruf
- @jakirkham
- @jeremyjliu
- @jjhelmus
- @jooh
- @jpigla
- @marcelotrevisani
- @melund
- @mfansler
- @mingwandroid
- @msarahan
- @rrigdon
- @scopatz
- @soapy1
- @WillyChen123
- @xhochy

4.5.80 4.7.12 (2019-09-12)

Enhancements

- add support for env file creation based on explicit specs in history (#9093)
- detect prefix paths when -p nor -n not given (#9135)
- Add config parameter to disable conflict finding (for faster time to errors) (#9190)

Bug fixes

- fix race condition with creation of repodata cache dir (#9073)
- fix ProxyError expected arguments (#9123)
- makedirs to initialize .conda folder when registering env - fixes permission errors with .conda folders not existing when package cache gets created (#9215)
- fix list duplicates errors in reading repodata/prefix data (#9132)
- fix neutered specs not being recorded in history, leading to unsatisfiable environments later (#9147)
- Standardize “conda env list” behavior between platforms (#9166)
- add JSON output to conda env create/update (#9204)

- speed up finding conflicting specs (speed regression in 4.7.11) (#9218)

Contributors

- @beenje
- @Bezier89
- @cjmartian
- @forrestwaters
- @jjhelmus
- @martin-raden
- @msarahan
- @nganani
- @rrigdon
- @soapy1
- @WesRoach
- @zheaton

4.5.81 4.7.11 (2019-08-06)

Enhancements

- add config for control of number of threads. These can be set in condarc or using environment variables. Names/default values are: `default_threads/None`, `repodata_threads/None`, `verify_threads/1`, `execute_threads/1` (#9044)

Bug fixes

- fix repodata_fns from condarc not being respected (#8998)
- Fix handling of UpdateModifiers other than FREEZE_INSTALLED (#8999)
- Improve conflict finding graph traversal (#9006)
- Fix setuptools being removed due to conda run_constrains (#9014)
- Avoid calling find_conflicts until all retries are spent (#9015)
- refactor _conda_activate.bat in hopes of improving behavior in parallel environments (#9021)
- Add support for local version specs in PYPI installed packages (#9025)
- fix boto3 initialization race condition (#9037)
- Fix return condition in package_cache_data (#9039)
- utilize libarchive_enabled attribute provided by conda-package-handling to fall back to .tar.bz2 files only. (#9041, #9053)
- Fix menu creation on windows having race condition, leading to popups about python.exe not being found (#9044)
- Improve list error when egg-link leads to extra egg-infos (#9045)
- Fix incorrect RemoveError when operating on an env that has one of conda's deps, but is not the env in which the current conda in use resides (#9054)

Docs

- Document new package format better
- Document `conda init` command
- Document availability of RSS feed for CDN-backed channels that clone

Contributors

- @Bezier89
- @forrestwaters
- @hajapy
- @ihnorton
- @matthewwardrop
- @msarahan
- @rogererens
- @rrigdon
- @soapy1

4.5.82 4.7.10 (2019-07-19)

Bug fixes

- fix merging of specs
- fix bugs in building of chains in prefix graph

Contributors

- @msarahan

4.5.83 4.7.9 (2019-07-18)

Bug fixes

- fix Non records in comprehension
- fix potential keyerror in depth-first search
- fix PackageNotFound attribute error

Contributors

- @jjhelmus
- @msarahan

4.5.84 4.7.8 (2019-07-17)

Improvements

- improve unsatisfiable messages - try to group and explain output better. Remove lots of extraneous stuff that was showing up in 4.7.7 (#8910)
- preload openssl on windows to avoid library conflicts and missing library issues (#8949)

Bug fixes

- fix handling of channels where more than one channel contains packages with similar name, subdir, version and build_number. This was causing mysterious unsatisfiable errors for some users. (#8938)
- reverse logic check in checking channel equality, because == is not reciprocal to != with py27 (no `__ne__`) (#8938)
- fix an infinite loop or otherwise large process with building the unsatisfiable info. Improve the depth-first search implementation. (#8941)
- streamline fallback paths to unfrozen solve in case frozen fails. (#8942)
- Environment activation output only shows `conda activate envname` now, instead of sometimes showing just `activate`. (#8947)

Contributors

- @forrestwaters
- @jjhelmus
- @katietz
- @msarahan
- @rrigdon
- @soapy1

4.5.85 4.7.7 (2019-07-12)

Improvements

- When an update command doesn't do anything because installed software conflicts with the update, information about the conflict is shown, rather than just saying "all requests are already satisfied" (#8899)

Bug fixes

- fix missing package_type attr in finding virtual packages (#8917)
- fix parallel operations of loading index to preserve channel ordering (#8921, #8922)
- filter PrefixRecords out from PackageRecords when making a graph to show unsatisfiable deps. Fixes comparison error between mismatched types. (#8924)
- install entry points before running post-link scripts, because post link scripts may depend on entry points. (#8925)

Contributors

- @jjhelmus
- @msarahan
- @rrigdon
- @soapy1

4.5.86 4.7.6 (2019-07-11)

Improvements

- Improve cuda virtual package conflict messages to show the `__cuda` virtual package as part of the conflict (#8834)

- add additional debugging info to Resolve.solve (#8895)

Bug fixes

- deduplicate error messages being shown for post-link scripts. Show captured stdout/stderr on failure (#8833)
- fix the checkout step in the Windows dev env setup instructions (#8827)
- bail out early when implicit python pinning renders an explicit spec unsatisfiable (#8834)
- handle edge cases in pinned specs better (#8843)
- extract package again if url is None (#8868)
- update docs regarding indexing and subdirs (#8874)
- remove warning about conda-build needing an update that was bothering people (#8884)
- only add repodata fn into cache key when fn is not repodata.json (#8900)
- allow conda to be downgraded with an explicit spec (#8892)
- add target to specs from historic specs (#8901)
- improve message when solving with a repodata file before repodata.json fails (#8907)
- fix distutils usage for "which" functionality. Fix inability to change python version in envs with noarch packages (#8909)
- fix anaconda metapackage being removed because history matching was too restrictive (#8911)
- make freezing less aggressive; add fallback to non-frozen solve (#8912)

Contributors

- @forrestwaters
- @jjhelmus
- @mcopes73
- @msarahan
- @richardjgowers
- @rrigdon
- @soapy1
- @twinssbc

4.5.87 4.7.5 (2019-06-24)

Improvements

- improve wording in informational message when a particular `*_repodata.json` can't be found. No need for alarm. (#8808)

Bug fixes

- restore tests being run on win-32 appveyor (#8801)
- fix Dist class handling of .conda files (#8816)
- fix strict channel priority handling when a package is unsatisfiable and thus not present in the collection (#8819)
- handle JSONDecodeError better when package is corrupted at extract time (#8820)

Contributors

- @dhirschfeld
- @msarahan
- @rrigdon

4.5.88 4.7.4 (2019-06-19)

Improvements

- Revert to and improve the unsatisfiability determination from 4.7.2 that was reverted in 4.7.3. It's faster. (#8783)

Bug fixes

- fix tcsh/csh init scripts (#8792)

Docs improvements

- clean up docs of run_command
- fix broken links
- update docs environment.yaml file to update conda-package-handling
- conda logo favicon
- update strict channel priority info
- noarch package content ported from conda-forge
- add info about conda-forge
- remove references to things as they were before conda 4.1. That was a long time ago. This is not a history book.

Contributors

- @jjhelmus
- @msarahan
- @rrigdon
- @soapy1

4.5.89 4.7.3 (2019-06-14)

Bug fixes

- target prefix overrid applies to entry points in addition to replacements in standard files (#8769)
- Revert to solver-based unsatisfiability determination (#8775)
- fix renaming of existing prompt function in powershell (#8774)

Contributors

- @jjhelmus
- @msarahan
- @rrigdon
- @ScottEvtuch

4.5.90 4.7.2 (2019-06-10)

Behavior changes

- unsatisfiability is determined in a slightly different way now. It no longer uses the SAT solver, but rather determines whether any specs have no candidates at all after running through get_reduced_index. This has been faster in benchmarks, but we welcome further data from your use cases about whether this was a good change. (#8741)
- when using the --only-deps flag for the `install` command, conda now explicitly records those specs in your history. This primarily serves to reduce conda accidentally removing packages that you have actually requested. (#8766)

Improvements

- UnsatisfiableError messages are now grouped into categories and explained a bit better. (#8741)
- –repodata-fn argument can be passed multiple times to have more fallback paths. `repodata_fns` conda config setting does the same thing, but saves you from needing to do it for every command invocation. (#8741)

Bug fixes

- fix channel flip-flopping that was happening when adding a channel other than earlier ones (#8741)
- refactor flow control for multiple repodata files to not use exceptions (#8741)
- force conda to use only old .tar.bz2 files if conda-build <3.18.3 is installed. Conda-build breaks when inspecting file contents, and this is fixed in conda-build 3.18.3 (#8741)
- use --force when using rsync to improve behavior with folders that may exist in the destination somehow. (#8750)
- handle EPERM errors when renaming, because MacOS lets you remove or create files, but not rename them. Thanks Apple. (#8755)
- fix conda removing packages installed via `install` with --only-deps flag when either `update` or `remove` commands are run. See behavior changes above. (#8766)

Contributors

- @csosborn
- @jjhelmus
- @katietz
- @msarahan
- @rrigdon

4.5.91 4.7.1 (2019-05-30)

Improvements

- Base initial solver specs map on explicitly requested specs (new and historic) (#8689)
- Improve anonymization of automatic error reporting (#8715)
- Add option to keep using .tar.bz2 files, in case new .conda isn't working for whatever reason (#8723)

Bug fixes

- fix parsing hyphenated PyPI specs (change hyphens in versions to .) (#8688)
- fix PrefixRecord creation when file inputs are .conda files (#8689)
- fix PrefixRecord creation for pip-installed packages (#8689)
- fix progress bar stopping at 75% (no extract progress with new libarchive) (#8689)
- preserve pre-4.7 download() interface in conda.exports (#8698)
- virtual packages (such as cuda) are represented by leading double underscores by convention, to avoid confusion with existing single underscore packages that serve other purposes (#8738)

Deprecations/Breaking Changes

- The `--prune` flag no longer does anything. Pruning is implicitly the standard behavior now as a result of the initial solver specs coming from explicitly requested specs. Conda will remove packages that are not explicitly requested and are not required directly or indirectly by any explicitly installed package.

Docs improvements

- Document removal of the `free` channel from defaults (#8682)
- Add reference to conda config --describe (#8712)
- Add a tutorial for .condarc modification (#8737)

Contributors

- @alexhall
- @cjmartian
- @kalefranz
- @martinkou
- @msarahan
- @rrigdon
- @soapy1

4.5.92 4.7.0 (2019-05-17)

Improvements

- Implement support for "virtual" CUDA packages, to make conda consider the system-installed CUDA driver and act accordingly (#8267)
- Support and prefer new .conda file format where available (#8265, #8639)
- Use comma-separated env names in prompt when stacking envs (#8431)
- show valid choices in error messages for enums (#8602)
- freeze already-installed packages when running `conda install` as a first attempt, to speed up the solve in existing envs. Fall back to full solve as necessary (#8260, #8626)
- add optimization criterion to prefer arch over noarch packages when otherwise equivalent (#8267)
- Remove `free` channel from defaults collection. Add `restore_free_channel` config parameter if you want to keep it. (#8579)

- Improve unsatisfiable hints (#8638)
- Add capability to use custom repodata filename, for smaller subsets of repodata (#8670)
- Parallelize SubdirData readup (#8670)
- Parallelize transaction verification and execution (#8670)

Bug fixes

- Fix PATH handling with deactivate.d scripts (#8464)
- Fix usage of deprecated collections ABCs (#)
- fix tcsh/csh initialization block (#8591)
- fix missing CWD display in powershell prompt (#8596)
- `wrap_subprocess_call`: fallback to sh if no bash (#8611)
- move `TemporaryDirectory` to avoid importing from `conda.compat` (#8671)
- fix missing conda-package-handling dependency in dev/start (#8624)
- fix `path_to_url` string index out of range error (#8265)
- fix conda init for xonsh (#8644)
- fix fish activation (#8645)
- improve error handling for read-only filesystems (#8665, #8674)
- break out of minimization when bisection has nowhere to go (#8672)
- Handle None values for link channel name gracefully (#8680)

Contributors

- @chrisburr
- @EternalPhane
- @jjhelmus
- @kalefranz
- @mbargull
- @msarahan
- @rrigdon
- @scopatz
- @seibert
- @soapy1
- @nehaljwani
- @nh3
- @teake
- @yuvalreches

4.5.93 4.6.14 (2019-04-17)

Bug fixes

- export extra function in powershell Conda.psm1 script (fixes anaconda powershell prompt) (#8570)

Contributors

- @msarahan

4.5.94 4.6.13 (2019-04-16)

Bug fixes

- disable `test_legacy_repodata` on win-32 (missing dependencies) (#8540)
- Fix activation problems on windows with bash, powershell, and batch. Improve tests. (#8550, #8564)
- pass -U flag to for pip dependencies in conda env when running "conda env update" (#8542)
- rename `conda.common.os` to `conda.common._os` to avoid shadowing os built-in (#8548)
- raise exception when pip subprocess fails with conda env (#8562)
- fix installing recursive requirements.txt files in conda env specs with python 2.7 (#8562)
- Don't modify powershell prompt when "changeps1" setting in condarc is False (#8465)

Contributors

- @dennispg
- @jjhelmus
- @jpgill86
- @mingwandroid
- @msarahan
- @noahp

4.5.95 4.6.12 (2019-04-10)

Bug fixes

- Fix compat import warning (#8507)
- Adjust collections import to avoid deprecation warning (#8499)
- Fix bug in CLI tests (#8468)
- Disallow the number sign in environment names (#8521)
- Workaround issues with noarch on certain repositories (#8523)
- Fix activation on Windows when spaces are in path (#8503)
- Fix conda init profile modification for powershell (#8531)
- Point conda.bat to condabin (#8517)
- Fix various bugs in activation (#8520, #8528)

Docs improvements

- Fix links in README (#8482)
- Changelogs for 4.6.10 and 4.6.11 (#8502)

Contributors

@Bezier89 @duncanmmacleod @ivigamberdiev @javabrett @jjhelmus @katietz @mingwandroid @msarahan @nehaljwani @rrigdon

4.5.96 4.6.11 (2019-04-04)

Bug fixes

- Remove sys.prefix from front of PATH in basic_posix (#8491)
- add import to fix conda.core.index.get_index (#8495)

Docs improvements

- Changelogs for 4.6.10

Contributors

- @jjhelmus
- @mingwandroid
- @msarahan

4.5.97 4.6.10 (2019-04-01)

Bug fixes

- Fix python-3 only FileNotFoundError usage in initialize.py (#8470)
- Fix more JSON encode errors for the _Null data type (#8471)
- Fix non-posix-compliant == in conda.sh (#8475, #8476)
- improve detection of pip dependency in environment.yml files to avoid warning message (#8478)
- fix condabin\conda.bat use of dp0, making PATH additions incorrect (#8480)
- init_fish_user: don't assume config file exists (#8481)
- Fix for chcp output ending with . (#8484)

Docs improvements

- Changelogs for 4.6.8, 4.6.9

Contributors

- @duncanmmacleod
- @nehaljwani
- @ilango100
- @jjhelmus
- @mingwandroid

- @msarahan
- @rrigdon

4.5.98 4.6.9 (2019-03-29)

Improvements

- Improve CI for docs commits (#8387, #8401, #8417)
- Implement `conda init --reverse` to undo rc file and registry changes (#8400)
- Improve handling of unicode systems (#8342, #8435)
- Force the "COMSPEC" environment variable to always point to cmd.exe on windows. This was an implicit assumption that was not always true. (#8457, #8461)

Bug fixes

- Add central C:/ProgramData/conda as a search path on Windows (#8272)
- remove direct use of ruamel_yaml (prefer internal abstraction, yaml_load) (#8392)
- Fix/improve `conda init` support for fish shell (#8437)
- Improve solver behavior in the presence of inconsistent environments (such as pip as a conda dependency of python, but also installed via pip itself) (#8444)
- Handle read-only filesystems for environments.txt (#8451, #8453)
- Fix conda env commands involving pip-installed dependencies being installed into incorrect locations (#8435)

Docs improvements

- updated cheatsheet (#8402)
- updated color theme (#8403)

Contributors

- @blackgear
- @dhirschfeld
- @jakirkham
- @jjhelmus
- @katietz
- @mingwandroid
- @msarahan
- @nehaljwani
- @rrigdon
- @soapy1
- @spamlrot-tic

4.5.99 4.6.8 (2019-03-06)

Bug fixes

- detect when parser fails to parse arguments (#8328)
- separate post-link script running from package linking. Do linking of all packages first, then run any post-link scripts after all packages are present. Ideally, more forgiving in presence of cycles. (#8350)
- quote path to temporary requirements files generated by conda env. Fixes issues with spaces. (#8352)
- improve some exception handling around checking for presence of folders in extraction of tarballs (#8360)
- fix reporting of packages when channel name is None (#8379)
- fix the post-creation helper message from "source activate" to "conda activate" (#8370)
- Add safety checks for directory traversal exploits in tarfiles. These may be disabled using the `safety_checks` configuration parameter. (#8374)

Docs improvements

- document MKL DLL hell and new Python env vars to control DLL search behavior (#8315)
- add github template for reporting speed issues (#8344)
- add in better use of sphinx admonitions (notes, warnings) for better accentuation in docs (#8348)
- improve skipping CI builds when only docs changes are involved (#8336)

Contributors

- @albertmichaelj
- @jjhelmus
- @matta9001
- @msarahan
- @rrigdon
- @soapy1
- @steffenvan

4.5.100 4.6.7 (2019-02-21)

Bug fixes

- skip scanning folders for contents during reversal of transactions. Just ignore folders. A bit messier, but a lot faster. (#8266)
- fix some logic in renaming trash files to fix permission errors (#8300)
- wrap pip subprocess calls in conda-env more cleanly and uniformly (#8307)
- revert conda prepending to PATH in cli main file on windows (#8307)
- simplify `conda run` code to use activation subprocess wrapper. Fix a few conda tests to use `conda run`. (#8307)

Docs improvements

- fixed duplicated “to” in managing envs section (#8298)
- flesh out docs on activation (#8314)
- correct git syntax for adding a remote in dev docs (#8316)
- unpin sphinx version in docs requirements (#8317)

Contributors

- @jjhelmus
- @MarckK
- @msarahan
- @rrigdon
- @samgd

4.5.101 4.6.6 (2019-02-18)

Bug fixes

- fix incorrect syntax prepending to PATH for conda CLI functionality (#8295)
- fix rename_tmp.bat operating on folders, leading to hung interactive dialogs. Operate only on files. (#8295)

Contributors

- @mingwandroid
- @msarahan

4.5.102 4.6.5 (2019-02-15)

Bug fixes

- Make super in resolve.py python 2 friendly (#8280)
- support unicode paths better in activation scripts on Windows (#)
- set PATH for conda.bat to include Conda’s root prefix, so that libraries can be found when using conda when the root env is not activated (#8287, #8292)
- clean up warnings/errors about rsync and trash files (#8290)

Contributors

- @jjhelmus
- @mingwandroid
- @msarahan
- @rrigdon

4.5.103 4.6.4 (2019-02-13)

Improvements

- allow configuring location of instrumentation records (#7849)
- prepend conda-env pip commands with env activation to fix library loading (#8263)

Bug fixes

- resolve #8176 SAT solver choice error handling (#8248)
- document `pip_interop_enabled` config parameter (#8250)
- ensure prefix temp files are inside prefix (#8253)
- ensure `script_caller` is bound before use (#8254)
- fix overzealous removal of folders after cleanup of failed post-link scripts (#8259)
- fix #8264: Allow 'int' datatype for values to non-sequence parameters (#8268)

Deprecations/Breaking Changes

- remove experimental `featureless_minimization_disabled` feature flag (#8249)

Contributors

- @davemasino
- @geremih
- @jjhelmus
- @kalefranz
- @msarahan
- @minrk
- @nehaljwani
- @prusse-martin
- @rrigdon
- @soapy1

4.5.104 4.6.3 (2019-02-07)

Improvements

- Implement `-stack` switch for powershell usage of conda (#8217)
- Enable system-wide initialization for conda shell support (#8219)
- Activate environments prior to running post-link scripts (#8229)
- Instrument more solve calls to prioritize future optimization efforts (#8231)
- print more env info when searching in envs (#8240)

Bug fixes

- resolve #8178, fix conda pip interop assertion error with egg folders (#8184)
- resolve #8157, fix token leakage in errors and config output (#8163)
- resolve #8185, fix conda package filtering with embedded/vendored python metadata (#8198)
- resolve #8199, fix errors on .* in version specs that should have been specific to the ~= operator (#8208)
- fix .bat scripts for handling paths on Windows with spaces (#8215)
- fix powershell scripts for handling paths on Windows with spaces (#8222)
- handle missing rename script more gracefully (especially when updating/installing conda itself) (#8212)

Contributors

- @dhirschfeld
- @jjhelmus
- @kalefranz
- @msarahan
- @murrayreadccdc
- @nehaljwani
- @rrigdon
- @soapy1

4.5.105 4.6.2 (2019-01-29)

Improvements

- Documentation restructuring/improvements (#8139, #8143)
- rewrite rm_rf to use native system utilities and rename trash files (#8134)

Bug fixes

- fix UnavailableInvalidChannel errors when only noarch subdir is present (#8154)
- document, but disable the `allow_conda_downgrades` flag, pending re-examination of the warning, which was blocking conda operations after an upgrade-downgrade cycle across minor versions. (#8160)
- fix conda env export missing pip entries without use of pip interop enabled setting (#8165)

Contributors

- @jjhelmus
- @msarahan
- @nehaljwani
- @rrigdon

4.5.106 4.5.13 (2019-01-29)

Improvements

- document the allow_conda_downgrades configuration parameter (#8034)
- remove conda upgrade message (#8161)

Contributors

- @msarahan
- @nehaljwani

4.5.107 4.6.1 (2019-01-21)

Improvements

- optimizations in `get_reduced_index` (#8117, #8121, #8122)

Bug Fixes

- fix faulty onerror call for rm (#8053)
- fix activate.bat to use more direct call to conda.bat (don't require conda init; fix non-interactive script) (#8113)

Contributors

- @jjhelmus
- @msarahan
- @pv

4.5.108 4.6.0 (2019-01-15)

New Feature Highlights

- resolve #7053 preview support for conda operability with pip; disabled by default (#7067, #7370, #7710, #8050)
- conda initialize (#6518, #7388, #7629)
- resolve #7194 add '–stack' flag to 'conda activate'; remove max_shlvl config (#7195, #7226, #7233)
- resolve #7087 add non-conda-installed python packages into PrefixData (#7067, #7370)
- resolve #2682 add 'conda run' preview support (#7320, #7625)
- resolve #626 conda wrapper for PowerShell (#7794, #7829)

Deprecations/Breaking Changes

- resolve #6915 remove 'conda env attach' and 'conda env upload' (#6916)
- resolve #7061 remove pkgs/pro from defaults (#7162)
- resolve #7078 add deprecation warnings for 'conda.cli.activate', 'conda.compat', and 'conda.install' (#7079)
- resolve #7194 add '–stack' flag to 'conda activate'; remove max_shlvl config (#7195)
- resolve #6979, #7086 remove Dist from majority of project (#7216, #7252)
- fix #7362 remove --license from conda info and related code paths (#7386)
- resolve #7309 deprecate 'conda info package_name' (#7310)

- remove ‘conda clean --source-cache’ and defer to conda-build (#7731)
- resolve #7724 move windows package cache and envs dirs back to .conda directory (#7725)
- disallow env names with colons (#7801)

Improvements

- import speedups (#7122)
- –help cleanup (#7120)
- fish autocompletion for conda env (#7101)
- remove reference to ‘system’ channel (#7163)
- add http error body to debug information (#7160)
- warn creating env name with space is not supported (#7168)
- support complete MatchSpec syntax in environment.yml files (#7178)
- resolve #4274 add option to remove an existing environment with ‘conda create’ (#7133)
- add ability for conda prompt customization via ‘env_prompt’ config param (#7047)
- resolve #7063 add license and license_family to MatchSpec for ‘conda search’ (#7064)
- resolve #7189 progress bar formatting improvement (#7191)
- raise log level for errors to error (#7229)
- add to conda.exports (#7217)
- resolve #6845 add option -S / --satisfied-skip-solve to exit early for satisfied specs (#7291)
- add NoBaseEnvironmentError and DirectoryNotACondaEnvironmentError (#7378)
- replace menuinst subprocessing by ctypes win elevation (4.6.0a3) (#7426)
- bump minimum requests version to stable, unbundled release (#7528)
- resolve #7591 updates and improvements from namespace PR for 4.6 (#7599)
- resolve #7592 compatibility shims (#7606)
- user-agent context refactor (#7630)
- solver performance improvements with benchmarks in common.logic (#7676)
- enable fuzzy-not-equal version constraint for pip interop (#7711)
- add -d short option for --dry-run (#7719)
- add --force-pkgs-dirs option to conda clean (#7719)
- address #7709 ensure --update-deps unlocks specs from previous user requests (#7719)
- add package timestamp information to output of ‘conda search --info’ (#7722)
- resolve #7336 ‘conda search’ tries “fuzzy match” before showing PackagesNotFound (#7722)
- resolve #7656 strict channel priority via ‘channel_priority’ config option or --strict-channel-priority CLI flag (#7729)
- performance improvement to cache **hash** value on PackageRecord (#7715)
- resolve #7764 change name of ‘condacmd’ dir to ‘condabin’; use on all platforms (#7773)
- resolve #7782 implement PEP-440 ‘~=’ compatible release operator (#7783)

- disable timestamp prioritization when not needed (#7894, #8012)
- compile pyc files for noarch packages in batches (#8015)
- disable per-file sha256 safety checks by default; add extra_safety_checks condarc option to enable them (#8017)
- shorten retries for file removal on windows, where in-use files can't be removed (#8024)
- expand env vars in `custom_channels`, `custom_multichannels`, `default_channels`, `migrated_custom_channels`, and `whitelist_channels` (#7826)
- encode repodata to utf-8 while caching, to fix unicode characters in repodata (#7873)

Bug Fixes

- fix #7107 verify hangs when a package is corrupted (#7131)
- fix #7145 progress bar uses stderr instead of stdout (#7146)
- fix typo in conda.fish (#7152)
- fix #2154 conda remove should complain if requested removals don't exist (#7135)
- fix #7094 exit early for --dry-run with explicit and clone (#7096)
- fix activation script sort order (#7176)
- fix #7109 incorrect chown with sudo (#7180)
- fix #7210 add suppressed --mkdir back to 'conda create' (fix for 4.6.0a1) (#7211)
- fix #5681 conda env create / update when --file does not exist (#7385)
- resolve #7375 enable conda config --set update_modifier (#7377)
- fix #5885 improve conda env error messages and add extra tests (#7395)
- msys2 path conversion (#7389)
- fix autocompletion in fish (#7575)
- fix #3982 following 4.4 activation refactor (#7607)
- fix #7242 configuration load error message (#7243)
- fix conda env compatibility with pip 18 (#7612)
- fix #7184 remove conflicting specs to find solution to user's active request (#7719)
- fix #7706 add condacmd dir to cmd.exe path on first activation (#7735)
- fix #7761 spec handling errors in 4.6.0b0 (#7780)
- fix #7770 'conda list regex' only applies regex to package name (#7784)
- fix #8076 load metadata from index to resolve inconsistent envs (#8083)

Non-User-Facing Changes

- resolve #6595 use OO inheritance in activate.py (#7049)
- resolve #7220 pep8 project renamed to pycodestyle (#7221)
- proxy test routine (#7308)
- add .mailmap and .cla-signers (#7361)
- add copyright headers (#7367)
- rename common.platform to common.os and split among windows, linux, and unix utils (#7396)

- fix windows test failures when symlink not available (#7369)
- test building conda using conda-build (#7251)
- solver test metadata updates (#7664)
- explicitly add Mapping, Sequence to common.compat (#7677)
- add debug messages to communicate solver stages (#7803)
- add undocumented sat_solver config parameter (#7811)

Preview Releases

- 4.6.0a1 at d5bec21d1f64c3bc66c2999cfc690681e9c46177 on 2018-04-20
- 4.6.0a2 at c467517ca652371ebc4224f0d49315b7ec225108 on 2018-05-01
- 4.6.0b0 at 21a24f02b2687d0895de04664a4ec23ccc75c33a on 2018-09-07
- 4.6.0b1 at 1471f043eed980d62f46944e223f0add6a9a790b on 2018-10-22
- 4.6.0rc1 at 64bde065f8343276f168d2034201115dff7c5753 on 2018-12-31

Contributors

- @cgranade
- @fabioz
- @geremih
- @goanpeca
- @jesse-
- @jjhelmus
- @kalefranz
- @makbigc
- @mandeep
- @mbargull
- @msarahan
- @nehaljwani
- @ohadravid
- @teake

4.5.109 4.5.12 (2018-12-10)

Improvements

- backport 'allow_conda_downgrade' configuration parameter, default is False (#7998)
- speed up verification by disabling per-file sha256 checks (#8017)
- indicate Python 3.7 support in setup.py file (#8018)
- speed up solver by reduce the size of reduced index (#8016)
- speed up solver by skipping timestamp minimization when not needed (#8012)

- compile pyc files more efficiently, will speed up install of noarch packages (#8025)
- avoid waiting for removal of files on Windows when possible (#8024)

Bug Fixes

- update integration tests for removal of 'features' key (#7726)
- fix conda.bat return code (#7944)
- ensure channel name is not NoneType (#8021)

Contributors

- @debionne
- @jjhelmus
- @kalefranz
- @msarahan
- @nehaljwani

4.5.110 4.5.11 (2018-08-21)

Improvements

- resolve #7672 compatibility with ruamel.yaml 0.15.54 (#7675)

Contributors

- @CJ-Wright
- @mbargull

4.5.111 4.5.10 (2018-08-13)

Bug Fixes

- fix conda env compatibility with pip 18 (#7627)
- fix py37 compat 4.5.x (#7641)
- fix #7451 don't print name, version, and size if unknown (#7648)
- replace glob with fnmatch in PrefixData (#7645)

Contributors

- @jesse-
- @nehaljwani

4.5.112 4.5.9 (2018-07-30)

Improvements

- resolve #7522 prevent conda from scheduling downgrades (#7598)
- allow skipping feature maximization in resolver (#7601)

Bug Fixes

- fix #7559 symlink stat in localfs adapter (#7561)
- fix #7486 activate with no PATH set (#7562)
- resolve #7522 prevent conda from scheduling downgrades (#7598)

Contributors

- @kalefranz
- @loriab

4.5.113 4.5.8 (2018-07-10)

Bug Fixes

- fix #7524 should_bypass_proxies for requests 2.13.0 and earlier (#7525)

Contributors

- @kalefranz

4.5.114 4.5.7 (2018-07-09)

Improvements

- resolve #7423 add upgrade error for unsupported repodata_version (#7415)
- raise CondaUpgradeError for conda version downgrades on environments (#7517)

Bug Fixes

- fix #7505 temp directory for UnlinkLinkTransaction should be in target prefix (#7516)
- fix #7506 requests monkeypatch fallback for old requests versions (#7515)

Contributors

- @kalefranz
- @nehaljwani

4.5.115 4.5.6 (2018-07-06)

Bug Fixes

- resolve #7473 py37 support (#7499)
- fix #7494 History spec parsing edge cases (#7500)
- fix requests 2.19 incompatibility with NO_PROXY env var (#7498)
- resolve #7372 disable http error uploads and CI cleanup (#7498, #7501)

Contributors

- @kalefranz

4.5.116 4.5.5 (2018-06-29)

Bug Fixes

- fix #7165 conda version check should be restricted to channel conda is from (#7289, #7303)
- fix #7341 ValueError n cannot be negative (#7360)
- fix #6691 fix history file parsing containing comma-joined version specs (#7418)
- fix msys2 path conversion (#7471)

Contributors

- @goanpeca
- @kalefranz
- @mingwandroid
- @mbargull

4.5.117 4.5.4 (2018-05-14)

Improvements

- resolve #7189 progress bar improvement (#7191 via #7274)

Bug Fixes

- fix twofold tarball extraction, improve progress update (#7275)
- fix #7253 always respect copy LinkType (#7269)

Contributors

- @jakirkham
- @kalefranz
- @mbargull

4.5.118 4.5.3 (2018-05-07)

Bug Fixes

- fix #7240 conda's configuration context is not initialized in conda.exports (#7244)

4.5.119 4.5.2 (2018-04-27)

Bug Fixes

- fix #7107 verify hangs when a package is corrupted (#7223)
- fix #7094 exit early for --dry-run with explicit and clone (#7224)
- fix activation/deactivation script sort order (#7225)

4.5.120 4.5.1 (2018-04-13)

Improvements

- resolve #7075 add anaconda.org search message to PackagesNotFoundError (#7076)
- add CondaError details to auto-upload reports (#7060)

Bug Fixes

- fix #6703,#6981 index out of bound when running deactivate on fish shell (#6993)
- properly close over $_CONDA_EXE variable (#7004)
- fix condarc map parsing with comments (#7021)
- fix #6919 csh prompt (#7041)
- add _file_created attribute (#7054)
- fix handling of non-ascii characters in custom_multichannels (#7050)
- fix #6877 handle non-zero return in CSH (#7042)
- fix #7040 update tqdm to version 4.22.0 (#7157)

4.5.121 4.5.0 (2018-03-20)

New Feature Highlights

- A new flag, '–envs', has been added to 'conda search'. In this mode, 'conda search' will look for the package query in existing conda environments on your system. If ran as UID 0 (i.e. root) on unix systems or as an Administrator user on Windows, all known conda environments for all users on the system will be searched. For example, 'conda search --envs openssl' will show the openssl version and environment location for all conda-installed openssl packages.

Deprecations/Breaking Changes

- resolve #6886 transition defaults from repo.continuum.io to repo.anaconda.com (#6887)
- resolve #6192 deprecate 'conda help' in favor of --help CLI flag (#6918)
- resolve #6894 add http errors to auto-uploaded error reports (#6895)

Improvements

- resolve #6791 conda search --envs (#6794)
- preserve exit status in fish shell (#6760)
- resolve #6810 add CONDA_EXE environment variable to activate (#6923)
- resolve #6695 outdated conda warning respects --quiet flag (#6935)
- add instructions to activate default environment (#6944)

API

- resolve #5610 add PrefixData, SubdirData, and PackageCacheData to conda/api.py (#6922)

Bug Fixes

- channel matchspec fixes (#6893)
- fix #6930 add missing return statement to S3Adapter (#6931)
- fix #5802, #6736 enforce disallowed_packages configuration parameter (#6932)
- fix #6860 infinite recursion in resolve.py for empty track_features (#6928)
- set encoding for PY2 stdout/stderr (#6951)
- fix #6821 non-deterministic behavior from MatchSpec merge clobbering (#6956)
- fix #6904 logic errors in prefix graph data structure (#6929)

Non-User-Facing Changes

- fix several lgtm.com flags (#6757, #6883)
- cleanups and refactors for conda 4.5 (#6889)
- unify location of record types in conda/models/records.py (#6924)
- resolve #6952 memoize url search in package cache loading (#6957)

4.5.122 4.4.11 (2018-02-23)

Improvements

- resolve #6582 swallow_broken_pipe context manager and Spinner refactor (#6616)
- resolve #6882 document max_shlvl (#6892)
- resolve #6733 make empty env vars sequence-safe for sequence parameters (#6741)
- resolve #6900 don't record conda skeleton environments in environments.txt (#6908)

Bug Fixes

- fix potential error in ensure_pad(); add more tests (#6817)
- fix #6840 handle error return values in conda.sh (#6850)
- use conda.gateways.disk for misc.py imports (#6870)
- fix #6672 don't update conda during conda-env operations (#6773)
- fix #6811 don't attempt copy/remove fallback for rename failures (#6867)
- fix #6667 aliased posix commands (#6669)
- fix #6816 fish environment autocomplete (#6885)
- fix #6880 build_number comparison not functional in match_spec (#6881)
- fix #6910 sort key prioritizes build string over build number (#6911)
- fix #6914, #6691 conda can fail to update packages even though newer versions exist (#6921)
- fix #6899 handle Unicode output in activate commands (#6909)

4.5.123 4.4.10 (2018-02-09)

Bug Fixes

- fix #6837 require at least futures 3.0.0 (#6855)
- fix #6852 ensure temporary path is writable (#6856)
- fix #6833 improve feature mismatch metric (via 4.3.34 #6853)

4.5.124 4.4.9 (2018-02-06)

Improvements

- resolve #6632 display package removal plan when deleting an env (#6801)

Bug Fixes

- fix #6531 don't drop credentials for conda-build workaround (#6798)
- fix external command execution issue (#6789)
- fix #5792 conda env export error common in path (#6795)
- fix #6390 add CorruptedEnvironmentError (#6778)
- fix #5884 allow --insecure CLI flag without contradicting meaning of ssl_verify (#6782)
- fix MatchSpec.match() accepting dict (#6808)
- fix broken Anaconda Prompt for users with spaces in paths (#6825)
- JSONDecodeError was added in Python 3.5 (#6848)
- fix #6796 update PATH/prompt on reactivate (#6828)
- fix #6401 non-ascii characters on windows using expanduser (#6847)
- fix #6824 import installers before invoking any (#6849)

4.5.125 4.4.8 (2018-01-25)

Improvements

- allow falsey values for default_python to avoid pinning python (#6682)
- resolve #6700 add message for no space left on device (#6709)
- make variable 'sourced' local for posix shells (#6726)
- add column headers to conda list results (#5726)

Bug Fixes

- fix #6713 allow parenthesis in prefix path for conda.bat (#6722)
- fix #6684 --force message (#6723)
- fix #6693 KeyError with '–update-deps' (#6694)
- fix aggressive_update_packages availability (#6727)
- fix #6745 don't truncate channel priority map in conda installer (#6746)
- add workaround for system Python usage by lsb_release (#6769)
- fix #6624 can't start new thread (#6653)

- fix #6628 'conda install --rev' in conda 4.4 (#6724)
- fix #6707 FileNotFoundError when extracting tarball (#6708)
- fix #6704 unexpected token in conda.bat (#6710)
- fix #6208 return for no pip in environment (#6784)
- fix #6457 env var cleanup (#6790)
- fix #6645 escape paths for argparse help (#6779)
- fix #6739 handle unicode in environment variables for py2 activate (#6777)
- fix #6618 RepresenterError with 'conda config --set' (#6619)
- fix #6699 suppress memory error upload reports (#6776)
- fix #6770 CRLF for cmd.exe (#6775)
- fix #6514 add message for case-insensitive filesystem errors (#6764)
- fix #6537 AttributeError value for url not set (#6754)
- fix #6748 only warn if unable to register environment due to EACCES (#6752)

4.5.126 4.4.7 (2018-01-08)

Improvements

- resolve #6650 add upgrade message for unicode errors in python 2 (#6651)

Bug Fixes

- fix #6643 difference between '==' and 'exact_match_' (#6647)
- fix #6620 KeyError(u'CONDA_PREFIX',) (#6652)
- fix #6661 remove env from environments.txt (#6662)
- fix #6629 'conda update --name' AssertionError (#6656)
- fix #6630 repodata AssertionError (#6657)
- fix #6626 add setuptools as constrained dependency (#6654)
- fix #6659 conda list explicit should be dependency sorted (#6671)
- fix #6665 KeyError for channel '' (#6668, #6673)
- fix #6627 AttributeError on 'conda activate' (#6655)

4.5.127 4.4.6 (2017-12-31)

Bug Fixes

- fix #6612 do not assume Anaconda Python on Windows nor Library\bin hack (#6615)
- recipe test improvements and associated bug fixes (#6614)

4.5.128 4.4.5 (2017-12-29)

Bug Fixes

- fix #6577, #6580 single quote in PS1 (#6585)
- fix #6584 os.getcwd() FileNotFound (#6589)

- fix #6592 deactivate command order (#6602)
- fix #6579 python not recognized as command (#6588)
- fix #6572 cached repodata PermissionsError (#6573)
- change instances of 'root' to 'base' (#6598)
- fix #6607 use subprocess rather than execv for conda command extensions (#6609)
- fix #6581 git-bash activation (#6587)
- fix #6599 space in path to base prefix (#6608)

4.5.129 4.4.4 (2017-12-24)

Improvements

- add SUDO_ env vars to info reports (#6563)
- add additional information to the #6546 exception (#6551)

Bug Fixes

- fix #6548 'conda update' installs packages not in prefix #6550
- fix #6546 update after creating an empty env (#6568)
- fix #6557 conda list FileNotFoundError (#6558)
- fix #6554 package cache FileNotFoundError (#6555)
- fix #6529 yaml parse error (#6560)
- fix #6562 repodata_record.json permissions error stack trace (#6564)
- fix #6520 --use-local flag (#6526)

4.5.130 4.4.3 (2017-12-22)

Improvements

- adjust error report message (#6534)

Bug Fixes

- fix #6530 package cache JsonDecodeError / ValueError (#6533)
- fix #6538 BrokenPipeError (#6540)
- fix #6532 remove anaconda metapackage hack (#6539)
- fix #6536 'conda env export' for old versions of pip (#6535)
- fix #6541 py2 and unicode in environments.txt (#6542)

Non-User-Facing Changes

- regression tests for #6512 (#6515)

4.5.131 4.4.2 (2017-12-22)

Deprecations/Breaking Changes

- resolve #6523 don't prune with --update-all (#6524)

Bug Fixes

- fix #6508 environments.txt permissions error stack trace (#6511)
- fix #6522 error message formatted incorrectly (#6525)
- fix #6516 hold channels over from get_index to install_actions (#6517)

4.5.132 4.4.1 (2017-12-21)

Bug Fixes

- fix #6512 reactivate does not accept arguments (#6513)

4.5.133 4.4.0 (2017-12-20)

Recommended change to enable conda in your shell

With the release of conda 4.4, we recommend a change to how the `conda` command is made available to your shell environment. All the old methods still work as before, but you'll need the new method to enable the new `conda activate` and `conda deactivate` commands.

For the "Anaconda Prompt" on Windows, there is no change.

For Bourne shell derivatives (bash, zsh, dash, etc.), you likely currently have a line similar to

```
export PATH="/opt/conda/bin:$PATH"
```

in your `~/.bashrc` file (or `~/.bash_profile` file on macOS). The effect of this line is that your base environment is put on PATH, but without actually *activating* that environment. (In 4.4 we've renamed the 'root' environment to the 'base' environment.) With conda 4.4, we recommend removing the line where the `PATH` environment variable is modified, and replacing it with

```
. /opt/conda/etc/profile.d/conda.sh
conda activate base
```

In the above, it's assumed that `/opt/conda` is the location where you installed miniconda or Anaconda. It may also be something like `~/Anaconda3` or `~/miniconda2`.

For system-wide conda installs, to make the `conda` command available to all users, rather than manipulating individual `~/.bashrc` (or `~/.bash_profile`) files for each user, just execute once

```
$ sudo ln -s /opt/conda/etc/profile.d/conda.sh /etc/profile.d/conda.sh
```

This will make the `conda` command itself available to all users, but conda's base (root) environment will *not* be activated by default. Users will still need to run `conda activate base` to put the base environment on PATH and gain access to the executables in the base environment.

After updating to conda 4.4, we also recommend pinning conda to a specific channel. For example, executing the command

```
$ conda config --system --add pinned_packages conda-canary::conda
```

will make sure that whenever conda is installed or changed in an environment, the source of the package is always being pulled from the `conda-canary` channel. This will be useful for people who use `conda-forge`, to prevent conda from flipping back and forth between 4.3 and 4.4.

New Feature Highlights

- **conda activate**: The logic and mechanisms underlying environment activation have been reworked. With conda 4.4, `conda activate` and `conda deactivate` are now the preferred commands for activating and deactivating environments. You'll find they are much more snappy than the `source activate` and `source deactivate` commands from previous conda versions. The `conda activate` command also has advantages of (1) being universal across all OSes, shells, and platforms, and (2) not having path collisions with scripts from other packages like python virtualenv's activate script.

- **constrained, optional dependencies**: Conda now allows a package to constrain versions of other packages installed alongside it, even if those constrained packages are not themselves hard dependencies for that package. In other words, it lets a package specify that, if another package ends up being installed into an environment, it must at least conform to a certain version specification. In effect, constrained dependencies are a type of "reverse" dependency. It gives a tool to a parent package to exclude other packages from an environment that might otherwise want to depend on it.

  Constrained optional dependencies are supported starting with conda-build 3.0 (via [conda/conda-build#2001[(https://github.com/conda/conda-build/pull/2001)). A new `run_constrained` keyword, which takes a list of package specs similar to the `run` keyword, is recognized under the `requirements` section of `meta.yaml`. For backward compatibility with versions of conda older than 4.4, a requirement may be listed in both the `run` and the `run_constrained` section. In that case older versions of conda will see the package as a hard dependency, while conda 4.4 will understand that the package is meant to be optional.

  Optional, constrained dependencies end up in `repodata.json` under a `constrains` keyword, parallel to the `depends` keyword for a package's hard dependencies.

- **enhanced package query language**: Conda has a built-in query language for searching for and matching packages, what we often refer to as `MatchSpec`. The MatchSpec is what users input on the command line when they specify packages for `create`, `install`, `update`, and `remove` operations. With this release, MatchSpec (rather than a regex) becomes the default input for `conda search`. We have also substantially enhanced our MatchSpec query language.

  For example,

  ```
  conda install conda-forge::python
  ```

  is now a valid command, which specifies that regardless of the active list of channel priorities, the python package itself should come from the `conda-forge` channel. As before, the difference between `python=3.5` and `python==3.5` is that the first contains a "*fuzzy*" version while the second contains an *exact* version. The fuzzy spec will match all python packages with versions `>=3.5` and `<3.6`. The exact spec will match only python packages with version `3.5`, `3.5.0`, `3.5.0.0`, etc. The canonical string form for a MatchSpec is thus

  ```
  (channel::)name(version(build_string))
  ```

  which should feel natural to experienced conda users. Specifications however are often necessarily more complicated than this simple form can support, and for these situations we've extended the specification to include an optional square bracket `[]` component containing comma-separated key-value pairs to allow matching on most any field contained in a package's metadata. Take, for example,

  ```
  conda search 'conda-forge/linux-64::*[md5=e42a03f799131d5af4196ce31a1084a7]' --info
  ```

  which results in information for the single package

```
cytoolz 0.8.2 py35_0
--------------------
file name   : cytoolz-0.8.2-py35_0.tar.bz2
name        : cytoolz
version     : 0.8.2
build string: py35_0
build number: 0
size        : 1.1 MB
arch        : x86_64
platform    : Platform.linux
license     : BSD 3-Clause
subdir      : linux-64
url         : https://conda.anaconda.org/conda-forge/linux-64/cytoolz-0.8.2-py35_0.
↪tar.bz2
md5         : e42a03f799131d5af4196ce31a1084a7
dependencies:
  - python 3.5*
  - toolz >=0.8.0
```

The square bracket notation can also be used for any field that we match on outside the package name, and will override information given in the "simple form" position. To give a contrived example, `python==3.5[version='>=2.7,<2.8']` will match `2.7.*` versions and not `3.5`.

- **environments track user-requested state**: Building on our enhanced MatchSpec query language, conda environments now also track and differentiate (a) packages added to an environment because of an explicit user request from (b) packages brought into an environment to satisfy dependencies. For example, executing

```
conda install conda-forge::scikit-learn
```

will confine all future changes to the scikit-learn package in the environment to the conda-forge channel, until the spec is changed again. A subsequent command `conda install scikit-learn=0.18` would drop the `conda-forge` channel restriction from the package. And in this case, scikit-learn is the only user-defined spec, so the solver chooses dependencies from all configured channels and all available versions.

- **errors posted to core maintainers**: In previous versions of conda, unexpected errors resulted in a request for users to consider posting the error as a new issue on conda's github issue tracker. In conda 4.4, we've implemented a system for users to opt-in to sending that same error report via an HTTP POST request directly to the core maintainers.

  When an unexpected error is encountered, users are prompted with the error report followed by a `[y/N]` input. Users can elect to send the report, with 'no' being the default response. Users can also permanently opt-in or opt-out, thereby skipping the prompt altogether, using the boolean `report_errors` configuration parameter.

- **various UI improvements**: To push through some of the big leaps with transactions in conda 4.3, we accepted some regressions on progress bars and other user interface features. All of those indicators of progress, and more, have been brought back and further improved.

- **aggressive updates**: Conda now supports an `aggressive_update_packages` configuration parameter that holds a sequence of MatchSpec strings, in addition to the `pinned_packages` configuration parameter. Currently, the default value contains the packages `ca-certificates`, `certifi`, and `openssl`. When manipulating configuration with the `conda config` command, use of the `--system` and `--env` flags will be especially helpful here. For example,

```
conda config --add aggressive_update_packages defaults::pyopenssl --system
```

would ensure that, system-wide, solves on all environments enforce using the latest version of `pyopenssl` from

the `defaults` channel.

```
conda config --add pinned_packages python=2.7 --env
```

would lock all solves for the current active environment to python versions matching `2.7.*`.

- **other configuration improvements**: In addition to `conda config --describe`, which shows detailed descriptions and default values for all available configuration parameters, we have a new `conda config --write-default` command. This new command simply writes the contents of `conda config --describe` to a condarc file, which is a great starter template. Without additional arguments, the command will write to the `.condarc` file in the user's home directory. The command also works with the `--system`, `--env`, and `--file` flags to write the contents to alternate locations.

  Conda exposes a tremendous amount of flexibility via configuration. For more information, The Conda Configuration Engine for Power Users blog post is a good resource.

Deprecations/Breaking Changes

- the conda 'root' environment is now generally referred to as the 'base' environment
- Conda 4.4 now warns when available information about per-path sha256 sums and file sizes do not match the recorded information. The warning is scheduled to be an error in conda 4.5. Behavior is configurable via the `safety_checks` configuration parameter.
- remove support for with_features_depends (#5191)
- resolve #5468 remove --alt-hint from CLI API (#5469)
- resolve #5834 change default value of 'allow_softlinks' from True to False (#5835)
- resolve #5842 add deprecation warnings for 'conda env upload' and 'conda env attach' (#5843)

API

- Add Solver from conda.core.solver with three methods to conda.api (4.4.0rc1) (#5838)

Improvements

- constrained, optional dependencies (#4982)
- conda shell function (#5044, #5141, #5162, #5169, #5182, #5210, #5482)
- resolve #5160 conda xontrib plugin (#5157)
- resolve #1543 add support and tests for --no-deps and --only-deps (#5265)
- resolve #988 allow channel name to be part of the package name spec (#5365, #5791)
- resolve #5530 add ability for users to choose to post unexpected errors to core maintainers (#5531, #5571, #5585)
- Solver, UI, History, and Other (#5546, #5583, #5740)
- improve 'conda search' to leverage new MatchSpec query language (#5597)
- filter out unwritable package caches from conda clean command (#4620)
- envs_manager, requested spec history, declarative solve, and private env tests (#4676, #5114, #5094, #5145, #5492)
- make python entry point format match pip entry points (#5010)
- resolve #5113 clean up CLI imports to improve process startup time (#4799)
- resolve #5121 add features/track_features support for MatchSpec (#5054)

- resolve #4671 hold verify backoff count in transaction context (#5122)
- resolve #5078 record package metadata after tarball extraction (#5148)
- resolve #3580 support stacking environments (#5159)
- resolve #3763, #4378 allow pip requirements.txt syntax in environment files (#3969)
- resolve #5147 add 'config files' to conda info (#5269)
- use --format=json to parse list of pip packages (#5205)
- resolve #1427 remove startswith '.' environment name constraint (#5284)
- link packages from extracted tarballs when tarball is gone (#5289)
- resolve #2511 accept config information from stdin (#5309)
- resolve #4302 add ability to set map parameters with conda config (#5310)
- resolve #5256 enable conda config --get for all primitive parameters (#5312)
- resolve #1992 add short flag -C for --use-index-cache (#5314)
- resolve #2173 add --quiet option to conda clean (#5313)
- resolve #5358 conda should exec to subcommands, not subprocess (#5359)
- resolve #5411 add 'conda config --write-default' (#5412)
- resolve #5081 make pinned packages optional dependencies (#5414)
- resolve #5430 eliminate current deprecation warnings (#5422)
- resolve #5470 make stdout/stderr capture in python_api customizable (#5471)
- logging simplifications/improvements (#5547, #5578)
- update license information (#5568)
- enable threadpool use for repodata collection by default (#5546, #5587)
- conda info now raises PackagesNotFoundError (#5655)
- index building optimizations (#5776)
- fix #5811 change safety_checks default to 'warn' for conda 4.4 (4.4.0rc1) (#5824)
- add constrained dependencies to conda's own recipe (4.4.0rc1) (#5823)
- clean up parser imports (4.4.0rc2) (#5844)
- resolve #5983 add --download-only flag to create, install, and update (4.4.0rc2) (#5988)
- add ca-certificates and certifi to aggressive_update_packages default (4.4.0rc2) (#5994)
- use environments.txt to list all known environments (4.4.0rc2) (#6313)
- resolve #5417 ensure unlink order is correctly sorted (4.4.0) (#6364)
- resolve #5370 index is only prefix and cache in --offline mode (4.4.0) (#6371)
- reduce redundant sys call during file copying (4.4.0rc3) (#6421)
- enable aggressive_update_packages (4.4.0rc3) (#6392)
- default conda.sh to dash if otherwise can't detect (4.4.0rc3) (#6414)
- canonicalize package names when comparing with pip (4.4.0rc3) (#6438)
- add target prefix override configuration parameter (4.4.0rc3) (#6413)

- resolve #6194 warn when conda is outdated (4.4.0rc3) (#6370)
- add information to displayed error report (4.4.0rc3) (#6437)
- csh wrapper (4.4.0) (#6463)
- resolve #5158 --override-channels (4.4.0) (#6467)
- fish update for conda 4.4 (4.4.0) (#6475, #6502)
- skip an unnecessary environments.txt rewrite (4.4.0) (#6495)

Bug Fixes

- fix some conda-build compatibility issues (#5089)
- resolve #5123 export toposort (#5124)
- fix #5132 signal handler can only be used in main thread (#5133)
- fix orphaned --clobber parser arg (#5188)
- fix #3814 don't remove directory that's not a conda environment (#5204)
- fix #4468 _license stack trace (#5206)
- fix #4987 conda update --all no longer displays full list of packages (#5228)
- fix #3489 don't error on remove --all if environment doesn't exist (#5231)
- fix #1509 bash doesn't need full path for pre/post link/unlink scripts on unix (#5252)
- fix #462 add regression test (#5286)
- fix #5288 confirmation prompt doesn't accept no (#5291)
- fix #1713 'conda package -w' is case dependent on Windows (#5308)
- fix #5371 try falling back to pip's vendored requests if no requests available (#5372)
- fix #5356 skip root logger configuration (#5380)
- fix #5466 scrambled URL of non-alias channel with token (#5467)
- fix #5444 environment.yml file not found (#5475)
- fix #3200 use proper unbound checks in bash code and test (#5476)
- invalidate PrefixData cache on rm_rf for conda-build (#5491, #5499)
- fix exception when generating JSON output (#5628)
- fix target prefix determination (#5642)
- use proxy to avoid segfaults (#5716)
- fix #5790 incorrect activation message (4.4.0rc1) (#5820)
- fix #5808 assertion error when loading package cache (4.4.0rc1) (#5815)
- fix #5809 _pip_install_via_requirements got an unexpected keyword argument 'prune' (4.4.0rc1) (#5814)
- fix #5811 change safety_checks default to 'warn' for conda 4.4 (4.4.0rc1) (#5824)
- fix #5825 --json output format (4.4.0rc1) (#5831)
- fix force_reinstall for case when packages aren't actually installed (4.4.0rc1) (#5836)
- fix #5680 empty pip subsection error in environment.yml (4.4.0rc2) (#6275)
- fix #5852 bad tokens from history crash conda installs (4.4.0rc2) (#6076)

- fix #5827 no error message on invalid command (4.4.0rc2) (#6352)
- fix exception handler for 'conda activate' (4.4.0rc2) (#6365)
- fix #6173 double prompt immediately after conda 4.4 upgrade (4.4.0rc2) (#6351)
- fix #6181 keep existing pythons pinned to minor version (4.4.0rc2) (#6363)
- fix #6201 incorrect subdir shown for conda search when package not found (4.4.0rc2) (#6367)
- fix #6045 help message and zsh shift (4.4.0rc3) (#6368)
- fix noarch python package resintall (4.4.0rc3) (#6394)
- fix #6366 shell activation message (4.4.0rc3) (#6369)
- fix #6429 AttributeError on 'conda remove' (4.4.0rc3) (#6434)
- fix #6449 problems with 'conda info --envs' (#6451)
- add debug exception for #6430 (4.4.0rc3) (#6435)
- fix #6441 NotImplementedError on 'conda list' (4.4.0rc3) (#6442)
- fix #6445 scale back directory activation in PWD (4.4.0rc3) (#6447)
- fix #6283 no-deps for conda update case (4.4.0rc3) (#6448)
- fix #6419 set PS1 in python code (4.4.0rc3) (#6446)
- fix #6466 sp_dir doesn't exist (#6470)
- fix #6350 --update-all removes too many packages (4.4.0) (#6491)
- fix #6057 unlink-link order for python noarch packages on windows 4.4.x (4.4.0) (#6494)

Non-User-Facing Changes

- eliminate index modification in Resolve init (#4333)
- new MatchSpec implementation (#4158, #5517)
- update conda.recipe for 4.4 (#5086)
- resolve #5118 organization and cleanup for 4.4 release (#5115)
- remove unused disk space check instructions (#5167)
- localfs adapter tests (#5181)
- extra config command tests (#5185)
- add coverage for confirm (#5203)
- clean up FileNotFoundError and DirectoryNotFoundError (#5237)
- add assertion that a path only has a single hard link before rewriting prefixes (#5305)
- remove pycrypto as requirement on windows (#5326)
- import cleanup, dead code removal, coverage improvements, and other housekeeping (#5472, #5474, #5480)
- rename CondaFileNotFoundError to PathNotFoundError (#5521)
- work toward repodata API (#5267)
- rename PackageNotFoundError to PackagesNotFoundError and fix message formatting (#5602)
- update conda 4.4 bld.bat windows recipe (#5573)
- remove last remnant of CondaEnvRuntimeError (#5643)

- fix typo (4.4.0rc2) (#6043)
- replace Travis-CI with CircleCI (4.4.0rc2) (#6345)
- key-value features (#5645); reverted in 4.4.0rc2 (#6347, #6492)
- resolve #6431 always add env_vars to info_dict (4.4.0rc3) (#6436)
- move shell inside conda directory (4.4.0) (#6479)
- remove dead code (4.4.0) (#6489)

4.5.134 4.3.34 (2018-02-09)

Bug Fixes

- fix #6833 improve feature mismatch metric (#6853)

4.5.135 4.3.33 (2018-01-24)

Bug Fixes

- fix #6718 broken 'conda install --rev' (#6719)
- fix #6765 adjust the feature score assigned to packages not installed (#6766)

4.5.136 4.3.32 (2018-01-10)

Improvements

- resolve #6711 fall back to copy/unlink for EINVAL, EXDEV rename failures (#6712)

Bug Fixes

- fix #6057 unlink-link order for python noarch packages on windows (#6277)
- fix #6509 custom_channels incorrect in 'conda config --show' (#6510)

4.5.137 4.3.31 (2017-12-15)

Improvements

- add delete_trash to conda_env create (#6299)

Bug Fixes

- fix #6023 assertion error for temp file (#6154)
- fix #6220 --no-builds flag for 'conda env export' (#6221)
- fix #6271 timestamp prioritization results in undesirable race-condition (#6279)

Non-User-Facing Changes

- fix two failing integration tests after anaconda.org API change (#6182)
- resolve #6243 mark root as not writable when sys.prefix is not a conda environment (#6274)
- add timing instrumentation (#6458)

4.5.138 4.3.30 (2017-10-17)

Improvements

- address #6056 add additional proxy variables to 'conda info --all' (#6083)

Bug Fixes

- address #6164 move add_defaults_to_specs after augment_specs (#6172)
- fix #6057 add additional detail for message 'cannot link source that does not exist' (#6082)
- fix #6084 setting default_channels from CLI raises NotImplementedError (#6085)

4.5.139 4.3.29 (2017-10-09)

Bug Fixes

- fix #6096 coerce to millisecond timestamps (#6131)

4.5.140 4.3.28 (2017-10-06)

Bug Fixes

- fix #5854 remove imports of pkg_resources (#5991)
- fix millisecond timestamps (#6001)

4.5.141 4.3.27 (2017-09-18)

Bug Fixes

- fix #5980 always delete_prefix_from_linked_data in rm_rf (#5982)

4.5.142 4.3.26 (2017-09-15)

Deprecations/Breaking Changes

- resolve #5922 prioritize channels within multi-channels (#5923)
- add https://repo.continuum.io/pkgs/main to defaults multi-channel (#5931)

Improvements

- add a channel priority minimization pass to solver logic (#5859)
- invoke cmd.exe with /D for pre/post link/unlink scripts (#5926)
- add boto3 use to s3 adapter (#5949)

Bug Fixes

- always remove linked prefix entry with rm_rf (#5846)
- resolve #5920 bump repodata pickle version (#5921)
- fix msys2 activate and deactivate (#5950)

4.5.143 4.3.25 (2017-08-16)

Deprecations/Breaking Changes

- resolve #5834 change default value of 'allow_softlinks' from True to False (#5839)

Improvements

- add non-admin check to optionally disable non-privileged operation (#5724)
- add extra warning message to always_softlink configuration option (#5826)

Bug Fixes

- fix #5763 channel url string splitting error (#5764)
- fix regex for repodata _mod and _etag (#5795)
- fix uncaught OSError for missing device (#5830)

4.5.144 4.3.24 (2017-07-31)

Bug Fixes

- fix #5708 package priority sort order (#5733)

4.5.145 4.3.23 (2017-07-21)

Improvements

- resolve #5391 PackageNotFound and NoPackagesFoundError clean up (#5506)

Bug Fixes

- fix #5525 too many Nones in CondaHttpError (#5526)
- fix #5508 assertion failure after test file not cleaned up (#5533)
- fix #5523 catch OSError when home directory doesn't exist (#5549)
- fix #5574 traceback formatting (#5580)
- fix #5554 logger configuration levels (#5555)
- fix #5649 create_default_packages configuration (#5703)

4.5.146 4.3.22 (2017-06-12)

Improvements

- resolve #5428 clean up cli import in conda 4.3.x (#5429)
- resolve #5302 add warning when creating environment with space in path (#5477)
- for ftp connections, ignore host IP from PASV as it is often wrong (#5489)
- expose common race condition exceptions in exports for conda-build (#5498)

Bug Fixes

- fix #5451 conda clean --json bug (#5452)
- fix #5400 confusing deactivate message (#5473)
- fix #5459 custom subdir channel parsing (#5478)
- fix #5483 problem with setuptools / pkg_resources import (#5496)

4.5.147 4.3.21 (2017-05-25)

Bug Fixes

- fix #5420 conda-env update error (#5421)
- fix #5425 is admin on win int not callable (#5426)

4.5.148 4.3.20 (2017-05-23)

Improvements

- resolve #5217 skip user confirm in python_api, force always_yes (#5404)

Bug Fixes

- fix #5367 conda info always shows 'unknown' for admin indicator on Windows (#5368)
- fix #5248 drop plan description information that might not always be accurate (#5373)
- fix #5378 duplicate log messages (#5379)
- fix #5298 record has 'build', not 'build_string' (#5382)
- fix #5384 silence logging info to avoid interfering with JSON output (#5393)
- fix #5356 skip root/conda logger init for cli.python_api (#5405)

Non-User-Facing Changes

- avoid persistent state after channel priority test (#5392)
- resolve #5402 add regression test for #5384 (#5403)
- clean up inner function definition inside for loop (#5406)

4.5.149 4.3.19 (2017-05-18)

Improvements

- resolve #3689 better error messaging for missing anaconda-client (#5276)
- resolve #4795 conda env export lacks -p flag (#5275)
- resolve #5315 add alias verify_ssl for ssl_verify (#5316)
- resolve #3399 add netrc existence/location to 'conda info' (#5333)
- resolve #3810 add --prefix to conda env update (#5335)

Bug Fixes

- fix #5272 conda env export ugliness under python2 (#5273)
- fix #4596 warning message from pip on conda env export (#5274)
- fix #4986 --yes not functioning for conda clean (#5311)
- fix #5329 unicode errors on Windows (#5328, #5357)
- fix sys_prefix_unfollowed for Python 3 (#5334)
- fix #5341 --json flag with conda-env (#5342)
- fix 5321 ensure variable PROMPT is set in activate.bat (#5351)

Non-User-Facing Changes

- test conda 4.3 with requests 2.14.2 (#5281)
- remove pycrypto as requirement on windows (#5325)
- fix typo avaialble -> available (#5345)
- fix test failures related to menuinst update (#5344, #5362)

4.5.150 4.3.18 (2017-05-09)

Improvements

- resolve #4224 warn when pysocks isn't installed (#5226)
- resolve #5229 add --insecure flag to skip ssl verification (#5230)
- resolve #4151 add admin indicator to conda info on windows (#5241)

Bug Fixes

- fix #5152 conda info spacing (#5166)
- fix --use-index-cache actually hitting the index cache (#5134)
- backport LinkPathAction verify from 4.4 (#5171)
- fix #5184 stack trace on invalid map configuration parameter (#5186)
- fix #5189 stack trace on invalid sequence config param (#5192)
- add support for the linux-aarch64 platform (#5190)
- fix repodata fetch with the `--offline` flag (#5146)
- fix #1773 conda remove spell checking (#5176)
- fix #3470 reduce excessive error messages (#5195)
- fix #1597 make extra sure --dry-run doesn't take any actions (#5201)
- fix #3470 extra newlines around exceptions (#5200)
- fix #5214 install messages for 'nothing_to_do' case (#5216)
- fix #598 stack trace for condarc write permission denied (#5232)
- fix #4960 extra information when exception can't be displayed (#5236)
- fix #4974 no matching dist in linked data for prefix (#5239)

- fix #5258 give correct element types for conda config --describe (#5259)
- fix #4911 separate shutil.copy2 into copy and copystat (#5261)

Non-User-Facing Changes

- resolve #5138 add test of rm_rf of symlinked files (#4373)
- resolve #4516 add extra trace-level logging (#5249, #5250)
- add tests for --update-deps flag (#5264)

4.5.151 4.3.17 (2017-04-24)

Improvements

- fall back to copy if hardlink fails (#5002)
- add timestamp metadata for tiebreaking conda-build 3 hashed packages (#5018)
- resolve #5034 add subdirs configuration parameter (#5030)
- resolve #5081 make pinned packages optional/constrained dependencies (#5088)
- resolve #5108 improve behavior and add tests for spaces in paths (#4786)

Bug Fixes

- quote prefix paths for locations with spaces (#5009)
- remove binstar logger configuration overrides (#4989)
- fix #4969 error in DirectoryNotFoundError (#4990)
- fix #4998 pinned string format (#5011)
- fix #5039 collecting main_info shouldn't fail on requests import (#5090)
- fix #5055 improve bad token message for anaconda.org (#5091)
- fix #5033 only re-register valid signal handlers (#5092)
- fix #5028 imports in main_list (#5093)
- fix #5073 allow client_ssl_cert{_key} to be of type None (#5096)
- fix #4671 backoff for package validate race condition (#5098)
- fix #5022 gnu_get_libc_version => linux_get_libc_version (#5099)
- fix #4849 package name match bug (#5103)
- fixes #5102 allow proxy_servers to be of type None (#5107)
- fix #5111 incorrect typify for str + NoneType (#5112)

Non-User-Facing Changes

- resolve #5012 remove CondaRuntimeError and RuntimeError (#4818)
- full audit ensuring relative import paths within project (#5090)
- resolve #5116 refactor conda/cli/activate.py to help menuinst (#4406)

4.5.152 4.3.16 (2017-03-30)

Improvements

- additions to configuration SEARCH_PATH to improve consistency (#4966)
- add 'conda config --describe' and extra config documentation (#4913)
- enable packaging pinning in condarc using pinned_packages config parameter as beta feature (#4921, #4964)

Bug Fixes

- fix #4914 handle directory creation on top of file paths (#4922)
- fix #3982 issue with CONDA_ENV and using powerline (#4925)
- fix #2611 update instructions on how to source conda.fish (#4924)
- fix #4860 missing information on package not found error (#4935)
- fix #4944 command not found error error (#4963)

4.5.153 4.3.15 (2017-03-20)

Improvements

- allow pkgs_dirs to be configured using `conda config` (#4895)

Bug Fixes

- remove incorrect elision of delete_prefix_from_linked_data() (#4814)
- fix envs_dirs order for read-only root prefix (#4821)
- fix break-point in conda clean (#4801)
- fix long shebangs when creating entry points (#4828)
- fix spelling and typos (#4868, #4869)
- fix #4840 TypeError reduce() of empty sequence with no initial value (#4843)
- fix zos subdir (#4875)
- fix exceptions triggered during activate (#4873)

4.5.154 4.3.14 (2017-03-03)

Improvements

- use cPickle in place of pickle for repodata (#4717)
- ignore pyc compile failure (#4719)
- use conda.exe for windows entry point executable (#4716, #4720)
- localize use of conda_signal_handler (#4730)
- add skip_safety_checks configuration parameter (#4767)
- never symlink executables using ORIGIN (#4625)
- set activate.bat codepage to CP_ACP (#4558)

Bug Fixes

- fix #4777 package cache initialization speed (#4778)
- fix #4703 menuinst PathNotFoundException (#4709)
- ignore permissions error if user_site can't be read (#4710)
- fix #4694 don't import requests directly in models (#4711)
- fix #4715 include resources directory in recipe (#4716)
- fix CondaHttpError for URLs that contain '%' (#4769)
- bug fixes for preferred envs (#4678)
- fix #4745 check for info/index.json with package is_extracted (#4776)
- make sure url gets included in CondaHTTPError (#4779)
- fix #4757 map-type configs set to None (#4774)
- fix #4788 partial package extraction (#4789)

Non-User-Facing Changes

- test coverage improvement (#4607)
- CI configuration improvements (#4713, #4773, #4775)
- allow sha256 to be None (#4759)
- add cache_fn_url to exports (#4729)
- add unicode paths for PY3 integration tests (#4760)
- additional unit tests (#4728, #4783)
- fix conda-build compatibility and tests (#4785)

4.5.155 4.3.13 (2017-02-17)

Improvements

- resolve #4636 environment variable expansion for pkgs_dirs (#4637)
- link, symlink, islink, and readlink for Windows (#4652, #4661)
- add extra information to CondaHTTPError (#4638, #4672)

Bug Fixes

- maximize requested builds after feature determination (#4647)
- fix #4649 incorrect assert statement concerning package cache directory (#4651)
- multi-user mode bug fixes (#4663)

Non-User-Facing Changes

- path_actions unit tests (#4654)
- remove dead code (#4369, #4655, #4660)
- separate repodata logic from index into a new core/repodata.py module (#4669)

4.5.156 4.3.12 (2017-02-14)

Improvements

- prepare conda for uploading to pypi (#4619)
- better general http error message (#4627)
- disable old python noarch warning (#4576)

Bug Fixes

- fix UnicodeDecodeError for ensure_text_type (#4585)
- fix determination of if file path is writable (#4604)
- fix #4592 BufferError cannot close exported pointers exist (#4628)
- fix run_script current working directory (#4629)
- fix pkgs_dirs permissions regression (#4626)

Non-User-Facing Changes

- fixes for tests when conda-bld directory doesn't exist (#4606)
- use requirements.txt and Makefile for travis-ci setup (#4600, #4633)
- remove hasattr use from compat functions (#4634)

4.5.157 4.3.11 (2017-02-09)

Bug Fixes

- fix attribute error in add_defaults_to_specs (#4577)

4.5.158 4.3.10 (2017-02-07)

Improvements

- remove .json from pickle path (#4498)
- improve empty repodata noarch warning and error messages (#4499)
- don't add python and lua as default specs for private envs (#4529, #4533)
- let default_python be None (#4547, #4550)

Bug Fixes

- fix #4513 null pointer exception for channel without noarch (#4518)
- fix ssl_verify set type (#4517)
- fix bug for windows multiuser (#4524)
- fix clone with noarch python packages (#4535)
- fix ipv6 for python 2.7 on Windows (#4554)

Non-User-Facing Changes

- separate integration tests with a marker (#4532)

4.5.159 4.3.9 (2017-01-31)

Improvements

- improve repodata caching for performance (#4478, #4488)
- expand scope of packages included by bad_installed (#4402)
- silence pre-link warning for old noarch (#4451)
- add configuration to optionally require noarch repodata (#4450)
- improve conda subprocessing (#4447)
- respect info/link.json (#4482)

Bug Fixes

- fix #4398 'hard' was used for link type at one point (#4409)
- fixed "No matches for wildcard '$activate_d/*.fish'" warning (#4415)
- print correct activate/deactivate message for fish shell (#4423)
- fix 'Dist' object has no attribute 'fn' (#4424)
- fix noarch generic and add additional integration test (#4431)
- fix #4425 unknown encoding (#4433)

Non-User-Facing Changes

- fail CI on conda-build fail (#4405)
- run doctests (#4414)
- make index record mutable again (#4461)
- additional test for conda list --json (#4480)

4.5.160 4.3.8 (2017-01-23)

Bug Fixes

- fix #4309 ignore EXDEV error for directory renames (#4392)
- fix #4393 by force-renaming certain backup files if the path already exists (#4397)

4.5.161 4.3.7 (2017-01-20)

Bug Fixes

- actually revert json output for leaky plan (#4383)
- fix not raising on pre/post-link error (#4382)
- fix find_commands and find_executable for symlinks (#4387)

4.5.162 4.3.6 (2017-01-18)

Bug Fixes

- fix 'Uncaught backoff with errno 41' warning on windows (#4366)
- revert json output for leaky plan (#4349)
- audit os.environ setting (#4360)
- fix #4324 using old dist string instead of dist object (#4361)
- fix #4351 infinite recursion via code in #4120 (#4370)
- fix #4368 conda -h (#4367)
- workaround for symlink race conditions on activate (#4346)

4.5.163 4.3.5 (2017-01-17)

Improvements

- add exception message for corrupt repodata (#4315)

Bug Fixes

- fix package not being found in cache after download (#4297)
- fix logic for Content-Length mismatch (#4311, #4326)
- use unicode_escape after etag regex instead of utf-8 (#4325)
- fix #4323 central condarc file being ignored (#4327)
- fix #4316 a bug in deactivate (#4316)
- pass target_prefix as env_prefix regardless of is_unlink (#4332)
- pass positional argument 'context' to BasicClobberError (#4335)

Non-User-Facing Changes

- additional package pinning tests (#4317)

4.5.164 4.3.4 (2017-01-13)

Improvements

- vendor url parsing from urllib3 (#4289)

Bug Fixes

- fix some bugs in windows multi-user support (#4277)
- fix problems with channels of type (#4290)
- include aliases for first command-line argument (#4279)
- fix for multi-line FTP status codes (#4276)

Non-User-Facing Changes

- make arch in IndexRecord a StringField instead of EnumField
- improve conda-build compatibility (#4266)

4.5.165 4.3.3 (2017-01-10)

Improvements

- respect Cache-Control max-age header for repodata (#4220)
- add 'local_repodata_ttl' configurability (#4240)
- remove questionable "nothing to install" logic (#4237)
- relax channel noarch requirement for 4.3; warn now, raise in future feature release (#4238)
- add additional info to setup.py warning message (#4258)

Bug Fixes

- remove features properly (#4236)
- do not use `IFS` to find activate/deactivate scripts to source (#4239)
- fix #4235 print message to stderr (#4241)
- fix relative path to python in activate.bat (#4242)
- fix args.channel references (#4245, #4246)
- ensure cache_fn_url right pad (#4255)
- fix #4256 subprocess calls must have env wrapped in str (#4259)

4.5.166 4.3.2 (2017-01-06)

Deprecations/Breaking Changes

- Further refine conda channels specification. To verify if the url of a channel represents a valid conda channel, we check that `noarch/repodata.json` and/or `noarch/repodata.json.bz2` exist, even if empty. (#3739)

Improvements

- add new 'path_conflict' and 'clobber' configuration options (#4119)
- separate fetch/extract pass for explicit URLs (#4125)
- update conda homepage to conda.io (#4180)

Bug Fixes

- fix pre/post unlink/link scripts (#4113)
- fix package version regex and bug in create_link (#4132)
- fix history tracking (#4143)
- fix index creation order (#4131)
- fix #4152 conda env export failure (#4175)
- fix #3779 channel UNC path encoding errors on windows (#4190)
- fix progress bar (#4191)

- use context.channels instead of args.channel (#4199)
- don't use local cached repodata for file:// urls (#4209)

Non-User-Facing Changes

- xfail anaconda token test if local token is found (#4124)
- fix open-ended test failures relating to python 3.6 release (#4145)
- extend timebomb for test_multi_channel_export (#4169)
- don't unlink dists that aren't in the index (#4130)
- add python 3.6 and new conda-build test targets (#4194)

4.5.167 4.3.1 (2016-12-19)

Improvements

- additional pre-transaction validation (#4090)
- export FileMode enum for conda-build (#4080)
- memoize disk permissions tests (#4091)
- local caching of repodata without remote server calls; new 'repodata_timeout_secs' configuration parameter (#4094)
- performance tuning (#4104)
- add additional fields to dist object serialization (#4102)

Bug Fixes

- fix a noarch install bug on windows (#4071)
- fix a spec mismatch that resulted in python versions getting mixed during packaging (#4079)
- fix rollback linked record (#4092)
- fix #4097 keep split in PREFIX_PLACEHOLDER (#4100)

4.5.168 4.3.0 (2016-12-14) Safety

New Features

- **Unlink and Link Packages in a Single Transaction**: In the past, conda hasn't always been safe and defensive with its disk-mutating actions. It has gleefully clobbered existing files, and mid-operation failures leave environments completely broken. In some of the most severe examples, conda can appear to "uninstall itself." With this release, the unlinking and linking of packages for an executed command is done in a single transaction. If a failure occurs for any reason while conda is mutating files on disk, the environment will be returned its previous state. While we've implemented some pre-transaction checks (verifying package integrity for example), it's impossible to anticipate every failure mechanism. In some circumstances, OS file permissions cannot be fully known until an operation is attempted and fails. And conda itself is not without bugs. Moving forward, unforeseeable failures won't be catastrophic. (#3833, #4030)
- **Progressive Fetch and Extract Transactions**: Like package unlinking and linking, the download and extract phases of package handling have also been given transaction-like behavior. The distinction is the rollback on error is limited to a single package. Rather than rolling back the download and extract operation for all packages, the single-package rollback prevents the need for having to re-download every package if an error is encountered. (#4021, #4030)

- **Generic- and Python-Type Noarch/Universal Packages**: Along with conda-build 2.1.0, a noarch/universal type for python packages is officially supported. These are much like universal python wheels. Files in a python noarch package are linked into a prefix just like any other conda package, with the following additional features

  1. conda maps the `site-packages` directory to the correct location for the python version in the environment,
  2. conda maps the python-scripts directory to either $PREFIX/bin$PREFIX/Scripts depending on platform,
  3. conda creates the python entry points specified in the conda-build recipe, and
  4. conda compiles pyc files at install time when prefix write permissions are guaranteed.

  Python noarch packages must be "fully universal." They cannot have OS- or python version-specific dependencies. They cannot have OS- or python version-specific "scripts" files. If these features are needed, traditional conda packages must be used. (#3712)

- **Multi-User Package Caches**: While the on-disk package cache structure has been preserved, the core logic implementing package cache handling has had a complete overhaul. Writable and read-only package caches are fully supported. (#4021)

- **Python API Module**: An oft requested feature is the ability to use conda as a python library, obviating the need to "shell out" to another python process. Conda 4.3 includes a `conda.cli.python_api` module that facilitates this use case. While we maintain the user-facing command-line interface, conda commands can be executed in-process. There is also a `conda.exports` module to facilitate longer-term usage of conda as a library across conda conda releases. However, conda's python code *is* considered internal and private, subject to change at any time across releases. At the moment, conda will not install itself into environments other than its original install environment. (#4028)

- **Remove All Locks**: Locking has never been fully effective in conda, and it often created a false sense of security. In this release, multi-user package cache support has been implemented for improved safety by hard-linking packages in read-only caches to the user's primary user package cache. Still, users are cautioned that undefined behavior can result when conda is running in multiple process and operating on the same package caches and/or environments. (#3862)

Deprecations/Breaking Changes

- Conda will refuse to clobber existing files that are not within the unlink instructions of the transaction. At the risk of being user-hostile, it's a step forward for conda. We do anticipate some growing pains. For example, conda will not clobber packages that have been installed with pip (or any other package manager). In other instances, conda packages that contain overlapping file paths but are from different package families will not install at the same time. The `--force` command line flag is the escape hatch. Using `--force` will let your operation proceed, but also makes clear that you want conda to do something it considers unsafe.

- Conda signed packages have been removed in 4.3. Vulnerabilities existed. An illusion of security is worse than not having the feature at all. We will be incorporating The Update Framework into conda in a future feature release. (#4064)

- Conda 4.4 will drop support for older versions of conda-build.

Improvements

- create a new "trace" log level enabled by `-v -v -v` or `-vvv` (#3833)
- allow conda to be installed with pip, but only when used as a library/dependency (#4028)
- the 'r' channel is now part of defaults (#3677)
- private environment support for conda (#3988)
- support v1 info/paths.json file (#3927, #3943)

- support v1 info/package_metadata.json (#4030)
- improved solver hint detection, simplified filtering (#3597)
- cache VersionOrder objects to improve performance (#3596)
- fix documentation and typos (#3526, #3572, #3627)
- add multikey configuration validation (#3432)
- some Fish autocompletions (#2519)
- reduce priority for packages removed from the index (#3703)
- add user-agent, uid, gid to conda info (#3671)
- add conda.exports module (#3429)
- make http timeouts configurable (#3832)
- add a pkgs_dirs config parameter (#3691)
- add an 'always_softlink' option (#3870, #3876)
- pre-checks for diskspace, etc for fetch and extract #(4007)
- address #3879 don't print activate message when quiet config is enabled (#3886)
- add zos-z subdir (#4060)
- add elapsed time to HTTP errors (#3942)

Bug Fixes

- account for the Windows Python 2.7 os.environ unicode aversion (#3363)
- fix link field in record object (#3424)
- anaconda api token bug fix; additional tests (#3673)
- fix #3667 unicode literals and unicode decode (#3682)
- add conda-env entrypoint (#3743)
- fix #3807 json dump on conda config --show --json (#3811)
- fix #3801 location of temporary hard links of index.json (#3813)
- fix invalid yml example (#3849)
- add arm platforms back to subdirs (#3852)
- fix #3771 better error message for assertion errors (#3802)
- fix #3999 spaces in shebang replacement (#4008)
- config --show-sources shouldn't show force by default (#3891)
- fix #3881 don't install conda-env in clones of root (#3899)
- conda-build dist compatibility (#3909)

Non-User-Facing Changes

- remove unnecessary eval (#3428)
- remove dead install_tar function (#3641)
- apply PEP-8 to conda-env (#3653)

- refactor dist into an object (#3616)
- vendor appdirs; remove conda's dependency on anaconda-client import (#3675)
- revert boto patch from #2380 (#3676)
- move and update ROOT_NO_RM (#3697)
- integration tests for conda clean (#3695, #3699)
- disable coverage on s3 and ftp requests adapters (#3696, #3701)
- github repo hygiene (#3705, #3706)
- major install refactor (#3712)
- remove test timebombs (#4012)
- LinkType refactor (#3882)
- move CrossPlatformStLink and make available as export (#3887)
- make Record immutable (#3965)
- project housekeeping (#3994, #4065)
- context-dependent setup.py files (#4057)

4.5.169 4.2.17 (unreleased)

Improvements

- silence pre-link warning for old noarch 4.2.x backport (#4453)

Bug Fixes

- remove incorrect elision of delete_prefix_from_linked_data() (#4813)
- fix CB #1825 context clobbering (#4867)
- fix #5101 api->conda regex substitution for Anaconda API channels (#5100)

Non-User-Facing Changes

- build 4.2.x against conda-build 2.1.2 and enforce passing (#4462)

4.5.170 4.2.16 (2017-01-20)

Improvements

- vendor url parsing from urllib3 (#4289)
- workaround for symlink race conditions on activate (#4346)

Bug Fixes

- do not replace \ with / in file:// URLs on Windows (#4269)
- include aliases for first command-line argument (#4279)
- fix for multi-line FTP status codes (#4276)
- fix errors with unknown type channels (#4291)
- change sys.exit to raise UpgradeError when info/files not found (#4388)

Non-User-Facing Changes

- start using doctests in test runs and coverage (#4304)
- additional package pinning tests (#4312)

4.5.171 4.2.15 (2017-01-10)

Improvements

- use 'post' instead of 'dev' for commits according to PEP-440 (#4234)
- do not use IFS to find activate/deactivate scripts to source (#4243)
- fix relative path to python in activate.bat (#4244)

Bug Fixes

- replace sed with python for activate and deactivate #4257

4.5.172 4.2.14 (2017-01-07)

Improvements

- use install.rm_rf for TemporaryDirectory cleanup (#3425)
- improve handling of local dependency information (#2107)
- add default channels to exports for Windows and Unix (#4103)
- make subdir configurable (#4178)

Bug Fixes

- fix conda/install.py single-file behavior (#3854)
- fix the api->conda substitution (#3456)
- fix silent directory removal (#3730)
- fix location of temporary hard links of index.json (#3975)
- fix potential errors in multi-channel export and offline clone (#3995)
- fix auxlib/packaging, git hashes are not limited to 7 characters (#4189)
- fix compatibility with requests >=2.12, add pyopenssl as dependency (#4059)
- fix #3287 activate in 4.1-4.2.3 clobbers non-conda PATH changes (#4211)

Non-User-Facing Changes

- fix open-ended test failures relating to python 3.6 release (#4166)
- allow args passed to cli.main() (#4193, #4200, #4201)
- test against python 3.6 (#4197)

4.5.173 4.2.13 (2016-11-22)

Deprecations/Breaking Changes

- show warning message for pre-link scripts (#3727)
- error and exit for install of packages that require conda minimum version 4.3 (#3726)

Improvements

- double/extend http timeouts (#3831)
- let descriptive http errors cover more http exceptions (#3834)
- backport some conda-build configuration (#3875)

Bug Fixes

- fix conda/install.py single-file behavior (#3854)
- fix the api->conda substitution (#3456)
- fix silent directory removal (#3730)
- fix #3910 null check for is_url (#3931)

Non-User-Facing Changes

- flake8 E116, E121, & E123 enabled (#3883)

4.5.174 4.2.12 (2016-11-02)

Bug Fixes

- fix #3732, #3471, #3744 CONDA_BLD_PATH (#3747)
- fix #3717 allow no-name channels (#3748)
- fix #3738 move conda-env to ruamel_yaml (#3740)
- fix conda-env entry point (#3745 via #3743)
- fix again #3664 trash emptying (#3746)

4.5.175 4.2.11 (2016-10-23)

Improvements

- only try once for windows trash removal (#3698)

Bug Fixes

- fix anaconda api token bug (#3674)
- fix #3646 FileMode enum comparison (#3683)
- fix #3517 conda install --mkdir (#3684)
- fix #3560 hack anaconda token coverup on conda info (#3686)
- fix #3469 alias envs_path to envs_dirs (#3685)

4.5.176 4.2.10 (2016-10-18)

Improvements

- add json output for `conda info -s` (#3588)
- ignore certain binary prefixes on windows (#3539)
- allow conda config files to have .yaml extensions or 'condarc' anywhere in filename (#3633)

Bug Fixes

- fix conda-build's handle_proxy_407 import (#3666)
- fix #3442, #3459, #3481, #3531, #3548 multiple networking and auth issues (#3550)
- add back linux-ppc64le subdir support (#3584)
- fix #3600 ensure links are removed when unlinking (#3625)
- fix #3602 search channels by platform (#3629)
- fix duplicated packages when updating environment (#3563)
- fix #3590 exception when parsing invalid yaml (#3593 via #3634)
- fix #3655 a string decoding error (#3656)

Non-User-Facing Changes

- backport conda.exports module to 4.2.x (#3654)
- travis-ci OSX fix (#3615 via #3657)

4.5.177 4.2.9 (2016-09-27)

Bug Fixes

- fix #3536 conda-env messaging to stdout with --json flag (#3537)
- fix #3525 writing to sys.stdout with --json flag for post-link scripts (#3538)
- fix #3492 make NULL falsey with python 3 (#3524)

4.5.178 4.2.8 (2016-09-26)

Improvements

- add "error" key back to json error output (#3523)

Bug Fixes

- fix #3453 conda fails with create_default_packages (#3454)
- fix #3455 --dry-run fails (#3457)
- dial down error messages for rm_rf (#3522)
- fix #3467 AttributeError encountered for map config parameter validation (#3521)

4.5.179 4.2.7 (2016-09-16)

Deprecations/Breaking Changes

- revert to 4.1.x behavior of `conda list --export` (#3450, #3451)

Bug Fixes

- don't add binstar token if it's given in the channel spec (#3427, #3440, #3444)
- fix #3433 failure to remove broken symlinks (#3436)

Non-User-Facing Changes

- use install.rm_rf for TemporaryDirectory cleanup (#3425)

4.5.180 4.2.6 (2016-09-14)

Improvements

- add support for client TLS certificates (#3419)
- address #3267 allow migration of channel_alias (#3410)
- conda-env version matches conda version (#3422)

Bug Fixes

- fix #3409 unsatisfiable dependency error message (#3412)
- fix #3408 quiet rm_rf (#3413)
- fix #3407 padding error messaging (#3416)
- account for the Windows Python 2.7 os.environ unicode aversion (#3363 via #3420)

4.5.181 4.2.5 (2016-09-08)

Deprecations/Breaking Changes

- partially revert #3041 giving conda config --add previous --prepend behavior (#3364 via #3370)
- partially revert #2760 adding back conda package command (#3398)

Improvements

- order output of conda config --show; make --json friendly (#3384 via #3386)
- clean the pid based lock on exception (#3325)
- improve file removal on all platforms (#3280 via #3396)

Bug Fixes

- fix #3332 allow download urls with :: in them (#3335)
- fix always_yes and not-set argparse args overriding other sources (#3374)
- fix ftp fetch timeout (#3392)
- fix #3307 add try/except block for touch lock (#3326)
- fix CONDA_CHANNELS environment variable splitting (#3390)
- fix #3378 CONDA_FORCE_32BIT environment variable (#3391)
- make conda info channel urls actually give urls (#3397)
- fix cio_test compatibility (#3395 via #3400)

4.5.182 4.2.4 (2016-08-18)

Bug Fixes

- fix #3277 conda list package order (#3278)
- fix channel priority issue with duplicated channels (#3283)

- fix local channel channels; add full conda-build unit tests (#3281)
- fix conda install with no package specified (#3284)
- fix #3253 exporting and importing conda environments (#3286)
- fix priority messaging on conda config --get (#3304)
- fix conda list --export; additional integration tests (#3291)
- fix conda update --all idempotence; add integration tests for channel priority (#3306)

Non-User-Facing Changes

- additional conda-env integration tests (#3288)

4.5.183 4.2.3 (2016-08-11)

Improvements

- added zsh and zsh.exe to Windows shells (#3257)

Bug Fixes

- allow conda to downgrade itself (#3273)
- fix breaking changes to conda-build from 4.2.2 (#3265)
- fix empty environment issues with conda and conda-env (#3269)

Non-User-Facing Changes

- add integration tests for conda-env (#3270)
- add more conda-build smoke tests (#3274)

4.5.184 4.2.2 (2016-08-09)

Improvements

- enable binary prefix replacement on windows (#3262)
- add `--verbose` command line flag (#3237)
- improve logging and exception detail (#3237, #3252)
- do not remove empty environment without asking; raise an error when a named environment can't be found (#3222)

Bug Fixes

- fix #3226 user condarc not available on Windows (#3228)
- fix some bugs in conda config --show* (#3212)
- fix conda-build local channel bug (#3202)
- remove subprocess exiting message (#3245)
- fix comment parsing and channels in conda-env environment.yml (#3258, #3259)
- fix context error with conda-env (#3232)
- fix #3182 conda install silently skipping failed linking (#3184)

4.5.185 4.2.1 (2016-08-01)

Improvements

- improve an error message that can happen during conda install --revision (#3181)
- use clean sys.exit with user choice 'No' (#3196)

Bug Fixes

- critical fix for 4.2.0 error when no git is on PATH (#3193)
- revert #3171 lock cleaning on exit pending further refinement
- patches for conda-build compatibility with 4.2 (#3187)
- fix a bug in --show-sources output that ignored aliased parameter names (#3189)

Non-User-Facing Changes

- move scripts in bin to shell directory (#3186)

4.5.186 4.2.0 (2016-07-28) Configuration

New Features

- **New Configuration Engine**: Configuration and "operating context" are the foundation of conda's functionality. Conda now has the ability to pull configuration information from a multitude of on-disk locations, including `.d` directories and a `.condarc` file *within* a conda environment), along with full `CONDA_` environment variable support. Helpful validation errors are given for improperly-specified configuration. Full documentation updates pending. (#2537, #3160, #3178)
- **New Exception Handling Engine**: Previous releases followed a pattern of premature exiting (with hard calls to `sys.exit()` when exceptional circumstances were encountered. This release replaces over 100 `sys.exit` calls with python exceptions. For conda developers, this will result in tests that are easier to write. For developers using conda, this is a first step on a long path toward conda being directly importable. For conda users, this will eventually result in more helpful and descriptive errors messages. (#2899, #2993, #3016, #3152, #3045)
- **Empty Environments**: Conda can now create "empty" environments when no initial packages are specified, alleviating a common source of confusion. (#3072, #3174)
- **Conda in Private Env**: Conda can now be configured to live within its own private environment. While it's not yet default behavior, this represents a first step toward separating the `root` environment into a "conda private" environment and a "user default" environment. (#3068)
- **Regex Version Specification**: Regular expressions are now valid version specifiers. For example, `^1\.[5-8]\.1$|2.2`. (#2933)

Deprecations/Breaking Changes

- remove conda init (#2759)
- remove conda package and conda bundle (#2760)
- deprecate conda-env repo; pull into conda proper (#2950, #2952, #2954, #3157, #3163, #3170)
- force use of ruamel_yaml (#2762)
- implement conda config --prepend; change behavior of --add to --append (#3041)
- exit on link error instead of logging it (#2639)

Improvements

- improve locking (#2962, #2989, #3048, #3075)
- clean up requests usage for fetching packages (#2755)
- remove excess output from conda --help (#2872)
- remove os.remove in update_prefix (#3006)
- better error behavior if conda is spec'd for a non-root environment (#2956)
- scale back try_write function on unix (#3076)

Bug Fixes

- remove psutil requirement, fixes annoying error message (#3135, #3183)
- fix #3124 add threading lock to memoize (#3134)
- fix a failure with multi-threaded repodata downloads (#3078)
- fix windows file url (#3139)
- address #2800, error with environment.yml and non-default channels (#3164)

Non-User-Facing Changes

- project structure enhancement (#2929, #3132, #3133, #3136)
- clean up channel handling with new channel model (#3130, #3151)
- add Anaconda Cloud / Binstar auth handler (#3142)
- remove dead code (#2761, #2969)
- code refactoring and additional tests (#3052, #3020)
- remove auxlib from project root (#2931)
- vendor auxlib 0.0.40 (#2932, #2943, #3131)
- vendor toolz 0.8.0 (#2994)
- move progressbar to vendor directory (#2951)
- fix conda.recipe for new quirks with conda-build (#2959)
- move captured function to common module (#3083)
- rename CHANGELOG to md (#3087)

4.5.187 4.1.13 (unreleased)

- improve handling of local dependency information, #2107
- show warning message for pre-link scripts, #3727
- error and exit for install of packages that require conda minimum version 4.3, #3726
- fix conda/install.py single-file behavior, #3854
- fix open-ended test failures relating to python 3.6 release, #4167
- fix #3287 activate in 4.1-4.2.3 clobbers non-conda PATH changes, #4211
- fix relative path to python in activate.bat, #4244

4.5.188 4.1.12 (2016-09-08)

- fix #2837 “File exists” in symlinked path with parallel activations, #3210
- fix prune option when installing packages, #3354
- change check for placeholder to be more friendly to long PATH, #3349

4.5.189 4.1.11 (2016-07-26)

- fix PS1 backup in activate script, #3135 via #3155
- correct resolution for ‘handle failures in binstar_client more generally’, #3156

4.5.190 4.1.10 (2016-07-25)

- ignore symlink failure because of read-only file system, #3055
- backport shortcut tests, #3064
- fix #2979 redefinition of $SHELL variable, #3081
- fix #3060 --clone root --copy exception, #3080

4.5.191 4.1.9 (2016-07-20)

- fix #3104, add global BINSTAR_TOKEN_PAT
- handle failures in binstar_client more generally

4.5.192 4.1.8 (2016-07-12)

- fix #3004 UNAUTHORIZED for url (null binstar token), #3008
- fix overwrite existing redirect shortcuts when symlinking envs, #3025
- partially revert no default shortcuts, #3032, #3047

4.5.193 4.0.11 2016-07-09

- allow auto_update_conda from sysrc, #3015 via #3021

4.5.194 4.1.7 (2016-07-07)

- add msys2 channel to defaults on Windows, #2999
- fix #2939 channel_alias issues; improve offline enforcement, #2964
- fix #2970, #2974 improve handling of file:// URLs inside channel, #2976

4.5.195 4.1.6 (2016-07-01)

- slow down exp backoff from 1 ms to 100 ms factor, #2944
- set max time on exp_backoff to ~6.5 sec,#2955
- fix #2914 add/subtract from PATH; kill folder output text, #2917
- normalize use of get_index behavior across clone/explicit, #2937
- wrap root prefix check with normcase, #2938

4.5.196 4.1.5 (2016-06-29)

- more conservative auto updates of conda #2900
- fix some permissions errors with more aggressive use of move_path_to_trash, #2882
- fix #2891 error if allow_other_channels setting is used, #2896
- fix #2886, #2907 installing a tarball directly from the package cache, #2908
- fix #2681, #2778 reverting #2320 lock behavior changes, #2915

4.5.197 4.0.10 (2016-06-29)

- fix #2846 revert the use of UNC paths; shorten trash filenames, #2859 via #2878
- fix some permissions errors with more aggressive use of move_path_to_trash, #2882 via #2894

4.5.198 4.1.4 (2016-06-27)

- fix #2846 revert the use of UNC paths; shorten trash filenames, #2859
- fix exp backoff on Windows, #2860
- fix #2845 URL for local file repos, #2862
- fix #2764 restore full path var on win; create to CONDA_PREFIX env var, #2848
- fix #2754 improve listing pip installed packages, #2873
- change root prefix detection to avoid clobbering root activate scripts, #2880
- address #2841 add lowest and highest priority indication to channel config output, #2875
- add SYMLINK_CONDA to planned instructions, #2861
- use CONDA_PREFIX, not CONDA_DEFAULT_ENV for activate.d, #2856
- call scripts with redirect on win; more error checking to activate, #2852

4.5.199 4.1.3 (2016-06-23)

- ensure conda-env auto update, along with conda, #2772
- make yaml booleans behave how everyone expects them to, #2784
- use accept-encoding for repodata; prefer repodata.json to repodata.json.bz2, #2821
- additional integration and regression tests, #2757, #2774, #2787
- add offline mode to printed info; use offline flag when grabbing channels, #2813
- show conda-env version in conda info, #2819
- adjust channel priority superseded list, #2820
- support epoch ! characters in command line specs, #2832
- accept old default names and new ones when canonicalizing channel URLs #2839
- push PATH, PS1 manipulation into shell scripts, #2796
- fix #2765 broken source activate without arguments, #2806
- fix standalone execution of install.py, #2756
- fix #2810 activating conda environment broken with git bash on Windows, #2795

- fix #2805, #2781 handle both file-based channels and explicit file-based URLs, #2812
- fix #2746 conda create --clone of root, #2838
- fix #2668, #2699 shell recursion with activate #2831

4.5.200 4.1.2 (2016-06-17)

- improve messaging for "downgrades" due to channel priority, #2718
- support conda config channel append/prepend, handle duplicates, #2730
- remove --shortcuts option to internal CLI code, #2723
- fix an issue concerning space characters in paths in activate.bat, #2740
- fix #2732 restore yes/no/on/off for booleans on the command line, #2734
- fix #2642 tarball install on Windows, #2729
- fix #2687, #2697 WindowsError when creating environments on Windows, #2717
- fix #2710 link instruction in conda create causes TypeError, #2715
- revert #2514, #2695, disabling of .netrc files, #2736
- revert #2281 printing progress bar to terminal, #2707

4.5.201 4.1.1 (2016-06-16)

- add auto_update_conda config parameter, #2686
- fix #2669 conda config --add channels can leave out defaults, #2670
- fix #2703 ignore activate symlink error if links already exist, #2705
- fix #2693 install duplicate packages with older version of Anaconda, #2701
- fix #2677 respect HTTP_PROXY, #2695
- fix #2680 broken fish integration, #2685, #2694
- fix an issue with conda never exiting, #2689
- fix #2688 explicit file installs, #2708
- fix #2700 conda list UnicodeDecodeError, #2706

4.5.202 4.0.9 (2016-06-15)

- add auto_update_conda config parameter, #2686

4.5.203 4.1.0 (2016-06-14) Channel Priority

- clean up activate and deactivate scripts, moving back to conda repo, #1727, #2265, #2291, #2473, #2501, #2484
- replace pyyaml with ruamel_yaml, #2283, #2321
- better handling of channel collisions, #2323, #2369 #2402, #2428
- improve listing of pip packages with conda list, #2275
- re-license progressbar under BSD 3-clause, #2334
- reduce the amount of extraneous info in hints, #2261
- add --shortcuts option to install shortcuts on windows, #2623

- skip binary replacement on windows, #2630
- don't show channel urls by default in conda list, #2282
- package resolution and solver tweaks, #2443, #2475, #2480
- improved version & build matching, #2442, #2488
- print progress to the terminal rather than stdout, #2281
- verify version specs given on command line are valid, #2246
- fix for try_write function in case of odd permissions, #2301
- fix a conda search --spec error, #2343
- update User-Agent for conda connections, #2347
- remove some dead code paths, #2338, #2374
- fixes a thread safety issue with http requests, #2377, #2383
- manage BeeGFS hard-links non-POSIX configuration, #2355
- prevent version downgrades during removes, #2394
- fix conda info --json, #2445
- truncate shebangs over 127 characters using /usr/bin/env, #2479
- extract packages to a temporary directory then rename, #2425, #2483
- fix help in install, #2460
- fix re-install bug when sha1 differs, #2507
- fix a bug with file deletion, #2499
- disable .netrc files, #2514
- dont fetch index on remove --all, #2553
- allow track_features to be a string *or* a list in .condarc, #2541
- fix #2415 infinite recursion in invalid_chains, #2566
- allow channel_alias to be different than binstar, #2564

4.5.204 4.0.8 (2016-06-03)

- fix a potential problem with moving files to trash, #2587

4.5.205 4.0.7 (2016-05-26)

- workaround for boto bug, #2380

4.5.206 4.0.6 (2016-05-11)

- log "custom" versions as updates rather than downgrades, #2290
- fixes a TypeError exception that can occur on install/update, #2331
- fixes an error on Windows removing files with long path names, #2452

4.5.207 4.0.5 (2016-03-16)

- improved help documentation for install, update, and remove, #2262
- fixes #2229 and #2250 related to conda update errors on Windows, #2251
- fixes #2258 conda list for pip packages on Windows, #2264

4.5.208 4.0.4 (2016-03-10)

- revert #2217 closing request sessions, #2233

4.5.209 4.0.3 (2016-03-10)

- adds a `conda clean --all` feature, #2211
- solver performance improvements, #2209
- fixes conda list for pip packages on windows, #2216
- quiets some logging for package downloads under python 3, #2217
- more urls for `conda list --explicit`, #1855
- prefer more "latest builds" for more packages, #2227
- fixes a bug with dependency resolution and features, #2226

4.5.210 4.0.2 (2016-03-08)

- fixes track_features in ~/.condarc being a list, see also #2203
- fixes incorrect path in lock file error #2195
- fixes issues with cloning environments, #2193, #2194
- fixes a strange interaction between features and versions, #2206
- fixes a bug in low-level SAT clause generation creating a preference for older versions, #2199

4.5.211 4.0.1 (2016-03-07)

- fixes an install issue caused by md5 checksum mismatches, #2183
- remove auxlib build dependency, #2188

4.5.212 4.0.0 (2016-03-04) Solver

- The solver has been retooled significantly. Performance should be improved in most circumstances, and a number of issues involving feature conflicts should be resolved.
- `conda update <package>` now handles dependencies properly according to the setting of the "update_deps" configuration: –update-deps: conda will also update any dependencies as needed to install the latest version of the requested packages. The minimal set of changes required to achieve this is sought. –no-update-deps: conda will update the packages *only* to the extent that no updates to the dependencies are required The previous behavior, which would update the packages without regard to their dependencies, could result in a broken configuration, and has been removed.
- Conda finally has an official logo.
- Fix `conda clean --packages` on Windows, #1944
- Conda sub-commands now support dashes in names, #1840

4.5.213 3.19.4 (unreleased)

- improve handling of local dependency information, #2107
- use install.rm_rf for TemporaryDirectory cleanup, #3425
- fix the api->conda substitution, #3456
- error and exit for install of packages that require conda minimum version 4.3, #3726
- show warning message for pre-link scripts, #3727
- fix silent directory removal, #3730
- fix conda/install.py single-file behavior, #3854

4.5.214 3.19.3 (2016-02-19)

- fix critical issue, see #2106

4.5.215 3.19.2 (2016-02-19)

- add basic activate/deactivate, conda activate/deactivate/ls for fish, see #545
- remove error when CONDA_FORCE_32BIT is set on 32-bit systems, #1985
- suppress help text for --unknown option, #2051
- fix issue with conda create --clone post-link scripts, #2007
- fix a permissions issue on windows, #2083

4.5.216 3.19.1 (2016-02-01)

- resolve.py: properly escape periods in version numbers, #1926
- support for pinning Lua by default, #1934
- remove hard-coded test URLs, a module cio_test is now expected when CIO_TEST is set

4.5.217 3.19.0 (2015-12-17)

- OpenBSD 5.x support, #1891
- improve install CLI to make Miniconda -f work, #1905

4.5.218 3.18.9 (2015-12-10)

- allow chaining default_channels (only applies to "system" condarc), from from CLI, #1886
- improve default for --show-channel-urls in conda list, #1900

4.5.219 3.18.8 (2015-12-03)

- always attempt to delete files in rm_rf, #1864

4.5.220 3.18.7 (2015-12-02)

- simplify call to menuinst.install()
- add menuinst as dependency on Windows
- add ROOT_PREFIX to post-link (and pre_unlink) environment

4.5.221 3.18.6 (2015-11-19)

- improve conda clean when user lacks permissions, #1807
- make show_channel_urls default to True, #1771
- cleaner write tests, #1735
- fix documentation, #1709
- improve conda clean when directories don't exist, #1808

4.5.222 3.18.5 (2015-11-11)

- fix bad menuinst exception handling, #1798
- add workaround for unresolved dependencies on Windows

4.5.223 3.18.4 (2015-11-09)

- allow explicit file to contain MD5 hashsums
- add --md5 option to "conda list --explicit"
- stop infinite recursion during certain resolve operations, #1749
- add dependencies even if strictness == 3, #1766

4.5.224 3.18.3 (2015-10-15)

- added a pruning step for more efficient solves, #1702
- disallow conda-env to be installed into non-root environment
- improve error output for bad command input, #1706
- pass env name and setup cmd to menuinst, #1699

4.5.225 3.18.2 (2015-10-12)

- add "conda list --explicit" which contains the URLs of all conda packages to be installed, and can used with the install/create --file option, #1688
- fix a potential issue in conda clean
- avoid issues with LookupErrors when updating Python in the root environment on Windows
- don't fetch the index from the network with conda remove
- when installing conda packages directly, "conda install .tar.bz2", unlink any installed package with that name (not just the installed one)
- allow menu items to be installed in non-root env, #1692

4.5.226 3.18.1 (2015-09-28)

- fix: removed reference to win_ignore_root in plan module

4.5.227 3.18.0 (2015-09-28)

- allow Python to be updated in root environment on Windows, #1657
- add defaults to specs after getting pinned specs (allows to pin a different version of Python than what is installed)
- show what older versions are in the solutions in the resolve debug log
- fix some issues with Python 3.5
- respect --no-deps when installing from .tar or .tar.bz2
- avoid infinite recursion with NoPackagesFound and conda update --all --file
- fix conda update --file
- toposort: Added special case to remove 'pip' dependency from 'python'
- show dotlog messages during hint generation with --debug
- disable the max_only heuristic during hint generation
- new version comparison algorithm, which consistently compares any version string, and better handles version strings using things like alpha, beta, rc, post, and dev. This should remove any inconsistent version comparison that would lead to conda installing an incorrect version.
- use the trash in rm_rf, meaning more things will get the benefit of the trash system on Windows
- add the ability to pass the --file argument multiple times
- add conda upgrade alias for conda update
- add update_dependencies condarc option and --update-deps/–no-update-deps command line flags
- allow specs with conda update --all
- add --show-channel-urls and --no-show-channel-urls command line options
- add always_copy condarc option
- conda clean properly handles multiple envs directories. This breaks backwards compatibility with some of the --json output. Some of the old –json keys are kept for backwards compatibility.

4.5.228 3.17.0 (2015-09-11)

- add windows_forward_slashes option to walk_prefix(), see #1513
- add ability to set CONDA_FORCE_32BIT environment variable, it should should only be used when running conda-build, #1555
- add config option to makes the python dependency on pip optional, #1577
- fix an UnboundLocalError
- print note about pinned specs in no packages found error
- allow wildcards in AND-connected version specs
- print pinned specs to the debug log
- fix conda create --clone with create_default_packages
- give a better error when a proxy isn't found for a given scheme
- enable running 'conda run' in offline mode
- fix issue where hardlinked cache contents were being overwritten
- correctly skip packages whose dependencies can't be found with conda update --all

- use clearer terminology in -m help text.
- use splitlines to break up multiple lines throughout the codebase
- fix AttributeError with SSLError

4.5.229 3.16.0 (2015-08-10)

- rename binstar -> anaconda, see #1458
- fix --use-local when the conda-bld directory doesn't exist
- fixed --offline option when using "conda create --clone", see #1487
- don't mask recursion depth errors
- add conda search --reverse-dependency
- check whether hardlinking is available before linking when using "python install.py --link" directly, see #1490
- don't exit nonzero when installing a package with no dependencies
- check which features are installed in an environment via track_features, not features
- set the verify flag directly on CondaSession (fixes conda skeleton not respecting the ssl_verify option)

4.5.230 3.15.1 (2015-07-23)

- fix conda with older versions of argcomplete
- restore the --force-pscheck option as a no-op for backwards compatibility

4.5.231 3.15.0 (2015-07-22)

- sort the output of conda info package correctly
- enable tab completion of conda command extensions using argcomplete. Command extensions that import conda should use conda.cli.conda_argparse.ArgumentParser instead of argparse.ArgumentParser. Otherwise, they should enable argcomplete completion manually.
- allow psutil and pycosat to be updated in the root environment on Windows
- remove all mentions of pscheck. The --force-pscheck flag has been removed.
- added support for S3 channels
- fix color issues from pip in conda list on Windows
- add support for other machine types on Linux, in particular ppc64le
- add non_x86_linux_machines set to config module
- allow ssl_verify to accept strings in addition to boolean values in condarc
- enable --set to work with both boolean and string values

4.5.232 3.14.1 (2015-06-29)

- make use of Crypto.Signature.PKCS1_PSS module, see #1388
- note when features are being used in the unsatisfiable hint

4.5.233 3.14.0 (2015-06-16)

- add ability to verify signed packages, see #1343 (and conda-build #430)
- fix issue when trying to add 'pip' dependency to old python packages
- provide option "conda info --unsafe-channels" for getting unobscured channel list, #1374

4.5.234 3.13.0 (2015-06-04)

- avoid the Windows file lock by moving files to a trash directory, #1133
- handle env dirs not existing in the Environments completer
- rename binstar.org -> anaconda.org, see #1348
- speed up 'source activate' by ~40%

4.5.235 3.12.0 (2015-05-05)

- correctly allow conda to update itself
- print which file leads to the "unable to remove file" error on Windows
- add support for the no_proxy environment variable, #1171
- add a much faster hint generation for unsatisfiable packages, which is now always enabled (previously it would not run if there were more than ten specs). The new hint only gives one set of conflicting packages, rather than all sets, so multiple passes may be necessary to fix such issues
- conda extensions that import conda should use conda.cli.conda_argparser.ArgumentParser instead of argparse.ArgumentParser to conform to the conda help guidelines (e.g., all help messages should be capitalized with periods, and the options should be preceded by "Options:" for the sake of help2man).
- add confirmation dialog to conda remove. Fixes conda remove --dry-run.

4.5.236 3.11.0 (2015-04-22)

- fix issue where forced update on Windows could cause a package to break
- remove detection of running processes that might conflict
- deprecate --force-pscheck (now a no-op argument)
- make conda search --outdated --names-only work, fixes #1252
- handle the history file not having read or write permissions better
- make multiple package resolutions warning easier to read
- add --full-name to conda list
- improvements to command help

4.5.237 3.10.1 (2015-04-06)

- fix logic in @memoized for unhashable args
- restored json cache of repodata, see #1249
- hide binstar tokens in conda info --json
- handle CIO_TEST='2 '
- always find the solution with minimal number of packages, even if there are many solutions

- allow comments at the end of the line in requirement files
- don't update the progressbar until after the item is finished running
- add conda/ to HTTP header User-Agent string

4.5.238 3.10.0 (2015-03-12)

- change default repo urls to be https
- add --offline to conda search
- add --names-only and --full-name to conda search
- add tab completion for packages to conda search

4.5.239 3.9.1 (2015-02-24)

- pscheck: check for processes in the current environment, see #1157
- don't write to the history file if nothing has changed, see #1148
- conda update --all installs packages without version restrictions (except for Python), see #1138
- conda update --all ignores the anaconda metapackage, see #1138
- use forward slashes for file urls on Windows
- don't symlink conda in the root environment from activate
- use the correct package name in the progress bar info
- use json progress bars for unsatisfiable dependencies hints
- don't let requests decode gz files when downloaded

4.5.240 3.9.0 (2015-02-16)

- remove (de)activation scripts from conda, those are now in conda-env
- pip is now always added as a Python dependency
- allow conda to be installed into environments which start with _
- add argcomplete tab completion for environments with the -n flag, and for package names with install, update, create, and remove

4.5.241 3.8.4 (2015-02-03)

- copy (de)activate scripts from conda-env
- Add noarch (sub) directory support

4.5.242 3.8.3 (2015-01-28)

- simplified how ROOT_PREFIX is obtained in (de)activate

4.5.243 3.8.2 (2015-01-27)

- add conda clean --source-cache to clean the conda build source caches
- add missing quotes in (de)activate.bat, fixes problem in Windows when conda is installed into a directory with spaces

- fix conda install --copy

4.5.244 3.8.1 (2015-01-23)

- add missing utf-8 decoding, fixes Python 3 bug when icondata to json file

4.5.245 3.8.0 (2015-01-22)

- move active script into conda-env, which is now a new dependency
- load the channel urls in the correct order when using concurrent.futures
- add optional 'icondata' key to json files in conda-meta directory, which contain the base64 encoded png file or the icon
- remove a debug print statement

4.5.246 3.7.4 (2014-12-18)

- add --offline option to install, create, update and remove commands, and also add ability to set "offline: True" in condarc file
- add conda uninstall as alias for conda remove
- add conda info --root
- add conda.pip module
- fix CONDARC pointing to non-existing file, closes issue #961
- make update -f work if the package is already up-to-date
- fix possible TypeError when printing an error message
- link packages in topologically sorted order (so that pre-link scripts can assume that the dependencies are installed)
- add --copy flag to install
- prevent the progressbar from crashing conda when fetching in some situations

4.5.247 3.7.3 (2014-11-05)

- conda install from a local conda package (or a tar fill which contains conda packages), will now also install the dependencies listed by the installed packages.
- add SOURCE_DIR environment variable in pre-link subprocess
- record all created environments in ~/.conda/environments.txt

4.5.248 3.7.2 (2014-10-31)

- only show the binstar install message once
- print the fetching repodata dot after the repodata is fetched
- write the install and remove specs to the history file
- add '-y' as an alias to '–yes'
- the `--file` option to conda config now defaults to os.environ.get('CONDARC')
- some improvements to documentation (–help output)
- add user_rc_path and sys_rc_path to conda info --json

- cache the proxy username and password
- avoid warning about conda in pscheck
- make ~/.conda/envs the first user envs dir

4.5.249 3.7.1 (2014-10-07)

- improve error message for forgetting to use source with activate and deactivate, see issue #601
- don't allow to remove the current environment, see issue #639
- don't fail if binstar_client can't be imported for other reasons, see issue #925
- allow spaces to be contained in conda run
- only show the conda install binstar hint if binstar is not installed
- conda info package_spec now gives detailed info on packages. conda info path has been removed, as it is duplicated by conda package -w path.

4.5.250 3.7.0 (2014-09-19)

- faster algorithm for --alt-hint
- don't allow channel_alias with allow_other_channels: false if it is set in the system .condarc
- don't show long "no packages found" error with update --all
- automatically add the Binstar token to urls when the binstar client is installed and logged in
- carefully avoid showing the binstar token or writing it to a file
- be more careful in conda config about keys that are the wrong type
- don't expect directories starting with conda- to be commands
- no longer recommend to run conda init after pip installing conda. A pip installed conda will now work without being initialized to create and manage other environments
- the rm function on Windows now works around access denied errors
- fix channel urls now showing with conda list with show_channel_urls set to true

4.5.251 3.6.4 (2014-09-08)

- fix removing packages that aren't in the channels any more
- Pretties output for --alt-hint

4.5.252 3.6.3 (2014-09-04)

- skip packages that can't be found with update --all
- add --use-local to search and remove
- allow --use-local to be used along with -c (–channels) and –override-channels. --override-channels now requires either -c or –use-local
- allow paths in has_prefix to be quoted, to allow for spaces in paths on Windows
- retain Unix style path separators for prefixes in has_prefix on Windows (if the placeholder path uses /, replace it with a path that uses /, not)
- fix bug in --use-local due to API changes in conda-build

- include user site directories in conda info -s
- make binary has_prefix replacement work with spaces after the prefix
- make binary has_prefix replacement replace multiple occurrences of the placeholder in the same null-terminated string
- don't show packages from other platforms as installed or cached in conda search
- be more careful about not warning about conda itself in pscheck
- Use a progress bar for the unsatisfiable packages hint generation
- Don't use TemporaryFile in try_write, as it is too slow when it fails
- Ignore InsecureRequestWarning when ssl_verify is False
- conda remove removes features tracked by removed packages in track_features

4.5.253 3.6.2 (2014-08-20)

- add --use-index-cache to conda remove
- fix a bug where features (like mkl) would be selected incorrectly
- use concurrent.future.ThreadPool to fetch package metadata asynchronously in Python 3.
- do the retries in rm_rf on every platform
- use a higher cutoff for package name misspellings
- allow changing default channels in "system" .condarc

4.5.254 3.6.1 (2014-08-13)

- add retries to download in fetch module
- improved error messages for missing packages
- more robust rm_rf on Windows
- print multiline help for subcommands correctly

4.5.255 3.6.0 (2014-08-11)

- correctly check if a package can be hard-linked if it isn't extracted yet
- change how the package plan is printed to better show what is new, updated, and downgraded
- use suggest_normalized_version in the resolve module. Now versions like 1.0alpha that are not directly recognized by verlib's NormalizedVersion are supported better
- conda run command, to run apps and commands from packages
- more complete --json API. Every conda command should fully support --json output now.
- show the conda_build and requests versions in conda info
- include packages from setup.py develop in conda list (with use_pip)
- raise a warning instead of dying when the history file is invalid
- use urllib.quote on the proxy password
- make conda search --outdated --canonical work
- pin the Python version during conda init

- fix some metadata that is written for Python during conda init
- allow comments in a pinned file
- allow installing and updating menuinst on Windows
- allow conda create with both --file and listed packages
- better handling of some nonexistent packages
- fix command line flags in conda package
- fix a bug in the ftp adapter

4.5.256 3.5.5 (2014-06-10)

- remove another instance pycosat version detection, which fails on Windows, see issue #761

4.5.257 3.5.4 (2014-06-10)

- remove pycosat version detection, which fails on Windows, see issue #761

4.5.258 3.5.3 (2014-06-09)

- fix conda update to correctly not install packages that are already up-to-date
- always fail with connection error in download
- the package resolution is now much faster and uses less memory
- add ssl_verify option in condarc to allow ignoring SSL certificate verification, see issue #737

4.5.259 3.5.2 (2014-05-27)

- fix bug in activate.bat and deactivate.bat on Windows

4.5.260 3.5.1 (2014-05-26)

- fix proxy support - conda now prompts for proxy username and password again
- fix activate.bat on Windows with spaces in the path
- update optional psutil dependency was updated to psutil 2.0 or higher

4.5.261 3.5.0 (2014-05-15)

- replace use of urllib2 with requests. requests is now a hard dependency of conda.
- add ability to only allow system-wise specified channels
- hide binstar from output of conda info

4.5.262 3.4.3 (2014-05-05)

- allow prefix replacement in binary files, see issue #710
- check if creating hard link is possible and otherwise copy, during install
- allow circular dependencies

4.5.263 3.4.2 (2014-04-21)

- conda clean --lock: skip directories that don't exist, fixes #648
- fixed empty history file causing crash, issue #644
- remove timezone information from history file, fixes issue #651
- fix PackagesNotFound error for missing recursive dependencies
- change the default for adding cache from the local package cache - known is now the default and the option to use index metadata from the local package cache is --unknown
- add --alt-hint as a method to get an alternate form of a hint for unsatisfiable packages
- add conda package --ls-files to list files in a package
- add ability to pin specs in an environment. To pin a spec, add a file called pinned to the environment's conda-meta directory with the specs to pin. Pinned specs are always kept installed, unless the --no-pin flag is used.
- fix keyboard interrupting of external commands. Now keyboard interrupting conda build correctly removes the lock file
- add no_link ability to conda, see issue #678

4.5.264 3.4.1 (2014-04-07)

- always use a pkgs cache directory associated with an envs directory, even when using -p option with an arbitrary a prefix which is not inside an envs dir
- add setting of PYTHONHOME to conda info --system
- skip packages with bad metadata

4.5.265 3.4.0 (2014-04-02)

- added revision history to each environment:
  - conda list --revisions
  - conda install --revision
  - log is stored in conda-meta/history
- allow parsing pip-style requirement files with --file option and in command line arguments, e.g. conda install 'numpy>=1.7', issue #624
- fix error message for --file option when file does not exist
- allow DEFAULTS in CONDA_ENVS_PATH, which expands to the defaults settings, including the condarc file
- don't install a package with a feature (like mkl) unless it is specifically requested (i.e., that feature is already enabled in that environment)
- add ability to show channel URLs when displaying what is going to be downloaded by setting "show_channel_urls: True" in condarc
- fix the --quiet option
- skip packages that have dependencies that can't be found

4.5.266 3.3.2 (2014-03-24)

- fix the --file option
- check install arguments before fetching metadata
- fix a printing glitch with the progress bars
- give a better error message for conda clean with no arguments
- don't include unknown packages when searching another platform

4.5.267 3.3.1 (2014-03-19)

- Fix setting of PS1 in activate.
- Add conda update --all.
- Allow setting CONDARC=' ' to use no condarc.
- Add conda clean --packages.
- Don't include bin/conda, bin/activate, or bin/deactivate in conda package.

4.5.268 3.3.0 (2014-03-18)

- allow new package specification, i.e. ==, >=, >, <=, <, != separated by ',' for example: >=2.3,<3.0
- add ability to disable self update of conda, by setting "self_update: False" in .condarc
- Try installing packages using the old way of just installing the maximum versions of things first. This provides a major speedup of solving the package specifications in the cases where this scheme works.
- Don't include python=3.3 in the specs automatically for the Python 3 version of conda. This allows you to do "conda create -n env package" for a package that only has a Python 2 version without specifying "python=2". This change has no effect in Python 2.
- Automatically put symlinks to conda, activate, and deactivate in each environment on Unix.
- On Unix, activate and deactivate now remove the root environment from the PATH. This should prevent "bleed through" issues with commands not installed in the activated environment but that are installed in the root environment. If you have "setup.py develop" installed conda on Unix, you should run this command again, as the activate and deactivate scripts have changed.
- Begin work to support Python 3.4.
- Fix a bug in version comparison
- Fix usage of sys.stdout and sys.stderr in environments like pythonw on Windows where they are nonstandard file descriptors.

4.5.269 3.2.1 (2014-03-12)

- fix installing packages with irrational versions
- fix installation in the api
- use a logging handler to print the dots

4.5.270 3.2.0 (2014-03-11)

- print dots to the screen for progress
- move logic functions from resolve to logic module

4.5.271 3.2.0a1 (2014-03-07)

- conda now uses pseudo-boolean constraints in the SAT solver. This allows it to search for all versions at once, rather than only the latest (issue #491).
- Conda contains a brand new logic submodule for converting pseudo-boolean constraints into SAT clauses.

4.5.272 3.1.1 (2014-03-07)

- check if directory exists, fixed issue #591

4.5.273 3.1.0 (2014-03-07)

- local packages in cache are now added to the index, this may be disabled by using the --known option, which only makes conda use index metadata from the known remote channels
- add --use-index-cache option to enable using cache of channel index files
- fix ownership of files when installing as root on Linux
- conda search: add '.' symbol for extracted (cached) packages

4.5.274 3.0.6 (2014-02-20)

- fix 'conda update' taking build number into account

4.5.275 3.0.5 (2014-02-17)

- allow packages from create_default_packages to be overridden from the command line
- fixed typo install.py, issue #566
- try to prevent accidentally installing into a non-root conda environment

4.5.276 3.0.4 (2014-02-14)

- conda update: don't try to update packages that are already up-to-date

4.5.277 3.0.3 (2014-02-06)

- improve the speed of clean --lock
- some fixes to conda config
- more tests added
- choose the first solution rather than the last when there are more than one, since this is more likely to be the one you want.

4.5.278 3.0.2 (2014-02-03)

- fix detection of prefix being writable

4.5.279 3.0.1 (2014-01-31)

- bug: not having track_features in condarc now uses default again
- improved test suite
- remove numpy version being treated special in plan module
- if the post-link.(bat|sh) fails, don't treat it as though it installed, i.e. it is not added to conda-meta
- fix activate if CONDA_DEFAULT_ENV is invalid
- fix conda config --get to work with list keys again
- print the total download size
- fix a bug that was preventing conda from working in Python 3
- add ability to run pre-link script, issue #548

4.5.280 3.0.0 (2014-01-24)

- removed build, convert, index, and skeleton commands, which are now part of the conda-build project: https://github.com/conda/conda-build
- limited pip integration to `conda list`, that means `conda install` no longer calls `pip install` # !!!
- add ability to call sub-commands named 'conda-x'
- The -c flag to conda search is now shorthand for --channel, not –canonical (this is to be consistent with other conda commands)
- allow changing location of .condarc file using the CONDARC environment variable
- conda search now shows the channel that the package comes from
- conda search has a new --platform flag for searching for packages in other platforms.
- remove condarc warnings: issue #526#issuecomment-33195012

4.5.281 2.3.1 (2014-01-17)

- add ability create info/no_softlink
- add conda convert command to convert non-platform-dependent packages from one platform to another (experimental)
- unify create, install, and update code. This adds many features to create and update that were previously only available to install. A backwards incompatible change is that conda create -f now means --force, not –file.

4.5.282 2.3.0 (2014-01-16)

- automatically prepend http://conda.binstar.org/ (or the value of channel_alias in the .condarc file) to channels whenever the channel is not a URL or the word 'defaults or 'system'
- recipes made with the skeleton pypi command will use setuptools instead of distribute
- re-work the setuptools dependency and entry_point logic so that non console_script entry_points for packages with a dependency on setuptools will get correct build script with conda skeleton pypi
- add -m, --mkdir option to conda install
- add ability to disable soft-linking

4.5.283 2.2.8 (2014-01-06)

- add check for chrpath (on Linux) before build is started, see issue #469
- conda build: fixed ELF headers not being recognized on Python 3
- fixed issues: #467, #476

4.5.284 2.2.7 (2014-01-02)

- fixed bug in conda build related to lchmod not being available on all platforms

4.5.285 2.2.6 (2013-12-31)

- fix test section for automatic recipe creation from pypi using --build-recipe
- minor Py3k fixes for conda build on Linux
- copy symlinks as symlinks, issue #437
- fix explicit install (e.g. from output of `conda list -e`) in root env
- add pyyaml to the list of packages which can not be removed from root environment
- fixed minor issues: #365, #453

4.5.286 2.2.5 (2013-12-17)

- conda build: move broken packages to conda-bld/broken
- conda config: automatically add the 'defaults' channel
- conda build: improve error handling for invalid recipe directory
- add ability to set build string, issue #425
- fix LD_RUN_PATH not being set on Linux under Python 3, see issue #427, thanks peter1000

4.5.287 2.2.4 (2013-12-10)

- add support for execution with the -m switch (issue #398), i.e. you can execute conda also as: python -m conda
- add a deactivate script for windows
- conda build adds .pth-file when it encounters an egg (TODO)
- add ability to preserve egg directory when building using build/preserve_egg_dir: True
- allow track_features in ~/.condarc
- Allow arbitrary source, issue #405
- fixed minor issues: #393, #402, #409, #413

4.5.288 2.2.3 (2013-12-03)

- add "foreign mode", i.e. disallow install of certain packages when using a "foreign" Python, such as the system Python
- remove activate/deactivate from source tarball created by sdist.sh, in order to not overwrite activate script from virtualenvwrapper

4.5.289 2.2.2 (2013-11-27)

- remove ARCH environment variable for being able to change architecture
- add PKG_NAME, PKG_VERSION to environment when running build.sh, .-post-link.sh and .-pre-unlink.sh

4.5.290 2.2.1 (2013-11-15)

- minor fixes related to make conda pip installable
- generated conda meta-data missing 'files' key, fixed issue #357

4.5.291 2.2.0 (2013-11-14)

- add conda init command, to allow installing conda via pip
- fix prefix being replaced by placeholder after conda build on Unix
- add 'use_pip' to condarc configuration file
- fixed activate on Windows to set CONDA_DEFAULT_ENV
- allow setting "always_yes: True" in condarc file, which implies always using the --yes option whenever asked to proceed

4.5.292 2.1.0 (2013-11-07)

- fix rm_egg_dirs so that the .egg_info file can be a zip file
- improve integration with pip
  - conda list now shows pip installed packages
  - conda install will try to install via "pip install" if no conda package is available (unless --no-pip is provided)
  - conda build has a new --build-recipe option which will create a recipe (stored in /conda-recipes) from pypi then build a conda package (and install it)
  - pip list and pip install only happen if pip is installed
- enhance the locking mechanism so that conda can call itself in the same process.

4.5.293 2.0.4 (2013-11-04)

- ensure lowercase name when generating package info, fixed issue #329
- on Windows, handle the .nonadmin files

4.5.294 2.0.3 (2013-10-28)

- update bundle format
- fix bug when displaying packages to be downloaded (thanks Crystal)

4.5.295 2.0.2 (2013-10-27)

- add --index-cache option to clean command, see issue #321
- use RPATH (instead of RUNPATH) when building packages on Linux

4.5.296 2.0.1 (2013-10-23)

- add --no-prompt option to conda skeleton pypi
- add create_default_packages to condarc (and --no-default-packages option to create command)

4.5.297 2.0.0 (2013-10-01)

- added user/root mode and ability to soft-link across filesystems
- added create --clone option for copying local environments
- fixed behavior when installing into an environment which does not exist yet, i.e. an error occurs
- fixed install --no-deps option
- added --export option to list command
- allow building of packages in "user mode"
- regular environment locations now used for build and test
- add ability to disallow specification names
- add ability to read help messages from a file when install location is RO
- restore backwards compatibility of share/clone for conda-api
- add new conda bundle command and format
- pass ARCH environment variable to build scripts
- added progress bar to source download for conda build, issue #230
- added ability to use url instead of local file to conda install --file and conda create --file options

4.5.298 1.9.1 (2013-09-06)

- fix bug in new caching of repodata index

4.5.299 1.9.0 (2013-09-05)

- add caching of repodata index
- add activate command on Windows
- add conda package --which option, closes issue 163
- add ability to install file which contains multiple packages, issue 256
- move conda share functionality to conda package --share
- update documentation
- improve error messages when external dependencies are unavailable
- add implementation for issue 194: post-link or pre-unlink may append to a special file ${PREFIX}/.messages.txt for messages, which is display to the user's console after conda completes all actions
- add conda search --outdated option, which lists only installed packages for which newer versions are available
- fixed numerous Py3k issues, in particular with the build command

4.5.300 1.8.2 (2013-08-16)

- add conda build --check option
- add conda clean --lock option
- fixed error in recipe causing conda traceback, issue 158
- fixes conda build error in Python 3, issue 238
- improve error message when test command fails, as well as issue 229
- disable Python (and other packages which are used by conda itself) to be updated in root environment on Windows
- simplified locking, in particular locking should never crash conda when files cannot be created due to permission problems

4.5.301 1.8.1 (2013-08-07)

- fixed conda update for no arguments, issue 237
- fix setting prefix before calling should_do_win_subprocess() part of issue 235
- add basic subversion support when building
- add --output option to conda build

4.5.302 1.8.0 (2013-07-31)

- add Python 3 support (thanks almarklein)
- add Mercurial support when building from source (thanks delicb)
- allow Python (and other packages which are used by conda itself) to be updated in root environment on Windows
- add conda config command
- add conda clean command
- removed the conda pip command
- improve locking to be finer grained
- made activate/deactivate work with zsh (thanks to mika-fischer)
- allow conda build to take tarballs containing a recipe as arguments
- add PKG_CONFIG_PATH to build environment variables
- fix entry point scripts pointing to wrong python when building Python 3 packages
- allow source/sha1 in meta.yaml, issue 196
- more informative message when there are unsatisfiable package specifications
- ability to set the proxy urls in condarc
- conda build asks to upload to binstar. This can also be configured by changing binstar_upload in condarc.
- basic tab completion if the argcomplete package is installed and eval "$(register-python-argcomplete conda)" is added to the bash profile.

4.5.303 1.7.2 (2013-07-02)

- fixed conda update when packages include a post-link step which was caused by subprocess being lazily imported, fixed by 0d0b860
- improve error message when 'chrpath' or 'patch' is not installed and needed by build framework
- fixed sharing/cloning being broken (issue 179)
- add the string LOCKERROR to the conda lock error message

4.5.304 1.7.1 (2013-06-21)

- fix "executable" not being found on Windows when ending with .bat when launching application
- give a better error message from when a repository does not exist

4.5.305 1.7.0 (2013-06-20)

- allow ${PREFIX} in app_entry
- add binstar upload information after conda build finishes

4.5.306 1.7.0a2 (2013-06-20)

- add global conda lock file for only allowing one instance of conda to run at the same time
- add conda skeleton command to create recipes from PyPI
- add ability to run post-link and pre-unlink script

4.5.307 1.7.0a1 (2013-06-13)

- add ability to build conda packages from "recipes", using the conda build command, for some examples, see: https://github.com/ContinuumIO/conda-recipes
- fixed bug in conda install --force
- conda update command no longer uses anaconda as default package name
- add proxy support
- added application API to conda.api module
- add -c/–channel and --override-channels flags (issue 121).
- add default and system meta-channels, for use in .condarc and with -c (issue 122).
- fixed ability to install ipython=0.13.0 (issue 130)

4.5.308 1.6.0 (2013-06-05)

- update package command to reflect changes in repodata
- fixed refactoring bugs in share/clone
- warn when anaconda processes are running on install in Windows (should fix most permissions errors on Windows)

4.5.309 1.6.0rc2 (2013-05-31)

- conda with no arguments now prints help text (issue 111)
- don't allow removing conda from root environment
- conda update python does no longer update to Python 3, also ensure that conda itself is always installed into the root environment (issue 110)

4.5.310 1.6.0rc1 (2013-05-30)

- major internal refactoring
- use new "depends" key in repodata
- uses pycosat to solve constraints more efficiently
- add hard-linking on Windows
- fixed linking across filesystems (issue 103)
- add conda remove --features option
- added more tests, in particular for new dependency resolver
- add internal DSL to perform install actions
- add package size to download preview
- add conda install --force and --no-deps options
- fixed conda help command
- add conda remove --all option for removing entire environment
- fixed source activate on systems where sourcing a gives "bash" as $0
- add information about installed versions to conda search command
- removed known "locations"
- add output about installed packages when update and install do nothing
- changed default when prompted for y/n in CLI to yes

4.5.311 1.5.2 (2013-04-29)

- fixed issue 59: bad error message when pkgs dir is not writable

4.5.312 1.5.1 (2013-04-19)

- fixed issue 71 and (73 duplicate): not being able to install packages starting with conda (such as 'conda-api')
- fixed issue 69 (not being able to update Python / NumPy)
- fixed issue 76 (cannot install mkl on OSX)

4.5.313 1.5.0 (2013-03-22)

- add conda share and clone commands
- add (hidden) --output-json option to clone, share and info commands to support the conda-api package
- add repo sub-directory type 'linux-armv6l'

4.5.314 1.4.6 (2013-03-12)

- fixed channel selection (issue #56)

4.5.315 1.4.5 (2013-03-11)

- fix issue #53 with install for meta packages
- add -q/–quiet option to update command

4.5.316 1.4.4 (2013-03-09)

- use numpy 1.7 as default on all platforms

4.5.317 1.4.3 (2013-03-09)

- fixed bug in conda.builder.share.clone_bundle()

4.5.318 1.4.2 (2013-03-08)

- feature selection fix for update
- Windows: don't allow linking or unlinking python from the root environment because the file lock, see issue #42

4.5.319 1.4.1 (2013-03-07)

- fix some feature selection bugs
- never exit in activate and deactivate
- improve help and error messages

4.5.320 1.4.0 (2013-03-05)

- fixed conda pip NAME==VERSION
- added conda info --license option
- add source activate and deactivate commands
- rename the old activate and deactivate to link and unlink
- add ability for environments to track "features"
- add ability to distinguish conda build packages from Anaconda packages by adding a "file_hash" meta-data field in info/index.json
- add conda.builder.share module

4.5.321 1.3.5 (2013-02-05)

- fixed detecting untracked files on Windows
- removed backwards compatibility to conda 1.0 version

4.5.322 1.3.4 (2013-01-28)

- fixed conda installing itself into environments (issue #10)
- fixed non-existing channels being silently ignored (issue #12)
- fixed trailing slash in ~/.condarc file cause crash (issue #13)

- fixed conda list not working when ~/.condarc is missing (issue #14)
- fixed conda install not working for Python 2.6 environment (issue #17)
- added simple first cut implementation of remove command (issue #11)
- pip, build commands: only package up new untracked files
- allow a system-wide <sys.prefix>/.condarc (~/.condarc takes precedence)
- only add pro channel is no condarc file exists (and license is valid)

4.5.323 1.3.3 (2013-01-23)

- fix conda create not filtering channels correctly
- remove (hidden) --test and --testgui options

4.5.324 1.3.2 (2013-01-23)

- fix deactivation of packages with same build number note that conda upgrade did not suffer from this problem, as was using separate logic

4.5.325 1.3.1 (2013-01-22)

- fix bug in conda update not installing new dependencies

4.5.326 1.3.0 (2013-01-22)

- added conda package command
- added conda index command
- added -c, --canonical option to list and search commands
- fixed conda --version on Windows
- add this changelog

4.5.327 1.2.1 (2012-11-21)

- remove ambiguity from conda update command

4.5.328 1.2.0 (2012-11-20)

- “conda upgrade” now updates from AnacondaCE to Anaconda (removed upgrade2pro
- add versioneer

4.5.329 1.1.0 (2012-11-13)

- Many new features implemented by Bryan

4.5.330 1.0.0 (2012-09-06)

- initial release

4.6 Glossary

4.6.1 .condarc

The Conda Runtime Configuration file, an optional `.yaml` file that allows you to configure many aspects of conda, such as which channels it searches for packages, proxy settings, and environment directories. A `.condarc` file is not included by default, but it is automatically created in your home directory when you use the `conda config` command. The `.condarc` file can also be located in a root environment, in which case it overrides any `.condarc` in the home directory. For more information, see *Using the .condarc conda configuration file* and *Administering a multi-user conda installation*. Pronounced "conda r-c".

4.6.2 Activate/Deactivate environment

Conda commands used to switch or move between installed environments. The `conda activate` command prepends the path of your current environment to the PATH environment variable so that you do not need to type it each time. `deactivate` removes it. These commands might also execute activation and deactivation logic specified by some installed packages in your environment. Even when an environment is deactivated, you can still execute programs in that environment by specifying their paths directly, as in `~/anaconda/envs/envname/bin/program_name`. When an environment is activated, you can execute the program in that environment with just `program_name`.

> **Note**
>
> Replace `envname` with the name of the environment and replace `program_name` with the name of the program.

4.6.3 Anaconda

A downloadable, free, open-source, high-performance, and optimized Python and R distribution. Anaconda includes *conda*, conda-build, Python, and 250+ automatically installed, open-source scientific packages and their dependencies that have been tested to work well together, including SciPy, NumPy, and many others. Use the `conda install` command to easily install 7,500+ popular open-source packages for data science--including advanced and scientific analytics--from the Anaconda repository. Use the `conda` command to install thousands more open-source packages.

Because Anaconda is a Python distribution, it can make installing Python quick and easy even for new users.

Available for Windows, macOS, and Linux, all versions of Anaconda are supported by the community.

See also *Miniconda* and *conda*.

4.6.4 Anaconda.org

A web-based, repository hosting service in the cloud. Packages created locally can be published to the cloud to be shared with others. Anaconda.org is a public version of Anaconda Repository and was formerly known as Anaconda Cloud.

4.6.5 Anaconda Navigator

A desktop graphical user interface (GUI) included in all versions of Anaconda that allows you to easily manage conda packages, environments, channels, and notebooks without a command line interface (CLI). See more about Navigator.

4.6.6 CEP 23

Conda Enhancement Proposal 23, which defines the specification for explicit environment files. CEP 23 establishes two distinct environment file formats:

- **Explicit format** - Files containing package URLs with `@EXPLICIT` header

- **Text spec format** - Files containing MatchSpec strings (requirements format)

This proposal standardizes environment export formats and ensures compatibility across conda tools. For more information, see the CEP 23 specification.

4.6.7 Channels

The locations of the repositories where conda looks for packages. Channels may point to a Cloud repository or a private location on a remote or local repository that you or your organization created. The `conda channel` command has a default set of channels to search, beginning with https://repo.anaconda.com/pkgs/, which you may override, for example, to maintain a private or internal channel. These default channels are referred to in conda commands and in the `.condarc` file by the channel name "defaults."

4.6.8 conda

The package and environment manager program bundled with Anaconda that installs and updates conda packages and their dependencies. Conda also lets you easily switch between conda environments on your local computer.

4.6.9 conda environment

A folder or directory that contains a specific collection of conda packages and their dependencies, so they can be maintained and run separately without interference from each other. For example, you may use a conda environment for only Python 2 and Python 2 packages, maintain another conda environment with only Python 3 and Python 3 packages, and maintain another for R language packages. Environments can be created from:

- The Navigator GUI
- The command line
- An environment specification file with the name `your-environment-name.yml`

4.6.10 conda package

A compressed file that contains everything that a software program needs in order to be installed and run, so that you do not have to manually find and install each dependency separately. A conda package includes system-level libraries, Python or R language modules, executable programs, and other components. You manage conda packages with conda.

4.6.11 conda repository

A cloud-based repository that contains 7,500+ open-source certified packages that are easily installed locally with the `conda install` command. Anyone can access the repository from:

- The Navigator GUI
- A terminal using conda commands
- https://repo.anaconda.com/pkgs/

4.6.12 Environment exporter

A plugin that defines how conda environments are exported to specific file formats. Environment exporters are part of conda's plugin architecture and allow extending the `conda export` command with custom formats.

Built-in exporters include:

- `environment-yaml` - YAML format for cross-platform sharing
- `environment-json` - JSON format for programmatic processing
- `explicit` - CEP 23 compliant explicit URLs format

- `requirements` - MatchSpec-based requirements format

Custom exporters can be created through the conda plugin system. See *Environment Exporters* for details.

4.6.13 Explicit format

A CEP 23 compliant environment file format that contains package URLs with full metadata for exact environment reproduction. Explicit files begin with an `@EXPLICIT` header followed by complete package URLs.

Example:

```
@EXPLICIT
https://repo.anaconda.com/pkgs/main/osx-64/python-3.9.7-h88f2d9e_0.tar.bz2
https://repo.anaconda.com/pkgs/main/osx-64/numpy-1.21.0-py39h2e5f516_0.tar.bz2
```

This format enables bit-for-bit identical environment reproduction on the same platform and architecture.

4.6.14 Metapackage

A metapackage is a very simple package that has at least a name and a version. It need not have any dependencies or build steps. *Metapackages* may list dependencies to several core, low-level libraries and may contain links to software files that are automatically downloaded when executed.

4.6.15 Miniconda

A free minimal installer for conda. Miniconda is a small, bootstrap version of Anaconda that includes only conda, Python, the packages they depend on, and a small number of other useful packages, including pip, zlib, and a few others. Use the `conda install` command to install 7,500+ additional conda packages from the Anaconda repository.

Miniconda is a Python distribution that can make installing Python quick and easy even for new users.

See also *Anaconda* and *conda*.

4.6.16 Noarch package

A conda package that contains nothing specific to any system architecture, so it may be installed from any system. When conda searches for packages on any system in a channel, conda checks both the system-specific subdirectory, such as `linux-64`, and the `noarch` directory. Noarch is a contraction of "no architecture".

4.6.17 Package manager

A collection of software tools that automates the process of installing, updating, configuring, and removing computer programs for a computer's operating system. Also known as a package management system. Conda is a package manager.

4.6.18 Packages

Software files and information about the software, such as its name, the specific version, and a description, bundled into a file that can be installed and managed by a package manager.

4.6.19 Plugins

Plugins, sometimes referred to as add-ons or extensions, are software or modules that add new functions to a host program (*e.g.*, conda) without directly altering the host program itself. Amongst other uses, plugins support is utilized to enable third-party developers to extend an application, support easily adding new features, and to reduce the size of an application by not loading unused features.

4.6.20 Repository

Any storage location from which software assets may be retrieved and installed on a local computer. See also *Anaconda.org* and *conda repository*.

4.6.21 Requirements format

A CEP 23 compliant environment file format that contains MatchSpec strings for conda-compatible package specifications. This format is similar to pip's `requirements.txt` but uses conda's MatchSpec syntax.

Example:

```
python=3.9.7
numpy=1.21.0
pandas>=1.3.0,<2.0
```

This format provides a balance between exactness and flexibility, allowing some version ranges while remaining conda-compatible.

4.6.22 Silent mode installation

When installing Miniconda or Anaconda in silent mode, screen prompts are not shown on screen and default settings are automatically accepted.

4.6.23 Solvers

The component of conda that decides which package versions can be installed together in the same environment. When you install, update, or remove packages, the solver evaluates the package specifications you request, along with their dependencies and any constraints, and determines a compatible set of packages.

Conda supports multiple solvers, so you can choose the one that best fits your needs. For more information on your solver options, see *Solvers*.

4.7 Developer guide

Welcome to the conda developer guide. Whether you are fixing a bug, adding a new feature, writing a plugin, or diving deep into conda's internals, this guide has everything you need to get started and contribute effectively.

4.7.1 Getting started

New to conda development? Start here.

Contributing 101
: Learn how the conda project is managed, how to open issues and pull requests, and what to expect from the review process.

Development environment
: Follow this guide to set up a local development environment and run conda from source.

4.7.2 Architecture & internals

Understand how conda is structured and how its most complex subsystems work.

Architecture
: A high-level overview of conda's architecture, its major components, and how they interact with each other.

Deep dives
: Detailed explorations of particularly complex subsystems such as the solver, activation, context, and logging.

4.7.3 Extending conda

Build on top of conda using its plugin system.

Plugins
: Learn how to extend and customize conda's behavior using the plugin system, including hooks for solvers, subcommands, auth handlers, and more.

4.7.4 Contributing & quality

Guides for writing tests, managing deprecations, and cutting releases.

Writing tests
: Guidelines and guides for writing unit and integration tests, using the HTTP test server, and testing on Windows with AppLocker.

Deprecations
: Learn the conda deprecation policy and how to mark APIs and behaviors as pending deprecated, deprecated, or removed.

Breaking changes
: Learn the process for planning, announcing, and shipping breaking changes that go beyond a public API removal, such as config default changes, default solver changes, and CLI removals.

Releasing
: Step-by-step instructions for preparing and publishing a new conda release, including the CalVer versioning scheme.

Type hinting
: Guidelines for adding and improving type annotations in the conda codebase, including custom types and local tooling.

4.7.5 API reference

API reference
: Auto-generated API documentation for all public modules, classes, and functions in the conda package.

Architecture

Conda is a complex system of many components and can be hard to understand for users and developers alike. The following C4 model based architecture diagrams should help in that regard. As a refresher, the C4 model tries to visualize complex software systems at different levels of detail, and explaining the functionality to different types of audience.

> **Note**
>
> These diagrams represent the state of conda at the time when the documentation was automatically build as part of the development process for conda 26.11.0.dev47 (Nov 13, 2026).

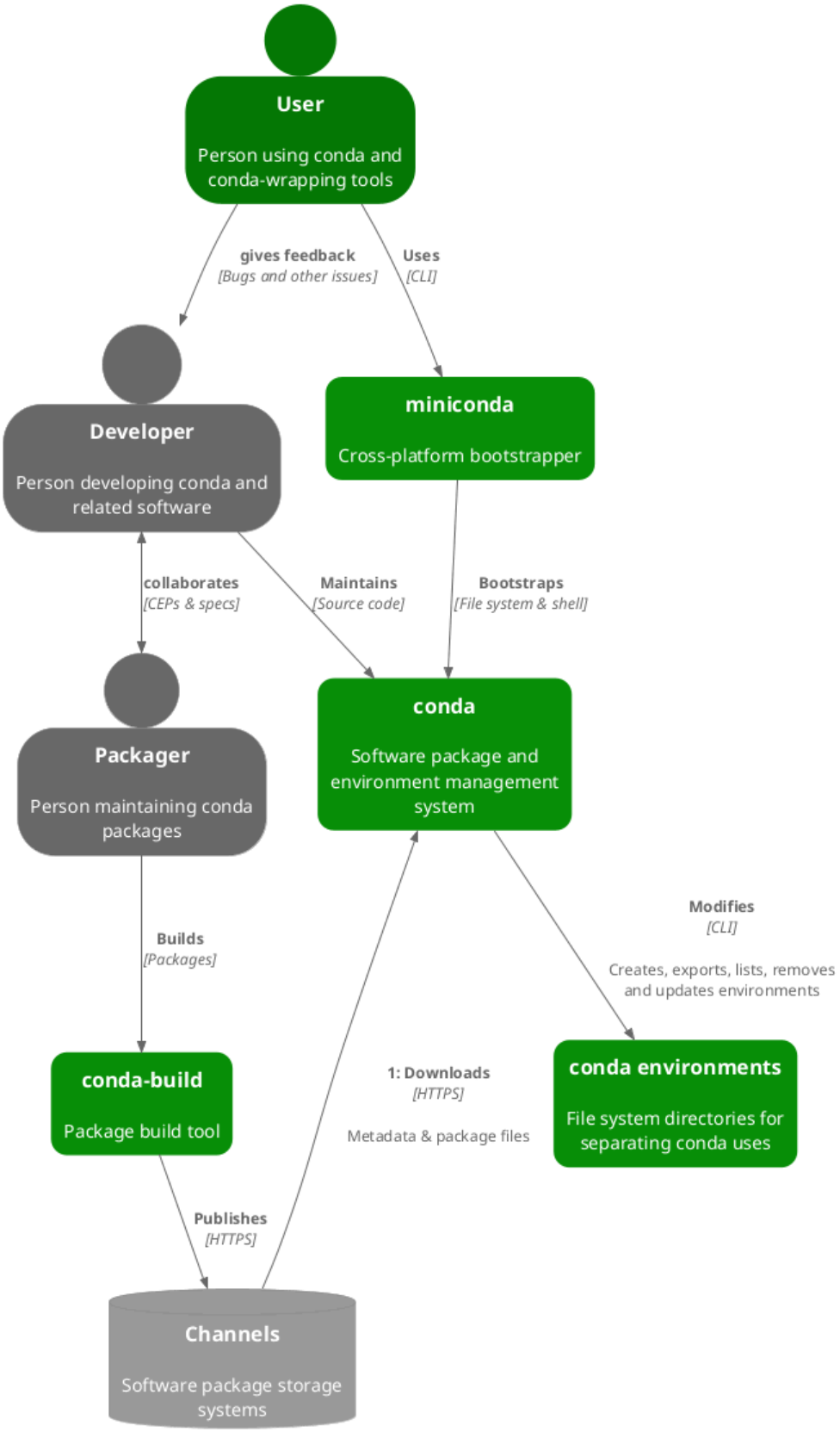
C4 stands for the for levels:

1. *Context*
2. *Container*
3. *Component*
4. *Code*

Level 1: Context

This is the overview, 30,000 feet view on conda, to better understand how conda in the center of the diagram interacts with other systems and how users relate to it.

More information about how to interpret this diagram can be found in the C4 model documentation about the System Context diagram.


User
Person using conda and conda-wrapping tools
gives feedback
[Bugs and other issues]
Uses
[CLI]
Developer
Person developing conda and related software
miniconda
Cross-platform bootstrapper
collaborates
[CEPs & specs]
Maintains
[Source code]
Bootstraps
[File system & shell]
Packager
Person maintaining conda packages
conda
Software package and environment management system
Builds
[Packages]
Modifies
[CLI]
Creates, exports, lists, removes and updates environments
conda-build
Package build tool
1: Downloads
[HTTPS]
Metadata & package files
conda environments
File system directories for separating conda uses
Publishes
[HTTPS]
Channels
Software package storage systems

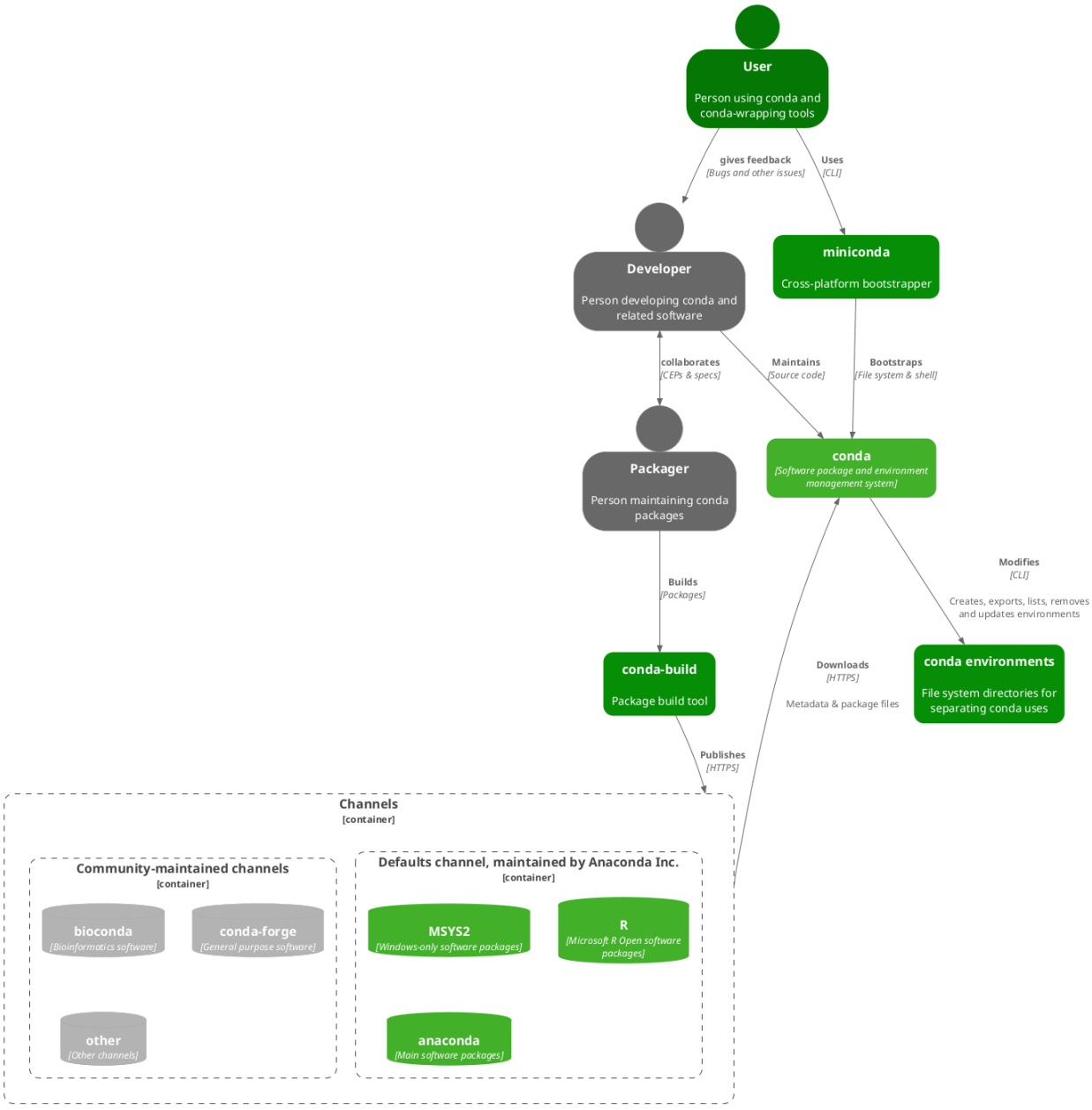
Level 2: Container

This level is zooming in to conda on a system level, which was in the center of the Level 1 diagram, to show the high-level shape of the software architecture of and the various responsibilities in conda, including major technology choices and communication patterns between the various containers.

More information about how to interpret the following diagrams can be found in the C4 model documentation about the Container diagram.

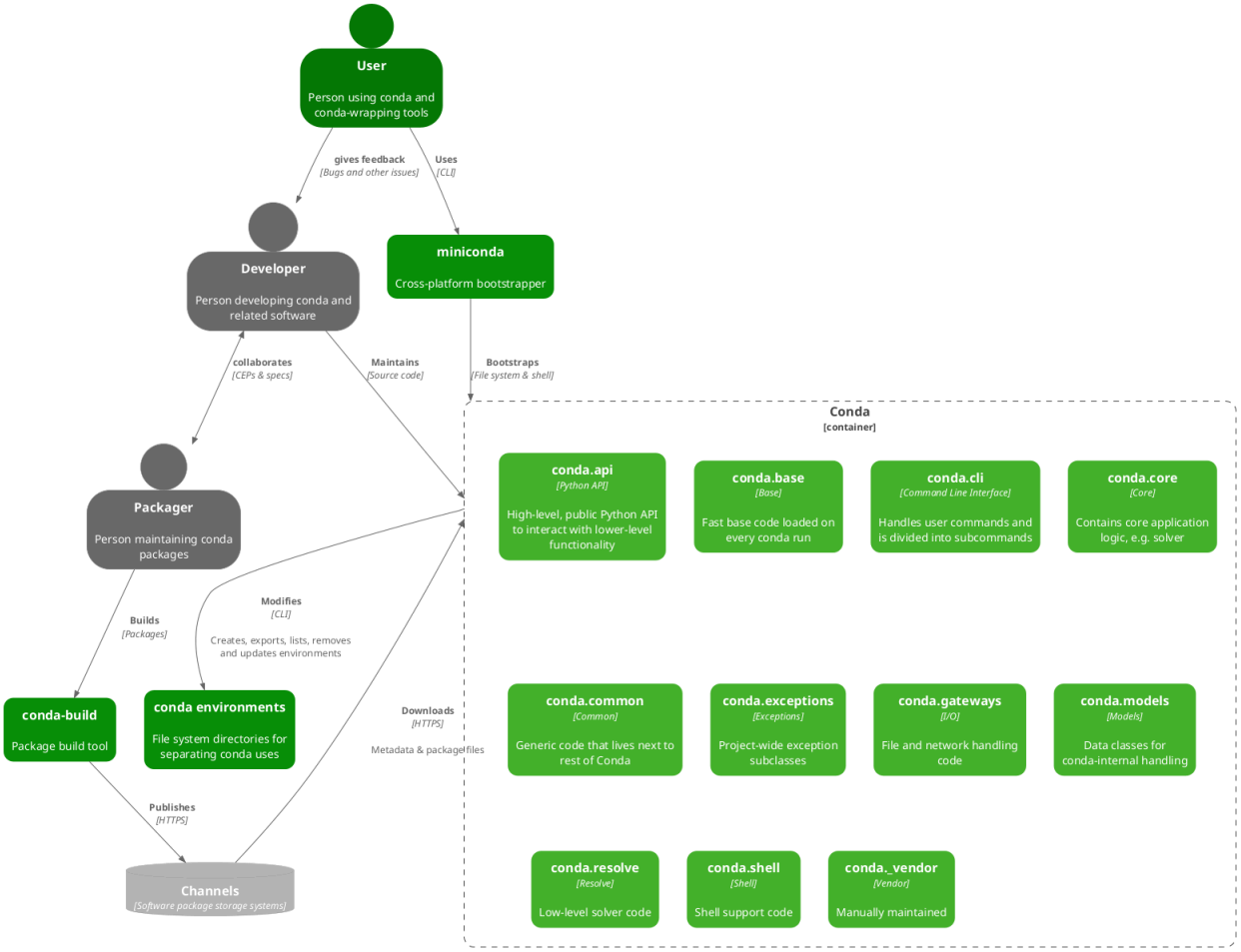
Channels

The following diagram focuses on the channels container from the level 1 diagram.



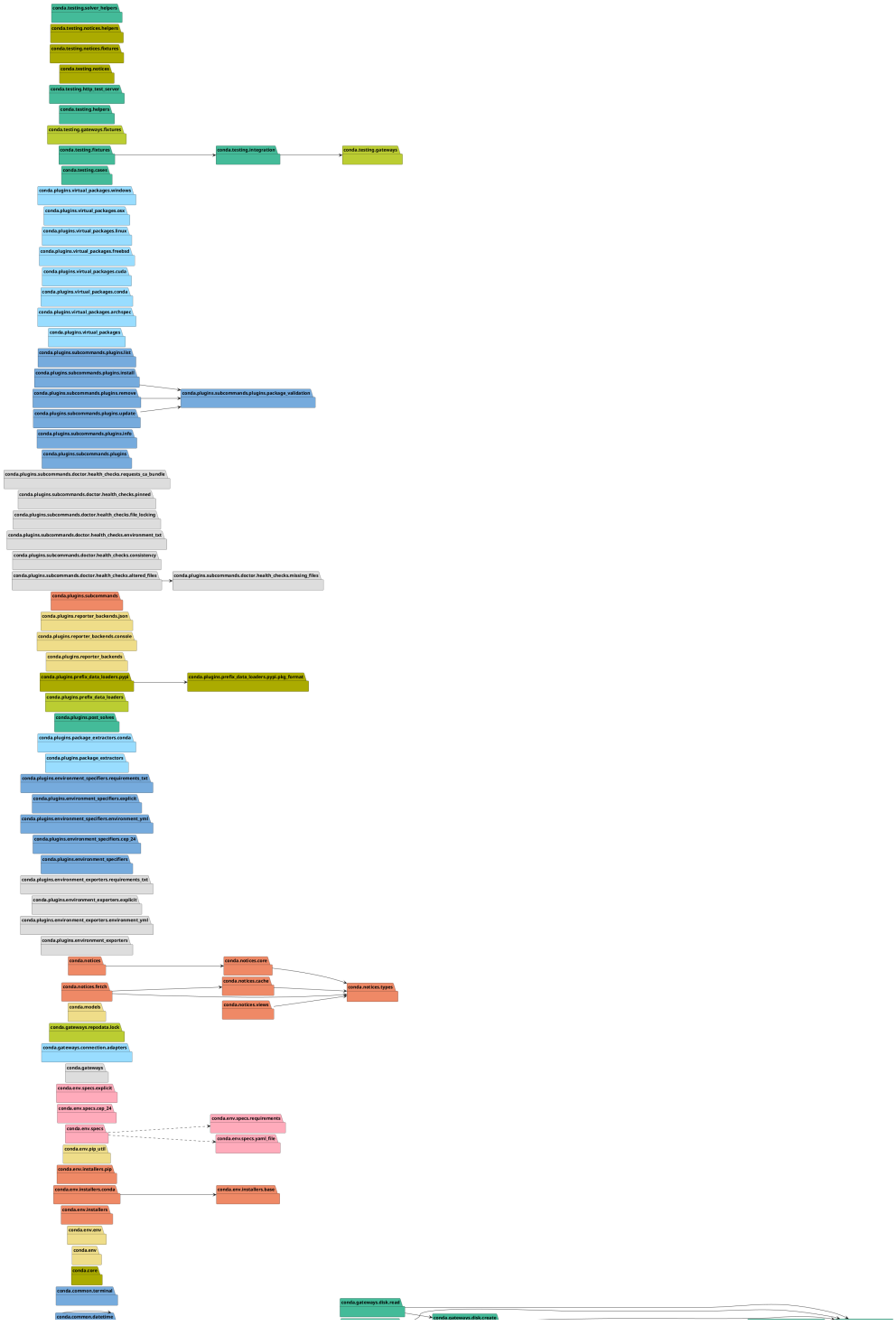
Conda

The following diagram focuses on the conda container from the level 1 diagram.



Level 3: Component

Yet another zoom-in, in which individual containers from Level 2 are decomposed to show major building blocks in conda and their interactions. Those building blocks are called components in the sense that they each have a higher function and relate to an identifiable responsibility and implementation details.

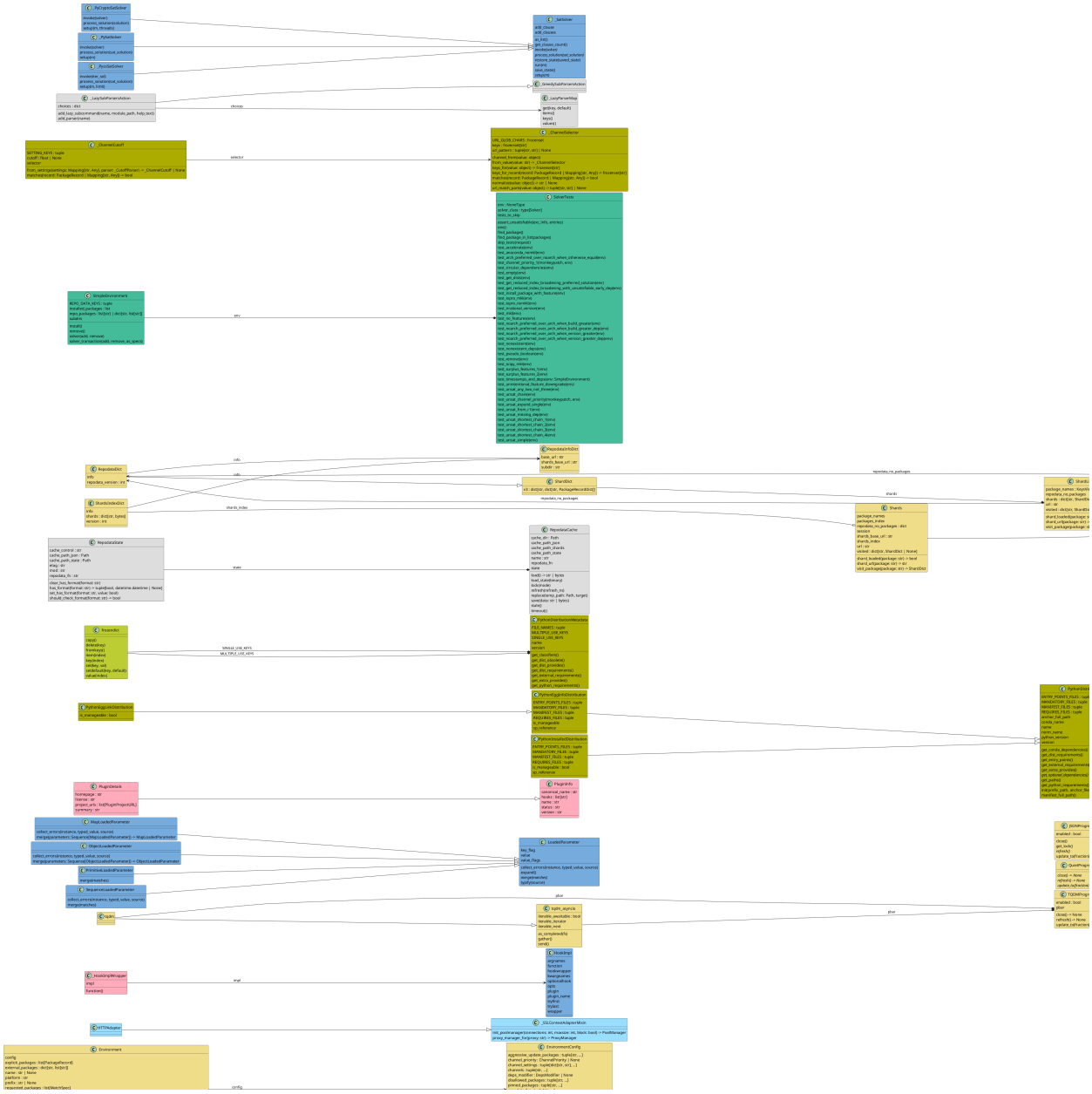


More information about how to interpret this diagram can be found in the C4 model documentation about the Component diagram.

Level 4: Code

This part is auto-generated based on the current code and shows how the code is structured and how it interacts. For brevity this ignores a number of subsystems like the public API and exports modules, utility and vendor packages.

More information about how to interpret this diagram can be found in the C4 model documentation about the Code diagram.



Contributing to conda

Thank you for your interest in improving conda! Below, we describe how our development process works and how you can be a part of it.

Already know how to contribute and need help setting up your development environment? Read the development environment guide here

Hosted on GitHub

All development currently takes place on GitHub. This means we make extensive use of the project management tools they provide such as issues and projects.

Code of Conduct

When you decide to contribute to this project, it is important to adhere to our code of conduct, which is currently the NumFOCUS Code of Conduct. Please read it carefully.

Conda Contributor License Agreement

To begin contributing to this repository, you need to sign the Conda Contributor License Agreement (CLA). In case you're new to CLAs, this is a rather standard procedure for larger projects. Django and Python for example both use similar agreements.

Click here to sign the Conda Contributor License Agreement.

A record of prior signatories is kept in a separate repo in conda's GitHub organization.

Generative AI

You're welcome to use generative AI tools when contributing to conda. However, please keep the following in mind:

- You are responsible for all of your contributions. Review and understand any AI-generated content before including it in a Pull Request.
- You have read and understood the conda license and conda contributor license agreement, especially in terms of submitting your contribution as an original work of authorship.
- Be prepared to engage during contribution review. Reviewers expect to discuss the changes with you directly. Do not copy and paste AI responses.
- Make minimal, focused changes. AI tools sometimes rewrite more than necessary, making reviews harder. Prefer small, targeted edits that match the existing style and patterns.
- Don't bypass tests. Ensure AI-assisted changes actually fix the underlying problem rather than altering tests to make them pass.
- Do not use AI agents or similar automated systems to submit your contributions or review Pull Requests autonomously.

Pull Requests consisting of unchecked AI-generated content may be closed. Maintainer time is limited and contributions should be high quality, regardless of the tools used to create them.

The Conda Code of Conduct applies.

Paid Contribution Schemes

While we support open source contributors receiving financial compensation, we do not accept contributions motivated by cryptocurrency payments, bounties, or similar gamification schemes. These systems incentivize low-quality, high-volume contributions that waste limited maintainer time.

If you would like to add conda to such a system, or believe your situation warrants an exception, please reach out ahead of time on Zulip.

Ways to contribute

Below are all the ways you can get involved in with conda.

Bug reports and feature requests

Bug reports and feature requests are always welcome. To file a new issue, head to the issue form.

It should be noted that `conda-build` issues need to be filed separately at its issue tracker.

For all other types of issues, please head to Anaconda.org's "Report a Bug" page. For even more information and documentation on everything related to Anaconda, head to the Support Center at anaconda.com.

Before submitting an issue via any of these channels, make sure to document it as well as possible and follow the submission guidelines (this makes everyone's job a lot easier!).

Contributing your changes to conda

Here are steps you need to take to contribute to conda:

1. Signup for a GitHub account (if you haven't already) and install Git on your system.
2. Sign the Conda Contributor License Agreement.
3. Fork the conda repository to your personal GitHub account by clicking the "Fork" button on https://github.com/conda/conda and follow GitHub's instructions.
4. Work on your proposed solution. *Visit this page if you need help getting your development environment setup*
5. When you are ready to submit a change, create a pull request.

Your first contribution

For your first contribution, we recommend:

- Look for issues labeled with `good first issue`
- Start with small changes like documentation or minor bug fixes
- Read through existing code to understand the project structure and our patterns
- Asking questions if you need help, see the *getting help* section below.

Change process

In order to maintain proper tracking and visibility, make sure that there are issues for adding features, fixing bugs, and major refactoring work. When creating a new issue, make sure to use the issue template.

When making a change, try to scope changes to what can be accomplished in less than a day of work with minimal code changes to facilitate rapid code reviews—typically fewer than 200 lines of code.

Create a new local branch for each change. Keep your changes focused on a single issue to make reviews easier.

Pull Requests

Pre-submission Steps

1. Install and run pre-commit hooks
2. Make sure all tests are passing
3. Rebase changes on main if other changes were merged during development

4. Use the Pull Request template if one exists for the project
5. Self-review your code with fresh eyes before requesting external review
6. Draft pull requests are acceptable for early feedback

The Pull Request title should clearly explain the contents of the change. Like the first line of a commit message, it should be a maximum of 50 characters, written in an imperative mood, and have no ending period.

Review requirements

Most code changes require one reviewer from the conda-maintainers team. When you submit a Pull Request, that group will automatically be assigned as the reviewer.

What to expect during code review

The primary goal of code review is to ensure maintainability of integrated code. Reviews will focus on:

- Code correctness and logic
- Maintainability and readability
- Consistency with project patterns and practices

Reviews will NOT focus on:

- Code formatting (handled by pre-commit hooks)
- Minor style preferences
- Test coverage and performance (measured by CI)

If you receive feedback, don't worry! Code review is a collaborative process to make the code as good as possible. Address the feedback and feel free to ask questions if anything is unclear.

Conda capitalization standards

1. Conda should be written in lowercase, whether in reference to the tool, ecosystem, packages, or organization.
2. References to the conda command should use code formatting (i.e. `conda`).
3. If the use of conda is not a command and if conda is at the beginning of a sentence, conda should be uppercase.

Examples

In sentences

Beginning a sentence:

- Conda is an open-source package and environment management system.
- `conda install` can be used to install packages.

Conda in the middle of a sentence:

- If a newer version of conda is available, you can use `conda update conda` to update to that version.
- You can find conda packages within conda channels. The `conda` command can search these channels.

In titles and headers

Titles and headers should use the same capitalization and formatting standards as sentences.

In links

Links should use the same capitalization conventions as sentences. Because the conda docs currently use reStructuredText (RST) as a markup language, and RST does not support nested inline markup, documentation writers should avoid using code backtick formatting inside links.

Getting help

If you need help with your contribution:

- Comment on the issue you're working on
- Join our community chat channels

We're here to help and appreciate your contribution!

Development Environment

Repository Setup

Clone the repo you just forked on GitHub to your local machine. Configure your repo to point to both "upstream" (the main conda repo) and your fork ("origin").

Bash (macOS, Linux, Windows)

```
# choose the repository location
# warning: not the location of an existing conda installation!
$ CONDA_PROJECT_ROOT="$HOME/conda"

# clone the project
# replace `your-username` with your actual GitHub username
$ git clone git@github.com:your-username/conda "$CONDA_PROJECT_ROOT"
$ cd "$CONDA_PROJECT_ROOT"

# set the `upstream` as the the main repository
$ git remote add upstream git@github.com:conda/conda
```

cmd.exe (Windows)

```
:: choose the repository location
:: warning: not the location of an existing conda installation!
> set "CONDA_PROJECT_ROOT=%HOMEPATH%\conda"

:: clone the project
:: replace `your-username` with your actual GitHub username
> git clone git@github.com:your-username/conda "%CONDA_PROJECT_ROOT%"
> cd "%CONDA_PROJECT_ROOT%"

:: set the `upstream` as the main repository
> git remote add upstream git@github.com:conda/conda
```

Development Environment Setup

Create a local development environment and activate it using the `dev/start` scripts:

Bash (macOS, Linux, Windows)

```
$ source ./dev/start
```

cmd.exe (Windows)

```
> .\dev\start.bat
```

This command will create a project-specific base environment (see `devenv` in your repo directory after running this command). If the base environment already exists this command will simply activate the already-created `devenv` environment.

To be sure that the conda code being interpreted is the code in the project directory, look at the value of `conda location:` in the output of `conda info --all`.

Choosing Your Installer

The `dev/start` script supports two different conda installers:

- **miniconda** (default): Uses the Anaconda defaults channel and official Miniconda installer
- **miniforge**: Uses the conda-forge channel and community-maintained Miniforge installer

Configuration Options

You can specify the installer type in several ways, in order of precedence:

1. **Command line flag** (highest priority)
2. **Configuration file** (`~/.condarc`)
3. **Interactive prompt** (lowest priority)

To avoid being prompted every time, you can set your preferred installer in your `~/.condarc` file:

```
# ~/.condarc
installer_type: miniforge  # or miniconda
```

When you run the script for the first time without specifying an installer and no configuration file setting, you'll be prompted to choose:

```
Choose conda installer:
  1) miniconda (default - Anaconda defaults channel)
  2) miniforge (conda-forge channel)
Enter choice [1]:
```

You can also specify the installer directly using the `-i` or `--installer` flag for Bash, or `/I` for `cmd.exe`:

Bash (macOS, Linux, Windows)

```
# Use miniconda (default behavior)
$ source ./dev/start -i miniconda
```

(continues on next page)

(continued from previous page)

```
# Use miniforge
$ source ./dev/start -i miniforge
```

cmd.exe (Windows)

```
:: Use miniconda (default behavior)
> .\dev\start.bat /I miniconda

:: Use miniforge
> .\dev\start.bat /I miniforge
```

Additional Options

The `dev/start` script supports several other options for customizing your development environment:

Bash (macOS, Linux, Windows)

```
# See all available options
$ source ./dev/start --help

# Use a specific Python version
$ source ./dev/start -p 3.11

# Force update packages
$ source ./dev/start -u

# Preview what would be done without making changes
$ source ./dev/start -n
```

cmd.exe (Windows)

```
:: See all available options
> .\dev\start.bat /?

:: Use a specific Python version
> .\dev\start.bat /P 3.11

:: Force update packages
> .\dev\start.bat /U

:: Preview what would be done without making changes
> .\dev\start.bat /N
```

Switching Between Installers

You can maintain separate development environments for different installers and switch between them:

Bash (macOS, Linux, Windows)

```
# Set up and activate miniconda-based environment
$ source ./dev/start -i miniconda

# Later, set up and activate miniforge-based environment
$ source ./dev/start -i miniforge
```

cmd.exe (Windows)

```
:: Set up and activate miniconda-based environment
> .\dev\start.bat /I miniconda

:: Later, set up and activate miniforge-based environment
> .\dev\start.bat /I miniforge
```

Each installer creates its own isolated environment, so you can test conda behavior with both the defaults and conda-forge channels.

Manual Setup

If you prefer to set up your development environment manually instead of using the automated scripts, follow these steps:

Prerequisites

- Conda is already installed and initialized on your system
- (Optional) If using Option B in step 4, install conda-pypi canary version: `conda install -n base conda-canary/label/dev::conda-pypi`

1. Create the development directory:

   ```
   $ mkdir -p ./devenv
   ```

2. Download and install miniforge:

   #### macOS

   ```
   $ curl -L "https://github.com/conda-forge/miniforge/releases/latest/download/
   ↪Miniforge3-MacOSX-$(uname -m).sh" -o ./devenv/miniforge.sh
   $ bash ./devenv/miniforge.sh -bfp ./devenv
   $ rm ./devenv/miniforge.sh
   ```

   #### Linux

   ```
   $ curl -L "https://github.com/conda-forge/miniforge/releases/latest/download/
   ↪Miniforge3-Linux-$(uname -m).sh" -o ./devenv/miniforge.sh
   $ bash ./devenv/miniforge.sh -bfp ./devenv
   $ rm ./devenv/miniforge.sh
   ```

Windows (PowerShell)

```
> (New-Object System.Net.WebClient).DownloadFile("https://github.com/conda-forge/
↪miniforge/releases/latest/download/Miniforge3-Windows-x86_64.exe", "$env:TEMP\
↪miniforge.exe")
> Start-Process $env:TEMP\miniforge.exe -ArgumentList "/InstallationType=JustMe","/
↪RegisterPython=0","/AddToPath=0","/NoRegistry=1","/NoShortcuts=1","/S","/D=$PWD\
↪devenv" -Wait -NoNewWindow
> rm $env:TEMP\miniforge.exe
```

3. Create and configure environment:

Bash (macOS, Linux)

```
# Install dependencies
$ ./devenv/bin/conda install -y -p ./devenv --override-channels -c conda-forge --
↪file ./tests/requirements.txt --file ./tests/requirements-ci.txt python=3.14
```

PowerShell (Windows)

```
# Install dependencies
> .\devenv\Scripts\conda.exe install -y -p .\devenv --override-channels -c conda-
↪forge --file .\tests\requirements.txt --file .\tests\requirements-ci.txt python=3.
↪14
```

4. Make conda source code available and activate environment:

   Choose one of the following options:

   Option A: Set PYTHONPATH

   Set PYTHONPATH to make the conda source code available, then activate the environment (`dev/start` does this automatically):

Bash (macOS, Linux)

```
# Set PYTHONPATH
$ export PYTHONPATH=$(pwd):$PYTHONPATH

# Activate environment
$ conda activate ./devenv
```

Fish (macOS, Linux)

```
# Set PYTHONPATH
$ set -gx PYTHONPATH (pwd) $PYTHONPATH

# Activate environment
$ conda activate ./devenv
```

PowerShell (Windows)

```
# Set PYTHONPATH
> $env:PYTHONPATH="$PWD;$env:PYTHONPATH"

# Activate environment
> conda activate .\devenv
```

Option B: Install conda in editable mode

Activate the environment first, then install conda in editable mode using conda-pypi:

Bash (macOS, Linux)

```
# Activate environment
$ conda activate ./devenv

# Install conda in editable mode
$ conda pypi install -e .
```

PowerShell (Windows)

```
# Activate environment
> conda activate .\devenv

# Install conda in editable mode
> conda pypi install -e .
```

This installs conda as an editable package in the environment, which can be more convenient for development.

Docker Alternative

Alternatively, for Linux development only, you can use the same Docker image the CI pipelines use. Note that you can run this from all three operating systems! We are using `docker compose`, which provides three actions for you:

- `unit-tests`: Run all unit tests.
- `integration-tests`: Run all integration tests.
- `interactive`: You are dropped in a pre-initialized Bash session, where you can run all your `pytest` commands as required.

Use them with `docker compose run <action>`. For example:

```
$ docker compose run unit-tests
```

This builds the same Docker image as used in continuous integration from the Github Container Registry and starts `bash` with the conda development mode already enabled.

By default, it will use Miniconda-based, Python 3.9 installation configured for the `defaults` channel. You can customize this with two environment variables:

- `CONDA_DOCKER_PYTHON`: `major.minor` value; e.g. `3.11`.
- `CONDA_DOCKER_DEFAULT_CHANNEL`: either `defaults` or `conda-forge`

For example, if you need a conda-forge based 3.12 image:

Bash (macOS, Linux, Windows)

```
$ CONDA_DOCKER_PYTHON=3.12 CONDA_DOCKER_DEFAULT_CHANNEL=conda-forge docker compose build␣
↪--no-cache
# --- in some systems you might also need to re-supply the same values as CLI flags:
# CONDA_DOCKER_PYTHON=3.12 CONDA_DOCKER_DEFAULT_CHANNEL=conda-forge docker compose build␣
↪--no-cache --build-arg python_version=3.12 --build-arg default_channel=conda-forge
$ CONDA_DOCKER_PYTHON=3.12 CONDA_DOCKER_DEFAULT_CHANNEL=conda-forge docker compose run␣
↪interactive
```

cmd.exe (Windows)

```
> set CONDA_DOCKER_PYTHON=3.12
> set CONDA_DOCKER_DEFAULT_CHANNEL=conda-forge
> docker compose build --no-cache
> docker compose run interactive
> set "CONDA_DOCKER_PYTHON="
> set "CONDA_DOCKER_DEFAULT_CHANNEL="
```

The `conda` repository will be mounted to `/opt/conda-src`, so all changes done in your editor will be reflected live while the Docker container is running.

Static Code Analysis

This project is configured with pre-commit to automatically run linting and other static code analysis on every commit. Running these tools prior to the PR/code review process helps in two ways:

1. it helps *you* by automating the nitpicky process of identifying and correcting code style/quality issues
2. it helps *us* where during code review we can focus on the substance of your contribution

Feel free to read up on everything pre-commit related in their docs but we've included the gist of what you need to get started below:

Bash (macOS, Linux, Windows)

```
# reuse the development environment created above
$ source ./dev/start
# or start the Docker image in interactive mode
# $ docker compose run interactive

# install pre-commit hooks for conda
$ cd "$CONDA_PROJECT_ROOT"
$ pre-commit install

# manually running pre-commit on current changes
# note: by default pre-commit only runs on staged files
$ pre-commit run

# automatically running pre-commit during commit
$ git commit
```

cmd.exe (Windows)

```
:: reuse the development environment created above
> .\dev\start.bat
:: or start the Docker image in interactive mode
:: > docker compose run interactive

:: install pre-commit hooks for conda
> cd "%CONDA_PROJECT_ROOT%"
> pre-commit install

:: manually running pre-commit on current changes
:: note: by default pre-commit only runs on staged files
> pre-commit run

:: automatically running pre-commit during commit
> git commit
```

Beware that some of the tools run by pre-commit can potentially modify the code (see black, blacken-docs, and darker). If pre-commit detects that any files were modified it will terminate the commit giving you the opportunity to review the code before committing again.

Strictly speaking using pre-commit on your local machine for commits is optional (if you don't install pre-commit you will still be able to commit normally). But once you open a PR to contribue your changes, pre-commit will be automatically run at which point any errors that occur will need to be addressed prior to proceeding.

Testing

We use pytest to run our test suite. Please consult pytest's docs for detailed instructions but generally speaking all you need is the following:

Bash (macOS, Linux, Windows)

```
# reuse the development environment created above
$ source ./dev/start
# or start the Docker image in interactive mode
# $ docker compose run interactive

# run conda's unit tests using GNU make
$ make unit

# or alternately with pytest
$ pytest --cov -m "not integration" conda tests

# or you can use pytest to focus on one specific test
$ pytest --cov tests/test_create.py -k create_install_update_remove_smoketest
```

If you are not measuring code coverage, `pytest` can be run without the `--cov` option. The `docker compose` tests pass `--cov`.

See *Benchmarks* for running and adding performance benchmarks, exporting measurements, and reviewing Bencher results.

Note: Some integration tests require you build a package with conda-build beforehand. This is taking care of if you run `docker compose run integration-tests`, but you need to do it manually in other modes:

```
$ conda install conda-build
$ conda-build tests/data/recipes/activate_deactivate_package tests/data/recipes/pre_link_
↪messages_package
```

Check `dev/linux/integration.sh` and `dev\windows\integration.bat` for more details.

Deep dives

Detailed explorations of particularly complex subsystems in conda. These guides go beyond the high-level architecture and explain the internal mechanics of specific components.

`conda install`

In this document we will explore what happens in Conda from the moment a user types their installation command until the process is finished successfully. For the sake of completeness, we will consider the following situation:

- The user is running commands on a Linux x64 machine with a working installation of Miniconda.
- This means we have a `base` environment with `conda`, `python`, and their dependencies.
- The `base` environment is already preactivated for Bash. For more details on activation, check *conda init and conda activate*.

Ok, so… what happens when you run `conda install numpy`? Roughly, these steps:

1. Command line interface
    - `argparse` parsers
    - Environment variables
    - Configuration files
    - Context initialization
    - Delegation of the task
2. Fetching the index
    - Retrieving all the channels and platforms
    - A note on channel priorities
3. Solving the install request
    - Requested packages + prefix state = list of specs
    - Index reduction (sometimes)
    - Running the solver
    - Post-processing the list of packages
4. Generating the transaction and the corresponding actions
5. Download and extraction
6. Integrity verification
7. Linking and unlinking files
8. Post-linking and post-activation tasks

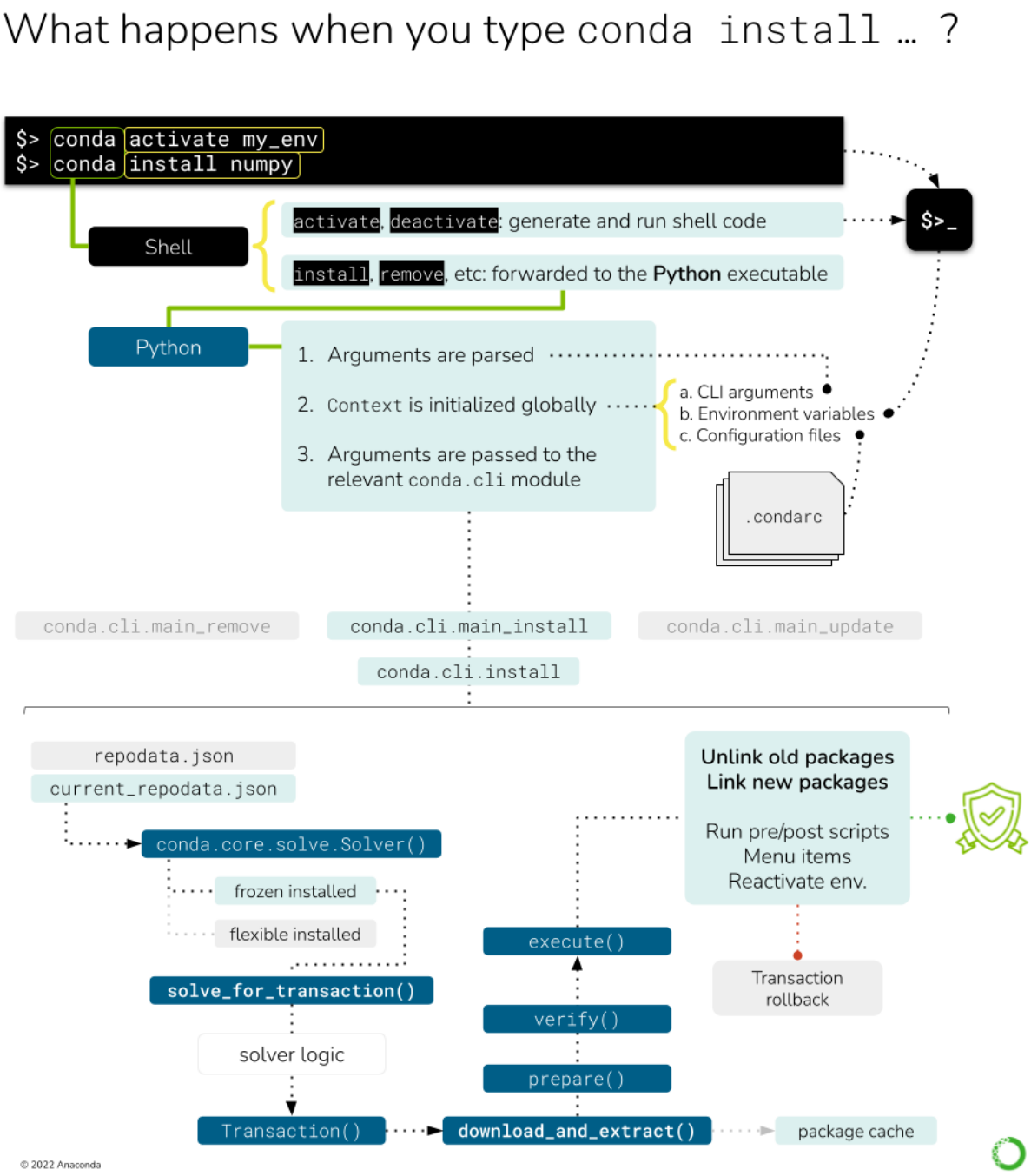


Fig. 1: This figure shows the different processes and objects involved in handling a simple `conda install` command.

Command line interface

First, a quick note on an implementation detail that might be not obvious.

When you type `conda install numpy` in your terminal, Bash takes those three words and looks for a `conda` command to pass a list of arguments `['conda', 'install', 'numpy']`. Before finding the `conda` executable located at `CONDA_HOME/condabin`, it probably finds the shell function defined here. This shell function runs the activation/deactivation logic on the shell if requested, or delegates over to the actual Python entry-points otherwise. This part of the logic can be found in `conda.shell`.

Once we are running the Python entry-point, we are in the `conda.cli` realm. The function called by the entry point is `conda.cli.main:main()`. Here, another check is done for `shell.*` subcommands, which generate the shell initializers you see in `~/.bashrc` and others. If you are curious where this happens, it's `conda.activate`.

Since our command is `conda install ...`, we still need to arrive somewhere else. You will notice that the rest of the logic is delegated to `conda.cli.main:_main()`, which will invoke the parser generators, initialize the context and loggers, and, eventually, pass the argument list over to the corresponding command function. These four steps are implemented in four functions/classes:

1. `conda.cli.conda_argparse:generate_parser()`: This uses `argparse` to generate the CLI. Each subcommand is initialized in separate functions. Note that the command line options are not generated dynamically from the `Context` object, but annotated manually. If this is needed (e.g. `--repodata-fn` is exposed in `Context.repodata_fn`), the `dest` variable of each CLI option should match the target attribute in the context object.
2. `conda.base.context.Context`: This object stores the configuration options in `conda` and will be initialized taking into account, among other things, the arguments parsed in the step above. This is covered in more detail in a separate deep dive: *conda config and context*.
3. `conda.gateways.logging:initialize_logging()`: Not too exciting and easy to follow. This part of the code base is more or less self-explanatory.
4. `conda.cli.conda_argparse:do_call()`: The argument parsing will populate a `func` value that contains the import path to the function responsible for that subcommand. For example, `conda install` is taken care of by `conda.cli.main_install`. By design, all the modules reported by `func` must contain an `execute()` function that implements the command logic. `execute()` takes the parsed arguments and the parser itself as arguments. For example, in the case of `conda install`, `execute()` only redirects to a certain mode in `conda.cli.install:install()`.

Let's go take a look at that module now. `conda.cli.install:install()` implements the logic behind `conda create`, `conda install`, `conda update` and `conda remove`. In essence, they all deal with the same task: changing which packages are present in an environment. If you go and read that function, you will see there are several lines of code handling diverse situations (new environments, clones, etc.) before we arrive to the next section. We will not discuss them here, but feel free to explore that section. It's mostly ensuring that the destination prefix exists, whether we are creating a new environment and massaging some command line flags that would allow us to skip the solver (e.g. `--clone`).

More information on environments

Check the concepts for *Environments*.

Fetching the index

At this point, we are ready to start doing some work! All of the previous code was telling us what to do, and now we know. We want `conda` to *install* `numpy` on our `base` environment. The first thing we need to know is where we can find packages with the name `numpy`. The answer is… the channels!

Users download packages from `conda` channels. These are normally hosted at `anaconda.org`. A channel is essentially a directory structure with these elements:

```
<channel>
├── channeldata.json
├── index.html
├── <platform> (e.g. linux-64)
│   ├── current_repodata.json
│   ├── current_repodata.json.bz2
│   ├── index.html
│   ├── repodata.json
│   ├── repodata.json.bz2
│   ├── repodata_from_packages.json
│   └── repodata_from_packages.json.bz2
└── noarch
    ├── current_repodata.json
    ├── current_repodata.json.bz2
    ├── index.html
    ├── repodata.json
    ├── repodata.json.bz2
    ├── repodata_from_packages.json
    └── repodata_from_packages.json.bz2
```

More info on Channels

You can find some more user-oriented notes on Channels at *What is a "channel"?* and *Repository structure and index*. If you are interested in more technical details, check the corresponding documentation pages at conda-build.

The important bits are:

- A channel contains one or more platform-specific directories (`linux-64`, `osx-64`, etc.), plus a platform-agnostic directory called `noarch`. In `conda` jargon, these are also referred to as channel *subdirs*. Officially, the `noarch` subdirectory is enough to make it a `conda` channel; e.g. no platform subdirectory is necessary.
- Each *subdir* contains at least a `repodata.json` file: a gigantic dictionary with *all* the metadata for each package available on that platform.
- In most cases, the same subdirs also contain the `*.tar.bz2` files for each of the published packages. This is what `conda` downloads and extracts once solving is complete. The anatomy of these files is well defined, both in content and naming structure. See *What is a package?*, *Package metadata* and/or Package naming conventions for more details.

Additionally, the channel's main directory might contain a `channeldata.json` file, with channel-wide metadata (this is not specific per platform). Not all channels include this, and in general it is not currently something that is commonly utilized.

Since conda's philosophy is to keep all packages ever published around for reproducibility, `repodata.json` is always growing, which presents a problem both for the download itself and the solver engine. To reduce download times and bandwidth usage, `repodata.json` is also served as a BZIP2 compressed file, `repodata.json.bz2`. This is what most `conda` clients end up downloading.

Note on 'current_repodata.json'

More *repodatas* variations can be found in some channels, but they are always reduced versions of the main one for the sake of performance. For example, `current_repodata.json` only contains the most recent version of each package, plus their dependencies. The rationale behind this optimization trick can be found here.

So, in essence, fetching the channel information means it can be expressed in pseudo-code like this:

```
platform = {}
noarch = {}
for channel in reversed(context.channels):
    platform_repodata = fetch_extract_and_read(
        channel.full_url / context.subdir / "repodata.json.bz2"
    )
    platform.update(platform_repodata)
    noarch_repodata = fetch_extract_and_read(
        channel.full_url / "noarch" / "repodata.json.bz2"
    )
    noarch.update(noarch_repodata)
```

Note that these dictionaries are keyed by *filename*, so higher priority channels will overwrite entries with the exact same filename (e.g. `numpy-1.19-py36h87ha43_0.tar.bz2`). If they don't have the same *filename* (e.g., same version and build number but different hash), this ambiguity will be resolved later in the solver, taking into account the channel priority mode.

In this example, `context.channels` has been populated through different, cascading mechanisms:

- The default settings as found in `~/.condarc` or equivalent.
- The `CONDA_CHANNELS` environment variable (rare usage).
- The command-line flags, such as `-c <channel>`, `--use-local` or `--override-channels`.
- The channels present in a command-line *spec*. Remember that users can say `channel::numpy` instead of simply `numpy` to require that numpy comes from that specific channel. That means that the repodata for such channel needs to be fetched, too!

The items in `context.channels` are supposed to be `conda.models.channels.Channel` objects, but the Solver API also allows strings that refer to their name, alias or full URL. In that case, you can use `Channel` objects to parse and retrieve the full URL for each subdir using the `Channel.urls()` method. Several helper functions can be found in `conda.core.index`, if needed.

Sadly, `fetch_extract_and_read()` does not exist as such, but as a combination of objects. The main driving function is actually `get_index()`, which passes the channel URLs to `fetch_index`, a wrapper that delegates directly to `conda.core.subdir_data.SubdirData` objects. This object implements caching, authentication, proxies and other things that complicate the simple idea of "just download the file, please". Most of the logic is in `SubdirData._load()`, which ends up calling `conda.core.subdir_data.fetch_repodata_remote_request()` to process the request. Finally, `SubdirData._process_raw_repodata_str()` does the parsing and loading.

Internally, the `SubdirData` stores all the package metadata as a list of `PackageRecord` objects. Its main usage is via `.query()` (one result at a time) or `.query_all()` (all possible matches). These `.query*` methods accept spec strings (e.g. `numpy =1.14`), `MatchSpec` and `PackageRecord` instances. Alternatively, if you want *all* records with no queries, use `SubdirData.iter_records()`.

Tricks to reduce the size of the index

`conda` supports the notion of trying with different versions of the index in an effort to minimize the solution space. A smaller index means a faster search, after all! The default logic starts with `current_repodata.json` files in the channel, which contain only the latest versions of each package plus their dependencies. If that fails, then the full `repodata.json` is used. This happens *before* the `Solver` is even invoked.

The second trick is done within the classic solver logic (pycosat): an informed index reduction. In essence, the index (whether it's `current_repodata.json` or full `repodata.json`) is pruned by the solver, trying to keep

only the parts that it anticipates will be needed. More details can be found on the `get_reduced_index` function. Interestingly, this optimization step also takes longer the bigger the index gets.

Note

The classic solver, now named `pycosat` (`classic` remains a supported alias), is provided by the standalone `conda-pycosat-solver` plugin rather than being registered by conda itself.

Channel priorities

`context.channels` returns an `IndexedSet` of `Channel` objects; essentially a list of unique items. The different channels in this list can have overlapping or even conflicting information for the same package name. For example, `defaults` and `conda-forge` will for sure contain packages that fullfil the `conda install numpy` request. Which one is chosen by `conda` in this case? It depends on the `context.channel_priority` setting: From the help message:

> Accepts values of 'strict', 'flexible', and 'disabled'. The default value is 'flexible'. With strict channel priority, packages in lower priority channels are not considered if a package with the same name appears in a higher priority channel. With flexible channel priority, the solver may reach into lower priority channels to fulfill dependencies, rather than raising an unsatisfiable error. With channel priority disabled, package version takes precedence, and the configured priority of channels is used only to break ties.

While `channel_priority=strict` is often faster to solve, it can make some environments unsatisfiable. Check more details *here*.

Solving the install request

At this point, we can start asking the solver things. After all, we have loaded the channels into our index, building the catalog of available packages and versions we can install. We also have the command line instructions and configurations needed to customize the solver request. So, let's just do it: "Solver, please install numpy on this prefix using these channels as package sources".

The details are complicated, but in essence, the `Solver` will:

1. Express the requested packages, command line options and prefix state as `MatchSpec` objects
2. Query the index for the best possible match that satisfy those constraints
3. Return a list of `PackageRecord` objects

The full details are covered in *Solvers* if you are curious. Just keep in mind that point (1) is conda-specific, while (2) can be tackled, in principle, by any SAT solver.

Generating the transaction and the corresponding actions

The Solver API defines three public methods:

- `.solve_final_state()`: this is the core function, described in the section above. Given some input state, it returns an `IndexedSet` of `PackageRecord` objects that reflect what the final state of the environment should look like. This is the largest method, and its details are fully covered *here*.
- `.solve_for_diff()`: this method takes the final state and diffs it with the current state of the environment, discovering which old records need to be removed, and which ones need to be added.
- `.solve_for_transaction()`: this method takes the diff and creates a `Transaction` object for this operation. This is what the main CLI logic expects back from the solver.

So what is a `Transaction` object and why is it needed? Transactional actions were introduced in conda 4.3. They seem to be the last iteration of a set of changes designed to check whether `conda` would be able to download and link the needed packages (e.g. check that there is enough space on disk, whether the user has enough permissions for the target paths, etc.). For more info, refer to PRs #3571, #3301, and #3034.

The transaction is essentially a set of `action` objects. Each action is allowed to run some checks to determine whether it can be executed successfully. If that's not the case, the failed checks will signal the parent transaction that the whole operation needs to be aborted and rolled back to leave things in the state they were before running that `conda` command. It is also responsible for some of the messages you will see in the CLI output, like the reports of what will be installed, updated or removed.

Transactions and parallelism

Since the transaction object knows about all the actions that need to happen, it also enables parallelism for verifying, downloading and (un)linking tasks. The level of parallelism can be changed through the following `context` settings:

- `default_threads`
- `verify_threads`
- `execute_threads`
- `repodata_threads`
- `fetch_threads`

There's only one class of transaction in `conda`: `UnlinkLinkTransaction`. It only accepts one input parameter: a list of `PrefixSetup` objects, which are just `namedtuple` objects with the following fields. These are populated by `Solver.solve_for_transaction` after running `Solver.solve_for_diff`:

- `target_prefix`: the environment path the command is running on.
- `unlink_precs`: `PackageRecord` objects that need to be unlinked (removed).
- `link_precs`: `PackageRecord` objects that need to be linked (added).
- `remove_specs`: `MatchSpec` objects that need to be marked as removed in the history (the user asked for these packages to be uninstalled).
- `update_specs`: `MatchSpec` objects that need to be marked as added in the history (the user asked for these packages to be installed or updated).
- `neutered_specs`: `MatchSpec` objects that were already in history but had to be relaxed in order to avoid solving conflicts.

Whatever happens after instantiation depends on the content of these `PrefixSetup` objects. Sometimes, the transaction results in no actions (see the `nothing_to_do` property) because the request asked by the user is already fulfilled by the current state of the environment.

However, most of the time the transaction will involve a number of actions. This is done via two public methods:

- `download_and_extract()`: essentially a forwarder to instantiate and call `ProgressiveFetchExtract`, responsible for deciding which `PackageRecords` need to be downloaded and extracted to the packages cache.
- `execute()`: the core logic is layed out here. It involves preparing, verifying and performing the rest of the actions. Among others:
  - Unlinking packages (removing a package from the environment)
  - Linking (adding a package to the environment)
  - Compiling bytecode (generating the `pyc` counterpart for each `py` module)

- Adding entry points (generate command line executables for the configured functions)
- Adding the JSON records (for each package, a JSON file is added to `conda-meta/`)
- Make menu items (create shortcuts for packages featuring a JSON file under `Menu/`)
- Remove menu items (remove the shortcuts created by that package)

It's important to notice that download and extraction happen separately from all the other actions. This separation is important and core to the idea of what a `conda` environment is. Essentially, when you create a new `conda` environment, you are not necessarily *copying* files over to the target prefix location. Instead, `conda` maintains a cache of every package ever downloaded to disk (both the tarball and the extracted contents). To save space and speed up environment creation and deletion, files are not copied over, but instead they are linked (usually via a hardlink). That's why these two tasks are separated in the transaction logic: you don't need to download and extract packages that are already in the cache; you only need to link them!

Transactions also drive reports

The type and number of actions can also be calculated by `_make_legacy_action_groups()`, which returns a list of *action groups* (one per `PrefixSetup`). Each action group is a just a dictionary following this specification:

```
{
  "FETCH": Iterable[PackageRecord],  # estimated by `ProgressiveFetchExtract`
  "PREFIX": str,
  "UNLINK": Iterable[PackageRecord],
  "LINK: Iterable[PackageRecord],
}
```

These simpler action groups are only used for reporting, either via a processed text report (via `print_transaction_summary`) or just the raw JSON (via `stdout_json_success`). As you can see, they do not know anything about other types of tasks.

Download and extraction

`conda` maintains a cache of downloaded tarballs and their extracted contents to save disk space and improve the performance of environment modifications. This requires some code to check whether a given `PackageRecord` is already present in the cache, and, if it's not, how to download the tarball and extract its contents in a performant way. This is all handled by the `ProgressiveFetchExtract` class, which can instantiate up to two `Action` objects for each passed `PackageRecord`:

- `CacheUrlAction`: downloads (if remote) or copies (if local) a tarball to the cache location.
- `ExtractPackageAction`: extracts the contents of the tarball.

These two actions only take place *if* the package is not in cache yet and if it has already been extracted, respectively. They can also revert the changes if the transaction is aborted (either due to an error or because the user pressed Ctrl+C).

Populating the prefix

When all the necessary packages have been downloaded and extracted to the cache, it is time to start populating the prefix with the needed files. This means we need to:

1. For each package that needs to be unlinked, run the pre-unlink logic (`deactivate` and `pre-unlink` scripts, as well as shortcut removal, if needed) and then unlink the package files.
2. For each package that needs to be linked, create the links and run the post-link logic (`post-link` and `activate` scripts, as well as creating the shortcuts, if needed).

> Note that when you are updating a package version, you are actually removing the installed version entirely and then adding the new one. In other words, an update is just unlink+link.

How is this implemented? For each `PrefixSetup` object passed to `UnlinkLinkTransaction`, a number of `ActionGroup` namedtuples (one per task *category*) will be instantiated and grouped together in a `PrefixActionGroup`. These are then passed to `.verify()`. This method will take each action, run its checks and, if all of them passed, will allow us to perform the actual execution in `.execute()`. If one of them fails, the transaction can be aborted and rolled back.

For all this to work, each action object follows the `PathAction` API contract:

```
class PathAction:
    _verified = False

    def verify(self):
        "Run checks to assess if the action can proceed"

    def execute(self):
        "Perform the action"

    def reverse(self):
        "Undo execute"

    def cleanup(self):
        "Remove artifacts from verification, execution or reversal"

    @property
    def verified(self):
        "True if verification was run and successful"
```

Additional `PathAction` subclasses will add more methods and properties, but this is what the transaction execution logic expects. To support all the different actions involved in populating the prefix, the `PathAction` class tree holds quite the graph:

```
PathAction
  PrefixPathAction
    CreateInPrefixPathAction
      LinkPathAction
        PrefixReplaceLinkAction
      MakeMenuAction
      CreatePythonEntryPointAction
      CreatePrefixRecordAction
      UpdateHistoryAction
    RemoveFromPrefixPathAction
      UnlinkPathAction
        RemoveLinkedPackageRecordAction
      RemoveMenuAction
  RegisterEnvironmentLocationAction
  UnregisterEnvironmentLocationAction
  CacheUrlAction
  ExtractPackageAction

MultiPathAction
  CompileMultiPycAction
    AggregateCompileMultiPycAction
```

(continues on next page)

(continued from previous page)

You are welcome to read on the docstring for each of those classes to understand which each one is doing; all of them are listed under `conda.core.path_actions`. In the following sections, we will only comment on the most important ones.

Linking the files in the environment

When conda *links* a file from the cache location to the prefix location, it can actually mean three different actions:

1. Creating a soft link
2. Creating a hard link
3. Copying the file

The difference between soft links and hard links is subtle, but important. You can find more info on the differences elsewhere (e.g. here), but for our purposes it means that:

- Hard links are cheaper to resolve, behave like a real file, but can only link files in the same mount point.
- Soft links can link files across mount points, but they don't behave exactly like files (more like forwarders), so it's possible that they break assumptions made in certain pieces of code.

Most of the time, `conda` will try to hard link files and, if that fails, it will copy them over. Copying a file is an expensive disk operation, both in terms of time and space, so it should be the last option. However, sometimes it's the only way. Especially, when the file needs to be modified to be used in the target prefix.

Ummm... what? Why would `conda` modify a file to install it? This has to do with relocatability. When a `conda` package is created, `conda-build` creates up to three temporary environments:

- Build environment: where compilers and other build tools are installed, separate from the host environment to support cross-compilation.
- Host environment: where build-time dependencies are installed, together with the package you are building.
- Test environment: where run-time dependencies are installed, together with the package you just built. It simulates what will happen when a user installs the package so you can run arbitrary checks on your package.

When you are building a package, references to the build-time paths can leak into the content of some files, both text and binary. This is not a problem for users who build their own packages from source, since they can choose this path and leave the files there. However, this is almost never true for `conda` packages. They are created in one machine and installed in another. To avoid "path not found" issues and other problems, `conda-build` marks those packages that hold references to the build-time paths by replacing them with placeholders. At install-time, `conda` will replace those placeholders with the target prefix and everything works!

But there's a problem: we can't modify the files on the cache location because they might be used across environments (with obviously different paths). In these cases, files are not linked, but copied; the path replacement only happens on the target copy, of course!

How does `conda` know how to link a given package or, more precisely, its extracted files? All of this is determined in the preparation routines contained in `UnlinkLinkTransaction._prepare()` (more specifically, through `determine_link_type()`), as well as `LinkPathAction.create_file_link_actions()`.

Note that the (un)linking actions also include the execution of pre-(un)link and post-(un)link scripts, if listed.

Action groups and actions, in detail

Once the old packages have been removed and the new ones have been linked through the appropriate means, we are done, right? Not yet! There's one step left: the post-linking logic.

It turns out that there's a number of smaller tasks that need to happen to make `conda` as convenient as it is. You can find all of them listed a few paragraphs above, but we'll cover them here, too. The execution order is determined in `UnlinLinkTransaction._execute`. All the possible groups are listed under `PrefixActionGroup`. Their order is roughly how they happen in practice:

1. `remove_menu_action_groups`, composed of `RemoveMenuAction` actions.
2. `unlink_action_groups`, includes `UnlinkPathAction`, `RemoveLinkedPackageRecordAction`, as well as the logic to run the pre- and post-unlink scripts.
3. `unregister_action_groups`, basically a single `UnregisterEnvironmentLocationAction` action.
4. `link_action_groups`, includes `LinkPathAction`, `PrefixReplaceLinkAction`, as well as the logic to run pre- and post-link scripts.
5. `entry_point_action_groups`, a collection of `CreatePythonEntryPointAction` actions.
6. `register_action_groups`, a single `RegisterEnvironmentLocationAction` action.
7. `compile_action_groups`, several `CompileMultiPycAction` that end up aggregated as a `AggregateCompileMultiPycAction` for performance.
8. `make_menu_action_groups`, composed of `MakeMenuAction` actions.
9. `prefix_record_groups`, records installed packages in the environment via `CreatePrefixRecordAction` actions.
10. `initial_action_groups`, includes any plugin-defined pre-transaction actions.
11. `final_action_groups`, includes any plugin-defined post-transaction actions.

Let's discuss these actions groups for the command we are describing in this guide: `conda install numpy`. The solution given by the solver says we need to:

- unlink Python 3.9.6
- link Python 3.9.9
- link numpy 1.19

This is what would happen:

1. No menu items are removed because Python 3.9.6 didn't create any.
2. Pre-unlink scripts for Python 3.9.6 would run, but in this case there are none.
3. Python 3.9.6 files are removed from the environment. This can be parallelized.
4. Post-unlink scripts are run, if any.
5. Pre-link scripts are run for Python 3.9.9 and numpy 1.19, if any.
6. Files in the Python 3.9.9 and numpy 1.19 packages are linked and/or copied to the prefix. This can be parallelized.
7. Entry points are created for the new packages, if any.
8. Post-link scripts are run.
9. `pyc` files are generated for the new packages.
10. The new packages are registered under `conda-meta/`.
11. The menu shortcuts are created for the new packages, if any.

Any of these steps can fail with a given exception. If that's the case, the first of those exceptions is printed to STDOUT. Additionally, if `rollback_enabled` is properly configured in the `context`, the transaction will be rolled back by calling the `.reverse()` method in each action, from last to first.

If no exceptions are reported, then the actions can run their cleanup routines.

And that's it! If this command had resulted in a new environment being created, you would get a message telling you how to activate the newly created environment.

Conclusion

This is what happens when you type `conda install`. It might be a bit more involved than you initially thought, but it all boils down to only some steps. TL;DR:

1. Parse arguments and initialize the context
2. Download and build the index
3. Tell the solver what we want
4. Convert the solution into a transaction
5. Verify and run each action contained in the transaction

`conda init` and `conda activate`

`conda` ships *virtual environments* by design. When you install Anaconda or Miniconda, you obtain a `base` environment that is essentially a regular environment with some extra checks. These checks have to do with what the `conda` command really is and how it is installed in your system.

Base prefix vs target prefix

Originally, the base installation for `conda` was called the *root* environment. Every other environment lived under `envs/` in that root environment. The root environment was later renamed to *base*, but the code still distinguishes between base and target using the old terminology:

- `context.root_prefix`: the path where the `base` conda installation is located.
- `context.target_prefix`: the environment `conda` is running a command on. Usually defaults to the activated environment, unless `-n` (name) or `-p` (prefix) is specified in the command line. Note that if you are operating on the `base` environment, the target prefix will have the same value as the root prefix.

When you type `conda` in your terminal, your shell will try to find either:

- a shell function named `conda`
- an executable file named `conda` in your `PATH` directories

If your `conda` installation has been properly initialized, it will find the shell function. If not, it might find the `conda` executable if it *happens* to be in `PATH`, but this is most often not the case. That's why initialization is there to begin with!

Conda initialization

Why is initialization needed at all to begin with? There are several reasons:

- Activation requires interacting with the shell context very closely
- It does not pollute `PATH` unnecessarily

- Improves performance in certain operations

The main idea behind initialization is to provide a `conda` shell function that allows the Python code to interact with the shell context more intimately. It also allows a cleaner `PATH` manipulation and snappier responses in some `conda` commands.

The `conda` shell function is mainly a forwarder function. It will delegate most of the commands to the real `conda` executable driven by the Python library. However, it will intercept two very specific subcommands:

- `conda activate`
- `conda deactivate`

This interception is needed because activation/deactivation requires exporting (or unsetting) environment variables back to the shell session (and not just temporarily in the Python process). This will be discussed in the next section.

So how is initialization performed? This is the job of the `conda init` subcommand, driven by the `conda.cli.main_init` module, which depends direcly on the `conda.core.initialize` module. Let's see how this is implemented.

`conda init` will initialize a shell permanently by writing some shell code in the relevant startup scripts of your shell (e.g. `~/.bashrc`). This is done through different functions defined in `conda.core.initialize`, namely:

- `init_sh_user`: initializes a Posix shell (like Bash) for the current user.
- `init_sh_system`: initializes a Posix shell (like Bash) globally, for all users.
- `init_fish_user`: initializes the Fish shell for the current user.
- `init_xonsh_user`: initializes the Xonsh shell for the current user.
- `init_cmd_exe_registry`: initializes Cmd.exe through the Windows Registry.
- `init_powershell_user`: initializes Powershell for the current user.
- `init_long_path`: configures Windows to support longer paths.

What each function does depends on the nature of each shell. In the case of Bash shells, the underlying `Activator` subclass (more below) can generate the hook code dynamically. In other Posix shells and Powershell, a script is sourced from its location in the `base` environment. With Cmd, the changes are introduced through the Windows Registry. The end result is the same: they will end up defining a `conda` shell function with the behavior described above.

Conda activate

All `Activator` classes can be found under `conda.activate`. Their job is essentially to write shell-native code programmatically. As of conda 4.11, these are the supported shells and their corresponding activators

- `posix`, `ash`, `bash`, `dash`, `zsh`: all driven by `PosixActivator`.
- `csh`, `tcsh`: `CshActivator`.
- `xonsh`: `XonshActivator`.
- `cmd.exe`: `CmdExeActivator`.
- `fish`: `FishActivator`.
- `powershell`: `PowerShellActivator`.

You can add all these classes through the `conda shell.<key>` command, where `key` is any of the names in the list above. These CLI interface offers several subcommands, connected directly to methods of the same name:

- `activate`: writes the shell code to activate a given environment.
- `deactivate`: writes the shell code to deactivate a given environment.

- `hook`: writes the shell code to register the initialization code for the `conda` shell code.
- `commands`: writes the shell code needed for autocompletion engines.
- `reactivate`: writes the shell code for deactivation followed by activation.

To be clear, we are saying these functions only *write* shell code. They do *not* execute it! This needs to be done by the shell itself! That's why we need a `conda` shell function, so these shell strings can be `eval`'d or `source`'d in-session.

Let's see what happens when you run `conda shell.bash activate`:

```
$ conda shell.bash activate
export PATH='/Users/username/.local/anaconda/bin:/opt/homebrew/bin:/opt/homebrew/sbin:/
↪usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin:/Users/username/.local/anaconda/condabin:/
↪opt/homebrew/bin:/opt/homebrew/sbin'
unset CONDA_PREFIX_1
PS1='(base) '
export CONDA_PREFIX='/Users/username/.local/anaconda'
export CONDA_SHLVL='1'
export CONDA_DEFAULT_ENV='base'
export CONDA_PROMPT_MODIFIER='(base) '
export CONDA_EXE='/Users/username/.local/anaconda/bin/conda'
export _CE_M=''
export _CE_CONDA=''
export CONDA_PYTHON_EXE='/Users/username/.local/anaconda/bin/python'
```

`_CE_M` and `_CE_CONDA` are empty here (a no-op). The shell still expands them so `python -m conda` works when they are set; conda itself will stop exporting them except when `--dev` is passed. For local development, set `PYTHONPATH` to the conda source root (see `dev/start`) instead of using `--dev`.

See? It only wrote some shell code to stdout, but it wasn't executed. We would need to do this to actually run it:

```
$ eval "$(conda shell.bash activate)"
```

And this is essentially what `conda activate` does: it calls the registered shell activator to obtain the required shell code and then it `eval`s it. In some shells with no `eval` equivalent, a temporary script is written and sourced or called. The final effect is the same.

Ok, but what is that shell code doing? Mainly setting your `PATH` correctly so the executables of your `base` environment can be found (like `python`). It also sets some extra variables to keep a reference to the path of the currently active environment, the shell prompt modifiers and other information for `conda` internals.

This command can also generate the code for any other environment you want, not just `base`. Just pass the name or path:

```
$ conda shell.bash activate mamba-poc
PS1='(mamba-poc) '
export PATH='/Users/username/.local/anaconda/envs/mamba-poc/bin:/opt/homebrew/bin:/opt/
↪homebrew/sbin:/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin:/Users/username/.local/
↪anaconda/condabin:/opt/homebrew/bin:/opt/homebrew/sbin'
export CONDA_PREFIX='/Users/username/.local/anaconda/envs/mamba-poc'
export CONDA_SHLVL='2'
export CONDA_DEFAULT_ENV='mamba-poc'
export CONDA_PROMPT_MODIFIER='(mamba-poc) '
export CONDA_EXE='/Users/username/.local/anaconda/bin/conda'
export _CE_M=''
export _CE_CONDA=''
```

(continues on next page)

(continued from previous page)

```
export CONDA_PYTHON_EXE='/Users/username/.local/anaconda/bin/python'
export CONDA_PREFIX_1='/Users/username/.local/anaconda'
```

Now the paths are different, as well as some numbers (e.g. `CONDA_SHLVL`). This is used by conda to keep track of what was activated before, so when you deactivate the last one, you can get back to the previous one seamlessly.

Activation/deactivation scripts

The activation/deactivation code can also include calls to activation/deactivation scripts. If present in the appropriate directories for your shell (e.g. `CONDA_PREFIX/etc/conda/activate.d/`), they will be called before deactivation or after activation, respectively. For example, compilers usually set up some environment variables to help configure the default flags. This is what happens when you activate an environment that contains Clang and Gfortran:

```
$ conda shell.bash activate compilers
PS1='(compilers) '
export PATH='/Users/username/.local/anaconda/envs/compilers/bin:/opt/homebrew/bin:/opt/
↪homebrew/sbin:/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin:/Users/username/.local/
↪anaconda/condabin:/opt/homebrew/bin:/opt/homebrew/sbin'
export CONDA_PREFIX='/Users/username/.local/anaconda/envs/compilers'
export CONDA_SHLVL='2'
export CONDA_DEFAULT_ENV='compilers'
export CONDA_PROMPT_MODIFIER='(compilers) '
export CONDA_EXE='/Users/username/.local/anaconda/bin/conda'
export _CE_M=''
export _CE_CONDA=''
export CONDA_PYTHON_EXE='/Users/username/.local/anaconda/bin/python'
export CONDA_PREFIX_1='/Users/username/.local/anaconda'
. "/Users/username/.local/anaconda/envs/compilers/etc/conda/activate.d/activate-gfortran_
↪osx-arm64.sh"
. "/Users/username/.local/anaconda/envs/compilers/etc/conda/activate.d/activate_clang_
↪osx-arm64.sh"
. "/Users/username/.local/anaconda/envs/compilers/etc/conda/activate.d/activate_clangxx_
↪osx-arm64.sh"
```

Those three lines are sourcing the relevant scripts. Similarly, for deactivation, notice how the deactivation scripts are executed first this time:

```
$ conda shell.bash deactivate
export PATH='/Users/username/.local/anaconda/bin:/opt/homebrew/bin:/opt/homebrew/sbin:/
↪usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin:/Users/username/.local/anaconda/condabin:/
↪opt/homebrew/bin:/opt/homebrew/sbin'
. "/Users/username/.local/anaconda/envs/compilers/etc/conda/deactivate.d/deactivate_
↪clangxx_osx-arm64.sh"
. "/Users/username/.local/anaconda/envs/compilers/etc/conda/deactivate.d/deactivate_
↪clang_osx-arm64.sh"
. "/Users/username/.local/anaconda/envs/compilers/etc/conda/deactivate.d/deactivate-
↪gfortran_osx-arm64.sh"
unset CONDA_PREFIX_1
PS1='(base) '
export CONDA_PREFIX='/Users/username/.local/anaconda'
export CONDA_SHLVL='1'
export CONDA_DEFAULT_ENV='base'
```

(continues on next page)

(continued from previous page)

```
export CONDA_PROMPT_MODIFIER='(base) '
export CONDA_EXE='/Users/username/.local/anaconda/bin/conda'
export _CE_M=''
export _CE_CONDA=''
export CONDA_PYTHON_EXE='/Users/username/.local/anaconda/bin/python'
```

`conda config` and context

The `context` object is central to many parts of the `conda` codebase. It serves as a centralized repository of settings. You normally import the singleton and access its (many) attributes directly:

```
from conda.base.context import context

context.quiet
# False
```

This singleton is initialized from a cascade of different possible sources. From lower to higher precedence:

1. Default values hardcoded in the `Context` class. These are defined via class attributes.
2. Values defined in the configuration files (`.condarc`), which have their own *precedence*.
3. Values set by the corresponding command line arguments, if any.
4. Values defined by their corresponding `CONDA_*` environment variables, if present.

The mechanism implementing this behavior is an elaborate object with several types of objects involved.

Anatomy of the `Context` class

`conda.base.context.Context` is an conda-specific subclass of the application-agnostic `conda.common.configuration.Configuration` class. This class implements the precedence order for the instantiation of each defined attribute, as well as the overall validation logic and help message reporting. But that's it, it's merely a storage of `ParameterLoader` objects which, in turn, instantiate the relevant `Parameter` subclasses in each attribute. Roughly:

```
class MyConfiguration(Configuration):
    string_field = ParameterLoader(PrimitiveParameter("default", str))
    list_of_int_field = ParameterLoader(SequenceParameter([1, 2, 3], int))
    map_of_foat_values_field = ParameterLoader(MapParameter({"key": 1.0}, float))
```

When `MyConfiguration` is instantiated, those class attributes are populated by the `.raw_data` dicionary that has been filled in with the values coming from the precedence chain stated above. The `raw_data` dictionary contains `RawParameter` objects, subclassed to deal with the specifics of their origin (YAML file, environment variable, command line flag). Each `ParameterLoader` object will pass the `RawParameter` object to the `.load()` method of its relevant `Parameter` subclass, which are designed to return their corresponding `LoadedParameter` object counterpart.

It's a bit confusing, but the delegation happens like this:

1. The `Configuration` subclass parses the raw values of the possible origins and stores them as the relevant `RawParameter` objects, which can be:
   - `EnvRawParameter`: for those coming from an environment variable
   - `ArgParseRawParameter`: for those coming from a command line flag
   - `YamlRawParameter`: for those coming from a configuration file
   - `DefaultValueRawParameter`: for those coming from the default value given to `ParameterLoader`

2. Each `Configuration` attribute is a `ParameterLoader`, which implements the `property` protocol via `__get__`. This means that, upon attribute access (e.g. `MyConfiguration.string_field`), the `ParameterLoader` can execute the loading logic. This means finding potential type matches in the raw data, loading them as `LoadedParameter` objects and merging them with the adequate precedence order.

The merging policy depends on the `(Loaded)Parameter` subtype. Below is a list of available subtypes:

- `PrimitiveParameter`: holds a single scalar value of type `str`, `int`, `float`, `complex`, `bool` or `NoneType`.
- `SequenceParameter`: holds an iterable (`list`) of other `Parameter` objects.
- `MapParameter`: holds a mapping (`dict`) of other `Parameter` objects.
- `ObjectParameter`: holds an object with attributes set to `Parameter` objects.

The main goal of the `Parameter` objects is to implement how to typify and turn the raw values into their `Loaded` counterparts. These implement the validation routines and define how parameters for the same key should be merged:

- `PrimitiveLoadedParameter`: value with highest precedence replaces the existing one.
- `SequenceLoadedParameter`: extends with no duplication, keeping precedence.
- `MapLoadedParameter`: cascading updates, highest precedence kept.
- `ObjectLoadedParameter`: same as `Map`.

After all of this, the `LoadedParameter` objects are *typified*: this is when type validation is performed. If everything goes well, you obtain your values just fine. If not, the validation errors are raised.

Take into account that the result is cached for faster subsequent access. This means that even if you change the value of the environment variables responsible for a given setting, this won't be reflected in the `context` object until you refresh it with `conda.base.context.reset_context()`.

Do not modify the Context object!

`ParameterLoader` does not implement the `__set__` method of the `property` protocol, so you can freely override an attribute defined in a `Configuration` subclass. You might think that this will redefine the value after passing through the validation machinery, but that's not true. You will simply overwrite it entirely with the raw value and that's probably not what you want.

Instead, consider the `context` object immutable. If you need to change a setting at runtime, it is probably *A Bad Idea*. The only situation where this is acceptable is during testing.

Setting values in the different origins

There's some magic behind the possible origins for the settings values. How these are tied to the final `Configuration` object might not be obvious at first. This is different for each `RawParameter` subclass:

- `DefaultValueRawParameter`: Users will never see this one. It only wraps the default value passed to the `ParameterLoader` class. Safe to ignore.
- `YamlRawParameter`: This one takes a YAML file and parses it as a dictionary. The keys in this file must match the attribute names in the `Configuration` class exactly (or one of their aliases). Matching happens automatically once this is properly set up. How the values are parsed depends on the YAML Loader, set internally by `conda`.
- `EnvRawParameter`: Values coming from certain environment variables can make it to the `Configuration` instance, provided they are formatted as `<APP_NAME>_<PARAMETER_NAME>`, all uppercase. The app name is defined by the `Configuration` subclass. The parameter name is defined by the attribute name in the class, transformed to upper case. For example, `context.ignore_pinned` can be set with `CONDA_IGNORE_PINNED`. The value of the variable is parsed in different ways depending on the type:

  - `PrimitiveParameter` is the easy one. The environment variable string is parsed as the expected type. Booleans are a bit different since several strings are recognized as such, and in a case-insensitive way:
    * `True` can be set with `true`, `yes`, `on` and `y`.
    * `False` can be set with `false`, `off`, `n`, `no`, `non`, `none` and `""` (empty string).
  - `SequenceParameter` can specify their own delimiter (e.g. `,`), so the environment variable string is processed into a list.
  - `MapParameter` and `ObjectParameter` do not support being set with environment variables.
- `ArgParseRawParameter`: These are a bit different because there is no automated mechanism that ties a given command line flag to the context object. This means that if you add a new setting to the `Context` class and you want that available in the CLI as a command line flag, you have to add it yourself. If that's the case, refer to `conda.cli.conda_argparse` and make sure that the `dest` value of your `argparse.Argument` matches the attribute name in `Context`. This way, `Configuration.__init__` can take the `argparse.Namespace` object, turn it into a dictionary, and make it pass through the loading machinery.

Programmatic `.condarc` File API

This guide explains how to programmatically read and write conda configuration files (`.condarc`) using the `ConfigurationFile` class. This is useful for tools that need to modify conda settings without shelling out to `conda config` commands.

Overview

The `conda.cli.condarc.ConfigurationFile` class provides a high-level interface for reading, modifying, and writing conda configuration files (`.condarc`). It handles:

- Configuration file validation
- Type checking and conversion
- Parameter existence checking
- Atomic file operations via context manager
- Support for sequence, map, and primitive parameters

Basic Usage

Reading Configuration

```
from conda.cli.condarc import ConfigurationFile

# Read user's .condarc file
config = ConfigurationFile.from_user_condarc()

# Access the configuration content
print(config.content)
# {'channels': ['defaults', 'conda-forge'], 'auto_update_conda': False}

# Get a specific key
key, value = config.get_key("channels")
print(f"{key}: {value}")
# channels: ['defaults', 'conda-forge']
```

Writing Configuration

```
from conda.cli.condarc import ConfigurationFile

# Create a configuration file instance
config = ConfigurationFile.from_user_condarc()

# Set a primitive parameter
config.set_key("auto_update_conda", False)

# Add to a sequence parameter
config.add("channels", "conda-forge", prepend=True)

# Set a map parameter
config.set_key("proxy_servers.http", "http://proxy.example.com")

# Write changes to file
config.write()
```

Using Context Manager

For atomic operations, use the context manager pattern:

```
from conda.cli.condarc import ConfigurationFile

# Changes are automatically written on successful exit
with ConfigurationFile.from_user_condarc() as config:
    config.set_key("channels", ["conda-forge", "defaults"])
    config.set_key("auto_update_conda", False)
# File is written here automatically
```

If an exception occurs within the context, changes are not written:

```
try:
    with ConfigurationFile.from_user_condarc() as config:
        config.set_key("channels", ["conda-forge"])
        raise ValueError("Something went wrong")
except ValueError:
    pass
# File was NOT modified because of the exception
```

Factory Methods

The `ConfigurationFile` class provides several factory methods for common configuration file locations:

User Configuration

```
from conda.cli.condarc import ConfigurationFile

# User's .condarc file (typically ~/.condarc)
config = ConfigurationFile.from_user_condarc()
```

System Configuration

```
from conda.cli.condarc import ConfigurationFile

# System-wide .condarc file
config = ConfigurationFile.from_system_condarc()
```

Environment Configuration

```
from conda.cli.condarc import ConfigurationFile

# Environment-specific .condarc file at {prefix}/.condarc
config = ConfigurationFile.from_env_condarc(prefix="/path/to/env")

# Or use CONDA_PREFIX environment variable
config = ConfigurationFile.from_env_condarc()
```

Custom Path

```
from pathlib import Path
from conda.cli.condarc import ConfigurationFile

# Custom configuration file path
config = ConfigurationFile(path=Path("/custom/path/.condarc"))
```

Parameter Types

Conda configuration parameters come in three types, and each supports different operations:

Primitive Parameters

Single scalar values (strings, numbers, booleans):

```
config.set_key("auto_update_conda", False)
config.set_key("channel_priority", "strict")
config.set_key("rollback_enabled", True)
```

Sequence Parameters

Lists of values:

```
# Add to end of list
config.add("channels", "conda-forge", prepend=False)

# Add to beginning of list
config.add("channels", "defaults", prepend=True)

# Remove from list
config.remove_item("channels", "conda-forge")
```

Map Parameters

Dictionaries with nested values:

```
# Set a map entry
config.set_key("proxy_servers.http", "http://proxy.example.com")
config.set_key("proxy_servers.https", "https://proxy.example.com")

# Add to a nested sequence within a map
config.add("conda_build.config_file", "/path/to/config.yaml")
```

Key Validation

The `ConfigurationFile` class validates keys against the current conda context:

```
from conda.cli.condarc import ConfigurationFile

config = ConfigurationFile.from_user_condarc()

# Check if a key exists
if config.key_exists("channels"):
    print("channels is a valid parameter")

# Attempting to set an invalid key raises an error
try:
    config.set_key("invalid_key", "value")
except Exception as e:
    print(f"Error: {e}")
    # Error: CondaKeyError: 'invalid_key': unknown parameter
```

Handling Missing Keys

When getting a key that doesn't exist, the method returns a sentinel value:

```
from conda.cli.condarc import ConfigurationFile, MISSING

config = ConfigurationFile(content={})

key, value = config.get_key("undefined_key")
if value is MISSING:
    print(f"Key '{key}' not found in config")
```

Working with Plugin Configuration

Plugin parameters use a `plugins.` prefix:

```
# Set a plugin-specific parameter
config.set_key("plugins.custom_solver.enabled", True)

# Add to a plugin sequence parameter
config.add("plugins.custom_reporters.backends", "custom_backend")
```

Advanced Usage

Custom Context

You can provide a custom context instance for testing or specialized configurations:

```
from conda.base.context import Context
from conda.cli.condarc import ConfigurationFile

# Create a custom context
custom_context = Context()

# Use it with ConfigurationFile
config = ConfigurationFile(path="/path/to/config", context=custom_context)
```

Warning Handlers

Customize how warnings are reported:

```
warnings = []


def collect_warnings(msg):
    warnings.append(msg)


config = ConfigurationFile(path="/path/to/config", warning_handler=collect_warnings)

config.add("channels", "defaults", prepend=False)
# If "defaults" already exists, a warning is collected
print(warnings)
```

Manual Read/Write Control

For more control over when files are read or written:

```
from conda.cli.condarc import ConfigurationFile

config = ConfigurationFile(path="/path/to/.condarc")

# Explicitly read
config.read()

# Make changes
config.content["channels"] = ["conda-forge"]

# Explicitly write
config.write()

# Or write to a different location
config.write(path="/different/path/.condarc")
```

Working with Content Directly

The `content` property provides direct access to the underlying configuration dictionary:

```
config = ConfigurationFile.from_user_condarc()

# Access content
current_channels = config.content.get("channels", [])

# Modify content directly (use with caution)
config.content["channels"] = ["conda-forge", "defaults"]
config.write()
```

Note: Direct content manipulation bypasses validation. Use the provided methods (`set_key`, `add`, `remove_item`, etc.) for safer operations.

Complete Example

Here's a complete example that demonstrates multiple operations:

```
from conda.cli.condarc import ConfigurationFile

# Use context manager for atomic operations
with ConfigurationFile.from_user_condarc() as config:
    # Set primitive parameters
    config.set_key("auto_update_conda", False)
    config.set_key("channel_priority", "strict")

    # Configure channels
    config.set_key("channels", [])  # Clear existing
    config.add("channels", "conda-forge", prepend=False)
    config.add("channels", "defaults", prepend=False)

    # Set proxy servers
    config.set_key("proxy_servers.http", "http://proxy.example.com:8080")
    config.set_key("proxy_servers.https", "https://proxy.example.com:8080")

    # Configure conda-build
    config.add("conda_build.config_file", "/path/to/conda_build_config.yaml")

    # Print current configuration
    print("Current configuration:")
    for key, value in config.content.items():
        print(f"  {key}: {value}")

# File is automatically written when exiting the context manager
print("Configuration saved!")
```

Migration Guide

If you were previously using the private functions from `conda.cli.main_config`, here's how to migrate:

Before (deprecated)

```
from conda.cli.main_config import (
    _read_rc,
    _write_rc,
    _set_key,
    _get_key,
    _remove_key,
)

# Old way
rc_config = _read_rc("/path/to/.condarc")
_set_key("auto_update_conda", False, rc_config)
_write_rc("/path/to/.condarc", rc_config)
```

After (recommended)

```
from conda.cli.condarc import ConfigurationFile

# New way
config = ConfigurationFile(path="/path/to/.condarc")
config.set_key("auto_update_conda", False)
config.write()

# Or even simpler with context manager
with ConfigurationFile(path="/path/to/.condarc") as config:
    config.set_key("auto_update_conda", False)
```

Error Handling

The `ConfigurationFile` class raises specific exceptions for different error conditions:

```
from conda.cli.condarc import ConfigurationFile
from conda.exceptions import CondaKeyError, CondaValueError, CouldntParseError

config = ConfigurationFile.from_user_condarc()

try:
    # Unknown parameter
    config.set_key("unknown_param", "value")
except CondaKeyError as e:
    print(f"Invalid key: {e}")

try:
    # Invalid operation for parameter type
    config.set_key("channels", "not-a-list")
except CondaKeyError as e:
    print(f"Invalid operation: {e}")

try:
    # Adding to a non-sequence parameter
    config.add("auto_update_conda", "value")
```

(continues on next page)

(continued from previous page)

```
except CondaValueError as e:
    print(f"Type error: {e}")
```

Thread Safety

The `ConfigurationFile` class is not thread-safe. If you need to access configuration from multiple threads, consider using locks or creating separate instances for each thread.

Best Practices

1. **Use context managers**: For atomic operations, always use the context manager pattern to ensure changes are only written on success.
2. **Validate keys**: Use `key_exists()` to check parameter validity before attempting operations.
3. **Use factory methods**: Prefer `from_user_condarc()`, `from_system_condarc()`, and `from_env_condarc()` over manual path construction.
4. **Handle missing keys**: Always check for `MISSING` when using `get_key()` to handle undefined parameters gracefully.
5. **Avoid direct content manipulation**: Use the provided methods (`set_key`, `add`, etc.) instead of modifying `content` directly to ensure proper validation.
6. **Custom warning handlers**: For library code, provide custom warning handlers to integrate with your logging system.

See Also

- *conda config and context* - Understanding the conda context object
- *Configuration* - User-facing configuration documentation
- *cli* - CLI API documentation

Solvers

The guide *conda install* didn't go into details of the solver black box. It did mention the high-level `Solver` API and how `conda` expects a transaction out of it, but we never got to learn what happens *inside* the solver itself. We only covered these three steps:

> The details are complicated, but in essence, the solver will:
>
> 1. Express the requested packages, command line options and prefix state as `MatchSpec` objects
> 2. Query the index for the best possible match that satisfy those constraints
> 3. Return a list of `PackageRecord` objects.

How do we transform the prefix state and configurations into a list of `MatchSpec` objects? How are those turned into a list of `PackageRecord` objects? Where are those `PackageRecord` objects coming from? We are going to cover these aspects in detail here.

`MatchSpec` vs `PackageRecord`

First, let's define what each object does:

- `PackageRecord` objects represent a concrete package tarball and its contents. They follow specific naming conventions and expose several fields. Inspect them directly in the class code.

- `MatchSpec` objects are essentially a query language to find `PackageRecord` objects. Internally, `conda` will translate your command line requests, like `numpy>=1.19`, `python=3.*` or `pytorch=1.8.*=*cuda*`, into instances of this class. This query language has its own syntax and rules, detailed here. The most important fields of a `MatchSpec` object are:
  - `name`: the name of the package (e.g. `pytorch`); always expected.
  - `version`: the version constraints (e.g. `1.8.*`); can be empty but if `build` is set, set it to `*` to avoid issues with the `.conda_build_form()` method.
  - `build`: the build string constraints (e.g. `*cuda*`); can be empty.

Create a MatchSpec object from a PackageRecord instance

You can create a `MatchSpec` object from a `PackageRecord` instance using the `.to_match_spec()` method. This will create a `MatchSpec` object with its fields set to exactly match the originating `PackageRecord`.

Note that there are two `PackageRecord` subclasses with extra fields, so we need to distinguish between three types, all of them useful:

- `PackageRecord`: A record as present in the index (channel).
- `PackageCacheRecord`: A record already extracted in the cache. Contains extra fields for the tarball path in disk and its extracted directory.
- `PrefixRecord`: A record installed in a prefix. Same as above, plus fields for the files that make the package and how they were linked in the prefix. It can also host information about which `MatchSpec` string resulted in this record being installed.

Remote state: the index

So the solver takes `MatchSpec` objects, queries the index for the best match and returns `PackageRecord` objects. Perfect. What's the index? It's the result of aggregating the requested conda channels in a single entity. For more information, check *Fetching the index*.

Excluding newer package records

The `exclude_newer` policy is not represented as a `MatchSpec`. Instead, it limits the candidate `PackageRecord` objects that may be considered by the solver. A record is included when its `indexed_timestamp`, or its `timestamp` if no `indexed_timestamp` is present, is at or before the effective cutoff. Records without either timestamp are kept for compatibility.

Different solver backends collect and reduce package records through different code paths, so this policy is part of the `Solver` API contract instead of only a `SubdirData` filter. A solver backend declares whether it supports global cutoffs, channel overrides, and package overrides. Conda raises an error before solving when the configured policy asks for a level that the selected backend does not support.

After solving, conda also validates the records that would be linked into the target prefix. This keeps the policy enforced even if a solver backend returns a package newer than the configured cutoff.

Local state: the prefix and context

When you do `conda install numpy`, do you think the solver will just see something like `specs=[MatchSpec("numpy")]`? Well, not that quick. The explicit instructions given by the user are only one part of the request we will send to the solver. Other pieces of implicit state are taken into account to build the final request. Namely, the state of your prefix. In total, these are the ingredients of the solver request:

1. Packages already present in your environment, if you are not *creating* a new one. This is exposed through the `conda.core.prefix_data.PrefixData` class, which provides an iterator method via `.iter_records()`. As we saw before, this yields `conda.models.records.PrefixRecord` objects, a `PackageRecord` subclass for installed records.
2. Past actions you have performed in that environment; the *History*. This is a journal of all the `conda install|update|remove` commands you have run in the past. In other words, the *specs* matched by those previous actions will receive extra protections in the solver.
3. Packages included in the *aggressive updates* list. These packages are always included in any requests to make sure they stay up-to-date under all circumstances.
4. Packages pinned to a specific version, either via `pinned_packages` in your `.condarc` or defined in a `$PREFIX/conda-meta/pinned` file.
5. In new environments, packages included in the `create_default_packages` list. These specs are injected in each `conda create` command, so the solver will see them as explicitly requested by the user.
6. And, finally, the specs the user is asking for. Sometimes this is explicit (e.g. `conda install numpy`) and sometimes a bit implicit (e.g. `conda update --all` is telling the solver to add all installed packages to the update list).

All of those sources of information produce a number a of `MatchSpec` objects, which are then combined and modified in very specific ways depending on the command line flags and their origin (e.g. specs coming from the pinned packages won't be modified, unless the user asks for it explicitly). This logic is intricate and will be covered in the next sections. A more technical description is also available in *Solver state*.

The high-level logic in `conda.cli.install`

The full solver logic does not start at the `conda.core.solve.Solver` API, but before that, all the way up in the `conda.cli.install` module. Here, some important decisions are already made:

- Whether the solver is not needed at all because:
  - The operation is an explicit package install
  - The user requested to roll back to a history checkpoint
  - We are just creating a copy of an existing environment (cloning)
- Which `repodata` source to use (see *here*). It not only depends on the current configuration (via `.condarc` or command line flags), but also on the value of `use_only_tar_bz2`.
- Whether the solver should start by freezing all installed packages (default for `conda install` and `conda remove` in existing environments).
- If the solver does not find a solution, whether we need to retry again without freezing the installed packages for the current `repodata` variant or if we should try with the next one.

So, roughly, the global logic there follows this pseudocode:

```
if operation in (explicit, rollback, clone):
    transaction = handle_without_solver()
else:
    repodatas = from_config or ("current_repodata.json", "repodata.json")
    freeze = (is_install or is_remove) and env_exists and update_modifier not in argv
    for repodata in repodatas:
        try:
            transaction = solve_for_transaction(...)
        except:
```

(continues on next page)

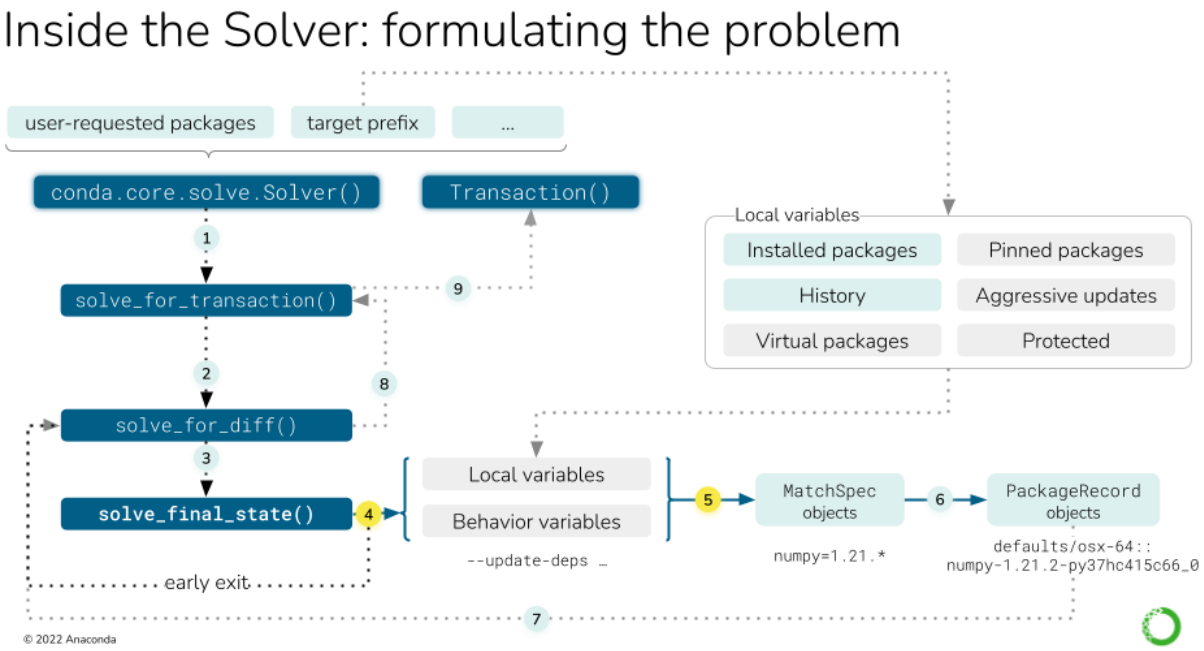


Fig. 2: Local variables affect the solving process explicitly and implicitly. As seen in *in the conda install deep dive*, the main actor is the `conda.core.solve.Solver` class. Before invoking the SAT solver, we can describe nine steps:

1. Instantiate the `Solver` class with the user-requested packages and the active environment (target prefix)
2. Call the `solve_for_transaction()` method on the instance, which calls `solve_for_diff()`.
3. Call `solve_final_state()`, which takes some more arguments from the CLI.
4. Under some circumstances, we can return early (e.g. the packages are already installed).
5. If we didn't return early, we collect all the local variables into a list of `MatchSpec` objects.

For steps six to nine, see *this figure*.

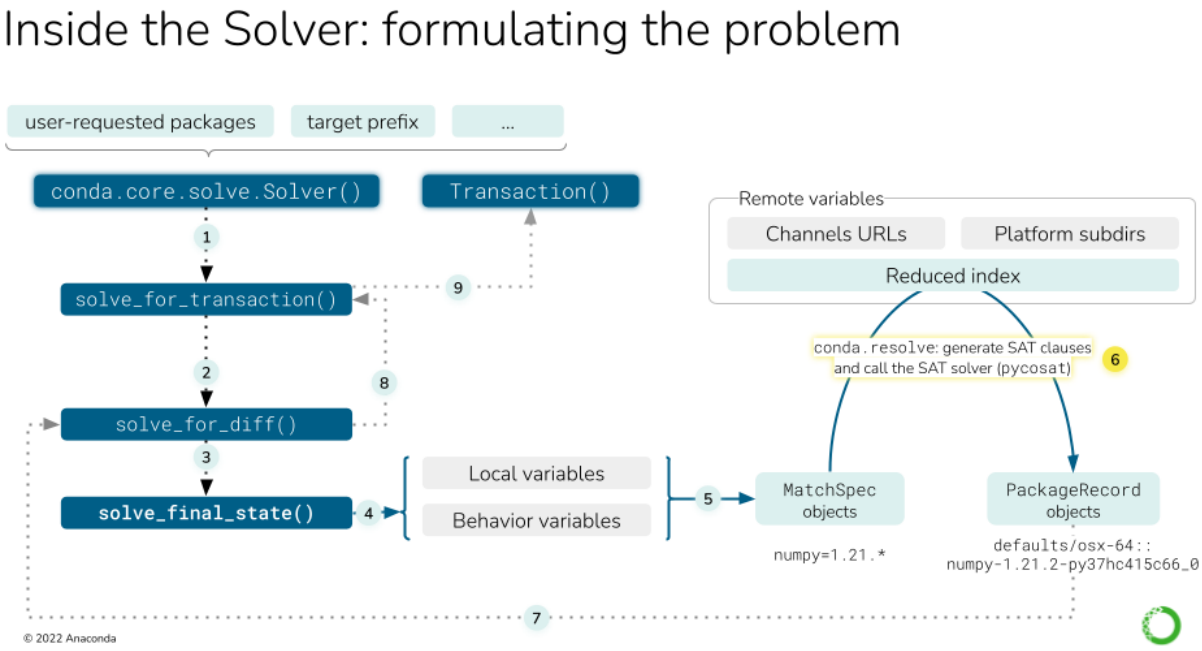


Fig. 3: The remote variables in a solve refer to, essentially, the package index (channels). This figure describes nine steps, focusing on 6-9. For steps 1-5, see *the previous figure*.

6. All the channels need to be fetched by now, but they have to be aggregated and reduced so the solver only handles the relevant bits. This step transforms "channels" into a list of available `PackageRecord` objects.
7. This is where the SAT solver will act. It will use the list of `MatchSpec` objects to pick a number of `PackageRecord` entries from the index, thus building the "final state of the solved environment". This is detailed later in this deep dive guide, if you need more info.
8. `solve_for_diff` takes the final state and compares it to the initial state, generating the differences between them (e.g. package A was updated to version 1.2, package B was removed).
9. `solve_for_transaction` takes the diff and some more metadata in the instance to generate the `Transaction` object.

(continued from previous page)

```
            if repodata is last:
                raise
            elif freeze:
                transaction = solve_for_transaction(freeze_installed=False)
            else:
                continue  # try next repodata

handle_txn(transaction)
```

Check *this other figure* for a schematic representation of this pseudocode.

We have, then, two reasons to re-run the full solver logic:

- Freezing the installed packages didn't work, so we try without freezing again.
- Using `current_repodata` did not work, so we try with full `repodata`.

These two strategies are stacked so in the end, before eventually failing, we will have tried four things:

1. Solve with `current_repodata.json` and `freeze_installed=True`
2. Solve with `current_repodata.json` and `freeze_installed=False`
3. Solve with `repodata.json` and `freeze_installed=True`
4. Solve with `repodata.json` and `freeze_installed=False`

Interestingly, those strategies are designed to improve `conda`'s average performance, but they should be seen as a risky bet. Those attempts can get expensive!

How to ask for a simpler approach

If you want to try the full thing without checking whether the optimized solves work, you can override the default behaviour with these flags in your `conda install` commands:

- `--repodata-fn=repodata.json`: do not use `current_repodata.json`
- `--update-specs`: do not try to freeze installed

Then, the `Solver` class has its own internal logic, which also features some retry loops. This will be discussed later and summarized.

Early exit tasks

Some tasks do not involve the solver at all. Let's enumerate them:

- Explicit package installs: no index or prefix state needed.
- Cloning an environment: the index might be needed if the cache has been cleared.
- History rollback: currently broken.
- Forced removal: prefix state needed. This happens in the `Solver` class.
- Skip solve if already satisfied: prefix state needed. This happens in the `Solver` class.

Explicit package installs

These commands do not need a solver because the requested packages are expressed with a direct URL or path to a specific tarball. Instead of a `MatchSpec`, we already have a `PackageRecord`-like entity! For this to work, all the requested packages neeed to be URLs or paths. They can be typed in the command line or in a text file including a `@EXPLICIT` line.

Since the solver is not involved, the dependencies of the explicit package(s) are not processed at all. This can leave the environment in an *inconsistent state*, which can be fixed by running `conda update --all`, for example.

Explicit installs are taken care of by the `explicit` function.

Cloning an environment

`conda create` has a `--clone` flag that allows you to create a fully-working copy of an existing environment. This is needed because you cannot relocate an environment using `cp`, `mv`, or your favorite file manager without unintended consequences. Some files in a conda environment might contain hardcoded paths to existing files in the original location, and those references will break if `cp` or `mv` is utilized (conda environments *can* be renamed via the `conda rename` command, however; see the *following section* for more information).

The `clone_env` function implements this functionality. It essentially takes the source environment, generates the URLs for each installed packages (filtering `conda`, `conda-env` and their dependencies) and passes the list of URLs to `explicit()`. If the source tarballs are not in the cache anymore, it will query the index for the best possible match for the current channels. As such, there's a slim chance that the copy is not exactly a clone of the original environment.

Renaming an environment

When the `conda rename` command is used to rename an already-existing environment, please keep in mind that the solver is not invoked at all, since the command essentially does a `conda create --clone` and `conda remove --all` of the environment.

History rollback

`conda install` has a `--revision` flag, which allows you to revert the state of the environment to a previous one. This is done through the `History` file, but its current implementation can be considered broken. Once fixed, we will cover it in detail.

Forced removals

Similar to explicit installs, you can remove a package without performing a full solve. If `conda remove` is invoked with `--force`, the specified package(s) will be removed directly, without analyzing their dependency tree and pruning the orphans. This can only happen after querying the active prefix for the installed packages, so it is handled in the `Solver` class. This part of the logic returns the list of `PackageRecord` objects already found in the `PrefixData` list after filtering out the ones that should be removed.

Skip solve if already satisfied

`conda install` and `update` have a rather obscure flag: `-S, --satisfied-skip-solve`:

> Exit early and do not run the solver if the requested specs are satisfied. Also skips aggressive updates as configured by 'aggressive_update_packages'. Similar to the default behavior of 'pip install'.

This is also implemented at the `Solver` level, because we also need a `PrefixData` instance. It essentially checks if all of the passed `MatchSpec` objects can match a `PackageRecord` already in prefix. If that's the case, we return the installed state as-is. If not, we proceed for the full solve.

Details of `Solver.solve_final_state()`

> **Note**
>
> From here on, the document only covers the classic solver logic (which uses `pycosat`). The `libmamba` solver has a different approach and is not documented here. Please refer to its documentation for more information.

> **Note**
>
> The classic solver, now named `pycosat` (`classic` remains a supported alias), is provided by the standalone `conda-pycosat-solver` plugin rather than being registered by conda itself. `conda-pycosat-solver` builds on `conda.core.solve.Solver`, so this deep dive remains a useful reference for its implementation.

This is where most of the intricacies of the `conda` logic are defined. In this step, the configuration, command line flags, user-requested specs and prefix state are aggregated to query the current index for the best match.

The aggregation of all those state bits will result in a list of `MatchSpec` objects. While it's easy to establish which package names will make it to the list, deciding which version and build string constraints the specs carry is a bit more involved.

This is currently implemented in the `conda.core.solve.Solver` class. Its main goal is to populate the `specs_map` dictionary, which maps package names (`str`) to `MatchSpec` objects. This happens at the beginning of the `.solve_final_state()` method. The full details of the `specs_map` population are covered in the *solver state technical reference*, but here's a little map of what submethods are involved:

1. Initialization of the `SolverStateContainer`: Often abbreviated as `ssc`, it's a helper class to store some state across attempts (remember there are several retry loops). Most importantly, it stores two key attributes (among others):
   - `specs_map`: same as above. This is where it lives across solver attempts.
   - `solution_precs`: a list of `PackageRecord` objects. It stores the solution returned by the SAT solver. It's always initialized to reflect the installed packages in the target prefix.
2. `Solver._collect_all_metadata()`: Initializes the `specs_map` with the specs found in the history or with the specs corresponding to the installed records. This method delegates to `Solver._prepare()`. This initializes the index by fetching the channels and reducing it. Then, a `conda.resolve.Resolve` instance is created with that index. The index is stored in the `Solver` instance as `._index` and the `Resolve` object as `._r`. They are also kept around in the `SolverStateContainer`, but as public attributes: `.index` and `.r`, respectively.
3. `Solver._remove_specs()`: If `conda remove` was called, it removes the relevant specs from `specs_map`.
4. `Solver._add_specs()`: For all the other `conda` commands (`create`, `install`, `update`), it adds (or modifies) the relevant specs to `specs_map`. This is one of the most complicated pieces of logic in the class!

> **Check the other parts of the Solver API**
>
> You can check the rest of the Solver API *here*.

At this point, the `specs_map` is adequately populated and we can call the SAT solver wrapped by the `conda.resolve.Resolve` class. This is done in `Solver._run_sat()`, but this method does some other things before actually solving the SAT problem:

- Before calling `._run_sat()`, inconsistency analysis is performed via `Solver._find_inconsistent_packages`. This will preemptively remove certain `PackageRecord` objects from

ssc.solution_precs if Resolve.bad_installed() determined they were causing inconsistencies. This actually runs a series of small solves to check that the installed records form a satisfiable set of clauses. Those that prevent that solution from being found are annotated as such and ignored during the real solve later.

- Make sure the requested package names are available in the index.
- Anticipate and minimize potentially conflicting specs. This happens in a while loop fed by Resolve.get_conflicting_specs(). If a spec is found to be conflicting, it is *neutered*: a new MatchSpec object is created, but without version and build string constrains (e.g. numpy >=1.19 becomes just numpy). Then, Resolve.get_conflicting_specs() is called again, and the loop continues until convergence: the list of conflicts cannot be reduced further, either because there are no conflicts left or because the existing conflicts cannot be resolved by constraint relaxation.
- Now, the SAT solver is called. This happens via Resolve.solve(). More on this below.
- If the solver failed, then UnsatisfiableError is raised. Depending on which attempt we are on, conda will try again with non-frozen installed packages or a different repodata, or it will give up and analyze the conflict cause core. This will be detailed later.
- If the solver succeeded, some bookkeeping needs to be done:
  - Neutered specs that happened to be in the history are annotated as such.
  - Inconsistent packages are added back to the solution, including potential orphans.
  - Constraint analysis is run via Solver.get_constrained_packages() and Solver.determine_constricting_specs() to help the user understand why some packages were not updated.

We are not done yet, though. After Solver._run_sat(), we still need to run the post-solver logic! After the solve, the final list of PackageRecord objects might still change if certain modifiers are set. This is handled in the Solver._post_sat_handling():

- --no-deps (DepsModifier.NO_DEPS): Remove dependencies of the explicitly requested packages from the final solution.
- --only-deps (DepsModifier.ONLY_DEPS): Remove explicitly requested packages from the final solution but leave their dependencies. This is done via PrefixGraph.remove_youngest_descendant_nodes_with_specs().
- --update-deps (UpdateModifier.UPDATE_DEPS): This is the most interesting one. It actually runs a second solve (!) where the user-requested specs are the originally requested specs plus their (now determined) dependencies.
- --prune: Removes orphan packages from the solution.

The Solver also checks for Conda updates

Interestingly, the Solver API is also responsible of checking if new conda versions are available in the configured channels. This is done here to take advantage of the fact that the index has been already built for the rest of the class.

Details of conda.resolve.Resolve

This is the class that actually wraps the SAT solver. conda.core.solve.Solver is a higher level API that configures the solver *request* and prepares the transaction. The actual solution is computed in this other module we are discussing now.

The Resolve object will mostly receive two arguments:

- The fetched `index`, as processed by `conda.index.get_index()`.
- The configured `channels`, so *channel priority* can be sorted out.

It will also hold certain states:

- The `index` will be grouped by name under a `.groups` dictionary (`str`, `[PackageRecord]`). Each group is later sorted so newer packages are listed first, helping reduce the index better.
- Another dictionary of `PackageRecord` groups will be created, keyed by their `track_features` entries, under the `.trackers` attribute.
- Some other dictionaries are initialized as caches.

The main methods in this class are:

- `bad_installed()`: This method uses a series of small solves to check if the installed packages are in a consistent state. In other words, if all the `PackageRecord` entries were expressed as `MatchSpec` objects, would the environment be solvable?
- `get_reduced_index()`: This method takes a full index and trims out the parts that are not necessary for the current request, thus reducing the solution space and speeding up the solver.
- `gen_clauses()`: This instantiates and configures the `Clauses` object, which is the real SAT solver wrapper. More on this later.
- `solve()`: The main method in the `Resolve` class. It will be discussed in the next section.
- `find_conflicts()`: If the solver didn't succeed, this method performs a conflict analysis to find the most plausible explanation for the current conflicts. It essentially relies on `build_conflict_map()` to "find the common dependencies that might be the cause of conflicts". `conda` can spend a lot of time in this method.

Disabling conflict analysis

Conflict analysis can be disabled through the `context.unsatisfiable_hints` options, but unfortunately that gets in the way of conda's iterative logic. It will shortcut early in the chain of attempts and prevent the solver from trying less constrained specs. This is a part of the logic that should be improved.

`Resolve.solve()`

As introduced above, this is the main method in the `Resolve` class. It will perform the following actions:

1. Reduce the index via `get_reduced_index`. If unsuccessful, try to detect if packages are missing or the wrong version was requested. We can raise early to trigger a new attempt in `conda.cli.install` (remember, unfrozen or next repodata) or, if it's the last attempt, we go straight to `find_conflicts()` to understand what's wrong.
2. Instantiate a new `Resolve` object with the reduced index to generate the `Clauses` object via `gen_clauses()`. This method relies on `push_MatchSpec()` to turn the `MatchSpec` object into an SAT clause inside the `Clauses` object (referred to as `C`).
3. Run `Clauses.sat()` to solve the SAT problem. If a solution cannot be found, deal with the error in the usual way: raise early to trigger another attempt or call `find_conflicts()` to try explaining why.
4. If no errors are found, then we have one or more solutions available, and we need to post-process them to find the *best* one. This is done in several steps:
    1. Minimize the amount of removed packages. The SAT clauses are generated via `Resolve.generate_removal_count()` and then `Clauses.minimize()` will use it to optimize the current solution.

2. Maximize how well each record in the solution matches the spec. The SAT clauses are now generated in `Resolve.generate_version_metrics()`. This returns five sets of clauses: channel, version, build, arch or noarch, and timestamp. At this point, only channels and versions are optimized.
3. Minimize the number of records with `track_feature` entries. SAT clauses are coming from `Resolve.generate_feature_count()`.
4. Minimize the number of records with `features` entries. SAT clauses are coming from `Resolve.generate_feature_metric()`.
5. Now, we continue the work started at (2). We will maximize the build number and choose arch-specific packages over noarch variants.
6. We also want to include as many *optional* specs in the solution as possible. Optimize for that thanks to the clauses generated by `Resolve.generate_install_count()`.
7. At the same time, we will minimize the number of necessary updates if keeping the installed versions also satisfies the request. Clauses generated with `Resolve.generate_update_count()`.
8. Steps (2) and (5) are also applied to indirect dependencies.
9. Minimize the number of packages in the solution. This is done by removing unnecessary packages.
10. Finally, maximize timestamps until convergence so the most recent packages are preferred.

5. At this point, the SAT solution indices can be translated back to *SAT names*. This is done in the `clean()` local function you can find in `Resolve.sat()`.
6. There's a chance we can find alternate solutions for the problem, and this is explored now, but eventually only the first one will be returned while translating the *SAT names* to `PackageRecord` objects.

The `Clauses` object wraps the SAT solver using several layers

The `Resolve` class exposes the solving logic, but when it comes to interacting with the SAT solver engine, that's done through the `Clauses` object tree. And we say "tree" because the actual engines are wrapped in several layers:

- `Resolve` generates `conda.common.logic.Clauses` objects as needed.
- `Clauses` is a tight wrapper around its private `conda.common._logic.Clauses` counterpart. Let's call the former `_Clauses`. It simply wraps the `_Clauses` API with `._eval()` calls and other shortcuts for convenience.
- `_Clauses` provides an API to process the raw SAT formulas or clauses. It will wrap one of the `conda.common._logic._SatSolver` subclasses. *These* are the ones that wrap the SAT solver engines! So far, there are three subclasses, selectable via the `context.sat_solver` setting:
  - `_PycoSatSolver`, keyed as `pycosat`. This is the default one, a Python wrapper around the `picosat` project.
  - `_PySatSolver`, keyed as `pysat`. Uses the `Glucose4` solver found in the `pysat` project.
  - `_PyCryptoSatSolver`, keyed as `pycryptosat`. Uses the Python bindings for the CryptoMiniSat project.

In principle, more SAT solvers can be added to `conda` if a wrapper that subscribes to the `_SatSolver` API is used. However, if the reason is choosing a better performing engine, consider the following:

- The wrapped SAT solvers are already using compiled languages.
- Generating the clauses is indeed written in pure Python and has a non-trivial overhead.
- Optimization tricks like reducing the index and constraining the solution space have their costs if the "bets" were not successful.

More about SAT solvers in general

This guide did not cover the details of what SAT solvers are or do. If you want to read about them, consider checking the following resources:

- Aaron Meurer's slides about Conda internals. These slides reveal a lot of details of `conda` back in 2015. Some things have changed, but the core SAT solver behaviour is still well explained there.
- "Understanding and Improving Conda's performance"
- All the talks regarding solvers from Packaging-Con 2021. Check which talks belong to the Solvers track and enjoy!

Work in progress

This page of the documentation is not yet finished and only contains a draft of the content.

Solver state

Note

This document is a technical reference for how solver inputs are assembled, which might not be the best way to learn about how the solver works. For that, refer to *conda install* and *Solvers*.

The `Solver` API will pass a collection of `MatchSpec` objects (from now on, we will refer to them as `specs`) to the underlying SAT solver. How this list is built from the prefix state and context options is not a straightforward process, but an elaborate logic. This is better understood if we examine the *ingredients* that participate in the construction of `specs`. We will label them like this:

These groups below will *not* change during the solver attempts:

1. `requested`: `MatchSpec` objects the user is *explicitly* asking for.
2. `installed`: Installed packages, expressed as `PrefixRecord` objects. Empty if the environment did not exist.
3. `history`: Specs asked in the past: the `History`. Empty if the environment did not exist.
4. `aggressive_updates`: Packages included in the *aggressive updates* list. These packages are always included in any requests to make sure they stay up-to-date under all circumstances.
5. `pinned`: Packages pinned to a specific version, either via `pinned_packages` in your `.condarc` or defined in a `$PREFIX/conda-meta/pinned` file.
6. `virtual`: System properties exposed as virtual packages (e.g. `__glibc=2.17`). They can't really be installed or uninstalled, but they do participate in the solver by adding runtime constraints.
7. `do_not_remove`: A fixed list of packages that receive special treatment by the solver due to poor metadata in the early days of conda packaging. A legacy leftover.

This one group *does* change during the solver lifetime:

8. `conflicting`: Specs that are suspected to be a conflict for the solver.

Also, two more sources that are not obvious at first. These are not labeled as a *source*, but they do participate in the `specs` collection:

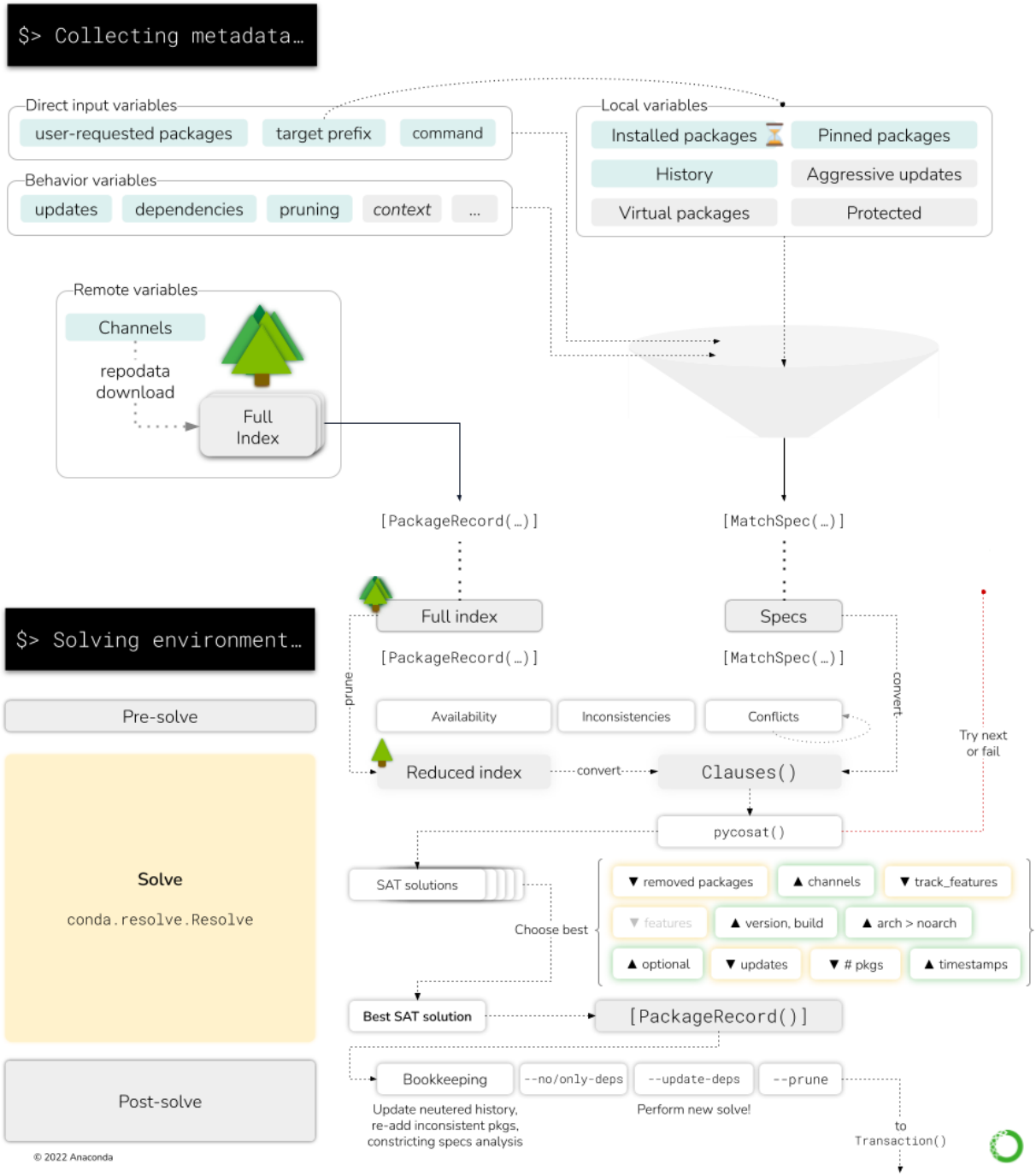


Fig. 4: Here you can see how the high level Solver API interacts with the low-level Resolve and Clauses objects.

The *Collecting metadata* step in the CLI report only compiles the necessary information from the CLI arguments, the prefix state and the chosen channels, presenting the SAT solver adapters with two critical pieces of information:

- The list of `MatchSpec` objects ("what the user wants in this environment")
- The list of `PackageRecord` objects ("the packages available in the channels")

So, in essence, the SAT solver takes the `MatchSpec` objects to select which `PackageRecord` objects satisfy the user request in the best way. The necessary computations are part of the "Solving environment…" step in the CLI report.

- In new environments, packages included in the `contex.create_default_packages` list. These `MatchSpec` objects are injected in each `conda create` command, so the solver will see them as explicitly requested by the user (`requested`).
- Specs added by command line modifiers. The specs here present aren't new (they are already in other categories), but they might end up in the `specs` list only when a flag is added. For example, `update --all` will add all the installed packages to the `specs` list, with no version constraint. Without this flag, the installed packages will still end up in the `specs` list, but with full constraints (`--freeze-installed` defaults for the first attempt) unless:
  - Frozen attempt failed.
  - `--update-specs` (or any other `UpdateModifier`) was passed, overriding `--freeze-installed`.

See? It gets involved. We will also use this vocabulary to help narrow down the type of change being done:

Types of `spec` objects:

- `specs`: map of package name to its currently corresponding `MatchSpec` instance.
- `spec`: specific instance of a `MatchSpec` object.
- Exact or frozen `spec`: a `spec` where both the `version` and `build` fields are constrained with == operators (exact match).
- Fully constrained or tight `spec`: a `spec` where both `version` and `build` are populated, but not necessarily with equality operators. It can also be inequalities (>, <, etc.) and fuzzy matches (`*something*`).
- Version-only `spec`: a `spec` where *only* the `version` field is populated. The `build` is not.
- Name-only, bare, or unconstrained `spec`: a `spec` with no `version` or `build` fields. Just the name of the package.
- Targeted `spec`: a `spec` with the `target` field populated. Extracted from the comments in the solver logic:

  > `target` is a reference to the package currently existing in the environment. Setting `target` instructs the solver to not disturb that package if it's not necessary. If the spec.name is being modified by inclusion in `specs_to_add`, we don't set `target`, since we *want* the solver to modify/update that package.
  >
  > TL;DR: when working with `MatchSpec` objects,
  >
  > - to minimize the version change, set `MatchSpec(name=name, target=prec.dist_str())`
  > - to freeze the package, set all the components of `MatchSpec` individually

- if the `spec` object does not have an adjective, it should be assumed it's being added to the `specs` map unmodified, as it came from its origin.

Pools (collections of `PackageRecord` objects):

- Installed pool: The installed packages, grouped by name. Each group should only contain one record!
- Explicit pool: The full index, but reduced for the specs in `requested`.

The following sections will get dry and to the point. They will state what output to expect from a given set of initial conditions. At least we'll try. Take into account that the `specs` list is kept around across attempts! In other words, the `specs` list is only really empty in the first attempt; if this fails, the subsequent attempts will only overwrite (update) the existing one. In practice, this should only affect how constrained packages are. The names should be the same.

It will also depend on whether we are adding (`conda install|create|update`) or removing (`conda remove`) packages. There's a common initialization part for both, but after that the logic is separate.

Common initialization

Note: This happens in `Solver._collect_all_metadata()`

This happens regardless of the type of command we are using (`install`, `update`, `create` or `remove`).

1. Add specs from `history`, if any.
2. Add specs from `do_not_remove`, but only if:
   - There's no spec for that name in `specs` already, *and*
   - A package with that name is *not* installed.
3. Add `virtual` packages as unconstrained `specs`.
4. Add all those installed packages, as unconstrained `specs`, that satisfy any of these conditions:
   - The history is empty (in that case, all installed packages are added)
   - The package name is part of `aggresive_updates`
   - The package was not installed by `conda`, but by pip or other PyPI tools instead.

Preparing the index

At this point, the populated `specs` and the `requested` specs are merged together. This temporary collection is used to determine how to reduce the index.

Processing specs for `conda install`

Preparation

1. Generate the explicit pool for the requested specs (via `Resolve._get_package_pool()`).
2. Detect potential conflicts (via (`Resolve.get_conflicting_specs()`).

Refine `specs` that match installed records

1. Check that each of `specs` match a single installed package or none! If there are two or more matches, it means that the environment is in bad shape and is basically broken. If the `spec` matches one installed package (let's call it *installed match*), we will modify the original `spec`.
2. We will turn the `spec` into an *exact* (frozen) spec if:
   1. The installed match is unmanageable (installed by pip, virtual, etc.)
   2. There's no history, we are not in `--freeze-installed` mode, and:
      - The spec is not a potential conflict, *and*
      - The package name *cannot* be found in the explicit pool index or, if it is, the installed match can be found in that explicit pool (to guarantee it will be found instead of creating one more conflict *just because*).
3. We relax the `spec` to a name-only `spec` if it's part of the aggressive updates list.
4. We turn it into a targeted `spec` if:
   1. The spec is in `history`. In that case, we take its *historic* spec counterpart and set the target to the installed match version and build.
   2. None of the above conditions were met. In other words, we'll try our best to match the installed package if none of the above applies, but if we fail we'll stick to whatever was already present in the `specs`.

Handle pinned specs

Processing specs for `conda remove`

Logging

Loggers record diagnostics: notes that help a developer or a user troubleshoot a problem, and warnings about conditions that are surprising but do not stop the command. Command results go through the reporter.

The conventions are:

- Every module creates one logger named after itself, at import time:

```
from logging import getLogger

log = getLogger(__name__)
```

- Library code emits records and does not configure handlers, levels, or propagation. When conda runs as a CLI, `conda.cli.main.init_loggers` attaches the handlers and sets the level from the verbosity flags. When another program imports conda, that program configures logging.
- Use lazy %-style arguments rather than f-strings so formatting only happens if the record is emitted.
- Pass `exc_info=True` to `log.debug` to record a traceback without showing it to the user.

Level	Use for	Visible when
`debug`	Internal state, decisions taken, caught-and-ignored exceptions	`-vvv` / `--debug`
`info`	Notable steps in normal operation	`-vv`
`warning`	Something is wrong or degraded, but conda can continue	Always (default level)

The `-v` flags map to levels in *`context.log_level`*:

Flags	`context.verbosity`	Logging level
(none)	0	`WARNING`
`-v`	1	`WARNING` (and more detailed command output)
`-vv`	2	`INFO`
`-vvv` / `--debug`	3	`DEBUG`
`-vvvv` / `--trace`	4	`TRACE`

From `conda/core/envs_manager.py`:

```
log.warning(
    "Unable to register environment. Could not create %s. Reason: %s",
    user_environments_txt_directory,
    exc,
)
```

A filter on the CLI handlers redacts channel tokens in URLs (`/t/<token>/`). Log those URLs as they are. Do not log other secrets.

See *Choosing between a logger and the reporter* for whether to log a message or render it.

How conda configures logging

Module-level loggers named after `__name__` form a hierarchy. `conda.core.link` sits under `conda.core`, which sits under `conda`. A parent's level and handlers apply to its children unless a child overrides them.

`conda.gateways.logging.initialize_logging` sets the `conda` logger to `WARNING` and attaches a stderr handler with `propagate = False`, so conda records do not reach the root logger. `conda.cli.main.init_loggers` calls that function, then `set_log_level(context.log_level)`. That sets the handler level from the verbosity flags and also attaches stderr handlers on the root logger and the `requests` loggers. Importing conda as a library does none of this. The embedding application configures logging.

Rendering output

This guide describes how conda displays output to users: which mechanism to use for which kind of output, and how to use it. There are four distinct channels, each with a different purpose:

What you want to show	Use	Goes to
Command results, progress, prompts (the "product" of the command)	The **reporter backend** via *conda.reporters*	stdout
Diagnostics, informational notes and warnings for developers and for troubleshooting	**Loggers** via `logging.getLogger()`	stderr (when handlers are configured)
Errors that end the command	**Exceptions** (subclasses of *CondaError*)	stderr, via the *exception handler*
Free-form user-facing text that predates the reporter backend	Legacy helpers in *conda.gateways.streams*	stdout/stderr (see *Legacy output channels*)

Direction of travel

The reporter backend is intended to be the **single output system** for the conda CLI. This is tracked in conda/conda#15477. The page you are reading documents the current state and the conventions that new code should follow. Parts of the legacy channel (the `conda.stdout`, `conda.stderr`, `conda.stdoutlog`, `conda.stderrlog` and `conda.stdout.verbose` loggers) are deprecated (removal in 27.3); see *Legacy output channels*.

The reporter backend

A *reporter backend* is a plugin (see *Reporter Backends*) that controls how conda renders output. conda ships two:

- `console`: the default, human-readable output (*conda.plugins.reporter_backends.console*).
- `json`: machine-readable output, used whenever `--json` is passed (*conda.plugins.reporter_backends.json*).

Every backend provides a *renderer* class derived from *ReporterRendererBase*. The renderer knows how to turn data into strings and how to create the interactive elements below:

Renderer method	Purpose
`render(data)`	Render arbitrary data. The default is `str(data)`; `console` appends a trailing newline and `json` serializes to JSON.
`detail_view(data)`	Render a `dict` as a table of aligned `key : value` rows.
`envs_list(data)`	Render a list of environments.
`progress_bar(description`	Create a *ProgressBarBase*.
`spinner(message, fail_message)`	Create a *SpinnerBase*.
`prompt(message, choices, default)`	Ask a question (used for yes/no confirmation).

Backend selection

Code that renders output should **never** import a backend or check which one is active. All selection happens in *conda.reporters*, which looks up *context.console* and asks the plugin manager for the matching backend:

- `context.json` is `True` (`--json`): the `json` backend is always used.
- Otherwise the `console` setting (`--console` or `console:` in `.condarc`) is used. It defaults to `console`.

The renderer lookup is cached. If you change the setting at runtime (normally only in tests), call *conda.base.context.reset_context()*, which also clears this cache.

API reference

Callers interact with the backend through the functions in *conda.reporters*:

Function	Purpose
render()	Write rendered data to stdout.
get_spinner()	Return a spinner context manager for long-running work.
get_progress_bar()	Return a progress bar object.
get_progress_bar_context_manag	Return the context manager that wraps progress bar usage (a null context by default).
confirm_yn()	Ask a yes/no question. Honors `--dry-run` and `--yes`.

Rendering data

render() takes the data and an optional `style`. Without a `style` the renderer's generic `render` method is used. With a `style`, the renderer method of that name is used (`"detail_view"`, `"envs_list"`, …). Any extra keyword arguments are passed on to the renderer method. An unknown style raises `AttributeError`.

```
from conda.reporters import render

# Generic rendering. In console mode this prints "hello\n"; in JSON mode, '"hello"'.
render("hello")

# A table of key/value pairs.
render({"name": "myenv", "frozen": False}, style="detail_view")
#  name : myenv
# frozen : False
```

(continues on next page)

(continued from previous page)

```
# Style-specific keyword arguments are forwarded to the renderer.
render(prefixes, style="envs_list", show_size=True)
```

Because the data is handed to the backend unchanged, **pass structured data, not pre-formatted text**, whenever the output has any structure. This is what lets the `json` backend emit something useful and lets other backends restyle the output.

`conda info` is a good example of this pattern. Each component produces its data and names the renderer style it needs, and the shared loop in *conda.cli.main_info.InfoRenderer* renders it (simplified):

```
from conda.reporters import render

style_by_component = {"base": None, "detail": "detail_view", "envs": "envs_list"}

for component in components:
    data = get_component_data(component)  # str, or dict when context.json is set
    if data:
        render(data, style=style_by_component[component])
```

For commands that only need to emit a JSON document, *conda.cli.common.stdout_json()* is a thin wrapper around `render(data)` that also swallows broken pipes. Its companion *conda.cli.common.stdout_json_success()* builds the conventional `{"success": true, ...}` payload. Use them for the `--json` branch of a command:

```
from conda.base.context import context
from conda.cli.common import stdout_json

if context.json:
    stdout_json({"env_vars": env_vars})
else:
    for key, value in env_vars.items():
        print(f"{key} = {value}")  # see "Plain text" below
```

Note

Writing a command's result in both a JSON and a non-JSON form is common today. Where a renderer style exists for your data (as with `detail_view` and `envs_list`), prefer a single `render(data, style=...)` call and let the backend decide.

Spinners

A spinner tells the user that a long-running step is in progress. Use *get_spinner()* as a context manager. If the body raises, `fail_message` is printed instead of `done`.

```
from conda.reporters import get_spinner

with get_spinner("Solving environment", fail_message="failed\n"):
    solution = solver.solve_final_state()
```

This is how the solver and transaction code report progress, for example in `conda.core.link`:

```
with get_spinner("Preparing transaction"):
    ...
with get_spinner("Verifying transaction"):
    ...
with get_spinner("Executing transaction"):
    ...
```

The `console` backend automatically degrades to a static `message: ...working... done` line when output is not a TTY, `TERM=dumb`, or `--quiet` is set. The `json` backend emits nothing. **Callers do not need to check any of these conditions.**

Progress bars

Progress bars report fractional progress, for example while downloading packages. *`get_progress_bar()`* returns an object you update with a fraction between `0` and `1`:

```
from conda.reporters import get_progress_bar

pbar = get_progress_bar("numpy-2.0.0")
try:
    for fraction in download():
        pbar.update_to(fraction)
    pbar.finish()
finally:
    pbar.close()
```

When several progress bars are active at once (for example during parallel downloads), wrap the work in *`get_progress_bar_context_manager()`* so a backend can coordinate them. The default is a null context, so this is always safe to use:

```
from conda.reporters import get_progress_bar, get_progress_bar_context_manager

with get_progress_bar_context_manager():
    pbar = get_progress_bar("Downloading")
    ...
```

The `console` backend returns a `tqdm` bar, or a silent bar under the same conditions as spinners. The `json` backend emits newline/NUL-delimited JSON progress records (`{"fetch": ..., "progress": ...}`), which is what `conda install --json` consumers parse.

Prompts

Use *`confirm_yn()`* to ask the user for confirmation. It handles the cross-cutting cases for you:

- If `--dry-run` is set, it raises *`DryRunExit`*.
- If `--yes` is set, it returns `True` without prompting.
- If the user answers "no" or presses Ctrl-C, it raises *`CondaSystemExit`* (a "clean" exit with return code `0`).
- Otherwise it returns `True`.

```
from conda.reporters import confirm_yn

confirm_yn()  # "Proceed ([y]/n)? "
confirm_yn("Remove these files", default="no")
```

Because "no" is signalled by an exception, code after the call can assume the user agreed.

Loggers

Logging covers creating a logger with `getLogger(__name__)`, choosing a level, and leaving handlers, levels, and propagation to the caller.

Choosing between a logger and the reporter

A useful rule of thumb: **if the text is the answer to what the user asked, render it; if it is commentary about how conda got there, log it.**

Situation	Use
`conda list`, `conda info`, `conda config --show` output	Reporter (`render`)
"Solving environment" status	Reporter (`get_spinner`)
"Proceed ([y]/n)?"	Reporter (`confirm_yn`)
Download progress	Reporter (`get_progress_bar`)
"Skipping unreadable `environments.txt`"	`log.warning`
"Found python 3.12 in the transaction"	`log.debug`
A package failed to link	Raise an exception (e.g. `CondaError` or subclass)

Logging output goes to stderr and the reporter writes to stdout. This keeps `conda list --json | jq` and similar pipelines clean even when warnings are emitted. Output is rendered as soon as either the `render` or `logging` functions are called. Raising a `CondaError` renders the error message as well as immediately halting program execution.

Errors

Do not print error messages. Raise an exception, and let the exception handler render it.

```
from conda.exceptions import CondaValueError

if not path.exists():
    raise CondaValueError(f"could not find environment file: {path}")
```

When an exception propagates out of a command, *conda.exception_handler.ExceptionHandler* decides what to show:

- *CondaError* subclasses are *expected* errors. Their message is formatted by *conda.exceptions.print_conda_exception()* and written to stderr. With `--json`, the error is serialized to a JSON object instead. Subclasses marked `reportable` additionally produce the full error report and may offer to upload it.
- Other exceptions are *unexpected*. The handler builds an error report (traceback, `conda info`, environment variables) and prints it to stderr, or emits it as JSON when `--json` is set.
- `KeyboardInterrupt`, `MemoryError` and out-of-disk-space errors are translated to the appropriate conda error first.

Using exceptions for errors provides consistent formatting, correct exit codes (`CondaError.return_code`), `--json` support, and the ability for *exception observer* and *error hint* plugins to react. A `print` or a `log.error` bypasses all of these.

Use a logger's `warning` level, not an exception, only when the command can still succeed.

Legacy output channels

Older code writes user-facing text without going through the reporter backend. You will see these in the code base and should not copy them into new code.

Custom loggers

The loggers `conda.stdout`, `conda.stderr`, `conda.stdoutlog`, `conda.stderrlog` and `conda.stdout.verbose` wrote straight to stdout/stderr and were configured by `conda.gateways.logging.initialize_std_loggers`. That function and the loggers are **deprecated** (removal in conda 27.3) and are no longer configured during normal CLI startup. Their meaning was also ambiguous (what is a `warning` on a logger named `stdout`?).

`conda.gateways.streams`

The module *`conda.gateways.streams`* contains helpers that replaced the custom loggers as a transitional step. They print to the corresponding stream and redact channel tokens:

Helper	Behavior
`stdout(text)` / `stderr(text)`	Always print.
`stdoutlog(text)` / `stderrlog(text)`	Print unless `context.json` is set.

Typical existing uses:

```
# conda/resolve.py
if not context.json:
    from .gateways.streams import stdoutlog

    stdoutlog("\nWarning: ... possible package resolutions ...")

# conda/instructions.py
from .gateways.streams import stdoutlog

stdoutlog(str(arg))
```

These calls bypass the selected reporter backend, so a non-default backend cannot restyle them and they duplicate the `--json` checks that `render` already performs. They remain only until usages are migrated.

Migration guidance

When touching code that uses one of these channels:

Existing code	Replace with
`stdoutlog(...)` conveying a command result or status	`render(...)`, `get_spinner(...)`, or `get_progress_bar(...)`
`stdoutlog(...)` conveying a caveat for the user	`log.warning(...)`
`stderrlog(...)` or `getLogger("conda.stderr*")` reporting a problem	`log.warning(...)`, or raise an exception if the command cannot continue
`print(...)` in a library function	`render(...)`, or `log.debug(...)` for diagnostics
`getLogger("conda.stdout").info(...)`	`render(...)`

Bare `print` calls are still present in some commands. Treat them the same way: if the text is output the user asked for, route it through the reporter; if it is diagnostic, use a logger.

Writing tests

Use the `capsys` fixture to assert on what a command renders to stdout and stderr, and `caplog` for log records. To exercise a specific backend, pass `--json` or set `console` through the usual context fixtures rather than patching *conda.reporters*.

```
import json

from conda.testing.fixtures import CondaCLIFixture


def test_info_json(conda_cli: CondaCLIFixture):
    out, err, rc = conda_cli("info", "--json")
    assert rc == 0, err
    assert "conda_version" in json.loads(out)


def test_warning_is_logged(caplog):
    caplog.set_level("WARNING", logger="conda.core.envs_manager")
    ...
```

Summary

- Render results, progress and prompts with *conda.reporters*. Pass structured data.
- Emit diagnostics with `getLogger(__name__)`. Never configure handlers in library code.
- Report failures by raising a *CondaError*; the exception handler renders it.
- Do not add new uses of the `conda.stdout*`/`conda.stderr*` loggers, or of `conda.gateways.streams.stdoutlog`/`stderrlog`.

Sharded repodata

This document provides an overview on how `conda` implements CEP-16 Sharded Repodata.

Sharded repodata is built around two kinds of files:

Shard index (`repodata_shards.msgpack.zst`)
: A zstandard-compressed msgpack file stored at `<channel>/<subdir>/repodata_shards.msgpack.zst`. It contains a mapping from package name to a SHA-256 hash that identifies the corresponding shard. This file is small relative to `repodata.json` because it grows only when new package *names* are added to the channel, not with every new package build. It is served with a short-lived `Cache-Control max-age` (typically 60 seconds to an hour) so that clients pick up new packages promptly.

Individual shards (`<sha256>.msgpack.zst`)
: Each shard is a zstandard-compressed msgpack file stored at `<shards_base_url><sha256>.msgpack.zst`. It contains the full repodata records (equivalent to the relevant slice of `repodata.json`) for every build of a single package name. Shards are content-addressable: the filename is the lower-case hex SHA-256 hash of the shard's contents. Because the URL changes whenever the content changes, shards can be served with `Cache-Control: immutable` and cached indefinitely by CDNs and clients alike.

Sharded Repodata in conda

Originally developed in `conda-libmamba-solver` and later ported into `conda`, we wanted a way to implement sharded repodata in Python that was independent of compiled solver code.

We do this by treating all repodata as if it was sharded repodata. Starting with a list of installed packages and to-be-installed packages, we gather all repodata for those packages and look for all package names listed in their dependencies. We repeat the process for every discovered package name that we have not already visited, fetching repodata shards or examining all artifacts with that package name as found in monolithic `repodata.json`. This process gathers all versions of all packages that we might depend on. We do not consider package versions at this stage; that's the solver's job.

As of this writing, `conda create -c conda-forge --dry-run python` finds 35 package names; `conda` 137 package names, and `vaex`, a dataframe library with a complex dependency tree, 678 package names. That's a lot less than the 31k packages total according to https://conda-forge.org/, and a manageable number to pre-process in Python before doing a solve with `libmamba`. As long as we can fetch those packages quickly enough, from cache or from the network, we will save RAM, disk space, bandwidth and time compared to parsing every package on the channel every time.

Threading and concurrency

In order to achieve concurrency, our sharded repodata implementation uses the Python threading module. We have two separate thread workers for fetching cache and network data. These threads communicate to each other via the following queues:

- **cache_in_queue** every requested shard goes here first where the cache worker sees if we have a valid cache record.
- **cache_miss_queue** for every shard not in cache, we send it this queue where the network worker thread downloads it.
- **shard_out_queue** once a shard has been fetched from either the cache or network worker threads, it is placed here so we can gather all needed shards at the end to build our repodata subset.

Source code

The shard handling code is split into `conda/_private/shards/shards.py`, `conda/_private/shards/cache.py`, `conda/_private/shards/subset.py`, `conda/_private/shards/typing.py`, and `conda/_private/shards/misc.py` in `conda/_private/`. `conda/gateways/shards/` re-exports `build_repodata_subset()`. When `context.repodata_use_shards` is enabled, `conda/plugins/manager.py` injects it into solver backends that accept a `build_repodata_subset` constructor parameter. Solver plugins such as `conda-libmamba-solver` pass the injected callable to their index helper, which calls it and converts the resulting repodata to solver objects in memory. If no channel provides sharded repodata, `build_repodata_subset()` returns `None` and the solver falls back to classic `repodata.json` loading.

conda/_private/shards/shards.py

`conda/_private/shards/shards.py` provides an interface to treat sharded repodata and monolithic `repodata.json` in the same way. It checks a channel for sharded repodata, returning an object that implements the `ShardLike` interface.

conda/_private/shards/subset.py

`conda/_private/shards/subset.py` accepts a list of `ShardLike` instances and a list of initial packages to compute a repodata subset. The traversal is simplified thanks to the `ShardLike` interface, so the algorithm doesn't have to worry too much about the type of each channel.

conda/_private/shards/cache.py

conda/_private/shards/cache.py implements a sqlite3 cache used to store individual shards. When traversing shards, the cache is checked before making a network request. The shards cache is a single database for all channels in $CONDA_PREFIX/pkgs/cache/repodata_shards.db.

The shards index repodata_shards.msgpack.zst is cached in the same way as repodata.json, in individual files in $CONDA_PREFIX/pkgs/cache/ named after URL hashes. A has_<format> remembers if a channel has shards, or not. If has_shards is false then we wait 7 days after last_checked to make another request looking for repodata_shards.msgpack.zst. The same system remembers whether a channel provides repodata.json.zst, and stores ETag and Last-Modified used to refresh the cache.

```
...
"has_shards": {
    "last_checked": "2025-10-15T17:19:44.408989Z",
    "value": true
},
```

conda/_private/shards/typing.py

conda/_private/shards/typing.py provides type hints for data structures used in sharded repodata, but it is not normative; it only includes fields used by the sharded repodata system.

conda/_private/shards/misc.py

conda/_private/shards/misc.py provides URL helpers, batching utilities, and connection-pool configuration used by the other shards modules.

tests/shards/

Tests under tests/shards/ cover the shards-related code in conda/_private/shards/*.py.

Example dependency graph for Python

This is what Python's dependencies look like on conda-forge as of this writing.

If sharded repodata is asked to install Python, we look for python in every active channel. The python shard(s) tells us we can fetch bzip2, libffi, ... in parallel, discovering a third layer including icu, ca-certificates, and others. ca-certificates also depends on some virtual packages, but the traversal quickly determines that these packages don't appear in any channel by checking the repodata_shards.msgpack.zst index. The solver will let us know if these missing packages are a problem, virtual or no.

The first draft of sharded repodata in conda literally generated classic repodata.json with package subsets to load into the solver, but now the solver gets a subset that yields individual package records, so that it can convert each record into solver objects in memory.

The subset gives the solver every possible dependency for a specific request. The transfer and parsing saved by not processing the full repodata makes up for the time spent generating a subset.

Shard hash validation

CEP-16 specifies that shards are content-addressable: the SHA-256 hash in the shard's filename is derived from the hash of its contents. This makes it possible to verify integrity without a round-trip to the server.

Conda does not validate that a downloaded shard's contents match the SHA-256 hash encoded in its filename.

This is a deliberate design decision for two reasons:

1. **Performance.** A solve request can involve hundreds of shard fetches. Hashing every shard after download would add measurable latency to each operation, working against the performance goals that motivated CEP-16 in the first place.
2. **Compatibility with certain channel providers.** Some channel providers serve sharded repodata in configurations where the content at a given shard hash URL cannot be guaranteed to match that hash — for example, when multiple upstream sources are aggregated and resolved transparently. Enforcing hash validation would break compatibility with these providers.

This behavior is intentional and should not be changed without careful consideration of both the performance impact and the downstream compatibility consequences.

conda install
: A thorough walkthrough of what happens when you run `conda install`: from parsing the command line through repodata fetching, the solver, transaction planning, and finally linking packages into the environment.

conda init and conda activate
: How conda manages shell integration, virtual environment activation and deactivation scripts, and the `conda init` bootstrap process.

conda config and context
: The central `Context` object: how settings are loaded from defaults, `.condarc` files, environment variables, and command-line flags, and how they are merged with the correct precedence.

Programmatic .condarc API
: How to programmatically read and write conda configuration files using the internal `.condarc` file API.

Solvers
: Inside the solver black box: how conda constructs the `MatchSpec` inputs, drives the SAT solver, and handles conflicts and retries.

Solver state
: A technical reference for how the list of `MatchSpec` objects passed to the SAT solver is constructed from prefix state, history, pins, and other inputs.

Logging
: How library code emits diagnostics with `getLogger(__name__)`, which level to use, and that the CLI configures handlers.

Rendering output
: How conda displays output: the reporter backend for command results, spinners, progress bars and prompts; loggers for diagnostics and warnings; and exceptions for errors.

Sharded repodata
: How conda implements CEP-16 sharded repodata: the fetch loop, threading model, SQLite shard cache, and the deliberate decision not to validate shard contents against their content-addressable SHA-256 filename hash.

Writing Tests

This section contains a series of guides and guidelines for writing tests in the `conda` repository.

Guides

Integration Tests This guide gives an overview of how to write integration tests using full command invocation. It also covers creating fixtures to use with these types of tests.

Benchmarks Guide for running and writing performance benchmarks, and interpreting Bencher results.

HTTP Test Server Fixture Guide for using the HTTP test server fixture to test scenarios requiring file serving (mock channels, remote files, etc.)

Testing with Windows AppLocker This guide explains how to set up a testing environment with AppLocker to ensure `conda` works correctly in these environments.

Integration Tests

Integration tests in `conda` test the application from a high level where each test can potentially cover large portions of the code. These tests may also use the local file system and/or perform network calls. In the following sections, we cover several examples of exactly how these tests look. When writing your own integration tests, these should serve as a good starting point.

`conda_cli` Fixture: Running CLI level tests

> **Note**
>
> The `conda_cli` fixture is a `scope="function"` fixture meaning it can only be used within tests or other `scope="function"` fixtures. For `"module"`, `"package"`, or `"session"` scoped fixtures use `session_conda_cli` instead.

CLI level tests are the highest level integration tests you can write. This means that the code in the test is executed as if you were running it from the command line. For example, you may want to write a test to confirm that an environment is created after successfully running `conda create`. A test like this would look like the following:

Listing 3: Integration test for `conda create`

```
from __future__ import annotations

import json
from typing import TYPE_CHECKING

if TYPE_CHECKING:
    from pathlib import Path

    from conda.testing.fixtures import CondaCLIFixture

pytest_plugins = "conda.testing.fixtures"


def test_conda_create(conda_cli: CondaCLIFixture, tmp_path: Path):
    # setup, create environment
    out, err, code = conda_cli("create", "--prefix", tmp_path, "--yes")

    assert f"conda activate {tmp_path}" in out
    assert not err  # no errors
    assert not code  # success!

    # verify everything worked using the `conda env list` command
    out, err, code = conda_cli("env", "list", "--json")

    assert any(
        tmp_path.samefile(path)
```

(continues on next page)

(continued from previous page)

```
        for path in json.loads(out).get("envs", [])
    )
    assert not err  # no errors
    assert not code  # success!

    # cleanup, remove environment
    out, err, code = conda_cli("remove", "--all", "--prefix", tmp_path)

    assert out
    assert not err  # no errors
    assert not code  # success!
```

Let's break down exactly what is going on in the code snippet above:

First, we rely on a fixture (`conda_cli`) that allows us to run a command using the current running process. This is much more efficient and quicker than running CLI tests via subprocesses.

In the test itself, we first create a new environment by effectively running `conda create`. This function returns the standard out, standard error, and the exit code of the command. This allows us to perform our inspections in order to determine whether the command ran successfully.

The second part of the test again uses the `conda_cli` fixture to call `conda env list`. This time, we pass the `--json` flag, which allows capturing JSON that we can better parse and more easily inspect. We then assert whether the environment we just created is actually in the list of environments available.

Finally, we destroy the environment we just created and ensure the standard error and the exit code are what we expect them to be.

Warning

It is preferred to use temporary directories (e.g., `tmp_path`) whenever possible for automatic cleanup after tests are run. Otherwise, remember to remove anything created during the test since it will be present when other tests are run and may result in unexpected race conditions.

`tmp_env` Fixture: Creating a temporary environment

Note

The `tmp_env` fixture is a `scope="function"` fixture meaning it can only be used within tests or other `scope="function"` fixtures. For `"module"`, `"package"`, or `"session"` scoped fixtures use `session_tmp_env` instead.

The `tmp_env` fixture is a convenient way to create a temporary environment for use in tests:

Listing 4: Integration test for creating an environment with `numpy`

```
from __future__ import annotations

from typing import TYPE_CHECKING

if TYPE_CHECKING:
    from conda.testing.fixtures import CondaCLIFixture, TmpEnvFixture
```

(continues on next page)

(continued from previous page)

```
pytest_plugins = "conda.testing.fixtures"


def test_environment_with_numpy(
    tmp_env: TmpEnvFixture,
    conda_cli: CondaCLIFixture,
):
    with tmp_env("numpy") as prefix:
        out, err, code = conda_cli("list", "--prefix", prefix)

        assert out
        assert not err  # no error
        assert not code  # success!
```

`path_factory` Fixture: Creating a unique (non-existing) path

The `path_factory` fixture extends pytest's tmp_path fixture to provide unique, unused paths. This makes it easier to generate new paths in tests:

Listing 5: Integration test for renaming an environment

```
from __future__ import annotations

from typing import TYPE_CHECKING

if TYPE_CHECKING:
    from pathlib import Path

    from conda.testing.fixtures import (
        CondaCLIFixture,
        PathFactoryFixture,
        TmpEnvFixture,
    )

pytest_plugins = "conda.testing.fixtures"


def test_conda_rename(
    path_factory: PathFactoryFixture,
    tmp_env: TmpEnvFixture,
    conda_cli: CondaCLIFixture,
    tmp_path: Path,
):
    # each call to `path_factory` returns a unique path
    assert path_factory() != path_factory()

    # each call to `path_factory` returns a path that is a child of `tmp_path`
    assert path_factory().parent == path_factory().parent == tmp_path

    with tmp_env() as prefix:
        out, err, code = conda_cli("rename", "--prefix", prefix, path_factory())
```

(continues on next page)

(continued from previous page)

```
        assert out
        assert not err  # no error
        assert not code  # success!
```

Tests with fixtures

Sometimes in integration tests, you may want to re-use the same type of environment more than once. Copying and pasting this setup and teardown code into each individual test can make these tests more difficult to read and harder to maintain.

To overcome this, `conda` tests make extensive use of `pytest` fixtures. Below is an example of the previously-shown test, except that we now make the focus of the test the `conda env list` command and move the creation and removal of the environment into a fixture:

Listing 6: Integration test for `conda create`

```
from __future__ import annotations

import json
from typing import TYPE_CHECKING

if TYPE_CHECKING:
    from pathlib import Path

    from conda.testing.fixtures import CondaCLIFixture, TmpEnvFixture

pytest_plugins = "conda.testing.fixtures"


@pytest.fixture
def env_one(tmp_env: TmpEnvFixture) -> Path:
    with tmp_env() as prefix:
        yield prefix


def test_conda_create(env_one: Path, conda_cli: CondaCLIFixture):
    # verify everything worked using the `conda env list` command
    out, err, code = conda_cli("env", "list", "--json")

    assert any(
        env_one.samefile(path)
        for path in json.loads(out).get("envs", [])
    )
    assert not err  # no errors
    assert not code  # success!
```

In the fixture named `env_one`, we create a new environment using the `tmp_env` fixture. We yield to mark the end of the setup. Since the `tmp_env` fixture extends `tmp_path` no additional teardown is needed.

This fixture will be run using the default scope in `pytest`, which is `function`. This means that the setup and teardown will occur before and after each test that requests this fixture. If you need to share an environment or other pieces of data between tests, just remember to set the fixture scope appropriately. Read here for more information on `pytest` fixture scopes.

Benchmarks

Conda uses pytest-benchmark to measure performance and Bencher to track measurements across commits.

Run locally

From the repository root, activate the *development environment* and install pytest-benchmark. For Bash, use Python 3.14 and Miniconda to match the current CI configuration:

```
source ./dev/start -p 3.14 -i miniconda
conda install --yes "pytest-benchmark>=5.1.0"
python -m pytest -m benchmark
```

Append a path or a pytest filter such as `-k test_generate_parser` to run a smaller selection. Run without coverage or parallel workers, which can affect timings.

To export the measurements as JSON, use:

```
python -m pytest -m benchmark --benchmark-json benchmark_results.json
```

The JSON file is ignored by Git. Local runs do not upload results to Bencher. Compare commits on the same machine with the same Python version, dependencies, and cache state. Local timings and CI timings can differ even with the same Python environment.

Write benchmarks

Place benchmarks alongside the tests for the code they measure. Mark each test with `@pytest.mark.benchmark` and pass the operation to the `benchmark` fixture. For example, measure constructing a `MatchSpec` with warm caches, then check the result after timing:

```
import pytest

from conda.models.match_spec import MatchSpec


@pytest.mark.benchmark
def test_match_spec_warm_cache(benchmark):
    MatchSpec("numpy")
    spec = benchmark(MatchSpec, "numpy")
    assert spec.name == "numpy"
```

Keep fixture setup and assertions outside the timed callable unless they are part of the operation being measured. Use fixed test data and stable test names and parameter IDs so Bencher can compare the same workload across commits.

Define whether each cache-sensitive case measures a cold or warm cache. If an operation mutates state, reset that state between iterations. Use `benchmark.pedantic` when setup must run before each round, with one iteration per round when each iteration requires fresh state. Ensure the benchmark checks the operation's result so an unintended no-op does not appear as a performance improvement.

Results and regression alerts

The `linux-benchmarks` job in the Tests workflow runs on Ubuntu 24.04 with Python 3.14 when the workflow detects code changes. Its `benchmark-results-v3` artifact contains the JSON measurements, workflow event, dependency list, and runner diagnostics, retained for seven days. A successful test run can have no benchmark results if that job was skipped.

The separate Track Benchmarks workflow uses the shared Bencher reporting action in `conda/actions` to upload available results to the Bencher project. Select `main` for historical results, or the relevant feature or release branch.

PR measurements use the exact base and head commits on the same runner with `PYTHONHASHSEED=0`, the head revision's resolved dependencies, benchmark tests, and fixtures. Running both revisions roughly doubles the benchmark execution time. Both results must contain the same benchmark names. If the base revision cannot run the complete head benchmark suite, the head results remain available and the reporting workflow posts a neutral check explaining why comparison is unavailable.

The reporting workflow creates a separate baseline for each PR workflow run and attempt. Select `pr-<number>` for its comparison. This baseline never replaces the `main` history. PR measurements are informational while the suite is stabilized. Single-round and cache-sensitive cases, together with shared-runner noise, can produce substantial timing changes without changes to the measured code.

The reporter disables alerts for every paired measurement and posts a neutral `Benchmark measurements (informational)` check with links to the producer run and Bencher. A neutral check does not establish that performance is unchanged. The original JSON artifacts retain every measurement and benchmark name. Historical branch alerts remain active.

Non-PR runs preserve the branch's history and use a t-test at `0.99`, with at least 10 and at most 64 historical measurements. The `0.99` value is a statistical prediction level, not a 1% slowdown allowance. A new benchmark or testbed may have too little history to produce an alert. Missing comparison warnings remain visible.

Testbeds include the producer's Ubuntu version, architecture, Python major/minor version, and CPU model. Ubuntu 24.04 measurements start separate histories from Ubuntu 22.04. Runner image versions are recorded in `runner_metadata.json` and `bencher noise` diagnostics in `noise.txt`, alongside the raw results. Noise measurements are diagnostic only and do not change timings or alert thresholds.

The same CPU model can still have different contention, cache state, or frequency. Dependencies can change between workflow runs even though each base/head pair shares an environment. Inspect the measurements and runner diagnostics before changing a threshold. Give benchmarks new names when their timed work or fixtures change so historical comparisons do not combine different workloads.

The versioned artifact name prevents the older reporting workflow from treating Ubuntu 24.04 measurements as Ubuntu 22.04 results while this change is in review.

HTTP Test Server Fixture

The HTTP test server fixture provides a way to test conda functionality that requires serving files over HTTP, such as:

- Mock conda channels with packages
- Remote environment files (`environment.yml`)
- Remote configuration files
- Any scenario where conda needs to fetch files from a URL

Overview

The `http_test_server` fixture starts a local HTTP server that serves files from a directory. The server runs on a random port and supports both IPv4 and IPv6.

The fixture can be used in two ways:

1. **Without `@pytest.mark.parametrize`** - Use a temporary directory that you populate dynamically
2. **With `@pytest.mark.parametrize`** - Serve files from a pre-existing directory

> **Tip**
>
> For proper type hints, import `HttpTestServerFixture` from `conda.testing.fixtures` under `TYPE_CHECKING`. See the *complete example* for the full import pattern.

Basic Usage

Dynamic Content (No Marker)

The simplest usage - no marker needed. The server automatically uses a temporary directory that you can populate:

```
def test_dynamic_repodata(http_test_server: HttpTestServerFixture):
    """Create content on the fly - no setup needed."""
    # Populate files directly in the server's directory
    (http_test_server.directory / "repodata.json").write_text('{"packages": {}}')

    # Make request
    response = requests.get(http_test_server.get_url("repodata.json"))
    assert response.status_code == 200
    assert response.json() == {"packages": {}}
```

This pattern is ideal for:

- Creating mock repodata files
- Testing with minimal setup
- Extending and creating your own fixtures programmatically

Pre-existing Directory (With Parametrize)

Use `@pytest.mark.parametrize()` with `indirect=True` when you have test data already prepared:

```
@pytest.mark.parametrize(
    "http_test_server",
    ["tests/data/mock-channel"],
    indirect=True,
)
def test_fetch_from_channel(http_test_server: HttpTestServerFixture):
    # Server serves files from tests/data/mock-channel/
    repodata_url = http_test_server.get_url("linux-64/repodata.json")

    response = requests.get(repodata_url)
    assert response.status_code == 200
```

The `indirect=True` parameter tells pytest to pass the directory path to the fixture rather than directly to the test function.

This pattern is ideal for:

- Complex directory structures
- Sharing test data across multiple tests
- Binary files (packages, archives)
- Large test datasets

Testing Multiple Directories

One of the benefits of using `@pytest.mark.parametrize` is that you can easily test the same logic against multiple directories:

```
@pytest.mark.parametrize(
    "http_test_server",
    [
        "tests/data/channel1",
        "tests/data/channel2",
        "tests/data/channel3",
    ],
    indirect=True,
)
def test_multiple_channels(http_test_server: HttpTestServerFixture):
    # This test runs three times, once for each channel directory
    response = requests.get(http_test_server.get_url("repodata.json"))
    assert response.status_code == 200
    assert "packages" in response.json()
```

Each test run will use a different directory, making it easy to verify behavior across multiple datasets.

You can also mix pre-existing directories with dynamic content by using `None`:

```
@pytest.mark.parametrize(
    "http_test_server",
    [
        "tests/data/channel1",
        None,
        "tests/data/channel2",
    ],
    indirect=True,
)
def test_mixed_sources(http_test_server: HttpTestServerFixture):
    # Runs 3 times: channel1, dynamic tmp dir, channel2
    # When None, http_test_server.directory is a fresh temporary directory
    ...
```

Complete Example: Testing a Mock Channel

Here's a full example with all imports showing dynamic content generation:

```
from __future__ import annotations

import json
from pathlib import Path
from typing import TYPE_CHECKING

import pytest
import requests

if TYPE_CHECKING:
    from conda.testing.fixtures import CondaCLIFixture, HttpTestServerFixture
```

(continues on next page)

(continued from previous page)

```
def test_install_from_mock_channel(
    http_test_server: HttpTestServerFixture,
    conda_cli: CondaCLIFixture,
    tmp_path: Path,
):
    """Test installing from a dynamically created mock channel."""
    # Create channel structure on the fly
    noarch = http_test_server.directory / "noarch"
    noarch.mkdir()

    # Create minimal repodata
    repodata = {"packages": {}, "packages.conda": {}, "repodata_version": 1}
    (noarch / "repodata.json").write_text(json.dumps(repodata))

    # Use the channel
    channel_url = http_test_server.url
    stdout, stderr, code = conda_cli(
        "search",
        f"--channel={channel_url}",
        "--override-channels",
        "*",
    )

    # Verify it worked (no packages found but channel was accessible)
    assert code == 0


@pytest.mark.parametrize(
    "http_test_server",
    ["tests/data/mock-channel"],
    # Assume the following structure:
    # tests/data/mock-channel/
    #   ├── noarch/
    #   │   └── repodata.json
    #   └── linux-64/
    #       ├── repodata.json
    #       └── example-pkg-1.0.0-0.tar.bz2
    indirect=True,
)
def test_install_from_preexisting_channel(
    http_test_server: HttpTestServerFixture,
    conda_cli: CondaCLIFixture,
    tmp_path: Path,
):
    """Test installing from pre-existing mock channel."""
    channel_url = http_test_server.url

    stdout, stderr, code = conda_cli(
        "create",
        f"--prefix={tmp_path}",
        f"--channel={channel_url}",
```

(continues on next page)

(continued from previous page)

```
        "example-pkg",
        "--yes",
    )

    assert code == 0
    assert (tmp_path / "conda-meta" / "example-pkg-1.0.0-0.json").exists()
```

Fixture API Reference

HttpTestServerFixture

The fixture returns an instance with these attributes and methods:

Attributes:

- `server: http.server.ThreadingHTTPServer` - The underlying server instance
- `host: str` - Server host (usually `127.0.0.1`)
- `port: int` - Server port (random)
- `url: str` - Base URL (e.g., `http://127.0.0.1:54321`)
- `directory: Path` - The directory being served (writable, use to populate content)

Methods:

- `get_url(path: str = "") -> str` - Get full URL for a path
  - Example: `get_url("linux-64/repodata.json")` → `"http://127.0.0.1:54321/linux-64/repodata.json"`

Using the `directory` attribute:

```
def test_dynamic_files(http_test_server: HttpTestServerFixture):
    # Write files directly to the served directory
    (http_test_server.directory / "file.txt").write_text("content")

    # Create subdirectories
    subdir = http_test_server.directory / "subdir"
    subdir.mkdir()
    (subdir / "nested.json").write_text('{"key": "value"}')

    # Files are immediately accessible via HTTP
    response = requests.get(http_test_server.get_url("subdir/nested.json"))
    assert response.json() == {"key": "value"}
```

Use in Downstream Projects

The HTTP test server fixture is part of the `conda.testing` module and can be used by downstream projects:

```
# In your project's conftest.py
pytest_plugins = "conda.testing.fixtures"
```

Then use it in your tests:

```
@pytest.mark.parametrize("http_test_server", ["tests/my-mock-channel"], indirect=True)
def test_with_mock_channel(http_test_server: HttpTestServerFixture):
    channel_url = http_test_server.url
    # ... your test code ...
```

Troubleshooting

"ValueError: Directory does not exist"

- This error occurs when using `@pytest.mark.parametrize()` with an invalid path
- Check that the directory path provided in parametrize exists
- Use absolute paths or paths relative to the repository root
- Use `Path(__file__).parent / "data"` if needed
- Or omit the parametrize decorator entirely to use a temporary directory

"ValueError: Path is not a directory"

- This error occurs when the parametrize value points to a file instead of a directory
- Ensure the path in `@pytest.mark.parametrize(..., indirect=True)` points to a directory
- Or use the fixture without parametrize for dynamic content

Address already in use

- The fixture uses random ports, so this is rare
- If it happens, the test will likely fail and retry automatically

Server not shutting down cleanly

- This is handled automatically by the fixture
- The server runs on a daemon thread and will be cleaned up when tests finish

Files not appearing in HTTP responses

- Make sure files are written before making the HTTP request
- Check that file paths don't have leading slashes when using `get_url()`
- Verify the directory structure with `list(http_test_server.directory.iterdir())`

Tips and Best Practices

1. **Prefer dynamic content**: Use the fixture without parametrize (dynamic content) for simple use cases. It's simpler and doesn't require maintaining test data files.
2. **Use parametrize for complex data**: Use `@pytest.mark.parametrize(..., indirect=True)` when you have complex directory structures, binary files, or data shared across many tests.
3. **Function scope for isolation**: Each test gets its own temporary directory with the `http_test_server` fixture (function scope), providing complete isolation.

4. **Organize test data**: When using parametrize, keep mock channel data in dedicated directories like `tests/data/mock-channels/` with README files explaining the structure.
5. **Test error scenarios**: Use dynamic content to easily test edge cases like malformed repodata, missing packages, or network timeouts.
6. **Cleanup is automatic**: The fixture handles cleanup automatically - no need to manually shut down servers or delete temporary files.

Examples from conda Test Suite

See these files for real-world usage examples:

- `tests/testing/test_http_test_server.py` - Tests for the fixture itself
- `tests/env/test_create.py::test_create_update_remote_env_file` - Using remote environment files
- `tests/gateways/test_connection.py` - Connection and download testing

Testing with Windows AppLocker

Windows environments with AppLocker enabled present unique challenges for `conda` development and testing. This guide explains how to set up a testing environment with AppLocker to ensure `conda` works correctly in these environments.

Why Test with AppLocker?

AppLocker is Microsoft's application control solution that allows organizations to:

- Control which applications and files users can run
- Create rules to allow or deny applications from running based on file attributes
- Create exceptions to rules

Many enterprise environments use AppLocker to restrict script execution, which can impact environmnet activation and execution processes. Testing with AppLocker ensures `conda` works properly in these restricted environments.

Setting Up AppLocker for Testing

Step 1: Enable the Application Identity Service

> **Note**
>
> The Application Identity Service is required for AppLocker to function properly.

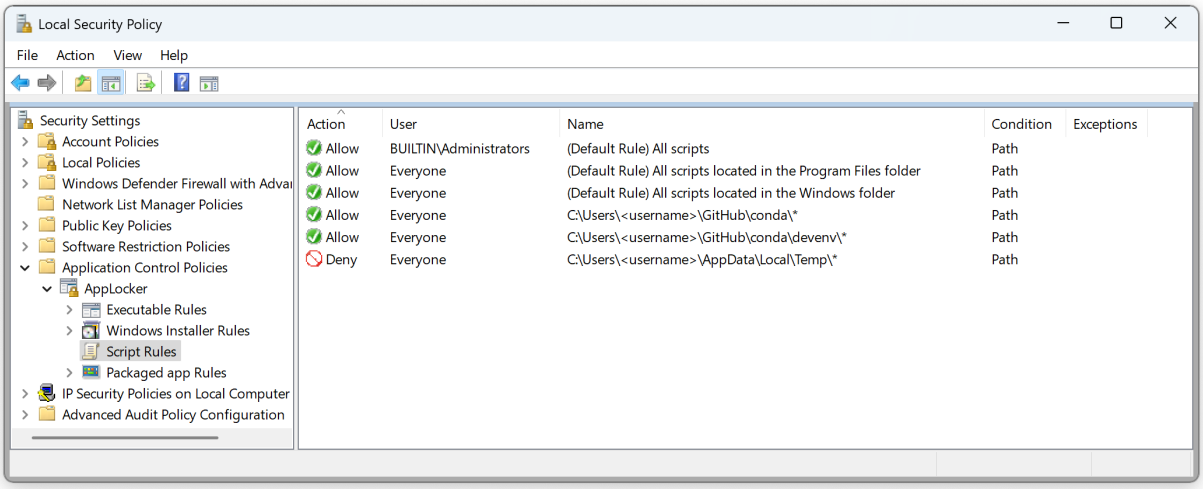
1. Open the **Services** application (press `Win+R`, type `services.msc`, and press Enter)
2. Find **Application Identity** in the list of services
3. Right-click on it and select **Properties**
4. *Optional*: Change **Startup type** to **Automatic** if you want the service to start on boot
5. Click **Start** to start the service
6. Click **OK** to close the properties window

Step 2: Configure AppLocker Enforcement

1. Open **Local Security Policy** (press `Win+R`, type `secpol.msc`, and press Enter)
2. Navigate to **Security Settings > Application Control Policies > AppLocker**
3. Right-click on **AppLocker** and select **Properties**
4. Under the **Enforcement** tab, check **Script Rules** and set it to **Enforce rules**
5. Click **OK** to close the properties window

Step 3: Create AppLocker Rules

1. In the **Local Security Policy** window, navigate to **Script Rules** under **AppLocker**
2. Right-click on **Script Rules** and select **Create Default Rules** to establish baseline rules
3. Create an Allow Rule for your development environment:
    - Right-click on **Script Rules** and select **Create New Rule...**
    - Choose **Allow** under Permissions and set the user/group to **Everyone**
    - Select **Path** as the condition
    - Enter the path to your development environment (e.g., path to `devenv`)
    - Complete the wizard without adding exceptions
4. Create an Allow Rule for the conda source code location using the same process
5. Create a Deny Rule for the `%TEMP%` directory:
    - Follow the same process but choose **Deny** under Permissions
    - Set the absolute path
6. Restart your computer to apply the rules



General Guidelines

- *Preferred test style (pytest)*
- *Organizing tests*
- *The "conda.testing" module*
- *Adding new fixtures*
- *The context object*

> **Note**
>
> It should be noted that existing tests may deviate from these guidelines, and that is okay. These guidelines are here to inform how we would like all new tests to look and function.

Preferred test style (pytest)

Although our codebase includes class-based `unittest` tests, our preferred format for all new tests are `pytest` style tests. These tests are written using functions and handle the setup and teardown of context for tests using fixtures. We recommend familiarizing yourself with `pytest` first before attempting to write tests for `conda`. Head over to their Getting Started Guide to learn more.

Organizing tests

Tests should be organized in a way that mirrors the main `conda` module. For example, if you were writing a test for a function in `conda/base/context.py`, you would place this test in `tests/base/test_context.py`.

The "conda.testing" module

This is a module that contains anything that could possibly help with writing tests, including fixtures, functions, and classes. Feel free to make additions to this module as you see fit, but be mindful of organization. For example, if your testing utilities are primarily only for the `base` module considering storing these in `conda.testing.base`.

Adding new fixtures

For fixtures that have a very limited scope or purpose, it is okay to define these alongside the tests themselves. However, if these fixtures could be used across multiple tests, then they should be saved in a separate `fixtures.py` file. The `conda.testing` module already contains several of these files.

If you want to add new fixtures within a new file, be sure to add a reference to this module in `tests/conftest.py::pytest_plugins`. This is our preferred way of making fixtures available to our tests. Because of the way these are included in the environment, you should be mindful of naming schemes and choose ones that likely will not collide with each other. Consider using a prefix to achieve this.

The context object

The context object in `conda` is used as a singleton. This means that everytime the `conda` command runs, only a single object is instantiated. This makes sense as it holds all the configuration for the program and re-instantiating it or making multiple copies would be inefficient.

Where this causes problems is during tests where you may want to run `conda` commands potentially hundreds of times within the same process. Therefore, it is important to always reset this object to a fresh state when writing tests.

This can be accomplished by using the `reset_context` function, which also lives in the `conda.base.context` module. The following example shows how you would modify the context object and then reset it using the `reset_conda_context` `pytest` fixture:

```
import os
import tempfile

from conda.base.context import reset_context, context
from conda.testing.fixtures import reset_conda_context

TEST_CONDARC = """
channels:
  - test-channel
"""


def test_that_uses_context(reset_conda_context):
    # We first created a temporary file to hold our test configuration
    with tempfile.TemporaryDirectory() as tempdir:
        condarc_file = os.path.join(tempdir, "condarc")

        with open(condarc_file, "w") as tmp_file:
            tmp_file.write(TEST_CONDARC)

        # We use the reset_context function to load our new configuration
        reset_context(search_path=(condarc_file,))

        # Run various test assertions, below is an example
        assert "test-channel" in context.channels
```

Using this testing fixture ensures that your context object is returned to the way it was before the test. For this specific test, it means that the `channels` setting will be returned to its default configuration. If you ever need to manually reset the context during a test, you can do so by manually invoking the `reset_context` command like in the following example:

```
from conda.base.context import reset_context


def test_updating_context_manually():
    # Load some custom variables into context here like above...

    reset_context()

    # Continue testing with a fresh context...
```

Deprecations

Conda abides by the Deprecation Schedule defined in CEP 9.

See also

For breaking changes that are *not* a public API removal — config default changes, default solver changes, CLI removals — see *Breaking changes*.

What must be deprecated

Public symbols must be deprecated before being removed.

Private symbols are no longer considered "features" eligible for deprecation.[1]

A private symbol is any symbol that starts with a single _ character; any symbol inside a module that starts with a single _ character; or any symbol not in a module's `__all__` list if it has such a list. This includes underscore-prefixed symbols in class instances.

Imported symbols (`import <package>` statements) are considered private.

Type checkers should be able to check these rules.

Using conda as a library

Use conda's public API. If you need a private symbol, pin your conda dependency to that specific conda release `conda==26.9.0`, and file an issue so we can consider exposing a suitable public API.

As a review, if any part of the import path starts with a single _ (`from conda._private.NiftyClass`) then it is private. Most other symbols are public when imported from their point of definition, unless a module has an `__all__` omitting that symbol.

Re-exporting a private symbol to the public API

typing.python.org defines conventions for *re-exporting* symbols to the public API. Type checkers understand these rules. This system helps conda to expose a designed set of APIs to the library user.

Marking code for deprecation

To help make deprecations as much of a no-brainer as possible we provide several helper decorators and functions to facilitate the correct deprecation process.

Tip

The conda pending/deprecation versions abide by the following formula:

```
pending_dep = (the next release + 4 months) round up to nearest .3 or .9
deprecation = pending_dep + 6 months
```

Example:

```
pending_dep = (25.1 + 4 months) round up to nearest .3 or .9 = 25.9
deprecation = 25.9 + 6 months = 26.3
```

Functions, Methods, Properties, and Classes

Warning

To deprecate Enums treat them like constants (see *Constants and Enums*).

The simplest use case is for deprecating any function, method, or property:

[1] Private symbols introduced before 2026-05 may require deprecation per the discretion of the conda maintainers.

Listing 7: Example file, `foo.py`.

```
from conda.deprecations import deprecated


@deprecated("23.9", "24.3")
def bar():
    ...
```

Listing 8: Example invocation.

```
>>> import foo
>>> foo.bar()
<stdin>:1: PendingDeprecationWarning: foo.bar is pending deprecation and will be removed␣
↪in 24.3.
```

As a minimum we must always specify two versions:

1. the future deprecation release in which the function, method, or property will be marked as deprecated; prior to that the feature will show up as pending deprecation (which we treat as a commenting period), and
2. the subsequent deprecation release in which the function, method, or property will be removed from the code base.

Additionally, you may provide an addendum to inform the user what they should do instead:

Listing 9: Example file, `foo.py`.

```
from conda.deprecations import deprecated


@deprecated("23.9", "24.3", addendum="Use `qux` instead.")
def bar():
    ...
```

Listing 10: Example invocation.

```
>>> import foo
>>> foo.bar()
<stdin>:1: PendingDeprecationWarning: foo.bar is pending deprecation and will be removed␣
↪in 24.3. Use `qux` instead.
```

Keyword Arguments

> **Warning**
>
> Deprecating or renaming a positional argument is unnecessarily complicated and is not supported. Instead, it is recommended to either (1) devise a custom way of detecting usage of a deprecated positional argument (e.g., type checking) and use the `conda.deprecations.deprecated.topic` function (see *Topics*) or (2) deprecate the function/method itself and define a new function/method without the deprecated argument.

Similarly to deprecating a function or method it is common to deprecate a keyword argument:

Listing 11: Example file, `foo.py`.

```
from conda.deprecations import deprecated


# prior implementation
# def bar(is_true=True):
#     ...


@deprecated.argument("23.9", "24.3", "is_true")
def bar():
    ...
```

Listing 12: Example invocation.

```
>>> import foo
>>> foo.bar(is_true=True)
<stdin>:1: PendingDeprecationWarning: foo.bar(is_true) is pending deprecation and will␣
↪be removed in 24.3.
```

Or to rename the keyword argument:

Listing 13: Example file, `foo.py`.

```
from conda.deprecations import deprecated


# prior implementation
# def bar(is_true=True):
#     ...


@deprecated.argument("23.9", "24.3", "is_true", rename="enabled")
def bar(enabled=True):
    ...
```

Listing 14: Example invocation.

```
>>> import foo
>>> foo.bar(is_true=True)
<stdin>:1: PendingDeprecationWarning: foo.bar(is_true) is pending deprecation and will␣
↪be removed in 24.3. Use 'enabled' instead.
```

argparse.Action

Occasionally, there is a need to deprecate CLI arguments. For this, we provide a helper function to monkeypatch any `argparse.Action`:

Listing 15: Example file, `foo.py`.

```
import argparse
from conda.deprecations import deprecated
```

(continues on next page)

(continued from previous page)

```
parser = argparse.ArgumentParser()
parser.add_argument(
    "--force",
    dest="yes",
    action=deprecated.action(
        "23.9",
        "24.3",
        argparse._StoreTrueAction,
        addendum="Use `--yes` instead.",
    ),
    default=False,
)
parser.parse_args()
```

```
python foo.py --force
foo.py:16: PendingDeprecationWarning: `--force` is pending deprecation and will be␣
↪removed in 24.3. Use `--yes` instead.
```

Constants and Enums

We also offer a way to deprecate global variables or constants:

Listing 16: Example file, `foo.py`.

```
from conda.deprecations import deprecated

deprecated.constant("23.9", "24.3", "ULTIMATE_CONSTANT", 42)
```

Listing 17: Example invocation.

```
>>> import foo
>>> foo.ULTIMATE_CONSTANT
<stdin>:1: PendingDeprecationWarning: foo.ULTIMATE_CONSTANT is pending deprecation and␣
↪will be removed in 24.3.
```

Enums work similarly:

Listing 18: Example file, `foo.py`.

```
from enum import Enum
from conda.deprecations import deprecated

class Bar(Enum):
    ULTIMATE_CONSTANT = 42

deprecated.constant("23.9", "24.3", "Bar", Bar)
del Bar
```

Listing 19: Example invocation.

```
>>> from foo import Bar
<stdin>:1: PendingDeprecationWarning: foo.Bar is pending deprecation and will be removed␣
↪in 24.3.
```

Note

Constants deprecation relies on the module's `__getattr__` introduced in PEP-562.

Deferred constants

When the value of a deprecated constant is expensive to materialize (for example, a class whose definition requires a heavy import), pass a zero-argument callable to the `factory=` keyword instead of passing `value` positionally. The factory is only invoked the first time the deprecated symbol is accessed; the result is cached for subsequent accesses, so identity is preserved:

Listing 20: Example file, `foo.py`.

```
from conda.deprecations import deprecated


def _make_heavy():
    import some_heavy_dependency
    return some_heavy_dependency.Thing


deprecated.constant(
    "23.9",
    "24.3",
    "HEAVY_THING",
    factory=_make_heavy,
    addendum="Use `lightweight_alternative` instead.",
)
```

Listing 21: Example invocation.

```
>>> import foo  # some_heavy_dependency is NOT imported
>>> foo.HEAVY_THING  # some_heavy_dependency is imported now, and only now
<stdin>:1: PendingDeprecationWarning: foo.HEAVY_THING is pending deprecation and will be␣
↪removed in 24.3. Use `lightweight_alternative` instead.
```

Modules

Entire modules can be also be deprecated:

Listing 22: Example file, `foo.py`.

```
from conda.deprecations import deprecated

deprecated.module("23.9", "24.3")
```

Listing 23: Example invocation.

```
>>> import foo
<stdin>:1: PendingDeprecationWarning: foo is pending deprecation and will be removed in␣
↪24.3.
```

Topics

Finally, there are a multitude of other ways in which code may be run that also needs to be deprecated. To this end we offer a general purpose deprecation function:

Listing 24: Example file, `foo.py`.

```
from conda.deprecations import deprecated

def bar(...):
    # some logic

    if ...:
        deprecated.topic("23.9", "24.3", topic="The <TOPIC>")

    # some more logic
```

Listing 25: Example invocation.

```
>>> import foo
>>> foo.bar(...)
<stdin>:1: PendingDeprecationWarning: The <TOPIC> is pending deprecation and will be␣
↪removed in 24.3.
```

Breaking changes

While conda strives towards providing a highly stable package manager experience, sometimes the introduction of breaking changes cannot be avoided. For example, we may need to change a configuration default, update the solver, or remove/change a CLI subcommand or flag. While these changes are usually few and far between, it's important that we define the appropriate procedure to follow for our maintainers. Additionally, CEP 9 defines the deprecation schedule for public API and behavior *removals*, but it does not by itself say how to plan, announce, and ship a user-visible breaking change that isn't a symbol removal. This page describes the process we use for that, building on CEP 9's schedule and CEP 8's release cadence.

Read this together with *Deprecations* (the warning mechanics in code) and *Releasing* (how a release actually gets cut).

Change categories

Config default changes and removals
: The setting itself stays supported; only its *implicit* default value changes (for example, flipping `add_pip_as_python_dependency` from `true` to `false`), or the setting is removed outright. Users who already set the value explicitly should see no behavior change.

Default solver (or other high-impact default) changes
: Swapping which plugin or implementation runs when the user hasn't chosen one, such as the 23.10.0 switch to `conda-libmamba-solver`. These tend to have the widest blast radius and are the most likely candidates for a dedicated special release (see below).

CLI subcommand or command option removals and behavior changes
: Removing a subcommand, removing or renaming a flag, or changing what an existing invocation does — not just deprecating the Python API that backs it.

Every category follows the same rollout tools in *Communication and rollout*; they differ mainly in *how* conda warns users ahead of time.

Relation to CEP 9

CEP 9 and *Deprecations* give us the schedule (pending deprecation → deprecation → removal) and the tooling (`deprecated`, `deprecated.argument`, `deprecated.action`, `deprecated.constant`, `deprecated.module`, `deprecated.topic`, etc.) for deprecating a **symbol**.

CEP 9 alone is not enough when:

- The change is a **default value**, not a symbol. There is no function or argument to decorate with `@deprecated(...)`; instead, warn with `deprecated.topic(...)` at the point where conda falls back to the implicit default — and only when the user hasn't set the value explicitly.
- The change needs **rollout coordination** beyond a warning: a release-notes announcement, a blog post, installer coordination, or a dedicated release. None of that is covered by CEP 9; it's the subject of this page.
- The behavior lives partly **outside conda's code**, e.g. an installer-level change (like protected base environments) that ships through Miniconda/Miniforge rather than through a `PendingDeprecationWarning`.

If a change *is* a straightforward public symbol or behavior removal, use CEP 9's schedule and `conda.deprecations` as documented in *Deprecations* and stop there — this page only adds process for changes that need broader communication.

Communication and rollout

Not every change needs every one of these; use judgment based on blast radius.

Warn on the CLI when practical
: If conda can detect that a user is relying on the soon-to-change default, emit a warning with `deprecated.topic(...)` (see the Topics section of *Deprecations*). Only warn when the behavior was *not* explicitly requested — never warn a user who set `solver: classic` or `add_pip_as_python_dependency: false` explicitly. Skip the warning entirely for settings that are a plain user opt-in/opt-out with no implicit default to migrate away from.

Announce in the *previous* release's notes
: Add a "Special announcement" section (see the 23.9.0 example below) or a clearly labeled `releases/news/` entry describing what will change and when, in the release *before* the one that ships the change. That gives users at least one release's notice in the changelog itself.

Publish a conda.org blog post for major changes
: For changes with a wide blast radius (default solver, protected base, etc.), a blog post reaches users who don't read `CHANGELOG.md`. Link it from the release notes. For extra impact, you can link directly to that blog post with an announcement banner.

Prefer a dedicated special release on an even month for the highest-impact default flips
: Deprecation releases land in March and September (CEP 9). Shipping a high-impact default flip in its own release on an even month keeps it out of deprecation-heavy releases, makes it easy to find in the changelog, and makes it easy to revert if needed. This is what we did for the libmamba solver switch in 23.10.0.

Provide a migration and opt-out path
: Document the config setting, CLI flag, or environment variable that restores the old behavior, and keep it working for at least one full deprecation cycle. Repeat this guidance in both the docs and the release notes announcing the change.

Consider a beta installer for installer-coupled behavior
: When the behavior depends on how Miniconda/Miniforge are built (for example, protected base environments), ship a beta installer build before the change reaches the general population, so downstream tooling can be tested against it first.

Previous examples

Default solver switch: `conda-libmamba-solver` (23.9.0 → 23.10.0)

- **23.9.0** shipped a "Special announcement" in the release notes stating the intent to switch, the opt-out flags (`--solver=classic`, `CONDA_SOLVER=classic`, `conda config --set solver classic`), and a link to the rollout blog post.
- **23.10.0** was dedicated to the switch itself: `solver: libmamba` became the new default (#12984), and the release notes restated the same opt-out paths.

Config default removal: implicit `defaults` channel (24.9.0 → 25.9.0)

- **24.9.0** marked the implicit `defaults` multichannel as pending deprecation, warned on the CLI only when the user hadn't configured `channels` explicitly, and documented the opt-back-in path (`conda config --add channels defaults`). (#14178 via #14227)
- **25.3.0** postponed the removal from the originally announced 25.3 to 25.9 in response to feedback — the schedule can slip without abandoning the process. (#14178 via #14662)
- **25.9.0** removed the implicit behavior. (#15196)

This is the closer precedent for a config-default change that didn't need a dedicated release: CLI warnings plus release-note announcements carried the communication load across releases.

See also

- CEP 8 — release cadence and versioning.
- CEP 9 — deprecation schedule for public API/behavior removals.
- *Deprecations* — marking and removing APIs and behaviors in code.
- *Previews* — the opposite end of the lifecycle: opt-in experimental features.
- *Releasing* — how a release, special or otherwise, actually gets cut.
- #16404 — an in-progress example applying this process to the `add_pip_as_python_dependency` default.

Releasing

The `conda` release process is documented in `RELEASE.md` at the root of the repository on GitHub. It covers the full end-to-end process: opening a release issue, running rever, publishing the release, and bumping feedstocks.

See also

Release process (RELEASE.md)

Plugins

As of version `22.11.0`, conda has support for user plugins, enabling extension and/or alterations to some of its functionality.

Quick start

This is an example of a minimal working conda plugin that defines a new subcommand:

Listing 26: example_plugin.py

```
import conda.plugins.types
from conda.base.context import context


def command(arguments: list[str]):
    print("Conda subcommand!")


@conda.plugins.hookimpl
def conda_subcommands():
    yield conda.plugins.types.CondaSubcommand(
        name="example",
        action=command,
        summary="Example of a conda subcommand",
    )
```

Let's break down what's going on here step-by-step:

1. First, we create the function `command` that serves as our subcommand. This function is passed a list of arguments which equal to `sys.argv[2:]`.
2. Next, we register this subcommand by using the `conda_subcommands` plugin hook. We do this by creating a function called `conda_subcommands` and then decorating it with `conda.plugins.hookimpl`.
3. The object we return from this function is `conda.plugins.types.CondaSubcommand`, which does several things:
    1. **name** is what we use to call this subcommand via the command line (i.e. "conda example")
    2. **action** is the function that will be called when we invoke "conda example"
    3. **summary** is the description of the of the subcommand that appears when users call "conda --help"

In order to actually use conda plugins, they must be packaged as Python packages. Furthermore, we also need to take advantage of a feature known as Python package entrypoints. We can define our Python package and the entry points by either using a `pyproject.toml` file (preferred) or a `setup.py` (legacy) for our project:

Listing 27: pyproject.toml

```
[build-system]
requires = ["setuptools", "setuptools-scm"]
build-backend = "setuptools.build_meta"
```

(continues on next page)

(continued from previous page)

```
[project]
name = "conda-example-plugin"
version = "1.0.0"
description = "Example conda plugin"
requires-python = ">=3.11"
dependencies = ["conda"]

[project.entry-points."conda"]
conda-example-plugin = "example_plugin"
```

Listing 28: setup.py

```
from setuptools import setup

setup(
    name="conda-example-plugin",
    install_requires="conda",
    entry_points={"conda": ["conda-example-plugin = example_plugin"]},
    py_modules=["example_plugin"],
)
```

In both examples shown above, we define an entry point for conda. It's important to make sure that the entry point is for "conda" and that it points to the correct module in your plugin package. Our package only consists a single Python module called `example_plugin`. If you have a large project, be sure to always point the entry point to the module containing the plugin hook declarations (i.e. where `conda.plugins.hookimpl` is used). We recommend using the *plugin* submodule in these cases, e.g. *large_project.plugin* (in *large_project/plugin.py*).

More examples

To see more examples of conda plugins, please visit the following resources:

- conda-plugins-template: This is a repository with full examples that could be used a starting point for your plugin

Using other plugin hooks

For examples of how to use other plugin hooks, please read their respective documentation pages:

Auth Handlers

The auth handlers plugin hook allows plugin authors to enable new modes of authentication within conda. Registered auth handlers will be available to configure on a per channel basis via the `channel_settings` configuration option in the `.condarc` file.

Auth handlers are subclasses of the *ChannelAuthBase* class, which is itself a subclass of requests.auth.AuthBase. The *ChannelAuthBase* class adds an additional `channel_name` property to the requests.auth.AuthBase class. This is necessary for appropriate handling of channel based authentication in conda.

For more information on how to implement your own auth handlers, please read the requests documentation on Custom Authentication.

class CondaAuthHandler

Return type to use when the defining the conda auth handlers hook.

Parameters

- **name** -- Name (e.g., `basic-auth`). This name should be unique and only one may be registered at a time.
- **handler** -- Type that will be used as the authentication handler during network requests.

handler: *type*[*ChannelAuthBase*]

name: str

User-facing name of the plugin used for selecting & filtering plugins and error messages.

conda_auth_handlers() → collections.abc.Iterable[*conda.plugins.types.CondaAuthHandler*]

Register a conda auth handler derived from the requests API.

This plugin hook allows attaching requests auth handler subclasses, e.g. when authenticating requests against individual channels hosted at HTTP/HTTPS services.

Example:

```
import os
from conda import plugins
from requests.auth import AuthBase


class EnvironmentHeaderAuth(AuthBase):
    def __init__(self, *args, **kwargs):
        self.username = os.environ["EXAMPLE_CONDA_AUTH_USERNAME"]
        self.password = os.environ["EXAMPLE_CONDA_AUTH_PASSWORD"]

    def __call__(self, request):
        request.headers["X-Username"] = self.username
        request.headers["X-Password"] = self.password
        return request


@plugins.hookimpl
def conda_auth_handlers():
    yield plugins.types.CondaAuthHandler(
        name="environment-header-auth",
        handler=EnvironmentHeaderAuth,
    )
```

Error Hints

The `conda_error_hints` plugin hook allows plugins to contribute structured next-step hints while conda renders a *conda.CondaError*.

This hook is for user-facing remediation, not telemetry. Plugins should yield *CondaErrorHint* objects and should not print directly. Conda appends the yielded hints to the error's existing guidance so terminal and `--json` output stay consistent.

Example

```
from conda import plugins
from conda.exceptions import PackagesNotFoundInChannelsError
```

(continues on next page)

(continued from previous page)

```
@plugins.hookimpl
def conda_error_hints(error):
    if isinstance(error, PackagesNotFoundInChannelsError):
        yield plugins.types.CondaErrorHint(
            text="Check whether the package exists on your expected channel.",
            hint_code="check_expected_channel",
        )
```

With that plugin installed, conda appends the hint to the normal terminal guidance output:

```
PackagesNotFoundInChannelsError: The following packages are not available
from current channels:
  - missing-package

Next steps:
  - (check_expected_channel) Check whether the package exists on your expected channel.
```

The same hint is included in `--json` output:

```
{
  "error": "...",
  "exception_name": "PackagesNotFoundInChannelsError",
  "guidance": {
    "hints": [
      {
        "hint_code": "check_expected_channel",
        "text": "Check whether the package exists on your expected channel."
      }
    ],
    "hint_codes": ["check_expected_channel"]
  }
}
```

Conda invokes each `conda_error_hints` implementation independently while rendering expected errors. If one plugin raises or yields an invalid hint, conda logs the plugin failure at DEBUG level and continues rendering the original error plus any other valid hints.

Hint order is deterministic: conda invokes implementations by plugin name, preserves the order yielded by each implementation, and appends plugin hints after core guidance hints. If two hints use the same `hint_code`, the first hint wins, so core guidance takes priority over plugin guidance. Hook wrappers are skipped for this hook so conda can isolate failures per implementation.

Error hints vs. exception observers

Use `conda_error_hints` when a plugin wants to add user-facing next steps to an expected *conda.CondaError*. Hints are part of conda's normal error guidance model: conda controls ordering, deduplicates by `hint_code`, renders them in the terminal `Next steps` section, and includes them in `--json` under `guidance.hints` and `guidance.hint_codes`.

Use *Exception Observers* when a plugin needs to observe failures for telemetry, logging, demand tracking, or other side effects. Observers are fire-and-forget callbacks modelled after `sys.excepthook()`: their return value is ignored, and they should not mutate exceptions or print user-facing messages.

API reference

class CondaErrorHint

Return type to use when defining a conda error hints plugin hook.

For details on how this is used, see *conda_error_hints()*.

Parameters

- **text** -- Human-readable description of the action to take.
- **hint_code** -- Stable machine-readable identifier. Use snake_case.

hint_code: str

Stable machine-readable identifier. Use snake_case.

text: str

Human-readable description of the action to take.

conda_error_hints(*error:* conda.CondaError) → collections.abc.Iterable[*conda.plugins.types.CondaErrorHint*]

Register user-facing hints for expected conda errors.

This hook is invoked while rendering a *conda.CondaError*. Plugins receive the current error and yield structured *CondaErrorHint* objects that conda appends to the error's existing guidance. Plugins should not print directly from this hook.

When a *CondaMultiError* is printed, conda invokes this hook once per nested leaf error (for example `RemoveError`), not on the multi-error container itself. Implementations should match concrete exception types; `isinstance(error, CondaMultiError)` will not see wrapped failures.

Conda invokes implementations in deterministic plugin-name order and preserves the hint order yielded by each plugin. Existing core guidance hints win when a plugin yields the same `hint_code`. Hook wrappers are skipped so conda can isolate failures per implementation.

Use this hook for user-facing next steps. Use *conda_exception_observers()* for telemetry, logging, demand tracking, or other side effects.

Example:

```
from conda import plugins
from conda.exceptions import PackagesNotFoundInChannelsError


@plugins.hookimpl
def conda_error_hints(error):
    if isinstance(error, PackagesNotFoundInChannelsError):
        yield plugins.types.CondaErrorHint(
            text="Check whether the package exists on your expected channel.",
            hint_code="check_expected_channel",
        )
```

Returns

An iterable of *CondaErrorHint* entries.

Environment Exporters

The environment exporter plugin hook allows you to create custom export formats for conda environments. This feature was introduced to enhance the *conda export* command with a plugin-based architecture for extending conda's environment export capabilities.

Overview

Environment exporters transform conda environments into various file formats that can be shared, stored, or used to recreate environments. The plugin system supports both structured formats (like YAML/JSON) and text-based formats (like explicit URLs or requirement specs).

Built-in exporters include:

- **environment-yaml** - YAML format for cross-platform environments
- **environment-json** - JSON format for programmatic processing
- **explicit** - CEP 23 compliant explicit URLs format
- **requirements** - *MatchSpec* -based requirements format

Plugin Architecture

Environment exporters are registered through the `conda_environment_exporters` plugin hook. Each exporter defines:

- A unique name and optional aliases
- Supported filename patterns for auto-detection
- An export function that transforms environments to the target format

class CondaEnvironmentExporter

EXPERIMENTAL

Return type to use when defining a conda environment exporter plugin hook supporting a single platform.

Parameters

- **name** -- name of the exporter (e.g., `environment-yaml`)
- **aliases** -- user-friendly format aliases (e.g., ("yaml",))
- **default_filenames** -- default filenames this exporter handles (e.g., ("environment.yml", "environment.yaml"))
- **export** -- callable that exports an Environment to string format for a single platform
- **multiplatform_export** -- callable that exports an Environment to string format for multiple platforms
- **description** -- user-friendly description of what the format does. Defaults to the name if not provided.
- **environment_format** -- EnvironmentFormat category. Defaults to EnvironmentFormat.environment.

aliases: `tuple[str, ...]`

default_filenames: `tuple[str, ...]`

description: `str | None = None`

environment_format: *EnvironmentFormat*

export: `SinglePlatformEnvironmentExport | None = None`

multiplatform_export: `MultiPlatformEnvironmentExport | None = None`

name: *str*

User-facing name of the plugin used for selecting & filtering plugins and error messages.

conda_environment_exporters() → collections.abc.Iterable[*conda.plugins.types.CondaEnvironmentExporter*]

Register new conda environment exporter

Environment exporters serialize conda Environment objects to different output formats (JSON, TOML, XML, etc.) for the 'conda export' command. This is separate from environment specifiers which parse input files.

Example:

```
import tomlkit
from conda import plugins
from conda.exceptions import CondaValueError
from conda.models.environment import Environment
from conda.plugins.types import CondaEnvironmentExporter


def export_toml(env: Environment) -> str:
    # Export Environment to TOML format
    # For formats that use the standard dictionary structure,
    # you can use the shared utility:
    from conda.plugins.environment_exporters.standard import to_dict

    env_dict = to_dict(env)

    # Create TOML document
    toml_doc = tomlkit.document()

    if env_dict.get("name"):
        toml_doc["name"] = env_dict["name"]

    if env_dict.get("channels"):
        toml_doc["channels"] = env_dict["channels"]

    if env_dict.get("dependencies"):
        toml_doc["dependencies"] = env_dict["dependencies"]

    if env_dict.get("variables"):
        toml_doc["variables"] = env_dict["variables"]

    return tomlkit.dumps(toml_doc)


@plugins.hookimpl
def conda_environment_exporters():
    yield CondaEnvironmentExporter(
        name="environment-toml",
        aliases=("toml",),
        default_filenames=("environment.toml",),
        export=export_toml,
    )
```

Creating an Environment Exporter Plugin

What follows are several examples showing how to create new export formats using the environment exporters plugin hook. Please check out our *Quick start* guide for more detailed instructions on how to create a conda plugin.

Basic Plugin Structure

Here's a minimal example of an environment exporter plugin:

```
import conda.plugins
from conda.models.environment import Environment


def export_simple_text(environment: Environment) -> str:
    """Export environment as a simple text list."""
    return "\n".join(
        (
            f"# Environment: {environment.name or environment.prefix}",
            f"# Platform: {environment.platform}",
            f"# Packages:",
            *(str(package) for package in environment.explicit_packages),
        )
    )


def export_multiplatform_text(environments: Iterable[Environment]) -> str:
    """Export environments as a simple text list."""
    return "\n".join(
        (
            f"# Environment: {environment.name or environment.prefix}",
            f"# Platforms: {', '.join((environment.platform for environment in␣
↪environments))}",
            f"# Packages:",
            *(
                str(package)
                for environment in environments
                for package in environment.explicit_packages
            ),
        )
    )


@conda.plugins.hookimpl
def conda_environment_exporters():
    yield conda.plugins.types.CondaEnvironmentExporter(
        name="simple-text",
        aliases=("simple", "txt-simple"),
        default_filenames=("environment.txt",),
        export=export_simple_text,
    )
    yield conda.plugins.types.CondaEnvironmentExporter(
        name="multiplatform-text",
        aliases=("multiplatform", "txt-multiplatform"),
        default_filenames=("environment.txt",),
```

(continues on next page)

(continued from previous page)

```
        export=export_multiplatform_text,
    )
```

See also

For a general introduction and examples of how to distribute conda plugins, see the ../plugins quick start guide.

Plugin Components

Below, we explain how to use the plugin you've created above with *conda export*.

Name and Aliases

The `name` field defines the canonical format name used with `--format`:

```
conda export --format=simple-text
```

The `aliases` tuple provides alternative names for convenience:

```
conda export --format=simple
conda export --format=txt-simple
```

Note

Aliases are automatically normalized to lowercase and stripped of whitespace. The plugin system will detect and prevent name collisions.

Default Filenames

The `default_filenames` tuple specifies filename patterns for automatic format detection:

```
# These would auto-detect the simple-text format
conda export --file=environment.txt
```

Export/Multiplatform Export Function

There are two export functions, one for single platform formats and one for multiplatform formats. Both return a string representation:

- `export`: receives a single *`Environment`* object
- `multiplatform_export`: receives a list of *`Environment`* objects

Advanced Example: JSON Exporter

Here's a more sophisticated example that creates a custom JSON format:

```
import json
from typing import Any, Dict
```

(continues on next page)

(continued from previous page)

```
import conda.plugins.types
from conda.models.environment import Environment


def export_custom_json(environment: Environment) -> str:
    """Export environment as custom JSON format."""
    data: Dict[str, Any] = {
        "format_version": "1.0",
        "environment": {
            "name": environment.name,
            "channels": [str(channel) for channel in environment.channels],
        },
    }

    # Add requested_packages as MatchSpec strings
    if environment.requested_packages:
        data["environment"]["requested_packages"] = [
            str(dep) for dep in environment.requested_packages
        ]

    # Add explicit packages with full metadata
    if environment.explicit_packages:
        data["environment"]["explicit_packages"] = [
            {
                "name": pkg.name,
                "version": pkg.version,
                "build": pkg.build,
                "channel": str(pkg.channel),
                "url": pkg.url,
                "md5": pkg.md5,
            }
            for pkg in environment.explicit_packages
        ]

    # Add environment variables
    if environment.variables:
        data["environment"]["variables"] = dict(environment.variables)

    return json.dumps(data, indent=2, sort_keys=True)


@conda.plugins.hookimpl
def conda_environment_exporters():
    yield conda.plugins.types.CondaEnvironmentExporter(
        name="custom-json",
        aliases=("cjson",),
        default_filenames=("environment.cjson", "env.cjson"),
        export=export_custom_json,
    )
```

Error Handling

Your export function should handle error cases appropriately:

```
from conda.exceptions import CondaValueError


def export_strict_format(environment: Environment) -> str:
    """Export that requires specific conditions."""
    if not environment.requested_packages:
        raise CondaValueError(
            "Cannot export strict format: no requested packages found. "
            "This format requires at least one requested package."
        )

    if not environment.name:
        raise CondaValueError(
            "Cannot export strict format: environment name is required."
        )

    # Continue with export...
    return formatted_content
```

Working with Different Package Types

Understanding Package Collections

The Environment model provides different package collections for different use cases:

requested_packages (*MatchSpec* objects)
: Represents user-requested packages. These are the packages the user explicitly asked for, either from history (when using `--from-history`) or converted from installed packages.

explicit_packages (*PackageRecord* objects)
: Represents all installed packages with full metadata including URLs, checksums, and build information. Used for exact reproduction.

external_packages (dict of str -> list[str])
: Represents external packages. These are packages that are not conda packages. For example, pip packages.

Example usage patterns:

```
def export_user_requested(environment: Environment) -> str:
    """Export only what the user explicitly requested."""
    if not environment.dependencies:
        raise CondaValueError("No requested packages found")

    lines = []
    for dep in environment.dependencies:
        lines.append(str(dep))  # e.g., "numpy=1.21.0"
    return "\n".join(lines)


def export_exact_reproduction(environment: Environment) -> str:
    """Export for exact environment reproduction."""
```

(continues on next page)

(continued from previous page)

```
    if not environment.explicit_packages:
        raise CondaValueError("No installed packages found")

    lines = ["@EXPLICIT"]
    for pkg in environment.explicit_packages:
        lines.append(pkg.url)  # Full package URL
    return "\n".join(lines)
```

Plugin Detection and Conflicts

Automatic Format Detection

When users run `conda export --file=filename.ext`, conda:

1. Checks all registered exporters for matching `default_filenames`
2. If exactly one match is found, uses that exporter
3. If no matches or multiple matches, raises an appropriate error

The detection system is case-insensitive and supports glob-like patterns.

Collision Prevention

The plugin system automatically prevents naming conflicts:

- Format names and aliases are normalized (lowercase, stripped)
- Duplicate format names or aliases raise *PluginError*
- This ensures deterministic behavior and clear error messages

Testing Your Plugin

Here's a basic test structure for your exporter plugin:

```
import pytest
from conda.models.environment import Environment
from conda.testing.fixtures import tmp_env
from my_export_plugin.exporters import export_custom_json


def test_custom_json_exporter(tmp_env):
    """Test the custom JSON exporter."""
    environment = Environment.from_prefix(tmp_env.prefix)
    result = export_custom_json(environment)

    # Verify the output format
    import json

    data = json.loads(result)
    assert data["format_version"] == "1.0"
    assert "environment" in data
    assert "name" in data["environment"]
```

(continues on next page)

(continued from previous page)

```
def test_empty_environment_handling(tmp_env):
    """Test exporter with empty environment."""
    environment = Environment(name="test-empty")

    # Should handle gracefully or raise appropriate error
    result = export_custom_json(environment)
    data = json.loads(result)
    assert data["environment"]["name"] == "test-empty"
```

Best Practices

1. **Validation**: Always validate inputs and provide clear error messages
2. **Documentation**: Include format specifications and examples in your plugin
3. **Backwards compatibility**: Consider versioning your format for future changes
4. **Performance**: Optimize for large environments with many packages
5. **Cross-platform**: Consider platform differences in your format design

Example Use Cases

Some ideas for custom environment exporters:

- **Docker integration**: Export as Dockerfile or Docker Compose
- **Language-specific**: Export as language package files (package.json, Gemfile, etc.)
- **Cloud deployment**: Export as cloud infrastructure templates
- **Version control**: Export in formats optimized for VCS tracking
- **Documentation**: Export as formatted documentation or reports

Further Reading

For more information about conda plugin development:

- *Plugin overview* - General plugin development guide
- *Environment specifiers* - Input counterpart to exporters
- `conda.models.environment.Environment` - Environment model API
- *conda export* - Export command documentation

Environment Specifiers

Conda can create environments from several file formats. Currently, conda natively supports creating environments from:

- A *YAML environment.yaml file*
- An *"explicit" text file*

For more information on how to manage conda environments, see the **`Managing environments`_** documentation.

Example plugin

The available readers can be extended with additional plugins via the `conda_environment_specifiers` hook.

> **Hint**
>
> To see a fully functioning example of a Environment Spec backend, checkout the *`yaml_file`* module.

conda_environment_specifiers() → collections.abc.Iterable[*conda.plugins.types.CondaEnvironmentSpecifier*]

EXPERIMENTAL Register new conda env spec type

The example below defines a type of conda env file called "random". It can parse a file with the file extension *.random*. This plugin will ignore whatever is in the input environment file and produce an environment with a random name and with random packages.

Example:

```
import json
import random
from pathlib import Path
from subprocess import run
from conda import plugins
from conda.plugins.types import EnvironmentSpecBase
from conda.base.context import context
from conda.models.environment import Environment
from conda.models.match_spec import MatchSpec

packages = ["python", "numpy", "scipy", "matplotlib", "pandas", "scikit-learn"]


class RandomSpec(EnvironmentSpecBase):
    extensions = {".random"}

    def __init__(self, filename: str):
        self.filename = filename

    def can_handle(self):
        # Return early if no filename was provided
        if self.filename is None:
            raise FileNotFoundError

        # The can_handle method should check the contents of the file and
        # not be reliant on file name or extension.  This allows a user to
        # select a plugin by name, but also allows a plugin to handle a file
        # with a different extension.

        # Some quick checks to ensure the file is in the proper format here.
        # NOTE: It should be good enough to determine if the file is the proper
        # format in order to not be greedy and cause issues with other plugins.
        # If more than one plugin can handle the file, the user will need to go
        # through an additional step to select the proper plugin.

    @property
```

(continues on next page)

(continued from previous page)

```
    def env(self):
        return Environment(
            platform=context.subdir,
            name="".join(random.choice("0123456789abcdef") for i in range(6)),
            requested_packages=[MatchSpec(random.choice(packages)) for i in␣
↪range(6)],
        )


@plugins.hookimpl
def conda_environment_specifiers():
    yield plugins.CondaEnvSpec(
        name="random",
        environment_spec=RandomSpec,
    )
```

Defining `EnvironmentSpecBase`

The first class we define is a subclass of *`EnvironmentSpecBase`*. The base class is an abstract base class which requires us to define our own implementations of its abstract methods:

- `can_handle` Determines if the defined plugin can read and operate on the provided file.
- `env` Expresses the provided environment file as a conda environment object.

The base class also provides default-implemented APIs for multi-platform specs. Single-platform specs (`environment.yml`, `requirements.txt`) can ignore them:

- `available_platforms` Tuple of platform subdirs this spec covers. Defaults to `(context.subdir,)`.
- `env_for(platform)` Return the `Environment` for a specific platform. Defaults to `env` if `platform == context.subdir`, raises otherwise.

Multi-platform specs (`conda-lock.yml`, `pixi.lock`) override both to expose the full platform set declared in the input file. Callers iterate with:

```
envs = (spec.env_for(p) for p in spec.available_platforms)
```

The class may also define the boolean class variable *detection_supported*. When set to `True`, the plugin will be included in the environment spec type discovery process. Otherwise, the plugin will only be able to be used when it is specifically selected. By default, this value is `True`.`

Be sure to be very specific when implementing the `can_handle` method. Here are a few guiding principles: * It should only return a `True` if the file can be parsed by the plugin. * If it can not handle the file, it should raise an error. The error should describe why it can not

handle the file.

- Making the `can_handle` method too permissive in the types of files it handles may lead to conflicts with other plugins.
- If multiple installed plugins are able to `can_handle` the same file type, conda will return an error to the user.
- Ensure that the `can_handle` method does not depend on a file extension or specific file name as users may specify the plugin by name to read in files.

Registering the plugin hook

In order to make the plugin available to conda, it must be registered with the plugin manager. Define a function with the `plugins.hookimpl` decorator to register our plugin which returns our class wrapped in a *CondaEnvironmentSpecifier* object. Note, that by default autodetection is enabled. Plugin authors can additionally define aliases for their plugin. This will allow users to select the plugin using the aliased names in addition to the provided name.

```
@plugins.hookimpl
def conda_environment_specifiers():
    yield plugins.CondaEnvSpec(
        name="random",
        environment_spec=RandomSpec,
        aliases=("rand", "rnd"),
    )
```

Using the Plugin

Once this plugin is registered, users will be able to create environments from the types of files specified by the plugin. The `conda create`, `conda env create`, and `conda install` commands all use these plugins when given a `--file` argument. For example to create a *random* environment using the plugin defined above:

```
conda create --file /doesnt/matter/any/way.random
```

Plugin detection

When conda is trying to determine which environment spec plugin to use it will loop through all registered plugins and call their `can_handle` function. If one (and only one) plugin returns a `True` value, conda will use that plugin to read the provided environment spec. However, if multiple plugins are detected an error will be raised.

Plugin authors may explicitly disable their plugin from being detected by disabling autodetection in their plugin class

```
class RandomSpec(EnvironmentSpecBase):
    detection_supported = False

    def __init__(self, filename: str):
        self.filename = filename

    def can_handle(self):
        return True

    @property
    def env(self):
        return Environment(
            platform=context.subdir,
            name="random-environment",
            requested_packages=[MatchSpec("python"), MatchSpec("numpy")],
        )
```

End users can bypass environment spec plugin detection and explicitly request a plugin to be used by configuring conda to use a particular installed plugin. This can be done by either:

- cli by providing the `--env-spec` flag, or
- environment variable by setting the `CONDA_ENV_SPEC` environment variable, or

- `.condarc` by setting the `environment_specifier` config field

Another example plugin

In this example, we want to build a more realistic environment spec plugin. This plugin has a scheme which expresses what it expects a valid environment file to contain. In this example, a valid environment file is a `.json` file that defines:

- an environment name (required)
- a list of conda dependencies

```
import os
from pydantic import BaseModel

from conda.plugins import hookimpl
from conda.plugins.types import CondaEnvironmentSpecifier, EnvironmentSpecBase
from conda.base.context import context
from conda.models.environment import Environment
from conda.models.match_spec import MatchSpec


class MySimpleEnvironment(BaseModel):
    """An model representing an environment file."""

    # required
    name: str

    # optional
    conda_deps: list[str] = []


class MySimpleSpec(EnvironmentSpecBase):
    def __init__(self, filename=None):
        self.filename = filename

    def _parse_data(self) -> MySimpleEnvironment:
        """ "Validate and convert the provided file into a MySimpleEnvironment"""
        with open(self.filename, "rb") as fp:
            json_data = fp.read()

        return MySimpleEnvironment.model_validate_json(json_data)

    def can_handle(self) -> bool:
        """
        Validates loader can process environment definition.
        This can handle if:
            * the file exists
            * the file can be read
            * the data can be parsed as JSON into a MySimpleEnvironment object

        :return: True if the file can be parsed and handled, False otherwise
        """
        if not os.path.exists(self.filename):
            raise FileNotFoundError
```

(continues on next page)

(continued from previous page)

```
        # Will raise an error if parsing fails
        self._parse_data()
        return True

    @property
    def env(self) -> Environment:
        """Returns the Environment representation of the environment spec file"""
        data = self._parse_data()
        return Environment(
            platform=context.subdir,
            name=data.name,
            requested_packages=[MatchSpec(dep) for dep in data.conda_deps],
        )


@hookimpl
def conda_environment_specifiers():
    yield CondaEnvironmentSpecifier(
        name="mysimple",
        environment_spec=MySimpleSpec,
    )
```

We can test this out by trying to create a conda environment with a new file that is compatible with the definied spec. Create a file `testenv.json`

```
{
   "name": "mysimpletest",
   "conda_deps": ["numpy", "pandas"]
}
```

Then, create the environment

```
$ conda create --file testenv.json
```

Exception Observers

The `conda_exception_observers` plugin hook allows plugins to observe exceptions as they pass through conda's error-reporting path. This is useful for telemetry, logging, and demand tracking -- for example, reporting which packages users tried to install but could not find in the configured channels.

Exception observer plugins are **purely observational** (modelled after CPython's `sys.excepthook()`): they cannot suppress, modify, or redirect the exception. Their return value is ignored. Any exception raised inside an observer is caught and logged at DEBUG level, so a buggy plugin can never disrupt conda.

Observers fire for every exception conda handles: `CondaError` and its subclasses, `MemoryError`, `KeyboardInterrupt`, `SystemExit`, and plain Python exceptions such as `RuntimeError` or `KeyError` that conda would otherwise report as an unexpected error. The `watch_for` parameter selects which of those trigger a given observer; see *Choosing a watch_for scope* below.

Tutorial: reporting missing packages to channels

Suppose you maintain a private conda channel and want to know which packages your users are looking for but cannot find. You can write a small plugin that fires whenever a *PackagesNotFoundInChannelsError* is raised and sends a lightweight report to each channel.

Step 1 -- write the hook

Listing 29: conda_missing_reporter/plugin.py

```python
from __future__ import annotations

import logging
from typing import TYPE_CHECKING

from conda import plugins

if TYPE_CHECKING:
    from conda.plugins.types import CondaExceptionEvent

log = logging.getLogger(__name__)


def report_missing(event: CondaExceptionEvent) -> None:
    """
    Send a fire-and-forget GET request to each channel's ``/missing``
    endpoint so that channel maintainers can track demand.

    Uses conda's built-in session so that proxy settings, SSL
    configuration, and per-channel auth are picked up automatically.
    """
    if event.offline or event.dry_run:
        return

    from conda.gateways.connection.session import get_session

    exc = event.exc_value

    specs = ",".join(str(s) for s in exc.packages)
    for url in {u.rstrip("/") for u in exc.channel_urls}:
        target = f"{url}/missing?specs={specs}"
        try:
            get_session(target).get(target, timeout=2)
        except Exception:
            log.debug("Failed to report to %s", target, exc_info=True)


@plugins.hookimpl
def conda_exception_observers():
    yield plugins.types.CondaExceptionObserver(
        name="missing-package-reporter",
        hook=report_missing,
        watch_for={"PackagesNotFoundInChannelsError"},
```

(continues on next page)

(continued from previous page)

```
    )
```

`watch_for` accepts a set of exception class names. Matching uses the exception's full MRO, so parent class names automatically match subclasses.

For non-`CondaError` exceptions, the conda runtime fields on *`CondaExceptionEvent`* may be `None` (see below).

Choosing a `watch_for` scope

Pick the narrowest scope that covers the exceptions you care about:

`watch_for`	Captures	Typical use
`{"CondaError"}`	All conda errors and their subclasses (missing packages, solver failures, channel errors).	Plugins focused on conda's own errors (channel demand tracking, solver analytics).
`{"PackagesNotFoundError"}`	One error class and its subclasses.	Narrowly targeted integrations.
`{"Exception"}`	Every standard exception, including non-`CondaError` types like `RuntimeError` or `KeyError`. Excludes `KeyboardInterrupt` and `SystemExit`.	Sensible default for error-tracking backends such as Sentry or Rollbar.
`{"BaseException"}`	Every exception, including `KeyboardInterrupt` and `SystemExit`.	Diagnostics or auditing where every exit path matters.
`{"MemoryError"}`, `{"KeyboardInterrupt"}`, `{"SystemExit"}`	Specific non-`CondaError` types.	Narrowly targeted observers.
`{"CondaError", "MemoryError"}`	Union of scopes. The observer fires when any class in the exception's MRO matches any entry in the set.	Combining a domain scope with a specific non-conda type.

Step 2 -- package and register

Package the hook as a standard conda plugin using entry points. See the *Plugins Quick start* for a full walkthrough of `pyproject.toml` setup, entry-point configuration, and installation.

Step 3 -- test it

You can test the plugin without packaging it by registering the plugin class directly with the plugin manager:

```
import sys

from conda import CondaError
from conda.exceptions import PackagesNotFoundInChannelsError
from conda.plugins.manager import CondaPluginManager


class FakeReporter:
    """Collects exception events for assertions."""

    def __init__(self):
        self.calls = []
```

(continues on next page)

(continued from previous page)

```
    @conda.plugins.hookimpl
    def conda_exception_observers(self):
        yield conda.plugins.types.CondaExceptionObserver(
            name="fake-reporter",
            hook=self.calls.append,
            watch_for={"PackagesNotFoundInChannelsError"},
        )


pm = CondaPluginManager()
reporter = FakeReporter()
pm.register(reporter)

exc = PackagesNotFoundInChannelsError(
    packages=["numpy"],
    channel_urls=["https://repo.anaconda.com/pkgs/main"],
)
try:
    raise exc
except CondaError:
    _, exc_val, exc_tb = sys.exc_info()
    pm.invoke_exception_observers(exc_val, exc_tb)

assert len(reporter.calls) == 1
event = reporter.calls[0]
assert event.exc_value is exc
assert event.exc_type is PackagesNotFoundInChannelsError
```

What the observer receives

Observers are called with a single frozen *CondaExceptionEvent* dataclass.

The exception triple is always populated:

Field	Description
`exc_type`	The exception class.
`exc_value`	The exception instance. For `CondaError` subclasses, access domain attributes like `exc_value.packages` here.
`exc_traceback`	The traceback object.

The remaining fields describe the conda runtime state. They are populated all-or-nothing: either the runtime was available and all fields are set, or it wasn't and they're all `None`. Check `conda_version is not None` to tell the two cases apart. `active_prefix` is the one exception -- it can be `None` even when the runtime is available (meaning no environment is active):

Field	Description
`argv`	A frozen copy of `sys.argv` at the time of the error.
`conda_version`	The conda version string.
`return_code`	The exit code conda will use for this error.
`active_prefix`	The currently active conda environment prefix, or `None` if no environment is active.
`target_prefix`	The prefix the command was operating on.
`channels`	The configured channel names at the time of error (canonical names, e.g. `defaults`, `conda-forge`).
`subdir`	The platform subdirectory (e.g., `linux-64`, `osx-arm64`).
`offline`	Whether conda is running in offline mode (`--offline`).
`dry_run`	Whether conda is running in dry-run mode (`--dry-run`).
`quiet`	Whether conda is running in quiet mode (`--quiet`).
`json`	Whether conda is running in JSON output mode (`--json`).

Warning

Observers run synchronously. `exc_traceback` is released as soon as dispatch returns, and keeping references to `exc_value` or `exc_traceback` can create reference cycles that prevent garbage collection. Capture or serialize any traceback data you need inside the callback. Do not hand these objects off to a background thread or deferred queue.

Design notes

- **Observational only** -- observers cannot change conda's behavior. This follows CPython's `sys.excepthook()` model.
- **All exception types** -- dispatch happens at the top of *`handle_exception()`*, before conda's own error-report dispatch, so observers see every exception regardless of how conda would classify it. Use `watch_for` to control scope.
- **Fault-tolerant** -- any exception (including `SystemExit`) raised by an observer is caught at the `BaseException` level, logged, and swallowed.
- **MRO-based matching** -- `watch_for` is checked against every class in the exception's method resolution order, so parent class names automatically match subclasses.
- **Frozen event object** -- `CondaExceptionEvent` is a frozen dataclass, preventing plugins from mutating shared state.
- **Optional runtime fields** -- conda-specific fields are `None` when the runtime isn't initialized, following CPython's flat args pattern (`threading.ExceptHookArgs`, `sys.UnraisableHookArgs`).

API reference

`class CondaExceptionEvent`

Structured exception event passed to exception observer plugin callbacks.

Frozen to prevent plugins from mutating exception state. Structured args follow the `threading.ExceptHookArgs` / `sys.UnraisableHookArgs` pattern for forward compatibility.

The exception triple (`exc_type`, `exc_value`, `exc_traceback`) is always populated. The remaining fields describe the conda runtime state and default to `None` when the runtime isn't initialized (e.g. `MemoryError` during early startup). Runtime fields are populated all-or-nothing: if `conda_version` is not `None`, the runtime was available and all other fields are populated (`active_prefix` may still be `None` when no environment is active).

> **Warning**
>
> Do not store references to `exc_value` or `exc_traceback` beyond the lifetime of the callback. This can create reference cycles and prevent garbage collection.

Parameters

- **exc_type** -- The exception class.
- **exc_value** -- The exception instance.
- **exc_traceback** -- The traceback object.
- **argv** -- The command-line arguments at the time of error (frozen copy of `sys.argv`). `None` if unavailable.
- **conda_version** -- The conda version string. `None` if unavailable.
- **return_code** -- The exit code conda will return for this error. `None` if unavailable.
- **active_prefix** -- The currently active conda environment prefix, or `None` if no environment is active (also `None` when the runtime is unavailable).
- **target_prefix** -- The prefix the command was operating on.
- **channels** -- The configured channel names at the time of error (canonical names, e.g. `defaults`, `conda-forge`).
- **subdir** -- The platform subdirectory (e.g., `linux-64`, `osx-arm64`).
- **offline** -- Whether conda is running in offline mode (`--offline`).
- **dry_run** -- Whether conda is running in dry-run mode (`--dry-run`).
- **quiet** -- Whether conda is running in quiet mode (`--quiet`).
- **json** -- Whether conda is running in JSON output mode (`--json`).

```
active_prefix: str | None = None

argv: tuple[str, ...] | None = None

channels: tuple[str, ...] | None = None

conda_version: str | None = None

dry_run: bool | None = None

exc_traceback: types.TracebackType

exc_type: type[BaseException]

exc_value: BaseException

json: bool | None = None

offline: bool | None = None

quiet: bool | None = None

return_code: int | None = None
```

subdir: `str | None = None`

target_prefix: `str | None = None`

class CondaExceptionObserver

Return type to use when defining a conda exception observer plugin hook.

Exception observers are purely observational, modelled after CPython's `sys.excepthook`. They cannot suppress, modify, or redirect the exception. Their return value is ignored.

For details on how this is used, see *conda_exception_observers()*.

> **Warning**
>
> Do not store references to `exc_value` or `exc_traceback` beyond the lifetime of the callback. This can create reference cycles and prevent garbage collection.

Parameters

- **name** -- Observer name (e.g., `missing-package-reporter`).
- **hook** -- Callable invoked with a *CondaExceptionEvent* instance. Must not raise; any exception is caught and logged.
- **watch_for** -- Set of exception class names this observer watches for. Matches against the full MRO. Examples:
  - `{"BaseException"}` — fires for every exception.
  - `{"Exception"}` — all standard exceptions (excludes `KeyboardInterrupt`, `SystemExit`).
  - `{"CondaError"}` — all conda errors and subclasses.
  - `{"PackagesNotFoundError"}` — a specific error and its subclasses (e.g. `PackagesNotFoundInChannelsError`).
  - `{"MemoryError"}`, `{"KeyboardInterrupt"}`, `{"SystemExit"}` — specific non-conda exceptions.
  - `{"CondaError", "MemoryError"}` — combine scopes.

  For non-`CondaError` exceptions the conda-specific fields on *CondaExceptionEvent* may be `None`.

hook: `collections.abc.Callable[[CondaExceptionEvent], None]`

name: `str`

User-facing name of the plugin used for selecting & filtering plugins and error messages.

watch_for: `set[str]`

conda_exception_observers() → collections.abc.Iterable[*conda.plugins.types.CondaExceptionObserver*]

Register exception observer callbacks in conda.

Exception observers are invoked when any exception is handled by the `ExceptionHandler`. They are **purely observational**: they cannot suppress, modify, or redirect the exception. Their return value is ignored. This follows the same model as CPython's `sys.excepthook`.

Any exception raised by an observer is caught at the `BaseException` level, logged at DEBUG, and swallowed -- a buggy plugin can never disrupt conda's error reporting path.

Observers run synchronously on the error path. They **must return promptly**. Network I/O, large file operations, or any potentially blocking call should be deferred to a daemon thread or subprocess.

Observers receive a frozen *CondaExceptionEvent* dataclass. The exception triple (`exc_type`, `exc_value`, `exc_traceback`) is always populated. Conda runtime fields (`argv`, `conda_version`, `return_code`, `active_prefix`, `target_prefix`, `channels`, `subdir`, `offline`, `dry_run`, `quiet`, `json`) are `None` when the runtime isn't initialized.

`watch_for` controls which exceptions trigger the observer via MRO matching: `{"CondaError"}` catches all conda errors, `{"BaseException"}` catches everything, `{"MemoryError"}` catches only OOM, etc. See *CondaExceptionObserver*.

Example:

```
import logging

from conda import plugins

log = logging.getLogger(__name__)


def report_missing(event):
    log.info("Missing packages: %s", event.exc_value.packages)
    log.info("Command was: %s", " ".join(event.argv))


@plugins.hookimpl
def conda_exception_observers():
    yield plugins.types.CondaExceptionObserver(
        name="missing-package-reporter",
        hook=report_missing,
        watch_for={"PackagesNotFoundInChannelsError"},
    )
```

Returns
An iterable of *CondaExceptionObserver* entries.

Health Checks

Conda doctor can be extended with the `conda_health_checks` plugin hook. Write new health checks using this hook, install the plugins you wrote, and they will run every time the `conda doctor` command is executed.

Basic Health Check

A health check requires two components:

- **name**: A human-readable name for the health check
- **action**: A function that performs the check and prints results

Example of a basic health check:

```
from conda.plugins import hookimpl
from conda.plugins.types import CondaHealthCheck
from conda.base.constants import OK_MARK, X_MARK
```

(continues on next page)

(continued from previous page)

```
def my_check(prefix: str, verbose: bool) -> None:
    """Check something in the environment."""
    if everything_ok(prefix):
        print(f"{OK_MARK} Everything looks good!")
    else:
        print(f"{X_MARK} Found a problem.")


@hookimpl
def conda_health_checks():
    yield CondaHealthCheck(
        name="My Custom Check",
        action=my_check,
    )
```

Health Checks with Fixes

Health checks can optionally provide a `fixer` function that repairs detected issues. When a user runs `conda doctor --fix`, the fixer function is called after the check.

The fixer function receives:

- **prefix**: The environment prefix path
- **args**: The parsed command-line arguments (includes `dry_run`, `yes`, etc.)

It should return an integer exit code (0 for success).

Example with a fix:

```
from conda.plugins import hookimpl
from conda.plugins.types import CondaHealthCheck
from conda.base.constants import OK_MARK, X_MARK
from conda.base.context import context
from conda.reporters import confirm_yn


def my_check(prefix: str, verbose: bool) -> None:
    if is_broken(prefix):
        print(f"{X_MARK} Something is broken.")
    else:
        print(f"{OK_MARK} All good!")


def my_fix(prefix: str, args) -> int:
    if not is_broken(prefix):
        print("Nothing to fix.")
        return 0

    print("Found issue to fix.")
    confirm_yn("Proceed with fix?", dry_run=context.dry_run)

    # Perform the fix
```

(continues on next page)

(continued from previous page)

```
    do_repair(prefix)
    print("Fixed!")
    return 0


@hookimpl
def conda_health_checks():
    yield CondaHealthCheck(
        name="My Fixable Check",
        action=my_check,
        fixer=my_fix,
        summary="Check for broken things",
        fix="Repair broken things",
    )
```

The `confirm_yn` function handles dry-run mode automatically by raising `DryRunExit` when `context.dry_run` is True.

API Reference

class CondaHealthCheck

Return type to use when defining conda health checks plugin hook.

Health checks are diagnostic actions that report on the state of a conda environment. They are invoked via `conda doctor`.

Health checks can optionally provide a fix capability, which is invoked via `conda doctor --fix` or `conda doctor --fix <name>`.

Fixer guidelines:

Fixers receive a `confirm` function that handles user confirmation and dry-run mode automatically. Simply call it with your message:

- In normal mode: Prompts the user for confirmation (default: no).
- In dry-run mode: Raises `DryRunExit` (handled by the framework).
- If user declines: Raises `CondaSystemExit` (handled by the framework).

Example

from conda.plugins.types import ConfirmCallback

def my_fixer(prefix: str, args: Namespace, confirm: ConfirmCallback) -> int:
issues = find_issues(prefix) if not issues:

print("No issues found.") return 0

print(f"Found {len(issues)} issues") confirm("Fix these issues?") # ... perform fix ... return 0

For details on how this is used, see *conda_health_checks()*.

Parameters

- **name** -- Health check identifier (e.g., `missing-files`).
- **action** -- Callable that performs the check: `action(prefix, verbose) -> None`.

- **fixer** -- Optional callable that fixes issues: `fixer(prefix, args, confirm) -> int`. The `confirm` parameter is a function to call for user confirmation. It raises an exception if the user declines or in dry-run mode.
- **summary** -- Short description of what the check detects (shown in `--list`).
- **fix** -- Short description of what the fix does (shown in `--list`).

action: `collections.abc.Callable[[str, bool], None]`

fix: `str | None = None`

fixer: `collections.abc.Callable[[str, argparse.Namespace, ConfirmCallback], int] | None = None`

name: `str`

User-facing name of the plugin used for selecting & filtering plugins and error messages.

summary: `str | None = None`

conda_health_checks() → collections.abc.Iterable[*conda.plugins.types.CondaHealthCheck*]

Register health checks for conda doctor.

This plugin hook allows you to add more "health checks" to conda doctor that you can write to diagnose problems in your conda environment. Check out the health checks already shipped with conda for inspiration.

Health checks can optionally provide a `fixer` callable that is invoked via `conda doctor --fix` or `conda doctor --fix <check-name>`.

Example (check only):

```
from conda import plugins


def example_health_check(prefix: str, verbose: bool):
    print("This is an example health check!")


@plugins.hookimpl
def conda_health_checks():
    yield plugins.types.CondaHealthCheck(
        name="example-health-check",
        action=example_health_check,
    )
```

Example (check with fix):

```
from conda import plugins


def my_health_check(prefix: str, verbose: bool):
    # Check and report issues
    print("Checking for issues...")


def my_health_fix(prefix: str, args) -> int:
    # Fix the issues
    print("Fixing issues...")
```

(continues on next page)

(continued from previous page)

```
    return 0  # exit code


@plugins.hookimpl
def conda_health_checks():
    yield plugins.types.CondaHealthCheck(
        name="my-check",
        action=my_health_check,
        fixer=my_health_fix,
        summary="Check for common issues",
        fix="Repair detected issues",
    )
```

Returns
An iterable of health check entries.

Package extractors

Package extractors handle the unpacking of different package archive formats. Each extractor specifies which file extensions it supports (e.g., `.conda`, `.tar.bz2`) and provides an extraction function.

This plugin hook allows adding support for new package formats without modifying conda's core code.

class CondaPackageExtractor

Return type to use when defining a conda package extractor plugin hook.

Package extractors handle the extraction of different package archive formats. Each extractor specifies which file extensions it supports and provides an extraction function to unpack the archive.

For details on how this is used, see *conda_package_extractors()*.

Parameters

- **name** -- Extractor name (e.g., `conda-package`, `wheel-package`).
- **extensions** -- List of file extensions this extractor handles (e.g., `[".conda", ".tar.bz2"]` or `[".whl"]`).
- **extract** -- Callable that extracts the package archive. Takes the source archive path and the destination directory where the package contents should be extracted.

extensions: list[str]

extract: PackageExtract

name: str

User-facing name of the plugin used for selecting & filtering plugins and error messages.

conda_package_extractors() → collections.abc.Iterable[*conda.plugins.types.CondaPackageExtractor*]

Register package extractors for different archive formats.

Package extractors handle the unpacking of package archives. Each extractor specifies which file extensions it supports (e.g., `.conda`, `.tar.bz2`) and provides an extraction function that takes the source archive path and destination directory.

Example:

```python
from conda import plugins
from conda.common.path import PathType


def extract_custom(source_path: PathType, destination_directory: PathType) -> None:
    # Custom extraction logic for a hypothetical package format
    ...


@plugins.hookimpl
def conda_package_extractors():
    yield plugins.types.CondaPackageExtractor(
        name="custom-package",
        extensions=[".custom"],
        extract=extract_custom,
    )
```

Returns
An iterable of *CondaPackageExtractor* entries.

Post-commands

Conda commands can be extended with the `conda_post_commands` plugin hook. By specifying the set of commands you would like to use in the `run_for` configuration option, you can execute code via the `action` option after these commands are run. The functions are provided a `command` argument representing the name of the command currently running. If the command fails for any reason, this plugin hook will not be run.

If you would like to target `conda env` commands, prefix the command name with `env_`. For example, `conda env list` would be passed to `run_for` as `env_list`.

class CondaPostCommand

Return type to use when defining a conda post-command plugin hook.

For details on how this is used, see *conda_post_commands()*.

Parameters

- **name** -- Post-command name (e.g., `custom_plugin_post_commands`).
- **action** -- Callable which contains the code to be run.
- **run_for** -- Represents the command(s) this will be run on (e.g. `install` or `create`).

action: `collections.abc.Callable[[str], None]`

name: `str`

User-facing name of the plugin used for selecting & filtering plugins and error messages.

run_for: `set[str]`

conda_post_commands() → collections.abc.Iterable[*conda.plugins.types.CondaPostCommand*]

Register post-command functions in conda.

Shell activation commands avoid loading the plugin manager for startup performance. These hooks run for `conda shell.* activate`, `deactivate`, `reactivate`, and `hook` only if the plugin manager was already loaded by another code path.

Example:

```python
from conda import plugins


def example_post_command(command):
    print("post-command action")


@plugins.hookimpl
def conda_post_commands():
    yield plugins.types.CondaPostCommand(
        name="example-post-command",
        action=example_post_command,
        run_for={"install", "create"},
    )
```

Pre-commands

Conda commands can be extended with the `conda_pre_commands` plugin hook. By specifying the set of commands you would like to use in the `run_for` configuration option, you can execute code via the `action` option before these commands are run. The functions are provided a `command` argument representing the name of the command currently running.

If you would like to target `conda env` commands, prefix the command name with `env_`. For example, `conda env list` would be passed to `run_for` as `env_list`.

class CondaPreCommand

Return type to use when defining a conda pre-command plugin hook.

For details on how this is used, see *conda_pre_commands()*.

Parameters

- **name** -- Pre-command name (e.g., `custom_plugin_pre_commands`).
- **action** -- Callable which contains the code to be run.
- **run_for** -- Represents the command(s) this will be run on (e.g. `install` or `create`).

action: `collections.abc.Callable[[str], None]`

name: `str`

User-facing name of the plugin used for selecting & filtering plugins and error messages.

run_for: `set[str]`

conda_pre_commands() → collections.abc.Iterable[*conda.plugins.types.CondaPreCommand*]

Register pre-command functions in conda.

Shell activation commands avoid loading the plugin manager for startup performance. These hooks run for `conda shell.* activate`, `deactivate`, `reactivate`, and `hook` only if the plugin manager was already loaded by another code path.

Example:

```python
from conda import plugins


def example_pre_command(command):
```

(continues on next page)

(continued from previous page)

```
    print("pre-command action")


@plugins.hookimpl
def conda_pre_commands():
    yield plugins.types.CondaPreCommand(
        name="example-pre-command",
        action=example_pre_command,
        run_for={"install", "create"},
    )
```

Pre-transactions

Solver transactions can be extended with the `conda_pre_transaction_actions` plugin hook. This plugin hook accepts a subclass of `Action` which will be instantiated and prepended to conda's transaction action list. When defining the class you must define the following methods:

- `execute`: this is the primary place to put code you wish to run during the action.
- `verify`: this is run before the action is executed. This is a good place to check for conditions that would cause the action to fail.
- `cleanup`: this is run after the action is executed. This is a good place to clean up any resources that were created during the action.
- `reverse`: in the case of a failure, this allows you to define any reversal procedures.

class CondaPreTransactionAction

Return type to use when defining a pre-transaction action hook.

For details on how this is used, see *conda_pre_transaction_actions()*.

Parameters

- **name** -- Pre transaction name (this is just a label)
- **action** -- Action class which implements plugin behavior. See *Action* for implementation details

action: *type*[*conda.core.path_actions.Action*]

name: str

User-facing name of the plugin used for selecting & filtering plugins and error messages.

conda_pre_transaction_actions() → collections.abc.Iterable[*conda.plugins.types.CondaPreTransactionAction*]

Register pre-transaction hooks.

Pre-transaction hooks run before all other actions run in a UnlinkLinkTransaction. For information about the Action class, see *Action*.

Example:

```
from conda import plugins
from conda.plugins.types import CondaPreTransactionAction
from conda.core.path_actions import Action
```

(continues on next page)

(continued from previous page)

```
class PrintAction(Action):
    def verify(self):
        print("Performing verification...")
        self._verified = True

    def execute(self):
        print(
            self.transaction_context,
            self.target_prefix,
            self.unlink_precs,
            self.link_precs,
            self.remove_specs,
            self.update_specs,
            self.neutered_specs,
        )

    def reverse(self):
        print("Reversing only happens when `execute` raises an exception.")

    def cleanup(self):
        print("Carrying out cleanup...")


class PrintActionPlugin:
    @plugins.hookimpl
    def conda_pre_transaction_actions(
        self,
    ) -> Iterable[CondaPreTransactionAction]:
        yield CondaPreTransactionAction(
            name="example-pre-transaction-action",
            action=PrintAction,
        )
```

Post-transactions

Solver transactions can be extended with the `conda_post_transaction_actions` plugin hook. This plugin hook accepts a subclass of `Action` which will be instantiated and appended to the end of conda's transaction action list. When defining the class you must define the following methods:

- `execute`: this is the primary place to put code you wish to run during the action.
- `verify`: this is run before the action is executed. This is a good place to check for conditions that would cause the action to fail.
- `cleanup`: this is run after the action is executed. This is a good place to clean up any resources that were created during the action.
- `reverse`: in the case of a failure, this allows you to define any reversal procedures.

class CondaPostTransactionAction

Return type to use when defining a post-transaction action hook.

For details on how this is used, see *conda_post_transaction_actions()*.

Parameters

- **name** -- Post transaction name (this is just a label)
- **action** -- Action class which implements plugin behavior. See *Action* for implementation details

action: *type*[*conda.core.path_actions.Action*]

name: str

User-facing name of the plugin used for selecting & filtering plugins and error messages.

conda_post_transaction_actions() → collections.abc.Iterable[*conda.plugins.types.CondaPostTransactionAction*]

Register post-transaction hooks.

Post-transaction hooks run after all other actions run in a UnlinkLinkTransaction. For information about the Action class, see *Action*.

Example:

```
from conda import plugins
from conda.plugins.types import CondaPostTransactionAction
from conda.core.path_actions import Action


class PrintAction(Action):
    def verify(self):
        print("Performing verification...")
        self._verified = True

    def execute(self):
        print(
            self.transaction_context,
            self.target_prefix,
            self.unlink_precs,
            self.link_precs,
            self.remove_specs,
            self.update_specs,
            self.neutered_specs,
        )

    def reverse(self):
        print("Reversing only happens when `execute` raises an exception.")

    def cleanup(self):
        print("Carrying out cleanup...")


class PrintActionPlugin:
    @plugins.hookimpl
    def conda_post_transaction_actions(
        self,
    ) -> Iterable[CondaPostTransactionAction]:
        yield CondaPostTransactionAction(
            name="example-post-transaction-action",
            action=PrintAction,
        )
```

`PrefixData` loaders

The `PrefixData` class exposes the contents of a given conda environment (prefix) as a series of `PrefixRecord` objects. This plugin hook allows users to write logic to expose non-conda packages as conda-like metadata. This can be useful to allow `conda list` to report additional ecosystems, like PyPI or others.

class CondaPrefixDataLoader

Define new loaders to expose non-conda packages in a given prefix as `PrefixRecord` objects.

Parameters

- **name** -- name of the loader
- **loader** -- a function that takes a prefix and a dictionary that maps package names to `PrefixRecord` objects. The newly loaded packages must be inserted in the passed dictionary accordingly, and also returned as a separate dictionary.

loader: *CondaPrefixDataLoaderCallable*

name: str

User-facing name of the plugin used for selecting & filtering plugins and error messages.

conda_prefix_data_loaders() → collections.abc.Iterable[*conda.plugins.types.CondaPrefixDataLoader*]

Register new loaders for PrefixData

The example below defines how to expose the packages installed by a hypothetical 'penguin' tool as conda packages.

Example:

```
from pathlib import Path

from conda import plugins
from conda.common.path import PathType
from conda.models.records import PrefixRecord
from conda.plugins.types import CondaPrefixDataLoader


@plugins.hookimpl
def conda_prefix_data_loaders():
    yield CondaPrefixDataLoader(
        "hypothetical",
        load_hypothetical_packages,
    )


def load_hypothetical_packages(
    prefix: PathType, records: dict[str, PrefixRecord]
) -> dict[str, PrefixRecord]:
    penguin_records = {}
    for info in Path(prefix).glob("lib/penguin/*.penguin-info"):
        name, version = info.name.rsplit("-", 1)
        kwargs = {}  # retrieve extra fields here
        penguin_records[name] = PrefixRecord(
            name=name, version=version, build_number=0, build="0", **kwargs
        )
    records.update(penguin_records)
    return penguin_records
```

Reporter Backends

Reporter backends is a plugin hook that allows you to customize the display of certain elements at the command line. Below is a list of all the elements that can currently be customized:

- Progress bar (displayed during package downloads)
- Spinner (displayed while conda is waiting for a long running task to finish)
- Confirmation input (yes/no dialogs)
- `conda info` layout

This can be configured using the `console` setting either as a command line option or by defining it in the `.condarc` file.

For more information on configuring and using reporter backends in conda itself see:

- *Settings: console*

Example plugin

The following example defines a simple plugin which uses the `pformat()` function for rendering parts of conda's output:

Hint

This is just a partial example. To see a fully functioning example of a reporter backend, checkout the *console* module.

```
from pprint import pformat

from conda import plugins
from conda.plugins.types import (
    CondaReporterBackend,
    ReporterRendererBase,
    ProgressBarBase,
)


class PprintReporterRenderer(ReporterRendererBase):
    """
    Implementation of the ReporterRendererBase abstract base class
    """

    def detail_view(self, data):
        return pformat(data)

    def envs_list(self, data):
        formatted_data = pformat(data)
        return f"Environments: {formatted_data}"

    def progress_bar(self, description, io_context_manager) -> ProgressBarBase:
        """Returns our custom progress bar implementation"""
        return PprintProgressBar(description, io_context_manager)
```

(continues on next page)

(continued from previous page)

```
class PprintProgressBar(ProgressBarBase):
    """
    Blank implementation of ProgressBarBase which does nothing.
    """

    def update_to(self, fraction) -> None:
        pass

    def refresh(self) -> None:
        pass

    def close(self) -> None:
        pass


@plugins.hookimpl
def conda_reporter_backends():
    yield CondaReporterBackend(
        name="pprint",
        description="Reporter backend based on the pprint module",
        renderer=PprintReporterRenderer,
    )
```

Below is a summary of everything we've defined:

Defining `ReporterRendererBase`

The first class we define, `PprintReporterRenderer`, is a subclass of *`ReporterRendererBase`*. The base class is an abstract base class which requires us to define our own implementations of its abstract methods. These abstract methods are used by conda when rendering output and are where all the customization we want to do occurs.

Defining `ProgressBarBase`

The second class we define is `PprintProgressBar`. For this example, it is just an empty implementation of the *`ProgressBarBase`*. Defining this effectively hides the progress bar when this reporter backend is configured. We do this in this tutorial because a full implementation would take too long to explain. Please check out *`TQDMProgressBar`* for a more realistic example using the tqdm library.

Registering the plugin hook

Finally, we define the `conda_reporter_backends` function with the `plugins.hookimpl` decorator to register our plugin which returns the `PprintReporterRenderer` class wrapped in a *`CondaReporterBackend`* object. By registering it with `name` set to `pprint`, we will be able to reference this plugin as a new backend for the `console` setting.

Using the reporter backend

To use our newly registered reporter backend, it can either be specified in our `.condarc` configuration file:

```
console: pprint
```

Or, it can be specified at the command line using the `--console` option:

```
conda info --envs --console=pprint
```

Further reading

For detailed information on how to create a conda plugin from scratch, please see the following repository which also contains a cookiecutter recipe you can use to easily bootstrap your project:

- conda-plugins-template

Below are relevant areas of the API docs for the reporter backends plugin hook:

- *CondaReporterBackend* metadata object that must be returned from the reporter backends hook definition.
- *conda_reporter_backends()* hookspec definition for reporter backends which contains an example of its usage.
- *console* our default implementation for the `console` reporter backend.
- *json* our default implementation for the `json` reporter backend.

Request Headers

The request headers plugin hooks allows plugin authors to add custom HTTP headers to HTTP requests within conda. This works by inserting these headers while preparing the requests.PreparedRequest object when calling requests.Session.request.

There are two hooks available for this purpose:

1. *conda_session_headers()*: This hook is used to define the custom headers that will be submitted with all HTTP requests (or a subset of hosts)
2. *conda_request_headers()*: This hook is used to define the custom headers that will be submitted with a specific HTTP request.

> **Warning**
>
> While the second hook (*conda_request_headers()*) can also create session headers, it is recommended to use the first hook (*conda_session_headers()*) for improved caching performance.

conda_request_headers(*host: str*, *path: str*) → collections.abc.Iterable[*conda.plugins.types.CondaRequestHeader*]

Register new HTTP request headers

The example below defines how to add HTTP headers for all requests with the hostname of `example.com` and a `path/to/endpoint.json` path.

Example:

```
from conda import plugins

HOSTS = {"example.com", "sub.example.com"}
ENDPOINT = "/path/to/endpoint.json"


@plugins.hookimpl
def conda_request_headers(host: str, path: str):
```

(continues on next page)

(continued from previous page)

```
    if host in HOSTS and path == ENDPOINT:
        yield plugins.types.CondaRequestHeader(
            name="Example-Header",
            value="example",
        )
```

conda_session_headers(*host: str*) → collections.abc.Iterable[*conda.plugins.types.CondaRequestHeader*]

Register new HTTP request headers

The example below defines how to add HTTP headers for all requests with the hostname of `example.com`.

Example:

```
from conda import plugins

HOSTS = {"example.com", "sub.example.com"}


@plugins.hookimpl
def conda_session_headers(host: str):
    if host in HOSTS:
        yield plugins.types.CondaRequestHeader(
            name="Example-Header",
            value="example",
        )
```

Both hook use the *CondaRequestHeader* class to define headers:

class CondaRequestHeader

Define vendor specific headers to include HTTP requests

For details on how this is used, see *conda_request_headers()* and *conda_session_headers()*.

Parameters

- **name** -- name of the header used in the HTTP request
- **value** -- value of the header used in the HTTP request

name: str

User-facing name of the plugin used for selecting & filtering plugins and error messages.

value: str

Settings

The settings plugin hook allows plugin authors to add new settings to conda. Users will be able to use these new parameters either in `.condarc` files or define them as environment variables. For more information on configuration in conda, see *Configuration*.

The plugin hooks relies on using the various *conda.common.configuration.Parameter* sub-classes (e.g. *conda.common.configuration.PrimitiveParameter* or *conda.common.configuration.SequenceParameter*). For more examples of how these parameter classes are used, please see the *conda.base.context.Context* class.

class CondaSetting

Return type to use when defining a conda setting plugin hook.

For details on how this is used, see *conda_settings()*.

Parameters

- **name** -- name of the setting (e.g., `config_param`)
- **description** -- description of the setting that should be targeted towards users of the plugin
- **parameter** -- Parameter instance containing the setting definition
- **aliases** -- alternative names of the setting

aliases: `tuple[str, ...] = ()`

description: `str`

name: `str`

User-facing name of the plugin used for selecting & filtering plugins and error messages.

parameter: *conda.common.configuration.Parameter*

conda_settings() → collections.abc.Iterable[*conda.plugins.types.CondaSetting*]

Register new setting

The example below defines a simple string type parameter

Example:

```
from conda import plugins
from conda.common.configuration import PrimitiveParameter, SequenceParameter


@plugins.hookimpl
def conda_settings():
    yield plugins.types.CondaSetting(
        name="example_option",
        description="This is an example option",
        parameter=PrimitiveParameter("default_value", element_type=str),
        aliases=("example_option_alias",),
    )
```

Solvers

The conda solvers can be extended with additional backends with the `conda_solvers` plugin hook. Registered solvers will be available for configuration with the `solver` configuration and `--solver` command line option.

class CondaSolver

Return type to use when defining a conda solver plugin hook.

For details on how this is used, see *conda_solvers()*.

Parameters

- **name** -- Solver name (e.g., `custom-solver`).
- **backend** -- Type that will be instantiated as the solver backend.

backend: *type*[*conda.core.solve.Solver*]

name: `str`

User-facing name of the plugin used for selecting & filtering plugins and error messages.

conda_solvers() → collections.abc.Iterable[*conda.plugins.types.CondaSolver*]

Register solvers in conda.

Example:

```
import logging

from conda import plugins
from conda.core import solve

log = logging.getLogger(__name__)


class VerboseSolver(solve.Solver):
    def solve_final_state(self, *args, **kwargs):
        log.info("My verbose solver!")
        return super().solve_final_state(*args, **kwargs)


@plugins.hookimpl
def conda_solvers():
    yield plugins.types.CondaSolver(
        name="verbose-classic",
        backend=VerboseSolver,
    )
```

Returns

An iterable of solver entries.

Subcommands

The conda CLI can be extended with the `conda_subcommands` plugin hook. Registered subcommands will be available under the `conda <subcommand>` command.

Subcommands can provide an alias or aliases for alternate command names. Aliases are registered with the same parser and action as the primary subcommand name, and must not overlap with built-in commands or other plugin subcommands.

class CondaSubcommand(** (Keyword-only parameters separator (PEP 3102)), name: str, summary: str, action: collections.abc.Callable[[argparse.Namespace], int | None], configure_parser: collections.abc.Callable[[argparse.ArgumentParser], None], aliases: str | collections.abc.Iterable[str] = ()*)

Return type to use when defining a conda subcommand plugin hook.

For details on how this is used, see *conda_subcommands()*.

Subcommands support two shapes, distinguished by `configure_parser`:

- If `configure_parser` is set, `action` receives the parsed `argparse.Namespace`.
- If `configure_parser` is omitted, `action` receives the remaining argv as `tuple[str, ...]`.

Parameters

- **name** -- Subcommand name (e.g., `conda my-subcommand-name`).
- **summary** -- Subcommand summary, will be shown in `conda --help`.

- **action** -- Callable that will be run when the subcommand is invoked.
- **aliases** -- Alternative name or names for the subcommand.
- **configure_parser** -- Callable that will be run when the subcommand parser is initialized.

action: collections.abc.Callable[[argparse.Namespace], int | None] | collections.abc.Callable[[tuple[str, ...]], int | None]

aliases: tuple[str, ...] = ()

configure_parser: collections.abc.Callable[[argparse.ArgumentParser], None] | None = None

summary: str

conda_subcommands() → collections.abc.Iterable[*conda.plugins.types.CondaSubcommand*]

Register external subcommands in conda.

Example:

```
from conda import plugins


def example_command(args):
    print("This is an example command!")


@plugins.hookimpl
def conda_subcommands():
    yield plugins.types.CondaSubcommand(
        name="example",
        aliases=("example-alias",),
        summary="example command",
        action=example_command,
    )
```

Returns
An iterable of subcommand entries.

Virtual Packages

Conda allows for the registering of virtual packages in the index data via the plugin system. This mechanism lets users write plugins that provide version identification for properties only known at runtime (e.g., OS information).

class CondaVirtualPackage

Return type to use when defining a conda virtual package plugin hook.

For details on how this is used, see *conda_virtual_packages()*.

Note

The `version` and `build` parameters can be provided in two ways:

1. Direct values: a string or `None` (where `None` translates to `0`)

2. Deferred callables: functions that return either a string, `None` (translates to `0`), or `NULL` (indicates the virtual package should not be exported)

Parameters

- **name** -- Virtual package name (e.g., `my_custom_os`).
- **version** -- Virtual package version (e.g., `1.2.3`).
- **build** -- Virtual package build string (e.g., `x86_64`).
- **override_entity** -- Can be set to either to "version" or "build", the corresponding value will be overridden if the environment variable `CONDA_OVERRIDE_<name>` is set.
- **empty_override** -- Value to use for version or build if the override environment variable is set to an empty string. By default, this is `NULL`.
- **version_validation** -- Optional version validation function to ensure that the override version follows a certain pattern.

build: `str | None | collections.abc.Callable[[], str | None | conda.auxlib._Null]`

empty_override: `None | conda.auxlib._Null`

name: `str`

User-facing name of the plugin used for selecting & filtering plugins and error messages.

override_entity: `Literal['version', 'build'] | None = None`

to_virtual_package() → *conda.models.records.PackageRecord* | conda.auxlib._Null

version: `str | None | collections.abc.Callable[[], str | None | conda.auxlib._Null]`

version_validation: `collections.abc.Callable[[str], str | None] | None = None`

conda_virtual_packages() → collections.abc.Iterable[*conda.plugins.types.CondaVirtualPackage*]

Register virtual packages in Conda.

Example:

```
from conda import plugins


@plugins.hookimpl
def conda_virtual_packages():
    yield plugins.types.CondaVirtualPackage(
        name="my_custom_os",
        version="1.2.3",
        build="x86_64",
    )
```

Returns

An iterable of virtual package entries.

More information about how plugins work

Plugins in `conda` are implemented with the use of Pluggy, a Python framework used by other projects, such as `pytest`, `tox`, and `devpi`. `pluggy` provides the ability to extend and modify the behavior of `conda` via function hooking, which results in plugin systems that are discoverable with the use of Python package entrypoints.

For more information about how it works, we suggest heading over to their documentation.

API

For even more detailed information about our plugin system, please the see the *Plugin API* section.

Error handling

Errors in `conda` are routed through *`conda.exception_handler.ExceptionHandler`*, which can print additional information about the `conda` installation when an *unexpected* exception is found. These automatic reports can be really verbose and can get in the way of communicating *expected* errors. See this issue in conda-build as an example.

To mark exceptions as *expected*, plugins should raise *`conda.CondaError`* or a subclass thereof. See conda_auth.exceptions for an example.

A note on licensing

For more information on which license to use for your custom plugin, please reference the "Choose an Open Source License" site. If you need help figuring out exactly which one to use, we advise communicating with a qualified legal professional.

Previews

Previews in conda help ensure maintainers can develop in an agile and nimble manner that increases the velocity of releasing new features. These features are hidden behind a configuration variable called, `preview`, and allow users to opt in to new functionality. During the preview stage of development, we'll typically solicit lots of user feedback that will be directly fed into our development process. When we're finally confident that we've got something everyone will love, we'll integrate the preview as a core component of conda.

The previews themselves reside in the *`conda._preview`* module. Within this module, you'll find various feature modules that we're experimenting with (e.g. *`conda._preview.env_setup`* contains a set of new commands overriding the traditional environment creation and management process). Preview modules can mirror the top-level `conda` module where that makes graduation easier, but CLI entry points should still be wired through conda's plugin framework.

> **⚠ Warning**
>
> Everything in the preview modules should be regarded as highly-experimental. These APIs are not stable and should not be relied upon!

To enable these modules, we make use of the `preview` configuration parameter that holds a list of preview features to enable. Shown below is a `.condarc` file with the `env-setup` feature enabled:

```
preview:
  - env-setup
```

Guidelines for creating previews

1. It is highly recommended to first draft a proposal as a feature request or as a collection of issues before creating a preview.
2. When creating your own preview feature, please use snake-case for the module name and kebab-case for the name that appears in the configuration file.
3. Preview subcommands should be exposed through `conda.plugins.previews`, which gates bundled preview hook implementations with their matching preview label.

Type hinting

`conda` is an old project that was created when Python type hinting didn't exist yet. As a result, many parts of its codebase are not annotated. That said, we strive to progressively improve the type annotation coverage with each contribution. On this page, we summarize the guiding principles that you should apply when adding type annotations in your contributions.

Main principles

We adhere to the Typing Best Practices recommended by the official Python documentation. Some additional tips can be found in Google's Python Guide: Type Annotations (ignore the formatting concerns, those are dealt with automatically by our pre-commit hooks).

From those tips, we emphasize:

- Be abstract on inputs and concrete on outputs (tip: explore the different abstract classes in `collections.abc`).
- Use `from __future__ import annotations` in your modules to opt-in for the most recent type hinting capabilities (e.g. deferred evaluation).
- If there's no return value, that actually means we are returning `None`. Annotate it!

Consider using multiple signatures

If your type hints are getting too complex (too many unions, many different return type hints), it may be time to consider whether you want to split the types in different signatures with `typing.overload` decorator. `conda.plugins.manager.CondaPluginManager.get_hook_results` is a good example for this use case.

Custom type hints

`conda` ships with some custom types for type hinting. They are spread around the codebase, so sometimes they are not easy to find. This is a summary:

- `conda.common.path.PathType`: Use it for values that represent file paths.
- `conda.common.path.PathTypes`: Use it for an iterable of `PathType` values.

Local development

Linters

We use `ruff` to lint and format our codebase. We have selected a few type hinting related rules, but we have not added `ANN` yet. That said, you can enable it temporarily in your `pyproject.toml` while developing locally.

Type hinting coverage

The `linux-typing` job in `tests.yml` runs MyPy and uploads the reports to codecov under the label `MyPy`. Its only purpose is to track type hinting coverage, not to report type hinting errors (yet).

In this codecov.io webapp report, you can find the coverage for every Python file in the codebase.

4.8 API Guide

As of conda 4.4, conda can be installed in any environment, not just environments with names starting with _ (underscore). That change was made, in part, so that conda can be used as a Python library.

As of conda 4.5, we do not support `pip install conda`. However, we are considering that as a supported bootstrap method in the future.

4.8.1 *conda*

OS-agnostic, system-level binary package manager.

`__main__`

Conda as a module entry point.

`_preview`

Private namespace for conda preview features.

This package contains opt-in preview features that are not yet part of the stable conda API. Code here may change or be removed across any release without deprecation notices.

Conventions

- Each preview lives in its own subdirectory named after its label, with hyphens replaced by underscores (e.g. `env-setup` → `env_setup/`).
- Each preview's `__init__.py` must expose: - `PREVIEW_LABEL: str` — the kebab-case label users put in `context.preview`.
- Preview command implementations should use conda's plugin framework. For subcommands, expose `conda_subcommands` hooks and register them through `conda.plugins.previews`. Built-in command overrides are only allowed for bundled preview subcommands routed through conda's preview plugin.

Graduation path

When a preview becomes the default behaviour, its code moves from `_preview/` into the main conda tree, the label gets a deprecation notice, and the subdirectory is removed.

`env_setup`

Preview feature: env-setup.

Scaffolding for the upcoming Rich CLI and runner facade for create/install/update/remove. Currently non-operational — all CLI entry points raise `OperationNotAllowed`.

cli

CLI modules for the env-setup preview feature.

main_create

CLI implementation for *conda create* — env-setup preview stub.

This module is part of the `env-setup` preview feature. It intercepts `conda create` when the preview is enabled and raises `OperationNotAllowed` until a functional implementation is available.

Functions

execute(args) → int	
conda_subcommands() → collections.abc.Iterable[conda.plugins.types.CondaSubcommand]	

execute(*args: argparse.Namespace*) → int

conda_subcommands() → collections.abc.Iterable[*conda.plugins.types.CondaSubcommand*]

main_install

CLI implementation for *conda install* — env-setup preview stub.

This module is part of the `env-setup` preview feature. It intercepts `conda install` when the preview is enabled and raises `OperationNotAllowed` until a functional implementation is available.

Functions

execute(args) → int	
conda_subcommands() → collections.abc.Iterable[conda.plugins.types.CondaSubcommand]	

execute(*args: argparse.Namespace*) → int

conda_subcommands() → collections.abc.Iterable[*conda.plugins.types.CondaSubcommand*]

PREVIEW_LABEL = 'env-setup'

_private

conda._private. Keep *__init__.py* empty to avoid side-effects.

If you are an API user, stop! This module has no stability guarantees. Symbols under *_conda* can be renamed or removed at any time. Instead, look for re-exported API under *conda.**.

exception_guidance

Structured error guidance for user-facing error messages.

`ErrorGuidance` can be attached to `conda.CondaError` via the `guidance` keyword so that terminal and *--json* output share the same logical cause / hint structure.

Classes

GuidanceHintTypedDict	dict() -> new empty dictionary
GuidanceHint	A single actionable hint with a stable `hint_code`.
ErrorGuidance	Structured guidance for solution-oriented error messages.

class GuidanceHintTypedDict

Bases: `TypedDict`

dict() -> new empty dictionary dict(mapping) -> new dictionary initialized from a mapping object's

(key, value) pairs

dict(iterable) -> new dictionary initialized as if via:
d = {} for k, v in iterable:

d[k] = v

dict(kwargs) -> new dictionary initialized with the name=value pairs**
in the keyword argument list. For example: dict(one=1, two=2)

Initialize self. See help(type(self)) for accurate signature.

text: *str*

hint_code: *str*

class GuidanceHint

A single actionable hint with a stable `hint_code`.

text: *str*

Human-readable description of the action to take.

hint_code: *str*

Stable machine-readable identifier. Use snake_case.

class ErrorGuidance

Structured guidance for solution-oriented error messages.

When attached to a `CondaError`, the display layer renders the cause and hints alongside the exception's `message`.

summary: *str | None = None*

Optional headline overriding `message` for user-facing output.

cause: *str | None = None*

Explanation of what went wrong, distinct from the raw error text.

hints: *tuple[GuidanceHint, ...] = ()*

Ordered list of actionable next steps. Rendered in order.

__post_init__() → None

property hint_codes: *tuple[str, ...]*

Hint codes extracted from `hints` for JSON serialization.

`__json__`() → dict[str, object]

Serialize for `CondaError.dump_map`.

classmethod **from_hints**(*hints: collections.abc.Iterable[GuidanceHint]*) → *ErrorGuidance* | None

Create guidance from hints, deduplicating by `hint_code`.

with_hints(*hints: collections.abc.Iterable[GuidanceHint]*) → *ErrorGuidance*

Return this guidance with additional hints appended.

Existing hints keep priority when `hint_code` values collide.

classmethod **coerce**(*value: Any*) → *ErrorGuidance* | None

Coerce *value* to `ErrorGuidance` or `None`.

Parameters

value -- `None` returns `None`, `ErrorGuidance` passes through, a `dict` is coerced (empty `dict` returns `None`).

Returns

`None` or an `ErrorGuidance` instance.

format(*exception:* conda.CondaError) → str

Format this guidance for terminal output.

Parameters

exception -- `str(exception)` is used as the headline unless `summary` is set.

Returns

Formatted string including the exception class name, cause, and actionable hints.

extract

Lean package extraction helpers for subprocess workers.

Functions

extract_conda_package_archive(source_full_path, ...[, ...]) → None	Extract a conda package archive without importing conda runtime state.

extract_conda_package_archive(*source_full_path: str | os.PathLike*, *destination_directory: str | os.PathLike*, *, *ensure_picklable_errors: bool = False*) → None

Extract a conda package archive without importing conda runtime state.

Parameters

- **source_full_path** -- Package archive to extract.
- **destination_directory** -- Directory to extract into.
- **ensure_picklable_errors** -- Replace exceptions that cannot survive a pickle round trip with a plain `RuntimeError`.

shards

cache

Cache suitable for shards, not allowed to change because they are named after their own sha256 hash.

Classes

AnnotatedRawShard	
ShardCache	Handle caching for individual shards (not the index of shards).

Functions

connect([dburi])	Get database connection.

Attributes

SHARD_CACHE_NAME

SHARD_CACHE_NAME = 'repodata_shards.db'

class AnnotatedRawShard(*url: str*, *package: str*, *compressed_shard: bytes*)

url

package

compressed_shard

connect(*dburi='cache.db'*)

Get database connection.

dburi: uri-style sqlite database filename; accepts certain ?= parameters.

class ShardCache(*base: pathlib.Path*, *create=True*)

Handle caching for individual shards (not the index of shards).

base: directory and filename prefix for cache.

base

__enter__()

__exit__(*exc_type*, *exc_val*, *exec_tb*)

close()

Clean up connection. ShardCache can no longer be used after close().

copy()

Copy cache with new connection. Useful for threads.

connect(*create=True*, *retry=True*)

Parameters

- **create** -- if True, create table if not exists.
- **retry** -- remove cache, log warning, and retry on error.

insert(*raw_shard:* AnnotatedRawShard)

Parameters

- **url** -- of shard
- **package** -- package name
- **raw_shard** -- msgpack.zst compressed shard data

retrieve(*url*) → *conda._private.shards.typing.ShardDict* | None

retrieve_multiple(*urls: list[str]*) → dict[str, *conda._private.shards.typing.ShardDict* | None]

Query database for cached shard urls.

Return a dict of urls in cache mapping to the Shard or None if not present.

clear_cache()

Truncate the database by removing all rows from tables

remove_cache()

Remove the sharded cache database.

decompression

Bounded decompression for sharded repodata.

Functions

decompress_shard(data, *, url, package) → bytes	Decompress an individual shard and translate errors for conda users.

Attributes

ZSTD_MAX_SHARD_SIZE
ZSTD_MAX_SHARD_INDEX_SIZE

ZSTD_MAX_SHARD_SIZE = 67108864

ZSTD_MAX_SHARD_INDEX_SIZE = 134217728

decompress_shard(*data: bytes*, *, *url: str*, *package: str*) → bytes

Decompress an individual shard and translate errors for conda users.

misc

Miscellaneous utility functions for sharded repodata processing.

This module contains utility functions that don't fit cleanly into other modules: - URL handling - Package name parsing - Data transformation helpers - Threading utilities

Functions

_shards_connections() → int	If context.repodata_threads is not set, find the size of the connection pool
_safe_urljoin_with_slash(base_url[, relative_url]) → str	Join base_url with relative_url, ensuring proper handling of all URL schemes.
_is_http_error_most_400_codes(status_code) → bool	Determine whether the *HTTPError* is an HTTP 400 error code (except for 416).
_ensure_hex_hash(record)	Convert bytes checksums to hex; leave unchanged if already str.
_iter_shard_records(shard[, repodata_version]) → collections.abc.Iterable[conda._private.shards.typing.PackageR	Yield every package record in a shard: classic `packages`,
ensure_hex_hashes(shard) → conda._private.shards.typing.ShardDict	Convert the raw-bytes hash digests of every record in `shard` to hex, in
spec_to_package_name(spec) → str | None	Given a dependency spec, return the package name, or None if the spec is
spec_to_package_extras(spec) → frozenset[str]	Given a dependency spec, return the CEP 44 extras it explicitly requests
filter_redundant_packages(repodata[, use_only_tar_bz2]) → conda._private.shards.typing.ShardDict	Given repodata or a single shard, remove any .tar.bz2 packages that have a
combine_batches_until_none(in_queue) → collections.abc.Iterator[collections.abc.Sequence[_T]]	Combine lists from in_queue until we see None. Yield combined lists.
exception_to_queue(func)	Decorator to send unhandled exceptions to the second argument out_queue.

Attributes

_T
_URLJOIN_SAFE_SCHEMES
SHARDS_CONNECTIONS_DEFAULT
HASH_FIELDS

_T

_URLJOIN_SAFE_SCHEMES

SHARDS_CONNECTIONS_DEFAULT = 10

_shards_connections() → int

If context.repodata_threads is not set, find the size of the connection pool in a typical https:// session. This should significantly reduce dropped connections. We match requests' default 10.

Is this shared between all sessions? Or do we get a different pool for a different get_session(url)?

Other adapters (file://, s3://) used in conda would have different concurrency behavior; we are not prepared to have separate threadpools per connection type.

_safe_urljoin_with_slash(*base_url: str*, *relative_url: str = ''*) → str

Join base_url with relative_url, ensuring proper handling of all URL schemes.

Python's urllib.parse.urljoin only handles schemes registered in `urllib.parse.uses_relative`. For unregistered schemes like `s3://`, it returns just `"."` instead of the resolved URL. This function falls back to a scheme-swap workaround for those cases.

The result always ends with "/" to enable proper string concatenation with filenames.

See: https://github.com/conda/conda-libmamba-solver/issues/866

_is_http_error_most_400_codes(*status_code: str | int*) → bool

Determine whether the *HTTPError* is an HTTP 400 error code (except for 416).

HASH_FIELDS = ('sha256', 'md5', 'legacy_bz2_md5', 'attestations_sha256')

_ensure_hex_hash(*record:* conda._private.shards.typing.PackageRecordDict)

Convert bytes checksums to hex; leave unchanged if already str.

_iter_shard_records(*shard:* conda._private.shards.typing.ShardDict, *repodata_version: int = 1*) → collections.abc.Iterable[*conda._private.shards.typing.PackageRecordDict*]

Yield every package record in a shard: classic `packages`, `packages.conda`, and (when `repodata_version >= 3`) all `v3` groups.

ensure_hex_hashes(*shard:* conda._private.shards.typing.ShardDict) → *conda._private.shards.typing.ShardDict*

Convert the raw-bytes hash digests of every record in `shard` to hex, in place, when the shard is decoded for use.

Shards are msgpack, where rattler (e.g. `rattler-index`) encodes hash digests as bytes or arrays of ints; `repodata.json` and downstream consumers expect hex strings. Calling this once as a shard is decoded keeps every reader (`iter_records_v3`, `build_repodata`, ...) free of raw bytes, and avoids re-normalizing the same records on each read. The shard cache stores the original compressed bytes unchanged; only the decoded in-memory dict is modified.

spec_to_package_name(*spec: str*) → str | None

Given a dependency spec, return the package name, or None if the spec is not parseable.

Uses conda's MatchSpec rather than libmambapy to avoid a hard dependency on a solver backend. With @functools.cache the performance is equivalent (benchmarked at ~10ms for 5000 unique specs either way).

spec_to_package_extras(*spec: str*) → frozenset[str]

Given a dependency spec, return the CEP 44 extras it explicitly requests (e.g. `pkg[extras=more]` -> `frozenset({"more"})`), or an empty set if there are none or the spec is not parseable.

Not cached with `functools.cache`: which extras syntaxes parse depends on the active MatchSpec parser (see `context.solver`), and MatchSpec already caches parse results per parser. Specs without `extras` skip parsing.

filter_redundant_packages(*repodata:* conda._private.shards.typing.ShardDict, *use_only_tar_bz2=False*) → *conda._private.shards.typing.ShardDict*

Given repodata or a single shard, remove any .tar.bz2 packages that have a .conda counterpart.

Also filters redundant stems under `v3["tar.bz2"]` when a matching stem exists in `v3["conda"]`, `packages.conda`, or classic `packages`.

Return a shallow copy if use_only_tar_bz2==False, else unmodified input.

combine_batches_until_none(*in_queue: queue.SimpleQueue[collections.abc.Sequence[_T] | None]*) → collections.abc.Iterator[collections.abc.Sequence[*_T*]]

Combine lists from in_queue until we see None. Yield combined lists.

exception_to_queue(*func*)

Decorator to send unhandled exceptions to the second argument out_queue.

shards

Models for sharded repodata, and to make monolithic repodata look like sharded repodata.

Classes

ShardFetch	Wrapper class that encapsulates fetching and caching of individual shards.
ShardBase	Abstract base class for shard-like objects.
ShardLike	Present a "classic" repodata.json as per-package shards.
Shards	Handle repodata_shards.msgpack.zst and individual per-package shards.

Functions

shard_mentioned_packages(shard[, extra, ...]) → collections.abc.Iterable[str]	Return all dependency names mentioned in a shard, not including the shard's
shard_mentioned_package_extras(shard[, extras, ...]) → collections.abc.Iterable[tuple[str, frozenset[str]]]	Return `(name, extras)` for each dependency spec in a shard, where
_shards_base_url(url, shards_base_url) → str	Return shards_base_url joined with base_url and url.
_repodata_shards(url, cache) → bytes	Fetch shards index with cache.
_shards_from_bytes(shards_data, shards_index_url) → Shards	Helper func to generate shards from shards byte data
fetch_shards_index(sd) → Shards | None	Check a SubdirData's URL for shards.
batch_retrieve_from_cache(shardlikes, packages, ...) → list[ShardFetch]	Given a list of ShardBase objects and a list of package names, fetch all URLs
batch_retrieve_from_network(wanted)	Fetch all shards in the wanted list from the network.
fetch_channels(url_to_channel, *[, require_shards]) → dict[str, ShardBase] | None	

class ShardFetch(*shardbase:* ShardBase, *package:* *str*, *shard_cache:* conda._private.shards.cache.ShardCache | *None* = *None*)

Wrapper class that encapsulates fetching and caching of individual shards.

Handles deferred fetching: shards can be requested via this class but will only be actually retrieved from the network when fetch() is called. This allows batching and coordinating shard retrieval across multiple channels.

Initialize a ShardFetch wrapper.

Parameters

- **shardbase** -- The ShardBase (Shards or ShardLike) instance
- **package** -- The package name to fetch
- **shard_cache** -- Optional cache to use for storage (required for Shards)

shardbase

package

url

shard_cache = None

_shard: *conda._private.shards.typing.ShardDict* | None = None

_fetched = False

fetch() → *conda._private.shards.typing.ShardDict*

Fetch the shard from the network or return cached result.

For Shards, performs the actual network fetch. For ShardLike, returns the shard immediately since it's in memory.

_fetch_from_shards() → *conda._private.shards.typing.ShardDict*

Fetch a single shard from a Shards instance.

_fetch_shards_impl(*packages: collections.abc.Iterable[str]*) → dict[str, *conda._private.shards.typing.ShardDict*]

Fetch multiple shards for a Shards instance.

Implements the core fetching logic for Shards, handling network requests, caching, and decompression.

_process_fetch_result(*future*, *url*, *package*, *results*, *shards*)

Process a single fetched shard result.

static fetch_batch(*shard_fetches: collections.abc.Iterable[*ShardFetch*]*) → None

Batch fetch multiple ShardFetch objects, grouping by ShardBase.

This efficiently fetches shards from multiple sources by grouping requests by their ShardBase instance and making coordinated network calls.

shard_mentioned_packages(*shard:* conda._private.shards.typing.ShardDict, *extra: collections.abc.Iterable[str] = ()*, *spec_to_package_name=spec_to_package_name*, *repodata_version: int = 1*) → collections.abc.Iterable[str]

Return all dependency names mentioned in a shard, not including the shard's own package name.

Callers are responsible for calling `conda._private.shards.misc.filter_redundant_packages` on `shard` first if "prefer .conda over .tar.bz2" semantics are required; *conda._private.shards.subset* already does this for every shard before invoking this function.

shard_mentioned_package_extras(*shard:* conda._private.shards.typing.ShardDict, *extras: collections.abc.Iterable[str] = ()*, *include_depends: bool = True*, *spec_to_package_name=spec_to_package_name*, *repodata_version: int = 1*) → collections.abc.Iterable[tuple[str, frozenset[str]]]

Return `(name, extras)` for each dependency spec in a shard, where `extras` are the CEP 44 extras explicitly requested by that spec (e.g. `pkg[extras=more]` -> `("pkg", frozenset({"more"}))`).

Specs come from every record's `depends` (when `include_depends` is true) and from the `extra_depends` groups named by `extras`. Groups are unioned across all builds/versions in the shard. Only groups named in `extras` are followed. Other groups stay inactive.

Callers are responsible for calling `conda._private.shards.misc.filter_redundant_packages` on `shard` first.

class ShardBase

Bases: abc.ABC

Abstract base class for shard-like objects.

Defines the common interface for both sharded repodata (Shards) and monolithic repodata presented as shards (ShardLike).

url: str

repodata_no_packages: *conda._private.shards.typing.RepodataDict*

visited: dict[str, *conda._private.shards.typing.ShardDict* | None]

_base_url: str

property package_names: collections.abc.KeysView[str]

Abstractmethod

Return the names of all packages available in this shard collection.

property base_url: str

Return self.url joined with base_url from repodata, or self.url if no base_url was present. Packages are found here.

Note base_url can be a relative or an absolute url. Uses _safe_urljoin_with_slash to handle non-HTTP schemes (s3://, etc.).

__contains__(*package: str*) → bool

Check if a package is available in this shard collection.

abstractmethod shard_url(*package: str*) → str

Return shard URL for a given package. For monolithic repodata, should not be fetched but is a unique identifier.

Raise KeyError if package is not in the index.

abstractmethod shard_loaded(*package: str*) → bool

Return True if the given package's shard is in memory.

visit_package(*package: str*) → *conda._private.shards.typing.ShardDict*

Return a shard that is already loaded in memory and mark as visited.

visit_shard(*package: str*, *shard:* conda._private.shards.typing.ShardDict)

Store new shard data in the visited dict.

build_repodata() → *conda._private.shards.typing.RepodataDict*

Return monolithic repodata including all visited shards.

Does not return "v3" repodata.

Prefer iter_records_v3() over this method.

iter_records() → collections.abc.Iterable[tuple[str, dict]]

Yield (key, record) tuples for all packages in visited shards, including v3 records.

For classic packages, *key* is the filename. For v3 packages, *key* is the shard key (not necessarily the download filename; see `record["fn"]`).

Prefer *iter_records_v3()* when section information is needed.

iter_records_v3() → collections.abc.Iterable[tuple[tuple[str, str], dict]]

Yield ((key, section), record) tuples for all packages in visited shards.

Section can be: "packages" for .tar.bz2 packages, "packages.conda" for .conda packages, "v3.whl", "v3.conda", "v3.tar.bz2" for v3 packages.

key is the same as the filename for "packages", "packages.conda" but is different from the filename for v3 packages.

class ShardLike(*repodata:* conda._private.shards.typing.RepodataDict, *url: str = ''*)

Bases: *ShardBase*

Present a "classic" repodata.json as per-package shards.

url: must be unique for all ShardLike used together.

repodata_no_packages: *conda._private.shards.typing.RepodataDict*

url = ''

shards: dict[str, *conda._private.shards.typing.ShardDict*]

visited: dict[str, *conda._private.shards.typing.ShardDict* | None]

__repr__()

property package_names: collections.abc.KeysView[str]

Return the names of all packages available in this shard collection.

shard_url(*package: str*) → str

Return shard URL for a given package.

Raise KeyError if package is not in the index.

shard_loaded(*package: str*) → bool

Return True if the given package's shard is in memory.

visit_package(*package: str*) → *conda._private.shards.typing.ShardDict*

Return a shard that is already in memory and mark as visited.

_shards_base_url(*url*, *shards_base_url*) → str

Return shards_base_url joined with base_url and url. Note shards_base_url can be a relative or an absolute url. Uses _safe_urljoin_with_slash to handle non-HTTP schemes (s3://, etc.).

class Shards(*shards_index:* conda._private.shards.typing.ShardsIndexDict, *url: str*)

Bases: *ShardBase*

Handle repodata_shards.msgpack.zst and individual per-package shards.

Parameters

- **shards_index** -- raw parsed msgpack dict. Don't change it or base_url,
- **wrong.** (*shards_base_url will be*)
- **url** -- URL of repodata_shards.msgpack.zst

_shards_base_url: str

shards_index

url

_base_url

session

repodata_no_packages

visited: dict[str, *conda._private.shards.typing.ShardDict* | None]

_shard_url_cache: dict[str, str]

property package_names

Return the names of all packages available in this shard collection.

property packages_index

property shards_base_url: str

Return self.url joined with shards_base_url. Note shards_base_url can be a relative or an absolute url.

shard_url(*package: str*) → str

Return shard URL for a given package.

Raise KeyError if package is not in the index.

shard_loaded(*package: str*) → bool

Return True if the given package's shard is in memory.

visit_package(*package: str*) → *conda._private.shards.typing.ShardDict*

Return a shard that is already in memory and mark as visited.

_repodata_shards(*url*, *cache:* conda.gateways.repodata.RepodataCache) → bytes

Fetch shards index with cache.

Update cache state.

Return shards data, either newly fetched or from cache.

In offline mode, returns cached data even if expired. If no cache exists in offline mode, raises RepodataIsEmpty to signal unavailability.

_shards_from_bytes(*shards_data: bytes*, *shards_index_url: str*) → *Shards*

Helper func to generate shards from shards byte data

fetch_shards_index(*sd:* conda.core.subdir_data.SubdirData) → *Shards* | None

Check a SubdirData's URL for shards.

Return shards index bytes from cache or network. Return None if not found; caller should fetch normal repodata.

TODO: If this function fails to retrieve the sharded repodata index file, it will
mark it is as not supporting this feature in cache. This can problematic because sometimes server errors can happen which will lead it to wrongly assuming the channel doesn't support sharding. We need to rethink our logic for determining shard support.

batch_retrieve_from_cache(*shardlikes: collections.abc.Sequence[*ShardBase*]*, *packages: list[str]*, *shard_cache:* conda._private.shards.cache.ShardCache) → list[*ShardFetch*]

Given a list of ShardBase objects and a list of package names, fetch all URLs from a shared local cache, and update shardlikes with those per-package shards. Return ShardFetch objects for items not found in cache (to be fetched from network).

batch_retrieve_from_network(*wanted: list[*ShardFetch*]*)

Fetch all shards in the wanted list from the network.

Coordinate batch fetching across multiple ShardBase instances.

fetch_channels(*url_to_channel: dict[str,* conda.models.channel.Channel*]*, ***, *require_shards: bool = True*) → dict[str, *ShardBase*] | None

Parameters

- **url_to_channel** -- not modified, must already be expanded to subdirs.

- **require_shards** -- Return None unless at least one channel provides shards.

Attempt to fetch the sharded index first and then fall back to retrieving a monolithic *repodata.json* file.

Returns

A dict mapping channel URLs to *Shard* or *ShardLike* objects. If *require_shards* is true, return None when no channels have shards. This dict preserves the key order of the input *url_to_channel*.

subset

Sharded repodata subsets.

Traverse dependencies of installed and to-be-installed packages to generate a useful subset for the solver.

The algorithm developed here is a direct result of the following CEP:

- https://conda.org/learn/ceps/cep-0016 (Sharded Repodata)

In this algorithm we treat a (channel, package name) as a node, its dependencies as edges. We then traverse all edges to discover all reachable (channel, package name) tuples. The solver should be able to find a solution with only this subset.

This subset is overgenerous since the user is unlikely to want to install very old packages and their dependencies. If this is too slow, we could deploy heuristics that automatically ignore older package versions. We could also allow the user to configure minimum versions of common packages and ignore older versions and their dependencies, falling back to a full solve if unsatisfiable.

We treat both sharded and monolithic repodata as if they were made up of per-package shards, computing a subset of both. This is because it is possible for the monolithic repodata to mention packages that exist in the true sharded repodata but would not be found by only traversing the shards.

We treat all repodata as sharded, even if no actual sharded repodata has been found.

Example usage

The following constructs several repodata (*noarch* and *linux-64*) from a single channel name and a list of root packages:

``` from conda.models.channel import Channel from _conda.shards_subset import build_repodata_subset

channel = Channel("conda-forge-sharded/linux-64") channel_data = build_repodata_subset(["python", "pandas"], [channel.url()]) repodata = {}

**for url in channel_data:**
    repodata[url] = channel_data.build_repodata()

# ... this is what's fed to the solver ```

## Classes

| | |
|---|---|
| *Node* | |
| *NodeId* | |
| *RepodataSubset* | Build a subset of repodata by traversing all packages that are dependencies |
| *QueueCache* | Implement insert() interface of .cache.ShardCache() as a queue, instead of |
```

Functions

_nodes_from_packages(root_packages, shardlikes[, ...]) → collections.abc.Iterator[tuple[NodeId, Node]]	Yield (NodeId, Node) for all root packages found in shardlikes.
build_repodata_subset(root_packages, channels[, ...]) → dict[str, conda._private.shards.shards.ShardBase] | None	Retrieve all necessary information to build a repodata subset.
cache_fetch_thread(in_queue, shard_out_queue, ...)	Fetch batches of shards from cache until in_queue sees None. Enqueue found
network_fetch_thread(in_queue, shard_out_queue, cache, ...)	Fetch shards from the network that are received on in_queue, until we see
offline_nofetch_thread(in_queue, shard_out_queue, ...)	For offline mode, where network requests are not allowed.

Attributes

THREAD_WAIT_TIMEOUT
REACHABLE_PIPELINED_MAX_TIMEOUTS
QUEUE_TIMEOUT
_T

```
THREAD_WAIT_TIMEOUT = 5

REACHABLE_PIPELINED_MAX_TIMEOUTS = 10

QUEUE_TIMEOUT = 1

class Node

    distance:  int = 9223372036854775807

    package:  str = ''

    channel:  str = ''

    visited:  bool = False

    shard_url:  str = ''

    requested_extras:  frozenset[str]

    expanded_extras:  frozenset[str]

    expanded:  bool = False

    to_id() → NodeId

class NodeId

    package:  str

    channel:  str

    shard_url:  str = ''
```

__hash__()

_nodes_from_packages(*root_packages: list[str]*, *shardlikes: collections.abc.Iterable[conda._private.shards.shards.ShardBase]*, *root_extras: collections.abc.Mapping[str, collections.abc.Iterable[str]] | None = None*) → collections.abc.Iterator[tuple[*NodeId*, *Node*]]

Yield (NodeId, Node) for all root packages found in shardlikes.

class RepodataSubset(*shardlikes: collections.abc.Iterable[conda._private.shards.shards.ShardBase]*, *spec_to_package_name: collections.abc.Callable[[str], str] = spec_to_package_name*, *repodata_version: int = 1*, *depth: int = sys.maxsize*)

Build a subset of repodata by traversing all packages that are dependencies and transitive dependencies of a root set of packages.

shardlikes: collections.abc.Sequence[*conda._private.shards.shards.ShardBase*]

DEFAULT_STRATEGY = 'pipelined'

_nodes: dict[*NodeId*, *Node*]

_use_only_tar_bz2: bool

_add_pip_as_python_dependency: bool

_spec_to_package_name: collections.abc.Callable[[str], str]

_repodata_version = 1

depth = 9223372036854775807

property node_count: int

Number of (channel, package) nodes discovered during traversal.

classmethod has_strategy(*strategy: str*) → bool

Return True if this class provides the named shard traversal strategy.

_expansion_specs(*node:* Node, *shard:* conda._private.shards.typing.ShardDict) → collections.abc.Iterator[tuple[str, frozenset[str]]]

Yield `(package name, extras requested by the spec)` for everything reachable from `shard` that has not been traversed for `node` yet.

On the first expansion that is the shard's `depends` plus every group in `node.requested_extras`. When the node is revisited because another dependency activated additional groups, only those new groups are traversed. Records the groups as expanded on the node.

_neighbors(*node:* Node) → collections.abc.Iterator[*Node*]

Retrieve all unvisited neighbors of a node, plus already visited neighbors that need to be revisited because this node's dependency specs activate additional extras groups on them.

Neighbors in the context are dependencies of a package.

NOTE: This method assumes that the required shards have already been retrieved from the network via batch_retrieve_from_network() before neighbors() is called. It uses visit_package() to access already-loaded shards.

_outgoing(*node:* Node)

All nodes that can be reached by this node, plus cost.

reachable(*root_packages*, *, *strategy=DEFAULT_STRATEGY*, *root_extras: collections.abc.Mapping[str, collections.abc.Iterable[str]] | None = None*) → None

Run named reachability strategy or the default.

Update *self.shardlikes* with reachable package records. Later, [shardlike.build_repodata() for shardlike in shardlikes] can be used to generate repodata.json-format subsets of each channel.

Parameters

- **root_packages** -- installed and requested package names.
- **strategy** -- traversal algorithm, "bfs" or "pipelined".
- **root_extras** -- optional mapping of root package name to an iterable of CEP 44 extras requested for it (e.g. *{"package": ["extras_group"]}*). Extras explicitly set on dependency specs of fetched packages (e.g. *pkg-u[extras=more]*) are followed transitively; groups that nobody requested stay inactive.

reachable_bfs(*root_packages*, *root_extras: collections.abc.Mapping[str, collections.abc.Iterable[str]] | None = None*)

Fetch all packages reachable from *root_packages*' by following dependencies using the "breadth-first search" algorithm.

Update associated *self.shardlikes* to contain enough data to build a repodata subset.

_reachable_bfs(*root_packages*, *shard_cache: conda._private.shards.cache.ShardCache*, *root_extras: collections.abc.Mapping[str, collections.abc.Iterable[str]] | None = None*)

Inner reachable_bfs() implementation.

reachable_pipelined(*root_packages*, *root_extras: collections.abc.Mapping[str, collections.abc.Iterable[str]] | None = None*)

Fetch all packages reachable from *root_packages*' by following dependencies.

Build repodata subset using concurrent threads to follow dependencies, fetch from cache, and fetch from network.

_reachable_pipelined(*root_packages*, *network_worker: collections.abc.Callable[[queue.SimpleQueue[collections.abc.Sequence[NodeId] | None], queue.SimpleQueue[list[tuple[NodeId, conda._private.shards.typing.ShardDict] | Exception]], RepodataSubset._reachable_pipelined.cache, collections.abc.Sequence[conda._private.shards.shards.ShardBase]], None]*, *cache: RepodataSubset._reachable_pipelined.cache*, *root_extras: collections.abc.Mapping[str, collections.abc.Iterable[str]] | None = None*)

Set up queues and threads for shard traversal with a configurable network_worker. Called by reachable_pipelined()

_pipelined_traversal(*root_packages*, *cache_in_queue: queue.SimpleQueue[list[NodeId] | None]*, *shard_out_queue: queue.SimpleQueue[list[tuple[NodeId, conda._private.shards.typing.ShardDict]] | Exception]*, *cache_thread: threading.Thread*, *network_thread: threading.Thread*, *root_extras: collections.abc.Mapping[str, collections.abc.Iterable[str]] | None = None*)

Run reachability algorithm given queues to submit and receive shards.

_visit_node(*parent_node: Node*, *mentioned_packages: collections.abc.Iterable[tuple[str, frozenset[str]]]*, *extras_by_package: collections.abc.Mapping[str, collections.abc.Iterable[str]] | None = None*) → collections.abc.Iterable[*NodeId*]

Broadcast mentioned packages across channels. yield pending NodeId's.

`mentioned_packages` are `(name, extras)` pairs; extras are the CEP 44 groups requested for that package by the spec that mentions it. A package that already has a node only yields again if the extras add groups to a node whose shard was already processed.

_drain_pending(*pending: set[*NodeId*]*, *shardlikes_by_url: dict[str,* conda._private.shards.shards.ShardBase*]*) → tuple[list[tuple[*NodeId*, *conda._private.shards.typing.ShardDict*]], list[*NodeId*]]

Check pending for in-memory shards. Clear pending.

Return a list of shards we have and shards we need to fetch.

build_repodata_subset(*root_packages: collections.abc.Iterable[str]*, *channels: dict[str,* conda.models.channel.Channel*]*, *algorithm: Literal['bfs', 'pipelined'] = RepodataSubset.DEFAULT_STRATEGY*, *spec_to_package_name_func: collections.abc.Callable[[str], str] = spec_to_package_name*, *repodata_version: int = 1*, *depth: int = sys.maxsize*, *root_extras: collections.abc.Mapping[str, collections.abc.Iterable[str]] | None = None*) → dict[str, *conda._private.shards.shards.ShardBase*] | None

Retrieve all necessary information to build a repodata subset.

This function implements the conda.gateways.shards.BuildRepodataSubset protocol, allowing it to be passed to solvers that support sharded repodata optimization.

Params:

root_packages: iterable of installed and requested package names channels: Channel objects; dict form preferred. algorithm: desired traversal algorithm ("bfs" or "pipelined") spec_to_package_name_func: callable to convert package specs to names.

Defaults to the standard spec_to_package_name.

repodata_version: repodata format version (1 = classic, 3 = v3). depth: the maximum depth of dependant packages to include in the repodata

subset.

root_extras: optional mapping of root package name to an iterable of

CEP 44 extras requested for it (e.g. *{"package": ["extras_group"]}*). Extras on dependency specs are followed transitively; see *RepodataSubset.reachable*.

Returns

None if there are no shards available, or a mapping of channel URL's to ShardBase objects where build_repodata() returns the computed subset.

_T

class QueueCache(*queue*)

Implement insert() interface of .cache.ShardCache() as a queue, instead of giving network thread direct access to the database.

queue: queue.SimpleQueue

insert(*shard:* conda._private.shards.cache.AnnotatedRawShard)

copy()

__enter__()

__exit__(*_)

cache_fetch_thread(*in_queue: queue.SimpleQueue[collections.abc.Sequence[*NodeId*] | None]*, *shard_out_queue: queue.SimpleQueue[collections.abc.Sequence[tuple[*NodeId*, conda._private.shards.typing.ShardDict*] | Exception]]*, *network_out_queue: queue.SimpleQueue[collections.abc.Sequence[*NodeId*] | None]*, *cache: conda._private.shards.cache.ShardCache*)

Fetch batches of shards from cache until in_queue sees None. Enqueue found shards to shard_out_queue, and not found shards to network_out_queue.

When we see None on in_queue, send None to both out queues and exit.

Parameters

- **in_queue** -- NodeId (URLs) to fetch.
- **shard_out_queue** -- fetched shards sent to queue.
- **network_out_queue** -- cache misses forwarded to queue. Same queue is network_fetch_thread's in_queue.
- **cache** -- used to retrieve shards.

network_fetch_thread(*in_queue: queue.SimpleQueue[collections.abc.Sequence[*NodeId*] | None]*, *shard_out_queue: queue.SimpleQueue[list[tuple[*NodeId*, conda._private.shards.typing.ShardDict*] | Exception]]*, *cache: conda._private.shards.cache.ShardCache | QueueCache*, *shardlikes: collections.abc.Sequence[conda._private.shards.shards.ShardBase]*)

Fetch shards from the network that are received on in_queue, until we see None.

Unhandled exceptions also go to shard_out_queue, and exit this thread.

Parameters

- **in_queue** -- NodeId (URLs) to fetch.
- **shard_out_queue** -- fetched shards sent to queue.
- **cache** -- once shards are decoded they are stored in cache.
- **shardlikes** -- list of (network-only) shard index objects.

offline_nofetch_thread(*in_queue: queue.SimpleQueue[collections.abc.Sequence[*NodeId*] | None]*, *shard_out_queue: queue.SimpleQueue[list[tuple[*NodeId*, conda._private.shards.typing.ShardDict*] | Exception]]*, *cache: conda._private.shards.cache.ShardCache*, *shardlikes: collections.abc.Sequence[conda._private.shards.shards.ShardBase]*)

For offline mode, where network requests are not allowed. Pretend that every network request is an empty shard. Don't save those to the cache.

Depending on how many shards are in sqlite3 and which packages were requested, the user may or may not get enough repodata for a solution.

Parameters

- **in_queue** -- NodeId (URLs) to fetch.
- **shard_out_queue** -- fetched shards sent to queue.
- **cache** -- once shards are decoded they are stored in cache.
- **shardlikes** -- list of (network-only) shard index objects.

typing

TypedDict declarations for shards.

These are helpful for auto-complete, but do not validate at runtime and are not normative. They are intentionally not shared with another project (conda) to reduce coupling.

Classes

PackageRecordDict	Basic package attributes that this module cares about.
ShardDict	dict() -> new empty dictionary
RepodataInfoDict	dict() -> new empty dictionary
RepodataDict	Packages plus info.
ShardsIndexDict	Shards index as deserialized from repodata_shards.msgpack.zst

Attributes

_ShardDictBase

class PackageRecordDict

Bases: TypedDict

Basic package attributes that this module cares about.

Initialize self. See help(type(self)) for accurate signature.

name: str

version: str

build: str

build_number: int

sha256: NotRequired[str | bytes]

md5: NotRequired[str | bytes]

legacy_bz2_md5: NotRequired[str | bytes]

attestations_sha256: NotRequired[str | bytes]

depends: NotRequired[list[str]]

constrains: NotRequired[list[str]]

extra_depends: NotRequired[dict[str, list[str]]]

noarch: NotRequired[str]

_ShardDictBase

class ShardDict

Bases: *_ShardDictBase*

dict() -> new empty dictionary dict(mapping) -> new dictionary initialized from a mapping object's

(key, value) pairs

dict(iterable) -> new dictionary initialized as if via:
d = {} for k, v in iterable:

d[k] = v

dict(kwargs) -> new dictionary initialized with the name=value pairs**
in the keyword argument list. For example: dict(one=1, two=2)

Initialize self. See help(type(self)) for accurate signature.

v3: `dict[str, dict[str, PackageRecordDict]]`

class RepodataInfoDict

Bases: TypedDict

dict() -> new empty dictionary dict(mapping) -> new dictionary initialized from a mapping object's

(key, value) pairs

dict(iterable) -> new dictionary initialized as if via:
d = {} for k, v in iterable:

d[k] = v

dict(kwargs) -> new dictionary initialized with the name=value pairs**
in the keyword argument list. For example: dict(one=1, two=2)

Initialize self. See help(type(self)) for accurate signature.

base_url: `str`

shards_base_url: `str`

subdir: `str`

class RepodataDict

Bases: *ShardDict*

Packages plus info.

Initialize self. See help(type(self)) for accurate signature.

info: *RepodataInfoDict*

repodata_version: `int`

class ShardsIndexDict

Bases: TypedDict

Shards index as deserialized from repodata_shards.msgpack.zst

Initialize self. See help(type(self)) for accurate signature.

info: *RepodataInfoDict*

version: int

shards: dict[str, bytes]

Classes

ShardBase	Abstract base class for shard-like objects.
ShardLike	Present a "classic" repodata.json as per-package shards.
Shards	Handle repodata_shards.msgpack.zst and individual per-package shards.

Functions

fetch_channels(url_to_channel, *[, require_shards]) → dict[str, ShardBase] | None	
fetch_shards_index(sd) → Shards | None	Check a SubdirData's URL for shards.

class ShardBase

Bases: abc.ABC

Abstract base class for shard-like objects.

Defines the common interface for both sharded repodata (Shards) and monolithic repodata presented as shards (ShardLike).

url: str

repodata_no_packages: *conda._private.shards.typing.RepodataDict*

visited: dict[str, *conda._private.shards.typing.ShardDict* | None]

_base_url: str

property package_names: collections.abc.KeysView[str]

Abstractmethod

Return the names of all packages available in this shard collection.

property base_url: str

Return self.url joined with base_url from repodata, or self.url if no base_url was present. Packages are found here.

Note base_url can be a relative or an absolute url. Uses _safe_urljoin_with_slash to handle non-HTTP schemes (s3://, etc.).

__contains__(*package: str*) → bool

Check if a package is available in this shard collection.

abstractmethod shard_url(*package: str*) → str

Return shard URL for a given package. For monolithic repodata, should not be fetched but is a unique identifier.

Raise KeyError if package is not in the index.

abstractmethod shard_loaded(*package: str*) → bool

Return True if the given package's shard is in memory.

visit_package(*package: str*) → *conda._private.shards.typing.ShardDict*

Return a shard that is already loaded in memory and mark as visited.

visit_shard(*package: str*, *shard:* conda._private.shards.typing.ShardDict)

Store new shard data in the visited dict.

build_repodata() → *conda._private.shards.typing.RepodataDict*

Return monolithic repodata including all visited shards.

Does not return "v3" repodata.

Prefer iter_records_v3() over this method.

iter_records() → collections.abc.Iterable[tuple[str, dict]]

Yield (key, record) tuples for all packages in visited shards, including v3 records.

For classic packages, *key* is the filename. For v3 packages, *key* is the shard key (not necessarily the download filename; see `record["fn"]`).

Prefer *iter_records_v3()* when section information is needed.

iter_records_v3() → collections.abc.Iterable[tuple[tuple[str, str], dict]]

Yield ((key, section), record) tuples for all packages in visited shards.

Section can be: "packages" for .tar.bz2 packages, "packages.conda" for .conda packages, "v3.whl", "v3.conda", "v3.tar.bz2" for v3 packages.

key is the same as the filename for "packages", "packages.conda" but is different from the filename for v3 packages.

class ShardLike(*repodata:* conda._private.shards.typing.RepodataDict, *url: str = ''*)

Bases: *ShardBase*

Present a "classic" repodata.json as per-package shards.

url: must be unique for all ShardLike used together.

repodata_no_packages: *conda._private.shards.typing.RepodataDict*

url = ''

shards: dict[str, *conda._private.shards.typing.ShardDict*]

visited: dict[str, *conda._private.shards.typing.ShardDict* | None]

__repr__()

property package_names: collections.abc.KeysView[str]

Return the names of all packages available in this shard collection.

shard_url(*package: str*) → str

Return shard URL for a given package.

Raise KeyError if package is not in the index.

shard_loaded(*package: str*) → bool

Return True if the given package's shard is in memory.

visit_package(*package: str*) → *conda._private.shards.typing.ShardDict*

Return a shard that is already in memory and mark as visited.

class Shards(*shards_index:* conda._private.shards.typing.ShardsIndexDict, *url: str*)

Bases: *ShardBase*

Handle repodata_shards.msgpack.zst and individual per-package shards.

Parameters

- **shards_index** -- raw parsed msgpack dict. Don't change it or base_url,
- **wrong.** (*shards_base_url will be*)
- **url** -- URL of repodata_shards.msgpack.zst

_shards_base_url: str

shards_index

url

_base_url

session

repodata_no_packages

visited: dict[str, *conda._private.shards.typing.ShardDict* | None]

_shard_url_cache: dict[str, str]

property package_names

Return the names of all packages available in this shard collection.

property packages_index

property shards_base_url: str

Return self.url joined with shards_base_url. Note shards_base_url can be a relative or an absolute url.

shard_url(*package: str*) → str

Return shard URL for a given package.

Raise KeyError if package is not in the index.

shard_loaded(*package: str*) → bool

Return True if the given package's shard is in memory.

visit_package(*package: str*) → *conda._private.shards.typing.ShardDict*

Return a shard that is already in memory and mark as visited.

fetch_channels(*url_to_channel: dict[str,* conda.models.channel.Channel*]*, *, *require_shards: bool = True*) → dict[str, *ShardBase*] | None

Parameters

- **url_to_channel** -- not modified, must already be expanded to subdirs.
- **require_shards** -- Return None unless at least one channel provides shards.

Attempt to fetch the sharded index first and then fall back to retrieving a monolithic *repodata.json* file.

Returns

A dict mapping channel URLs to *Shard* or *ShardLike* objects. If *require_shards* is true, return None when no channels have shards. This dict preserves the key order of the input *url_to_channel*.

fetch_shards_index(*sd:* conda.core.subdir_data.SubdirData) → *Shards* | None

Check a SubdirData's URL for shards.

Return shards index bytes from cache or network. Return None if not found; caller should fetch normal repodata.

TODO: If this function fails to retrieve the sharded repodata index file, it will

mark it is as not supporting this feature in cache. This can problematic because sometimes server errors can happen which will lead it to wrongly assuming the channel doesn't support sharding. We need to rethink our logic for determining shard support.

zstd

Internal zstd backend selector.

Import `zstd` for library API (`zstd.ZstdFile`, `zstd.compress`, …). Import `ZstdError` for exception handling. Import `capped_decompress` for bounded one-shot decompression of untrusted data (mimics

`zstandard.decompress(..., max_output_size=...)` and `ZstdDecompressor(max_window_size=..)`).

Functions

capped_decompress(data, max_output_size) → bytes	One-shot decompress of untrusted data with output and window bounds.

capped_decompress(*data: bytes*, *max_output_size: int*) → bytes

One-shot decompress of untrusted data with output and window bounds.

Replaces `zstandard.decompress(..., max_output_size=...)` and `ZstdDecompressor(max_window_size=..)` (window cap was only in subset.py).

_version

version: str

__version__: str

__version_tuple__: tuple[int | str, ...]

version_tuple: tuple[int | str, ...]

commit_id: str | None

__commit_id__: str | None

activate

Conda activate and deactivate logic.

Implementation for all shell interface logic exposed via *conda shell.* [activate|deactivate|reactivate|hook|commands]*. This includes a custom argument parser, an abstract shell class, and special path handling for Windows.

See conda.cli.main.main_sourced for the entry point into this module.

Classes

_Activator	
PosixActivator	
CshActivator	
XonshActivator	
CmdExeActivator	
FishActivator	
PowerShellActivator	
JSONFormatMixin	Returns the necessary values for activation as JSON, so that tools can use them.

Functions

_touch_last_activated(prefix) → None	Record an activation by updating conda-meta/last_activated's mtime.
expand(path) → str	
backslash_to_forwardslash(paths) → str | tuple[str, ...] | None	
_build_activator_cls(shell)	Dynamically construct the activator class.

Attributes

_MSYS2_CYGWIN_PREFIX_RE
BUILTIN_COMMANDS
activator_map
formatter_map

_MSYS2_CYGWIN_PREFIX_RE: re.Pattern[str]

BUILTIN_COMMANDS

_touch_last_activated(*prefix: str*) → None

Record an activation by updating conda-meta/last_activated's mtime.

This is a best-effort activity and thus why we are wrapping the touch command from gateways.disk.update to ignore any OS errors.

class _Activator(*arguments: collections.abc.Iterable[str] | None = None*)

pathsep_join: str

sep: str

path_conversion: collections.abc.Callable[[str | collections.abc.Iterable[str] | None], str | tuple[str, ...] | None]

script_extension: str

tempfile_extension: str | None

Temporary file's extension, None writes to stdout instead.

command_join: str

unset_var_tmpl: str

export_var_tmpl: str

path_var_tmpl: str

set_var_tmpl: str

run_script_tmpl: str

hook_source_path: pathlib.Path | None

inline_hook_source: bool

needs_line_ending_fix: bool

_raw_arguments = None

get_export_unset_vars(*export_metavars=True*, ***kwargs*)

Parameters

- **export_metavars** -- whether to export *conda_exe_vars* meta variables.
- **kwargs** -- environment variables to export. .. if you pass and set any other variable to None, then it emits it to the dict with a value of None.

Returns

A dict of env vars to export ordered the same way as kwargs. And a list of env vars to unset.

_finalize(*commands*, *ext*)

activate()

deactivate()

reactivate()

hook(*auto_activate: bool | None = None*) → str

execute()

template_unset_var(*key: str*) → str

template_export_var(*key: str*, *value: str*) → str

template_path_var(*key: str*, *value: str*) → str

template_set_var(*key: str*, *value: str*) → str

template_run_script(*script: str*) → str

_hook_preamble() → str | None

_hook_postamble() → str | None

_parse_and_set_args() → None

_yield_commands(*cmds_dict*)

build_activate(*env_name_or_prefix: str*) → dict[str, Any]

build_stack(*env_name_or_prefix: str*) → dict[str, Any]

_build_activate_stack(*env_name_or_prefix: str*, *stack: bool*) → dict[str, Any]

build_deactivate()

build_reactivate()

_get_starting_path_list()

_get_path_dirs(*prefix*)

_add_prefix_to_path(*prefix*, *starting_path_dirs=None*)

_remove_prefix_from_path(*prefix*, *starting_path_dirs=None*)

_replace_prefix_in_path(*old_prefix*, *new_prefix*, *starting_path_dirs=None*)

_update_prompt(*set_vars*, *conda_prompt_modifier*)

_default_env(*prefix*)

_prompt_modifier(*prefix*, *conda_default_env*)

_get_activate_scripts(*prefix*)

_get_deactivate_scripts(*prefix*)

_get_environment_env_vars(*prefix*)

_resolve_prefix(*env_name_or_prefix: str*) → str

Hook for shell-specific activation path validation.

expand(*path: str*) → str

backslash_to_forwardslash(*paths: str | collections.abc.Iterable[str] | None*) → str | tuple[str, ...] | None

class PosixActivator(*arguments: collections.abc.Iterable[str] | None = None*)

Bases: *_Activator*

pathsep_join

sep = '/'

path_conversion

script_extension = '.sh'

tempfile_extension = None

Temporary file's extension, None writes to stdout instead.

command_join = Multiline-String

```
"""
"""
```

needs_line_ending_fix = True

unset_var_tmpl = "export %s=''"

export_var_tmpl = "export %s='%s'"

path_var_tmpl = "export %s='%s'"

set_var_tmpl = "%s='%s'"

run_script_tmpl = '. "%s"'

hook_source_path

inline_hook_source = True

_update_prompt(*set_vars*, *conda_prompt_modifier*)

class CshActivator(*arguments: collections.abc.Iterable[str] | None = None*)

Bases: *_Activator*

pathsep_join

sep = '/'

path_conversion

script_extension = '.csh'

tempfile_extension = None

Temporary file's extension, None writes to stdout instead.

command_join = Multiline-String

```
""";
"""
```

needs_line_ending_fix = True

unset_var_tmpl = 'unsetenv %s'

export_var_tmpl = 'setenv %s "%s"'

path_var_tmpl = 'setenv %s "%s"'

set_var_tmpl = "set %s='%s'"

run_script_tmpl = 'source "%s"'

hook_source_path

inline_hook_source = False

_update_prompt(*set_vars*, *conda_prompt_modifier*)

class XonshActivator(*arguments: collections.abc.Iterable[str] | None = None*)

Bases: *_Activator*

pathsep_join

sep = '/'

path_conversion

script_extension = '.sh'

tempfile_extension = None

Temporary file's extension, None writes to stdout instead.

command_join = Multiline-String

```
"""
"""
```

needs_line_ending_fix = False

unset_var_tmpl = Multiline-String

```
"""try:
    del $%s
except KeyError:
    pass"""
```

export_var_tmpl = "$%s = '%s'"

path_var_tmpl = "$%s = '%s'"

set_var_tmpl = "$%s = '%s'"

run_script_tmpl = 'source-bash --suppress-skip-message -n "%s"'

hook_source_path

inline_hook_source = True

template_path_var(*key: str*, *value: str*) → str

class CmdExeActivator(*arguments: collections.abc.Iterable[str] | None = None*)

Bases: *_Activator*

pathsep_join

sep = '\\'

path_conversion

script_extension = '.bat'

tempfile_extension = '.env'

Temporary file's extension, None writes to stdout instead.

command_join = Multiline-String

```
"""
"""
```

needs_line_ending_fix = False

unset_var_tmpl = '%s='

export_var_tmpl = '%s=%s'

path_var_tmpl = '%s=%s'

set_var_tmpl = '%s=%s'

run_script_tmpl = '_CONDA_SCRIPT=%s'

hook_source_path = None

inline_hook_source = None

_update_prompt(*set_vars*, *conda_prompt_modifier*)

_hook_preamble() → None

_resolve_prefix(*env_name_or_prefix: str*) → str

Hook for shell-specific activation path validation.

class FishActivator(*arguments: collections.abc.Iterable[str] | None = None*)

Bases: *_Activator*

pathsep_join

sep = '/'

path_conversion

script_extension = '.fish'

tempfile_extension = None

Temporary file's extension, None writes to stdout instead.

command_join = Multiline-String

```
""";
"""
```

needs_line_ending_fix = True

unset_var_tmpl = 'set -e %s || true'

export_var_tmpl = 'set -gx %s "%s"'

path_var_tmpl = 'set -gx %s "%s"'

set_var_tmpl = 'set -g %s "%s"'

run_script_tmpl = 'source "%s"'

hook_source_path

inline_hook_source = True

class PowerShellActivator(*arguments: collections.abc.Iterable[str] | None = None*)

Bases: *_Activator*

pathsep_join

sep = '/'

path_conversion

script_extension = '.ps1'

tempfile_extension = None

Temporary file's extension, None writes to stdout instead.

command_join = Multiline-String

```
"""
"""
```

needs_line_ending_fix = False

unset_var_tmpl = '$Env:%s = $null'

export_var_tmpl = '$Env:%s = "%s"'

path_var_tmpl = '$Env:%s = "%s"'

set_var_tmpl = '$Env:%s = "%s"'

run_script_tmpl = '. "%s"'

hook_source_path

inline_hook_source = True

_hook_preamble() → str

_hook_postamble() → str

class JSONFormatMixin(*arguments: collections.abc.Iterable[str] | None = None*)

Bases: *_Activator*

Returns the necessary values for activation as JSON, so that tools can use them.

pathsep_join

tempfile_extension = None

Temporary file's extension, None writes to stdout instead.

command_join

_hook_preamble()

_finalize(*commands*, *ext*)

_yield_commands(*cmds_dict*)

activator_map: dict[str, type[_Activator]]

formatter_map

_build_activator_cls(*shell*)

Dynamically construct the activator class.

Detect the base activator and any number of formatters (appended using '+' to the base name). For example, *posix+json* (as in *conda shell.posix+json activate*) would use the *PosixActivator* base class and add the *JSONFormatMixin*.

api

Collection of conda's high-level APIs.

Classes

Solver	**Beta** While in beta, expect both major and minor changes across minor releases.
SubdirData	**Beta** While in beta, expect both major and minor changes across minor releases.
PackageCacheData	**Beta** While in beta, expect both major and minor changes across minor releases.
PrefixData	**Beta** While in beta, expect both major and minor changes across minor releases.

Attributes

DepsModifier	Flags to enable alternate handling of dependencies.
UpdateModifier	Flags to enable alternate handling for updates of existing packages in the environment.

DepsModifier

Flags to enable alternate handling of dependencies.

UpdateModifier

Flags to enable alternate handling for updates of existing packages in the environment.

class Solver(*prefix*, *channels*, *subdirs=()*, *specs_to_add=()*, *specs_to_remove=()*)

Beta While in beta, expect both major and minor changes across minor releases.

A high-level API to conda's solving logic. Three public methods are provided to access a solution in various forms.

- *solve_final_state()*
- *solve_for_diff()*
- *solve_for_transaction()*

Beta

Parameters

- **prefix** (*str*) -- The conda prefix / environment location for which the *Solver* is being instantiated.
- **channels** (Sequence[Channel]) -- A prioritized list of channels to use for the solution.
- **subdirs** (*Sequence[str]*) -- A prioritized list of subdirs to use for the solution.
- **specs_to_add** (set[MatchSpec]) -- The set of package specs to add to the prefix.
- **specs_to_remove** (set[MatchSpec]) -- The set of package specs to remove from the prefix.

_internal

solve_final_state(*update_modifier=NULL*, *deps_modifier=NULL*, *prune=NULL*, *ignore_pinned=NULL*, *force_remove=NULL*)

Beta While in beta, expect both major and minor changes across minor releases.

Gives the final, solved state of the environment.

Parameters

- **deps_modifier** (DepsModifier) -- An optional flag indicating special solver handling for dependencies. The default solver behavior is to be as conservative as possible with dependency updates (in the case the dependency already exists in the environment), while still ensuring all dependencies are satisfied. Options include * NO_DEPS * ONLY_DEPS * UPDATE_DEPS * UPDATE_DEPS_ONLY_DEPS * FREEZE_INSTALLED
- **prune** (*bool*) -- If `True`, the solution will not contain packages that were previously brought into the environment as dependencies but are no longer required as dependencies and are not user-requested.
- **ignore_pinned** (*bool*) -- If `True`, the solution will ignore pinned package configuration for the prefix.
- **force_remove** (*bool*) -- Forces removal of a package without removing packages that depend on it.

Returns

In sorted dependency order from roots to leaves, the package references for the solved state of the environment.

Return type

tuple[PackageRef]

solve_for_diff(*update_modifier=NULL*, *deps_modifier=NULL*, *prune=NULL*, *ignore_pinned=NULL*, *force_remove=NULL*, *force_reinstall=False*)

Beta While in beta, expect both major and minor changes across minor releases.

Gives the package references to remove from an environment, followed by the package references to add to an environment.

Parameters

- **deps_modifier** (DepsModifier) -- See *solve_final_state()*.
- **prune** (*bool*) -- See *solve_final_state()*.
- **ignore_pinned** (*bool*) -- See *solve_final_state()*.
- **force_remove** (*bool*) -- See *solve_final_state()*.
- **force_reinstall** (*bool*) -- For requested specs_to_add that are already satisfied in the environment, instructs the solver to remove the package and spec from the environment, and then add it back--possibly with the exact package instance modified, depending on the spec exactness.

Returns

A two-tuple of PackageRef sequences. The first is the group of packages to remove from the environment, in sorted dependency order from leaves to roots. The second is the group of packages to add to the environment, in sorted dependency order from roots to leaves.

Return type

tuple[PackageRef], tuple[PackageRef]

`solve_for_transaction`(*update_modifier=NULL*, *deps_modifier=NULL*, *prune=NULL*, *ignore_pinned=NULL*, *force_remove=NULL*, *force_reinstall=False*)

Beta While in beta, expect both major and minor changes across minor releases.

Gives an UnlinkLinkTransaction instance that can be used to execute the solution on an environment.

Parameters

- **deps_modifier** (DepsModifier) -- See *solve_final_state()*.
- **prune** (*bool*) -- See *solve_final_state()*.
- **ignore_pinned** (*bool*) -- See *solve_final_state()*.
- **force_remove** (*bool*) -- See *solve_final_state()*.
- **force_reinstall** (*bool*) -- See *solve_for_diff()*.

Return type
UnlinkLinkTransaction

`class SubdirData`(*channel*)

Beta While in beta, expect both major and minor changes across minor releases.

High-level management and usage of repodata.json for subdirs.

Beta While in beta, expect both major and minor changes across minor releases.

Parameters
channel (*str or* Channel) -- The target subdir for the instance. Must either be a url that includes a subdir or a Channel that includes a subdir. e.g.:

- 'https://repo.anaconda.com/pkgs/main/linux-64'
- Channel('https://repo.anaconda.com/pkgs/main/linux-64')
- Channel('conda-forge/osx-64')

`_internal`

`query`(*package_ref_or_match_spec*)

Beta While in beta, expect both major and minor changes across minor releases.

Run a query against this specific instance of repodata.

Parameters
package_ref_or_match_spec (*PackageRef or* MatchSpec *or str*) -- Either an exact PackageRef to match against, or a MatchSpec query object. A str will be turned into a MatchSpec automatically.

Returns
tuple[PackageRecord]

`static query_all`(*package_ref_or_match_spec*, *channels=None*, *subdirs=None*)

Beta While in beta, expect both major and minor changes across minor releases.

Run a query against all repodata instances in channel/subdir matrix.

Parameters

- **package_ref_or_match_spec** (*PackageRef or* MatchSpec *or str*) -- Either an exact PackageRef to match against, or a MatchSpec query object. A str will be turned into a MatchSpec automatically.

- **channels** (*Iterable[Channel or str] or None*) -- An iterable of urls for channels or Channel objects. If None, will fall back to context.channels.
- **subdirs** (*Iterable[str] or None*) -- If None, will fall back to context.subdirs.

Returns
tuple[PackageRecord]

`iter_records()`

Beta While in beta, expect both major and minor changes across minor releases.

Returns

A generator over all records contained in the repodata.json
instance. Warning: this is a generator that is exhausted on first use.

Return type
Iterable[*PackageRecord*]

`reload()`

Beta While in beta, expect both major and minor changes across minor releases.

Update the instance with new information. Backing information (i.e. repodata.json) is lazily downloaded/loaded on first use by the other methods of this class. You should only use this method if you are *sure* you have outdated data.

Returns
SubdirData

`class PackageCacheData(pkgs_dir)`

Beta While in beta, expect both major and minor changes across minor releases.

High-level management and usage of package caches.

Beta While in beta, expect both major and minor changes across minor releases.

Parameters
pkgs_dir (*str*)

`_internal`

`get(package_ref, default=NULL)`

Beta While in beta, expect both major and minor changes across minor releases.

Parameters

- **package_ref** (*PackageRef*) -- A PackageRef instance representing the key for the PackageCacheRecord being sought.
- **default** -- The default value to return if the record does not exist. If not specified and no record exists, KeyError is raised.

Returns
PackageCacheRecord

`query(package_ref_or_match_spec)`

Beta While in beta, expect both major and minor changes across minor releases.

Run a query against this specific package cache instance.

Parameters
package_ref_or_match_spec (*PackageRef or MatchSpec or str*) -- Either an exact PackageRef to match against, or a MatchSpec query object. A str will be turned into a MatchSpec automatically.

Returns
tuple[PackageCacheRecord]

`static query_all`(*package_ref_or_match_spec*, *pkgs_dirs=None*)

Beta While in beta, expect both major and minor changes across minor releases.

Run a query against all package caches.

Parameters

- **package_ref_or_match_spec** (*PackageRef or MatchSpec or str*) -- Either an exact `PackageRef` to match against, or a `MatchSpec` query object. A `str` will be turned into a `MatchSpec` automatically.
- **pkgs_dirs** (*Iterable[str] or None*) -- If None, will fall back to context.pkgs_dirs.

Returns
tuple[PackageCacheRecord]

`iter_records`()

Beta While in beta, expect both major and minor changes across minor releases.

Returns

A generator over all records contained in the package
cache instance. Warning: this is a generator that is exhausted on first use.

Return type
Iterable[*PackageCacheRecord*]

`property is_writable`

Beta While in beta, expect both major and minor changes across minor releases.

Indicates if the package cache location is writable or read-only.

Returns
bool

`static first_writable`(*pkgs_dirs=None*)

Beta While in beta, expect both major and minor changes across minor releases.

Get an instance object for the first writable package cache.

Parameters
pkgs_dirs (*Iterable[str]*) -- If None, will fall back to context.pkgs_dirs.

Returns
An instance for the first writable package cache.

Return type
PackageCacheData

`reload`()

Beta While in beta, expect both major and minor changes across minor releases.

Update the instance with new information. Backing information (i.e. contents of the pkgs_dir) is lazily loaded on first use by the other methods of this class. You should only use this method if you are *sure* you have outdated data.

Returns
PackageCacheData

class PrefixData(*prefix_path*)

Beta While in beta, expect both major and minor changes across minor releases.

High-level management and usage of conda environment prefixes.

Beta While in beta, expect both major and minor changes across minor releases.

Parameters
prefix_path (*str*)

_internal

get(*package_ref*, *default=NULL*)

Beta While in beta, expect both major and minor changes across minor releases.

Parameters

- **package_ref** (*PackageRef*) -- A PackageRef instance representing the key for the PrefixRecord being sought.
- **default** -- The default value to return if the record does not exist. If not specified and no record exists, KeyError is raised.

Returns
PrefixRecord

query(*package_ref_or_match_spec*)

Beta While in beta, expect both major and minor changes across minor releases.

Run a query against this specific prefix instance.

Parameters
package_ref_or_match_spec (*PackageRef or MatchSpec or str*) -- Either an exact PackageRef to match against, or a MatchSpec query object. A str will be turned into a MatchSpec automatically.

Returns
tuple[PrefixRecord]

iter_records()

Beta While in beta, expect both major and minor changes across minor releases.

Returns

A generator over all records contained in the prefix.
Warning: this is a generator that is exhausted on first use.

Return type
Iterable[*PrefixRecord*]

property is_writable

Beta While in beta, expect both major and minor changes across minor releases.

Indicates if the prefix is writable or read-only.

Returns
True if the prefix is writable. False if read-only. None if the prefix does not exist as a conda environment.

Return type
bool or None

reload()

Beta While in beta, expect both major and minor changes across minor releases.

Update the instance with new information. Backing information (i.e. contents of the conda-meta directory) is lazily loaded on first use by the other methods of this class. You should only use this method if you are *sure* you have outdated data.

Returns
PrefixData

auxlib

Auxlib is an auxiliary library to the python standard library.

The aim is to provide core generic features for app development in python. Auxlib fills in some python stdlib gaps much like pytoolz has for functional programming, pyrsistent has for data structures, or boltons has generally.

Major areas addressed include:

- packaging: package versioning, with a clean and less invasive alternative to versioneer
- entity: robust base class for type-enforced data models and transfer objects
- type_coercion: intelligent type coercion utilities
- *Configuration*: a map implementation designed specifically to hold application configuration and context information
- factory: factory pattern implementation
- path: file path utilities especially helpful when working with various python package formats
- logz: logging initialization routines to simplify python logging setup
- crypt: simple, but correct, pycrypto wrapper

[2021-11-09] Our version of auxlib has deviated from the upstream project by a significant amount (especially compared with the other vendored packages). Further, the upstream project has low popularity and is no longer actively maintained. Consequently it was decided to absorb, refactor, and replace auxlib. As a first step of this process we moved conda._vendor.auxlib to conda.auxlib.

collection

Common collection classes.

Classes

AttrDict	Sub-classes dict, and further allows attribute-like access to dictionary items.

Functions

first(seq[, key, default, apply])	Give the first value that satisfies the key test.
last(seq[, key, default, apply])	

class AttrDict(**args*, ***kwargs*)

Bases: dict

Sub-classes dict, and further allows attribute-like access to dictionary items.

Examples

```
>>> d = AttrDict({'a': 1})
>>> d.a, d['a'], d.get('a')
(1, 1, 1)
>>> d.b = 2
>>> d.b, d['b']
(2, 2)
```

Initialize self. See help(type(self)) for accurate signature.

__dict__

first(*seq*, *key=bool*, *default=None*, *apply=lambda x: ...*)

Give the first value that satisfies the key test.

Parameters

- **seq** (*iterable*)
- **key** (*callable*) -- test for each element of iterable
- **default** -- returned when all elements fail test
- **apply** (*callable*) -- applied to element before return, but not to default value

Returns: first element in seq that passes key, mutated with optional apply

Examples

```
>>> first([0, False, None, [], (), 42])
42
>>> first([0, False, None, [], ()]) is None
True
>>> first([0, False, None, [], ()], default='ohai')
'ohai'
>>> import re
>>> m = first(re.match(regex, 'abc') for regex in ['b.*', 'a(.*)'])
>>> m.group(1)
'bc'
```

The optional *key* argument specifies a one-argument predicate function like that used for *filter()*. The *key* argument, if supplied, must be in keyword form. For example: >>> first([1, 1, 3, 4, 5], key=lambda x: x % 2 == 0) 4

last(*seq*, *key=bool*, *default=None*, *apply=lambda x: ...*)

compat

Functions

shlex_split_unicode(to_split[, posix])	
Utf8NamedTemporaryFile([mode, buffering, newline, ...])	

shlex_split_unicode(*to_split*, *posix=True*)

Utf8NamedTemporaryFile(*mode='w+b'*, *buffering=-1*, *newline=None*, *suffix=None*, *prefix=None*, *dir=None*, *delete=True*)

decorators

Classes

classproperty

Functions

memoizemethod(method)	Decorator to cause a method to cache it's results in self for each
clear_memoized_methods(obj, *method_names)	Clear the memoized method or @memoizedproperty results for the given
memoizedproperty(func)	Decorator to cause a method to cache it's results in self for each

memoizemethod(*method*)

Decorator to cause a method to cache it's results in self for each combination of inputs and return the cached result on subsequent calls. Does not support named arguments or arg values that are not hashable.

```
>>> class Foo (object):
...   @memoizemethod
...   def foo(self, x, y=0):
...     print('running method with', x, y)
...     return x + y + 3
...
>>> foo1 = Foo()
>>> foo2 = Foo()
>>> foo1.foo(10)
running method with 10 0
13
>>> foo1.foo(10)
13
>>> foo2.foo(11, y=7)
running method with 11 7
21
>>> foo2.foo(11)
running method with 11 0
14
>>> foo2.foo(11, y=7)
21
>>> class Foo (object):
...   def __init__(self, lower):
...     self.lower = lower
...   @memoizemethod
...   def range_tuple(self, upper):
```

(continues on next page)

(continued from previous page)

```
...     print('running function')
...     return tuple(i for i in range(self.lower, upper))
...   @memoizemethod
...   def range_iter(self, upper):
...     print('running function')
...     return (i for i in range(self.lower, upper))
...
>>> foo = Foo(3)
>>> foo.range_tuple(6)
running function
(3, 4, 5)
>>> foo.range_tuple(7)
running function
(3, 4, 5, 6)
>>> foo.range_tuple(6)
(3, 4, 5)
>>> foo.range_iter(6)
Traceback (most recent call last):
TypeError: Can't memoize a generator or non-hashable object!
```

clear_memoized_methods(*obj*, **method_names*)

Clear the memoized method or @memoizedproperty results for the given method names from the given object.

```
>>> v = [0]
>>> def inc():
...     v[0] += 1
...     return v[0]
...
>>> class Foo(object):
...    @memoizemethod
...    def foo(self):
...        return inc()
...    @memoizedproperty
...    def g(self):
...       return inc()
...
>>> f = Foo()
>>> f.foo(), f.foo()
(1, 1)
>>> clear_memoized_methods(f, 'foo')
>>> (f.foo(), f.foo(), f.g, f.g)
(2, 2, 3, 3)
>>> (f.foo(), f.foo(), f.g, f.g)
(2, 2, 3, 3)
>>> clear_memoized_methods(f, 'g', 'no_problem_if_undefined')
>>> f.g, f.foo(), f.g
(4, 2, 4)
>>> f.foo()
2
```

memoizedproperty(*func*)

Decorator to cause a method to cache it's results in self for each combination of inputs and return the cached result on subsequent calls. Does not support named arguments or arg values that are not hashable.

```
>>> class Foo (object):
...   _x = 1
...   @memoizedproperty
...   def foo(self):
...     self._x += 1
...     print('updating and returning {0}'.format(self._x))
...     return self._x
...
>>> foo1 = Foo()
>>> foo2 = Foo()
>>> foo1.foo
updating and returning 2
2
>>> foo1.foo
2
>>> foo2.foo
updating and returning 2
2
>>> foo1.foo
2
```

class classproperty(*getter=None*, *setter=None*)

__get = None

__set = None

__doc__

__name__

__module__

__get__(*obj*, *type_=None*)

__set__(*obj*, *value*)

setter(*setter*)

entity

This module provides serializable, validatable, type-enforcing domain objects and data transfer objects. It has many of the same motivations as the python Marshmallow package. It is most similar to Schematics.

Tutorial

Chapter 1: Entity and Field Basics

```
>>> class Color(Enum):
...     blue = 0
...     black = 1
...     red = 2
>>> class Car(Entity):
...     weight = NumberField(required=False)
```

(continues on next page)

(continued from previous page)

```
...     wheels = IntField(default=4, validation=lambda x: 3 <= x <= 4)
...     color = EnumField(Color)
```

```
>>> # create a new car object
>>> car = Car(color=Color.blue, weight=4242.46)
>>> car
Car(weight=4242.46, color=0)
```

```
>>> # it has 4 wheels, all by default
>>> car.wheels
4
```

```
>>> # but a car can't have 5 wheels!
>>> #  the `validation=` field is a simple callable that returns a
>>> #  boolean based on validity
>>> car.wheels = 5
Traceback (most recent call last):
ValidationError: Invalid value 5 for wheels
```

```
>>> # we can call .dump() on car, and just get back a standard
>>> #  python dict actually, it's an ordereddict to match attribute
>>> #  declaration order
>>> type(car.dump())
<class '...OrderedDict'>
>>> car.dump()
OrderedDict([('weight', 4242.46), ('wheels', 4), ('color', 0)])
```

```
>>> # and json too (note the order!)
>>> car.json()
'{"weight": 4242.46, "wheels": 4, "color": 0}'
```

```
>>> # green cars aren't allowed
>>> car.color = "green"
Traceback (most recent call last):
ValidationError: 'green' is not a valid Color
```

```
>>> # but black cars are!
>>> car.color = "black"
>>> car.color
<Color.black: 1>
```

```
>>> # car.color really is an enum, promise
>>> type(car.color)
<enum 'Color'>
```

```
>>> # enum assignment can be with any of (and preferentially)
>>> #   (1) an enum literal,
>>> #   (2) a valid enum value, or
>>> #   (3) a valid enum name
```

(continues on next page)

(continued from previous page)

```
>>> car.color = Color.blue; car.color.value
0
>>> car.color = 1; car.color.name
'black'
```

```
>>> # let's do a round-trip marshalling of this thing
>>> same_car = Car.from_json(car.json())  # or equally Car.from_json(json.dumps(car.
↪dump()))
>>> same_car == car
True
```

```
>>> # actually, they're two different instances
>>> same_car is not car
True
```

```
>>> # this works too
>>> cloned_car = Car(**car.dump())
>>> cloned_car == car
True
```

```
>>> # while we're at it, these are all equivalent too
>>> car == Car.from_objects(car)
True
>>> car == Car.from_objects({"weight": 4242.46, "wheels": 4, "color": 1})
True
>>> car == Car.from_json('{"weight": 4242.46, "color": 1}')
True
```

```
>>> # .from_objects() even lets you stack and combine objects
>>> class DumbClass:
...     color = 0
...     wheels = 3
>>> Car.from_objects(DumbClass(), dict(weight=2222, color=1))
Car(weight=2222, wheels=3, color=0)
>>> # and also pass kwargs that override properties pulled
>>> #  off any objects
>>> Car.from_objects(DumbClass(), {'weight': 2222, 'color': 1}, color=2, weight=33)
Car(weight=33, wheels=3, color=2)
```

Chapter 2: Entity and Field Composition

```
>>> # now let's get fancy
>>> # a ComposableField "nests" another valid Entity
>>> # a ListField's first argument is a "generic" type,
>>> #   which can be a valid Entity, any python primitive
>>> #   type, or a list of Entities/types
>>> class Fleet(Entity):
...     boss_car = ComposableField(Car)
...     cars = ListField(Car)
```

```
>>> # here's our fleet of company cars
>>> company_fleet = Fleet(boss_car=Car(color='red'), cars=[car, same_car, cloned_car])
>>> company_fleet.pretty_json()
{
  "boss_car": {
    "wheels": 4
    "color": 2,
  },
  "cars": [
    {
      "weight": 4242.46,
      "wheels": 4
      "color": 1,
    },
    {
      "weight": 4242.46,
      "wheels": 4
      "color": 1,
    },
    {
      "weight": 4242.46,
      "wheels": 4
      "color": 1,
    }
  ]
}
```

```
>>> # the boss' car is red of course (and it's still an Enum)
>>> company_fleet.boss_car.color.name
'red'
```

```
>>> # and there are three cars left for the employees
>>> len(company_fleet.cars)
3
```

Chapter 3: Immutability

```
>>> class ImmutableCar(ImmutableEntity):
...     wheels = IntField(default=4, validation=lambda x: 3 <= x <= 4)
...     color = EnumField(Color)
>>> icar = ImmutableCar.from_objects({'wheels': 3, 'color': 'blue'})
>>> icar
ImmutableCar(wheels=3, color=0)
```

```
>>> icar.wheels = 4
Traceback (most recent call last):
AttributeError: Assignment not allowed. ImmutableCar is immutable.
```

```
>>> class FixedWheelCar(Entity):
...     wheels = IntField(default=4, immutable=True)
...     color = EnumField(Color)
```

(continues on next page)

(continued from previous page)

```
>>> fwcar = FixedWheelCar.from_objects(icar)
>>> fwcar.json()
'{"wheels": 3, "color": 0}'
```

```
>>> # repainting the car is easy
>>> fwcar.color = Color.red
>>> fwcar.color.name
'red'
```

```
>>> # can't really change the number of wheels though
>>> fwcar.wheels = 18
Traceback (most recent call last):
AttributeError: The wheels field is immutable.
```

Chapter X: The del and null Weeds

```
>>> old_date = lambda: isoparse('1982-02-17')
>>> class CarBattery(Entity):
...     # NOTE: default value can be a callable!
...     first_charge = DateField(required=False)  # default=None, nullable=False
...     latest_charge = DateField(default=old_date, nullable=True)  # required=True
...     expiration = DateField(default=old_date, required=False, nullable=False)
```

```
>>> # starting point
>>> battery = CarBattery()
>>> battery
CarBattery()
>>> battery.json()
'{"latest_charge": "1982-02-17T00:00:00", "expiration": "1982-02-17T00:00:00"}'
```

```
>>> # first_charge is not assigned a default value. Once one is assigned, it can be␣
↪deleted,
>>> #   but it can't be made null.
>>> battery.first_charge = isoparse('2016-03-23')
>>> battery
CarBattery(first_charge=datetime.datetime(2016, 3, 23, 0, 0))
>>> battery.first_charge = None
Traceback (most recent call last):
ValidationError: Value for first_charge not given or invalid.
>>> del battery.first_charge
>>> battery
CarBattery()
```

```
>>> # latest_charge can be null, but it can't be deleted. The default value is a callable.
>>> del battery.latest_charge
Traceback (most recent call last):
AttributeError: The latest_charge field is required and cannot be deleted.
>>> battery.latest_charge = None
>>> battery.json()
'{"latest_charge": null, "expiration": "1982-02-17T00:00:00"}'
```

```
>>> # expiration is assigned by default, can't be made null, but can be deleted.
>>> battery.expiration
datetime.datetime(1982, 2, 17, 0, 0)
>>> battery.expiration = None
Traceback (most recent call last):
ValidationError: Value for expiration not given or invalid.
>>> del battery.expiration
>>> battery.json()
'{"latest_charge": null}'
```

Classes

Field	Fields are doing something very similar to boxing and unboxing
BooleanField	Fields are doing something very similar to boxing and unboxing
IntegerField	Fields are doing something very similar to boxing and unboxing
NumberField	Fields are doing something very similar to boxing and unboxing
StringField	Fields are doing something very similar to boxing and unboxing
DateField	Fields are doing something very similar to boxing and unboxing
EnumField	Fields are doing something very similar to boxing and unboxing
ListField	Fields are doing something very similar to boxing and unboxing
MapField	Fields are doing something very similar to boxing and unboxing
ComposableField	Fields are doing something very similar to boxing and unboxing
Entity	
ImmutableEntity	

Attributes

BoolField
IntField

class Field(*default=NULL*, *required=True*, *validation=None*, *in_dump=True*, *default_in_dump=True*, *nullable=False*, *immutable=False*, *aliases=()*)

Fields are doing something very similar to boxing and unboxing of c#/java primitives. __set__ should take a "primitive" or "raw" value and create a "boxed" or "programmatically usable" value of it. While __get__ should return the boxed value, dump in turn should unbox the value into a primitive or raw value.

Parameters

- **types** (*primitive literal or* `type` *or sequence of types*)
- **default** (*any, callable,* `optional`) -- If default is callable, it's guaranteed to return a

valid value at the time of Entity creation.

- **required** (*boolean, optional*)
- **validation** (*callable, optional*)
- **dump** (*boolean, optional*)

_order_helper = 0

_required = True

_validation = None

_in_dump = True

_default_in_dump = True

_nullable = False

_immutable = False

_aliases = ()

property name

set_name(*name*)

__get__(*instance*, *instance_type*)

__set__(*instance*, *val*)

__delete__(*instance*)

box(*instance*, *instance_type*, *val*)

unbox(*instance*, *instance_type*, *val*)

dump(*instance*, *instance_type*, *val*)

validate(*instance*, *val*)

Returns
if val is valid

Return type
True

Raises
ValidationError --

property required

property type

property default

property in_dump

property default_in_dump

property nullable

property is_nullable

property immutable

class BooleanField(*default=NULL*, *required=True*, *validation=None*, *in_dump=True*, *default_in_dump=True*, *nullable=False*, *immutable=False*, *aliases=()*)

Bases: *Field*

Fields are doing something very similar to boxing and unboxing of c#/java primitives. __set__ should take a "primitive" or "raw" value and create a "boxed" or "programmatically usable" value of it. While __get__ should return the boxed value, dump in turn should unbox the value into a primitive or raw value.

Parameters

- **types** (*primitive literal or type or sequence of types*)
- **default** (*any, callable,* optional) -- If default is callable, it's guaranteed to return a valid value at the time of Entity creation.
- **required** (*boolean,* optional)
- **validation** (*callable,* optional)
- **dump** (*boolean,* optional)

_type

box(*instance*, *instance_type*, *val*)

BoolField

class IntegerField(*default=NULL*, *required=True*, *validation=None*, *in_dump=True*, *default_in_dump=True*, *nullable=False*, *immutable=False*, *aliases=()*)

Bases: *Field*

Fields are doing something very similar to boxing and unboxing of c#/java primitives. __set__ should take a "primitive" or "raw" value and create a "boxed" or "programmatically usable" value of it. While __get__ should return the boxed value, dump in turn should unbox the value into a primitive or raw value.

Parameters

- **types** (*primitive literal or type or sequence of types*)
- **default** (*any, callable,* optional) -- If default is callable, it's guaranteed to return a valid value at the time of Entity creation.
- **required** (*boolean,* optional)
- **validation** (*callable,* optional)
- **dump** (*boolean,* optional)

_type

IntField

class NumberField(*default=NULL*, *required=True*, *validation=None*, *in_dump=True*, *default_in_dump=True*, *nullable=False*, *immutable=False*, *aliases=()*)

Bases: *Field*

Fields are doing something very similar to boxing and unboxing of c#/java primitives. __set__ should take a "primitive" or "raw" value and create a "boxed" or "programmatically usable" value of it. While __get__ should return the boxed value, dump in turn should unbox the value into a primitive or raw value.

Parameters

- **types** (*primitive literal or type or sequence of types*)
- **default** (*any, callable,* optional) -- If default is callable, it's guaranteed to return a valid value at the time of Entity creation.
- **required** (*boolean,* optional)
- **validation** (*callable,* optional)
- **dump** (*boolean,* optional)

_type

class StringField(*default=NULL*, *required=True*, *validation=None*, *in_dump=True*, *default_in_dump=True*, *nullable=False*, *immutable=False*, *aliases=()*)

Bases: *Field*

Fields are doing something very similar to boxing and unboxing of c#/java primitives. __set__ should take a "primitive" or "raw" value and create a "boxed" or "programmatically usable" value of it. While __get__ should return the boxed value, dump in turn should unbox the value into a primitive or raw value.

Parameters

- **types** (*primitive literal or type or sequence of types*)
- **default** (*any, callable,* optional) -- If default is callable, it's guaranteed to return a valid value at the time of Entity creation.
- **required** (*boolean,* optional)
- **validation** (*callable,* optional)
- **dump** (*boolean,* optional)

_type

box(*instance*, *instance_type*, *val*)

class DateField(*default=NULL*, *required=True*, *validation=None*, *in_dump=True*, *default_in_dump=True*, *nullable=False*, *immutable=False*, *aliases=()*)

Bases: *Field*

Fields are doing something very similar to boxing and unboxing of c#/java primitives. __set__ should take a "primitive" or "raw" value and create a "boxed" or "programmatically usable" value of it. While __get__ should return the boxed value, dump in turn should unbox the value into a primitive or raw value.

Parameters

- **types** (*primitive literal or type or sequence of types*)
- **default** (*any, callable,* optional) -- If default is callable, it's guaranteed to return a valid value at the time of Entity creation.
- **required** (*boolean,* optional)
- **validation** (*callable,* optional)
- **dump** (*boolean,* optional)

_type

box(*instance*, *instance_type*, *val*)

dump(*instance*, *instance_type*, *val*)

class **EnumField**(*enum_class*, *default=NULL*, *required=True*, *validation=None*, *in_dump=True*, *default_in_dump=True*, *nullable=False*, *immutable=False*, *aliases=()*)

Bases: *Field*

Fields are doing something very similar to boxing and unboxing of c#/java primitives. __set__ should take a "primitive" or "raw" value and create a "boxed" or "programmatically usable" value of it. While __get__ should return the boxed value, dump in turn should unbox the value into a primitive or raw value.

Parameters

- **types** (*primitive literal or type or sequence of types*)
- **default** (*any, callable,* optional) -- If default is callable, it's guaranteed to return a valid value at the time of Entity creation.
- **required** (*boolean,* optional)
- **validation** (*callable,* optional)
- **dump** (*boolean,* optional)

_type

box(*instance*, *instance_type*, *val*)

dump(*instance*, *instance_type*, *val*)

class **ListField**(*element_type*, *default=NULL*, *required=True*, *validation=None*, *in_dump=True*, *default_in_dump=True*, *nullable=False*, *immutable=False*, *aliases=()*)

Bases: *Field*

Fields are doing something very similar to boxing and unboxing of c#/java primitives. __set__ should take a "primitive" or "raw" value and create a "boxed" or "programmatically usable" value of it. While __get__ should return the boxed value, dump in turn should unbox the value into a primitive or raw value.

Parameters

- **types** (*primitive literal or type or sequence of types*)
- **default** (*any, callable,* optional) -- If default is callable, it's guaranteed to return a valid value at the time of Entity creation.
- **required** (*boolean,* optional)
- **validation** (*callable,* optional)
- **dump** (*boolean,* optional)

_type

_element_type

box(*instance*, *instance_type*, *val*)

unbox(*instance*, *instance_type*, *val*)

dump(*instance*, *instance_type*, *val*)

validate(*instance*, *val*)

Returns
if val is valid

Return type
True

Raises
ValidationError --

class MapField(*default=NULL*, *required=True*, *validation=None*, *in_dump=True*, *default_in_dump=True*, *nullable=False*, *immutable=True*, *aliases=()*)

Bases: *Field*

Fields are doing something very similar to boxing and unboxing of c#/java primitives. __set__ should take a "primitive" or "raw" value and create a "boxed" or "programmatically usable" value of it. While __get__ should return the boxed value, dump in turn should unbox the value into a primitive or raw value.

Parameters

- **types** (*primitive literal or type or sequence of types*)
- **default** (*any, callable,* optional) -- If default is callable, it's guaranteed to return a valid value at the time of Entity creation.
- **required** (*boolean,* optional)
- **validation** (*callable,* optional)
- **dump** (*boolean,* optional)

_type

box(*instance*, *instance_type*, *val*)

class ComposableField(*field_class*, *default=NULL*, *required=True*, *validation=None*, *in_dump=True*, *default_in_dump=True*, *nullable=False*, *immutable=False*, *aliases=()*)

Bases: *Field*

Fields are doing something very similar to boxing and unboxing of c#/java primitives. __set__ should take a "primitive" or "raw" value and create a "boxed" or "programmatically usable" value of it. While __get__ should return the boxed value, dump in turn should unbox the value into a primitive or raw value.

Parameters

- **types** (*primitive literal or type or sequence of types*)
- **default** (*any, callable,* optional) -- If default is callable, it's guaranteed to return a valid value at the time of Entity creation.
- **required** (*boolean,* optional)
- **validation** (*callable,* optional)
- **dump** (*boolean,* optional)

_type

box(*instance*, *instance_type*, *val*)

dump(*instance*, *instance_type*, *val*)

class Entity(**kwargs)

__fields__

_lazy_validate = False

classmethod from_objects(*objects, **override_fields)

Construct a new object of type cls from existing objects or dicts.

Allows the creation of new objects of concrete Entity subclasses by combining information from several sources. This can be any combination of objects and dictionaries passed in as positional arguments. When looking for the value of the fields of the Entity subclass, the first object that provides an attribute (or, in the case of a dict an entry) that has the name of the field or one of its aliases will take precedence. Any keyword arguments passed in will override this and take precedence.

Parameters

- **cls** (Entity subclass) -- The class to create, usually determined by call, e.g. PrefixRecord.from_objects(...).
- ***objects** (tuple(object or dict)) -- Any combination of objects and dicts in order of decending precedence.
- ****override_fields** (dict(str, object)) -- Any individual fields overriding possible contents from *objects.

classmethod from_json(json_str)

classmethod load(data_dict)

validate()

__repr__()

classmethod __register__()

json(indent=None, separators=None, **kwargs)

pretty_json(indent=2, separators=(',', ': '), **kwargs)

dump()

classmethod __dump_fields()

__eq__(other)

__hash__()

property _initd

class ImmutableEntity(**kwargs)

Bases: Entity

__setattr__(attribute, value)

__delattr__(item)

exceptions

Classes

AuxlibError	Mixin to identify exceptions associated with the auxlib package.

Functions

Raise(exception)

Raise(*exception*)

class AuxlibError

Mixin to identify exceptions associated with the auxlib package.

exception ValidationError(*key*, *value=None*, *valid_types=None*, *msg=None*)

Bases: *AuxlibError*, TypeError

Mixin to identify exceptions associated with the auxlib package.

Initialize self. See help(type(self)) for accurate signature.

__cause__ = None

exception ThisShouldNeverHappenError

Bases: *AuxlibError*, AttributeError

Mixin to identify exceptions associated with the auxlib package.

Initialize self. See help(type(self)) for accurate signature.

ish

Functions

dals(string)	dedent and left-strip
_get_attr(obj, attr_name[, aliases])	
find_or_none(key, search_maps[, aliases, _map_index])	Return the value of the first key found in the list of search_maps,
find_or_raise(key, search_maps[, aliases, _map_index])	

dals(*string*)

dedent and left-strip

_get_attr(*obj*, *attr_name*, *aliases=()*)

find_or_none(*key*, *search_maps*, *aliases=()*, *_map_index=0*)

Return the value of the first key found in the list of search_maps, otherwise return None.

Examples

```
>>> from .collection import AttrDict
>>> d1 = AttrDict({'a': 1, 'b': 2, 'c': 3, 'e': None})
>>> d2 = AttrDict({'b': 5, 'e': 6, 'f': 7})
>>> find_or_none('c', (d1, d2))
3
>>> find_or_none('f', (d1, d2))
7
>>> find_or_none('b', (d1, d2))
2
>>> print(find_or_none('g', (d1, d2)))
None
>>> find_or_none('e', (d1, d2))
6
```

find_or_raise(*key*, *search_maps*, *aliases=()*, *_map_index=0*)

logz

Functions

set_root_level([level])
attach_stderr([level])
detach_stderr()
initialize_logging([level])
fullname(obj)
request_header_sort_key(item)
response_header_sort_key(item)
stringify(obj[, content_max_len])

Attributes

root_log
NullHandler
DEBUG_FORMATTER
INFO_FORMATTER
request_header_sort_dict
response_header_sort_dict

root_log

NullHandler

DEBUG_FORMATTER

INFO_FORMATTER

set_root_level(*level=INFO*)

attach_stderr(*level=INFO*)

detach_stderr()

initialize_logging(*level=INFO*)

fullname(*obj*)

request_header_sort_dict

request_header_sort_key(*item*)

response_header_sort_dict

response_header_sort_key(*item*)

stringify(*obj*, *content_max_len=0*)

type_coercion

Collection of functions to coerce conversion of types with an intelligent guess.

Functions

numberify(value)	
boolify(value[, nullable, return_string])	Convert a number, string, or sequence type into a pure boolean.
typify(value[, type_hint])	Take a primitive value, usually a string, and try to make a more relevant type out of it.
maybecall(value) → T	

numberify(*value*)

Examples

```
>>> [numberify(x) for x in ('1234', 1234, '0755', 0o0755, False, 0, '0', True, 1, '1
↪')]
  [1234, 1234, 755, 493, 0, 0, 0, 1, 1, 1]
>>> [numberify(x) for x in ('12.34', 12.34, 1.2+3.5j, '1.2+3.5j')]
[12.34, 12.34, (1.2+3.5j), (1.2+3.5j)]
```

boolify(*value*, *nullable=False*, *return_string=False*)

Convert a number, string, or sequence type into a pure boolean.

Parameters
value (*number, string, sequence*) -- pretty much anything

Returns
boolean representation of the given value

Return type
bool

Examples

```
>>> [boolify(x) for x in ('yes', 'no')]
[True, False]
>>> [boolify(x) for x in (0.1, 0+0j, True, '0', '0.0', '0.1', '2')]
[True, False, True, False, False, True, True]
>>> [boolify(x) for x in ("true", "yes", "on", "y")]
[True, True, True, True]
>>> [boolify(x) for x in ("no", "non", "none", "off", "")]
[False, False, False, False, False]
>>> [boolify(x) for x in ([], set(), dict(), tuple())]
[False, False, False, False]
>>> [boolify(x) for x in ([1], set([False]), dict({'a': 1}), tuple([2]))]
[True, True, True, True]
```

typify(*value*, *type_hint=None*)

Take a primitive value, usually a string, and try to make a more relevant type out of it. An optional type_hint will try to coerce the value to that type.

Parameters

- **value** (*Any*) -- Usually a string, not a sequence
- **type_hint** (type *or* *tuple[type]*)

Examples

```
>>> typify('32')
32
>>> typify('32', float)
32.0
>>> typify('32.0')
32.0
>>> typify('32.0.0')
'32.0.0'
>>> [typify(x) for x in ('true', 'yes', 'on')]
[True, True, True]
>>> [typify(x) for x in ('no', 'FALSe', 'off')]
[False, False, False]
>>> [typify(x) for x in ('none', 'None', None)]
[None, None, None]
```

maybecall(*value: Callable[[], T] | T*) → *T*

__version__ = '0.0.43'

__author__ = 'Kale Franz'

__email__ = 'kale@franz.io'

__url__ = 'https://github.com/kalefranz/auxlib'

__license__ = 'ISC'

__copyright__ = '(c) 2015 Kale Franz. All rights reserved.'

__summary__ = 'auxiliary library to the python standard library'

base

Code in `conda.base` is the lowest level of the application stack. It is loaded and executed virtually every time the application is executed. Any code within, and any of its imports, must be highly performant.

Conda modules importable from `conda.base` are

- `conda.base`
- `conda.common`

Modules prohibited from importing `conda.base` are:

- `conda.common`

All other `conda` modules may import from `conda.base`.

constants

This file should hold most string literals and magic numbers used throughout the code base. The exception is if a literal is specifically meant to be private to and isolated within a module. Think of this as a "more static" source of configuration information.

Another important source of "static" configuration is conda/models/enums.py.

Classes

SafetyChecks	Create a collection of name/value pairs.
PathConflict	Create a collection of name/value pairs.
DepsModifier	Flags to enable alternate handling of dependencies.
UpdateModifier	Create a collection of name/value pairs.
ChannelPriorityMeta	Metaclass for Enum
ValueEnum	Subclass of enum that returns the value of the enum as its str representation
ChannelPriority	Subclass of enum that returns the value of the enum as its str representation
SatSolverChoice	Subclass of enum that returns the value of the enum as its str representation
NoticeLevel	Subclass of enum that returns the value of the enum as its str representation

Attributes

PREFIX_PLACEHOLDER
machine_bits
APP_NAME
SEARCH_PATH
SEARCH_PATH
DEFAULT_CHANNEL_ALIAS
CONDA_HOMEPAGE_URL
DEFAULTS_CHANNEL_NAME
PLATFORMS
KNOWN_SUBDIRS
PLATFORM_DIRECTORIES

continues on next page

Table 42 – continued from previous page

WINDOWS_LAUNCHER_STUB_PATH	
RECOGNIZED_URL_SCHEMES	
DEFAULT_CHANNELS_UNIX	
DEFAULT_CHANNELS_WIN	
DEFAULT_CUSTOM_CHANNELS	
DEFAULT_CHANNELS	
ROOT_ENV_NAME	
RESERVED_ENV_NAMES	
UNUSED_ENV_NAME	
ROOT_NO_RM	
DEFAULT_AGGRESSIVE_UPDATE_PACKAGES	
COMPATIBLE_SHELLS	
COMPATIBLE_SHELLS	
MAX_CHANNEL_PRIORITY	
CONDA_PACKAGE_EXTENSION_V1	
CONDA_PACKAGE_EXTENSION_V2	
CONDA_PACKAGE_EXTRACTOR_NAME	
PARTIAL_EXTENSION	Suffix appended to package filenames during incomplete downloads.
CONDA_TARBALL_EXTENSION	
CONDA_TEMP_EXTENSION	
CONDA_TEMP_EXTENSIONS	
CONDA_LOGS_DIR	
UNKNOWN_CHANNEL	
REPODATA_FN	
REPODATA_SHARDS_FN	Index for sharded repodata data of a channel
NOTICES_FN	Default name of the notices file on the server we look for.
NOTICES_CACHE_FN	Name of cache file where read notice IDs are stored.
NOTICES_CACHE_SUBDIR	Determines the subdir for notices cache.
NOTICES_DECORATOR_DISPLAY_INTERVAL	Determines how often (in seconds) notices are displayed while running commands.
NOTICES_DECORATOR_DISPLAY_INTERVAL_NS	Determines how often (in nanoseconds) notices are displayed while running commands.
DRY_RUN_PREFIX	
PREFIX_NAME_DISALLOWED_CHARS	
WINDOWS_PROBLEMATIC_CHARS	
DEFAULT_SOLVER	The name of the default solver, currently "rattler".
CLASSIC_SOLVER	
DEFAULT_JSON_REPORTER_BACKEND	The name of the default json reporter backend.
DEFAULT_CONSOLE_REPORTER_BACKEND	The name of the default console reporter backend.
DEFAULT_CONDA_LIST_FIELDS	The default `conda list` columns.
CONDA_LIST_FIELDS	
PACKAGE_CACHE_MAGIC_FILE	
PREFIX_MAGIC_FILE	
PREFIX_FROZEN_FILE	
PREFIX_CREATION_TIMESTAMP_FILE	
PREFIX_LAST_ACTIVATED_FILE	
PREFIX_STATE_FILE	
PREFIX_PINNED_FILE	
PACKAGE_ENV_VARS_DIR	
CONDA_ENV_VARS_UNSET_VAR	
RESERVED_ENV_VARS	

continues on next page

Table 42 – continued from previous page

NAMESPACES_MAP
NAMESPACE_PACKAGE_NAMES
NAMESPACES
NO_PLUGINS
EXPLICIT_MARKER
OK_MARK
X_MARK
CMD_LINE_SOURCE
ENV_VARS_SOURCE
CONFIGURATION_SOURCES

PREFIX_PLACEHOLDER: Final = '/opt/anaconda1anaconda2anaconda3'

machine_bits: Final

APP_NAME: Final = 'conda'

SEARCH_PATH: tuple[str, ...]

SEARCH_PATH = ('C:/ProgramData/conda/.condarc', 'C:/ProgramData/conda/condarc',
'C:/ProgramData/conda/condarc.d')

DEFAULT_CHANNEL_ALIAS: Final = 'https://conda.anaconda.org'

CONDA_HOMEPAGE_URL: Final = 'https://conda.io'

DEFAULTS_CHANNEL_NAME: Final = 'defaults'

PLATFORMS: Final = ('emscripten-wasm32', 'wasi-wasm32', 'freebsd-64', 'linux-32',
'linux-64', 'linux-aarch64',...

KNOWN_SUBDIRS: Final = ('noarch', 'emscripten-wasm32', 'wasi-wasm32', 'freebsd-64',
'linux-32', 'linux-64',...

PLATFORM_DIRECTORIES = ('noarch', 'emscripten-wasm32', 'wasi-wasm32', 'freebsd-64',
'linux-32', 'linux-64',...

WINDOWS_LAUNCHER_STUB_PATH: Final

RECOGNIZED_URL_SCHEMES: Final = ('http', 'https', 'ftp', 's3', 'file')

DEFAULT_CHANNELS_UNIX: Final = ('https://repo.anaconda.com/pkgs/main',
'https://repo.anaconda.com/pkgs/r')

DEFAULT_CHANNELS_WIN: Final = ('https://repo.anaconda.com/pkgs/main',
'https://repo.anaconda.com/pkgs/r',...

DEFAULT_CUSTOM_CHANNELS: Final

DEFAULT_CHANNELS: Final = ('https://repo.anaconda.com/pkgs/main',
'https://repo.anaconda.com/pkgs/r')

ROOT_ENV_NAME: Final = 'base'

RESERVED_ENV_NAMES: Final

UNUSED_ENV_NAME: Final = 'unused-env-name'

ROOT_NO_RM: Final = ('python', 'pycosat', 'ruamel.yaml', 'conda', 'openssl', 'requests')

DEFAULT_AGGRESSIVE_UPDATE_PACKAGES: Final = ('ca-certificates', 'certifi', 'openssl')

COMPATIBLE_SHELLS: tuple[str, ...]

COMPATIBLE_SHELLS = ('bash', 'cmd.exe', 'fish', 'tcsh', 'xonsh', 'zsh', 'powershell')

MAX_CHANNEL_PRIORITY: Final = 10000

CONDA_PACKAGE_EXTENSION_V1: Final = '.tar.bz2'

CONDA_PACKAGE_EXTENSION_V2: Final = '.conda'

CONDA_PACKAGE_EXTRACTOR_NAME: Final = 'conda-package'

PARTIAL_EXTENSION: Final = '.partial'

Suffix appended to package filenames during incomplete downloads.

CONDA_TARBALL_EXTENSION: Final = '.tar.bz2'

CONDA_TEMP_EXTENSION: Final = '.c~'

CONDA_TEMP_EXTENSIONS: Final

CONDA_LOGS_DIR: Final = '.logs'

UNKNOWN_CHANNEL: Final = '<unknown>'

REPODATA_FN: Final = 'repodata.json'

REPODATA_SHARDS_FN: Final = 'repodata_shards.msgpack.zst'

Index for sharded repodata data of a channel

NOTICES_FN: Final = 'notices.json'

Default name of the notices file on the server we look for.

NOTICES_CACHE_FN: Final = 'notices.cache'

Name of cache file where read notice IDs are stored.

NOTICES_CACHE_SUBDIR: Final = 'notices'

Determines the subdir for notices cache.

NOTICES_DECORATOR_DISPLAY_INTERVAL: Final = 86400

Determines how often (in seconds) notices are displayed while running commands.

NOTICES_DECORATOR_DISPLAY_INTERVAL_NS: Final = 86400000000000

Determines how often (in nanoseconds) notices are displayed while running commands.

DRY_RUN_PREFIX: Final = 'Dry run action:'

PREFIX_NAME_DISALLOWED_CHARS: Final

WINDOWS_PROBLEMATIC_CHARS: Final = ('!', '^', '%', '=', '(', ')')

class SafetyChecks

Bases: enum.Enum

Create a collection of name/value pairs.

Example enumeration:

```
>>> class Color(Enum):
...     RED = 1
...     BLUE = 2
...     GREEN = 3
```

Access them by:

- attribute access:

```
>>> Color.RED
<Color.RED: 1>
```

- value lookup:

```
>>> Color(1)
<Color.RED: 1>
```

- name lookup:

```
>>> Color['RED']
<Color.RED: 1>
```

Enumerations can be iterated over, and know how many members they have:

```
>>> len(Color)
3
```

```
>>> list(Color)
[<Color.RED: 1>, <Color.BLUE: 2>, <Color.GREEN: 3>]
```

Methods can be added to enumerations, and members can have their own attributes -- see the documentation for details.

disabled = 'disabled'

warn = 'warn'

enabled = 'enabled'

__str__() → str

class PathConflict

Bases: enum.Enum

Create a collection of name/value pairs.

Example enumeration:

```
>>> class Color(Enum):
...     RED = 1
...     BLUE = 2
...     GREEN = 3
```

Access them by:

- attribute access:

```
>>> Color.RED
<Color.RED: 1>
```

- value lookup:

```
>>> Color(1)
<Color.RED: 1>
```

- name lookup:

```
>>> Color['RED']
<Color.RED: 1>
```

Enumerations can be iterated over, and know how many members they have:

```
>>> len(Color)
3
```

```
>>> list(Color)
[<Color.RED: 1>, <Color.BLUE: 2>, <Color.GREEN: 3>]
```

Methods can be added to enumerations, and members can have their own attributes -- see the documentation for details.

clobber = 'clobber'

warn = 'warn'

prevent = 'prevent'

__str__() → str

class DepsModifier

Bases: enum.Enum

Flags to enable alternate handling of dependencies.

NOT_SET = 'not_set'

NO_DEPS = 'no_deps'

ONLY_DEPS = 'only_deps'

__str__() → str

class UpdateModifier

Bases: enum.Enum

Create a collection of name/value pairs.

Example enumeration:

```
>>> class Color(Enum):
...     RED = 1
...     BLUE = 2
...     GREEN = 3
```

Access them by:

- attribute access:

```
>>> Color.RED
<Color.RED: 1>
```

- value lookup:

```
>>> Color(1)
<Color.RED: 1>
```

- name lookup:

```
>>> Color['RED']
<Color.RED: 1>
```

Enumerations can be iterated over, and know how many members they have:

```
>>> len(Color)
3
```

```
>>> list(Color)
[<Color.RED: 1>, <Color.BLUE: 2>, <Color.GREEN: 3>]
```

Methods can be added to enumerations, and members can have their own attributes -- see the documentation for details.

SPECS_SATISFIED_SKIP_SOLVE = 'specs_satisfied_skip_solve'

FREEZE_INSTALLED = 'freeze_installed'

UPDATE_DEPS = 'update_deps'

UPDATE_SPECS = 'update_specs'

UPDATE_ALL = 'update_all'

__str__() → str

class ChannelPriorityMeta

Bases: enum.EnumMeta

Metaclass for Enum

__call__(*value*, **args*, ***kwargs*)

Either returns an existing member, or creates a new enum class.

This method is used both when an enum class is given a value to match to an enumeration member (i.e. Color(3)) and for the functional API (i.e. Color = Enum('Color', names='RED GREEN BLUE')).

The value lookup branch is chosen if the enum is final.

When used for the functional API:

value will be the name of the new class.

names should be either a string of white-space/comma delimited names (values will start at *start*), or an iterator/mapping of name, value pairs.

module should be set to the module this class is being created in; if it is not set, an attempt to find that module will be made, but if it fails the class will not be picklable.

qualname should be set to the actual location this class can be found at in its module; by default it is set to the global scope. If this is not correct, unpickling will fail in some circumstances.

type, if set, will be mixed in as the first base class.

class ValueEnum

Bases: enum.Enum

Subclass of enum that returns the value of the enum as its str representation

__str__() → str

class ChannelPriority

Bases: *ValueEnum*

Subclass of enum that returns the value of the enum as its str representation

__name__ = 'ChannelPriority'

STRICT = 'strict'

FLEXIBLE = 'flexible'

DISABLED = 'disabled'

class SatSolverChoice

Bases: *ValueEnum*

Subclass of enum that returns the value of the enum as its str representation

PYCOSAT = 'pycosat'

PYCRYPTOSAT = 'pycryptosat'

PYSAT = 'pysat'

DEFAULT_SOLVER: Final = 'rattler'

The name of the default solver, currently "rattler".

CLASSIC_SOLVER: Final = 'classic'

DEFAULT_JSON_REPORTER_BACKEND: Final = 'json'

The name of the default json reporter backend.

DEFAULT_CONSOLE_REPORTER_BACKEND: Final = 'classic'

The name of the default console reporter backend.

DEFAULT_CONDA_LIST_FIELDS: Final = ('name', 'version', 'build', 'channel_name')

The default `conda list` columns.

CONDA_LIST_FIELDS: Final

class NoticeLevel

Bases: *ValueEnum*

Subclass of enum that returns the value of the enum as its str representation

CRITICAL = 'critical'

```
    WARNING = 'warning'

    INFO = 'info'

PACKAGE_CACHE_MAGIC_FILE: Final[conda.common.path.PathType] = 'urls.txt'

PREFIX_MAGIC_FILE: Final[conda.common.path.PathType]

PREFIX_FROZEN_FILE: Final[conda.common.path.PathType]

PREFIX_CREATION_TIMESTAMP_FILE: Final[conda.common.path.PathType]

PREFIX_LAST_ACTIVATED_FILE: Final[conda.common.path.PathType]

PREFIX_STATE_FILE: Final[conda.common.path.PathType]

PREFIX_PINNED_FILE: Final[conda.common.path.PathType]

PACKAGE_ENV_VARS_DIR: Final[conda.common.path.PathType]

CONDA_ENV_VARS_UNSET_VAR: Final = '***unset***'

RESERVED_ENV_VARS: Final = ('PATH',)

NAMESPACES_MAP: Final

NAMESPACE_PACKAGE_NAMES: Final

NAMESPACES: Final

NO_PLUGINS: Final = False

EXPLICIT_MARKER: Final = '@EXPLICIT'

OK_MARK: Final = ''

X_MARK: Final = ''

CMD_LINE_SOURCE: Final = 'cmd_line'

ENV_VARS_SOURCE: Final = 'envvars'

CONFIGURATION_SOURCES: Final
```

context

Conda's global configuration object.

The context aggregates all configuration files, environment variables, and command line arguments into one global stateful object to be used across all of conda.

Classes

Context
ContextStackObject
ContextStack

Functions

user_data_dir([appname, appauthor, version, roaming]) → conda.common.path.PathType	
mockable_context_envs_dirs(root_writable, root_prefix, ...) → tuple[conda.common.path.PathType, ...]	
channel_alias_validation(value) → str | Literal[True]	
default_python_default() → str	
default_python_validation(value) → str | Literal[True]	
list_fields_validation(value) → str | Literal[True]	
ssl_verify_validation(value) → str | Literal[True]	
reset_context([search_path, argparse_args]) → Context	
fresh_context([env, search_path, argparse_args]) → collections.abc.Iterator[Context]	
stack_context(pushing[, search_path, argparse_args]) → None	
stack_context_default(pushing[, argparse_args]) → None	
replace_context([pushing, search_path, argparse_args]) → None	
replace_context_default([pushing, argparse_args]) → None	
env_name(prefix) → conda.common.path.PathType | str | None	
locate_prefix_by_name(name[, envs_dirs]) → conda.common.path.PathType	Find the location of a prefix given a conda env name. If the location does not exist, an
validate_channels(channels) → tuple[str, ...]	Validate if the given channel URLs are allowed based on the context's allowlist
determine_target_prefix(ctx[, args]) → conda.common.path.PathType	Get the prefix to operate in. The prefix may not yet exist.

Attributes

_platform_map
non_x86_machines
_arch_names
_win_sysconfig_map
user_rc_path
sys_rc_path
context_stack
conda_tests_ctxt_mgmt_def_pol
context

_platform_map

non_x86_machines

_arch_names

_win_sysconfig_map

user_rc_path: conda.common.path.PathType = b'.'

sys_rc_path: conda.common.path.PathType

user_data_dir(*appname: str | None = None*, *appauthor: str | None | Literal[False] = None*, *version: str | None = None*, *roaming: bool = False*) → conda.common.path.PathType

mockable_context_envs_dirs(*root_writable: bool*, *root_prefix: conda.common.path.PathType*, *_envs_dirs: conda.common.path.PathsType*) → tuple[conda.common.path.PathType, ...]

channel_alias_validation(*value: str*) → str | Literal[True]

default_python_default() → str

default_python_validation(*value: str*) → str | Literal[True]

list_fields_validation(*value: collections.abc.Iterable[str]*) → str | Literal[True]

ssl_verify_validation(*value: str*) → str | Literal[True]

class Context(*search_path: conda.common.path.PathsType | None = None*, *argparse_args: argparse.Namespace | None = None*, ***kwargs*)

Bases: *conda.common.configuration.Configuration*

add_pip_as_python_dependency

allow_conda_downgrades

allow_cycles

allow_softlinks

auto_update_conda

auto_activate

_default_activation_env

auto_stack

notify_outdated_conda

clobber

changeps1

env_prompt

environment_specifier

_create_default_packages

register_envs

protect_frozen_envs

default_python

download_only

enable_private_envs

force_32bit

non_admin_enabled

prefix_data_interoperability

_default_threads

_repodata_threads

_fetch_threads

_verify_threads

_execute_threads

_aggressive_update_packages

safety_checks

extra_safety_checks

_signing_metadata_url_base

path_conflict

pinned_packages

exclude_newer

exclude_newer_package

disallowed_packages

rollback_enabled

track_features

use_index_cache

separate_format_cache

_root_prefix

_envs_dirs

_pkgs_dirs

_subdir

_subdirs

_export_platforms

local_repodata_ttl

ssl_verify

client_ssl_cert

client_ssl_cert_key

proxy_servers

remote_connect_timeout_secs

remote_read_timeout_secs

remote_max_retries

remote_backoff_factor

add_anaconda_token

allow_non_channel_urls

_channel_alias

channel_priority

_channels

channel_settings

_custom_channels

_custom_multichannels

_default_channels

_migrated_channel_aliases

migrated_custom_channels

override_channels_enabled

show_channel_urls

use_local

allowlist_channels

denylist_channels

repodata_fns

_use_only_tar_bz2

always_softlink

always_copy

always_yes

_debug

_trace

```
_dev
dry_run
_error_upload_url
force
json
_console
list_fields
offline
quiet
ignore_pinned
_report_errors
shortcuts
number_channel_notices
shortcuts_only
_verbosity
_experimental
preview
no_lock
repodata_use_zst
repodata_use_shards
envvars_force_uppercase
deps_modifier
update_modifier
sat_solver
solver_ignore_timestamps
solver
force_remove
force_reinstall
target_prefix_override
unsatisfiable_hints
unsatisfiable_hints_check_depth
```

bld_path

anaconda_upload

_croot

_conda_build

_override_virtual_packages

no_plugins

post_build_validation() → list[*conda.common.configuration.ValidationError*]

property plugin_manager: *conda.plugins.manager.CondaPluginManager*

This is the preferred way of accessing the `PluginManager` object for this application and is located here to avoid problems with cyclical imports elsewhere in the code.

property plugins: *conda.plugins.config.PluginConfig*

Preferred way of accessing settings introduced by the settings plugin hook

property exclude_newer_policy: *conda.core.exclude_newer.ExcludeNewerPolicy*

Resolved policy for excluding newly indexed package records.

property dev: bool

property experimental: str

property error_upload_url: str

property report_errors: str

property conda_build_local_paths: tuple[conda.common.path.PathType, ...]

property conda_build_local_urls: tuple[str, ...]

property croot: conda.common.path.PathType

This is where source caches and work folders live

property local_build_root: conda.common.path.PathType

property conda_build: dict[str, Any]

property arch_name: str

property platform: str

property default_threads: int | None

property repodata_threads: int | None

property fetch_threads: int | None

If both are not overriden (0), return experimentally-determined value of 5

property verify_threads: int | None

property execute_threads: int | None

property subdir: str

_native_subdir() → str

property subdirs: tuple[str, str]

known_subdirs() → frozenset[str]

property export_platforms: tuple[str, ...]

property bits: int

root_writable() → bool

property envs_dirs: tuple[conda.common.path.PathType, ...]

property pkgs_dirs: tuple[conda.common.path.PathType, ...]

trash_dir() → conda.common.path.PathType

property default_prefix: conda.common.path.PathType

property active_prefix: conda.common.path.PathType

property shlvl: int

property aggressive_update_packages: tuple[*conda.models.match_spec.MatchSpec*, ...]

property target_prefix: conda.common.path.PathType

root_prefix() → conda.common.path.PathType

property conda_prefix: conda.common.path.PathType

property av_data_dir: conda.common.path.PathType

Where critical artifact verification data (e.g., various public keys) can be found.

property signing_metadata_url_base: str | None

Base URL for artifact verification signing metadata (*.root.json, key_mgr.json).

property conda_exe_vars_dict: dict[str, str | None]

The vars can refer to each other if necessary since the dict is ordered. None means unset it.

channel_alias() → *conda.models.channel.Channel*

property migrated_channel_aliases: tuple[*conda.models.channel.Channel*, ...]

property prefix_specified: bool

default_channels() → list[*conda.models.channel.Channel*]

custom_multichannels() → dict[str, tuple[*conda.models.channel.Channel*, ...]]

custom_channels() → dict[str, *conda.models.channel.Channel*]

property channels: tuple[str, ...]

property config_files: tuple[conda.common.path.PathType, ...]

property use_only_tar_bz2: bool

property binstar_upload: bool | None

property trace: bool

Alias for context.verbosity >=4.

property debug: bool

Alias for context.verbosity >=3.

property info: bool

Alias for context.verbosity >=2.

property verbose: bool

Alias for context.verbosity >=1.

property verbosity: int

Verbosity level.

For cleaner and readable code it is preferable to use the following alias properties:
context.trace context.debug context.info context.verbose context.log_level

property log_level: int

Map context.verbosity to logging level.

preview_enabled(*value: str*) → bool

Return True if the given preview feature label is enabled by the user.

property override_virtual_packages: dict[str, str | None]

Remove any dunders in the virtual_package name keys

solver_user_agent() → str

user_agent() → str

_override(*key: str*, *value: Any*) → collections.abc.Iterator[None]

Temporarily override an attribute on the context.

Uses `__dict__` directly because `ParameterLoader` (and other non-data descriptors used by the context) have no `__set__`: a plain `setattr` would shadow the descriptor permanently in `__dict__` and `reset_context()` could not restore it.

Caches derived from context values (`memoizedproperty` results and the registered reset callbacks, such as the `Channel.from_value` cache) are invalidated on entry and again on exit, so that values computed before the override do not mask it and values computed during it do not outlive it.

requests_version() → str

python_implementation_name_version() → tuple[str, str]

platform_system_release() → tuple[str, str]

os_distribution_name_version() → tuple[str, str]

libc_family_version() → tuple[str | None, str | None]

property console: str

property default_activation_env: str

property create_default_packages: tuple[str, ...]

Returns a list of *create_default_packages*, removing any explicit packages.

property default_activation_prefix: pathlib.Path

Return the prefix of the default_activation_env.

If the default_activation_env is an environment name, get the corresponding prefix; otherwise it is already a prefix, so just return it.

Returns
Prefix of the default_activation_env

property environment_context_keys: list[str]

property environment_settings: dict[str, Any]

Returns a dict of environment related settings

category_map: ClassVar[dict[str, tuple[str, ...]]]

get_descriptions() → dict[str, str]

description_map() → dict[str, str]

reset_context(*search_path: conda.common.path.PathsType = SEARCH_PATH*, *argparse_args: argparse.Namespace | None = None*) → *Context*

fresh_context(*env: dict[str, str] | None = None*, *search_path: conda.common.path.PathsType = SEARCH_PATH*, *argparse_args: argparse.Namespace | None = None*, ***kwargs*) → collections.abc.Iterator[*Context*]

class ContextStackObject(*search_path: conda.common.path.PathsType = SEARCH_PATH*, *argparse_args: argparse.Namespace | None = None*)

set_value(*search_path: conda.common.path.PathsType = SEARCH_PATH*, *argparse_args: argparse.Namespace | None = None*) → None

apply()

class ContextStack

_stack

_stack_idx = 0

_last_search_path = None

_last_argparse_args = None

push(*search_path: conda.common.path.PathsType*, *argparse_args: argparse.Namespace | None*) → None

apply()

pop()

replace(*search_path: conda.common.path.PathsType*, *argparse_args: argparse.Namespace | None*) → None

context_stack

stack_context(*pushing: bool*, *search_path: conda.common.path.PathsType = SEARCH_PATH*, *argparse_args: argparse.Namespace | None = None*) → None

stack_context_default(*pushing: bool*, *argparse_args: argparse.Namespace | None = None*) → None

replace_context(*pushing: bool | None = None*, *search_path: collections.abc.Iterable[str] = SEARCH_PATH*, *argparse_args: argparse.Namespace | None = None*) → None

replace_context_default(*pushing: bool | None = None*, *argparse_args: argparse.Namespace | None = None*) → None

conda_tests_ctxt_mgmt_def_pol

env_name(*prefix: conda.common.path.PathType*) → conda.common.path.PathType | str | None

locate_prefix_by_name(*name: str*, *envs_dirs: conda.common.path.PathsType | None = None*) → conda.common.path.PathType

Find the location of a prefix given a conda env name. If the location does not exist, an error is raised.

validate_channels(*channels: collections.abc.Iterator[str]*) → tuple[str, ...]

Validate if the given channel URLs are allowed based on the context's allowlist and denylist configurations.

Parameters
channels -- A list of channels (either URLs or names) to validate.

Raises

- ***ChannelNotAllowed*** -- If any URL is not in the allowlist.
- ***ChannelDenied*** -- If any URL is in the denylist.

determine_target_prefix(*ctx: Context*, *args: argparse.Namespace | None = None*) → conda.common.path.PathType

Get the prefix to operate in. The prefix may not yet exist.

Parameters

- **ctx** -- the context of conda
- **args** -- the argparse args from the command line

Returns
Path to the target environment prefix.

Raises
CondaEnvironmentNotFoundError -- If the prefix is invalid.

context

cli

actions

Collection of custom argparse actions.

Classes

NullCountAction	Information about how to convert command line strings to Python objects.
ExtendConstAction	A derivative of _AppendConstAction and Python 3.8's _ExtendAction

class NullCountAction(*option_strings*, *dest*, *default=None*, *required=False*, *help=None*, *deprecated=False*)

Bases: argparse._CountAction

Information about how to convert command line strings to Python objects.

Action objects are used by an ArgumentParser to represent the information needed to parse a single argument from one or more strings from the command line. The keyword arguments to the Action constructor are also all attributes of Action instances.

Keyword Arguments

- **which** (*- option_strings -- A list of command-line option strings*) -- should be associated with this action.
- **object** (*- dest -- The name of the attribute to hold the created*)
- **be** (*- nargs -- The number of command-line arguments that should*) -- consumed. By default, one argument will be consumed and a single value will be produced. Other values include:
  - N (an integer) consumes N arguments (and produces a list)
  - '?' consumes zero or one arguments
  - '*' consumes zero or more arguments (and produces a list)
  - '+' consumes one or more arguments (and produces a list)

  Note that the difference between the default and nargs=1 is that with the default, a single value will be produced, while with nargs=1, a list containing a single value will be produced.
- **the** (*- metavar -- The name to be used for the option's argument with*) -- option uses an action that takes no values.
- **specified.** (*- default -- The value to be produced if the option is not*)
- **and** (*- type -- A callable that accepts a single string argument,*) -- returns the converted value. The standard Python types str, int, float, and complex are useful examples of such callables. If None, str is used.
- **None,** (*- choices -- A container of values that should be allowed. If not*) -- after a command-line argument has been converted to the appropriate type, an exception will be raised if it is not a member of this collection.
- **the** -- command line. This is only meaningful for optional command-line arguments.
- **argument.** (*- help -- The help string describing the*)
- **the** -- help string. If None, the 'dest' value will be used as the name.

static _ensure_value(*namespace*, *name*, *value*)

__call__(*parser*, *namespace*, *values*, *option_string=None*)

class ExtendConstAction(*option_strings*, *dest*, *const*, *default=None*, *type=None*, *choices=None*, *required=False*, *help=None*, *metavar=None*)

Bases: argparse.Action

A derivative of _AppendConstAction and Python 3.8's _ExtendAction

__call__(*parser*, *namespace*, *values*, *option_string=None*)

common

Common utilities for conda command line tools.

Functions

is_active_prefix(prefix) → bool	Determines whether the args we pass in are pointing to the active prefix.
specs_from_args(args) → list[str]	
strip_comment(line) → str	
spec_from_line(line) → str	
specs_from_url(url) → list[str]	
names_in_specs(names, specs)	
disp_features(features)	
stdout_json(d)	
stdout_json_success([success])	
check_non_admin()	
validate_subdir_config()	Validates that the configured subdir is ok. A subdir that is different from
print_activate(env_name_or_prefix)	
get_name_prefix_from_env_file(file) → tuple[str | None, str | None]	Get name and prefix from an environment spec file.
validate_environment_files_consistency(files) → None	Validates that all the provided environment files are of the same format type.
validate_file_exists(filename)	Validate the existence of an environment file.

Attributes

spec_pat

is_active_prefix(*prefix: str*) → bool

Determines whether the args we pass in are pointing to the active prefix. Can be used a validation step to make sure operations are not being performed on the active prefix.

specs_from_args(*args: collections.abc.Iterable[str]*) → list[str]

spec_pat

strip_comment(*line: str*) → str

spec_from_line(*line: str*) → str

specs_from_url(*url: str*) → list[str]

names_in_specs(*names*, *specs*)

disp_features(*features*)

stdout_json(*d*)

stdout_json_success(*success=True*, ***kwargs*)

check_non_admin()

validate_subdir_config()

Validates that the configured subdir is ok. A subdir that is different from the native system is only allowed if it comes from the global configuration, or from an environment variable.

Raises

OperationNotAllowed -- Active environment is not allowed to request non-native platform packages

print_activate(*env_name_or_prefix*)

get_name_prefix_from_env_file(*file: str*) → tuple[str | None, str | None]

Get name and prefix from an environment spec file.

Intentionally swallows all errors and logs them.

validate_environment_files_consistency(*files: list[str]*) → None

Validates that all the provided environment files are of the same format type.

This function checks if all provided environment files are of the same format type using the conda plugin system's environment specifiers. It prevents mixing different environment file formats (e.g., YAML, explicit package lists, requirements.txt).

Raises

EnvironmentFileTypeMismatchError -- When files with different formats are found

validate_file_exists(*filename: str*)

Validate the existence of an environment file.

This function checks if the given `filename` exists as an environment file. If the *filename* has a URL scheme supported by `CONDA_SESSION_SCHEMES`, it assumes the file is accessible and returns without further validation. Otherwise, it expands the given path and verifies its existence. If the file does not exist, an `EnvironmentFileNotFound` exception is raised.

Parameters

filename -- The path or URL of the environment file to validate.

Raises

EnvironmentFileNotFound -- If the file does not exist and is not a valid URL.

conda_argparse

Conda command line interface parsers.

Classes

ArgumentParser	Object for parsing command line strings into Python objects.
_GreedySubParsersAction	A custom subparser action to conditionally act as a greedy consumer.
_LazyParserMap	A dict that triggers lazy parser loading on key access.
_LazySubParsersAction	Extends _GreedySubParsersAction with lazy loading of subcommand parsers.

Functions

_user_rc_path() → str	
_sys_rc_path() → str	
_escaped_user_rc_path() → str	
_escaped_sys_rc_path() → str	
_configure_builtin_subcommand(sub_parsers, name) → ArgumentParser	
generate_pre_parser(*[, with_plugins]) → ArgumentParser	
generate_parser(**kwargs) → ArgumentParser	
do_call(args, parser)	Serves as the primary entry point for commands referred to in this file and for
find_builtin_commands(parser) → tuple[str, ...]	
_exec(executable_args, env_vars)	
_exec_win(executable_args, env_vars)	
_exec_unix(executable_args, env_vars)	
configure_parser_plugins(sub_parsers) → None	For each of the provided plugin-based subcommands, we'll create

Attributes

BUILTIN_SUBCOMMANDS	Built-in subcommand module, help, and parser metadata.
BUILTIN_COMMANDS	Names (and aliases) of built-in commands; these cannot be overridden by plugin subcommands.
_PLUGIN_FREE_BUILTIN_COMMANDS	Built-in commands that intentionally skip plugin discovery.

_user_rc_path() → str

_sys_rc_path() → str

_escaped_user_rc_path() → str

_escaped_sys_rc_path() → str

BUILTIN_SUBCOMMANDS: dict[str, dict]

Built-in subcommand module, help, and parser metadata.

BUILTIN_COMMANDS

Names (and aliases) of built-in commands; these cannot be overridden by plugin subcommands.

_PLUGIN_FREE_BUILTIN_COMMANDS

Built-in commands that intentionally skip plugin discovery.

_configure_builtin_subcommand(*sub_parsers*, *name: str*) → *ArgumentParser*

generate_pre_parser(*, *with_plugins: bool = True*, ***kwargs*) → *ArgumentParser*

generate_parser(***kwargs*) → *ArgumentParser*

do_call(*args: argparse.Namespace*, *parser:* ArgumentParser)

Serves as the primary entry point for commands referred to in this file and for all registered plugin subcommands.

find_builtin_commands(*parser: argparse.ArgumentParser*) → tuple[str, ...]

class ArgumentParser(**args*, *add_help=True*, ***kwargs*)

Bases: argparse.ArgumentParser

Object for parsing command line strings into Python objects.

Keyword Arguments

- **(default** (- *color - Allow color output in help messages*) -- os.path.basename(sys.argv[0]))
- **(default** -- auto-generated from arguments)
- **does** (- *description -- A description of what the program*)
- **descriptions** (- *epilog -- Text following the argument*)
- **one** (- *parents -- Parsers whose arguments should be copied into this*)
- **messages** (- *formatter_class -- HelpFormatter class for printing help*)
- **arguments** (- *argument_default -- The default value for all*)
- **containing** (- *fromfile_prefix_chars -- Characters that prefix files*) -- additional arguments
- **arguments**
- **conflicts** (- *conflict_handler -- String indicating how to handle*)
- **option** (- *add_help -- Add a -h/-help*)
- **unambiguously** (- *allow_abbrev -- Allow long options to be abbreviated*)
- **with** (- *exit_on_error -- Determines whether or not ArgumentParser exits*) -- error info when an error occurs
- **choices** (- *suggest_on_error - Enables suggestions for mistyped argument*) -- and subparser names (default: True)
- **(default** -- False)

_check_value(*action*, *value*)

parse_args(**args*, *override_args=None*, ***kwargs*)

class _GreedySubParsersAction(*option_strings*, *prog*, *parser_class*, *dest=SUPPRESS*, *required=False*, *help=None*, *metavar=None*)

Bases: argparse._SubParsersAction

A custom subparser action to conditionally act as a greedy consumer.

This is a workaround since argparse.REMAINDER does not work as expected, see https://github.com/python/cpython/issues/61252.

__call__(*parser*, *namespace*, *values*, *option_string=None*)

_get_subactions()

Sort actions for subcommands to appear alphabetically in help blurb.

class _LazyParserMap(*action*)

Bases: dict

A dict that triggers lazy parser loading on key access.

Used as the _name_parser_map for _LazySubParsersAction so that any consumer reading a specific subcommand parser (e.g. sphinx-argparse navigating a :path: directive) gets the fully-configured parser rather than the lightweight stub registered at generate_parser() time.

Initialize self. See help(type(self)) for accurate signature.

_action

property _ready

True once the owning action is initialized and not mid-build.

_resolve(*key*)

Ensure *key* is fully loaded (builtin or plugin).

_resolve_all()

Load every lazy subcommand and all plugins.

_known_names()

Builtin names known without running any configure_parser or plugin discovery.

The union of already-materialized entries in the underlying dict (stubs and real parsers), pending lazy-loader names, and aliases. Plugin-provided subcommand names are not included here; callers that need them should use `items()` / `values()` / `_resolve_all()`.

__getitem__(*key*)

Return self[key].

get(*key*, *default=None*)

Return the value for key if key is in the dictionary, else default.

__contains__(*key*)

True if the dictionary has the specified key, else False.

items()

Return a set-like object providing a view on the dict's items.

values()

Return an object providing a view on the dict's values.

keys()

Return a set-like object providing a view on the dict's keys.

__iter__()

Implement iter(self).

__len__()

Return len(self).

class _LazySubParsersAction(**args*, ***kwargs*)

Bases: *_GreedySubParsersAction*

Extends _GreedySubParsersAction with lazy loading of subcommand parsers.

Registers lightweight stub parsers for all built-in subcommands at generate_parser() time (name + help text only). The real configure_parser is called on demand when a subcommand is actually invoked.

Plugin subcommand discovery is deferred until a non-builtin command is requested or --help is displayed.

_name_parser_map

choices

_lazy_loaders: dict[str, tuple[str, dict]]

_lazy_aliases: dict[str, str]

_loading_name: str | None = None

_plugins_loaded = False

_building = False

add_lazy_subcommand(*name*, *module_path*, *help_text*, ***kwargs*)

Register a lazy-loaded subcommand with a stub parser.

add_parser(*name*, ***kwargs*)

During lazy loading, return the existing stub instead of creating a new parser.

_ensure_loaded(*name*)

Load a lazy subcommand's full parser configuration.

_ensure_plugins_loaded()

Discover and register plugin subcommands on demand.

__call__(*parser*, *namespace*, *values*, *option_string=None*)

_get_subactions()

Ensure plugins are discovered before listing subcommands in help.

_exec(*executable_args*, *env_vars*)

_exec_win(*executable_args*, *env_vars*)

_exec_unix(*executable_args*, *env_vars*)

configure_parser_plugins(*sub_parsers*) → None

For each of the provided plugin-based subcommands, we'll create a new subparser for an improved help printout and calling the *configure_parser()* with the newly created subcommand specific argument parser.

condarc

Configuration file manipulation utilities for conda.

This module provides classes and functions for working with conda configuration files (.condarc), including reading, writing, and validating configuration keys.

Classes

_MissingSentinel	Sentinel value to indicate a missing configuration key.
ParameterTypeGroups	Groups configuration parameters by their parameter type.
ConfigurationFile	Represents and manipulates a conda configuration (.condarc) file.

Functions

_register_enum_representers() → None	Register YAML representers for conda enum types.
validate_provided_parameters(parameters, ...) → None	Validate that provided parameters exist in the configuration context.

Attributes

MISSING

_register_enum_representers() → None

Register YAML representers for conda enum types.

This function registers custom YAML representers for all conda enum types to ensure they are serialized as strings rather than complex objects.

This is called once at module load time to ensure representers are available before any YAML serialization occurs.

class _MissingSentinel

Sentinel value to indicate a missing configuration key.

This is used by ConfigurationFile.get_key() to distinguish between a key that doesn't exist and a key that exists but has a None value.

__repr__()

__bool__()

__eq__(*other*)

__hash__()

MISSING

class ParameterTypeGroups(*context:* conda.common.configuration.Configuration)

Groups configuration parameters by their parameter type.

Organizes configuration parameters from a Configuration instance into sequence and map parameters, handling both regular and plugin parameters separately.

This is primarily used by ConfigurationFile to efficiently determine which operations are valid for different configuration keys.

Initialize ParameterTypeGroups by grouping parameters by type.

Parameters
context -- Configuration instance containing configuration parameters.

_grouped_parameter

property sequence_parameters: list[str]

List of sequence parameter names.

property plugin_sequence_parameters: list[str]

List of plugin sequence parameter names.

property map_parameters: list[str]

List of map parameter names.

property plugin_map_parameters: list[str]

List of plugin map parameter names.

validate_provided_parameters(*parameters: collections.abc.Sequence[str]*, *plugin_parameters: collections.abc.Sequence[str]*, *context:* conda.common.configuration.Configuration) → None

Validate that provided parameters exist in the configuration context.

Compares the provided parameters with the available parameters in the context and raises an error if any are invalid.

Parameters

- **parameters** -- Regular parameter names to validate.
- **plugin_parameters** -- Plugin parameter names to validate.
- **context** -- Configuration instance containing available parameters.

Raises

ArgumentError -- If any provided parameters are not valid.

class ConfigurationFile(*path: str | os.PathLike[str] | pathlib.Path | None = None*, *context:* conda.common.configuration.Configuration | *None = None*, *content: dict[str, Any] | None = None*, *warning_handler: collections.abc.Callable[[str], None] | None = None*)

Represents and manipulates a conda configuration (.condarc) file.

Provides methods to read, write, and modify configuration files while validating keys and maintaining proper structure.

Can be used as a context manager for atomic edits:

with ConfigurationFile.from_user_condarc() as config:
config.set_key("channels", ["conda-forge"]) config.add("channels", "defaults")

File is automatically written when exiting the context

_path = None

_content = None

_read_state: tuple[pathlib.Path, pathlib.Path, str | None] | None = None

_context = None

_context_params: *ParameterTypeGroups* | None = None

warning_handler

classmethod from_user_condarc(*context:* conda.common.configuration.Configuration | *None = None*) → *ConfigurationFile*

Create a ConfigurationFile instance for the default user .condarc file.

Parameters

context -- Optional Configuration instance. If None, uses the global context.

Returns

ConfigurationFile instance configured for the user's .condarc file.

classmethod from_system_condarc(*context:* conda.common.configuration.Configuration | *None* = *None*) → *ConfigurationFile*

Create a ConfigurationFile instance for the system .condarc file.

Parameters
context -- Optional Configuration instance. If None, uses the global context.

Returns
ConfigurationFile instance configured for the system .condarc file.

classmethod from_env_condarc(*prefix: str | os.PathLike[str] | pathlib.Path | None = None*, *context:* conda.common.configuration.Configuration | *None* = *None*) → *ConfigurationFile*

Create a ConfigurationFile instance for an environment-specific .condarc file.

Environment-specific .condarc files are located at *{prefix}/.condarc* and allow per-environment configuration overrides.

Parameters

- **prefix** -- Path to the conda environment. If None, uses $CONDA_PREFIX or sys.prefix.
- **context** -- Optional Configuration instance. If None, uses the global context.

Returns
ConfigurationFile instance configured for the environment's .condarc file.

__enter__() → *ConfigurationFile*

Enter the context manager.

__exit__(*exc_type*, *exc_val*, *exc_tb*) → None

Exit the context manager and write changes if no exception occurred.

property path: **str | os.PathLike[str] | pathlib.Path**

Get the path to the configuration file.

Returns
Path to the configuration file.

Raises
AttributeError -- If path has not been set.

property context: *conda.common.configuration.Configuration*

Get the context instance.

If no context was provided during initialization, falls back to the global conda context singleton. This lazy import is necessary to avoid circular dependencies at module load time.

Returns
Configuration instance.

property context_params: *ParameterTypeGroups*

property content: **dict[str, Any]**

Get the configuration content, reading from file if needed.

Returns
Dictionary containing configuration content.

read(*path: str | os.PathLike[str] | pathlib.Path | None = None*) → dict[str, Any]

Read configuration content from file.

Parameters
path -- Optional path to read from. If None, uses the instance path.

Returns
Dictionary containing configuration content.

write(*path: str | os.PathLike[str] | pathlib.Path | None = None*) → None

Write configuration content to file.

Parameters
path -- Optional path to write to. If None, uses the instance path.

Raises
CondaError -- If the file cannot be written.

key_exists(*key: str*) → bool

Check if a configuration key is valid.

Parameters
key -- Configuration key to check.

Returns
True if the key is valid, False otherwise.

add(*key: str*, *item: Any*, *prepend: bool = False*) → None

Add an item to a sequence configuration parameter.

Parameters

- **key** -- Configuration key name (may contain dots for nested keys).
- **item** -- Item to add to the sequence.
- **prepend** -- If True, add to the beginning; if False, add to the end.

Raises

- ***CondaValueError*** -- If the key is not a known sequence parameter.
- ***CouldntParseError*** -- If the key should be a list but isn't.

get_key(*key: str*) → tuple[str, Any | *_MissingSentinel*]

Get a configuration value by key.

Parameters
key -- Configuration key name (may contain dots for nested keys).

Returns
Tuple of (key, value) or (key, MISSING) if key doesn't exist.

set_key(*key: str*, *item: Any*) → None

Set a configuration value for a primitive or map parameter.

Parameters

- **key** -- Configuration key name (may contain dots for nested keys).
- **item** -- Value to set.

Raises
CondaKeyError -- If the key is unknown or invalid.

remove_item(*key: str*, *item: Any*) → None

Remove an item from a sequence configuration parameter.

Parameters

- **key** -- Configuration key name.
- **item** -- Item to remove from the sequence.

Raises

CondaKeyError -- If the key is unknown, undefined, or the item is not present.

clear_key(*key: str*) → None

Clear all values from a sequence configuration parameter.

The key remains explicitly configured with an empty list. This differs from `remove_key()`, which removes the key entirely and may allow a value from another configuration source or a default to take effect.

Parameters

key -- Sequence configuration key name.

Raises

- *CondaKeyError* -- If the key is unknown or is not a sequence parameter.
- *CouldntParseError* -- If the configured value is not a sequence.

remove_key(*key: str*) → None

Remove a configuration key entirely.

Parameters

key -- Configuration key name (may contain dots for nested keys).

Raises

CondaKeyError -- If the key is undefined in the config.

find_commands

Utilities for finding executables and *conda-** commands.

Functions

find_executable(executable[, include_others])	
find_commands([include_others])	

find_executable(*executable*, *include_others=True*)

find_commands(*include_others=True*)

formats

Helpers for rendering CLI help text.

Functions

get_available_environment_formats(formats, *[, indent]) → str	Render epilog text for grouped environment format plugins.
get_multiplatform_lockfile(formats) → str | None	Return first `default_filenames` from the first multiplatform lockfile.

get_available_environment_formats(*formats: collections.abc.Mapping[conda.plugins.types.EnvironmentFormat, collections.abc.Sequence[conda.plugins.types.CondaPluginWithEnvironmentFormat]], *, indent: int = 0*) → str

Render epilog text for grouped environment format plugins.

Each plugin is one bullet line: `NAME`, `NAME (aliases: ...)`, `NAME: FILENAME, ...`, or combined aliases plus filenames.

get_multiplatform_lockfile(*formats: collections.abc.Mapping[conda.plugins.types.EnvironmentFormat, collections.abc.Sequence[conda.plugins.types.CondaEnvironmentExporter]]*) → str | None

Return first `default_filenames` from the first multiplatform lockfile.

helpers

Collection of helper functions to standardize reused CLI arguments.

Classes

LazyChoicesAction	Information about how to convert command line strings to Python objects.
_ValidatePackages	Used to validate match specs of packages

Functions

add_parser_create_install_update(p[, prefix_required, ...])	
add_parser_pscheck(p) → None	
add_parser_show_channel_urls(p) → None	
add_parser_help(p) → None	So we can use consistent capitalization and periods in the help. You must
add_parser_prefix(p[, prefix_required]) → argparse._MutuallyExclusiveGroup	
add_parser_prefix_to_group(m) → None	
add_parser_json(p) → argparse._ArgumentGroup	
add_output_and_prompt_options(p) → argparse._ArgumentGroup	
add_parser_frozen_env(p)	
add_parser_channels(p) → argparse._ArgumentGroup	
add_parser_solver_mode(p, *[, include_only_deps]) → argparse._ArgumentGroup	

continues on next page

Table 58 – continued from previous page

add_parser_update_modifiers(solver_mode_option *[, ...])	
add_parser_prune(p) → None	
add_parser_solver(p) → None	Add a command-line flag for alternative solver backends.
add_parser_networking(p) → argparse._ArgumentGroup	
add_parser_package_install_options(p) → argparse._ArgumentGroup	
add_parser_known(p) → None	
add_parser_default_packages(p) → None	
add_parser_platform(parser, *[, help, known_subdirs_only]) → None	
add_parser_verbose(parser) → None	
add_parser_environment_specifier(p) → None	
comma_separated_stripped(value) → list[str]	Custom type for argparse to handle comma-separated strings with stripping

class LazyChoicesAction(*option_strings*, *dest*, *choices_func*, ***kwargs*)

Bases: argparse.Action

Information about how to convert command line strings to Python objects.

Action objects are used by an ArgumentParser to represent the information needed to parse a single argument from one or more strings from the command line. The keyword arguments to the Action constructor are also all attributes of Action instances.

Keyword Arguments

- **which** (- *option_strings -- A list of command-line option strings*) -- should be associated with this action.
- **object** (- *dest -- The name of the attribute to hold the created*)
- **be** (- *nargs -- The number of command-line arguments that should*) -- consumed. By default, one argument will be consumed and a single value will be produced. Other values include:
  - N (an integer) consumes N arguments (and produces a list)
  - '?' consumes zero or one arguments
  - '*' consumes zero or more arguments (and produces a list)
  - '+' consumes one or more arguments (and produces a list)

  Note that the difference between the default and nargs=1 is that with the default, a single value will be produced, while with nargs=1, a list containing a single value will be produced.
- **the** (- *metavar -- The name to be used for the option's argument with*) -- option uses an action that takes no values.
- **specified.** (- *default -- The value to be produced if the option is not*)
- **and** (- *type -- A callable that accepts a single string argument,*) -- returns the converted value. The standard Python types str, int, float, and complex are useful examples of such callables. If None, str is used.

- **None,** (*- choices -- A container of values that should be allowed. If not*) -- after a command-line argument has been converted to the appropriate type, an exception will be raised if it is not a member of this collection.
- **the** -- command line. This is only meaningful for optional command-line arguments.
- **argument.** (*- help -- The help string describing the*)
- **the** -- help string. If None, the 'dest' value will be used as the name.

choices_func

_cached_choices = None

property choices

Dynamically evaluate choices for help generation and validation.

__call__(*parser*, *namespace*, *values*, *option_string=None*)

class _ValidatePackages(*option_strings*, *dest*, *nargs=None*, *const=None*, *default=None*, *type=None*, *choices=None*, *required=False*, *help=None*, *metavar=None*, *deprecated=False*)

Bases: argparse._StoreAction

Used to validate match specs of packages

static _validate_no_denylist_channels(*packages_specs*)

Ensure the packages do not contain denylist_channels

__call__(*parser*, *namespace*, *values*, *option_string=None*)

add_parser_create_install_update(*p*, *prefix_required=False*, ***, *include_only_deps=True*)

add_parser_pscheck(*p: argparse.ArgumentParser*) → None

add_parser_show_channel_urls(*p: argparse.ArgumentParser | argparse._ArgumentGroup*) → None

add_parser_help(*p: argparse.ArgumentParser*) → None

So we can use consistent capitalization and periods in the help. You must use the add_help=False argument to ArgumentParser or add_parser to use this. Add this first to be consistent with the default argparse output.

add_parser_prefix(*p: argparse.ArgumentParser*, *prefix_required: bool = False*) → argparse._MutuallyExclusiveGroup

add_parser_prefix_to_group(*m: argparse._MutuallyExclusiveGroup*) → None

add_parser_json(*p: argparse.ArgumentParser*) → argparse._ArgumentGroup

add_output_and_prompt_options(*p: argparse.ArgumentParser*) → argparse._ArgumentGroup

add_parser_frozen_env(*p: argparse.ArgumentParser*)

add_parser_channels(*p: argparse.ArgumentParser*) → argparse._ArgumentGroup

add_parser_solver_mode(*p: argparse.ArgumentParser*, ***, *include_only_deps: bool = True*) → argparse._ArgumentGroup

add_parser_update_modifiers(*solver_mode_options: argparse.ArgumentParser*, ***, *include_update_all: bool = True*)

add_parser_prune(*p: argparse.ArgumentParser*) → None

add_parser_solver(*p: argparse.ArgumentParser*) → None

Add a command-line flag for alternative solver backends.

See `context.solver` for more info.

add_parser_networking(*p: argparse.ArgumentParser*) → argparse._ArgumentGroup

add_parser_package_install_options(*p: argparse.ArgumentParser*) → argparse._ArgumentGroup

add_parser_known(*p: argparse.ArgumentParser*) → None

add_parser_default_packages(*p: argparse.ArgumentParser*) → None

add_parser_platform(*parser: argparse.ArgumentParser | argparse._ArgumentGroup*, *, *help: str | None = None*, *known_subdirs_only: bool = True*) → None

add_parser_verbose(*parser: argparse.ArgumentParser | argparse._ArgumentGroup*) → None

add_parser_environment_specifier(*p: argparse.ArgumentParser*) → None

comma_separated_stripped(*value: str*) → list[str]

Custom type for argparse to handle comma-separated strings with stripping

install

Conda package installation logic.

Core logic for *conda [create|install|update|remove]* commands.

See conda.cli.main_create, conda.cli.main_install, conda.cli.main_update, and conda.cli.main_remove for the entry points into this module.

Classes

TryRepodata	
Repodatas	

Functions

reinstall_packages(args, specs, **kwargs) → int	Reinstall packages using conda install.
clone(src_arg, dst_prefix[, json, quiet, index_args])	
get_revision(arg[, json])	
get_index_args(args) → dict[str, Any]	Returns a dict of args required for fetching an index
validate_install_command(prefix[, command])	Executes a set of validations that are required before any installation
ensure_update_specs_exist(prefix, specs)	Checks that each spec that is requested as an update exists in the prefix
install(args, parser[, command])	Logic for *conda install*, *conda update*, and *conda create*.
install_clone(args, parser)	Executes an install of a new conda environment by cloning.
install_revision(args, parser)	Install a previous version of a conda environment
revert_actions(prefix[, revision, index])	

continues on next page

Table 60 – continued from previous page

handle_txn(unlink_link_transaction, prefix, args, newenv) → dict | None	Handles executing an unlink_link_transaction, and reporting changes. If *defer_json_success*

reinstall_packages(*args*, *specs: list[str]*, ***kwargs*) → int

Reinstall packages using conda install.

Helper for health fixes that need to reinstall packages.

Parameters

- **args** -- Parsed arguments namespace
- **specs** -- Package specs to reinstall
- **kwargs** -- Override default install options (e.g., force_reinstall=True)

clone(*src_arg*, *dst_prefix*, *json=False*, *quiet=False*, *index_args=None*)

get_revision(*arg*, *json=False*)

get_index_args(*args*) → dict[str, Any]

Returns a dict of args required for fetching an index

Parameters
args -- The args provided by the cli

Returns
dict of index args

class TryRepodata(*notify_success*, *repodata*, *last_repodata*, *index_args*, *allowed_errors*)

notify_success

repodata

last_repodata

index_args

allowed_errors

__enter__()

__exit__(*exc_type*, *exc_value*, *traceback*)

class Repodatas(*repodata_fns*, *index_args*, *allows_errors=()*)

repodata_fns

index_args

success = False

allowed_errors

__iter__()

succeed()

validate_install_command(*prefix: str*, *command: str = 'install'*)

Executes a set of validations that are required before any installation command is executed. This includes:

- ensure the configuration is valid
- ensuring the user in not an admin
- ensure the user is not forcing 32bit installs in the root prefix

Parameters

- **prefix** -- The prefix where the environment will be created
- **command** -- Type of operation being performed

Raises

error if the configuration for the install is bad --

ensure_update_specs_exist(*prefix: str*, *specs: list[str]*)

Checks that each spec that is requested as an update exists in the prefix

Parameters

- **prefix** -- The target install prefix
- **specs** -- List of specs to be updated

Raises

- ***CondaError*** -- if there is an invalid spec provided
- ***PackageNotInstalledError*** -- if the requested specs to install don't exist in the prefix

install(*args*, *parser*, *command='install'*)

Logic for *conda install*, *conda update*, and *conda create*.

install_clone(*args*, *parser*)

Executes an install of a new conda environment by cloning.

install_revision(*args*, *parser*)

Install a previous version of a conda environment

revert_actions(*prefix*, *revision=-1*, *index:* conda.core.index.Index | *None = None*)

handle_txn(*unlink_link_transaction*, *prefix*, *args*, *newenv*, *remove_op=False*, *defer_json_success=False*, *confirm=True*) → dict | None

Handles executing an unlink_link_transaction, and reporting changes. If *defer_json_success* argument is provided, will return a dict of actions that have been taken as part of the transaction.

Parameters

- **unlink_link_transaction** -- transaction to execute, output from the solver
- **prefix** -- prefix to execute the transaction on
- **args** -- cli args
- **newenv** -- boolean noting that the environment is being created for the first time
- **remove_op** -- defaults to false, boolean noting that the user is requesting to remove a package from the environment
- **defer_json_success** -- when true, will return the set of actions executed in dict form

- **confirm** -- defaults to true; when false, skip the transaction summary and the confirmation prompt

main

Entry point for all conda subcommands.

Functions

init_loggers()	
main_subshell(*args[, post_parse_hook])	Entrypoint for the "subshell" invocation of CLI interface. E.g. *conda create*.
main_sourced(shell, *args, **kwargs)	Entrypoint for the "sourced" invocation of CLI interface. E.g. *conda activate*.
main(*args, **kwargs)	

init_loggers()

main_subshell(**args*, *post_parse_hook=None*, ***kwargs*)

Entrypoint for the "subshell" invocation of CLI interface. E.g. *conda create*.

main_sourced(*shell*, **args*, ***kwargs*)

Entrypoint for the "sourced" invocation of CLI interface. E.g. *conda activate*.

main(**args*, ***kwargs*)

main_clean

CLI implementation for *conda clean*.

Removes cached package tarballs, index files, package metadata, temporary files, and log files.

Functions

configure_parser(sub_parsers, **kwargs) → argparse.ArgumentParser	
_get_size(*parts, warnings) → int	
_get_pkgs_dirs(pkg_sizes) → dict[str, tuple[str, ...]]	
_get_total_size(pkg_sizes) → int	
_rm_rf(*parts, quiet, verbose) → None	
find_tarballs() → dict[str, Any]	
find_pkgs() → dict[str, Any]	
rm_pkgs(pkgs_dirs, warnings, total_size, pkg_sizes, *, ...) → None	
find_index_cache() → list[str]	
find_pkgs_dirs() → list[str]	
find_tempfiles(paths) → list[str]	
find_logfiles() → list[str]	
rm_items(items, *, quiet, verbose, dry_run, name) → None	
_execute(args, parser)	
execute(args, parser) → int	

configure_parser(*sub_parsers: argparse._SubParsersAction*, ***kwargs*) → argparse.ArgumentParser

_get_size(**parts: str*, *warnings: list[str] | None*) → int

_get_pkgs_dirs(*pkg_sizes: dict[str, dict[str, int]]*) → dict[str, tuple[str, ...]]

_get_total_size(*pkg_sizes: dict[str, dict[str, int]]*) → int

_rm_rf(**parts: str*, *quiet: bool*, *verbose: bool*) → None

find_tarballs() → dict[str, Any]

find_pkgs() → dict[str, Any]

rm_pkgs(*pkgs_dirs: dict[str, tuple[str]]*, *warnings: list[str]*, *total_size: int*, *pkg_sizes: dict[str, dict[str, int]]*, *, *quiet: bool*, *verbose: bool*, *dry_run: bool*, *name: str*) → None

find_index_cache() → list[str]

find_pkgs_dirs() → list[str]

find_tempfiles(*paths: collections.abc.Iterable[str]*) → list[str]

find_logfiles() → list[str]

rm_items(*items: list[str]*, *, *quiet: bool*, *verbose: bool*, *dry_run: bool*, *name: str*) → None

_execute(*args*, *parser*)

execute(*args: argparse.Namespace*, *parser: argparse.ArgumentParser*) → int

main_commands

Mock CLI implementation for *conda activate*.

A mock implementation of the activate shell command for better UX.

Functions

configure_parser(sub_parsers, **kwargs) → arg-parse.ArgumentParser	
execute(args, parser) → int	

configure_parser(*sub_parsers: argparse._SubParsersAction*, ***kwargs*) → argparse.ArgumentParser

execute(*args: argparse.Namespace*, *parser: argparse.ArgumentParser*) → int

main_compare

CLI implementation for *conda compare*.

Compare the packages in an environment with the packages listed in an environment file.

Functions

configure_parser(sub_parsers, **kwargs) → argparse.ArgumentParser	
compare_packages(active_pkgs, specification_pkgs) → tuple[int, list[str]]	
execute(args, parser) → int	

configure_parser(*sub_parsers: argparse._SubParsersAction*, ***kwargs*) → argparse.ArgumentParser

compare_packages(*active_pkgs*, *specification_pkgs*) → tuple[int, list[str]]

execute(*args: argparse.Namespace*, *parser: argparse.ArgumentParser*) → int

main_config

CLI implementation for *conda config*.

Allows for programmatically interacting with conda's configuration files (e.g., *~/.condarc*).

Functions

configure_parser(sub_parsers, **kwargs) → argparse.ArgumentParser	
execute(args, parser) → int	
format_dict(d)	
parameter_description_builder(name[, context, plugins])	
describe_all_parameters([context, plugins]) → str	Return a string with the descriptions of all available configuration
print_config_item(key, value)	
set_keys(*args, path) → None	Set multiple configuration keys in a file.
execute_config(args, parser) → int | None	Execute the conda config command based on provided arguments.
_key_exists(key, warnings[, context]) → bool	Deprecated. Use ConfigurationFile.key_exists() instead.
_get_key(key, warnings, config[, context]) → tuple[str, Any]	Deprecated. Use ConfigurationFile.get_key() instead.
_set_key(key, item, config) → None	Deprecated. Use ConfigurationFile.set_key() instead.
_remove_item(key, item, config) → None	Deprecated. Use ConfigurationFile.remove_item() instead.
_remove_key(key, config) → None	Deprecated. Use ConfigurationFile.remove_key() instead.
_read_rc(path) → dict	Deprecated. Use ConfigurationFile.read() instead.
_write_rc(path, config) → None	Deprecated. Use ConfigurationFile.write() instead.
_validate_provided_parameters(parameters, ...) → None	Deprecated. Use validate_provided_parameters() instead.

configure_parser(*sub_parsers: argparse._SubParsersAction*, ***kwargs*) → argparse.ArgumentParser

execute(*args: argparse.Namespace*, *parser: argparse.ArgumentParser*) → int

format_dict(*d*)

parameter_description_builder(*name*, *context=None*, *plugins=False*)

describe_all_parameters(*context=None*, *plugins=False*) → str

Return a string with the descriptions of all available configuration

When `context` has no parameters, this function returns ""

print_config_item(*key*, *value*)

set_keys(**args: tuple[str, Any]*, *path: str | os.PathLike[str] | pathlib.Path*) → None

Set multiple configuration keys in a file.

Parameters

- **args** -- Variable number of (key, value) tuples to set.
- **path** -- Path to the configuration file.

execute_config(*args: argparse.Namespace*, *parser: argparse.ArgumentParser*) → int | None

Execute the conda config command based on provided arguments.

Handles various config subcommands including show, show-sources, describe, validate, write-default, and modification operations (add, set, remove, etc.).

Parameters

- **args** -- Parsed command line arguments.
- **parser** -- Argument parser instance.

_key_exists(*key: str*, *warnings: list[str]*, *context=None*) → bool

Deprecated. Use ConfigurationFile.key_exists() instead.

_get_key(*key: str*, *warnings: list[str]*, *config: dict[str, Any]*, *context=None*) → tuple[str, Any]

Deprecated. Use ConfigurationFile.get_key() instead.

_set_key(*key: str*, *item: Any*, *config: dict*) → None

Deprecated. Use ConfigurationFile.set_key() instead.

_remove_item(*key: str*, *item: Any*, *config: dict*) → None

Deprecated. Use ConfigurationFile.remove_item() instead.

_remove_key(*key: str*, *config: dict*) → None

Deprecated. Use ConfigurationFile.remove_key() instead.

_read_rc(*path: str | os.PathLike[str] | pathlib.Path*) → dict

Deprecated. Use ConfigurationFile.read() instead.

_write_rc(*path: str | os.PathLike[str] | pathlib.Path*, *config: dict*) → None

Deprecated. Use ConfigurationFile.write() instead.

_validate_provided_parameters(*parameters: collections.abc.Sequence[str]*, *plugin_parameters: collections.abc.Sequence[str]*, *context*) → None

Deprecated. Use validate_provided_parameters() instead.

main_create

CLI implementation for *conda create*.

Creates new conda environments with the specified packages.

Functions

epilog() → str	Build `conda create` epilog (examples and plugin-driven format list).
configure_parser(sub_parsers, **kwargs) → argparse.ArgumentParser	
execute(args, parser) → int	

epilog() → str

Build `conda create` epilog (examples and plugin-driven format list).

configure_parser(*sub_parsers: argparse._SubParsersAction*, ***kwargs*) → argparse.ArgumentParser

execute(*args: argparse.Namespace*, *parser: argparse.ArgumentParser*) → int

main_env

Entry point for all conda-env subcommands.

Functions

configure_parser(sub_parsers, **kwargs) → argparse.ArgumentParser	
execute(args, parser) → int	

configure_parser(*sub_parsers: argparse._SubParsersAction*, ***kwargs*) → argparse.ArgumentParser

execute(*args: argparse.Namespace*, *parser: argparse.ArgumentParser*) → int

main_env_config

CLI implementation for *conda-env config*.

Allows for programmatically interacting with conda-env's configuration files (e.g., *~/.condarc*).

Functions

configure_parser(sub_parsers, **kwargs) → argparse.ArgumentParser	
execute(args, parser) → int	

configure_parser(*sub_parsers: argparse._SubParsersAction*, ***kwargs*) → argparse.ArgumentParser

execute(*args: argparse.Namespace*, *parser: argparse.ArgumentParser*) → int

main_env_create

CLI implementation for *conda-env create*.

Creates new conda environments with the specified packages.

Functions

epilog() → str	Build `conda env create` epilog (examples and plugin-driven format list).
configure_parser(sub_parsers, **kwargs) → argparse.ArgumentParser	

epilog() → str

Build `conda env create` epilog (examples and plugin-driven format list).

configure_parser(*sub_parsers: argparse._SubParsersAction*, ***kwargs*) → argparse.ArgumentParser

main_env_list

CLI implementation for *conda-env list*, now aliased to *conda info --envs*.

Lists available conda environments.

Functions

configure_parser(sub_parsers, **kwargs) → argparse.ArgumentParser	

configure_parser(*sub_parsers: argparse._SubParsersAction*, ***kwargs*) → argparse.ArgumentParser

main_env_remove

CLI implementation for *conda-env remove*.

Removes the specified conda environment.

Functions

configure_parser(sub_parsers, **kwargs) → argparse.ArgumentParser	

configure_parser(*sub_parsers: argparse._SubParsersAction*, ***kwargs*) → argparse.ArgumentParser

main_env_update

CLI implementation for *conda-env update*.

Updates the conda environments with the specified packages.

Functions

configure_parser(sub_parsers, **kwargs) → argparse.ArgumentParser	

continues on next page

Table 72 – continued from previous page

execute(args, parser) → int	

configure_parser(*sub_parsers: argparse._SubParsersAction*, ***kwargs*) → argparse.ArgumentParser

execute(*args: argparse.Namespace*, *parser: argparse.ArgumentParser*) → int

main_env_vars

CLI implementation for *conda-env config vars*.

Allows for configuring conda-env's vars.

Functions

configure_parser(sub_parsers, **kwargs) → argparse.ArgumentParser	
execute_list(args, parser) → int	
execute_set(args, parser) → int	
execute_unset(args, parser) → int	

configure_parser(*sub_parsers: argparse._SubParsersAction*, ***kwargs*) → argparse.ArgumentParser

execute_list(*args: argparse.Namespace*, *parser: argparse.ArgumentParser*) → int

execute_set(*args: argparse.Namespace*, *parser: argparse.ArgumentParser*) → int

execute_unset(*args: argparse.Namespace*, *parser: argparse.ArgumentParser*) → int

main_export

CLI implementation for *conda export*.

Dumps specified environment package specifications to the screen.

Functions

epilog() → str	Build `conda export` epilog (examples and plugin-driven format list).
configure_parser(sub_parsers, **kwargs) → argparse.ArgumentParser	
execute(args, parser) → int	

exception CondaExportWarning

Bases: Warning

Base class for warning categories.

Initialize self. See help(type(self)) for accurate signature.

epilog() → str

Build `conda export` epilog (examples and plugin-driven format list).

configure_parser(*sub_parsers: argparse._SubParsersAction*, ***kwargs*) → argparse.ArgumentParser

execute(*args: argparse.Namespace*, *parser: argparse.ArgumentParser*) → int

main_info

CLI implementation for *conda info*.

Display information about current conda installation.

Classes

InfoRenderer	Provides a `render` method for rendering `InfoComponents`

Functions

configure_parser(sub_parsers, **kwargs) → argparse.ArgumentParser	
get_user_site() → list[str]	Method used to populate `site_dirs` in `conda info`.
get_installer_info(prefix) → dict[str, str] | None	Read Constructor installer metadata from a prefix.
dump_record(prec) → dict[str, Any]	Returns a dictionary of key/value pairs from `prec`. Keys included in `IGNORE_FIELDS` are not returned.
pretty_package(prec) → None	Pretty prints contents of a `PackageRecord`
get_info_dict() → dict[str, Any]	Returns a dictionary of contextual information.
get_env_vars_str(info_dict) → str	Returns a printable string representing environment variables from the dictionary returned by `get_info_dict`.
get_main_info_display(info_dict) → dict[str, str]	Returns the data that can be used to display information for conda info
get_main_info_str(info_dict) → str	Returns a printable string of the contents of `info_dict`.
iter_info_components(args, context) → collections.abc.Iterable[InfoComponents]	Determine which components to display.
execute(args, parser) → int	Implements `conda info` command.

Attributes

INSTALLER_INFO_FIELDS	
IGNORE_FIELDS	
SKIP_FIELDS	
InfoComponents	Possible components for the info command to render.

INSTALLER_INFO_FIELDS = ('name', 'version', 'platform', 'type')

configure_parser(*sub_parsers: argparse._SubParsersAction*, ***kwargs*) → argparse.ArgumentParser

get_user_site() → list[str]

Method used to populate `site_dirs` in `conda info`.

Returns

List of directories.

get_installer_info(*prefix: str*) → dict[str, str] | None

Read Constructor installer metadata from a prefix.

IGNORE_FIELDS: set[str]

SKIP_FIELDS: set[str]

dump_record(*prec:* conda.models.records.PackageRecord) → dict[str, Any]

Returns a dictionary of key/value pairs from `prec`. Keys included in `IGNORE_FIELDS` are not returned.

Parameters
prec -- A `PackageRecord` object.

Returns
A dictionary of elements dumped from `prec`

pretty_package(*prec:* conda.models.records.PackageRecord) → None

Pretty prints contents of a `PackageRecord`

Parameters
prec -- A `PackageRecord`

get_info_dict() → dict[str, Any]

Returns a dictionary of contextual information.

Returns
Dictionary of conda information to be sent to stdout.

get_env_vars_str(*info_dict: dict[str, Any]*) → str

Returns a printable string representing environment variables from the dictionary returned by `get_info_dict`.

Parameters
info_dict -- The returned dictionary from `get_info_dict()`.

Returns
String to print.

get_main_info_display(*info_dict: dict[str, Any]*) → dict[str, str]

Returns the data that can be used to display information for conda info

get_main_info_str(*info_dict: dict[str, Any]*) → str

Returns a printable string of the contents of `info_dict`.

Parameters
info_dict -- The output of `get_info_dict()`.

Returns
String to print.

InfoComponents

Possible components for the info command to render.

class InfoRenderer(*context:* conda.base.context.Context, *show_size: bool = False*)

Provides a `render` method for rendering `InfoComponents`

_context

_show_size = False

_component_style_map

property _info_dict

property _info_dict_envs: list[str]

property _info_dict_envs_details: dict[str, dict[str, str | bool | None | int]]

render(*components: collections.abc.Iterable[InfoComponents]*)

Iterates through the registered components, obtains the data to render via a `_<component>_component` method and then renders it.

_base_component() → str | dict

_channels_component() → str | dict

_detail_component() → dict[str, str]

_envs_component()

_system_component() → str

_json_all_component() → dict[str, Any]

iter_info_components(*args: argparse.Namespace*, *context:* conda.base.context.Context) → collections.abc.Iterable[*InfoComponents*]

Determine which components to display.

Parameters

- **args** -- The parsed command line arguments.
- **context** -- The conda context.

Returns

An iterable of components to display.

execute(*args: argparse.Namespace*, *parser: argparse.ArgumentParser*) → int

Implements `conda info` command.

- `conda info`
- `conda info --base`
- `conda info <package_spec> ...`
- `conda info --unsafe-channels`
- `conda info --envs`
- `conda info --system`

main_init

CLI implementation for *conda init*.

Prepares the user's profile for running conda, and sets up the conda shell interface.

Functions

configure_parser(sub_parsers, **kwargs) → argparse.ArgumentParser	
execute(args, parser) → int	

configure_parser(*sub_parsers: argparse._SubParsersAction*, ***kwargs*) → argparse.ArgumentParser

execute(*args: argparse.Namespace*, *parser: argparse.ArgumentParser*) → int

main_install

CLI implementation for *conda install*.

Installs the specified packages into an existing environment.

Functions

configure_parser(sub_parsers, **kwargs) → argparse.ArgumentParser	
execute(args, parser) → int	

configure_parser(*sub_parsers: argparse._SubParsersAction*, ***kwargs*) → argparse.ArgumentParser

execute(*args: argparse.Namespace*, *parser: argparse.ArgumentParser*) → int

main_list

CLI implementation for *conda list*.

Lists all packages installed into an environment.

Functions

configure_parser(sub_parsers, **kwargs) → argparse.ArgumentParser	
print_export_header(subdir)	
get_packages(installed, regex)	
list_packages(prefix[, regex, format, reverse, ...]) → tuple[int, list[str] | list[dict[str, Any]]]	
print_packages(prefix[, regex, format, reverse, ...]) → int	
print_explicit(prefix[, add_md5, remove_auth, add_sha256])	
execute(args, parser) → int	

configure_parser(*sub_parsers: argparse._SubParsersAction*, ***kwargs*) → argparse.ArgumentParser

print_export_header(*subdir*)

get_packages(*installed*, *regex*)

list_packages(*prefix*, *regex=None*, *format='human'*, *reverse=False*, *show_channel_urls=None*, *reload_records=True*, *fields=None*, *regex_is_full_name=False*, *show_size=False*) → tuple[int, list[str] | list[dict[str, Any]]]

print_packages(*prefix*, *regex=None*, *format='human'*, *reverse=False*, *piplist=False*, *json=False*, *show_channel_urls=None*, *fields=None*, *regex_is_full_name=False*, *show_size=False*) → int

print_explicit(*prefix*, *add_md5=False*, *remove_auth=True*, *add_sha256=False*)

execute(*args: argparse.Namespace*, *parser: argparse.ArgumentParser*) → int

main_mock_activate

Mock CLI implementation for *conda activate*.

A mock implementation of the activate shell command for better UX.

Functions

configure_parser(sub_parsers, **kwargs) → argparse.ArgumentParser	
execute(args, parser) → int	

configure_parser(*sub_parsers: argparse._SubParsersAction*, ***kwargs*) → argparse.ArgumentParser

execute(*args: argparse.Namespace*, *parser: argparse.ArgumentParser*) → int

main_mock_deactivate

Mock CLI implementation for *conda deactivate*.

A mock implementation of the deactivate shell command for better UX.

Functions

configure_parser(sub_parsers, **kwargs) → argparse.ArgumentParser	
execute(args, parser) → int	

configure_parser(*sub_parsers: argparse._SubParsersAction*, ***kwargs*) → argparse.ArgumentParser

execute(*args: argparse.Namespace*, *parser: argparse.ArgumentParser*) → int

main_notices

CLI implementation for *conda notices*.

Manually retrieves channel notifications, caches them and displays them.

Functions

configure_parser(sub_parsers, **kwargs) → argparse.ArgumentParser	
execute(args, parser) → int	Command that retrieves channel notifications, caches them and displays them.

configure_parser(*sub_parsers: argparse._SubParsersAction*, ***kwargs*) → argparse.ArgumentParser

execute(*args: argparse.Namespace*, *parser: argparse.ArgumentParser*) → int

Command that retrieves channel notifications, caches them and displays them.

main_package

CLI implementation for *conda package*.

Provides some low-level tools for creating conda packages.

Functions

configure_parser(sub_parsers, **kwargs) → argparse.ArgumentParser	
remove(prefix, files)	Remove files for a given prefix.
execute(args, parser) → int	
get_installed_version(prefix, name)	
create_info(name, version, build_number, requires_py)	
fix_shebang(tmp_dir, path)	
_add_info_dir(t, tmp_dir, files, has_prefix, info)	
create_conda_pkg(prefix, files, info, tar_path[, ...])	Create a conda package and return a list of warnings.
make_tarbz2(prefix[, name, version, build_number, files])	
which_package(path)	Return the package containing the path.
which_prefix(path)	Return the prefix for the provided path.

Attributes

shebang_pat

configure_parser(*sub_parsers: argparse._SubParsersAction*, ***kwargs*) → argparse.ArgumentParser

remove(*prefix*, *files*)

Remove files for a given prefix.

execute(*args: argparse.Namespace*, *parser: argparse.ArgumentParser*) → int

get_installed_version(*prefix*, *name*)

create_info(*name*, *version*, *build_number*, *requires_py*)

shebang_pat

fix_shebang(*tmp_dir*, *path*)

_add_info_dir(*t*, *tmp_dir*, *files*, *has_prefix*, *info*)

create_conda_pkg(*prefix*, *files*, *info*, *tar_path*, *update_info=None*)

Create a conda package and return a list of warnings.

make_tarbz2(*prefix*, *name='unknown'*, *version='0.0'*, *build_number=0*, *files=None*)

which_package(*path*)

Return the package containing the path.

Provided the path of a (presumably) conda installed file, iterate over the conda packages the file came from. Usually the iteration yields only one package.

which_prefix(*path*)

Return the prefix for the provided path.

Provided the path of a (presumably) conda installed file, return the environment prefix in which the file in located.

main_pip

PEP 621 compatible entry point used when *conda init* has not updated the user shell profile.

Functions

pip_installed_post_parse_hook(args, p)
main(*args, **kwargs)

pip_installed_post_parse_hook(*args*, *p*)

main(**args*, ***kwargs*)

main_remove

CLI implementation for *conda remove*.

Removes the specified packages from an existing environment.

Functions

configure_parser(sub_parsers, **kwargs) → argparse.ArgumentParser
execute(args, parser) → int

configure_parser(*sub_parsers: argparse._SubParsersAction*, ***kwargs*) → argparse.ArgumentParser

execute(*args: argparse.Namespace*, *parser: argparse.ArgumentParser*) → int

main_rename

CLI implementation for *conda rename*.

Renames an existing environment by cloning it and then removing the original environment.

Functions

configure_parser(sub_parsers, **kwargs) → argparse.ArgumentParser

continues on next page

Table 88 – continued from previous page

execute(args, parser) → int	Executes the command for renaming an existing environment.

configure_parser(*sub_parsers: argparse._SubParsersAction*, ***kwargs*) → argparse.ArgumentParser

execute(*args: argparse.Namespace*, *parser: argparse.ArgumentParser*) → int

Executes the command for renaming an existing environment.

main_run

CLI implementation for *conda run*.

Runs the provided command within the specified environment.

Functions

configure_parser(sub_parsers, **kwargs) → argparse.ArgumentParser	
execute(args, parser) → int	

configure_parser(*sub_parsers: argparse._SubParsersAction*, ***kwargs*) → argparse.ArgumentParser

execute(*args: argparse.Namespace*, *parser: argparse.ArgumentParser*) → int

main_search

CLI implementation for *conda search*.

Query channels for packages matching the provided package spec.

Functions

configure_parser(sub_parsers, **kwargs) → argparse.ArgumentParser	
execute(args, parser) → int	Implements *conda search* commands.
_pretty_record_format(record) → str	Format a *PackageRecord* for *pretty_record()*
pretty_record(record[, print]) → None	Pretty prints a *PackageRecord*.

configure_parser(*sub_parsers: argparse._SubParsersAction*, ***kwargs*) → argparse.ArgumentParser

execute(*args: argparse.Namespace*, *parser: argparse.ArgumentParser*) → int

Implements *conda search* commands.

conda search <spec> searches channels for packages. *conda search <spec> --envs* searches environments for packages.

_pretty_record_format(*record:* conda.models.records.PackageRecord) → str

Format a *PackageRecord* for *pretty_record()*

pretty_record(*record:* conda.models.records.PackageRecord, *print=print*) → None

Pretty prints a *PackageRecord*.

Parameters
record -- The *PackageRecord* object to print.

main_update

CLI implementation for *conda update*.

Updates the specified packages in an existing environment.

Functions

configure_parser(sub_parsers, **kwargs) → argparse.ArgumentParser	
execute(args, parser) → int	

configure_parser(*sub_parsers: argparse._SubParsersAction*, ***kwargs*) → argparse.ArgumentParser

execute(*args: argparse.Namespace*, *parser: argparse.ArgumentParser*) → int

Functions

main(*args, **kwargs)	

main(**args*, ***kwargs*)

common

Code in `conda.common` is not conda-specific. Technically, it sits *aside* the application stack and not *within* the stack. It is able to stand independently on its own. The *only* allowed imports of conda code in `conda.common` modules are imports of other `conda.common` modules.

If objects are needed from other parts of conda, they should be passed directly as arguments to functions and methods.

_logic

Classes

_ClauseList	Storage for the CNF clauses, represented as a list of tuples of ints.
_ClauseArray	Storage for the CNF clauses, represented as a flat int array.
_SatSolver	Simple wrapper to call a SAT solver given a _ClauseList/_ClauseArray instance.
_PycoSatSolver	Simple wrapper to call a SAT solver given a _ClauseList/_ClauseArray instance.
_PyCryptoSatSolver	Simple wrapper to call a SAT solver given a _ClauseList/_ClauseArray instance.
_PySatSolver	Simple wrapper to call a SAT solver given a _ClauseList/_ClauseArray instance.
Clauses	

Attributes

TRUE
FALSE
_sat_solver_str_to_cls
_sat_solver_cls_to_str

TRUE = 9223372036854775807

FALSE = -9223372036854775807

class _ClauseList

Storage for the CNF clauses, represented as a list of tuples of ints.

_clause_list = []

append

extend

get_clause_count()

Return number of stored clauses.

save_state()

Get state information to be able to revert temporary additions of supplementary clauses. _ClauseList: state is simply the number of clauses.

restore_state(*saved_state*)

Restore state saved via *save_state*. Removes clauses that were added after the state has been saved.

as_list()

Return clauses as a list of tuples of ints.

as_array()

Return clauses as a flat int array, each clause being terminated by 0.

class _ClauseArray

Storage for the CNF clauses, represented as a flat int array. Each clause is terminated by int(0).

_clause_array

_array_append

_array_extend

extend(*clauses*)

append(*clause*)

get_clause_count()

Return number of stored clauses. This is an O(n) operation since we don't store the number of clauses explicitly due to performance reasons (Python interpreter overhead in self.append).

save_state()

Get state information to be able to revert temporary additions of supplementary clauses. _ClauseArray: state is the length of the int array, NOT number of clauses.

restore_state(*saved_state*)

Restore state saved via *save_state*. Removes clauses that were added after the state has been saved.

as_list()

Return clauses as a list of tuples of ints.

as_array()

Return clauses as a flat int array, each clause being terminated by 0.

class _SatSolver(***run_kwargs*)

Simple wrapper to call a SAT solver given a _ClauseList/_ClauseArray instance.

_run_kwargs

_clauses

add_clause

add_clauses

get_clause_count()

as_list()

save_state()

restore_state(*saved_state*)

run(*m*, ***kwargs*)

abstractmethod setup(*m*, ***kwargs*)

Create a solver instance, add the clauses to it, and return it.

abstractmethod invoke(*solver*)

Start the actual SAT solving and return the calculated solution.

abstractmethod process_solution(*sat_solution*)

Process the solution returned by self.invoke. Returns a list of satisfied variables or None if no solution is found.

class _PycoSatSolver(***run_kwargs*)

Bases: *_SatSolver*

Simple wrapper to call a SAT solver given a _ClauseList/_ClauseArray instance.

setup(*m*, *limit=0*, ***kwargs*)

Create a solver instance, add the clauses to it, and return it.

invoke(*iter_sol*)

Start the actual SAT solving and return the calculated solution.

process_solution(*sat_solution*)

Process the solution returned by self.invoke. Returns a list of satisfied variables or None if no solution is found.

class _PyCryptoSatSolver(***run_kwargs*)

Bases: *_SatSolver*

Simple wrapper to call a SAT solver given a _ClauseList/_ClauseArray instance.

setup(*m*, *threads=1*, ***kwargs*)

Create a solver instance, add the clauses to it, and return it.

invoke(*solver*)

Start the actual SAT solving and return the calculated solution.

process_solution(*solution*)

Process the solution returned by self.invoke. Returns a list of satisfied variables or None if no solution is found.

class _PySatSolver(***run_kwargs*)

Bases: *_SatSolver*

Simple wrapper to call a SAT solver given a _ClauseList/_ClauseArray instance.

setup(*m*, ***kwargs*)

Create a solver instance, add the clauses to it, and return it.

invoke(*solver*)

Start the actual SAT solving and return the calculated solution.

process_solution(*sat_solution*)

Process the solution returned by self.invoke. Returns a list of satisfied variables or None if no solution is found.

_sat_solver_str_to_cls

_sat_solver_cls_to_str

class Clauses(*m=0*, *sat_solver_str=_sat_solver_cls_to_str[_PycoSatSolver]*)

unsat = False

m = 0

_sat_solver

add_clause

add_clauses

get_clause_count()

as_list()

new_var()

assign(*vals*)

Combine(*args*, *polarity*)

Eval(*func*, *args*, *polarity*)

Prevent(*func*, **args*)

Require(*func*, **args*)

Not(*x*, *polarity=None*, *add_new_clauses=False*)

And(*f*, *g*, *polarity*, *add_new_clauses=False*)

Or(*f*, *g*, *polarity*, *add_new_clauses=False*)

Xor(*f*, *g*, *polarity*, *add_new_clauses=False*)

ITE(*c*, *t*, *f*, *polarity*, *add_new_clauses=False*)

All(*iter*, *polarity=None*)

Any(*iter*, *polarity*)

AtMostOne_NSQ(*vals*, *polarity*)

AtMostOne_BDD(*vals*, *polarity=None*)

ExactlyOne_NSQ(*vals*, *polarity*)

ExactlyOne_BDD(*vals*, *polarity*)

LB_Preprocess(*literals*, *coeffs*)

BDD(*literals*, *coeffs*, *nterms*, *lo*, *hi*, *polarity*)

LinearBound(*literals*, *coeffs*, *lo*, *hi*, *preprocess*, *polarity*)

_run_sat(*m*, *limit=0*)

sat(*additional=None*, *includeIf=False*, *limit=0*)

Calculate a SAT solution for the current clause set.

Returned is the list of those solutions. When the clauses are unsatisfiable, an empty list is returned.

minimize(*literals*, *coeffs*, *bestsol=None*, *trymax=False*)

Minimize the objective function given by (coeff, integer) pairs in zip(coeffs, literals). The actual minimization is multiobjective: first, we minimize the largest active coefficient value, then we minimize the sum.

_os

linux

Functions

linux_get_libc_version() → tuple[str, str] | tuple[None, None]	If on linux, returns (libc_family, version), otherwise (None, None).

linux_get_libc_version() → tuple[str, str] | tuple[None, None]

If on linux, returns (libc_family, version), otherwise (None, None).

osx

Functions

copy_acl(source_fd, destination_fd) → None	Copy a file's access-control list between open file descriptors.

continues on next page

Table 96 – continued from previous page

mac_ver() → str	Returns macOS version, without compatibility modes for 11.x.

copy_acl(*source_fd: int*, *destination_fd: int*) → None

Copy a file's access-control list between open file descriptors.

mac_ver() → str

Returns macOS version, without compatibility modes for 11.x. https://github.com/conda/conda/issues/13832 If Python was compiled against macOS <=10.15, we might get 10.16 instead of 11.0. For these cases, we must set SYSTEM_VERSION_COMPAT=0 and call sw_vers directly.

unix

Functions

get_free_space_on_unix(dir_name)	
is_admin_on_unix()	

get_free_space_on_unix(*dir_name*)

is_admin_on_unix()

windows

Classes

SW	Enum where members are also (and must be) ints
ERROR	Enum where members are also (and must be) ints

Functions

get_free_space_on_windows(dir_name)	
is_admin_on_windows()	
_wait_and_close_handle(process_handle)	Waits until spawned process finishes and closes the handle for it.
run_as_admin(args[, wait])	Run command line argument list (*args*) with elevated privileges.

Attributes

PHANDLE

PHANDLE

class SW

Bases: enum.IntEnum

Enum where members are also (and must be) ints

Initialize self. See help(type(self)) for accurate signature.

HIDE = 0

MAXIMIZE = 3

MINIMIZE = 6

RESTORE = 9

SHOW = 5

SHOWDEFAULT = 10

SHOWMAXIMIZED = 3

SHOWMINIMIZED = 2

SHOWMINNOACTIVE = 7

SHOWNA = 8

SHOWNOACTIVATE = 4

SHOWNORMAL = 1

class ERROR

Bases: enum.IntEnum

Enum where members are also (and must be) ints

Initialize self. See help(type(self)) for accurate signature.

ZERO = 0

FILE_NOT_FOUND = 2

PATH_NOT_FOUND = 3

BAD_FORMAT = 11

ACCESS_DENIED = 5

ASSOC_INCOMPLETE = 27

DDE_BUSY = 30

DDE_FAIL = 29

DDE_TIMEOUT = 28

DLL_NOT_FOUND = 32

NO_ASSOC = 31

OOM = 8

SHARE = 26

get_free_space_on_windows(*dir_name*)

is_admin_on_windows()

_wait_and_close_handle(*process_handle*)

Waits until spawned process finishes and closes the handle for it.

run_as_admin(*args*, *wait=True*)

Run command line argument list (*args*) with elevated privileges.

If *wait* is True, the process will block until completion.

Notes

- no stdin / stdout / stderr pipe support
- does not automatically quote arguments (i.e. for paths that may contain spaces)

See: - http://stackoverflow.com/a/19719292/1170370 on 20160407 MCS. - msdn.microsoft.com/en-us/library/windows/desktop/bb762153(v=vs.85).aspx - https://github.com/ContinuumIO/menuinst/blob/master/menuinst/windows/win_elevate.py - https://github.com/saltstack/salt-windows-install/blob/master/deps/salt/python/App/Lib/site-packages/win32/Demos/pipes/runproc.py # NOQA - https://github.com/twonds/twisted/blob/master/twisted/internet/_dumbwin32proc.py - https://stackoverflow.com/a/19982092/2127762 - https://www.codeproject.com/Articles/19165/Vista-UAC-The-Definitive-Guide - https://github.com/JustAMan/pyWinClobber/blob/master/win32elevate.py

on_win = False

compat

Common compatibility code.

Functions

encode_for_env_var(value) → str	Environment names and values need to be string.
encode_environment(env)	
isiterable(obj)	
open_utf8(file[, mode, buffering, encoding, errors, ...])	
ensure_binary(value)	
ensure_text_type(value) → str	
ensure_utf8_encoding(value)	

Attributes

on_win
on_mac
on_linux
ENCODE_ENVIRONMENT
NoneType
primitive_types

on_win = False

on_mac = False

on_linux = True

ENCODE_ENVIRONMENT = True

encode_for_env_var(*value*) → str

Environment names and values need to be string.

encode_environment(*env*)

isiterable(*obj*)

open_utf8(*file*, *mode='r'*, *buffering=-1*, *encoding=None*, *errors=None*, *newline=None*, *closefd=True*)

NoneType

primitive_types

ensure_binary(*value*)

ensure_text_type(*value*) → str

ensure_utf8_encoding(*value*)

configuration

A generalized application configuration utility.

Features include:

- lazy eval
- merges configuration files
- parameter type validation, with custom validation
- parameter aliases

Easily extensible to other source formats, e.g. json and ini

Classes

ParameterFlag	Create a collection of name/value pairs.
RawParameter	
EnvRawParameter	
ArgParseRawParameter	
YamlRawParameter	
DefaultValueRawParameter	Wraps a default value as a RawParameter, for usage in ParameterLoader.
LoadedParameter	Represents a Parameter that has been loaded with configuration value.
PrimitiveLoadedParameter	LoadedParameter type that holds a single python primitive value.
MapLoadedParameter	LoadedParameter type that holds a map (i.e. dict) of LoadedParameters.
SequenceLoadedParameter	LoadedParameter type that holds a sequence (i.e. list) of LoadedParameters.
ObjectLoadedParameter	LoadedParameter type that holds a mapping (i.e. object) of LoadedParameters.

continues on next page

Table 103 – continued from previous page

ConfigurationObject	Dummy class to mark whether a Python object has config parameters within.
Parameter	The Parameter class represents an unloaded configuration parameter, holding type, default
PrimitiveParameter	Parameter type for a Configuration class that holds a single python primitive value.
MapParameter	Parameter type for a Configuration class that holds a map (i.e. dict) of Parameters.
SequenceParameter	Parameter type for a Configuration class that holds a sequence (i.e. list) of Parameters.
ObjectParameter	Parameter type for a Configuration class that holds an object with Parameter fields.
ParameterLoader	ParameterLoader class contains the top level logic needed to load a parameter from start to
ConfigurationType	metaclass for Configuration
Configuration	

Functions

pretty_list(iterable[, padding])	
pretty_map(dictionary[, padding])	
expand_environment_variables(unexpanded)	
raise_errors(errors)	
custom_expandvars(template[, mapping]) → str	Expand variables in a string.
unique_sequence_map(*, unique_key)	Used to validate properties on *Configuration* subclasses defined as a

Attributes

EMPTY_MAP
DEFAULT_CONDARC_FILENAME
ALTERNATIVE_CONDARC_FILENAME
CONDARC_FILENAMES
YAML_EXTENSIONS
_RE_CUSTOM_EXPANDVARS

EMPTY_MAP

pretty_list(*iterable*, *padding=' '*)

pretty_map(*dictionary*, *padding=' '*)

expand_environment_variables(*unexpanded*)

exception ConfigurationError(*message: str | None*, **args: Any*, *caused_by: Any = None*, *guidance: _private.exception_guidance.ErrorGuidance | _private.exception_guidance.ErrorGuidanceTypedDict | None = None*, ***kwargs*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception ConfigurationLoadError(*path*, *message_addition=''*, ***kwargs*)

Bases: *ConfigurationError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception ValidationError(*parameter_name*, *parameter_value*, *source*, *msg=None*, ***kwargs*)

Bases: *ConfigurationError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

parameter_name

parameter_value

source

exception MultipleKeysError(*source*, *keys*, *preferred_key*)

Bases: *ValidationError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

source

keys

exception InvalidTypeError(*parameter_name*, *parameter_value*, *source*, *wrong_type*, *valid_types*, *msg=None*)

Bases: *ValidationError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

wrong_type

valid_types

exception CustomValidationError(*parameter_name*, *parameter_value*, *source*, *custom_message*)

Bases: *ValidationError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception MultiValidationError(*errors*, **args*, ***kwargs*)

Bases: *conda.CondaMultiError*, *ConfigurationError*

Container for multiple conda errors raised together.

This is a control-flow artifact, not a user-facing error type. It must not carry its own `guidance`; printers apply guidance and `conda_error_hints` to each nested leaf error instead.

Initialize self. See help(type(self)) for accurate signature.

raise_errors(*errors*)

class ParameterFlag

Bases: enum.Enum

Create a collection of name/value pairs.

Example enumeration:

```
>>> class Color(Enum):
...     RED = 1
...     BLUE = 2
...     GREEN = 3
```

Access them by:

- attribute access:

```
>>> Color.RED
<Color.RED: 1>
```

- value lookup:

```
>>> Color(1)
<Color.RED: 1>
```

- name lookup:

```
>>> Color['RED']
<Color.RED: 1>
```

Enumerations can be iterated over, and know how many members they have:

```
>>> len(Color)
3
```

```
>>> list(Color)
[<Color.RED: 1>, <Color.BLUE: 2>, <Color.GREEN: 3>]
```

Methods can be added to enumerations, and members can have their own attributes -- see the documentation for details.

final = 'final'

top = 'top'

bottom = 'bottom'

__str__()

classmethod from_name(*name*)

classmethod from_value(*value*)

classmethod from_string(*string*)

class RawParameter(*source*, *key*, *raw_value*)

source

key

__repr__()

abstractmethod value(*parameter_obj*)

abstractmethod keyflag()

abstractmethod valueflags(*parameter_obj*)

classmethod make_raw_parameters(*source*, *from_map*)

class EnvRawParameter(*source*, *key*, *raw_value*)

Bases: *RawParameter*

source = 'envvars'

value(*parameter_obj*)

keyflag()

valueflags(*parameter_obj*)

property __important_split_value

classmethod make_raw_parameters(*appname*)

class ArgParseRawParameter(*source*, *key*, *raw_value*)

Bases: *RawParameter*

source = 'cmd_line'

value(*parameter_obj*)

keyflag()

valueflags(*parameter_obj*)

classmethod make_raw_parameters(*args_from_argparse*)

class YamlRawParameter(*source*, *key*, *raw_value*, *key_comment*)

Bases: *RawParameter*

_key_comment

value(*parameter_obj*)

keyflag()

valueflags(*parameter_obj*)

static _get_yaml_key_comment(*commented_dict*, *key*)

classmethod _get_yaml_list_comments(*value*)

static _get_yaml_list_comment_item(*item*)

static _get_yaml_map_comments(*value*)

classmethod make_raw_parameters(*source*, *from_map*)

classmethod make_raw_parameters_from_file(*filepath*)

Read the provided file path and convert the contents into configuration parameters.

This function will cache the result for each filepath. In order to re-read the same file path with updated content, be sure to clear this cache.

For example:

YamlRawParameter.cache_clear()

classmethod cache_clear() → None

class DefaultValueRawParameter(*source*, *key*, *raw_value*)

Bases: *RawParameter*

Wraps a default value as a RawParameter, for usage in ParameterLoader.

value(*parameter_obj*)

keyflag()

valueflags(*parameter_obj*)

class LoadedParameter(*name*, *value*, *key_flag*, *value_flags*, *validation=None*)

Represents a Parameter that has been loaded with configuration value.

Parameters

- **name** (*str*) -- name of the loaded parameter
- **value** (*LoadedParameter or primitive*) -- the value of the loaded parameter
- **key_flag** (*ParameterFlag or None*) -- priority flag for the parameter itself
- **value_flags** (*Any or None*) -- priority flags for the parameter values
- **validation** (*callable*) -- Given a parameter value as input, return a boolean indicating validity, or alternately return a string describing an invalid value.

_type = None

_element_type = None

_name

value

key_flag

value_flags

_validation = None

__eq__(*other*)

__hash__()

collect_errors(*instance*, *typed_value*, *source='<<merged>>'*)

Validate a LoadedParameter typed value.

Parameters

- **instance** (*Configuration*) -- the instance object used to create the LoadedParameter.

- **typed_value** (*Any*) -- typed value to validate.
- **source** (*str*) -- string description for the source of the typed_value.

expand()

Recursively expands any environment values in the Loaded Parameter.

Returns LoadedParameter

abstractmethod merge(*matches*)

Recursively merges matches into one LoadedParameter.

Parameters
matches (*List<LoadedParameter>*) -- list of matches of this parameter.

Returns: LoadedParameter

typify(*source*)

Recursively types a LoadedParameter.

Parameters
source (*str*) -- string describing the source of the LoadedParameter.

Returns: a primitive, sequence, or map representing the typed value.

static _typify_data_structure(*value*, *source*, *type_hint=None*)

static _match_key_is_important(*loaded_parameter*)

static _first_important_matches(*matches*)

class PrimitiveLoadedParameter(*name*, *element_type*, *value*, *key_flag*, *value_flags*, *validation=None*)

Bases: *LoadedParameter*

LoadedParameter type that holds a single python primitive value.

The python primitive types are str, int, float, complex, bool, and NoneType. In addition, python 2 has long and unicode types.

Parameters

- **element_type** (*type or tuple[type]*) -- Type-validation of parameter's value.
- **value** (*primitive value*) -- primitive python value.

_type

_element_type

__eq__(*other*)

__hash__()

merge(*matches*)

Recursively merges matches into one LoadedParameter.

Parameters
matches (*List<LoadedParameter>*) -- list of matches of this parameter.

Returns: LoadedParameter

class MapLoadedParameter(*name*, *value*, *element_type*, *key_flag*, *value_flags*, *validation=None*)

Bases: LoadedParameter

LoadedParameter type that holds a map (i.e. dict) of LoadedParameters.

Parameters

- **value** (*Mapping*) -- Map of string keys to LoadedParameter values.
- **element_type** (Parameter) -- The Parameter type that is held in value.
- **value_flags** (*Mapping*) -- Map of priority value flags.

_type

_element_type

collect_errors(*instance*, *typed_value*, *source='<<merged>>'*)

Validate a LoadedParameter typed value.

Parameters

- **instance** (Configuration) -- the instance object used to create the LoadedParameter.
- **typed_value** (*Any*) -- typed value to validate.
- **source** (*str*) -- string description for the source of the typed_value.

merge(*parameters: collections.abc.Sequence[*MapLoadedParameter*]*) → *MapLoadedParameter*

Recursively merges matches into one LoadedParameter.

Parameters
matches (*List<LoadedParameter>*) -- list of matches of this parameter.

Returns: LoadedParameter

class SequenceLoadedParameter(*name*, *value*, *element_type*, *key_flag*, *value_flags*, *validation=None*)

Bases: LoadedParameter

LoadedParameter type that holds a sequence (i.e. list) of LoadedParameters.

Parameters

- **value** (*Sequence*) -- Sequence of LoadedParameter values.
- **element_type** (Parameter) -- The Parameter type that is held in the sequence.
- **value_flags** (*Sequence*) -- Sequence of priority value_flags.

_type

_element_type

collect_errors(*instance*, *typed_value*, *source='<<merged>>'*)

Validate a LoadedParameter typed value.

Parameters

- **instance** (Configuration) -- the instance object used to create the LoadedParameter.
- **typed_value** (*Any*) -- typed value to validate.
- **source** (*str*) -- string description for the source of the typed_value.

merge(*matches*)

Recursively merges matches into one LoadedParameter.

Parameters
matches (*List<LoadedParameter>*) -- list of matches of this parameter.

Returns: LoadedParameter

class ObjectLoadedParameter(*name*, *value*, *element_type*, *key_flag*, *value_flags*, *validation=None*)

Bases: *LoadedParameter*

LoadedParameter type that holds a mapping (i.e. object) of LoadedParameters.

Parameters

- **value** (*Sequence*) -- Object with LoadedParameter fields.
- **element_type** (*object*) -- The Parameter type that is held in the sequence.
- **value_flags** (*Sequence*) -- Sequence of priority value_flags.

_type

_element_type

collect_errors(*instance*, *typed_value*, *source='<<merged>>'*)

Validate a LoadedParameter typed value.

Parameters

- **instance** (Configuration) -- the instance object used to create the LoadedParameter.
- **typed_value** (*Any*) -- typed value to validate.
- **source** (*str*) -- string description for the source of the typed_value.

merge(*parameters: collections.abc.Sequence[*ObjectLoadedParameter*]*) → *ObjectLoadedParameter*

Recursively merges matches into one LoadedParameter.

Parameters
matches (*List<LoadedParameter>*) -- list of matches of this parameter.

Returns: LoadedParameter

class ConfigurationObject

Dummy class to mark whether a Python object has config parameters within.

to_json()

Return a serializable object with defaults filled in

class Parameter(*default*, *validation=None*)

The Parameter class represents an unloaded configuration parameter, holding type, default and validation information until the parameter is loaded with a configuration.

Parameters

- **default** (*Any*) -- the typed, python representation default value given if the Parameter is not found in a Configuration.
- **validation** (*callable*) -- Given a parameter value as input, return a boolean indicating validity, or alternately return a string describing an invalid value.

_type = None

_element_type = None

_default

_validation = None

property default

Returns a DefaultValueRawParameter that wraps the actual default value.

get_all_matches(*name*, *names*, *instance*)

Finds all matches of a Parameter in a Configuration instance

Parameters

- **name** (*str*) -- canonical name of the parameter to search for
- **names** (*tuple(str)*) -- alternative aliases of the parameter
- **instance** (Configuration) -- instance of the configuration to search within

Returns (List(RawParameter)): matches of the parameter found in the configuration.

abstractmethod load(*name*, *match*)

Loads a Parameter with the value in a RawParameter.

Parameters

- **name** (*str*) -- name of the parameter to pass through
- **match** (RawParameter) -- the value of the RawParameter match

Returns a LoadedParameter

typify(*name*, *source*, *value*)

class PrimitiveParameter(*default*, *element_type=None*, *validation=None*)

Bases: *Parameter*

Parameter type for a Configuration class that holds a single python primitive value.

The python primitive types are str, int, float, complex, bool, and NoneType. In addition, python 2 has long and unicode types.

Parameters

- **default** (*primitive value*) -- default value if the Parameter is not found.
- **element_type** (*type or tuple[type]*) -- Type-validation of parameter's value. If None, type(default) is used.

_type

_element_type

load(*name*, *match*)

Loads a Parameter with the value in a RawParameter.

Parameters

- **name** (*str*) -- name of the parameter to pass through
- **match** (RawParameter) -- the value of the RawParameter match

Returns a LoadedParameter

class MapParameter(*element_type*, *default=frozendict()*, *validation=None*)

Bases: *Parameter*

Parameter type for a Configuration class that holds a map (i.e. dict) of Parameters.

Parameters

- **element_type** (Parameter) -- The Parameter type held in the MapParameter.
- **default** (*Mapping*) -- The parameter's default value. If None, will be an empty dict.

_type

_element_type

get_all_matches(*name*, *names*, *instance*)

Finds all matches of a Parameter in a Configuration instance

Parameters

- **name** (*str*) -- canonical name of the parameter to search for
- **names** (*tuple(str)*) -- alternative aliases of the parameter
- **instance** (Configuration) -- instance of the configuration to search within

Returns (List(RawParameter)): matches of the parameter found in the configuration.

load(*name*, *match*)

Loads a Parameter with the value in a RawParameter.

Parameters

- **name** (*str*) -- name of the parameter to pass through
- **match** (RawParameter) -- the value of the RawParameter match

Returns a LoadedParameter

class SequenceParameter(*element_type*, *default=()*, *validation=None*, *string_delimiter=','*)

Bases: *Parameter*

Parameter type for a Configuration class that holds a sequence (i.e. list) of Parameters.

Parameters

- **element_type** (Parameter) -- The Parameter type that is held in the sequence.
- **default** (*Sequence*) -- default value, empty tuple if not given.
- **string_delimiter** (*str*) -- separation string used to parse string into sequence.

_type

_element_type

string_delimiter = ','

get_all_matches(*name*, *names*, *instance*)

Finds all matches of a Parameter in a Configuration instance

Parameters

- **name** (*str*) -- canonical name of the parameter to search for
- **names** (*tuple(str)*) -- alternative aliases of the parameter

- **instance** (Configuration) -- instance of the configuration to search within

Returns (List(RawParameter)): matches of the parameter found in the configuration.

load(*name*, *match*)

Loads a Parameter with the value in a RawParameter.

Parameters

- **name** (*str*) -- name of the parameter to pass through
- **match** (RawParameter) -- the value of the RawParameter match

Returns a LoadedParameter

class ObjectParameter(*element_type*, *default=ConfigurationObject()*, *validation=None*)

Bases: *Parameter*

Parameter type for a Configuration class that holds an object with Parameter fields.

Parameters

- **element_type** (*object*) -- The object type with parameter fields held in ObjectParameter.
- **default** (*Sequence*) -- default value, empty tuple if not given.

_type

_element_type

get_all_matches(*name*, *names*, *instance*)

Finds all matches of a Parameter in a Configuration instance

Parameters

- **name** (*str*) -- canonical name of the parameter to search for
- **names** (*tuple(str)*) -- alternative aliases of the parameter
- **instance** (Configuration) -- instance of the configuration to search within

Returns (List(RawParameter)): matches of the parameter found in the configuration.

load(*name*, *match*)

Loads a Parameter with the value in a RawParameter.

Parameters

- **name** (*str*) -- name of the parameter to pass through
- **match** (RawParameter) -- the value of the RawParameter match

Returns a LoadedParameter

class ParameterLoader(*parameter_type*, *aliases=()*, *expandvars=False*)

ParameterLoader class contains the top level logic needed to load a parameter from start to finish.

Parameters

- **parameter_type** (Parameter) -- the type of Parameter that is stored in the loader.
- **aliases** (*tuple(str)*) -- alternative aliases for the Parameter
- **expandvars** (*bool*) -- whether or not to recursively expand environmental variables.

_name = None

_names = None

type

aliases = ()

_expandvars = False

_set_name(*name*)

property name

property names

__get__(*instance*, *instance_type*)

_raw_parameters_from_single_source(*raw_parameters*)

static raw_parameters_from_single_source(*name*, *names*, *raw_parameters*)

class ConfigurationType(*name*, *bases*, *attr*)

Bases: type

metaclass for Configuration

property _parameter_loaders: dict[str, *ParameterLoader*]

__call__(**args*, ***kwargs*)

DEFAULT_CONDARC_FILENAME: Final = '.condarc'

ALTERNATIVE_CONDARC_FILENAME: Final = 'condarc'

CONDARC_FILENAMES

YAML_EXTENSIONS = ('.yml', '.yaml')

_RE_CUSTOM_EXPANDVARS

custom_expandvars(*template: str*, *mapping: collections.abc.Mapping[str, Any] = {}*, */ (Positional-only parameter separator (PEP 570))*, ***kwargs*) → str

Expand variables in a string.

Inspired by *string.Template* and modified to mirror *os.path.expandvars* functionality allowing custom variables without mutating *os.environ*.

Expands POSIX and Windows CMD environment variables as follows:

- $VARIABLE → value of VARIABLE
- ${VARIABLE} → value of VARIABLE
- %VARIABLE% → value of VARIABLE

Invalid substitutions are left as-is:

- $MISSING → $MISSING
- ${MISSING} → ${MISSING}
- %MISSING% → %MISSING%
- $$ → $$

- %% → %%
- $ → $
- % → %

class Configuration(*search_path=()*, *app_name=None*, *argparse_args=None*, ***kwargs*)

raw_data

cache

_reset_callbacks: dict[collections.abc.Callable, None]

_validation_errors

classmethod _set_parameter_names_and_aliases()

Build parameter_names_and_aliases from the class's parameter loaders.

static _expand_search_path(*search_path: conda.common.path.PathsType*, ***kwargs*) → collections.abc.Iterable[pathlib.Path]

classmethod _load_search_path(*search_path: collections.abc.Iterable[pathlib.Path]*) → collections.abc.Iterable[tuple[pathlib.Path, dict]]

_set_search_path(*search_path: conda.common.path.PathsType*, ***kwargs*)

_set_env_vars(*app_name=None*)

_set_argparse_args(*argparse_args*)

_set_raw_data(*raw_data: collections.abc.Mapping[collections.abc.Hashable, dict]*)

name_for_alias(*alias: str*, *ignore_private: bool = True*) → str | None

Find the canonical parameter name for a given alias.

This method searches through all configuration parameters to find the canonical parameter name that corresponds to the given alias. It's useful for resolving parameter aliases to their primary names in configuration contexts.

Parameters

- **alias** (*str*) -- The parameter alias to look up.
- **ignore_private** (*bool, optional*) -- If True (default), exclude private parameters (those starting with underscore) from the search. If False, include all parameters regardless of privacy.

Returns

The canonical parameter name if the alias is found, otherwise None.

Return type

str | None

Example

```
>>> config = Configuration()
>>> config.name_for_alias("channel_priority")
'channel_priority'
>>> config.name_for_alias("unknown_alias")
None
```

_get_parameter_loader(*parameter_name*)

Get parameter loader with fallback for missing parameters.

_reset_cache()

register_reset_callback(*callback*)

register_reset_callaback(*callback*)

check_source(*source*)

validate_all()

static _collect_validation_error(*func*, **args*, ***kwargs*)

validate_configuration()

post_build_validation()

collect_all() → dict[str | pathlib.Path, dict]

Collect all explicitly set parameters from all sources. Do not include default values.

The possible sources are:

- Environment variables (constants.ENV_VARS_SOURCE, currently *envvars*)
- Command line arguments (constants.CMD_LINE_SOURCE, currently *cmd_line*)
- Configuration files (direct paths to the configuration files)

Returns

A dictionary of all explicitly set parameters from all sources. The keys are the source names, and the values are dictionaries of the parameters and their values.

Return type

dict[str | Path, dict]

describe_parameter(*parameter_name*)

list_parameters(*aliases: bool = False*)

typify_parameter(*parameter_name*, *value*, *source*)

abstractmethod get_descriptions()

unique_sequence_map(*, *unique_key: str*)

Used to validate properties on *Configuration* subclasses defined as a `SequenceParameter(MapParameter())` where the map contains a single key that should be regarded as unique. This decorator will handle removing duplicates and merging to a single sequence.

constants

Common constants.

NULL

TRACE = 5

datetime

Date, time, duration, and timestamp helpers.

Functions

parse_duration(value) → datetime.timedelta | None	Parse plain, compact, or ISO 8601 duration strings.
parse_datetime_to_timestamp(value) → float | None	Parse an ISO/RFC 3339 datetime into a POSIX timestamp.
normalize_timestamp_seconds(value) → float	Normalize a seconds-or-milliseconds POSIX timestamp to seconds.

Attributes

COMPACT_DURATION_UNITS
MAX_SECONDS_TIMESTAMP
_COMPACT_DURATION_RE
_ISO8601_DURATION_RE
_DATE_ONLY_RE
_COMPACT_DATE_ONLY_RE

COMPACT_DURATION_UNITS

MAX_SECONDS_TIMESTAMP = 253402300799

_COMPACT_DURATION_RE

_ISO8601_DURATION_RE

_DATE_ONLY_RE

_COMPACT_DATE_ONLY_RE

parse_duration(*value: str*) → datetime.timedelta | None

Parse plain, compact, or ISO 8601 duration strings.

parse_datetime_to_timestamp(*value: str*) → float | None

Parse an ISO/RFC 3339 datetime into a POSIX timestamp.

Naive datetimes are interpreted as UTC. Date-only values are interpreted as the start of the following UTC day.

normalize_timestamp_seconds(*value: str | int | float*) → float

Normalize a seconds-or-milliseconds POSIX timestamp to seconds.

io

Common I/O utilities.

Classes

DeltaSecondsFormatter	Logging formatter with additional attributes for run time logging.
ContextDecorator	Base class for a context manager class (implementing __enter__() and __exit__()) that also
SwallowBrokenPipe	Ignore BrokenPipeError and errors related to stdout or stderr being closed by a downstream program.
CaptureTarget	Constants used for contextmanager captured.
time_recorder	Context decorator for recording execution times to a CSV file.

Functions

dashlist(iterable[, indent]) → str	Format an iterable as a dashed list with optional indentation.
env_vars([var_map, callback, stack_callback]) → collections.abc.Generator[None, None, None]	Temporarily set environment variables.
env_var(name, value[, callback, stack_callback]) → collections.abc.Generator[None, None, None]	Temporarily set a single environment variable.
env_unmodified([callback]) → collections.abc.Generator[None, None, None]	Context manager that ensures environment variables are not modified.
captured([stdout, stderr]) → collections.abc.Generator[Any, None, None]	Capture outputs of sys.stdout and sys.stderr.
argv(args_list) → collections.abc.Generator[None, None, None]	Temporarily replace sys.argv with the given list.
disable_logger(logger_name) → collections.abc.Generator[None, None, None]	Temporarily disable a logger by setting its level above CRITICAL.
stderr_log_level(level[, logger_name]) → collections.abc.Generator[None, None, None]	Temporarily set a logger to output to stderr at the given level.
attach_stderr_handler([level, logger_name, propagate, ...]) → None	Attach a new *stderr* handler to the given logger and configure both.
timeout(timeout_secs, func, *args[, default_return]) → Any	Enforce a maximum time for a callable to complete.
_load_concurrency() → None	Lazily define executor classes and as_completed on first access.
__getattr__(name)	
get_instrumentation_record_file() → str	Return the path to the instrumentation record file.
print_instrumentation_data() → None	Print aggregated instrumentation data from the record file as JSON.

Attributes

_FORMATTER
swallow_broken_pipe
_LAZY_CONCURRENCY

class DeltaSecondsFormatter(*fmt: str | None = None*, *datefmt: str | None = None*)

Bases: logging.Formatter

Logging formatter with additional attributes for run time logging.

`delta_secs`

Elapsed seconds since last log/format call (or creation of logger).

`relative_created_secs`

Like *relativeCreated*, time relative to the initialization of the *logging* module but conveniently scaled to seconds as a *float* value.

Initialize the formatter with optional format strings.

Parameters

- **fmt** -- Log message format string.
- **datefmt** -- Date format string.

prev_time

format(*record: logging.LogRecord*) → str

Format the specified record as text.

The record's attribute dictionary is used as the operand to a string formatting operation which yields the returned string. Before formatting the dictionary, a couple of preparatory steps are carried out. The message attribute of the record is computed using LogRecord.getMessage(). If the formatting string uses the time (as determined by a call to usesTime(), formatTime() is called to format the event time. If there is exception information, it is formatted using formatException() and appended to the message.

_FORMATTER

dashlist(*iterable: collections.abc.Iterable[Any]*, *indent: int = 2*) → str

Format an iterable as a dashed list with optional indentation.

Parameters

- **iterable** -- Items to format.
- **indent** -- Number of spaces for indentation.

Returns

Formatted string with each item on a new line prefixed by ``- ``.

class ContextDecorator

Base class for a context manager class (implementing __enter__() and __exit__()) that also makes it a decorator.

__call__(*f: collections.abc.Callable[..., Any]*) → collections.abc.Callable[..., Any]

class SwallowBrokenPipe

Bases: *ContextDecorator*

Ignore BrokenPipeError and errors related to stdout or stderr being closed by a downstream program.

__enter__() → None

__exit__(*exc_type:* type*[BaseException]* | *None*, *exc_val: BaseException* | *None*, *exc_tb: Any*) → bool | None

swallow_broken_pipe

class CaptureTarget

Bases: enum.Enum

Constants used for contextmanager captured.

Used similarly like the constants PIPE, STDOUT for stdlib's subprocess.Popen.

STRING = -1

STDOUT = -2

env_vars(*var_map: dict[str, str] | None = None*, *callback: collections.abc.Callable[[], None] | None = None*, *stack_callback: collections.abc.Callable[[bool], None] | None = None*) → collections.abc.Generator[None, None, None]

Temporarily set environment variables.

Parameters

- **var_map** -- Dictionary of environment variable names to values.
- **callback** -- Optional callback invoked when entering and exiting the context.
- **stack_callback** -- Optional callback invoked with True when entering, False when exiting.

env_var(*name: str*, *value: str*, *callback: collections.abc.Callable[[], None] | None = None*, *stack_callback: collections.abc.Callable[[bool], None] | None = None*) → collections.abc.Generator[None, None, None]

Temporarily set a single environment variable.

Parameters

- **name** -- Environment variable name.
- **value** -- Environment variable value.
- **callback** -- Optional callback invoked when entering and exiting the context.
- **stack_callback** -- Optional callback invoked with True when entering, False when exiting.

env_unmodified(*callback: collections.abc.Callable[[], None] | None = None*) → collections.abc.Generator[None, None, None]

Context manager that ensures environment variables are not modified.

Parameters

callback -- Optional callback invoked when entering and exiting the context.

captured(*stdout: CaptureTarget | None | Any = CaptureTarget.STRING*, *stderr: CaptureTarget | None | Any = CaptureTarget.STRING*) → collections.abc.Generator[Any, None, None]

Capture outputs of sys.stdout and sys.stderr.

If stdout is STRING, capture sys.stdout as a string, if stdout is None, do not capture sys.stdout, leaving it untouched, otherwise redirect sys.stdout to the file-like object given by stdout.

Behave correspondingly for stderr with the exception that if stderr is STDOUT, redirect sys.stderr to stdout target and set stderr attribute of yielded object to None.

```
>>> from conda.common.io import captured
>>> with captured() as c:
...     print("hello world!")
...
>>> c.stdout
'hello world!\n'
```

Parameters

- **stdout** -- capture target for sys.stdout, one of STRING, None, or file-like object
- **stderr** -- capture target for sys.stderr, one of STRING, STDOUT, None, or file-like object

Yields

CapturedText --

has attributes stdout, stderr which are either strings, None or the corresponding file-like function argument.

argv(*args_list: list[str]*) → collections.abc.Generator[None, None, None]

Temporarily replace sys.argv with the given list.

Parameters

args_list -- List of arguments to use as sys.argv.

disable_logger(*logger_name: str*) → collections.abc.Generator[None, None, None]

Temporarily disable a logger by setting its level above CRITICAL.

Parameters

logger_name -- Name of the logger to disable.

stderr_log_level(*level: int*, *logger_name: str | None = None*) → collections.abc.Generator[None, None, None]

Temporarily set a logger to output to stderr at the given level.

Parameters

- **level** -- Logging level (e.g., `logging.INFO`).
- **logger_name** -- Name of the logger; if None, uses the root logger.

attach_stderr_handler(*level: int = WARNING*, *logger_name: str | None = None*, *propagate: bool = False*, *formatter: logging.Formatter | None = None*, *filters: collections.abc.Iterable[Any] | None = None*) → None

Attach a new *stderr* handler to the given logger and configure both.

This function creates a new StreamHandler that writes to *stderr* and attaches it to the logger given by *logger_name* (which maybe *None*, in which case the root logger is used). If the logger already has a handler by the name of *stderr*, it is removed first.

The given *level* is set **for the handler**, not for the logger; however, this function also sets the level of the given logger to the minimum of its current effective level and the new handler level, ensuring that the handler will receive the required log records, while minimizing the number of unnecessary log events. It also sets the loggers *propagate* property according to the *propagate* argument. The *formatter* argument can be used to set the formatter of the handler.

timeout(*timeout_secs: int | float*, *func: collections.abc.Callable[..., Any]*, **args: Any*, *default_return: Any = None*, ***kwargs: Any*) → Any

Enforce a maximum time for a callable to complete.

Not yet implemented on Windows.

Parameters

- **timeout_secs** -- Maximum seconds to allow the callable to run.
- **func** -- Callable to execute.
- **args** -- Positional arguments to pass to the callable.
- **default_return** -- Value to return if timeout or KeyboardInterrupt occurs.

- **kwargs** -- Keyword arguments to pass to the callable.

Returns
Return value of the callable, or `default_return` on timeout/interrupt.

_load_concurrency() → None

Lazily define executor classes and as_completed on first access.

Importing concurrent.futures + threading costs ~45 modules. Deferring this to first use keeps `import conda.common.io` lightweight for code paths that never use parallel I/O (dashlist, time_recorder, etc.).

_LAZY_CONCURRENCY

__getattr__(*name: str*)

get_instrumentation_record_file() → str

Return the path to the instrumentation record file.

Returns
Expanded path from `CONDA_INSTRUMENTATION_RECORD_FILE` or default location.

class time_recorder(*entry_name: str | None = None*, *module_name: str | None = None*)

Bases: *ContextDecorator*

Context decorator for recording execution times to a CSV file.

record_file = b'.'

start_time: float | None = None

total_call_num: dict[str, int]

total_run_time: dict[str, float]

entry_name = None

module_name = None

_set_entry_name(*f: collections.abc.Callable[..., Any]*) → None

__call__(*f: collections.abc.Callable[..., Any]*) → collections.abc.Callable[..., Any]

__enter__() → *time_recorder*

__exit__(*exc_type:* type*[BaseException] | None*, *exc_val: BaseException | None*, *exc_tb: Any*) → None

classmethod log_totals() → None

_ensure_dir() → None

print_instrumentation_data() → None

Print aggregated instrumentation data from the record file as JSON.

iterators

Replacements for parts of the toolz library.

Functions

groupby_to_dict(keyfunc, iterable) → dict[K, list[T]]	A *toolz*-style groupby implementation.
unique(iterable) → collections.abc.Iterable[T]	A *toolz* inspired *unique* implementation.

Attributes

T

T

groupby_to_dict(*keyfunc: collections.abc.Callable[[T], K]*, *iterable: collections.abc.Iterable[T]*) → dict[K, list[*T*]]

A *toolz*-style groupby implementation.

Returns a dictionary of { key: [group] } instead of iterators.

unique(*iterable: collections.abc.Iterable[T]*) → collections.abc.Iterable[*T*]

A *toolz* inspired *unique* implementation.

Returns a generator of unique elements in the sequence

logic

The basic idea to nest logical expressions is instead of trying to denest things via distribution, we add new variables. So if we have some logical expression expr, we replace it with x and add expr <-> x to the clauses, where x is a new variable, and expr <-> x is recursively evaluated in the same way, so that the final clauses are ORs of atoms.

To use this, create a new Clauses object with the max var, for instance, if you already have [[1, 2, -3]], you would use C = Clause(3). All functions return a new literal, which represents that function, or True or False if the expression can be resolved fully. They may also add new clauses to C.clauses, which will then be delivered to the SAT solver.

All functions take atoms as arguments (an atom is an integer, representing a literal or a negated literal, or boolean constants True or False; that is, it is the callers' responsibility to do the conversion of expressions recursively. This is done because we do not have data structures representing the various logical classes, only atoms.

The polarity argument can be set to True or False if you know that the literal being used will only be used in the positive or the negative, respectively (e.g., you will only use x, not -x). This will generate fewer clauses. It is probably best if you do not take advantage of this directly, but rather through the Require and Prevent functions.

Classes

Clauses

Functions

minimal_unsatisfiable_subset(clauses, sat, explicit_specs)	Given a set of clauses, find a minimal unsatisfiable subset (an

Attributes

TRUE
FALSE
PycoSatSolver
PyCryptoSatSolver
PySatSolver

TRUE = 9223372036854775807

FALSE = -9223372036854775807

PycoSatSolver = 'pycosat'

PyCryptoSatSolver = 'pycryptosat'

PySatSolver = 'pysat'

class Clauses(*m=0*, *sat_solver=PycoSatSolver*)

names

indices

_clauses

property m

property unsat

get_clause_count()

as_list()

_check_variable(*variable*)

_check_literal(*literal*)

add_clause(*clause*)

add_clauses(*clauses*)

name_var(*m*, *name*)

new_var(*name=None*)

from_name(*name*)

from_index(*m*)

_assign(*vals*, *name=None*)

_convert(*x*)

_eval(*func*, *args*, *no_literal_args*, *polarity*, *name*)

Prevent(*what*, **args*)

Require(*what*, **args*)

Not(*x*, *polarity=None*, *name=None*)

And(*f*, *g*, *polarity=None*, *name=None*)

Or(*f*, *g*, *polarity=None*, *name=None*)

Xor(*f*, *g*, *polarity=None*, *name=None*)

ITE(*c*, *t*, *f*, *polarity=None*, *name=None*)

If c Then t Else f.

In this function, if any of c, t, or f are True and False the resulting expression is resolved.

All(*iter*, *polarity=None*, *name=None*)

Any(*vals*, *polarity=None*, *name=None*)

AtMostOne_NSQ(*vals*, *polarity=None*, *name=None*)

AtMostOne_BDD(*vals*, *polarity=None*, *name=None*)

AtMostOne(*vals*, *polarity=None*, *name=None*)

ExactlyOne_NSQ(*vals*, *polarity=None*, *name=None*)

ExactlyOne_BDD(*vals*, *polarity=None*, *name=None*)

ExactlyOne(*vals*, *polarity=None*, *name=None*)

LinearBound(*equation*, *lo*, *hi*, *preprocess=True*, *polarity=None*, *name=None*)

sat(*additional=None*, *includeIf=False*, *names=False*, *limit=0*)

Calculate a SAT solution for the current clause set.

Returned is the list of those solutions. When the clauses are unsatisfiable, an empty list is returned.

itersolve(*constraints=None*, *m=None*)

minimize(*objective*, *bestsol=None*, *trymax=False*)

minimal_unsatisfiable_subset(*clauses*, *sat*, *explicit_specs*)

Given a set of clauses, find a minimal unsatisfiable subset (an unsatisfiable core)

A set is a minimal unsatisfiable subset if no proper subset is unsatisfiable. A set of clauses may have many minimal unsatisfiable subsets of different sizes.

sat should be a function that takes a tuple of clauses and returns True if the clauses are satisfiable and False if they are not. The algorithm will work with any order-reversing function (reversing the order of subset and the order False < True), that is, any function where (A <= B) iff (sat(B) <= sat(A)), where A <= B means A is a subset of B and False < True).

path

Common path utilities.

_cygpath

Functions

nt_to_posix(path, prefix[, cygdrive]) → str	A fallback implementation of *cygpath --unix*.
_get_root(prefix) → str	
_get_RE_WIN_ROOT(prefix) → re.Pattern	
_to_unix_root(match) → str	
_to_unix_mount(match) → str	
_to_unix_drive(match, cygdrive) → str	
posix_to_nt(path, prefix[, cygdrive]) → str	A fallback implementation of *cygpath --windows*.
_to_win_drive(match) → str	
_to_win_mount(match) → str	
_to_win_root(match, root) → str	
_resolve_path(path, sep) → str	
resolve_paths(paths, pathsep, sep) → str	

Attributes

RE_WIN_MOUNT
RE_WIN_DRIVE
RE_UNIX_DRIVE
RE_UNIX_MOUNT
RE_UNIX_ROOT

nt_to_posix(*path: conda.common.path.PathType*, *prefix: conda.common.path.PathType | None*, *cygdrive: bool = False*) → str

A fallback implementation of *cygpath --unix*.

Parameters

- **path** -- The path to convert.
- **prefix** -- The Windows style prefix directory to use for the conversion. If not provided, no checks for root paths will be made.
- **cygdrive** -- Whether to use the Cygwin-style drive prefix.

_get_root(*prefix: str*) → str

_get_RE_WIN_ROOT(*prefix: str*) → re.Pattern

_to_unix_root(*match: re.Match*) → str

RE_WIN_MOUNT

_to_unix_mount(*match: re.Match*) → str

RE_WIN_DRIVE

_to_unix_drive(*match: re.Match*, *cygdrive: bool*) → str

posix_to_nt(*path: conda.common.path.PathType*, *prefix: conda.common.path.PathType | None*, *cygdrive: bool = False*) → str

A fallback implementation of *cygpath --windows*.

Parameters

- **path** -- The path to convert.
- **prefix** -- The Windows style prefix directory to use for the conversion. If not provided, no checks for root paths will be made.
- **cygdrive** -- Unused. Present to keep the signature consistent with *nt_to_posix*.

RE_UNIX_DRIVE

_to_win_drive(*match: re.Match*) → str

RE_UNIX_MOUNT

_to_win_mount(*match: re.Match*) → str

RE_UNIX_ROOT

_to_win_root(*match: re.Match*, *root: str*) → str

_resolve_path(*path: str*, *sep: str*) → str

resolve_paths(*paths: str*, *pathsep: str*, *sep: str*) → str

directories

Common directory utilities.

Functions

tokenized_startswith(test_iterable, startswith_iterable)	
get_all_directories(files) → list[tuple[str, ...]]	
get_leaf_directories(files) → collections.abc.Sequence[str]	
explode_directories(child_directories) → set[str]	

tokenized_startswith(*test_iterable*, *startswith_iterable*)

get_all_directories(*files: collections.abc.Iterable[str]*) → list[tuple[str, ...]]

get_leaf_directories(*files: collections.abc.Iterable[str]*) → collections.abc.Sequence[str]

explode_directories(*child_directories: collections.abc.Iterable[tuple[str, ...]]*) → set[str]

python

Common Python specific path utilities.

Functions

is_valid_import_path(path) → bool	Python import paths are a sequence of non-keyword Python identifiers separated by one period.

continues on next page

Table 119 – continued from previous page

pyc_path(py_path, python_major_minor_version)	This must not return backslashes on Windows as that will break
missing_pyc_files(python_major_minor_version, files)	
parse_entry_point_def(ep_definition)	
get_python_short_path([python_version])	
get_python_site_packages_short_path(python_ve	
get_major_minor_version(string[, with_dot]) → str	
get_python_noarch_target_path(source_short_path ...)	

Attributes

_VERSION_REGEX

is_valid_import_path(*path: str*) → bool

Python import paths are a sequence of non-keyword Python identifiers separated by one period.

pyc_path(*py_path*, *python_major_minor_version*)

This must not return backslashes on Windows as that will break tests and leads to an eventual need to make url_to_path return backslashes too and that may end up changing files on disc or to the result of comparisons with the contents of them.

missing_pyc_files(*python_major_minor_version*, *files*)

parse_entry_point_def(*ep_definition*)

get_python_short_path(*python_version=None*)

get_python_site_packages_short_path(*python_version*)

_VERSION_REGEX

get_major_minor_version(*string: str*, *with_dot=True*) → str

get_python_noarch_target_path(*source_short_path*, *target_site_packages_short_path*)

windows

Common Windows path utilities.

Functions

win_path_ok(path)	
win_path_double_escape(path)	
win_path_backout(path)	
_path_to(paths[, prefix]) → str | tuple[str, ...] | None	
win_path_to_unix(paths[, prefix, cygdrive]) → str | tuple[str, ...] | None	Convert Windows paths to Unix paths.
unix_path_to_win(paths[, prefix, cygdrive]) → str | tuple[str, ...] | None	Convert Unix paths to Windows paths.

win_path_ok(*path*)

win_path_double_escape(*path*)

win_path_backout(*path*)

_path_to(*paths: conda.common.path.PathType | conda.common.path.PathsType | None, prefix: conda.common.path.PathType | None = None, *, cygdrive: bool, to_unix: bool*) → str | tuple[str, ...] | None

win_path_to_unix(*paths: conda.common.path.PathType | conda.common.path.PathsType | None, prefix: conda.common.path.PathType | None = None, *, cygdrive: bool = False*) → str | tuple[str, ...] | None

Convert Windows paths to Unix paths.

> **Note**
>
> Produces unexpected results when run on Unix.

Parameters

- **paths** -- The path(s) to convert.
- **prefix** -- The (Windows path-style) prefix directory to use for the conversion. If not provided, no checks for prefix paths will be made.
- **cygdrive** -- Whether to use the Cygwin-style drive prefix.

unix_path_to_win(*paths: conda.common.path.PathType | conda.common.path.PathsType | None, prefix: conda.common.path.PathType | None = None, *, cygdrive: bool = False*) → str | tuple[str, ...] | None

Convert Unix paths to Windows paths.

> **Note**
>
> Produces unexpected results when run on Unix.

Parameters

- **paths** -- The path(s) to convert.
- **prefix** -- The (Windows path-style) prefix directory to use for the conversion. If not provided, no checks for prefix paths will be made.
- **cygdrive** -- Unused. Present to keep the signature consistent with *win_path_to_unix*.

Functions

explode_directories(child_directories) → set[str]	
get_all_directories(files) → list[tuple[str, ...]]	
get_leaf_directories(files) → collections.abc.Sequence[str]	

continues on next page

Table 122 – continued from previous page

tokenized_startswith(test_iterable, startswith_iterable)	
get_major_minor_version(string[, with_dot]) → str	
get_python_noarch_target_path(source_short_path ...)	
get_python_short_path([python_version])	
get_python_site_packages_short_path(python_ve	
missing_pyc_files(python_major_minor_version, files)	
parse_entry_point_def(ep_definition)	
pyc_path(py_path, python_major_minor_version)	This must not return backslashes on Windows as that will break
unix_path_to_win(paths[, prefix, cygdrive]) → str | tuple[str, ...] | None	Convert Unix paths to Windows paths.
win_path_backout(path)	
win_path_double_escape(path)	
win_path_ok(path)	
win_path_to_unix(paths[, prefix, cygdrive]) → str | tuple[str, ...] | None	Convert Windows paths to Unix paths.

explode_directories(*child_directories: collections.abc.Iterable[tuple[str, ...]]*) → set[str]

get_all_directories(*files: collections.abc.Iterable[str]*) → list[tuple[str, ...]]

get_leaf_directories(*files: collections.abc.Iterable[str]*) → collections.abc.Sequence[str]

tokenized_startswith(*test_iterable*, *startswith_iterable*)

get_major_minor_version(*string: str*, *with_dot=True*) → str

get_python_noarch_target_path(*source_short_path*, *target_site_packages_short_path*)

get_python_short_path(*python_version=None*)

get_python_site_packages_short_path(*python_version*)

missing_pyc_files(*python_major_minor_version*, *files*)

parse_entry_point_def(*ep_definition*)

pyc_path(*py_path*, *python_major_minor_version*)

This must not return backslashes on Windows as that will break tests and leads to an eventual need to make url_to_path return backslashes too and that may end up changing files on disc or to the result of comparisons with the contents of them.

unix_path_to_win(*paths: conda.common.path.PathType | conda.common.path.PathsType | None*, *prefix: conda.common.path.PathType | None = None*, ***, *cygdrive: bool = False*) → str | tuple[str, ...] | None

Convert Unix paths to Windows paths.

> **Note**
>
> Produces unexpected results when run on Unix.

Parameters

- **paths** -- The path(s) to convert.
- **prefix** -- The (Windows path-style) prefix directory to use for the conversion. If not provided, no checks for prefix paths will be made.
- **cygdrive** -- Unused. Present to keep the signature consistent with *win_path_to_unix*.

win_path_backout(*path*)

win_path_double_escape(*path*)

win_path_ok(*path*)

win_path_to_unix(*paths: conda.common.path.PathType | conda.common.path.PathsType | None*, *prefix: conda.common.path.PathType | None = None*, ***, *cygdrive: bool = False*) → str | tuple[str, ...] | None

Convert Windows paths to Unix paths.

> **Note**
>
> Produces unexpected results when run on Unix.

Parameters

- **paths** -- The path(s) to convert.
- **prefix** -- The (Windows path-style) prefix directory to use for the conversion. If not provided, no checks for prefix paths will be made.
- **cygdrive** -- Whether to use the Cygwin-style drive prefix.

serialize

YAML and JSON serialization and deserialization functions.

json

JSON serialization utilities for conda.

Classes

CondaJSONEncoder	Extensible JSON <https://json.org> encoder for Python data structures.

Functions

write(...)
dump(obj, fp, **kwargs) → None
dumps(obj, **kwargs) → str
read(...)
load(fp, **kwargs) → Any
loads(s, **kwargs) → Any

Attributes

JSONDecodeError

class CondaJSONEncoder(*, *skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *sort_keys=False*, *indent=None*, *separators=None*, *default=None*)

Bases: `json.JSONEncoder`

Extensible JSON <https://json.org> encoder for Python data structures.

Supports the following objects and types by default:

Python	JSON
dict, frozendict	object
list, tuple	array
str	string
int, float	number
True	true
False	false
None	null

To extend this to recognize other objects, subclass and implement a `.default()` method with another method that returns a serializable object for `o` if possible, otherwise it should call the superclass implementation (to raise `TypeError`).

Constructor for JSONEncoder, with sensible defaults.

If skipkeys is false, then it is a TypeError to attempt encoding of keys that are not str, int, float, bool or None. If skipkeys is True, such items are simply skipped.

If ensure_ascii is true, the output is guaranteed to be str objects with all incoming non-ASCII and non-printable characters escaped. If ensure_ascii is false, the output can contain non-ASCII and non-printable characters.

If check_circular is true, then lists, dicts, and custom encoded objects will be checked for circular references during encoding to prevent an infinite recursion (which would cause an RecursionError). Otherwise, no such check takes place.

If allow_nan is true, then NaN, Infinity, and -Infinity will be encoded as such. This behavior is not JSON specification compliant, but is consistent with most JavaScript based encoders and decoders. Otherwise, it will be a ValueError to encode such floats.

If sort_keys is true, then the output of dictionaries will be sorted by key; this is useful for regression tests to ensure that JSON serializations can be compared on a day-to-day basis.

If indent is a non-negative integer, then JSON array elements and object members will be pretty-printed with that indent level. An indent level of 0 will only insert newlines. None is the most compact representation.

If specified, separators should be an (item_separator, key_separator) tuple. The default is (', ', ': ') if *indent* is `None` and (',', ': ') otherwise. To get the most compact JSON representation, you should specify (',', ':') to eliminate whitespace.

If specified, default is a function that gets called for objects that can't otherwise be serialized. It should return a JSON encodable version of the object or raise a `TypeError`.

default(*obj: Any*) → Any

Implement this method in a subclass such that it returns a serializable object for `o`, or calls the base implementation (to raise a `TypeError`).

For example, to support arbitrary iterators, you could implement default like this:

```
def default(self, o):
    try:
        iterable = iter(o)
    except TypeError:
        pass
    else:
        return list(iterable)
    # Let the base class default method raise the TypeError
    return super().default(o)
```

write(*obj: Any*, *, *fp: io.IO[str]*, ***kwargs*) → None

write(*obj: Any*, *, *path: conda.common.path.PathType*, ***kwargs*) → None

write(*obj: Any*, ***kwargs*) → str

dump(*obj: Any*, *fp: io.IO[str]*, ***kwargs*) → None

dumps(*obj: Any*, ***kwargs*) → str

read(*, *text: str*, ***kwargs*) → Any

read(*, *fp: io.IO[str]*, ***kwargs*) → Any

read(*, *path: conda.common.path.PathType*, ***kwargs*) → Any

load(*fp: io.IO[str]*, ***kwargs*) → Any

loads(*s: str*, ***kwargs*) → Any

JSONDecodeError

yaml

YAML serialization utilities for conda.

Functions

_representer() → type[ruamel.yaml.representer.Round	
_yaml() → ruamel.yaml.YAML	Create a parser with fresh state for each load or dump.
write(…)	
dump(obj, fp) → None	

continues on next page

Table 126 – continued from previous page

dumps(obj) → str	
read(...)	
load(fp) → Any	
loads(s) → Any	
__getattr__(name) → Any	

_representer() → *type*[ruamel.yaml.representer.RoundTripRepresenter]

_yaml() → ruamel.yaml.YAML

Create a parser with fresh state for each load or dump.

write(*obj: Any*) → str

write(*obj: Any*, *, *fp: io.IO[str]*) → None

write(*obj: Any*, *, *path: conda.common.path.PathType*) → None

dump(*obj: Any*, *fp: io.IO[str]*) → None

dumps(*obj: Any*) → str

read(*, *text: str*) → Any

read(*, *fp: io.IO[str]*) → Any

read(*, *path: conda.common.path.PathType*) → Any

load(*fp: io.IO[str]*) → Any

loads(*s: str*) → Any

__getattr__(*name: str*) → Any

Functions

_yaml_round_trip()	
_yaml_safe()	
yaml_round_trip_load(string)	
yaml_safe_load(string)	
yaml_round_trip_dump(object[, stream])	Dump object to string or stream.
yaml_safe_dump(object[, stream])	Dump object to string or stream.

Attributes

deprecated

deprecated

_yaml_round_trip()

_yaml_safe()

yaml_round_trip_load(*string*)

yaml_safe_load(*string*)

Examples

```
>>> yaml_safe_load("key: value")
{'key': 'value'}
```

yaml_round_trip_dump(*object*, *stream=None*)

Dump object to string or stream.

yaml_safe_dump(*object*, *stream=None*)

Dump object to string or stream.

signals

Intercept signals and handle them gracefully.

Functions

get_signal_name(signum)	
signal_handler(handler)	

Attributes

INTERRUPT_SIGNALS	

INTERRUPT_SIGNALS = ('SIGABRT', 'SIGINT', 'SIGTERM', 'SIGQUIT', 'SIGBREAK')

get_signal_name(*signum*)

Examples

```
>>> from signal import SIGINT
>>> get_signal_name(SIGINT)
'SIGINT'
```

signal_handler(*handler*)

terminal

Utility functions for terminal output.

Functions

_stream_is_tty(stream) → bool	Return True if the given stream object is connected to a TTY.
is_tty() → bool	Return True if stdout is connected to a TTY.
is_stdin_tty() → bool	Return True if stdin is connected to a TTY.
is_stderr_tty() → bool	Return True if stderr is connected to a TTY.
is_interactive_tty() → bool	Return True if both stdout and stdin are connected to a TTY.

continues on next page

Table 131 – continued from previous page

term_dumb() → bool	Return True when TERM indicates a non-capable terminal.
no_color() → bool	Return True when *color* output should be suppressed.
force_color() → bool	Return True when color output should be forced even in non-TTY contexts.
should_use_color() → bool	Determine whether ANSI color output should be produced.

_stream_is_tty(*stream*) → bool

Return True if the given stream object is connected to a TTY.

is_tty() → bool

Return True if stdout is connected to a TTY.

is_stdin_tty() → bool

Return True if stdin is connected to a TTY.

is_stderr_tty() → bool

Return True if stderr is connected to a TTY.

is_interactive_tty() → bool

Return True if both stdout and stdin are connected to a TTY.

Use this before showing an interactive prompt that reads from stdin, so the prompt is skipped (instead of hanging or erroring) when either stream has been redirected, e.g. `some-command < /dev/null`.

term_dumb() → bool

Return True when TERM indicates a non-capable terminal.

TERM=dumb is the widely-followed terminfo convention for terminals that cannot render any ANSI escape sequences (color, bold, animations). TERM=unknown is treated the same way as a defensive fallback.

no_color() → bool

Return True when *color* output should be suppressed.

Respects the NO_COLOR standard (https://no-color.org/): color is suppressed when the NO_COLOR environment variable is set to any value. Non-color ANSI formatting (bold, underline, etc.) may still be used.

force_color() → bool

Return True when color output should be forced even in non-TTY contexts.

Respects the FORCE_COLOR environment variable when it is set to a non-empty value.

should_use_color() → bool

Determine whether ANSI color output should be produced.

The precedence order (highest to lowest) is:

1. NO_COLOR set -> no color.
2. non-empty FORCE_COLOR set -> color even in non-TTY.
3. TERM=dumb/TERM=unknown -> no color.
4. stdout is a TTY -> color.
5. Otherwise -> no color.

toposort

Topological sorting implementation.

Functions

_toposort(data) → collections.abc.Iterator[T]	Dependencies are expressed as a dictionary whose keys are items
pop_key(data) → T	Pop an item from the graph that has the fewest dependencies in the case of a tie
_safe_toposort(data) → collections.abc.Iterator[T]	Dependencies are expressed as a dictionary whose keys are items
toposort(data[, safe]) → list[T]	Return a topologically sorted list of items from a dependency graph.

Attributes

T

T

_toposort(*data: dict[T, set[T]]*) → collections.abc.Iterator[*T*]

Dependencies are expressed as a dictionary whose keys are items and whose values are a set of dependent items. Output is a list of sets in topological order. The first set consists of items with no dependences, each subsequent set consists of items that depend upon items in the preceding sets.

pop_key(*data: dict[T, set[T]]*) → *T*

Pop an item from the graph that has the fewest dependencies in the case of a tie The winners will be sorted alphabetically

_safe_toposort(*data: dict[T, set[T]]*) → collections.abc.Iterator[*T*]

Dependencies are expressed as a dictionary whose keys are items and whose values are a set of dependent items. Output is a list of sets in topological order. The first set consists of items with no dependencies, each subsequent set consists of items that depend upon items in the preceding sets.

toposort(*data: collections.abc.Mapping[T, collections.abc.Iterable[T]]*, *safe: bool = True*) → list[*T*]

Return a topologically sorted list of items from a dependency graph.

Dependencies are expressed as a mapping whose keys are items and whose values are an iterable of dependent items.

url

Common URL utilities.

Classes

Url	Object used to represent a Url. The string representation of this object is a url string.
_SplitUrlParts	

Functions

percent_decode(path) → str	
path_to_url(path) → str	
urlparse(url) → Url	
url_to_s3_info(url) → tuple[str, str]	Convert an s3 url to a tuple of bucket and key.
is_url(url) → bool	
is_ipv4_address(string_ip) → bool	
is_ipv6_address(string_ip) → bool	
is_ip_address(string_ip) → bool	
join(*args)	
has_scheme(value) → bool	
strip_scheme(url) → str	
mask_anaconda_token(url) → str	
split_anaconda_token(url)	
split_platform(known_subdirs, url, *[, package_extensions]) → tuple[str, str | None]	Remove a trailing platform subdir, optionally followed by a known filename.
_split_platform_re(known_subdirs) → re.Pattern[str]	
has_platform(url, known_subdirs) → bool | None	
split_scheme_auth_token(url) → tuple[str | None, str | None, str | None, str | None]	
split_conda_url_easy_parts(known_subdirs, url) → _SplitUrlParts	
get_proxy_username_and_pass(scheme) → tuple[str, str]	
add_username_and_password(url, username, password) → str	Inserts *username* and *password* into provided *url*
maybe_add_auth(url, auth[, force]) → str	Add auth if the url doesn't currently have it.
maybe_unquote(url) → str	
remove_auth(url) → str	Remove embedded authentication from URL.

Attributes

file_scheme	def url_to_path(url):
url_attrs	
join_url	

percent_decode(*path: str*) → str

file_scheme = 'file://'

def url_to_path(url): assert url.startswith(file_scheme), "{} is not a file-scheme URL".format(url) decoded = percent_decode(url[len(file_scheme):]) if decoded.startswith('/') and decoded[2] == ':':

A Windows path. decoded.replace('/', '')

return decoded

path_to_url(*path: str*) → str

url_attrs = ('scheme', 'path', 'query', 'fragment', 'username', 'password', 'hostname', 'port')

class Url

Bases: namedtuple('Url', *url_attrs*)

Object used to represent a Url. The string representation of this object is a url string.

This object was inspired by the urllib3 implementation as it gives you a way to construct URLs from various parts. The motivation behind this object was making something that is interoperable with built the *urllib.parse.urlparse* function and has more features than the built-in *ParseResult* object.

Initialize self. See help(type(self)) for accurate signature.

property auth: str | None

property netloc: str | None

__str__() → str

Return str(self).

as_dict() → dict[str, str | None]

Provide a public interface for namedtuple's _asdict

replace(**kwargs*) → Self

Provide a public interface for namedtuple's _replace

classmethod from_parse_result(*parse_result: urllib.parse.ParseResult*) → Self

urlparse(*url: str*) → *Url*

url_to_s3_info(*url: str*) → tuple[str, str]

Convert an s3 url to a tuple of bucket and key.

Examples

```
>>> url_to_s3_info("s3://bucket-name.bucket/here/is/the/key")
('bucket-name.bucket', '/here/is/the/key')
```

is_url(*url: Any*) → bool

Examples

```
>>> is_url(None)
False
>>> is_url("s3://some/bucket")
True
```

is_ipv4_address(*string_ip: str*) → bool

Examples

```
>>> [is_ipv4_address(ip) for ip in ('8.8.8.8', '192.168.10.10', '255.255.255.255')]
[True, True, True]
>>> [is_ipv4_address(ip) for ip in ('8.8.8', '192.168.10.10.20', '256.255.255.255',
↪'::1')]
[False, False, False, False]
```

is_ipv6_address(*string_ip: str*) → bool

Examples

>> [is_ipv6_address(ip) for ip in ('::1', '2001:db8:85a3::370:7334', '1234:'*7+'1234')] [True, True, True] >> [is_ipv6_address(ip) for ip in ('192.168.10.10', '1234:'*8+'1234')] [False, False]

is_ip_address(*string_ip: str*) → bool

Examples

>> is_ip_address('192.168.10.10') True >> is_ip_address('::1') True >> is_ip_address('www.google.com') False

join(**args: str*)

join_url

has_scheme(*value: str*) → bool

strip_scheme(*url: str*) → str

Examples

```
>>> strip_scheme("https://www.conda.io")
'www.conda.io'
>>> strip_scheme("s3://some.bucket/plus/a/path.ext")
'some.bucket/plus/a/path.ext'
```

mask_anaconda_token(*url: str*) → str

split_anaconda_token(*url: str*)

Examples

```
>>> split_anaconda_token("https://1.2.3.4/t/tk-123-456/path")
(u'https://1.2.3.4/path', u'tk-123-456')
>>> split_anaconda_token("https://1.2.3.4/t//path")
(u'https://1.2.3.4/path', u'')
>>> split_anaconda_token("https://some.domain/api/t/tk-123-456/path")
(u'https://some.domain/api/path', u'tk-123-456')
>>> split_anaconda_token("https://1.2.3.4/conda/t/tk-123-456/path")
(u'https://1.2.3.4/conda/path', u'tk-123-456')
>>> split_anaconda_token("https://1.2.3.4/path")
(u'https://1.2.3.4/path', None)
>>> split_anaconda_token("https://10.2.3.4:8080/conda/t/tk-123-45")
(u'https://10.2.3.4:8080/conda', u'tk-123-45')
```

split_platform(*known_subdirs: collections.abc.Iterable[str]*, *url: str*, ***, *package_extensions: collections.abc.Iterable[str] | None = None*) → tuple[str, str | None]

Remove a trailing platform subdir, optionally followed by a known filename.

Parameters

- **known_subdirs** -- Recognized platform subdirectory names.
- **url** -- Channel or package URL to split.
- **package_extensions** -- Registered lowercase package extensions to reuse instead of discovering them through plugins.

Examples

```
>>> from conda.base.constants import KNOWN_SUBDIRS
>>> split_platform(KNOWN_SUBDIRS, "https://1.2.3.4/t/tk-123/path/linux-ppc64le")
(u'https://1.2.3.4/t/tk-123/path', u'linux-ppc64le')
```

_split_platform_re(*known_subdirs: collections.abc.Iterable[str]*) → re.Pattern[str]

has_platform(*url: str*, *known_subdirs: collections.abc.Iterable[str]*) → bool | None

split_scheme_auth_token(*url: str*) → tuple[str | None, str | None, str | None, str | None]

Examples

```
>>> split_scheme_auth_token("https://u:p@conda.io/t/x1029384756/more/path")
('conda.io/more/path', 'https', 'u:p', 'x1029384756')
>>> split_scheme_auth_token(None)
(None, None, None, None)
```

class _SplitUrlParts

Bases: NamedTuple

scheme: str | None

auth: str | None

token: str | None

platform: str | None

package_filename: str | None

hostname: str | None

port: str | None

path: str | None

query: str | None

split_conda_url_easy_parts(*known_subdirs: collections.abc.Iterable[str]*, *url: str*) → *_SplitUrlParts*

get_proxy_username_and_pass(*scheme: str*) → tuple[str, str]

add_username_and_password(*url: str*, *username: str*, *password: str*) → str

Inserts *username* and *password* into provided *url*

```
>>> add_username_and_password('https://anaconda.org', 'TestUser', 'Password')
'https://TestUser:Password@anaconda.org'
```

maybe_add_auth(*url: str*, *auth: str*, *force: bool = False*) → str

Add auth if the url doesn't currently have it.

By default, does not replace auth if it already exists. Setting `force` to `True` overrides this behavior.

Examples

```
>>> maybe_add_auth("https://www.conda.io", "user:passwd")
'https://user:passwd@www.conda.io'
>>> maybe_add_auth("https://www.conda.io", "")
'https://www.conda.io'
```

maybe_unquote(*url: str*) → str

remove_auth(*url: str*) → str

Remove embedded authentication from URL.

```
>>> remove_auth("https://user:password@anaconda.com")
'https://anaconda.com'
```

core

Code in `conda.core` is the core logic. It is strictly forbidden from having side effects. No printing to stdout or stderr, no disk manipulation, no http requests. All side effects should be implemented through `conda.gateways`. Objects defined in `conda.models` should be heavily preferred for `conda.core` function/method arguments and return values.

Conda modules importable from `conda.core` are

- `conda.common`
- `conda.core`
- `conda.models`
- `conda.gateways`

Conda modules strictly off limits for import within `conda.core` are

- `conda.api`
- `conda.cli`
- `conda.client`

envs_manager

Tools for managing conda environments.

Functions

get_user_environments_txt_file([userhome]) → str	Gets the path to the user's environments.txt file.
register_env(location) → None	Registers an environment by adding it to environments.txt file.
unregister_env(location) → None	Unregisters an environment by removing its entry from the environments.txt file if certain conditions are met.
list_all_known_prefixes() → list[str]	Lists all known conda environment prefixes.
query_all_prefixes(spec) → collections.abc.Iterator[tuple[str, tuple]]	Queries all known prefixes for a given specification.
_clean_environments_txt(environments_txt_file[, ...]) → tuple[str, ...]	Cleans the environments.txt file by removing specified locations.

continues on next page

Table 137 – continued from previous page

_rewrite_environments_txt(environments_txt_file, prefixes) → None	Rewrites the environments.txt file with the specified prefixes.

get_user_environments_txt_file(*userhome: str = '~'*) → str

Gets the path to the user's environments.txt file.

Parameters
userhome -- The home directory of the user.

Returns
Path to the environments.txt file.

register_env(*location: str*) → None

Registers an environment by adding it to environments.txt file.

Parameters
location -- The file path of the environment to register.

unregister_env(*location: str*) → None

Unregisters an environment by removing its entry from the environments.txt file if certain conditions are met.

The environment is only unregistered if its associated 'conda-meta' directory exists and contains no significant files other than 'history'. If these conditions are met, the environment's path is removed from environments.txt.

Parameters
location -- The file path of the environment to unregister.

list_all_known_prefixes() → list[str]

Lists all known conda environment prefixes.

Returns
A list of all known conda environment prefixes.

query_all_prefixes(*spec: str*) → collections.abc.Iterator[tuple[str, tuple]]

Queries all known prefixes for a given specification.

Parameters
spec -- The specification to query for.

Returns
An iterator of tuples containing the prefix and the query results.

_clean_environments_txt(*environments_txt_file: str*, *remove_location: str | None = None*) → tuple[str, ...]

Cleans the environments.txt file by removing specified locations.

Parameters

- **environments_txt_file** -- The file path of environments.txt.
- **remove_location** -- Optional location to remove from the file.

Returns
A tuple of the cleaned lines.

_rewrite_environments_txt(*environments_txt_file: str*, *prefixes: list[str]*) → None

Rewrites the environments.txt file with the specified prefixes.

Parameters

- **environments_txt_file** -- The file path of environments.txt.
- **prefixes** -- List of prefixes to write into the file.

exclude_newer

Helpers for applying the `exclude_newer` package policy.

Classes

_CutoffParser	Resolve user-provided cutoff values into absolute POSIX timestamps.
_ChannelSelector	Resolved `channel_settings` channel selector.
_ChannelCutoff	Resolved channel-specific `exclude_newer` cutoff.
ExcludeNewerPolicy	Resolved `exclude_newer` configuration.

Functions

_get_timestamp(record) → float | None
_get_value(record, key) → Any

class _CutoffParser

Resolve user-provided cutoff values into absolute POSIX timestamps.

now: `float`

static is_false_override(*value: object*) → bool

optional_cutoff(*value: str | int | float | bool | None*) → float | None

cutoff(*value: str | int | float | bool*) → float

duration_cutoff(*duration_seconds: float*, *value: object*) → float

static invalid(*value: str*) → *conda.exceptions.CondaValueError*

class _ChannelSelector

Resolved `channel_settings` channel selector.

URL_GLOB_CHARS

keys: `frozenset[str]`

url_pattern: `tuple[str, str] | None`

classmethod from_value(*value: str*) → *_ChannelSelector*

matches(*record:* conda.models.records.PackageRecord | *collections.abc.Mapping[str, Any]*) → bool

classmethod keys_for_record(*record:* conda.models.records.PackageRecord | *collections.abc.Mapping[str, Any]*) → frozenset[str]

classmethod keys_for(*value: object*) → frozenset[str]

static channel_from(*value: object*)

classmethod url_match_parts(*value: object*) → tuple[str, str] | None

static normalize(*value: object*) → str | None

class _ChannelCutoff

Resolved channel-specific `exclude_newer` cutoff.

SETTING_KEYS = ('exclude_newer', 'exclude-newer')

selector: *_ChannelSelector*

cutoff: float | None

classmethod from_settings(*settings: collections.abc.Mapping[str, Any]*, *parser:* _CutoffParser) → *_ChannelCutoff* | None

matches(*record:* conda.models.records.PackageRecord | *collections.abc.Mapping[str, Any]*) → bool

class ExcludeNewerPolicy

Resolved `exclude_newer` configuration.

Cutoffs are stored as absolute POSIX timestamps. A package record is included when its `indexed_timestamp` or `timestamp` is missing or older than the effective cutoff for that package name.

global_cutoff: float | None = None

channel_cutoffs: tuple[*_ChannelCutoff*, ...] = ()

package_cutoffs: dict[str, float | None] | None = None

now: float = 0.0

classmethod from_values(*exclude_newer: str | int | float | None*, *exclude_newer_package: collections.abc.Mapping[str, str | bool | int | float | None] | None*, *channel_settings: collections.abc.Iterable[collections.abc.Mapping[str, Any]] | None = None*, *now: float | None = None*) → *ExcludeNewerPolicy*

property active: bool

property has_global_cutoff: bool

property has_package_overrides: bool

property has_channel_overrides: bool

_cutoff_for_record(*record:* conda.models.records.PackageRecord | *collections.abc.Mapping[str, Any]*) → float | None

should_include(*record:* conda.models.records.PackageRecord | *collections.abc.Mapping[str, Any]*) → bool

filter_records(*records: collections.abc.Iterable[*conda.models.records.PackageRecord*]*) → tuple[*conda.models.records.PackageRecord*, ...]

excluded_records(*records: collections.abc.Iterable[*conda.models.records.PackageRecord*]*) → tuple[*conda.models.records.PackageRecord*, ...]

_get_timestamp(*record:* conda.models.records.PackageRecord | *collections.abc.Mapping[str, Any]*) → float | None

_get_value(*record:* conda.models.records.PackageRecord | *collections.abc.Mapping[str, Any]*, *key: str*) → Any

index

Tools for fetching the current index.

Classes

Index	The Index provides information about available packages from all relevant sources.
ReducedIndex	Index that contains a subset of available packages.

Functions

dist_str_in_index(index, dist_str) → bool	Check if a distribution string matches any package in the index.
get_archspec_name() → str | None	Determine the architecture specification name for the current environment.

Attributes

LAST_CHANNEL_URLS

LAST_CHANNEL_URLS = []

class Index(*channels: collections.abc.Iterable[str | conda.models.channel.Channel] = ()*, *prepend: bool = True*, *platform: str | None = None*, *subdirs: tuple[str, ...] | None = None*, *use_local: bool = False*, *use_cache: bool | None = None*, *prefix: conda.common.path.PathType | conda.core.prefix_data.PrefixData | None = None*, *repodata_fn: str | None = context.repodata_fns[-1]*, *use_system: bool = False*, *exclude_newer_policy: conda.core.exclude_newer.ExcludeNewerPolicy | None = None*)

Bases: collections.UserDict

The Index provides information about available packages from all relevant sources.

There are four types of sources for package information, namely

Channels
: represent packages available from standard sources identified with a url, mostly online, but can also be on a local filesystem using the file:// scheme. Programatically, channels are represented by *conda.models.channel.Channel*, their data is fetched using *conda.core.subdir_data.SubdirData*.

  For more information see *What is a "channel"?*.

  Individual packages from channels are usually represented by *conda.models.records.PackageRecord*.

Prefix
: represents packages that are already installed. Every *Index* can be associated with exactly one Prefix, which is the location of one of the conda *Environments*. The package information about the installed packages is represented by *conda.core.prefix_data.PrefixData*.

  Individual packages from prefixes are usually represented by *conda.models.records.PrefixRecord*.

Package Cache
: represents packages that are locally unpacked, but may not be installed in the environment associated with

this index. These are usually packages that have been installed in any environment of the local conda installation, but may have been removed from all environments by now.

Individual packages from the package are usually represented by *conda.models.records.PackageCacheRecord*.

Virtual Packages
represent properties of the system, not actual conda packages in the normal sense. These are, for example, system packages that inform the solver about the operating system in use, or track features that can be used to steer package priority.

Individual virtual packages are represented by special *conda.models.records.PackageRecord*, see *conda.models.records.PackageRecord.virtual_package()* and *conda.models.records.PackageRecord.feature()*.

Initializes a new index with the desired components.

Parameters

- **channels** -- channels identified by canonical names or URLS or Channel objects; for more details, see *conda.models.channel.Channel.from_value()*
- **prepend** -- if `True` (default), add configured channel with higher priority than passed channels; if `False`, do *not* add configured channels.
- **platform** -- see `subdirs`.
- **subdirs** -- platform and subdirs determine the selection of subdirs in the channels; if both are `None`, subdirs is taken from the configuration; if both are given, `subdirs` takes precedence and `platform` is ignored; if only `platform` is given, subdirs will be `(platform, "noarch")`; if `subdirs` is given, subdirs will be `subdirs`.
- **use_local** -- if `True`, add the special "local" channel for locally built packages with lowest priority.
- **use_cache** -- if `True`, add packages from the package cache.
- **prefix** -- associate prefix with this index and add its packages.
- **repodata_fn** -- filename of the repodata, default taken from config, almost always "repodata.json".
- **use_system** -- if `True`, add system packages, that is virtual packages defined by plugins, usually used to make intrinsic information about the system, such as cpu architecture or operating system, available to the solver.

_channels

_subdirs = None

_repodata_fn

channels: dict[str | *conda.models.channel.Channel*, list[*conda.core.subdir_data.SubdirData*]]

expanded_channels: tuple[*conda.models.channel.Channel*, ...] = ()

use_cache = True

use_system = False

exclude_newer_policy

property cache_entries: tuple[*conda.models.records.PackageCacheRecord*, ...]

Contents of the package cache if active.

Returns
All packages available from the package cache.

property system_packages: dict[*conda.models.records.PackageRecord*, *conda.models.records.PackageRecord*]

System packages provided by plugins.

Returns
Identity mapping of the available system packages in a `dict`.

property features: dict[*conda.models.records.PackageRecord*, *conda.models.records.PackageRecord*]

Active tracking features.

Returns
Identity mapping of the local tracking features in a `dict`.

reload(*, *prefix: bool = False*, *cache: bool = False*, *features: bool = False*, *system: bool = False*) → None

Reload one or more of the index components.

Can be used to refresh the index with new information, for example after a new package has been installed into the index.

Parameters

- **prefix** -- if `True`, reload the prefix data.
- **cache** -- if `True`, reload the package cache.
- **features** -- if `True`, reload the tracking features.
- **system** -- if `True`, reload the system packages.

__repr__() → str

get_reduced_index(*specs: collections.abc.Iterable*[conda.models.match_spec.MatchSpec]) → *ReducedIndex*

Create a reduced index with a subset of packages.

Can be used to create a reduced index as a subset from an existing index.

Parameters
specs -- the specs that span the subset.

Returns
a reduced index with the same sources as this index, but limited to `specs` and their dependency graph.

property data: dict[*conda.models.records.PackageRecord*, *conda.models.records.PackageRecord*]

The entire index as a dict; avoid if possible.

> **Warning**
>
> This returns the entire contents of the index as a single identity mapping in a `dict`. This may be convenient, but it comes at a cost because all sources must be fully loaded at significant overhead for *PackageRecord* construction for **every** package.

Hence, all uses of `data`, including all iteration over the entire index, is strongly discouraged.

_supplement_index_dict_with_prefix() → None

Supplement the index with information from its prefix.

_supplement_index_dict_with_cache() → None

_realize() → None

_retrieve_from_channels(*key:* conda.models.records.PackageRecord) → *conda.models.records.PackageRecord* | None

_retrieve_all_from_channels(*key:* conda.models.records.PackageRecord) → list[*conda.models.records.PackageRecord*]

_update_from_prefix(*key:* conda.models.records.PackageRecord, *prec:* conda.models.records.PackageRecord | *None*) → *conda.models.records.PackageRecord* | None

_update_from_cache(*key:* conda.models.records.PackageRecord, *prec:* conda.models.records.PackageRecord | *None*) → *conda.models.records.PackageRecord* | None

__getitem__(*key:* conda.models.records.PackageRecord) → *conda.models.records.PackageRecord*

__contains__(*key:* conda.models.records.PackageRecord) → bool

__copy__() → Self

copy() → Self

Lazy shallow copy that does not realize unrealized package records.

Overrides `UserDict.copy`, which reads `self.data` and calls `update` and therefore forces `Index._realize()` — constructing a PackageRecord for every package of every channel — even though `__copy__` preserves unrealized state. See conda/conda-build#4961.

class ReducedIndex(*specs:* *collections.abc.Iterable*[conda.models.match_spec.MatchSpec], *channels:* *tuple*[*str*, *...*] = (), *prepend:* *bool* = *True*, *platform:* *str* | *None* = *None*, *subdirs:* *tuple*[*str*, *...*] | *None* = *None*, *use_local:* *bool* = *False*, *use_cache:* *bool* | *None* = *None*, *prefix:* *conda.common.path.PathType* | conda.core.prefix_data.PrefixData | *None* = *None*, *repodata_fn:* *str* | *None* = *context.repodata_fns[-1]*, *use_system:* *bool* = *False*, *exclude_newer_policy:* conda.core.exclude_newer.ExcludeNewerPolicy | *None* = *None*)

Bases: *Index*

Index that contains a subset of available packages.

Like *Index*, this makes information about packages from the same four sources available. However, the contents of the reduced index is limited to a subset of packages relevant to a given specification. This works by taking into account all packages that match the given specification together with their dependencies and their dependencies dependencies, etc.

> **Note**
>
> See *Index.get_reduced_index()* for convenient construction.

Initialize a new reduced index.

Parameters

- **specs** -- the collection of specifications that span the subset of packages.
- **args** (*all other*) -- see *Index*.

specs

__repr__() → str

_derive_reduced_index() → None

dist_str_in_index(*index: dict[Any, Any]*, *dist_str: str*) → bool

Check if a distribution string matches any package in the index.

Parameters

- **index** -- The package index.
- **dist_str** -- The distribution string to match against the index.

Returns

True if there is a match; False otherwise.

get_archspec_name() → str | None

Determine the architecture specification name for the current environment.

Returns

The architecture name if available, otherwise None.

initialize

Backend logic for *conda init*.

Sections in this module are

1. top-level functions
2. plan creators
3. plan runners
4. individual operations
5. helper functions

The top-level functions compose and execute full plans.

A plan is created by composing various individual operations. The plan data structure is a list of dicts, where each dict represents an individual operation. The dict contains two keys--*function* and *kwargs*--where function is the name of the individual operation function within this module.

Each individual operation must

a) return a *Result* (i.e. NEEDS_SUDO, MODIFIED, or NO_CHANGE)

b) have no side effects if context.dry_run is True

c) be verbose and descriptive about the changes being made or proposed is context.verbose

The plan runner functions take the plan (list of dicts) as an argument, and then coordinate the execution of each individual operation. The docstring for *run_plan_elevated()* has details on how that strategy is implemented.

Classes

Result	

Functions

install(conda_prefix)	
initialize(conda_prefix, shells, for_user, for_system, ...)	
initialize_dev(shell[, dev_env_prefix, conda_source_root])	
_initialize_dev_bash(prefix, env_vars, unset_env_vars)	
_initialize_dev_cmdexe(prefix, env_vars, unset_env_vars)	
add_condabin_to_path(conda_prefix, shells, for_user, ...)	
make_install_plan(conda_prefix)	
make_initialize_plan(conda_prefix, shells, for_user, ...)	Creates a plan for initializing conda in shells.
make_condabin_plan(conda_prefix, shells, for_user, ...)	Creates a plan to add $PREFIX/condabin to $PATH.
run_plan(plan)	
run_plan_elevated(plan)	The strategy of this function differs between unix and Windows. Both strategies use a
run_plan_from_stdin()	
run_plan_from_temp_file(temp_path)	
print_plan_results(plan[, stream])	
make_entry_point(target_path, conda_prefix, module, func)	
make_entry_point_exe(target_path, conda_prefix)	
install_anaconda_prompt(target_path, conda_prefix, reverse)	
_install_file(target_path, file_content)	
install_conda_sh(target_path, conda_prefix)	
install_Scripts_activate_bat(target_path, conda_prefix)	
install_activate_bat(target_path, conda_prefix)	
install_deactivate_bat(target_path, conda_prefix)	
install_activate(target_path, conda_prefix)	
install_deactivate(target_path, conda_prefix)	
install_condabin_conda_bat(target_path, conda_prefix)	
install_library_bin_conda_bat(target_path, conda_prefix)	

continues on next page

Table 144 – continued from previous page

install_condabin_conda_activate_bat(target_patl ...)	
install_condabin_rename_tmp_bat(target_path, conda_prefix)	
install_condabin_conda_auto_activate_bat(targ ...)	
install_condabin_hook_bat(target_path, conda_prefix)	
install_conda_fish(target_path, conda_prefix)	
install_conda_psm1(target_path, conda_prefix)	
install_conda_hook_ps1(target_path, conda_prefix)	
install_conda_xsh(target_path, conda_prefix)	
install_conda_csh(target_path, conda_prefix)	
_config_fish_content(conda_prefix)	
_config_fish_content_to_add_condabin_to_patl	
init_fish_user(target_path, conda_prefix[, reverse, ...])	
_config_xonsh_content(conda_prefix)	
_config_xonsh_content_to_add_condabin_to_pat	
init_xonsh_user(target_path, conda_prefix[, reverse, ...])	
_bashrc_content(conda_prefix, shell)	
_bashrc_content_to_add_condabin_to_path(cond ...)	
init_sh_user(target_path, conda_prefix, shell[, ...])	
init_sh_system(target_path, conda_prefix[, reverse, ...])	
_read_windows_registry(target_path)	
_write_windows_registry(target_path, value_value, ...)	
init_cmd_exe_registry(target_path, conda_prefix[, reverse])	
add_condabin_to_path_registry(target_path, conda_prefix)	
init_long_path(target_path)	
_powershell_profile_content(conda_prefix)	
*_powershell_profile_content_to_add_condabin_*	
init_powershell_user(target_path, conda_prefix[, ...])	
remove_conda_in_sp_dir(target_path)	
make_conda_egg_link(target_path, conda_source_root)	
modify_easy_install_pth(target_path, conda_source_root)	
make_dev_egg_info_file(target_path)	
make_diff(old, new)	
_get_python_info(prefix)	

Attributes

CONDA_INITIALIZE_RE_BLOCK
CONDA_INITIALIZE_PS_RE_BLOCK
ContentTypeOptions

CONDA_INITIALIZE_RE_BLOCK = '^# >>> conda initialize >>>(?:\\n|\\r\\n)([\\s\\S]*?)# <<< conda initialize <<<(?:\\n|\\r\\n)?'

CONDA_INITIALIZE_PS_RE_BLOCK = '^#region conda initialize(?:\\n|\\r\\n)([\\s\\S]*?)#endregion(?:\\n|\\r\\n)?'

class Result

NEEDS_SUDO = 'needs sudo'

MODIFIED = 'modified'

NO_CHANGE = 'no change'

ContentTypeOptions

install(*conda_prefix*)

initialize(*conda_prefix*, *shells*, *for_user*, *for_system*, *anaconda_prompt*, *reverse=False*)

initialize_dev(*shell*, *dev_env_prefix=None*, *conda_source_root=None*)

_initialize_dev_bash(*prefix*, *env_vars*, *unset_env_vars*)

_initialize_dev_cmdexe(*prefix*, *env_vars*, *unset_env_vars*)

add_condabin_to_path(*conda_prefix*, *shells*, *for_user*, *for_system*, *reverse=False*)

make_install_plan(*conda_prefix*)

make_initialize_plan(*conda_prefix*, *shells*, *for_user*, *for_system*, *anaconda_prompt*, *reverse=False*)

Creates a plan for initializing conda in shells.

Bash: On Linux, when opening the terminal, .bashrc is sourced (because it is an interactive shell). On macOS on the other hand, the .bash_profile gets sourced by default when executing it in Terminal.app. Some other programs do the same on macOS so that's why we're initializing conda in .bash_profile. On Windows, there are multiple ways to open bash depending on how it was installed. Git Bash, Cygwin, and MSYS2 all use .bash_profile by default.

PowerShell: There's several places PowerShell can store its path, depending on if it's Windows PowerShell, PowerShell Core on Windows, or PowerShell Core on macOS/Linux. The easiest way to resolve it is to just ask different possible installations of PowerShell where their profiles are.

make_condabin_plan(*conda_prefix*, *shells*, *for_user*, *for_system*, *reverse=False*)

Creates a plan to add $PREFIX/condabin to $PATH.

run_plan(*plan*)

run_plan_elevated(*plan*)

The strategy of this function differs between unix and Windows. Both strategies use a subprocess call, where the subprocess is run with elevated privileges. The executable invoked with the subprocess is *python -m conda.core.initialize*, so see the *if __name__ == "__main__"* at the bottom of this module.

For unix platforms, we convert the plan list to json, and then call this module with *sudo python -m conda.core.initialize* while piping the plan json to stdin. We collect json from stdout for the results of the plan execution with elevated privileges.

For Windows, we create a temporary file that holds the json content of the plan. The subprocess reads the content of the file, modifies the content of the file with updated execution status, and then closes the file. This process then reads the content of that file for the individual operation execution results, and then deletes the file.

run_plan_from_stdin()

run_plan_from_temp_file(*temp_path*)

print_plan_results(*plan*, *stream=None*)

make_entry_point(*target_path*, *conda_prefix*, *module*, *func*)

make_entry_point_exe(*target_path*, *conda_prefix*)

install_anaconda_prompt(*target_path*, *conda_prefix*, *reverse*)

_install_file(*target_path*, *file_content*)

install_conda_sh(*target_path*, *conda_prefix*)

install_Scripts_activate_bat(*target_path*, *conda_prefix*)

install_activate_bat(*target_path*, *conda_prefix*)

install_deactivate_bat(*target_path*, *conda_prefix*)

install_activate(*target_path*, *conda_prefix*)

install_deactivate(*target_path*, *conda_prefix*)

install_condabin_conda_bat(*target_path*, *conda_prefix*)

install_library_bin_conda_bat(*target_path*, *conda_prefix*)

install_condabin_conda_activate_bat(*target_path*, *conda_prefix*)

install_condabin_rename_tmp_bat(*target_path*, *conda_prefix*)

install_condabin_conda_auto_activate_bat(*target_path*, *conda_prefix*)

install_condabin_hook_bat(*target_path*, *conda_prefix*)

install_conda_fish(*target_path*, *conda_prefix*)

install_conda_psm1(*target_path*, *conda_prefix*)

install_conda_hook_ps1(*target_path*, *conda_prefix*)

install_conda_xsh(*target_path*, *conda_prefix*)

install_conda_csh(*target_path*, *conda_prefix*)

_config_fish_content(*conda_prefix*)

_config_fish_content_to_add_condabin_to_path(*conda_prefix*)

init_fish_user(*target_path*, *conda_prefix*, *reverse=False*, *content_type:* ContentTypeOptions = *'initialize'*)

_config_xonsh_content(*conda_prefix*)

_config_xonsh_content_to_add_condabin_to_path(*conda_prefix*)

init_xonsh_user(*target_path*, *conda_prefix*, *reverse=False*, *content_type:* ContentTypeOptions = *'initialize'*)

_bashrc_content(*conda_prefix*, *shell*)

_bashrc_content_to_add_condabin_to_path(*conda_prefix*, *shell*)

init_sh_user(*target_path*, *conda_prefix*, *shell*, *reverse=False*, *content_type:* ContentTypeOptions = *'initialize'*)

init_sh_system(*target_path*, *conda_prefix*, *reverse=False*, *content_type:* ContentTypeOptions = *'initialize'*)

_read_windows_registry(*target_path*)

_write_windows_registry(*target_path*, *value_value*, *value_type*)

init_cmd_exe_registry(*target_path*, *conda_prefix*, *reverse=False*)

add_condabin_to_path_registry(*target_path*, *conda_prefix*, *reverse=False*)

init_long_path(*target_path*)

_powershell_profile_content(*conda_prefix*)

_powershell_profile_content_to_add_condabin_to_path(*conda_prefix*)

init_powershell_user(*target_path*, *conda_prefix*, *reverse=False*, *content_type:* ContentTypeOptions = *'initialize'*)

remove_conda_in_sp_dir(*target_path*)

make_conda_egg_link(*target_path*, *conda_source_root*)

modify_easy_install_pth(*target_path*, *conda_source_root*)

make_dev_egg_info_file(*target_path*)

make_diff(*old*, *new*)

_get_python_info(*prefix*)

launchers

Locate and verify Windows entry point launchers.

continues on next page

Table 146 – continued from previous page

Functions

get_windows_launcher_stub(target_prefix, *, ...[, ...]) → tuple[str, str]	Return the target Python's launcher path and expected SHA-256.
_find_launcher(prefix, paths_data, short_path) → tuple[str, str] | None	
verify_windows_launcher(path, sha256) → None	Verify the expected SHA-256 immediately before copying a launcher.

get_windows_launcher_stub(*target_prefix: str | os.PathLike[str]*, *, *source_prefixes: collections.abc.Iterable[str | os.PathLike[str]]*, *source_package_infos: collections.abc.Iterable[*conda.models.package_info.PackageInfo*] = ()*) → tuple[str, str]

Return the target Python's launcher path and expected SHA-256.

_find_launcher(*prefix: str | os.PathLike[str]*, *paths_data:* conda.models.records.PathsData | *None*, *short_path: str*) → tuple[str, str] | None

verify_windows_launcher(*path: str*, *sha256: str*) → None

Verify the expected SHA-256 immediately before copying a launcher.

link

Package installation implemented as a series of link/unlink transactions.

Classes

PrefixSetup	
ActionGroup	
PrefixActionGroup	A container for groups of actions carried out during an UnlinkLinkTransaction.
ChangeReport	
UnlinkLinkTransaction	

Functions

determine_link_type(extracted_package_dir, target_prefix)	
make_unlink_actions(transaction_context, ...)	
match_specs_to_dists(packages_info_to_link, specs) → tuple[list[conda.models.match_spec.MatchSpec ...]	Find which specs belong to each package to link
run_script(prefix, prec[, action, env_prefix, activate]) → bool	Call the post-link (or pre-unlink) script, returning True on success,
messages(prefix)	

determine_link_type(*extracted_package_dir*, *target_prefix*)

make_unlink_actions(*transaction_context*, *target_prefix*, *prefix_record*)

match_specs_to_dists(*packages_info_to_link*, *specs*) → tuple[list[*conda.models.match_spec.MatchSpec*], ...]

Find which specs belong to each package to link

class PrefixSetup

Bases: NamedTuple

target_prefix: str

unlink_precs: tuple[*conda.models.records.PackageRecord*, ...]

link_precs: tuple[*conda.models.records.PackageRecord*, ...]

remove_specs: tuple[*conda.models.match_spec.MatchSpec*, ...]

update_specs: tuple[*conda.models.match_spec.MatchSpec*, ...]

neutered_specs: tuple[*conda.models.match_spec.MatchSpec*, ...]

class ActionGroup

Bases: NamedTuple

type: str

pkg_data: *conda.models.package_info.PackageInfo* | None

actions: collections.abc.Iterable[*conda.core.path_actions.Action*]

target_prefix: str

class PrefixActionGroup

A container for groups of actions carried out during an UnlinkLinkTransaction.

Parameters

- **remove_menu_action_groups** -- Actions which remove menu items.
- **unlink_action_groups** -- Actions which unlink files.
- **unregister_action_groups** -- Actions which unregister environment locations.
- **link_action_groups** -- Actions which link files.
- **register_action_groups** -- Actions which register environment locations.
- **compile_action_groups** -- Actions which compile pyc files.
- **make_menu_action_groups** -- Actions which create menu items.
- **entry_point_action_groups** -- Actions which create python entry points.
- **prefix_record_groups** -- Actions which create package json files in conda-meta/.
- **initial_action_groups** -- User-defined actions which run before all other actions.
- **final_action_groups** -- User-defined actions which run after all other actions.

remove_menu_action_groups: collections.abc.Iterable[*ActionGroup*]

unlink_action_groups: collections.abc.Iterable[*ActionGroup*]

unregister_action_groups: collections.abc.Iterable[*ActionGroup*]

link_action_groups: collections.abc.Iterable[*ActionGroup*]

register_action_groups: collections.abc.Iterable[*ActionGroup*]

compile_action_groups: collections.abc.Iterable[*ActionGroup*]

make_menu_action_groups: collections.abc.Iterable[*ActionGroup*]

entry_point_action_groups: collections.abc.Iterable[*ActionGroup*]

prefix_record_groups: collections.abc.Iterable[*ActionGroup*]

initial_action_groups: collections.abc.Iterable[*ActionGroup*] = ()

final_action_groups: collections.abc.Iterable[*ActionGroup*] = ()

__iter__() → collections.abc.Generator[collections.abc.Iterable[*ActionGroup*], None, None]

class ChangeReport

Bases: NamedTuple

prefix: str

specs_to_remove: collections.abc.Iterable[*conda.models.match_spec.MatchSpec*]

specs_to_add: collections.abc.Iterable[*conda.models.match_spec.MatchSpec*]

removed_precs: collections.abc.Iterable[*conda.models.records.PackageRecord*]

new_precs: collections.abc.Iterable[*conda.models.records.PackageRecord*]

updated_precs: collections.abc.Iterable[*conda.models.records.PackageRecord*]

downgraded_precs: collections.abc.Iterable[*conda.models.records.PackageRecord*]

superseded_precs: collections.abc.Iterable[*conda.models.records.PackageRecord*]

fetch_precs: collections.abc.Iterable[*conda.models.records.PackageRecord*]

revised_precs: collections.abc.Iterable[*conda.models.records.PackageRecord*]

class UnlinkLinkTransaction(**setups*)

prefix_setups

prefix_action_groups

_pfe = None

_prepared = False

_verified = False

verify_executor

execute_executor

property nothing_to_do

download_and_extract()

prepare()

verify()

_cleanup_transaction_artifacts()

Remove the temp dir and any prefixes this transaction created.

_verify_pre_link_message(*all_link_groups*)

execute()

_get_pfe()

classmethod _prepare(*transaction_context*, *target_prefix*, *unlink_precs*, *link_precs*, *remove_specs*, *update_specs*, *neutered_specs*)

static _verify_individual_level(*prefix_action_group*)

static _verify_prefix_level(*target_prefix_AND_prefix_action_group_tuple*)

static _verify_transaction_level(*prefix_setups*)

_verify(*prefix_setups*, *prefix_action_groups*)

_execute(*all_action_groups*)

static _execute_actions(*axngroup*)

static _execute_post_link_actions(*axngroup*)

static _reverse_actions(*axngroup*, *reverse_from_idx=-1*)

static _get_python_info(*target_prefix*, *prefix_recs_to_unlink*, *packages_info_to_link*) → tuple[str | None, str | None]

Return the python version and location of the site-packages directory at the end of the transaction

static _make_link_actions(*transaction_context*, *package_info*, *target_prefix*, *requested_link_type*, *requested_spec*)

static _make_entry_point_actions(*transaction_context*, *package_info*, *target_prefix*, *requested_link_type*, *requested_spec*, *packages_info_to_link*, *link_action_groups*)

static _make_compile_actions(*transaction_context*, *package_info*, *target_prefix*, *requested_link_type*, *requested_spec*, *link_action_groups*)

_make_legacy_action_groups()

print_transaction_summary()

_change_report_str(*change_report*)

static _calculate_change_report(*prefix*, *unlink_precs*, *link_precs*, *download_urls*, *specs_to_remove*, *specs_to_add*)

run_script(*prefix: str*, *prec*, *action: str = 'post-link'*, *env_prefix: str = None*, *activate: bool = False*) → bool

Call the post-link (or pre-unlink) script, returning True on success, False on failure.

messages(*prefix*)

package_cache_data

Tools for managing the package cache (previously downloaded packages).

Classes

PackageCacheType	This metaclass does basic caching of PackageCache instance objects.
PackageCacheData	
UrlsData	
ProgressiveFetchExtract	

Functions

get_softlinked_package_dirs() → set[pathlib.Path]	Return cache directories known environments may reference through symlinks.
do_cache_action(prec, cache_action, progress_bar[, ...])	This function gets called from *ProgressiveFetchExtract.execute*.
do_extract_action(prec, extract_action, progress_bar)	This function gets called after do_cache_action completes.
do_cleanup(actions)	
do_reverse(actions)	
done_callback(future, actions, progress_bar, exceptions)	

Attributes

FileNotFoundError
THREADSAFE_EXTRACT
EXTRACT_THREADS
EXTRACT_PROCESSES
EXTRACT_PROCESS_EXTENSIONS

FileNotFoundError

THREADSAFE_EXTRACT = False

EXTRACT_THREADS

EXTRACT_PROCESSES

EXTRACT_PROCESS_EXTENSIONS

get_softlinked_package_dirs() → set[pathlib.Path]

Return cache directories known environments may reference through symlinks.

class PackageCacheType

Bases: type

This metaclass does basic caching of PackageCache instance objects.

__call__(*pkgs_dir: str | os.PathLike | pathlib.Path*)

class PackageCacheData(*pkgs_dir*)

cache: dict[str, PackageCacheData]

pkgs_dir

__package_cache_records = None

__is_writable

_urls_data

insert(*package_cache_record*)

load()

reload()

get(*package_ref:* conda.models.records.PackageRecord, *default=NULL*)

remove(*package_ref*, *default=NULL*)

query(*package_ref_or_match_spec*)

iter_records()

classmethod query_all(*package_ref_or_match_spec*, *pkgs_dirs=None*)

classmethod first_writable(*pkgs_dirs=None*)

classmethod writable_caches(*pkgs_dirs=None*)

classmethod read_only_caches(*pkgs_dirs=None*)

classmethod all_caches_writable_first(*pkgs_dirs=None*)

classmethod get_all_extracted_entries()

classmethod get_entry_to_link(*package_ref*, *target_prefix=None*)

classmethod tarball_file_in_cache(*tarball_path*, *md5sum=None*, *exclude_caches=()*)

classmethod clear()

tarball_file_in_this_cache(*tarball_path*, *md5sum=None*)

property _package_cache_records

property is_writable

_check_writable()

static _clean_tarball_path_and_get_md5sum(*tarball_path*, *md5sum=None*)

_scan_for_dist_no_channel(*dist_str*)

itervalues()

values()

__repr__()

_make_single_record(*package_filename*)

static _dedupe_pkgs_dir_contents(*pkgs_dir_contents*)

class UrlsData(*pkgs_dir*)

pkgs_dir

__contains__(*url*)

__iter__()

add_url(*url*)

get_url(*package_path*)

class ProgressiveFetchExtract(*link_prefs*)

Parameters

link_prefs (*tuple[PackageRecord]*) -- A sequence of PackageRecord`s to ensure available in a known package cache, typically for a follow-on :class:`UnlinkLinkTransaction. Here, "available" means the package tarball is both downloaded and extracted to a package directory.

static make_actions_for_record(*pref_or_spec*)

link_precs

paired_actions

_prepared = False

_executed = False

prepare()

property cache_actions

property extract_actions

execute()

Run each action in self.paired_actions. Each action in cache_actions runs before its corresponding extract_actions.

static _progress_bar(*prec_or_spec*, *position=None*, *leave=False*, *context_manager=None*) → *conda.plugins.types.ProgressBarBase*

__hash__()

__eq__(*other*)

do_cache_action(*prec*, *cache_action*, *progress_bar*, *download_total=1.0*, ***, *cancelled*)

This function gets called from *ProgressiveFetchExtract.execute*.

do_extract_action(*prec*, *extract_action*, *progress_bar*)

This function gets called after do_cache_action completes.

do_cleanup(*actions*)

do_reverse(*actions*)

done_callback(*future: concurrent.futures.Future*, *actions: tuple[*conda.core.path_actions.CacheUrlAction | conda.core.path_actions.ExtractPackageAction, ...*]*, *progress_bar:* conda.plugins.types.ProgressBarBase, *exceptions: list[Exception]*, *finish: bool = False*)

path_actions

Atomic actions that make up a package installation or removal transaction.

Classes

Action	Base class for path manipulation actions, including linking, unlinking, and others.
PathAction	Base class for path manipulation actions, including linking, unlinking, and others.
MultiPathAction	Base class for path manipulation actions, including linking, unlinking, and others.
PrefixPathAction	Base class for path manipulation actions, including linking, unlinking, and others.
CreateInPrefixPathAction	Base class for path manipulation actions, including linking, unlinking, and others.
LinkPathAction	Base class for path manipulation actions, including linking, unlinking, and others.
PrefixReplaceLinkAction	Base class for path manipulation actions, including linking, unlinking, and others.
MakeMenuAction	Base class for path manipulation actions, including linking, unlinking, and others.
CompileMultiPycAction	Base class for path manipulation actions, including linking, unlinking, and others.
AggregateCompileMultiPycAction	Bunch up all of our compile actions, so that they all get carried out at once.
CreatePythonEntryPointAction	Base class for path manipulation actions, including linking, unlinking, and others.
CreatePrefixRecordAction	Base class for path manipulation actions, including linking, unlinking, and others.
UpdateHistoryAction	Base class for path manipulation actions, including linking, unlinking, and others.
RegisterEnvironmentLocationAction	Base class for path manipulation actions, including linking, unlinking, and others.
RemoveFromPrefixPathAction	Base class for path manipulation actions, including linking, unlinking, and others.
UnlinkPathAction	Base class for path manipulation actions, including linking, unlinking, and others.
RemoveMenuAction	Base class for path manipulation actions, including linking, unlinking, and others.
RemoveLinkedPackageRecordAction	Base class for path manipulation actions, including linking, unlinking, and others.
UnregisterEnvironmentLocationAction	Base class for path manipulation actions, including linking, unlinking, and others.
CacheUrlAction	Base class for path manipulation actions, including linking, unlinking, and others.

continues on next page

Table 152 – continued from previous page

ExtractPackageAction	Base class for path manipulation actions, including linking, unlinking, and others.

Attributes

FileNotFoundError
_MENU_RE
REPR_IGNORE_KWARGS

FileNotFoundError

_MENU_RE

REPR_IGNORE_KWARGS = ('transaction_context', 'package_info', 'hold_path')

class Action(*transaction_context: dict[str, str] | None = None*, *target_prefix: str | None = None*, *unlink_precs: collections.abc.Iterable[conda.models.records.PackageRecord] | None = None*, *link_precs: collections.abc.Iterable[conda.models.records.PackageRecord] | None = None*, *remove_specs: collections.abc.Iterable[conda.models.match_spec.MatchSpec] | None = None*, *update_specs: collections.abc.Iterable[conda.models.match_spec.MatchSpec] | None = None*, *neutered_specs: collections.abc.Iterable[conda.models.match_spec.MatchSpec] | None = None*)

Bases: `abc.ABC`

Base class for path manipulation actions, including linking, unlinking, and others.

Pre and post-transaction plugins should inherit this class to implement their own verification, execution, reversing, and cleanup steps. These methods are guaranteed to be called in the following order:

1. `verify`
2. `execute`
3. `reverse` (only if `execute` raises an exception)
4. `cleanup`

Parameters

- **transaction_context** -- Mapping between target prefixes and PrefixActionGroup instances.
- **target_prefix** -- Target prefix for the action.
- **unlink_precs** -- Package records to be unlinked.
- **link_precs** -- Package records to link.
- **remove_specs** -- Specs to be removed.
- **update_specs** -- Specs to be updated.
- **neutered_specs** -- Specs to be neutered.

_verified = False

transaction_context = None

target_prefix = None

unlink_precs = None

link_precs = None

remove_specs = None

update_specs = None

neutered_specs = None

abstractmethod verify() → Exception | None

Carry out any pre-execution verification.

Should set self._verified = True upon success.

Returns

On failure, this function should return (not raise!) an exception object. At the end of the verification run, all errors will be raised as a CondaMultiError.

abstractmethod execute() → None

Execute the action.

Called after `self.verify()`. If this function raises an exception, `self.reverse()` will be called.

abstractmethod reverse() → None

Reverse what was done in execute.

Called only if `self.execute()` raises an exception.

abstractmethod cleanup() → None

Carry out any post-execution tasks.

property verified

__repr__()

class PathAction(*transaction_context: dict[str, str] | None = None*, *target_prefix: str | None = None*, *unlink_precs: collections.abc.Iterable[*conda.models.records.PackageRecord*] | None = None*, *link_precs: collections.abc.Iterable[*conda.models.records.PackageRecord*] | None = None*, *remove_specs: collections.abc.Iterable[*conda.models.match_spec.MatchSpec*] | None = None*, *update_specs: collections.abc.Iterable[*conda.models.match_spec.MatchSpec*] | None = None*, *neutered_specs: collections.abc.Iterable[*conda.models.match_spec.MatchSpec*] | None = None*)

Bases: *Action*

Base class for path manipulation actions, including linking, unlinking, and others.

Pre and post-transaction plugins should inherit this class to implement their own verification, execution, reversing, and cleanup steps. These methods are guaranteed to be called in the following order:

1. `verify`
2. `execute`
3. `reverse` (only if `execute` raises an exception)
4. `cleanup`

Parameters

- **transaction_context** -- Mapping between target prefixes and PrefixActionGroup instances.
- **target_prefix** -- Target prefix for the action.
- **unlink_precs** -- Package records to be unlinked.
- **link_precs** -- Package records to link.
- **remove_specs** -- Specs to be removed.
- **update_specs** -- Specs to be updated.
- **neutered_specs** -- Specs to be neutered.

abstract property target_full_path

class MultiPathAction(*transaction_context: dict[str, str] | None = None*, *target_prefix: str | None = None*, *unlink_precs: collections.abc.Iterable[conda.models.records.PackageRecord] | None = None*, *link_precs: collections.abc.Iterable[conda.models.records.PackageRecord] | None = None*, *remove_specs: collections.abc.Iterable[conda.models.match_spec.MatchSpec] | None = None*, *update_specs: collections.abc.Iterable[conda.models.match_spec.MatchSpec] | None = None*, *neutered_specs: collections.abc.Iterable[conda.models.match_spec.MatchSpec] | None = None*)

Bases: *Action*

Base class for path manipulation actions, including linking, unlinking, and others.

Pre and post-transaction plugins should inherit this class to implement their own verification, execution, reversing, and cleanup steps. These methods are guaranteed to be called in the following order:

1. `verify`
2. `execute`
3. `reverse` (only if `execute` raises an exception)
4. `cleanup`

Parameters

- **transaction_context** -- Mapping between target prefixes and PrefixActionGroup instances.
- **target_prefix** -- Target prefix for the action.
- **unlink_precs** -- Package records to be unlinked.
- **link_precs** -- Package records to link.
- **remove_specs** -- Specs to be removed.
- **update_specs** -- Specs to be updated.
- **neutered_specs** -- Specs to be neutered.

abstract property target_full_paths

class PrefixPathAction(*transaction_context*, *target_prefix*, *target_short_path*)

Bases: PathAction

Base class for path manipulation actions, including linking, unlinking, and others.

Pre and post-transaction plugins should inherit this class to implement their own verification, execution, reversing, and cleanup steps. These methods are guaranteed to be called in the following order:

1. verify
2. execute
3. reverse (only if execute raises an exception)
4. cleanup

Parameters

- **transaction_context** -- Mapping between target prefixes and PrefixActionGroup instances.
- **target_prefix** -- Target prefix for the action.
- **unlink_precs** -- Package records to be unlinked.
- **link_precs** -- Package records to link.
- **remove_specs** -- Specs to be removed.
- **update_specs** -- Specs to be updated.
- **neutered_specs** -- Specs to be neutered.

transaction_context

target_prefix

target_short_path

property target_short_paths

property target_full_path

class CreateInPrefixPathAction(*transaction_context*, *package_info*, *source_prefix*, *source_short_path*, *target_prefix*, *target_short_path*)

Bases: PrefixPathAction

Base class for path manipulation actions, including linking, unlinking, and others.

Pre and post-transaction plugins should inherit this class to implement their own verification, execution, reversing, and cleanup steps. These methods are guaranteed to be called in the following order:

1. verify
2. execute
3. reverse (only if execute raises an exception)
4. cleanup

Parameters

- **transaction_context** -- Mapping between target prefixes and PrefixActionGroup instances.

- **target_prefix** -- Target prefix for the action.
- **unlink_precs** -- Package records to be unlinked.
- **link_precs** -- Package records to link.
- **remove_specs** -- Specs to be removed.
- **update_specs** -- Specs to be updated.
- **neutered_specs** -- Specs to be neutered.

package_info

source_prefix

source_short_path

verify()

Carry out any pre-execution verification.

Should set self._verified = True upon success.

Returns

On failure, this function should return (not raise!) an exception object. At the end of the verification run, all errors will be raised as a CondaMultiError.

cleanup()

Carry out any post-execution tasks.

property source_full_path

class LinkPathAction(*transaction_context*, *package_info*, *extracted_package_dir*, *source_short_path*, *target_prefix*, *target_short_path*, *link_type*, *source_path_data*)

Bases: *CreateInPrefixPathAction*

Base class for path manipulation actions, including linking, unlinking, and others.

Pre and post-transaction plugins should inherit this class to implement their own verification, execution, reversing, and cleanup steps. These methods are guaranteed to be called in the following order:

1. `verify`
2. `execute`
3. `reverse` (only if `execute` raises an exception)
4. `cleanup`

Parameters

- **transaction_context** -- Mapping between target prefixes and PrefixActionGroup instances.
- **target_prefix** -- Target prefix for the action.
- **unlink_precs** -- Package records to be unlinked.
- **link_precs** -- Package records to link.
- **remove_specs** -- Specs to be removed.
- **update_specs** -- Specs to be updated.
- **neutered_specs** -- Specs to be neutered.

classmethod create_file_link_actions(*transaction_context*, *package_info*, *target_prefix*, *requested_link_type*)

classmethod create_directory_actions(*transaction_context*, *package_info*, *target_prefix*, *requested_link_type*, *file_link_actions*)

classmethod create_python_entry_point_windows_exe_action(*transaction_context*, *package_info*, *target_prefix*, *requested_link_type*, *entry_point_def*, *source_package_infos=()*)

link_type

_execute_successful = False

source_path_data

prefix_path_data = None

verify()

Carry out any pre-execution verification.

Should set self._verified = True upon success.

Returns

On failure, this function should return (not raise!) an exception object. At the end of the verification run, all errors will be raised as a CondaMultiError.

execute()

Execute the action.

Called after `self.verify()`. If this function raises an exception, `self.reverse()` will be called.

reverse()

Reverse what was done in execute.

Called only if `self.execute()` raises an exception.

class PrefixReplaceLinkAction(*transaction_context*, *package_info*, *extracted_package_dir*, *source_short_path*, *target_prefix*, *target_short_path*, *link_type*, *prefix_placeholder*, *file_mode*, *source_path_data*)

Bases: *LinkPathAction*

Base class for path manipulation actions, including linking, unlinking, and others.

Pre and post-transaction plugins should inherit this class to implement their own verification, execution, reversing, and cleanup steps. These methods are guaranteed to be called in the following order:

1. `verify`
2. `execute`
3. `reverse` (only if `execute` raises an exception)
4. `cleanup`

Parameters

- **transaction_context** -- Mapping between target prefixes and PrefixActionGroup instances.
- **target_prefix** -- Target prefix for the action.

- **unlink_precs** -- Package records to be unlinked.
- **link_precs** -- Package records to link.
- **remove_specs** -- Specs to be removed.
- **update_specs** -- Specs to be updated.
- **neutered_specs** -- Specs to be neutered.

prefix_placeholder

file_mode

intermediate_path = None

verify()

Carry out any pre-execution verification.

Should set self._verified = True upon success.

Returns

On failure, this function should return (not raise!) an exception object. At the end of the verification run, all errors will be raised as a CondaMultiError.

execute()

Execute the action.

Called after `self.verify()`. If this function raises an exception, `self.reverse()` will be called.

class MakeMenuAction(*transaction_context*, *package_info*, *target_prefix*, *target_short_path*)

Bases: *CreateInPrefixPathAction*

Base class for path manipulation actions, including linking, unlinking, and others.

Pre and post-transaction plugins should inherit this class to implement their own verification, execution, reversing, and cleanup steps. These methods are guaranteed to be called in the following order:

1. `verify`
2. `execute`
3. `reverse` (only if `execute` raises an exception)
4. `cleanup`

Parameters

- **transaction_context** -- Mapping between target prefixes and PrefixActionGroup instances.
- **target_prefix** -- Target prefix for the action.
- **unlink_precs** -- Package records to be unlinked.
- **link_precs** -- Package records to link.
- **remove_specs** -- Specs to be removed.
- **update_specs** -- Specs to be updated.
- **neutered_specs** -- Specs to be neutered.

classmethod create_actions(*transaction_context*, *package_info*, *target_prefix*, *requested_link_type*)

_execute_successful = False

execute()

Execute the action.

Called after `self.verify()`. If this function raises an exception, `self.reverse()` will be called.

reverse()

Reverse what was done in execute.

Called only if `self.execute()` raises an exception.

class CompileMultiPycAction(*transaction_context*, *package_info*, *target_prefix*, *source_short_paths*, *target_short_paths*)

Bases: *MultiPathAction*

Base class for path manipulation actions, including linking, unlinking, and others.

Pre and post-transaction plugins should inherit this class to implement their own verification, execution, reversing, and cleanup steps. These methods are guaranteed to be called in the following order:

1. `verify`
2. `execute`
3. `reverse` (only if `execute` raises an exception)
4. `cleanup`

Parameters

- **transaction_context** -- Mapping between target prefixes and PrefixActionGroup instances.
- **target_prefix** -- Target prefix for the action.
- **unlink_precs** -- Package records to be unlinked.
- **link_precs** -- Package records to link.
- **remove_specs** -- Specs to be removed.
- **update_specs** -- Specs to be updated.
- **neutered_specs** -- Specs to be neutered.

classmethod create_actions(*transaction_context*, *package_info*, *target_prefix*, *requested_link_type*, *file_link_actions*)

transaction_context

package_info

target_prefix

source_short_paths

target_short_paths

prefix_path_data = None

prefix_paths_data

_execute_successful = False

property target_full_paths

property source_full_paths

verify()

Carry out any pre-execution verification.

Should set self._verified = True upon success.

Returns

On failure, this function should return (not raise!) an exception object. At the end of the verification run, all errors will be raised as a CondaMultiError.

cleanup()

Carry out any post-execution tasks.

execute()

Execute the action.

Called after `self.verify()`. If this function raises an exception, `self.reverse()` will be called.

reverse()

Reverse what was done in execute.

Called only if `self.execute()` raises an exception.

class AggregateCompileMultiPycAction(**individuals*, ***kw*)

Bases: *CompileMultiPycAction*

Bunch up all of our compile actions, so that they all get carried out at once. This avoids clobbering and is faster when we have several individual packages requiring compilation.

_individuals = ()

execute()

Execute the action.

Called after `self.verify()`. If this function raises an exception, `self.reverse()` will be called.

class CreatePythonEntryPointAction(*transaction_context*, *package_info*, *target_prefix*, *target_short_path*, *module*, *func*)

Bases: *CreateInPrefixPathAction*

Base class for path manipulation actions, including linking, unlinking, and others.

Pre and post-transaction plugins should inherit this class to implement their own verification, execution, reversing, and cleanup steps. These methods are guaranteed to be called in the following order:

1. `verify`
2. `execute`
3. `reverse` (only if `execute` raises an exception)
4. `cleanup`

Parameters

- **transaction_context** -- Mapping between target prefixes and PrefixActionGroup instances.

- **target_prefix** -- Target prefix for the action.
- **unlink_precs** -- Package records to be unlinked.
- **link_precs** -- Package records to link.
- **remove_specs** -- Specs to be removed.
- **update_specs** -- Specs to be updated.
- **neutered_specs** -- Specs to be neutered.

classmethod create_actions(*transaction_context*, *package_info*, *target_prefix*, *requested_link_type*, *source_package_infos=()*)

module

func

prefix_path_data

_execute_successful = False

execute()

Execute the action.

Called after `self.verify()`. If this function raises an exception, `self.reverse()` will be called.

reverse()

Reverse what was done in execute.

Called only if `self.execute()` raises an exception.

class CreatePrefixRecordAction(*transaction_context*, *package_info*, *target_prefix*, *target_short_path*, *requested_link_type*, *requested_spec:* conda.models.match_spec.MatchSpec | *list*[conda.models.match_spec.MatchSpec], *all_link_path_actions*)

Bases: *CreateInPrefixPathAction*

Base class for path manipulation actions, including linking, unlinking, and others.

Pre and post-transaction plugins should inherit this class to implement their own verification, execution, reversing, and cleanup steps. These methods are guaranteed to be called in the following order:

1. `verify`
2. `execute`
3. `reverse` (only if `execute` raises an exception)
4. `cleanup`

Parameters

- **transaction_context** -- Mapping between target prefixes and PrefixActionGroup instances.
- **target_prefix** -- Target prefix for the action.
- **unlink_precs** -- Package records to be unlinked.
- **link_precs** -- Package records to link.
- **remove_specs** -- Specs to be removed.
- **update_specs** -- Specs to be updated.

- **neutered_specs** -- Specs to be neutered.

classmethod create_actions(*transaction_context*, *package_info*, *target_prefix*, *requested_link_type*, *requested_spec: conda.models.match_spec.MatchSpec | list[conda.models.match_spec.MatchSpec]*, *all_link_path_actions*)

requested_link_type

requested_spec

all_link_path_actions

_execute_successful = False

execute()

Execute the action.

Called after `self.verify()`. If this function raises an exception, `self.reverse()` will be called.

reverse()

Reverse what was done in execute.

Called only if `self.execute()` raises an exception.

class UpdateHistoryAction(*transaction_context*, *target_prefix*, *target_short_path*, *remove_specs*, *update_specs*, *neutered_specs*)

Bases: *CreateInPrefixPathAction*

Base class for path manipulation actions, including linking, unlinking, and others.

Pre and post-transaction plugins should inherit this class to implement their own verification, execution, reversing, and cleanup steps. These methods are guaranteed to be called in the following order:

1. `verify`
2. `execute`
3. `reverse` (only if `execute` raises an exception)
4. `cleanup`

Parameters

- **transaction_context** -- Mapping between target prefixes and PrefixActionGroup instances.
- **target_prefix** -- Target prefix for the action.
- **unlink_precs** -- Package records to be unlinked.
- **link_precs** -- Package records to link.
- **remove_specs** -- Specs to be removed.
- **update_specs** -- Specs to be updated.
- **neutered_specs** -- Specs to be neutered.

classmethod create_actions(*transaction_context*, *target_prefix*, *remove_specs*, *update_specs*, *neutered_specs*)

remove_specs

update_specs

neutered_specs

hold_path

_execute_successful = False

execute()

Execute the action.

Called after `self.verify()`. If this function raises an exception, `self.reverse()` will be called.

reverse()

Reverse what was done in execute.

Called only if `self.execute()` raises an exception.

cleanup()

Carry out any post-execution tasks.

class RegisterEnvironmentLocationAction(*transaction_context*, *target_prefix*)

Bases: *PathAction*

Base class for path manipulation actions, including linking, unlinking, and others.

Pre and post-transaction plugins should inherit this class to implement their own verification, execution, reversing, and cleanup steps. These methods are guaranteed to be called in the following order:

1. `verify`
2. `execute`
3. `reverse` (only if `execute` raises an exception)
4. `cleanup`

Parameters

- **transaction_context** -- Mapping between target prefixes and PrefixActionGroup instances.
- **target_prefix** -- Target prefix for the action.
- **unlink_precs** -- Package records to be unlinked.
- **link_precs** -- Package records to link.
- **remove_specs** -- Specs to be removed.
- **update_specs** -- Specs to be updated.
- **neutered_specs** -- Specs to be neutered.

transaction_context

target_prefix

_execute_successful = False

verify()

Carry out any pre-execution verification.

Should set self._verified = True upon success.

Returns

On failure, this function should return (not raise!) an exception object. At the end of the verification run, all errors will be raised as a CondaMultiError.

execute()

Execute the action.

Called after `self.verify()`. If this function raises an exception, `self.reverse()` will be called.

reverse()

Reverse what was done in execute.

Called only if `self.execute()` raises an exception.

cleanup()

Carry out any post-execution tasks.

abstract property target_full_path

class RemoveFromPrefixPathAction(*transaction_context*, *linked_package_data*, *target_prefix*, *target_short_path*)

Bases: *PrefixPathAction*

Base class for path manipulation actions, including linking, unlinking, and others.

Pre and post-transaction plugins should inherit this class to implement their own verification, execution, reversing, and cleanup steps. These methods are guaranteed to be called in the following order:

1. `verify`
2. `execute`
3. `reverse` (only if `execute` raises an exception)
4. `cleanup`

Parameters

- **transaction_context** -- Mapping between target prefixes and PrefixActionGroup instances.
- **target_prefix** -- Target prefix for the action.
- **unlink_precs** -- Package records to be unlinked.
- **link_precs** -- Package records to link.
- **remove_specs** -- Specs to be removed.
- **update_specs** -- Specs to be updated.
- **neutered_specs** -- Specs to be neutered.

linked_package_data

verify()

Carry out any pre-execution verification.

Should set self._verified = True upon success.

Returns

On failure, this function should return (not raise!) an exception object. At the end of the verification run, all errors will be raised as a CondaMultiError.

class UnlinkPathAction(*transaction_context*, *linked_package_data*, *target_prefix*, *target_short_path*, *link_type=LinkType.hardlink*)

Bases: *RemoveFromPrefixPathAction*

Base class for path manipulation actions, including linking, unlinking, and others.

Pre and post-transaction plugins should inherit this class to implement their own verification, execution, reversing, and cleanup steps. These methods are guaranteed to be called in the following order:

1. `verify`
2. `execute`
3. `reverse` (only if `execute` raises an exception)
4. `cleanup`

Parameters

- **transaction_context** -- Mapping between target prefixes and PrefixActionGroup instances.
- **target_prefix** -- Target prefix for the action.
- **unlink_precs** -- Package records to be unlinked.
- **link_precs** -- Package records to link.
- **remove_specs** -- Specs to be removed.
- **update_specs** -- Specs to be updated.
- **neutered_specs** -- Specs to be neutered.

holding_short_path

holding_full_path

link_type

execute()

Execute the action.

Called after `self.verify()`. If this function raises an exception, `self.reverse()` will be called.

reverse()

Reverse what was done in execute.

Called only if `self.execute()` raises an exception.

cleanup()

Carry out any post-execution tasks.

class RemoveMenuAction(*transaction_context*, *linked_package_data*, *target_prefix*, *target_short_path*)

Bases: *RemoveFromPrefixPathAction*

Base class for path manipulation actions, including linking, unlinking, and others.

Pre and post-transaction plugins should inherit this class to implement their own verification, execution, reversing, and cleanup steps. These methods are guaranteed to be called in the following order:

1. `verify`
2. `execute`
3. `reverse` (only if `execute` raises an exception)
4. `cleanup`

Parameters

- **transaction_context** -- Mapping between target prefixes and PrefixActionGroup instances.
- **target_prefix** -- Target prefix for the action.
- **unlink_precs** -- Package records to be unlinked.
- **link_precs** -- Package records to link.
- **remove_specs** -- Specs to be removed.
- **update_specs** -- Specs to be updated.
- **neutered_specs** -- Specs to be neutered.

classmethod create_actions(*transaction_context*, *linked_package_data*, *target_prefix*)

execute()

Execute the action.

Called after `self.verify()`. If this function raises an exception, `self.reverse()` will be called.

reverse()

Reverse what was done in execute.

Called only if `self.execute()` raises an exception.

cleanup()

Carry out any post-execution tasks.

class RemoveLinkedPackageRecordAction(*transaction_context*, *linked_package_data*, *target_prefix*, *target_short_path*)

Bases: *UnlinkPathAction*

Base class for path manipulation actions, including linking, unlinking, and others.

Pre and post-transaction plugins should inherit this class to implement their own verification, execution, reversing, and cleanup steps. These methods are guaranteed to be called in the following order:

1. `verify`
2. `execute`
3. `reverse` (only if `execute` raises an exception)
4. `cleanup`

Parameters

- **transaction_context** -- Mapping between target prefixes and PrefixActionGroup instances.
- **target_prefix** -- Target prefix for the action.
- **unlink_precs** -- Package records to be unlinked.
- **link_precs** -- Package records to link.
- **remove_specs** -- Specs to be removed.
- **update_specs** -- Specs to be updated.
- **neutered_specs** -- Specs to be neutered.

execute()

Execute the action.

Called after `self.verify()`. If this function raises an exception, `self.reverse()` will be called.

reverse()

Reverse what was done in execute.

Called only if `self.execute()` raises an exception.

class UnregisterEnvironmentLocationAction(*transaction_context*, *target_prefix*)

Bases: *PathAction*

Base class for path manipulation actions, including linking, unlinking, and others.

Pre and post-transaction plugins should inherit this class to implement their own verification, execution, reversing, and cleanup steps. These methods are guaranteed to be called in the following order:

1. `verify`
2. `execute`
3. `reverse` (only if `execute` raises an exception)
4. `cleanup`

Parameters

- **transaction_context** -- Mapping between target prefixes and PrefixActionGroup instances.
- **target_prefix** -- Target prefix for the action.
- **unlink_precs** -- Package records to be unlinked.
- **link_precs** -- Package records to link.
- **remove_specs** -- Specs to be removed.
- **update_specs** -- Specs to be updated.
- **neutered_specs** -- Specs to be neutered.

transaction_context

target_prefix

_execute_successful = False

verify()

Carry out any pre-execution verification.

Should set self._verified = True upon success.

Returns

On failure, this function should return (not raise!) an exception object. At the end of the verification run, all errors will be raised as a CondaMultiError.

execute()

Execute the action.

Called after `self.verify()`. If this function raises an exception, `self.reverse()` will be called.

reverse()

Reverse what was done in execute.

Called only if `self.execute()` raises an exception.

cleanup()

Carry out any post-execution tasks.

abstract property target_full_path

class CacheUrlAction(*url*, *target_pkgs_dir*, *target_package_basename*, *sha256=None*, *size=None*, *md5=None*)

Bases: *PathAction*

Base class for path manipulation actions, including linking, unlinking, and others.

Pre and post-transaction plugins should inherit this class to implement their own verification, execution, reversing, and cleanup steps. These methods are guaranteed to be called in the following order:

1. `verify`
2. `execute`
3. `reverse` (only if `execute` raises an exception)
4. `cleanup`

Parameters

- **transaction_context** -- Mapping between target prefixes and PrefixActionGroup instances.
- **target_prefix** -- Target prefix for the action.
- **unlink_precs** -- Package records to be unlinked.
- **link_precs** -- Package records to link.
- **remove_specs** -- Specs to be removed.
- **update_specs** -- Specs to be updated.
- **neutered_specs** -- Specs to be neutered.

url

target_pkgs_dir

target_package_basename

sha256 = None

size = None

md5 = None

hold_path

verify()

Carry out any pre-execution verification.

Should set self._verified = True upon success.

Returns

On failure, this function should return (not raise!) an exception object. At the end of the verification run, all errors will be raised as a CondaMultiError.

execute(*progress_update_callback=None*)

Execute the action.

Called after `self.verify()`. If this function raises an exception, `self.reverse()` will be called.

_execute_local(*source_path*, *target_package_cache*, *progress_update_callback=None*)

_execute_channel(*target_package_cache*, *progress_update_callback=None*)

reverse()

Reverse what was done in execute.

Called only if `self.execute()` raises an exception.

cleanup()

Carry out any post-execution tasks.

property target_full_path

__str__()

class ExtractPackageAction(*source_full_path*, *target_pkgs_dir*, *target_extracted_dirname*, *record_or_spec*, *sha256*, *size*, *md5*)

Bases: *PathAction*

Base class for path manipulation actions, including linking, unlinking, and others.

Pre and post-transaction plugins should inherit this class to implement their own verification, execution, reversing, and cleanup steps. These methods are guaranteed to be called in the following order:

1. `verify`
2. `execute`
3. `reverse` (only if `execute` raises an exception)
4. `cleanup`

Parameters

- **transaction_context** -- Mapping between target prefixes and PrefixActionGroup instances.
- **target_prefix** -- Target prefix for the action.
- **unlink_precs** -- Package records to be unlinked.

- **link_precs** -- Package records to link.
- **remove_specs** -- Specs to be removed.
- **update_specs** -- Specs to be updated.
- **neutered_specs** -- Specs to be neutered.

source_full_path

target_pkgs_dir

target_extracted_dirname

hold_path

record_or_spec

sha256

size

md5

verify()

Carry out any pre-execution verification.

Should set self._verified = True upon success.

Returns

On failure, this function should return (not raise!) an exception object. At the end of the verification run, all errors will be raised as a CondaMultiError.

execute(*progress_update_callback=None*)

Execute the action.

Called after `self.verify()`. If this function raises an exception, `self.reverse()` will be called.

_prepare_extract()

_finish_extract()

reverse()

Reverse what was done in execute.

Called only if `self.execute()` raises an exception.

cleanup()

Carry out any post-execution tasks.

property target_full_path

__str__()

portability

Tools for cross-OS portability.

Functions

_subdir_is_win(subdir) → bool	Determine if the given *subdir* corresponds to a Windows operating system.
update_prefix(path, new_prefix[, placeholder, mode, ...]) → None	Update the prefix in a file or directory.
_run_codesign(paths) → None	
_codesign_or_enqueue(path) → None	
batch_codesign_calls() → collections.abc.Iterator[None]	Batch osx-arm64 `codesign` calls made by *update_prefix()*.
replace_prefix(mode, data, placeholder, new_prefix[, ...]) → bytes	Replaces *placeholder* text with the *new_prefix* provided. The *mode* provided can
binary_replace(data, search, replacement[, encoding, ...]) → bytes	Replaces occurrences of a search string with a replacement string in a given byte string.
has_pyzzer_entry_point(data) → bool	Check if the given byte string contains a pyzzer entry point.
replace_pyzzer_entry_point_shebang(all_data, ...) → bytes	Code adapted from pyzzer. This is meant to deal with entry point exe's created by distlib,
replace_long_shebang(mode, data) → bytes	Replace long shebang lines in text mode with a shorter one.
generate_shebang_for_entry_point(executable[, ...]) → str	This function can be used to generate a shebang line for Python entry points.

Attributes

SHEBANG_REGEX
MAX_SHEBANG_LENGTH
POPULAR_ENCODINGS
_codesign_batch_state

SHEBANG_REGEX = b'^(#!(?:[ ]*)(/(?:\\\\ |[^ \\n\\r\\t])*)(.*))$'

MAX_SHEBANG_LENGTH = 127

POPULAR_ENCODINGS = ('utf-8', 'utf-16-le', 'utf-16-be', 'utf-32-le', 'utf-32-be')

exception _PaddingError

Bases: Exception

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

_subdir_is_win(*subdir: str*) → bool

Determine if the given *subdir* corresponds to a Windows operating system.

Parameters
subdir -- The subdirectory name which may contain an OS identifier.

Returns
Returns True if *subdir* indicates a Windows OS; otherwise, False.

update_prefix(*path: str*, *new_prefix: str*, *placeholder: str = PREFIX_PLACEHOLDER*, *mode: conda.models.enums.FileMode = FileMode.text*, *subdir: str = context.subdir*) → None

Update the prefix in a file or directory.

Parameters

- **path** -- The path to the file or directory.
- **new_prefix** -- The new prefix to replace the old prefix with.
- **placeholder** -- The placeholder to use for the old prefix. Defaults to PREFIX_PLACEHOLDER.
- **mode** -- The file mode. Defaults to FileMode.text.
- **subdir** -- The subdirectory. Defaults to context.subdir.

_codesign_batch_state

_run_codesign(*paths: list[str]*) → None

_codesign_or_enqueue(*path: str*) → None

batch_codesign_calls() → collections.abc.Iterator[None]

Batch osx-arm64 `codesign` calls made by *update_prefix()*.

Direct `update_prefix` callers keep the old immediate-signing behavior. Transaction verification wraps prefix rewrites in this context so the rewritten intermediates can be signed with fewer subprocesses before they are linked into the target prefix. See #15975.

replace_prefix(*mode:* conda.models.enums.FileMode, *data: bytes*, *placeholder: str*, *new_prefix: str*, *subdir: str = 'noarch'*) → bytes

Replaces *placeholder* text with the *new_prefix* provided. The *mode* provided can either be text or binary.

We use the *POPULAR_ENCODINGS* module level constant defined above to make several passes at replacing the placeholder. We do this to account for as many encodings as possible. If this causes any performance problems in the future, it could potentially be removed (i.e. just using the most popular "utf-8" encoding").

More information/discussion available here: https://github.com/conda/conda/pull/9946

Parameters

- **mode** -- The mode of operation.
- **data** -- The original data to be updated.
- **placeholder** -- The placeholder to be replaced.
- **new_prefix** -- The new prefix to be used.
- **subdir** -- The subdirectory to be used.

Returns

The updated data after prefix replacement.

binary_replace(*data: bytes*, *search: bytes*, *replacement: bytes*, *encoding: str = 'utf-8'*, *subdir: str = 'noarch'*) → bytes

Replaces occurrences of a search string with a replacement string in a given byte string.

Parameters

- **data** -- The byte string in which to perform the replacements.
- **search** -- The string to search for in the byte string.
- **replacement** -- The string to replace occurrences of the search string with.

- **encoding** -- The encoding to use when encoding and decoding strings. Defaults to "utf-8".
- **subdir** -- The subdirectory to search for. Defaults to "noarch".

Returns
The byte string with the replacements made.

Raises
_PaddingError -- If the padding calculation results in a negative value.

This function performs replacements only for pyzzer-type entry points on Windows. For all other cases, it returns the original byte string unchanged.

has_pyzzer_entry_point(*data: bytes*) → bool

Check if the given byte string contains a pyzzer entry point.

Parameters
data -- The byte string to check.

Returns
True if the byte string contains a pyzzer entry point, False otherwise.

replace_pyzzer_entry_point_shebang(*all_data: bytes*, *placeholder: bytes*, *new_prefix: str*) → bytes

Code adapted from pyzzer. This is meant to deal with entry point exe's created by distlib, which consist of a launcher, then a shebang, then a zip archive of the entry point code to run. We need to change the shebang. https://bitbucket.org/vinay.sajip/pyzzer/src/5d5740cb04308f067d5844a56fbe91e7a27efccc/pyzzer/__init__.py?at=default&fileviewer=file-view-default#__init__.py-112 # NOQA

Parameters

- **all_data** -- The byte string to search for the entry point.
- **placeholder** -- The placeholder string to search for in the entry point.
- **new_prefix** -- The new prefix to replace the placeholder string with.

Returns
The updated byte string with the replaced shebang.

replace_long_shebang(*mode: conda.models.enums.FileMode*, *data: bytes*) → bytes

Replace long shebang lines in text mode with a shorter one.

Parameters

- **mode** -- The file mode.
- **data** -- The byte string to search for the entry point.

Returns
The updated byte string with the replaced shebang.

generate_shebang_for_entry_point(*executable: str*, *with_usr_bin_env: bool = False*) → str

This function can be used to generate a shebang line for Python entry points.

Use cases: - At install/link time, to generate the *noarch: python* entry points. - conda init uses it to create its own entry point during conda-build

Parameters

- **executable** -- The path to the Python executable.
- **with_usr_bin_env** -- Whether to use the */usr/bin/env* approach.

Returns
The generated shebang line.

prefix_data

Tools for managing the packages installed within an environment.

Classes

PrefixDataType	Basic caching of PrefixData instance objects.
PrefixRecordDict	Lazily convert dict entries to PrefixRecord.
PrefixData	The PrefixData class aims to be the representation of the state

Functions

get_conda_anchor_files_and_records(...) → dict[conda.common.path.PathType, conda.models.records.PrefixRecord]	Return the anchor files for the conda records of python packages.
delete_prefix_from_linked_data(path) → bool	Here, path may be a complete prefix or a dist inside a prefix

Attributes

T

T

class PrefixDataType

Bases: type

Basic caching of PrefixData instance objects.

__call__(*prefix_path: conda.common.path.PathType*, *interoperability: bool | None = None*) → *PrefixData*

class PrefixRecordDict(*dict=None*, */*, ***kwargs*)

Bases: collections.UserDict

Lazily convert dict entries to PrefixRecord.

__getitem__(*package_name: str*) → *conda.models.records.PrefixRecord*

class PrefixData(*prefix_path: conda.common.path.PathType*, *interoperability: bool | None = None*)

The PrefixData class aims to be the representation of the state of a conda environment on disk. The directory where the environment lives is called prefix.

This class supports different types of tasks:

- Reading and querying *conda-meta/*.json* files as *PrefixRecord* objects
- Reading and writing environment-specific configuration (env vars, state file, nonadmin markers, etc)
- Existence checks and validations of name, path, and magic files / markers
- Exposing non-conda packages installed in prefix as *PrefixRecord*, via the plugin system

cache: dict[tuple[pathlib.Path, bool | None], *PrefixData*]

prefix_path: pathlib.Path

_magic_file: pathlib.Path

_frozen_file: pathlib.Path

_last_activated_file: pathlib.Path

__prefix_records: dict[str, *conda.models.records.PrefixRecord*] | None = None

__is_writable: bool | None | conda.auxlib._Null

interoperability: bool

classmethod from_name(*name: str*, ***kwargs*) → *PrefixData*

Creates a PrefixData instance from an environment name.

The name will be validated with *PrefixData.validate_name()* if it does not exist.

Parameters
name -- The name of the environment. Must not contain path separators (/,).

Raises
CondaValueError -- If *name* contains a path separator.

classmethod from_context(*validate: bool = False*, ***kwargs*) → *PrefixData*

Creates a PrefixData instance from the path specified by *context.target_prefix*.

The path and name will be validated with *PrefixData.validate_path()* and *PrefixData.validate_name()*, respectively, if *validate* is *True*.

Parameters

- **validate** -- Whether the path and name should be validated. Useful for environments about to be created.
- **kwargs** -- Additional keyword arguments to pass to the constructor.

property name: str

Returns the name of the environment, if available.

If the environment doesn't live in one the configured *envs_dirs*, an empty string is returned. The construct *prefix_data.name or prefix_data.prefix_path* can be helpful in those cases.

__eq__(*other: Any*) → bool

exists() → bool

Check whether the PrefixData path exists and is a directory.

is_environment() → bool

Check whether the PrefixData path is a valid conda environment.

This is assessed by checking if *conda-meta/history* marker file exists.

is_frozen() → bool

Check whether the environment is marked as frozen, as per CEP 22.

This is assessed by checking if *conda-meta/frozen* marker file exists.

is_base() → bool

Check whether the configured path refers to the *base* environment.

property is_active: bool

Whether this prefix is the currently active environment.

Though is_active isn't an on disk characteristic of an environment, it is a nice to have when assessing the state of an environment.

property is_writable: bool | None | conda.auxlib._Null

Check whether the configured path is writable. This is assessed by checking whether *conda-meta/history* is writable. It if is, it is assumed that the rest of the directory tree is writable too.

Note: The value is cached in the instance. Use *.assert_writable()* for a non- cached check.

assert_exists() → None

Check whether the environment path exists.

Raises
EnvironmentLocationNotFound -- If the check returns False.

assert_environment() → None

Check whether the environment path exists and is a valid conda environment.

Raises
DirectoryNotACondaEnvironmentError -- If the check returns False.

assert_writable() → None

Check whether the environment path is a valid conda environment and is writable.

Raises
EnvironmentNotWritableError -- If the check returns False.

assert_not_frozen() → None

Check whether the environment path is a valid conda environment and is not marked as frozen (as per CEP 22).

Raises
EnvironmentIsFrozenError -- If the environment is marked as frozen.

validate_path(*expand_path: bool = False*) → None

Validate the path of the environment.

It runs the following checks:

- Make sure the path does not contain *:* or *;* (OS-dependent).
- Disallow immediately nested environments (e.g. *$CONDA_ROOT* and *$CONDA_ROOT/my-env*).
- Warn if there are spaces in the path.

Parameters
expand_path -- Whether to process ~ and environment variables in the string. The expanded value will replace *.prefix_path*.

Raises
CondaValueError -- If the environment contains *:*, *;*, or is nested.

validate_name(*allow_base: bool = False*) → None

Validate the name of the environment.

Parameters
allow_base -- Whether to allow *base* as a valid name.

Raises

- *CondaValueError* -- If the name is protected, or if it contains disallowed characters
- (/, ` `, ` -- , `#).

property created: datetime.datetime | None

Returns the time when the environment was created, as evidenced by the *conda-meta* directory creation time (if available). Falls back to *conda-meta/created_at*.

This may return None in the following cases:

- If the environment doesn't exist.
- If the creation time cannot be determined. This can happen for existing environments created with conda versions before 25.11 in Linux systems, where the 'birthtime' metadata is only available in kernels 4.11+, and not implemented in Python as of Python 3.14.

property last_modified: datetime.datetime | None

Returns the time when the environment was last modified, as evidenced by the *conda-meta/history* file modification time. If the environment does not exist, returns None.

property last_activated: datetime.datetime | None

Returns the time when the environment was last activated, as evidenced by the *conda-meta/last_activated* filesystem modification time. If the environment or file doesn't exist, returns None.

size() → int

Compute the total size of a conda environment prefix.

This aggregates the installed size of all packages in the environment, plus the size of non-manifest files under conda-meta (history, markers, etc.)

Returns
Total size in bytes.

load() → None

reload() → Self

_get_json_fn(*prefix_record:* conda.models.records.PrefixRecord) → str

insert(*prefix_record:* conda.models.records.PrefixRecord, *remove_auth:* *bool* = *True*) → None

remove(*package_name:* *str*) → None

get(*package_name:* *str*, *default:* T = *NULL*) → *conda.models.records.PrefixRecord* | *T*

iter_records() → collections.abc.Iterable[*conda.models.records.PrefixRecord*]

iter_records_sorted() → collections.abc.Iterable[*conda.models.records.PrefixRecord*]

map_records() → frozendict.frozendict[str, *conda.models.records.PrefixRecord*]

Map the records to a frozendict of name -> record.

Returns
A mapping of name -> record.

all_subdir_urls() → set[str]

query(*package_ref_or_match_spec:* conda.models.records.PackageRecord | conda.models.match_spec.MatchSpec | *str*) → collections.abc.Iterable[*conda.models.records.PrefixRecord*]

get_conda_packages() → list[*conda.models.records.PrefixRecord*]

Get conda packages sorted alphabetically by name.

Returns
Sorted conda package records.

get_python_packages() → list[*conda.models.records.PrefixRecord*]

Get Python packages (installed via pip) sorted alphabetically by name.

Returns
Sorted Python package records.

property _prefix_records: dict[str, *conda.models.records.PrefixRecord*] | None

_load_single_record(*prefix_record_json_path: conda.common.path.PathType*) → None

_get_environment_state_file() → dict[str, dict[str, str]]

_write_environment_state_file(*state: dict[str, dict[str, str]]*) → None

_ensure_no_reserved_env_vars(*env_vars_names: collections.abc.Iterable[str]*) → None

Ensure that the set of env_var_names does not contain any reserved env vars. Will raise an OperationNotAllowed if a reserved env var is found.

get_environment_env_vars() → dict[str, str] | dict[bytes, bytes]

set_environment_env_vars(*env_vars: dict[str, str]*) → dict[str, str] | None

unset_environment_env_vars(*env_vars: list[str]*) → dict[str, str] | None

set_creation_time() → None

Writes a .creation-time file in conda-meta with the current timestamp, meant to be used by .created property as a fallback.

set_nonadmin() → None

Creates $PREFIX/.nonadmin if sys.prefix/.nonadmin exists (on Windows).

get_pinned_specs() → tuple[*conda.models.match_spec.MatchSpec*]

Find pinned specs from file and return a tuple of MatchSpec.

get_conda_anchor_files_and_records(*site_packages_short_path: conda.common.path.PathType*, *python_records: collections.abc.Iterable[*conda.models.records.PrefixRecord*]*) → dict[conda.common.path.PathType, *conda.models.records.PrefixRecord*]

Return the anchor files for the conda records of python packages.

delete_prefix_from_linked_data(*path: str | os.PathLike | pathlib.Path*) → bool

Here, path may be a complete prefix or a dist inside a prefix

solve

The classic solver implementation.

Classes

BaseSolver	A high-level API to conda's solving logic. Three public methods are provided to access a
Solver	A high-level API to conda's solving logic. Three public methods are provided to access a
SolverStateContainer	

Functions

get_pinned_specs(prefix) → tuple[conda.models.match_spec.MatchSpec]	Find pinned specs from file and return a tuple of MatchSpec.
diff_for_unlink_link_precs(prefix, final_precs[, ...]) → tuple[tuple[conda.models.records.PackageRecord, ...], tuple[conda.models.records.PackageRecord, ...]]	

class BaseSolver(*prefix: str*, *channels: collections.abc.Iterable[*conda.models.channel.Channel*]* | *None* = *None*, *subdirs: collections.abc.Iterable[str]* = *()*, *specs_to_add: collections.abc.Iterable[*conda.models.match_spec.MatchSpec*]* = *()*, *specs_to_remove: collections.abc.Iterable[*conda.models.match_spec.MatchSpec*]* = *()*, *repodata_fn: str* = *REPODATA_FN*, *command=NULL*)

A high-level API to conda's solving logic. Three public methods are provided to access a solution in various forms.

- *solve_final_state()*
- *solve_for_diff()*
- *solve_for_transaction()*

Parameters

- **prefix** (*str*) -- The conda prefix / environment location for which the *Solver* is being instantiated.
- **channels** (Sequence[Channel]) -- A prioritized list of channels to use for the solution.
- **subdirs** (*Sequence[str]*) -- A prioritized list of subdirs to use for the solution.
- **specs_to_add** (set[MatchSpec]) -- The set of package specs to add to the prefix.
- **specs_to_remove** (set[MatchSpec]) -- The set of package specs to remove from the prefix.

_index: *conda.core.index.ReducedIndex* | None

_provided_index: *conda.core.index.Index* | dict | None

_r: *conda.resolve.Resolve* | None

supports_exclude_newer_global: ClassVar[bool] = False

supports_exclude_newer_channel: ClassVar[bool] = False

supports_exclude_newer_package: ClassVar[bool] = False

classmethod __init_subclass__(***kwargs*)

Require solver subclasses to opt in to each policy capability.

External solvers currently inherit from `Solver` rather than `BaseSolver`, so they would otherwise inherit classic's capabilities.

prefix

_channels

channels

subdirs

specs_to_add

specs_to_add_names

specs_to_remove

unmerged_specs_to_add

unmerged_specs_to_remove

neutered_specs = ()

_command

_repodata_fn = 'repodata.json'

_prepared = False

_pool_cache

exclude_newer_policy

_validate_exclude_newer_support() → None

_validate_exclude_newer_link_precs(*link_precs: tuple[*conda.models.records.PackageRecord, *...]*) → None

solve_for_transaction(*update_modifier=NULL*, *deps_modifier=NULL*, *prune=NULL*, *ignore_pinned=NULL*, *force_remove=NULL*, *force_reinstall=NULL*, *should_retry_solve=False*) → *conda.core.link.UnlinkLinkTransaction*

Return an UnlinkLinkTransaction instance that can be used to execute the solution on an environment.

Parameters

- **deps_modifier** (DepsModifier) -- See *solve_final_state()*.
- **prune** (*bool*) -- See *solve_final_state()*.
- **ignore_pinned** (*bool*) -- See *solve_final_state()*.
- **force_remove** (*bool*) -- See *solve_final_state()*.
- **force_reinstall** (*bool*) -- See *solve_for_diff()*.
- **should_retry_solve** (*bool*) -- See *solve_final_state()*.

Return type

UnlinkLinkTransaction

solve_for_diff(*update_modifier=NULL*, *deps_modifier=NULL*, *prune=NULL*, *ignore_pinned=NULL*, *force_remove=NULL*, *force_reinstall=NULL*, *should_retry_solve=False*) → tuple[tuple[*conda.models.records.PackageRecord*, ...], tuple[*conda.models.records.PackageRecord*, ...]]

Gives the package references to remove from an environment, followed by the package references to add to an environment.

Parameters

- **deps_modifier** (DepsModifier) -- See *solve_final_state()*.
- **prune** (*bool*) -- See *solve_final_state()*.
- **ignore_pinned** (*bool*) -- See *solve_final_state()*.
- **force_remove** (*bool*) -- See *solve_final_state()*.
- **force_reinstall** (*bool*) --

  For requested specs_to_add that are already satisfied in the environment, instructs the solver to remove the package and spec from the environment, and then add it back--possibly with the exact package instance modified, depending on the spec exactness.

- **should_retry_solve** (*bool*) -- See *solve_final_state()*.

Returns

A two-tuple of PackageRef sequences. The first is the group of packages to remove from the environment, in sorted dependency order from leaves to roots. The second is the group of packages to add to the environment, in sorted dependency order from roots to leaves.

Return type

tuple[PackageRef], tuple[PackageRef]

abstractmethod solve_final_state(*update_modifier=NULL*, *deps_modifier=NULL*, *prune=NULL*, *ignore_pinned=NULL*, *force_remove=NULL*, *should_retry_solve=False*) → tuple[*conda.models.records.PackageRecord*, ...]

Gives the final, solved state of the environment.

_notify_conda_outdated(*link_precs*)

_notify_pip_as_python_deprecation(*link_precs*)

class Solver(*prefix: str*, *channels: collections.abc.Iterable[*conda.models.channel.Channel*] | None = None*, *subdirs: collections.abc.Iterable[str] = ()*, *specs_to_add: collections.abc.Iterable[*conda.models.match_spec.MatchSpec*] = ()*, *specs_to_remove: collections.abc.Iterable[*conda.models.match_spec.MatchSpec*] = ()*, *repodata_fn: str = REPODATA_FN*, *command=NULL*)

Bases: *BaseSolver*

A high-level API to conda's solving logic. Three public methods are provided to access a solution in various forms.

- *solve_final_state()*
- `solve_for_diff()`
- `solve_for_transaction()`

Parameters

- **prefix** (*str*) -- The conda prefix / environment location for which the *Solver* is being instantiated.
- **channels** (Sequence[Channel]) -- A prioritized list of channels to use for the solution.
- **subdirs** (*Sequence[str]*) -- A prioritized list of subdirs to use for the solution.
- **specs_to_add** (set[MatchSpec]) -- The set of package specs to add to the prefix.
- **specs_to_remove** (set[MatchSpec]) -- The set of package specs to remove from the prefix.

supports_exclude_newer_global = True

supports_exclude_newer_channel = True

supports_exclude_newer_package = True

solve_final_state(*update_modifier=NULL*, *deps_modifier=NULL*, *prune=NULL*, *ignore_pinned=NULL*, *force_remove=NULL*, *should_retry_solve=False*) → tuple[*conda.models.records.PackageRecord*, ...]

Gives the final, solved state of the environment.

Parameters

- **update_modifier** (UpdateModifier) -- An optional flag directing how updates are handled regarding packages already existing in the environment.
- **deps_modifier** (DepsModifier) -- An optional flag indicating special solver handling for dependencies. The default solver behavior is to be as conservative as possible with dependency updates (in the case the dependency already exists in the environment), while still ensuring all dependencies are satisfied. Options include * NO_DEPS * ONLY_DEPS * UPDATE_DEPS * UPDATE_DEPS_ONLY_DEPS * FREEZE_INSTALLED
- **prune** (*bool*) -- If True, the solution will not contain packages that were previously brought into the environment as dependencies but are no longer required as dependencies and are not user-requested.
- **ignore_pinned** (*bool*) -- If True, the solution will ignore pinned package configuration for the prefix.
- **force_remove** (*bool*) -- Forces removal of a package without removing packages that depend on it.
- **should_retry_solve** (*bool*) -- Indicates whether this solve will be retried. This allows us to control whether to call find_conflicts (slow) in ssc.r.solve

Returns

In sorted dependency order from roots to leaves, the package references for the solved state of the environment.

Return type

tuple[*PackageRecord*, ...]

determine_constricting_specs(*spec*, *solution_precs*)

get_request_package_in_solution(*solution_precs*, *specs_map*)

get_constrained_packages(*pre_packages*, *post_packages*, *index_keys*)

_collect_all_metadata(*ssc*)

_remove_specs(*ssc*)

_find_inconsistent_packages(*ssc*)

_package_has_updates(*ssc*, *spec*, *installed_pool*)

_should_freeze(*ssc*, *target_prec*, *conflict_specs*, *explicit_pool*, *installed_pool*)

_add_specs(*ssc*)

_run_sat(*ssc*)

_post_sat_handling(*ssc*)

_prepare(*prepared_specs*) → tuple[*conda.core.index.ReducedIndex*, *conda.resolve.Resolve*]

class SolverStateContainer(*prefix*, *update_modifier*, *deps_modifier*, *prune*, *ignore_pinned*, *force_remove*, *should_retry_solve*)

update_modifier

deps_modifier

prune

ignore_pinned

force_remove

should_retry_solve

prefix

index = None

r = None

specs_map

solution_precs = None

add_back_map

final_environment_specs = None

prefix_data()

specs_from_history_map()

track_features_specs()

pinned_specs()

set_repository_metadata(*index*, *r*)

_init_solution_precs()

working_state_reset()

get_pinned_specs(*prefix: str*) → tuple[*conda.models.match_spec.MatchSpec*]

Find pinned specs from file and return a tuple of MatchSpec.

diff_for_unlink_link_precs(*prefix*, *final_precs: collections.abc.Sequence[conda.models.records.PackageRecord]*, *specs_to_add: collections.abc.Iterable[conda.models.match_spec.MatchSpec] = ()*, *force_reinstall=NULL*) → tuple[tuple[*conda.models.records.PackageRecord*, ...], tuple[*conda.models.records.PackageRecord*, ...]]

subdir_data

Tools for managing a subdir's repodata.json.

Classes

SubdirDataType	
PackageRecordList	Lazily convert dicts to PackageRecord.
SubdirData	Repodata for a channel subdir, with instance caching via `SubdirDataType`.

Functions

_search_package_via_shards(spec, channel_urls, subdirs) → tuple[conda.models.records.PackageRecord, ...]	
_search_package(spec, channel_urls, subdirs[, repodata_fn]) → tuple[conda.models.records.PackageRecord, ...]	
query_all(match_spec[, channels, subdirs, repodata_fn]) → tuple[conda.models.records.PackageRecord, ...]	Execute a query against all repodata instances in the channel/subdir

Attributes

REPODATA_PICKLE_VERSION
MAX_REPODATA_VERSION
REPODATA_HEADER_RE

REPODATA_PICKLE_VERSION = 30

MAX_REPODATA_VERSION = 2

REPODATA_HEADER_RE = b'"(_etag|_mod|_cache_control)":[ ]?"(.*?[^\\\\])"[,}\\s]'

class SubdirDataType

Bases: `type`

__call__(*channel:* conda.models.channel.Channel, *repodata_fn: str = REPODATA_FN*) → *SubdirData*

Return a cached or newly constructed `SubdirData` instance.

Parameters

- **channel** -- The channel object.

- **repodata_fn** -- The name of the repodata file. Defaults to REPODATA_FN.

Returns
A SubdirData instance.

class PackageRecordList(*initlist=None*)

Bases: collections.UserList

Lazily convert dicts to PackageRecord.

This class inherits from the built-in UserList class and provides a way to lazily convert dictionaries to PackageRecord objects. It overrides the __getitem__ method to check if the item at the given index is a PackageRecord object. If not, it converts the dictionary to a PackageRecord object and stores it in the data list.

Parameters
data -- The list of items.

__getitem__(*i: int*) → *conda.models.records.PackageRecord*

Returns the PackageRecord at index i. If i is a slice, returns a new instance of PackageRecordList containing the PackageRecords at the indices in i. If the PackageRecord at index i is not already an instance of PackageRecord, it is converted to one and stored back in the data list.

Parameters
i -- An integer or slice object indicating the index or indices of the PackageRecord(s) to be returned.

Returns
If i is a slice, returns a new instance of PackageRecordList containing the PackageRecords at the indices in i. If i is an integer, returns the PackageRecord at index i.

class SubdirData(*channel:* conda.models.channel.Channel, *repodata_fn: str = REPODATA_FN*, *RepoInterface:* type*[SubdirData.__init__.RepoInterface] = CondaRepoInterface*)

Repodata for a channel subdir, with instance caching via SubdirDataType.

Initializes a new instance of the SubdirData class.

Parameters

- **channel** -- The channel object.
- **repodata_fn** -- The repodata filename.
- **RepoInterface** -- The RepoInterface class.

cache: dict[tuple[str, str], *PackageRecordList* | *SubdirData*]

classmethod clear_cached_local_channel_data(*exclude_file: bool = True*) → None

Clear the cached local channel data.

This method is used to clear the cached local channel data. It is primarily used during unit tests to handle changes in the CONDA_USE_ONLY_TAR_BZ2 environment variable during the process lifetime.

Parameters
exclude_file -- A flag indicating whether to exclude file:// URLs from the cache.

static query_all(*package_ref_or_match_spec:* conda.models.match_spec.MatchSpec | *str*, *channels:* *collections.abc.Iterable[*conda.models.channel.Channel | *str]* | *None = None*, *subdirs:* *collections.abc.Iterable[str]* | *None = None*, *repodata_fn: str = REPODATA_FN*) → tuple[*conda.models.records.PackageRecord*, ...]

Execute a query against all repodata instances in the channel/subdir matrix.

Parameters

- **package_ref_or_match_spec** -- A *MatchSpec* query object. A *str* will be turned into a *MatchSpec* automatically.
- **channels** -- An iterable of urls for channels or *Channel* objects. If None, will fall back to *context.channels*.
- **subdirs** -- If None, will fall back to context.subdirs.
- **repodata_fn** -- The filename of the repodata.

Returns
A tuple of *PackageRecord* objects.

query(*package_ref_or_match_spec: str | conda.models.match_spec.MatchSpec*) → collections.abc.Iterator[*conda.models.records.PackageRecord*]

A function that queries for a specific package reference or MatchSpec object.

Parameters
package_ref_or_match_spec -- The package reference or MatchSpec object to query.

Yields
PackageRecord objects.

channel

url_w_subdir

url_w_credentials

_base_url

_base_url_w_credentials

repodata_fn = 'repodata.json'

RepoInterface

_loaded = False

_key_mgr = None

property _repo: *conda.gateways.repodata.RepoInterface*

Changes as we mutate self.repodata_fn.

property repo_cache: *conda.gateways.repodata.RepodataCache*

Returns the *RepodataCache* object associated with the current instance of *SubdirData*.

property repo_fetch: *conda.gateways.repodata.RepodataFetch*

Object to get repodata. Not cached since self.repodata_fn is mutable.

Replaces self._repo & self.repo_cache.

reload() → Self

Update the instance with new information.

property cache_path_base: str

Get the base path for caching the repodata.json file.

This method returns the base path for caching the repodata.json file. It is used to construct the full path for caching the file.

property url_w_repodata_fn: str

Get the URL with the repodata filename.

This method returns the URL with the repodata filename.

property cache_path_json: pathlib.Path

Get the path to the cache file.

This method returns the path to the cache file.

property cache_path_state: pathlib.Path

Out-of-band etag and other state needed by the RepoInterface.

Get the path to the cache state file.

This method returns the path to the cache state file.

property cache_path_pickle: str

Get the path to the cache pickle file.

This method returns the path to the cache pickle file.

load() → Self

Load the internal state of the SubdirData instance.

This method loads the internal state of the SubdirData instance.

iter_records() → collections.abc.Iterator[*conda.models.records.PackageRecord*]

A function that iterates over package records.

This function checks if the package records are loaded. If not loaded, it loads them. It returns an iterator over the package records. The package_records attribute could potentially be replaced with fully-converted UserList data after going through the entire list.

_iter_records_by_name(*name: str*) → collections.abc.Iterator[*conda.models.records.PackageRecord*]

A function that iterates over package records by name.

This function iterates over package records and yields those whose name matches the given name. If include_self is True, it also yields the record with the given name.

_load() → dict[str, Any]

Try to load repodata. If e.g. we are downloading *current_repodata.json*, fall back to *repodata.json* when the former is unavailable.

_pickle_me() → None

Pickle the object to the specified file.

_read_local_repodata(*state:* conda.gateways.repodata.RepodataState) → dict[str, Any]

Read local repodata from the cache and process it.

_pickle_valid_checks(*pickled_state: dict[str, Any]*, *mod: str*, *etag: str*) → collections.abc.Iterator[tuple[str, Any, Any]]

Throw away the pickle if these don't all match.

Parameters

- **pickled_state** -- The pickled state to compare against.
- **mod** -- The modification information to check.
- **etag** -- The etag to compare against.

Yields
Tuples of the form (check_name, pickled_value, current_value).

_read_pickled(*state:* conda.gateways.repodata.RepodataState) → dict[str, Any] | None

Read pickled repodata from the cache and process it.

Parameters
state -- The repodata state.

Returns
A dictionary containing the processed pickled repodata, or None if the repodata is not pickled.

_process_raw_repodata_str(*raw_repodata_str: str*, *state:* conda.gateways.repodata.RepodataState | *None* = *None*) → dict[str, Any]

State contains information that was previously in-band in raw_repodata_str.

Process the raw repodata string and return the processed repodata.

Parameters
raw_repodata_str -- The raw repodata string.

Returns
A dictionary containing the processed repodata.

_process_raw_repodata(*repodata: dict[str, Any]*, *state:* conda.gateways.repodata.RepodataState | *None* = *None*) → dict[str, Any]

Process the raw repodata dictionary and return the processed repodata.

Parameters

- **repodata** -- The raw repodata dictionary.
- **state** -- The repodata state. Defaults to None.

Returns
A dictionary containing the processed repodata.

_get_base_url(*repodata: dict*, *with_credentials: bool* = *True*) → str

In repodata_version=1, .tar.bz2 and .conda artifacts are assumed to be colocated next to repodata.json, in the same server and directory.

In repodata_version=2, repodata.json files can define a 'base_url' field to override that default assumption. See CEP-15 for more details.

This method deals with both cases and returns the appropriate value.

Get the base URL for the given repodata.

Parameters

- **repodata** -- The parsed repodata.json payload.
- **with_credentials** -- Whether to include credentials in the returned URL.

Returns
The base URL corresponding to the provided repodata.

_search_package_via_shards(*spec:* conda.models.match_spec.MatchSpec, *channel_urls: tuple[str, ...]* | *list[str]*, *subdirs: tuple[str, ...]* | *list[str]*) → tuple[*conda.models.records.PackageRecord*, ...]

_search_package(*spec:* conda.models.match_spec.MatchSpec, *channel_urls: tuple[str, ...] | list[str]*, *subdirs: tuple[str, ...] | list[str]*, *repodata_fn: str = REPODATA_FN*) → tuple[*conda.models.records.PackageRecord*, ...]

query_all(*match_spec:* conda.models.match_spec.MatchSpec, *channels: collections.abc.Iterable[*conda.models.channel.Channel | *str] | None = None*, *subdirs: collections.abc.Iterable[str] | None = None*, *repodata_fn: str = REPODATA_FN*) → tuple[*conda.models.records.PackageRecord*, ...]

Execute a query against all repodata instances in the channel/subdir matrix. Will try to load data from shards if conda is configured to use shards and the channel provides sharded repodata. If not, will fall back to providing data from monolithic repodata.

Parameters

- **match_spec** -- A *MatchSpec* query object.
- **channels** -- An iterable of urls for channels or *Channel* objects. If None, will fall back to *context.channels*.
- **subdirs** -- If None, will fall back to context.subdirs.
- **repodata_fn** -- The filename of the repodata.

Returns

A tuple of *PackageRecord* objects.

deprecations

Tools to aid in deprecating code.

Classes

DeprecationHandler	Factory to create a deprecation handle for the specified version.
_ConstantDeprecationRegistry	Registry of deprecated module constants.

Attributes

T
_UNSET
deprecated

T

exception DeprecatedError

Bases: RuntimeError

Unspecified run-time error.

Initialize self. See help(type(self)) for accurate signature.

_UNSET

class DeprecationHandler(*version: str*)

Factory to create a deprecation handle for the specified version.

Parameters
version -- The version to compare against when checking deprecation statuses.

_version: *str | None*

_version_tuple: *tuple[int, ...] | None*

_version_object: *packaging.version.Version | None*

static _get_version_tuple(*version: str*) → tuple[int, ...] | None

Return version as non-empty tuple of ints if possible, else None.

Parameters
version -- Version string to parse.

_version_less_than(*version: str*) → bool

Test whether own version is less than the given version.

Parameters
version -- Version string to compare against.

__call__(*deprecate_in: str*, *remove_in: str*, *, *addendum: str | None = None*, *stack: int = 0*, *deprecation_type: type[Warning] = DeprecationWarning*) → collections.abc.Callable[[collections.abc.Callable[*P*, *T*]], collections.abc.Callable[*P*, *T*]]

Deprecation decorator for functions, methods, & classes.

Parameters

- **deprecate_in** -- Version in which code will be marked as deprecated.
- **remove_in** -- Version in which code is expected to be removed.
- **addendum** -- Optional additional messaging. Useful to indicate what to do instead.
- **stack** -- Optional stacklevel increment.

argument(*deprecate_in: str*, *remove_in: str*, *argument: str*, *, *rename: str | None = None*, *addendum: str | None = None*, *stack: int = 0*, *deprecation_type: type[Warning] = DeprecationWarning*) → collections.abc.Callable[[collections.abc.Callable[*P*, *T*]], collections.abc.Callable[*P*, *T*]]

Deprecation decorator for keyword arguments.

Parameters

- **deprecate_in** -- Version in which code will be marked as deprecated.
- **remove_in** -- Version in which code is expected to be removed.
- **argument** -- The argument to deprecate.
- **rename** -- Optional new argument name.
- **addendum** -- Optional additional messaging. Useful to indicate what to do instead.
- **stack** -- Optional stacklevel increment.

action(*deprecate_in: str*, *remove_in: str*, *action: ActionType*, *, *addendum: str | None = None*, *stack: int = 0*, *deprecation_type: type[Warning] = FutureWarning*) → ActionType

Wraps any argparse.Action to issue a deprecation warning.

module(*deprecate_in: str*, *remove_in: str*, *, *addendum: str | None = None*, *stack: int = 0*) → None

Deprecation function for modules.

Parameters

- **deprecate_in** -- Version in which code will be marked as deprecated.
- **remove_in** -- Version in which code is expected to be removed.
- **addendum** -- Optional additional messaging. Useful to indicate what to do instead.
- **stack** -- Optional stacklevel increment.

constant(*deprecate_in: str*, *remove_in: str*, *constant: str*, *value: Any = _UNSET*, ***, *factory: collections.abc.Callable[[], Any] | None = None*, *addendum: str | None = None*, *stack: int = 0*, *deprecation_type: type[Warning] = DeprecationWarning*) → None

Deprecation function for module constant/global.

Parameters

- **deprecate_in** -- Version in which code will be marked as deprecated.
- **remove_in** -- Version in which code is expected to be removed.
- **constant**
- **value** -- The value to return when the deprecated constant is accessed. Mutually exclusive with `factory`.
- **factory** -- A zero-argument callable invoked the first time the deprecated constant is accessed; the result is cached. Use this to avoid paying registration-time costs (e.g., heavy imports) for a symbol that may never be accessed. Mutually exclusive with `value`.
- **addendum** -- Optional additional messaging. Useful to indicate what to do instead.
- **stack** -- Optional stacklevel increment.

topic(*deprecate_in: str*, *remove_in: str*, ***, *topic: str*, *addendum: str | None = None*, *stack: int = 0*, *deprecation_type: type[Warning] = DeprecationWarning*) → None

Deprecation function for a topic.

Parameters

- **deprecate_in** -- Version in which code will be marked as deprecated.
- **remove_in** -- Version in which code is expected to be removed.
- **topic** -- The topic being deprecated.
- **addendum** -- Optional additional messaging. Useful to indicate what to do instead.
- **stack** -- Optional stacklevel increment.

_get_module(*stack: int*) → tuple[types.ModuleType, str]

Detect the module from which we are being called.

Parameters
stack -- The stacklevel increment.

Returns
The module and module name.

_generate_message(*deprecate_in: str*, *remove_in: str*, *prefix: str*, *addendum: str | None*, ***, *deprecation_type: type[Warning]*) → tuple[*type*[Warning] | None, str]

Generate the standardized deprecation message and determine whether the deprecation is pending, active, or past.

Parameters

- **deprecate_in** -- Version in which code will be marked as deprecated.

- **remove_in** -- Version in which code is expected to be removed.
- **prefix** -- The message prefix, usually the function name.
- **addendum** -- Additional messaging. Useful to indicate what to do instead.
- **deprecation_type** -- The warning type to use for active deprecations.

Returns
The warning category (if applicable) and the message.

class _ConstantDeprecationRegistry

Registry of deprecated module constants.

Serves as a module's __getattr__, issuing deprecation warnings when registered constants are accessed.

deprecations: dict[str, tuple[str, type[Warning], int, Any, bool]]

fullname: str

fallback: collections.abc.Callable[[str], Any] | None

__call__(*name: str*) → Any

register(*constant: str*, *message: str*, *category:* type*[Warning]*, *stack: int*, *value: Any*, ***, *factory: bool = False*) → None

deprecated

env

env

Environment object describing the conda environment.yaml file.

Classes

Dependencies	A `dict` subclass that parses the raw dependencies into a conda and pip list
EnvironmentYaml	A class representing an `environment.yaml` file

Functions

field_type_validation(field_name, value, value_type) → None	Validates the type of a value
prefix_validation(prefix)	Validate the contents of the prefix field.
name_validation(name)	Validate the contents of the name field.
dependencies_validation(dependencies)	Validate the contents of the dependencies field.
channels_validation(channels)	Validate the contents of the channels field.
variables_validation(variables)	Validate the contents of the variables field.
get_schema_errors(data) → list[conda.exceptions.EnvironmentFileInvalid]	Parses environment.yaml data to build a list of schema errors
validate_keys(data, kwargs)	Check for unknown keys, remove them and print a warning

continues on next page

Table 167 – continued from previous page

from_environment(name, prefix[, no_builds, ...])	Build an `EnvironmentYaml` from an existing environment prefix.
from_yaml(yamlstr, **kwargs) → EnvironmentYaml	Load and return a `EnvironmentYaml` from a given `yaml` string
_expand_channels(data)	Expands `EnvironmentYaml` variables for the channels found in the `yaml` data
load_file(filename)	Load and return an yaml string from a given file
from_file(filename)	Load and return an `EnvironmentYaml` from a given file
print_result(args, prefix, result)	Print the result of an install operation

Attributes

REQUIRED_KEYS
OPTIONAL_KEYS
VALID_KEYS
SCHEMA_VALIDATORS

REQUIRED_KEYS

OPTIONAL_KEYS

VALID_KEYS

field_type_validation(*field_name: str*, *value: Any*, *value_type: Any*) → None

Validates the type of a value

prefix_validation(*prefix: str*)

Validate the contents of the prefix field.

Will ensure:

- prefix is a string

name_validation(*name: str*)

Validate the contents of the name field.

Will ensure:

- name is a string

dependencies_validation(*dependencies: list*)

Validate the contents of the dependencies field.

Will ensure:

- dependencies are a list
- all string dependencies are MatchSpec compatible
- the only other type allowed is a dict

channels_validation(*channels: list*)

Validate the contents of the channels field.

Will ensure:

- channels is a list

- all entries are strings

variables_validation(*variables: dict[str, str]*)

Validate the contents of the variables field.

Will ensure:

- variables is a dict
- all entries are strings

SCHEMA_VALIDATORS

get_schema_errors(*data: dict*) → list[*conda.exceptions.EnvironmentFileInvalid*]

Parses environment.yaml data to build a list of schema errors

Will produce errors to ensure:

- all required fields are present
- all fields contain valid data

Parameters

data -- The contents of the environment.yaml

Returns

Validation errors for the environment file schema.

validate_keys(*data*, *kwargs*)

Check for unknown keys, remove them and print a warning

from_environment(*name*, *prefix*, *no_builds=False*, *ignore_channels=False*, *from_history=False*)

Build an `EnvironmentYaml` from an existing environment prefix.

Parameters

- **name** -- Environment name.
- **prefix** -- Path to the environment prefix.
- **no_builds** -- Omit build strings from dependency specs.
- **ignore_channels** -- Do not add channels from installed packages.
- **from_history** -- Base dependencies on explicit specs from install history.

Returns

Representation of the environment suitable for export to YAML.

from_yaml(*yamlstr: str*, ***kwargs*) → *EnvironmentYaml*

Load and return a `EnvironmentYaml` from a given `yaml` string

Parameters

yamlstr -- The contents of the environment.yaml.

Returns

A representation of the environment file

_expand_channels(*data*)

Expands `EnvironmentYaml` variables for the channels found in the `yaml` data

load_file(*filename*)

Load and return an yaml string from a given file

from_file(*filename*)

Load and return an `EnvironmentYaml` from a given file

class Dependencies(*raw*, **args*, ***kwargs*)

Bases: `dict`

A `dict` subclass that parses the raw dependencies into a conda and pip list

Initialize self. See help(type(self)) for accurate signature.

raw

parse()

Parse the raw dependencies into a conda and pip list

add(*package_name*)

Add a package to the `EnvironmentYaml`

class EnvironmentYaml(*name=None*, *filename=None*, *channels=None*, *dependencies=None*, *prefix=None*, *variables=None*)

A class representing an `environment.yaml` file

name = None

filename = None

prefix = None

dependencies

variables = None

channels = None

add_channels(*channels*)

Add channels to the `EnvironmentYaml`

remove_channels()

Remove all channels from the `EnvironmentYaml`

to_dict(*stream=None*)

Convert information related to the `EnvironmentYaml` into a dictionary

to_yaml(*stream=None*)

Convert information related to the `EnvironmentYaml` into a `yaml` string

save()

Save the `EnvironmentYaml` data to a `yaml` file

to_environment_model() → *conda.models.environment.Environment*

Convert the `Environment` into a `model.Environment` object

print_result(*args*, *prefix*, *result*)

Print the result of an install operation

installers

base

Dynamic installer loading.

Functions

get_installer(name, *[, file])	Load the environment installer module for the given name.

get_installer(*name*, *, *file=None*)

Load the environment installer module for the given name.

Parameters

- **name** -- Installer name (e.g., `conda` or `pip`).
- **file** -- Optional path to the environment file, included in error output.

Raises

InvalidInstaller -- If no installer module can be imported for `name`.

conda

Conda-flavored installer.

Functions

solve(prefix, specs, args, env, *, **kwargs) → conda.core.solve.Solver	Solve the environment.
dry_run(specs, args, env, *_, **kwargs) → conda.env.env.EnvironmentYaml	Do a dry run of the environment solve.
install(prefix, specs, args, env, *_, **kwargs) → dict | None	Install packages into a conda environment.

solve(*prefix: str*, *specs: list[str]*, *args: argparse.Namespace*, *env:* conda.models.environment.Environment, *, ***kwargs*) → *conda.core.solve.Solver*

Solve the environment.

Parameters

- **prefix** -- Installation target directory
- **specs** -- Package specifications to install
- **args** -- Command-line arguments
- **env** -- Environment object

Returns

Solver object

dry_run(*specs: list[str]*, *args: argparse.Namespace*, *env:* conda.models.environment.Environment, *_, ***kwargs*) → *conda.env.env.EnvironmentYaml*

Do a dry run of the environment solve.

Parameters

- **specs** -- Package specifications to install
- **args** -- Command-line arguments
- **env** -- Environment object

Returns
Solved environment object

install(*prefix: str*, *specs: list[str]*, *args: argparse.Namespace*, *env:* conda.models.environment.Environment, **_*, ***kwargs*) → dict | None

Install packages into a conda environment.

This function handles two main paths: 1. For environments with explicit_specs (from @EXPLICIT files): Bypasses the solver

and directly installs packages using conda.misc.explicit() as required by CEP-23.

2. For regular Environment instances: Uses the solver to determine the optimal package set before installation.

Parameters

- **prefix** -- The target installation path for the environment
- **specs** -- Package specifications to install
- **args** -- Command-line arguments from the conda command
- **env** -- Environment object containing dependencies and channels

Returns
Installation result information

Note

This implementation follows CEP-23, which states: "When an explicit input file is processed, the conda client SHOULD NOT invoke a solver."

pip

Pip-flavored installer.

Functions

install(prefix, specs, args, *_[, workdir, ...])	Installs the pip dependencies in specs using a temporary pip requirements file.

install(*prefix*, *specs*, *args*, **_*, *workdir=None*, *requirements_sources=None*, ***kwargs*)

Installs the pip dependencies in specs using a temporary pip requirements file.

Parameters

- **prefix** (*string*) -- The path to the python and pip executables.
- **specs** (*iterable of strings*) -- Each element should be a valid pip dependency. Ignored when requirements_sources is nonempty. See: https://pip.pypa.io/en/stable/user_guide/#requirements-files

  https://pip.pypa.io/en/stable/reference/pip_install/#requirements-file-format
- **workdir** (*str | None, optional*) -- Working directory for resolving relative paths in specs (e.g. -e ./local_pkg). Caller should derive from the environment file path. None for URLs or when no file path is available.

- **requirements_sources** (*list[tuple[list[str], str | None]] | None*, optional) -- Pip requirements grouped by source file workdir. When present, conda writes one temporary requirements file and passes it to one pip subprocess so pip resolves all specs together.

pip_util

Functions related to core conda functionality that relates to pip

NOTE: This modules used to in conda, as conda/pip.py

Functions

get_pip_workdir(file_path) → str | None	Derive the working directory for pip install when resolving relative paths
pip_subprocess(args, prefix, cwd)	Run pip in a subprocess
get_pip_installed_packages(stdout)	Return the list of pip packages installed based on the command output

get_pip_workdir(*file_path: str | None*) → str | None

Derive the working directory for pip install when resolving relative paths in requirements (e.g. -e ./local_pkg).

Returns None for URLs or when no usable path is provided.

pip_subprocess(*args*, *prefix*, *cwd*)

Run pip in a subprocess

get_pip_installed_packages(*stdout*)

Return the list of pip packages installed based on the command output

specs

cep_24

Define cep-0024 compliant YAML spec.

Classes

Cep24YamlFileSpec	**EXPERIMENTAL**

class Cep24YamlFileSpec(*filename: str | None = None*, ***kwargs*)

Bases: *conda.plugins.types.EnvironmentSpecBase*

EXPERIMENTAL

Base class for all environment specifications.

Environment specs parse different types of environment definition files (environment.yml, requirements.txt, pyproject.toml, etc.) into a common Environment object model.

_environment = None

filename = None

can_handle()

Validates loader can process environment definition. This can handle if:

- the provided file exists
- the provided file is compliant with the CEP-0024

Returns
True if the file can be handled

Raises
Exception -- if the file can not be handled

property env: *conda.models.environment.Environment*

Express the provided environment file as a conda environment object.

Returns
the conda environment represented by the file.

explicit

Define explicit spec.

Classes

ExplicitSpec	The ExplicitSpec class handles explicit environment files. These are ones

class ExplicitSpec(*filename: str | None = None*, ***kwargs*)

Bases: *conda.plugins.types.EnvironmentSpecBase*

The ExplicitSpec class handles explicit environment files. These are ones which are marked with the @EXPLICIT marker.

Initialize the explicit specification.

Parameters

- **filename** -- Path to the requirements file
- **kwargs** -- Additional arguments

filename = None

can_handle() → bool

Validates that this spec can process the environment definition. This checks if:

- a filename was provided
- the file has the "@EXPLICIT" marker

Returns
True if the file can be handled, False otherwise

property env: *conda.models.environment.Environment*

Build an environment from the explicit file.

Returns
An Environment object containing the package specifications

Raises
`ValueError` -- If the file cannot be read

`requirements`

Define requirements.txt spec.

Classes

RequirementsSpec	Reads dependencies from requirements files (including explicit files)

class RequirementsSpec(*filename: str | None = None*, ***kwargs*)

Bases: *conda.plugins.types.EnvironmentSpecBase*

Reads dependencies from requirements files (including explicit files) and returns an Environment object from it.

Initialize the requirements specification.

Parameters

- **filename** -- Path to the requirements file
- **kwargs** -- Additional arguments

msg: str | None = None

filename = None

can_handle() → bool

Validates that this spec can process the environment definition. This checks if:

- a filename was provided
- the file exists
- the file content is valid for this specifier type

Returns
True if the file can be handled, False otherwise

property env: *conda.models.environment.Environment*

Build an environment from the requirements file.

Returns
An Environment object containing the package specifications

Raises
`ValueError` -- If the file cannot be read

`yaml_file`

Define YAML spec.

Classes

YamlFileSpec	**EXPERIMENTAL**

class YamlFileSpec(*filename=None*, ***kwargs*)

Bases: *conda.plugins.types.EnvironmentSpecBase*

EXPERIMENTAL

Base class for all environment specifications.

Environment specs parse different types of environment definition files (environment.yml, requirements.txt, pyproject.toml, etc.) into a common Environment object model.

detection_supported = False

_environment = None

filename = None

msg = None

can_handle()

Validates loader can process environment definition. This can handle if:

- the provided file exists
- the yaml file can be loaded and is not empty

Returns
True or False

property env: *conda.models.environment.Environment*

Express the provided environment file as a conda environment object.

Returns
the conda environment represented by the file.

Functions

detect([filename]) → SpecTypes	Return the appropriate spec type to use.

Attributes

context
FileSpecTypes

context

exception EnvironmentSpecPluginNotDetected(*plugin_specs: list[conda.plugins.types.CondaEnvironmentSpecifier]*, **args*, ***kwargs*)

Bases: *SpecNotFound*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception SpecNotFound(*msg: str*, **args*, ***kwargs*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

FileSpecTypes

detect(*filename: str | None = None*) → SpecTypes

Return the appropriate spec type to use.

Raises

SpecNotFound -- Raised if no suitable spec class could be found given the input

exception_handler

Error handling and error reporting.

Classes

ExceptionHandler

Functions

conda_exception_handler(func, *args, **kwargs) → T | int

Attributes

T

T

class ExceptionHandler

__call__(*func: collections.abc.Callable[*P, T*]*, **args:* P, ***kwargs:* P) → *T* | int

write_out(**content: str*) → None

property http_timeout

property user_agent

property error_upload_url

handle_exception(*exc_val: BaseException*, *exc_tb: types.TracebackType*) → int

handle_application_exception(*exc_val: BaseException*, *exc_tb: types.TracebackType*) → int

_print_conda_exception(*exc_val: BaseException*, *exc_tb: types.TracebackType*) → None

handle_unexpected_exception(*exc_val: BaseException*, *exc_tb: types.TracebackType*) → int

handle_reportable_application_exception(*exc_val: BaseException*, *exc_tb: types.TracebackType*) → int

get_error_report(*exc_val: BaseException*, *exc_tb: types.TracebackType*) → dict[str, str]

print_unexpected_error_report(*error_report: dict[str, str]*) → None

print_expected_error_report(*error_report: dict[str, str]*) → None

property _isatty: bool

_upload(*error_report: dict[str, str]*) → None

Determine whether or not to upload the error report.

_ask_upload() → bool

_execute_upload(*error_report: dict[str, Any]*) → None

_post_upload(*do_upload: bool*) → None

conda_exception_handler(*func: collections.abc.Callable[..., T]*, **args*, ***kwargs*) → *T* | int

exceptions

Conda exceptions.

Functions

format_env_spec_available_plugins(plugin_specs ...])	
maybe_raise(error, context)	
_get_errors(exc_val) → collections.abc.Iterable[BaseException]	Yield non-container errors, flattening nested ``CondaMultiError``s.
_get_guidance(error) → conda._private.exception_guidance.ErrorGuidance | None	Combine core guidance with plugin hints for a leaf `CondaError`.
_format_leaf_errors(leaves) → str	Format each leaf error for human output with its own guidance.
_json_error_map(error) → dict[str, Any]	Build JSON error map with print-time guidance (leaf errors only).
print_conda_exception(exc_val[, exc_tb])	
_format_exc([exc_val, exc_tb])	

Attributes

UnsatisfiableConflictEntry

UnsatisfiableConflictEntry

exception ResolvePackageNotFound(*bad_deps: collections.abc.Iterable[collections.abc.Iterable[*conda.models.match_spec.MatchSpec*]]*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

bad_deps

_formatted_chains

exception LockError(*message: str*)

Bases: *conda.CondaError*, OSError

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception ArgumentError(*message: str*, ***kwargs*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

return_code = 2

exception Help(*message: str | None*, **args: Any*, *caused_by: Any = None*, *guidance:* _private.exception_guidance.ErrorGuidance | *_private.exception_guidance.ErrorGuidanceTypedDict | None = None*, ***kwargs*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

return_code = 0

exception ActivateHelp

Bases: *Help*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception DeactivateHelp

Bases: *Help*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception GenericHelp(*command: str*)

Bases: *Help*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception CondaSignalInterrupt(*signum: int*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception TooManyArgumentsError(*expected: int*, *received: int*, *offending_arguments: collections.abc.Iterable[str]*, *optional_message: str = ''*, **args*)

Bases: *ArgumentError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

expected

received

offending_arguments

optional_message = ''

exception ClobberError(*message: str*, *path_conflict:* conda.base.constants.PathConflict, ***kwargs*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

path_conflict

__repr__()

Return repr(self).

exception BasicClobberError(*source_path: conda.common.path.PathType*, *target_path: conda.common.path.PathType*, *context:* conda.base.context.Context)

Bases: *ClobberError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception KnownPackageClobberError(*target_path: conda.common.path.PathType*, *colliding_dist_being_linked:* conda.models.records.PackageRecord | *str*, *colliding_linked_dist:* conda.models.records.PackageRecord | *str*, *context:* conda.base.context.Context)

Bases: *ClobberError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception UnknownPackageClobberError(*target_path: conda.common.path.PathType*, *colliding_dist_being_linked:* conda.models.records.PackageRecord | *str*, *context:* conda.base.context.Context)

Bases: *ClobberError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception SharedLinkPathClobberError(*target_path: conda.common.path.PathType*, *incompatible_package_dists: collections.abc.Iterable[*conda.models.records.PackageRecord | *str]*, *context:* conda.base.context.Context)

Bases: *ClobberError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception CommandNotFoundError(*command: str*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception PathNotFoundError(*path: conda.common.path.PathType*)

Bases: *conda.CondaError*, OSError

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception DirectoryNotFoundError(*path: conda.common.path.PathType*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception EnvironmentLocationNotFound(*location: conda.common.path.PathType*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception EnvironmentNameNotFound(*environment_name: str*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception NoBaseEnvironmentError

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception DirectoryNotACondaEnvironmentError(*target_directory: conda.common.path.PathType*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception CondaEnvironmentError(*message: str*, **args*)

Bases: *conda.CondaError*, OSError

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception DryRunExit

Bases: *conda.CondaExitZero*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception CondaSystemExit(**args*)

Bases: *conda.CondaExitZero*, SystemExit

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception PaddingError(*dist: str*, *placeholder: str*, *placeholder_length: int*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception LinkError(*message: str*, ***kwargs*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception LinkSourceNotFoundError(*src: str*, ***kwargs*)

Bases: *conda.CondaError*

Raised when the source of a link does not exist.

Carries the missing `src` path so callers can remediate without parsing the error message. Unlike *PathNotFoundError*, the source is expected to be an already-extracted package file, so a missing source usually points at a corrupt package cache entry to clean up.

Raised by *create_link()*, which is not package-cache specific, so the `conda clean --packages` advice in the message does not fit every caller.

Initialize self. See help(type(self)) for accurate signature.

src

exception LinkScriptError(*message: str*, *action: str*, *prec:* conda.models.records.PackageRecord, *path: str*, ***kwargs*)

Bases: *LinkError*

Raised when a pre-link or post-link script exits non-zero.

Carries structured information about the failure (the script's `action`, the `prec` it ran for and the `path` of the script) so callers can dispatch on the type and fields instead of parsing the message.

Failing unlink scripts only log a warning and report failure by returning `False`, so `action` is always `"pre-link"` or `"post-link"`.

Initialize self. See help(type(self)) for accurate signature.

action

prec

path

exception CondaOSError(*message: str*, ***kwargs*)

Bases: *conda.CondaError*, OSError

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception ProxyError(*message: str | None = None*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception CondaIOError(*message: str*, **args*)

Bases: *conda.CondaError*, OSError

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception CondaFileIOError(*filepath: conda.common.path.PathType*, *message: str*, **args*)

Bases: *CondaIOError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

filepath

exception CondaKeyError(*key: Any*, *message: str*, **args*)

Bases: *conda.CondaError*, KeyError

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

key

msg

exception ChannelError(*message: str | None*, **args: Any*, *caused_by: Any = None*, *guidance: _private.exception_guidance.ErrorGuidance | _private.exception_guidance.ErrorGuidanceTypedDict | None = None*, ***kwargs*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception ChannelNotAllowed(*channel:* conda.models.channel.Channel)

Bases: *ChannelError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

warning = 'Channel not included in allowlist'

exception ChannelDenied(*channel:* conda.models.channel.Channel)

Bases: *ChannelNotAllowed*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

warning = 'Channel included in denylist'

exception UnavailableInvalidChannel(*channel: conda.models.channel.Channel | str*, *status_code: str | int*, *response: requests.models.Response | None = None*, *shards_only_hint: bool = False*)

Bases: *ChannelError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

status_code: str | int

exception OperationNotAllowed(*message: str*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception CondaImportError(*message: str*)

Bases: *conda.CondaError*, ImportError

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception ParseError(*message: str*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception CouldntParseError(*reason: str*)

Bases: *ParseError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

reason

exception ChecksumMismatchError(*url: str*, *target_full_path: conda.common.path.PathType*, *checksum_type: str*, *expected_checksum: str*, *actual_checksum: str*, *partial_download: bool = False*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception CondaHTTPError(*message: str*, *url: str*, *status_code: int | str*, *reason: str*, *elapsed_time: datetime.timedelta | str*, *response: requests.Response | None = None*, *caused_by: Any = None*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception CondaSSLError(*message: str | None*, **args: Any*, *caused_by: Any = None*, *guidance: _private.exception_guidance.ErrorGuidance | _private.exception_guidance.ErrorGuidanceTypedDict | None = None*, ***kwargs*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception AuthenticationError(*message: str | None*, **args: Any*, *caused_by: Any = None*, *guidance: _private.exception_guidance.ErrorGuidance | _private.exception_guidance.ErrorGuidanceTypedDict | None = None*, ***kwargs*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception PackagesNotFoundError(*packages: collections.abc.Iterable[*conda.models.match_spec.MatchSpec | conda.models.records.PackageRecord | *str]*, *channel_urls: collections.abc.Iterable[str] = ()*)

Bases: *conda.CondaError*

Base error for missing packages.

Prefer raising one of the more specific subclasses:

- *PackagesNotFoundInChannelsError* – packages unavailable in channels.
- *PackageNotInstalledError* – packages missing from a target prefix.

Initialize self. See help(type(self)) for accurate signature.

packages

channel_urls = ()

exception PackagesNotFoundInChannelsError(*packages: collections.abc.Iterable[*conda.models.match_spec.MatchSpec | conda.models.records.PackageRecord | *str]*, *channel_urls: collections.abc.Iterable[str]*)

Bases: *PackagesNotFoundError*

The requested packages are not available from the configured channels.

Initialize self. See help(type(self)) for accurate signature.

exception PackageNotInstalledError(*prefix: conda.common.path.PathType*, *package_name: str | collections.abc.Iterable[*conda.models.match_spec.MatchSpec | conda.models.records.PackageRecord | *str]*)

Bases: *PackagesNotFoundError*

The requested package(s) are missing from the target environment/prefix.

Initialize self. See help(type(self)) for accurate signature.

exception PackagesNotFoundInPrefixError(*packages: collections.abc.Iterable[*conda.models.match_spec.MatchSpec | conda.models.records.PackageRecord | *str]*, *prefix: conda.common.path.PathType*)

Bases: *PackageNotInstalledError*

The requested package(s) are missing from the target environment/prefix.

This is a backwards-compatible alias for *PackageNotInstalledError*.

Initialize self. See help(type(self)) for accurate signature.

exception NoChannelsConfiguredError(*packages: collections.abc.Iterable[str] = ()*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception UnsatisfiableError(*bad_deps: UnsatisfiableConflictMap*, *chains: bool = True*, *strict: bool = False*)

Bases: *conda.CondaError*

Raised when dependency constraints cannot be satisfied.

Parameters

- **bad_deps** -- Classified resolver conflicts, or empty when hints are off.
- **chains** -- Unused; retained for call-site compatibility.
- **strict** -- Whether strict channel priority was in effect.

Initialize self. See help(type(self)) for accurate signature.

_format_chain_str(*bad_deps: list[list[str]]*) → list[str]

unsatisfiable = []

_get_guidance(*bad_deps: UnsatisfiableConflictMap*, *strict: bool = False*) → conda._private.exception_guidance.ErrorGuidanceTypedDict | None

Build guidance dict for *UnsatisfiableError*, or None.

exception RemoveError(*message: str*, ***, *guidance: conda._private.exception_guidance.ErrorGuidance | conda._private.exception_guidance.ErrorGuidanceTypedDict | None = None*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception DisallowedPackageError(*package_ref: conda.models.records.PackageRecord*, ***kwargs*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception SpecsConfigurationConflictError(*requested_specs: collections.abc.Iterable[conda.models.match_spec.MatchSpec]*, *pinned_specs: collections.abc.Iterable[conda.models.match_spec.MatchSpec]*, *prefix: conda.common.path.PathType*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception CondaIndexError(*message: str*)

Bases: *conda.CondaError*, IndexError

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception CondaValueError(*message: str*, **args*, ***kwargs*)

Bases: *conda.CondaError*, ValueError

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception PlatformMismatchError(*incompatible: collections.abc.Iterable[tuple[str, collections.abc.Iterable[str]]]*, *subdir: str*)

Bases: *CondaValueError*

Raised when one or more environment specs do not cover the requested platform.

The message is derived from a list of `(source, available_platforms)` pairs so the wording stays consistent whether the failure comes from a single file (`conda create`, `conda update`) or several (`Environment.from_cli` with multiple `-f` / `--file` arguments).

Initialize self. See help(type(self)) for accurate signature.

exception CyclicalDependencyError(*packages_with_cycles: collections.abc.Iterable[conda.models.records.PackageRecord]*, ***kwargs*)

Bases: *conda.CondaError*, ValueError

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception CorruptedEnvironmentError(*environment_location: conda.common.path.PathType*, *corrupted_file: conda.common.path.PathType*, ***kwargs*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception CondaHistoryError(*message: str*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception CondaUpgradeError(*message: str*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception CondaVerificationError(*message: str*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception SafetyError(*message: str*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception CondaMemoryError(*caused_by: Any*, ***kwargs*)

Bases: *conda.CondaError*, MemoryError

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception NotWritableError(*path: conda.common.path.PathType*, *errno: int*, ***kwargs*)

Bases: *conda.CondaError*, OSError

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

errno

exception NoWritableEnvsDirError(*envs_dirs: collections.abc.Iterable[conda.common.path.PathType]*, ***kwargs*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception NoWritablePkgsDirError(*pkgs_dirs: collections.abc.Iterable[conda.common.path.PathType]*, ***kwargs*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception EnvironmentIsFrozenError(*prefix: conda.common.path.PathType*, *message: str = ''*, ***kwargs*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception EnvironmentNotWritableError(*environment_location: conda.common.path.PathType*, ***kwargs*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception CondaDependencyError(*message: str*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception BinaryPrefixReplacementError(*path: conda.common.path.PathType*, *placeholder: str*, *new_prefix: conda.common.path.PathType*, *original_data_length: int*, *new_data_length: int*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception InvalidSpec(*message: str*, ***kwargs*)

Bases: *conda.CondaError*, ValueError

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception InvalidVersionSpec(*invalid_spec: str* | conda.models.match_spec.MatchSpec, *details: str*)

Bases: *InvalidSpec*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception InvalidMatchSpec(*invalid_spec: str* | conda.models.match_spec.MatchSpec, *details: str*)

Bases: *InvalidSpec*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception EncodingError(*caused_by: Any*, ***kwargs*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception NoSpaceLeftError(*caused_by: Any*, ***kwargs*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception CondaEnvException(*message: str*, **args*, ***kwargs*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception EnvironmentFileInvalid(*msg: str*, **args*, ***kwargs*)

Bases: *CondaEnvException*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception EnvironmentFileNotFound(*filename: conda.common.path.PathType*, **args*, ***kwargs*)

Bases: *CondaEnvException*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

filename

exception EnvironmentFileExtensionNotValid(*filename: conda.common.path.PathType*, **args*, ***kwargs*)

Bases: *CondaEnvException*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

filename

exception EnvironmentFileTypeMismatchError(*file_types: dict[str, str]*, **args*, ***kwargs*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception EnvironmentFileEmpty(*filename: conda.common.path.PathType*, **args*, ***kwargs*)

Bases: *CondaEnvException*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

filename

exception EnvironmentFileNotDownloaded(*username: str*, *packagename: str*, **args*, ***kwargs*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

username

packagename

exception PluginError(*message: str | None*, **args: Any*, *caused_by: Any = None*, *guidance: _private.exception_guidance.ErrorGuidance | _private.exception_guidance.ErrorGuidanceTypedDict | None = None*, ***kwargs*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception SpecNotFound(*msg: str*, **args*, ***kwargs*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

format_env_spec_available_plugins(*plugin_specs: list[conda.plugins.types.CondaEnvironmentSpecifier]*, *formats_header: str = 'Available formats'*, *indent: int = 4*, *cmd_suggestion: bool = True*)

exception AmbiguousEnvironmentSpecPlugin(*msg: str*, *plugins: list[conda.plugins.types.CondaEnvironmentSpecifier]*, **args*, ***kwargs*)

Bases: *PluginError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception EnvironmentSpecPluginSelectionError(*msg: str*, *plugin_specs: list[conda.plugins.types.CondaEnvironmentSpecifier]*, **args*, ***kwargs*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception EnvironmentSpecPluginNotDetected(*plugin_specs: list[conda.plugins.types.CondaEnvironmentSpecifier]*, **args*, ***kwargs*)

Bases: *SpecNotFound*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception EnvironmentExporterNotDetected(*filename: str*, *exporters: collections.abc.Iterable[conda.plugins.types.CondaEnvironmentExporter]*, **args*, ***kwargs*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

filename

exception SpecNotFoundInPackageCache(*msg: str*, **args*, ***kwargs*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

maybe_raise(*error: BaseException*, *context:* conda.base.context.Context)

_get_errors(*exc_val: BaseException*) → collections.abc.Iterable[BaseException]

Yield non-container errors, flattening nested ``CondaMultiError``s.

_get_guidance(*error:* conda.CondaError) → *conda._private.exception_guidance.ErrorGuidance* | None

Combine core guidance with plugin hints for a leaf `CondaError`.

_format_leaf_errors(*leaves: collections.abc.Iterable[BaseException]*) → str

Format each leaf error for human output with its own guidance.

_json_error_map(*error: BaseException*) → dict[str, Any]

Build JSON error map with print-time guidance (leaf errors only).

print_conda_exception(*exc_val:* conda.CondaError, *exc_tb: types.TracebackType* | *None* = *None*)

_format_exc(*exc_val: BaseException* | *None* = *None*, *exc_tb: types.TracebackType* | *None* = *None*)

exception InvalidInstaller(*name: str*, ***, *file: str* | *None* = *None*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception OfflineError(*message: str | None*, **args: Any*, *caused_by: Any = None*, *guidance: _private.exception_guidance.ErrorGuidance | _private.exception_guidance.ErrorGuidanceTypedDict | None = None*, ***kwargs*)

Bases: *conda.CondaError*, RuntimeError

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception CondaUpdatePackageError(*spec: str | list[str]*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exports

Backported exports for conda-build.

Functions

_resolve()	
get_default_urls()	
rm_rf(path[, max_retries, trash])	
get_index([channel_urls, prepend, platform, ...])	
package_cache()	
linked_data(prefix[, ignore_channels])	Return a dictionary of the linked packages in prefix.
linked(prefix[, ignore_channels])	Return the Dists of linked packages in prefix.
is_linked(prefix, dist)	Return the install metadata for a linked package in a prefix, or None
download(url, dst_path[, session, md5sum, urlstxt, ...])	

Attributes

non_x86_linux_machines
arch_name
binstar_upload
bits
default_prefix
default_python
envs_dirs
pkgs_dirs
platform
root_dir
root_writable
subdir
conda_build
get_rc_urls
get_local_urls
load_condarc
PaddingError

continues on next page

Table 185 – continued from previous page

LinkError
CondaOSError

_resolve()

non_x86_linux_machines

arch_name

binstar_upload

bits

default_prefix

default_python

envs_dirs

pkgs_dirs

platform

root_dir

root_writable

subdir

conda_build

get_rc_urls

get_local_urls

load_condarc

PaddingError

LinkError

CondaOSError

get_default_urls()

rm_rf(*path*, *max_retries=5*, *trash=True*)

get_index(*channel_urls=()*, *prepend=True*, *platform=None*, *use_local=False*, *use_cache=False*, *unknown=None*, *prefix=None*)

package_cache()

linked_data(*prefix*, *ignore_channels=False*)

Return a dictionary of the linked packages in prefix.

linked(*prefix*, *ignore_channels=False*)

Return the Dists of linked packages in prefix.

is_linked(*prefix*, *dist*)

Return the install metadata for a linked package in a prefix, or None if the package is not linked in the prefix.

download(*url*, *dst_path*, *session=None*, *md5sum=None*, *urlstxt=False*, *retries=3*, *sha256=None*, *size=None*)

gateways

Gateways isolate interaction of conda code with the outside world. Disk manipulation, database interaction, and remote requests should all be through various gateways. Functions and methods in `conda.gateways` must use `conda.models` for arguments and return values.

Conda modules importable from `conda.gateways` are

- `conda.common`
- `conda.models`
- `conda.gateways`

Conda modules off limits for import within `conda.gateways` are

- `conda.api`
- `conda.cli`
- `conda.client`
- `conda.core`

Conda modules strictly prohibited from importing `conda.gateways` are

- `conda.api`
- `conda.cli`
- `conda.client`

anaconda_client

Anaconda-client (binstar) token management for CondaSession.

Functions

replace_first_api_with_conda(url)	
_get_binstar_token_directory()	
read_binstar_tokens()	
set_binstar_token(url, token)	
remove_binstar_token(url)	

replace_first_api_with_conda(*url*)

_get_binstar_token_directory()

read_binstar_tokens()

set_binstar_token(*url*, *token*)

remove_binstar_token(*url*)

connection

adapters

ftp

Defines FTP transport adapter for CondaSession (requests.Session).

Taken from requests-ftp (https://github.com/Lukasa/requests-ftp/blob/master/requests_ftp/ftp.py).

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

Classes

FTPAdapter	A Requests Transport Adapter that handles FTP urls.

Functions

_new_makepasv(self)	
data_callback_factory(variable)	Returns a callback suitable for use by the FTP library. This callback
build_text_response(request, data, code)	Build a response for textual data.
build_binary_response(request, data, code)	Build a response for data whose encoding is unknown.
build_response(request, data, code, encoding)	Builds a response object from the data returned by ftplib, using the
get_status_code_from_code_response(code)	Handle complicated code response, even multi-lines.

Attributes

_old_makepasv

_old_makepasv

_new_makepasv(*self*)

class FTPAdapter

Bases: `conda.gateways.connection.BaseAdapter`

A Requests Transport Adapter that handles FTP urls.

func_table

send(*request*, ***kwargs*)

Sends a PreparedRequest object over FTP. Returns a response object.

close()

Dispose of any internal state.

list(*path*, *request*)

Executes the FTP LIST command on the given path.

retr(*path*, *request*)

Executes the FTP RETR command on the given path.

nlst(*path*, *request*)

Executes the FTP NLST command on the given path.

get_username_password_from_header(*request*)

Given a PreparedRequest object, reverse the process of adding HTTP Basic auth to obtain the username and password. Allows the FTP adapter to piggyback on the basic auth notation without changing the control flow.

get_host_and_path_from_url(*request*)

Given a PreparedRequest object, split the URL in such a manner as to determine the host and the path. This is a separate method to wrap some of urlparse's craziness.

data_callback_factory(*variable*)

Returns a callback suitable for use by the FTP library. This callback will repeatedly save data into the variable provided to this function. This variable should be a file-like structure.

build_text_response(*request*, *data*, *code*)

Build a response for textual data.

build_binary_response(*request*, *data*, *code*)

Build a response for data whose encoding is unknown.

build_response(*request*, *data*, *code*, *encoding*)

Builds a response object from the data returned by ftplib, using the specified encoding.

get_status_code_from_code_response(*code*)

Handle complicated code response, even multi-lines.

We get the status code in two ways: - extracting the code from the last valid line in the response - getting it from the 3 first digits in the code After a comparison between the two values, we can safely set the code or raise a warning. .. rubric:: Examples

- get_status_code_from_code_response('200 Welcome') == 200
- multi_line_code = '226-File successfully transferredn226 0.000 seconds' get_status_code_from_code_response(multi_line_code) == 226
- multi_line_with_code_conflicts = '200-File successfully transferredn226 0.000 seconds' get_status_code_from_code_response(multi_line_with_code_conflicts) == 226

For more detail see RFC 959, page 36, on multi-line responses:
https://www.ietf.org/rfc/rfc959.txt "Thus the format for multi-line replies is that the first line

> will begin with the exact required reply code, followed immediately by a Hyphen, "-" (also known as Minus), followed by text. The last line will begin with the same code, followed immediately by Space <SP>, optionally some text, and the Telnet end-of-line code."

http

Defines HTTP transport adapter for CondaSession (requests.Session).

Closely derived from pip:

https://github.com/pypa/pip/blob/8c24fd2a80bad21aa29aec02fb48bd89a1e8f5e1/src/pip/_internal/network/session.py#L254

Under the MIT license:

Copyright (c) 2008-2023 The pip developers (see AUTHORS.txt file on the pip repository)

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Classes

_SSLContextAdapterMixin	Mixin to add the `ssl_context` constructor argument to HTTP adapters.
HTTPAdapter	Mixin to add the `ssl_context` constructor argument to HTTP adapters.

class _SSLContextAdapterMixin(*, *ssl_context: ssl.SSLContext | None = None*, ***kwargs: Any*)

Mixin to add the `ssl_context` constructor argument to HTTP adapters.

The additional argument is forwarded directly to the pool manager. This allows us to dynamically decide what SSL store to use at runtime, which is used to implement the optional `truststore` backend.

_ssl_context = None

init_poolmanager(*connections: int*, *maxsize: int*, *block: bool = DEFAULT_POOLBLOCK*, ***pool_kwargs: Any*) → urllib3.PoolManager

proxy_manager_for(*proxy: str*, ***proxy_kwargs: Any*) → urllib3.ProxyManager

class HTTPAdapter(*, *ssl_context: ssl.SSLContext | None = None*, ***kwargs: Any*)

Bases: *_SSLContextAdapterMixin*, `conda.gateways.connection.HTTPAdapter`

Mixin to add the `ssl_context` constructor argument to HTTP adapters.

The additional argument is forwarded directly to the pool manager. This allows us to dynamically decide what SSL store to use at runtime, which is used to implement the optional `truststore` backend.

localfs

Defines local filesystem transport adapter for CondaSession (requests.Session).

Classes

LocalFSAdapter	The Base Transport Adapter

class LocalFSAdapter

Bases: conda.gateways.connection.BaseAdapter

The Base Transport Adapter

send(*request*, *stream=None*, *timeout=None*, *verify=None*, *cert=None*, *proxies=None*)

Sends PreparedRequest object. Returns Response object.

Parameters

- **request** -- The PreparedRequest being sent.
- **stream** -- (optional) Whether to stream the request content.
- **timeout** (*float or tuple*) -- (optional) How long to wait for the server to send data before giving up, as a float, or a (connect timeout, read timeout) tuple.
- **verify** -- (optional) Either a boolean, in which case it controls whether we verify the server's TLS certificate, or a string, in which case it must be a path to a CA bundle to use
- **cert** -- (optional) Any user-provided SSL certificate to be trusted.
- **proxies** -- (optional) The proxies dictionary to apply to the request.

close()

Cleans up adapter specific items.

s3

Defines S3 transport adapter for CondaSession (requests.Session).

Classes

S3Adapter	The Base Transport Adapter

class S3Adapter

Bases: conda.gateways.connection.BaseAdapter

The Base Transport Adapter

send(*request: conda.gateways.connection.PreparedRequest*, *stream: bool = False*, *timeout: None | float | tuple[float, float] | tuple[float, None] = None*, *verify: bool | str = True*, *cert: None | bytes | str | tuple[bytes | str, bytes | str] = None*, *proxies: dict[str, str] | None = None*) → conda.gateways.connection.Response

Sends PreparedRequest object. Returns Response object.

Parameters

- **request** -- The PreparedRequest being sent.
- **stream** -- (optional) Whether to stream the request content.
- **timeout** (*float or tuple*) -- (optional) How long to wait for the server to send data before giving up, as a float, or a (connect timeout, read timeout) tuple.

- **verify** -- (optional) Either a boolean, in which case it controls whether we verify the server's TLS certificate, or a string, in which case it must be a path to a CA bundle to use
- **cert** -- (optional) Any user-provided SSL certificate to be trusted.
- **proxies** -- (optional) The proxies dictionary to apply to the request.

close()

Cleans up adapter specific items.

direct_download(*url: str*, *fileobj: IO[bytes]*, *progress_callback: collections.abc.Callable[[float], None] | None = None*, *size: int | None = None*) → None

Download S3 object directly to file object using boto3's optimized download_fileobj, which handles multipart downloads and retries.

This avoids the extra copy that occurs when using send() which buffers to a SpooledTemporaryFile first.

Parameters

- **url** -- S3 URL (s3://bucket/key)
- **fileobj** -- File object to write to (must be opened in binary write mode)
- **progress_callback** -- Optional callback(fraction) where fraction is 0.0-1.0
- **size** -- Optional content length (required for progress reporting)

Raises

CondaError -- On S3 errors or if boto3 is not installed

_send_boto3(*resp: conda.gateways.connection.Response*, *request: conda.gateways.connection.PreparedRequest*) → conda.gateways.connection.Response

_write_tempfile(*writer_callable*)

download

Download logic for conda indices and packages.

Classes

TmpDownload	Context manager to handle downloads to a tempfile.

Functions

disable_ssl_verify_warning()	
download(url, target_full_path[, md5, sha256, size, ...])	
download_inner(url, target_full_path, md5, sha256, ...)	
download_partial_file(target_full_path, *, url, ...)	Create or open locked partial download file, moving onto target_full_path
download_http_errors(url)	Translate request failures into conda exceptions.
download_text(url)	

Attributes

CHUNK_SIZE

CHUNK_SIZE = 16384

disable_ssl_verify_warning()

download(*url*, *target_full_path*, *md5=None*, *sha256=None*, *size=None*, *progress_update_callback=None*)

download_inner(*url*, *target_full_path*, *md5*, *sha256*, *size*, *progress_update_callback*)

download_partial_file(*target_full_path: str | pathlib.Path*, *, *url: str*, *sha256: str*, *md5: str*, *size: int*)

Create or open locked partial download file, moving onto target_full_path when finished. Preserve partial file on exception.

download_http_errors(*url: str*)

Translate request failures into conda exceptions.

download_text(*url*)

class TmpDownload(*url*, *verbose=True*)

Context manager to handle downloads to a tempfile.

url

verbose = True

__enter__()

__exit__(*exc_type*, *exc_value*, *traceback*)

session

Requests session configured with all accepted scheme adapters.

Classes

EnforceUnusedAdapter	The Base Transport Adapter
CondaSessionType	Takes advice from https://github.com/requests/requests/issues/1871#issuecomment-33327847
CondaSession	A Requests session.
CondaHttpAuth	Base class that all auth implementations derive from

Functions

get_channel_name_from_url(url) → str | None	Given a URL, determine the channel it belongs to and return its name.
_validate_plugin_headers(headers) → None	Validate plugin-provided headers and raise PluginError if any forbidden

continues on next page

Table 197 – continued from previous page

get_session(url)	Function that determines the correct Session object to be returned
get_session_storage_key(auth) → str	Function that determines which storage key to use for our CondaSession object caching

Attributes

RETRIES
CONDA_SESSION_SCHEMES
FORBIDDEN_HEADERS
FORBIDDEN_METHOD_OVERRIDE_HEADERS
FORBIDDEN_HTTP_METHODS

RETRIES = 3

CONDA_SESSION_SCHEMES

FORBIDDEN_HEADERS

FORBIDDEN_METHOD_OVERRIDE_HEADERS

FORBIDDEN_HTTP_METHODS

class EnforceUnusedAdapter

Bases: `conda.gateways.connection.BaseAdapter`

The Base Transport Adapter

send(*request: requests.models.Request*, **args*, ***kwargs*)

Sends PreparedRequest object. Returns Response object.

Parameters

- **request** -- The `PreparedRequest` being sent.
- **stream** -- (optional) Whether to stream the request content.
- **timeout** (*float or tuple*) -- (optional) How long to wait for the server to send data before giving up, as a float, or a (connect timeout, read timeout) tuple.
- **verify** -- (optional) Either a boolean, in which case it controls whether we verify the server's TLS certificate, or a string, in which case it must be a path to a CA bundle to use
- **cert** -- (optional) Any user-provided SSL certificate to be trusted.
- **proxies** -- (optional) The proxies dictionary to apply to the request.

abstractmethod close()

Cleans up adapter specific items.

get_channel_name_from_url(*url: str*) → str | None

Given a URL, determine the channel it belongs to and return its name.

_validate_plugin_headers(*headers: dict*) → None

Validate plugin-provided headers and raise PluginError if any forbidden headers are detected. This prevents plugins from compromising conda's core network stack.

Parameters
headers -- Dictionary of headers to validate

Raises
PluginError -- If any forbidden headers are detected

get_session(*url: str*)

Function that determines the correct Session object to be returned based on the URL that is passed in.

get_session_storage_key(*auth*) → str

Function that determines which storage key to use for our CondaSession object caching

class CondaSessionType

Bases: type

Takes advice from https://github.com/requests/requests/issues/1871#issuecomment-33327847 and creates one Session instance per thread.

__call__(***kwargs*)

class CondaSession(*auth: conda.gateways.connection.AuthBase | tuple[str, str] | None = None*)

Bases: conda.gateways.connection.Session

A Requests session.

Provides cookie persistence, connection-pooling, and configuration.

Basic Usage:

```
>>> import requests
>>> s = requests.Session()
>>> s.get('https://httpbin.org/get')
<Response [200]>
```

Or as a context manager:

```
>>> with requests.Session() as s:
...     s.get('https://httpbin.org/get')
<Response [200]>
```

Parameters
auth -- Optionally provide requests.AuthBase compliant objects

auth

prepare_request(*request: requests.models.Request*) → requests.models.PreparedRequest

Constructs a PreparedRequest for transmission and returns it. The PreparedRequest has settings merged from the Request instance and those of the Session.

Parameters
request -- Request instance to prepare with this session's settings.

Return type
requests.PreparedRequest

classmethod cache_clear()

class CondaHttpAuth

Bases: conda.gateways.connection.AuthBase

Base class that all auth implementations derive from

__call__(*request*)

static _apply_basic_auth(*request*)

static add_binstar_token(*url*)

static handle_407(*response*, ***kwargs*)

Prompts the user for the proxy username and password and modifies the proxy in the session object to include it.

This method is modeled after

- requests.auth.HTTPDigestAuth.handle_401()
- requests.auth.HTTPProxyAuth
- the previous conda.fetch.handle_proxy_407()

It both adds 'username:password' to the proxy URL, as well as adding a 'Proxy-Authorization' header. If any of this is incorrect, please file an issue.

Classes

DirectDownloadAdapter	Protocol for adapters that support optimized direct-to-file downloads.

class DirectDownloadAdapter

Bases: Protocol

Protocol for adapters that support optimized direct-to-file downloads.

Adapters implementing this protocol can bypass the standard streaming path and write directly to a file object, avoiding intermediate buffering.

direct_download(*url: str*, *fileobj: IO[bytes]*, *progress_callback: collections.abc.Callable[[float], None] | None = None*, *size: int | None = None*) → None

Download directly to a file object.

Parameters

- **url** -- URL to download from
- **fileobj** -- File object to write to (binary write mode)
- **progress_callback** -- Optional callback(fraction) where fraction is 0.0-1.0
- **size** -- Content length (required for progress reporting)

disk

create

Disk utility functions for creating new files or directories.

Classes

TemporaryDirectory	Create and return a temporary directory. This has the same

Functions

write_as_json_to_file(file_path, obj)	
create_python_entry_point(target_full_path, ...)	
extract_tarball(tarball_full_path[, ...])	
make_menu(prefix, file_path[, remove])	Create cross-platform menu items (e.g. Windows Start Menu)
create_hard_link_or_copy(src, dst)	
_is_unix_executable_using_ORIGIN(path)	
_do_softlink(src, dst)	
copy(src, dst)	
_do_copy(src, dst)	
create_link(src, dst[, link_type, force, fallback_copy])	
compile_multiple_pyc(python_exe_full_path, ...)	
create_package_cache_directory(pkgs_dir)	
create_envs_directory(envs_dir)	
first_writable_envs_dir([create])	

Attributes

mkdir_p
python_entry_point_template
application_entry_point_template

class TemporaryDirectory(*suffix=''*, *prefix='tmp'*, *dir=None*)

Create and return a temporary directory. This has the same behavior as mkdtemp but can be used as a context manager. For .. rubric:: Example

with TemporaryDirectory() as tmpdir:

...

Upon exiting the context, the directory and everything contained in it are removed.

name = None

_closed = False

__repr__()

__enter__()

cleanup(*_warn=False*, *_warnings=_warnings*)

__exit__(*exc*, *value*, *tb*)

__del__()

mkdir_p

python_entry_point_template = ''

application_entry_point_template = ''

write_as_json_to_file(*file_path*, *obj*)

create_python_entry_point(*target_full_path*, *python_full_path*, *module*, *func*)

extract_tarball(*tarball_full_path*, *destination_directory=None*, *progress_update_callback=None*)

make_menu(*prefix*, *file_path*, *remove=False*)

Create cross-platform menu items (e.g. Windows Start Menu)

Passes all menu config files %PREFIX%/Menu/*.json to `menuinst.install`. `remove=True` will remove the menu items.

create_hard_link_or_copy(*src*, *dst*)

_is_unix_executable_using_ORIGIN(*path*)

_do_softlink(*src*, *dst*)

copy(*src*, *dst*)

_do_copy(*src*, *dst*)

create_link(*src*, *dst*, *link_type=LinkType.hardlink*, *force=False*, *fallback_copy=True*)

compile_multiple_pyc(*python_exe_full_path*, *py_full_paths*, *pyc_full_paths*, *prefix*, *py_ver*)

create_package_cache_directory(*pkgs_dir*)

create_envs_directory(*envs_dir*)

first_writable_envs_dir(*create=True*)

delete

Disk utility functions for deleting files and folders.

Functions

rmtree(path)	
unlink_or_rename_to_trash(path)	If files are in use, especially on windows, we can't remove them.
remove_empty_parent_paths(path)	
rm_rf(path[, clean_empty_parents]) → bool	Completely delete path
delete_trash(prefix)	
backoff_rmdir(dirpath[, max_tries])	
path_is_clean(path)	Sometimes we can't completely remove a path because files are considered in use

rmtree(*path*)

unlink_or_rename_to_trash(*path*)

If files are in use, especially on windows, we can't remove them. The fallback path is to rename them (but keep their folder the same), which maintains the file handle validity. See comments at: https://serverfault.com/a/503769

remove_empty_parent_paths(*path*)

rm_rf(*path: str | os.PathLike*, *clean_empty_parents: bool = False*) → bool

Completely delete path max_retries is the number of times to retry on failure. The default is 5. This only applies to deleting a directory. If removing path fails and trash is True, files will be moved to the trash directory.

delete_trash(*prefix*)

backoff_rmdir(*dirpath*, *max_tries=MAX_TRIES*)

path_is_clean(*path*)

Sometimes we can't completely remove a path because files are considered in use by python (hardlinking confusion). For our tests, it is sufficient that either the folder doesn't exist, or nothing but temporary file copies are left.

link

Disk utility functions for symlinking files and folders.

Portions of the code within this module are taken from https://github.com/jaraco/jaraco.windows which is MIT licensed by Jason R. Coombs.

https://github.com/jaraco/skeleton/issues/1#issuecomment-285448440

lchmod

link

islink

lock

Record locking to manage potential repodata / repodata metadata file contention between conda processes. Try to acquire a lock on a single byte in the metadat file; modify both files; then release the lock.

Functions

_lock_noop(fd, lock_attempts)	When locking is not available.
_lock_impl(fd, lock_attempts)	
lock(fd, *[, lock_attempts])	
locking_supported()	Return a bool to report whether file locking is supported or not

Attributes

LOCK_BYTE
LOCK_ATTEMPTS
LOCK_SLEEP

LOCK_BYTE = 21

LOCK_ATTEMPTS = 10

LOCK_SLEEP = 1

_lock_noop(*fd*, *lock_attempts*)

When locking is not available.

_lock_impl(*fd*, *lock_attempts*)

lock(*fd*, *, *lock_attempts=LOCK_ATTEMPTS*)

locking_supported()

Return a bool to report whether file locking is supported or not

permissions

Disk utility functions for modifying file and directory permissions.

Functions

make_writable(path)	
make_read_only(path)	
recursive_make_writable(path[, max_tries])	
make_executable(path)	
is_executable(path)	

make_writable(*path*)

make_read_only(*path*)

recursive_make_writable(*path*, *max_tries=MAX_TRIES*)

make_executable(*path*)

is_executable(*path*)

read

Disk utility functions for reading and processing file contents.

Functions

yield_lines(path)	Generator function for lines in file. Empty generator if path does not exist.
compute_sum(path, algo) → str	
read_package_info(record, package_cache_record)	
read_index_json(extracted_package_directory)	
read_index_json_from_tarball(package_tarball_ful	
read_repodata_json(extracted_package_directory)	
read_icondata(extracted_package_directory)	
read_package_metadata(extracted_package_directory	
read_paths_json(extracted_package_directory)	

continues on next page

Table 207 – continued from previous page

read_has_prefix(path)	Reads *has_prefix* file and return dict mapping filepaths to tuples(placeholder, FileMode).
read_no_link(info_dir)	
read_soft_links(extracted_package_directory, files)	

Attributes

listdir

listdir

yield_lines(*path*)

Generator function for lines in file. Empty generator if path does not exist.

Parameters
path (*str*) -- path to file

Returns
each line in file, not starting with '#'

Return type
iterator

compute_sum(*path: str | os.PathLike*, *algo: Literal['md5', 'sha256']*) → str

read_package_info(*record*, *package_cache_record*)

read_index_json(*extracted_package_directory*)

read_index_json_from_tarball(*package_tarball_full_path*)

read_repodata_json(*extracted_package_directory*)

read_icondata(*extracted_package_directory*)

read_package_metadata(*extracted_package_directory*)

read_paths_json(*extracted_package_directory*)

read_has_prefix(*path*)

Reads *has_prefix* file and return dict mapping filepaths to tuples(placeholder, FileMode).

A line in *has_prefix* contains one of:

- filepath
- placeholder mode filepath

Mode values are one of:

- text
- binary

read_no_link(*info_dir*)

read_soft_links(*extracted_package_directory*, *files*)

test

Disk utility functions testing path properties (e.g., writable, hardlinks, softlinks, etc.).

Functions

file_path_is_writable(path) → bool	
paths_on_same_device(left, right) → bool	Return whether two paths reside on the same filesystem device.
hardlink_supported(source_file, dest_dir)	
softlink_supported(source_file, dest_dir)	

file_path_is_writable(*path*) → bool

paths_on_same_device(*left: str | os.PathLike[str]*, *right: str | os.PathLike[str]*) → bool

Return whether two paths reside on the same filesystem device.

hardlink_supported(*source_file*, *dest_dir*)

softlink_supported(*source_file*, *dest_dir*)

update

Disk utility functions for modifying existing files or directories.

Functions

update_file_in_place_as_binary(file_full_path, callback)	
rename(source_path, destination_path[, force])	
rename_context(source[, destination, dry_run])	Used for removing a directory when there are dependent actions (i.e. you need to ensure
backoff_rename(source_path, destination_path[, force])	
touch(path[, mkdir, sudo_safe])	

Attributes

SHEBANG_REGEX

SHEBANG_REGEX

exception CancelOperation

Bases: Exception

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

update_file_in_place_as_binary(*file_full_path*, *callback*)

rename(*source_path*, *destination_path*, *force=False*)

rename_context(*source: str*, *destination: str | None = None*, *dry_run: bool = False*)

Used for removing a directory when there are dependent actions (i.e. you need to ensure other actions succeed before removing it).

Example

with rename_context(directory):
    # Do dependent actions here

backoff_rename(*source_path*, *destination_path*, *force=False*)

touch(*path*, *mkdir=False*, *sudo_safe=False*)

Functions

exp_backoff_fn(fn, *args, **kwargs)	Mostly for retrying file operations that fail on Windows due to virus scanners
mkdir_p(path)	
mkdir_p_sudo_safe(path)	

Attributes

on_win
TRACE
MAX_TRIES

on_win = False

TRACE = 5

MAX_TRIES = 7

exp_backoff_fn(*fn*, **args*, ***kwargs*)

Mostly for retrying file operations that fail on Windows due to virus scanners

mkdir_p(*path*)

mkdir_p_sudo_safe(*path*)

logging

Configure logging for conda.

Classes

TokenURLFilter	Filter instances are used to perform arbitrary filtering of LogRecords.
StdStreamHandler	Log StreamHandler that always writes to the current sys stream.

Functions

initialize_logging()
initialize_std_loggers()
set_conda_log_level([level])
set_all_logger_level([level])
set_file_logging([logger_name, level, path])
set_log_level(log_level)

Attributes

_VERBOSITY_LEVELS

_VERBOSITY_LEVELS

class TokenURLFilter(*name=''*)

Bases: logging.Filter

Filter instances are used to perform arbitrary filtering of LogRecords.

Loggers and Handlers can optionally use Filter instances to filter records as desired. The base filter class only allows events which are below a certain point in the logger hierarchy. For example, a filter initialized with "A.B" will allow events logged by loggers "A.B", "A.B.C", "A.B.C.D", "A.B.D" etc. but not "A.BB", "B.A.B" etc. If initialized with the empty string, all events are passed.

Initialize a filter.

Initialize with the name of the logger which, together with its children, will have its events allowed through the filter. If no name is specified, allow every event.

TOKEN_URL_PATTERN

TOKEN_REPLACE

filter(*record*)

Since Python 2's getMessage() is incapable of handling any strings that are not unicode when it interpolates the message with the arguments, we fix that here by doing it ourselves.

At the same time we replace tokens in the arguments which was not happening until now.

class StdStreamHandler(*sys_stream*)

Bases: logging.StreamHandler

Log StreamHandler that always writes to the current sys stream.

Parameters
sys_stream -- stream name, either "stdout" or "stderr" (attribute of module sys)

terminator = Multiline-String

```
"""
"""
```

sys_stream

__getattr__(*attr*)

emit(*record*)

Emit a record.

If a formatter is specified, it is used to format the record. The record is then written to the stream with a trailing newline. If exception information is present, it is formatted using traceback.print_exception and appended to the stream. If the stream has an 'encoding' attribute, it is used to determine how to do the output to the stream.

initialize_logging()

initialize_std_loggers()

set_conda_log_level(*level=WARNING*)

set_all_logger_level(*level=DEBUG*)

set_file_logging(*logger_name=None*, *level=DEBUG*, *path=None*)

set_log_level(*log_level: int*)

repodata

Repodata interface.

lock

Backwards compatibility import.

Moved to prevent circular imports.

zstd

Zstd interface for repodata.

Classes

ZstdRepoInterface	Helper class that provides a standard way to create an ABC using

Functions

withext(url, ext)	Change file extension in URL.
timeme(message)	Context manager for timing operations.
build_headers(json_path, state)	Build caching headers for a path and state.
download_repodata(url, json_path, session, state[, ...])	Download url to dest_path, optionally decompressing zstd.
_is_http_error_most_400_codes(e) → bool	Determine if HTTP error is a 4xx error (except 416).
request_url_zstd_state(url, state, *, session, cache, ...) → dict | None	Download .json.zst file and return parsed JSON.

withext(*url*, *ext*)

Change file extension in URL.

timeme(*message*)

Context manager for timing operations.

build_headers(*json_path: pathlib.Path*, *state:* conda.gateways.repodata.RepodataState)

Build caching headers for a path and state.

download_repodata(*url*, *json_path: pathlib.Path*, *session: conda.gateways.connection.Session*, *state:* conda.gateways.repodata.RepodataState | *None*, *is_zst=False*, *dest_path: pathlib.Path* | *None* = *None*)

Download url to dest_path, optionally decompressing zstd.

json_path: Path of old cached data (ignore etag if not exists). dest_path: Path to write new data.

_is_http_error_most_400_codes(*e: requests.HTTPError*) → bool

Determine if HTTP error is a 4xx error (except 416).

exception ZstSkip

Bases: Exception

Exception to skip zst format check.

Initialize self. See help(type(self)) for accurate signature.

request_url_zstd_state(*url*, *state:* conda.gateways.repodata.RepodataState, ***, *session: conda.gateways.connection.Session*, *cache:* conda.gateways.repodata.RepodataCache, *temp_path: pathlib.Path*) → dict | None

Download .json.zst file and return parsed JSON.

Parameters

- **url** -- URL to download from
- **state** -- Repodata state
- **session** -- Session object
- **cache** -- Repodata cache
- **temp_path** -- Temporary path to write json to

Returns
Parsed JSON or None if error

Return type
dict | None

class ZstdRepoInterface(*url: str*, *repodata_fn: str* | *None*, ***, *cache:* conda.gateways.repodata.RepodataCache, ***kwargs*)

Bases: *conda.gateways.repodata.RepoInterface*

Helper class that provides a standard way to create an ABC using inheritance.

_cache

_url

_repodata_fn

property _log

property _stderrlog

repodata(*state: dict | conda.gateways.repodata.RepodataState*) → str | None

Fetch newest repodata if necessary.

Always writes to `cache_path_json`.

repodata_parsed(*state: dict | conda.gateways.repodata.RepodataState*) → dict | None

Use this to avoid a redundant parse when repodata is updated.

When repodata is not updated, it doesn't matter whether this function or the caller reads from a file.

_repodata_state_copy(*state: dict | conda.gateways.repodata.RepodataState*) → *conda.gateways.repodata.RepodataState*

Create a copy of state to avoid modifying the caller's dict.

Classes

PackageCacheData	
Channel	Channel:
RepoInterface	Helper class that provides a standard way to create an ABC using
CondaRepoInterface	Provides an interface for retrieving repodata data from channels.
RepodataState	Load/save info file that accompanies cached *repodata.json*.
RepodataCache	Handle caching for a single repodata.json + repodata.info.json
RepodataFetch	Combine RepodataCache and RepoInterface to provide subdir_data.SubdirData()

Functions

stringify(obj[, content_max_len])	
maybe_unquote(url) → str	
get_session(url)	Function that determines the correct Session object to be returned
mkdir_p_sudo_safe(path)	
lock(fd, *[, lock_attempts])	
get_repo_interface() → type[RepoInterface]	
_add_http_value_to_dict(resp, http_key, d, dict_key)	
_shard_index_exists(subdir_url) → bool	
conda_http_errors(url, repodata_fn)	Use in a with: statement to translate requests exceptions to conda ones.
_md5_not_for_security(data)	
cache_fn_url(url[, repodata_fn])	
get_cache_control_max_age(cache_control_value)	
create_cache_dir()	

Attributes

CONDA_HOMEPAGE_URL	
REPODATA_FN	
REPODATA_SHARDS_FN	Index for sharded repodata data of a channel
context	
join_url	
deprecated	
CHECK_ALTERNATE_FORMAT_INTERVAL	
LAST_MODIFIED_KEY	
ETAG_KEY	
CACHE_CONTROL_KEY	
URL_KEY	
CACHE_STATE_SUFFIX	
FORMAT_JSON	
FORMAT_SHARDS	
FORMAT_ZST	
ERROR_SNIPPET_LENGTH	

exception CondaError(*message: str | None*, **args: Any*, *caused_by: Any = None*, *guidance:* _private.exception_guidance.ErrorGuidance | *_private.exception_guidance.ErrorGuidanceTypedDict | None = None*, ***kwargs*)

Bases: Exception

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

return_code: int = 1

reportable: bool = False

_kwargs: dict[str, Any] | None = None

_caused_by: Any = None

_guidance: *_private.exception_guidance.ErrorGuidance* | None = None

message

property guidance: *_private.exception_guidance.ErrorGuidance* | None

Structured guidance for this exception, if provided.

Returns an ErrorGuidance instance or None. The result is always the canonical type after dict coercion; callers do not need to handle raw dicts.

__repr__() → str

Return repr(self).

__str__() → str

Return str(self).

dump_map() → dict[str, Any]

stringify(*obj*, *content_max_len=0*)

CONDA_HOMEPAGE_URL: Final = 'https://conda.io'

REPODATA_FN: Final = 'repodata.json'

REPODATA_SHARDS_FN: Final = 'repodata_shards.msgpack.zst'

Index for sharded repodata data of a channel

context

join_url

maybe_unquote(*url: str*) → str

class PackageCacheData(*pkgs_dir*)

cache: dict[str, PackageCacheData]

pkgs_dir

__package_cache_records = None

__is_writable

_urls_data

insert(*package_cache_record*)

load()

reload()

get(*package_ref:* conda.models.records.PackageRecord, *default=NULL*)

remove(*package_ref*, *default=NULL*)

query(*package_ref_or_match_spec*)

iter_records()

classmethod query_all(*package_ref_or_match_spec*, *pkgs_dirs=None*)

classmethod first_writable(*pkgs_dirs=None*)

classmethod writable_caches(*pkgs_dirs=None*)

classmethod read_only_caches(*pkgs_dirs=None*)

classmethod all_caches_writable_first(*pkgs_dirs=None*)

classmethod get_all_extracted_entries()

classmethod get_entry_to_link(*package_ref*, *target_prefix=None*)

classmethod tarball_file_in_cache(*tarball_path*, *md5sum=None*, *exclude_caches=()*)

classmethod clear()

tarball_file_in_this_cache(*tarball_path*, *md5sum=None*)

property _package_cache_records

property is_writable

_check_writable()

static _clean_tarball_path_and_get_md5sum(*tarball_path*, *md5sum=None*)

_scan_for_dist_no_channel(*dist_str*)

itervalues()

values()

__repr__()

_make_single_record(*package_filename*)

static _dedupe_pkgs_dir_contents(*pkgs_dir_contents*)

deprecated

exception CondaDependencyError(*message: str*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception CondaHTTPError(*message: str*, *url: str*, *status_code: int | str*, *reason: str*, *elapsed_time: datetime.timedelta | str*, *response: requests.Response | None = None*, *caused_by: Any = None*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception CondaSSLError(*message: str | None*, **args: Any*, *caused_by: Any = None*, *guidance: _private.exception_guidance.ErrorGuidance | _private.exception_guidance.ErrorGuidanceTypedDict | None = None*, ***kwargs*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception NotWritableError(*path: conda.common.path.PathType*, *errno: int*, ***kwargs*)

Bases: *conda.CondaError*, OSError

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

errno

exception ProxyError(*message: str | None = None*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception UnavailableInvalidChannel(*channel: conda.models.channel.Channel | str*, *status_code: str | int*, *response: requests.models.Response | None = None*, *shards_only_hint: bool = False*)

Bases: ChannelError

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

status_code: str | int

class Channel(*scheme: str | None = None*, *auth: str | None = None*, *location: str | None = None*, *token: str | None = None*, *name: str | None = None*, *platform: str | None = None*, *package_filename: str | None = None*)

Channel: scheme <> auth <> location <> token <> channel <> subchannel <> platform <> package_filename

Package Spec: channel <> subchannel <> namespace <> package_name

static _reset_state() → None

scheme = None

auth = None

location = None

token = None

name = ''

platform = None

package_filename = None

property channel_location: str | None

property channel_name: str

property subdir: str | None

static from_url(*url: str*) → *Channel*

static from_channel_name(*channel_name: str*) → *Channel*

static from_value(*value: str | None*) → *Channel*

Construct a new *Channel* from a single value.

Parameters

value -- Anyone of the following forms:

***None*, or one of the special strings "<unknown>", "None:///<unknown>", or "None":**
represents the unknown channel, used for packages with unknown origin.

A URL including a scheme like `file://` or `https://`:
represents a channel URL.

A local directory path:
represents a local channel; relative paths must start with `./`.

A package file (i.e. the path to a file ending in `.conda` or `.tar.bz2`):
represents a channel for a single package

A known channel name:
represents a known channel, e.g. from the users `.condarc` file or the global configuration.

Returns
A channel object.

static make_simple_channel(*channel_alias:* Channel, *channel_url:* *str*, *name:* *str* | *None* = *None*) → *Channel*

property canonical_name: str

urls(*with_credentials:* *bool* = *False*, *subdirs:* *collections.abc.Iterable[str]* | *None* = *None*) → list[str]

Generate URLs for this channel across specified platforms.

Parameters

- **with_credentials** -- If True, include authentication credentials (token, auth) in URLs.
- **subdirs** -- Specific platform subdirs to generate URLs for. If None, uses the channel's platform (if defined) or falls back to *context.subdirs*. If this is explicitly provided, overrides any platform defined in the channel.

Examples

```
>>> channel = Channel("conda-forge")
>>> channel.urls()  # Uses context.subdirs
['https://conda.anaconda.org/conda-forge/linux-64',
'https://conda.anaconda.org/conda-forge/noarch']
```

```
>>> channel = Channel("conda-forge/linux-aarch64")
>>> channel.urls()  # Uses channel's platform
['https://conda.anaconda.org/conda-forge/linux-aarch64',
'https://conda.anaconda.org/conda-forge/noarch']
```

```
>>> channel.urls(subdirs=("osx-64", "noarch"))
['https://conda.anaconda.org/conda-forge/osx-64',
'https://conda.anaconda.org/conda-forge/noarch']
```

Returns
List of URLs for accessing this channel's specified subdirectories.

Return type
list[str]

url(*with_credentials:* *bool* = *False*) → str | None

property base_url: str | None

property base_urls: tuple[str | None, ...]

property subdir_url: str

property channels: tuple[*Channel*, ...]

__str__() → str

__repr__() → str

__eq__(*other: Any*) → bool

__hash__() → int

__nonzero__() → bool

__bool__() → bool

__json__() → dict[str, Any]

property url_channel_wtf: tuple[str | None, str]

dump() → dict[str, Any]

get_session(*url: str*)

Function that determines the correct Session object to be returned based on the URL that is passed in.

mkdir_p_sudo_safe(*path*)

lock(*fd*, *, *lock_attempts=LOCK_ATTEMPTS*)

CHECK_ALTERNATE_FORMAT_INTERVAL

LAST_MODIFIED_KEY = 'mod'

ETAG_KEY = 'etag'

CACHE_CONTROL_KEY = 'cache_control'

URL_KEY = 'url'

CACHE_STATE_SUFFIX = '.info.json'

FORMAT_JSON = 'json'

FORMAT_SHARDS = 'shards'

FORMAT_ZST = 'zst'

ERROR_SNIPPET_LENGTH = 32

exception RepodataIsEmpty(*channel:* conda.models.channel.Channel | *str*, *status_code: str* | *int*, *response: requests.models.Response* | *None* = None, *shards_only_hint: bool* = *False*)

Bases: *conda.exceptions.UnavailableInvalidChannel*

Subclass used to determine when empty repodata should be cached, e.g. for a channel that doesn't provide current_repodata.json

Initialize self. See help(type(self)) for accurate signature.

exception RepodataOnDisk

Bases: Exception

Indicate that RepoInterface.repodata() successfully wrote repodata to disk, instead of returning a string.

Initialize self. See help(type(self)) for accurate signature.

class RepoInterface

Bases: abc.ABC

Helper class that provides a standard way to create an ABC using inheritance.

repodata(*state: dict*) → str

Given a mutable state dictionary with information about the cache, return repodata.json (or current_repodata.json) as a str. This function also updates state, which is expected to be saved by the caller.

exception Response304ContentUnchanged

Bases: Exception

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

get_repo_interface() → *type*[*RepoInterface*]

class CondaRepoInterface(*url: str*, *repodata_fn: str | None*, ***kwargs*)

Bases: *RepoInterface*

Provides an interface for retrieving repodata data from channels.

_url: str

Channel URL.

_repodata_fn: str

Filename of the repodata file; defaults to value of conda.base.constants.REPODATA_FN.

repodata(*state:* RepodataState) → str | None

Given a mutable state dictionary with information about the cache, return repodata.json (or current_repodata.json) as a str. This function also updates state, which is expected to be saved by the caller.

_add_http_value_to_dict(*resp*, *http_key*, *d*, *dict_key*)

_shard_index_exists(*subdir_url: str*) → bool

conda_http_errors(*url*, *repodata_fn*)

Use in a with: statement to translate requests exceptions to conda ones.

class RepodataState(*cache_path_json: pathlib.Path | str = ''*, *cache_path_state: pathlib.Path | str = ''*, *repodata_fn=''*, *dict=None*)

Bases: collections.UserDict

Load/save info file that accompanies cached *repodata.json*.

_aliased

_strings

cache_path_json

cache_path_state

repodata_fn = ''

property mod: str

Last-Modified header or ""

property etag: str

Etag header or ""

property cache_control: str

Cache-Control header or ""

has_format(*format: str*) → tuple[bool, datetime.datetime | None]

set_has_format(*format: str*, *value: bool*)

clear_has_format(*format: str*)

Remove 'has_{format}' instead of setting to False.

should_check_format(*format: str*) → bool

Return True if named format should be attempted.

__contains__(*key: str*) → bool

__setitem__(*key: str*, *item: Any*) → None

__getitem__(*key: str*) → Any

class RepodataCache(*base*, *repodata_fn*)

Handle caching for a single repodata.json + repodata.info.json (<hex-string>*.json inside *dir*)

Avoid race conditions while loading, saving repodata.json and cache state.

Also support bytes as in repodata_shards.msgpack.zst

base: directory and filename prefix for cache, e.g. /cache/dir/abc123; writes /cache/dir/abc123.json

cache_dir

name = ''

repodata_fn

state

property cache_path_json

property cache_path_shards

property cache_path_state

Out-of-band etag and other state needed by the RepoInterface.

load(*, *state_only=False*, *binary=False*) → str | bytes

load_state(*binary=False*)

Update self.state without reading repodata.

Return self.state.

save(*data: str | bytes*)

Write data to <repodata> cache path, by calling self.replace().

replace(*temp_path: pathlib.Path*, *target=None*)

Rename path onto <repodata> path, synchronize state.

Relies on path's mtime not changing on move. *temp_path* should be adjacent to *self.cache_path_json* to be on the same filesystem.

refresh(*refresh_ns=0*)

Update access time in cache info file to indicate a HTTP 304 Not Modified response.

lock(*mode='a+'*)

Lock .info.json file. Hold lock while modifying related files.

mode: "a+" then seek(0) to write/create; "r+" to read.

stale()

Compare state refresh_ns against cache control header and context.local_repodata_ttl.

timeout()

Return number of seconds until cache times out (<= 0 if already timed out).

_shards_known() → bool

class RepodataFetch(*cache_path_base: pathlib.Path*, *channel:* conda.models.channel.Channel, *repodata_fn: str*, ***, *repo_interface_cls*)

Combine RepodataCache and RepoInterface to provide subdir_data.SubdirData() with what it needs.

Provide a variety of formats since some `RepoInterface` have to `json.loads(...)` anyway, and some clients don't need the Python data structure at all.

cache_path_base: *pathlib.Path*

channel: *conda.models.channel.Channel*

repodata_fn: *str*

url_w_subdir: *str*

url_w_credentials: *str*

repo_interface_cls: *Any*

fetch_latest_parsed() → tuple[dict, *RepodataState*]

Retrieve parsed latest or latest-cached repodata as a dict; update cache.

Returns
Repodata contents, state including cache headers

fetch_latest_path() → tuple[pathlib.Path, *RepodataState*]

Retrieve latest or latest-cached repodata; update cache.

Returns
Path to uncompressed repodata contents, RepodataState

property url_w_repodata_fn

property cache_path_json

property cache_path_state

Out-of-band etag and other state needed by the RepoInterface.

property repo_cache: *RepodataCache*

property _repo: *RepoInterface*

Changes as we mutate self.repodata_fn.

fetch_latest() → tuple[dict | str, *RepodataState*]

Return up-to-date repodata and cache information. Fetch repodata from remote if cache has expired; return cached data if cache has not expired; return stale cached data or dummy data if in offline mode.

read_cache() → tuple[str, *RepodataState*]

Read repodata from disk, without trying to fetch a fresh version.

_md5_not_for_security(*data*)

cache_fn_url(*url*, *repodata_fn=REPODATA_FN*)

get_cache_control_max_age(*cache_control_value: str | None*)

create_cache_dir()

shards

Interface to sharded repodata code.

typing

Minimal set of interfaces required to describe shards code to clients (solvers)

Classes

Shards	Base class for protocol classes.
BuildRepodataSubset	Protocol for build_repodata_subset callable.

class Shards

Bases: Protocol

Base class for protocol classes.

Protocol classes are defined as:

```
class Proto(Protocol):
    def meth(self) -> int:
        ...
```

Such classes are primarily used with static type checkers that recognize structural subtyping (static duck-typing).

For example:

```
class C:
    def meth(self) -> int:
        return 0

def func(x: Proto) -> int:
    return x.meth()

func(C())  # Passes static type check
```

See PEP 544 for details. Protocol classes decorated with @typing.runtime_checkable act as simple-minded runtime protocols that check only the presence of given attributes, ignoring their type signatures. Protocol classes can be generic, they are defined as:

```
class GenProto[T](Protocol):
    def meth(self) -> T:
        ...
```

base_url: *str*

property package_names: *collections.abc.KeysView[str]*

Return the names of all packages available in this shard collection.

__contains__(*package: str*) → bool

Check if a package is available in this shard collection.

iter_records() → collections.abc.Iterator[tuple[str, dict]]

Yield (key, record) tuples for all packages in visited shards, including v3 records.

For classic packages, *key* is the filename. For v3 packages, *key* is the shard key (not necessarily the download filename; see `record["fn"]`).

Prefer *iter_records_v3()* when section information is needed.

iter_records_v3() → collections.abc.Iterable[tuple[tuple[str, str], dict]]

Yield ((key, section), record) tuples for all packages in visited shards.

Section can be: "packages" for .tar.bz2 packages, "packages.conda" for .conda packages, "v3.whl", "v3.conda", "v3.tar.bz2" for v3 packages.

key is the same as the filename for "packages", "packages.conda" but is different from the filename for v3 packages.

class BuildRepodataSubset

Bases: `Protocol`

Protocol for build_repodata_subset callable.

This function is used by solvers to construct a minimal subset of repodata based on the root packages that might be installed and the available channels. It traverses package dependencies to discover all reachable (channel, package) tuples, which are then used by the solver to reduce search space.

__call__(*root_packages: collections.abc.Iterable[str]*, *channels: dict[str, Any]*, *algorithm: Literal['bfs', 'pipelined'] = 'pipelined'*, *repodata_version: int = 1*, *root_extras: collections.abc.Mapping[str, collections.abc.Iterable[str]] | None = None*) → dict[str, *Shards*] | None

Retrieve a minimal subset of repodata based on root packages.

Parameters

- **root_packages** -- Iterable of installed and requested package names
- **channels** -- Dictionary mapping channel URLs to Channel objects
- **algorithm** -- Traversal algorithm to use ("bfs" or "pipelined")
- **repodata_version** -- repodata format version (1 = classic, 3 = v3).
- **root_extras** -- Optional mapping of root package name to an iterable of CEP 44 extras requested for it (e.g. *{"httpx": ["cli"]}*), so that shards for its requested *extra_depends* groups are also fetched. Extras explicitly requested by dependency specs of fetched packages are followed transitively.

Returns

A dictionary mapping channel URLs to Shards objects containing the subset of packages needed, or None if shards are unavailable

Classes

RepodataSubset	Build a subset of repodata by traversing all packages that are dependencies
BuildRepodataSubset	Protocol for build_repodata_subset callable.

Functions

build_repodata_subset(root_packages, channels[, ...]) → dict[str, conda._private.shards.shards.ShardBase] | None	Retrieve all necessary information to build a repodata subset.

class RepodataSubset(*shardlikes: collections.abc.Iterable[*conda._private.shards.shards.ShardBase*]*, *spec_to_package_name: collections.abc.Callable[[str], str] = spec_to_package_name*, *repodata_version: int = 1*, *depth: int = sys.maxsize*)

Build a subset of repodata by traversing all packages that are dependencies and transitive dependencies of a root set of packages.

shardlikes: `collections.abc.Sequence[conda._private.shards.shards.ShardBase]`

DEFAULT_STRATEGY = 'pipelined'

_nodes: `dict[NodeId, Node]`

_use_only_tar_bz2: `bool`

_add_pip_as_python_dependency: `bool`

_spec_to_package_name: `collections.abc.Callable[[str], str]`

_repodata_version = 1

depth = 9223372036854775807

property node_count: `int`

Number of (channel, package) nodes discovered during traversal.

classmethod has_strategy(*strategy: str*) → bool

Return True if this class provides the named shard traversal strategy.

_expansion_specs(*node:* Node, *shard:* conda._private.shards.typing.ShardDict) → collections.abc.Iterator[tuple[str, frozenset[str]]]

Yield `(package name, extras requested by the spec)` for everything reachable from `shard` that has not been traversed for `node` yet.

On the first expansion that is the shard's `depends` plus every group in `node.requested_extras`. When the node is revisited because another dependency activated additional groups, only those new groups are traversed. Records the groups as expanded on the node.

_neighbors(*node:* Node) → collections.abc.Iterator[*Node*]

Retrieve all unvisited neighbors of a node, plus already visited neighbors that need to be revisited because this node's dependency specs activate additional extras groups on them.

Neighbors in the context are dependencies of a package.

NOTE: This method assumes that the required shards have already been retrieved from the network via batch_retrieve_from_network() before neighbors() is called. It uses visit_package() to access already-loaded shards.

_outgoing(*node:* Node)

All nodes that can be reached by this node, plus cost.

reachable(*root_packages*, *, *strategy=DEFAULT_STRATEGY*, *root_extras:* *collections.abc.Mapping[str, collections.abc.Iterable[str]] | None = None*) → None

Run named reachability strategy or the default.

Update *self.shardlikes* with reachable package records. Later, [shardlike.build_repodata() for shardlike in shardlikes] can be used to generate repodata.json-format subsets of each channel.

Parameters

- **root_packages** -- installed and requested package names.
- **strategy** -- traversal algorithm, "bfs" or "pipelined".
- **root_extras** -- optional mapping of root package name to an iterable of CEP 44 extras requested for it (e.g. *{"package": ["extras_group"]}*). Extras explicitly set on dependency specs of fetched packages (e.g. *pkg-u[extras=more]*) are followed transitively; groups that nobody requested stay inactive.

reachable_bfs(*root_packages*, *root_extras:* *collections.abc.Mapping[str, collections.abc.Iterable[str]] | None = None*)

Fetch all packages reachable from *root_packages*' by following dependencies using the "breadth-first search" algorithm.

Update associated *self.shardlikes* to contain enough data to build a repodata subset.

_reachable_bfs(*root_packages*, *shard_cache:* conda._private.shards.cache.ShardCache, *root_extras:* *collections.abc.Mapping[str, collections.abc.Iterable[str]] | None = None*)

Inner reachable_bfs() implementation.

reachable_pipelined(*root_packages*, *root_extras:* *collections.abc.Mapping[str, collections.abc.Iterable[str]] | None = None*)

Fetch all packages reachable from *root_packages*' by following dependencies.

Build repodata subset using concurrent threads to follow dependencies, fetch from cache, and fetch from network.

_reachable_pipelined(*root_packages*, *network_worker:* *collections.abc.Callable[[queue.SimpleQueue[collections.abc.Sequence[NodeId] | None], queue.SimpleQueue[list[tuple[NodeId, conda._private.shards.typing.ShardDict] | Exception]], RepodataSubset._reachable_pipelined.cache, collections.abc.Sequence[conda._private.shards.shards.ShardBase]], None]*, *cache:* *RepodataSubset._reachable_pipelined.cache*, *root_extras:* *collections.abc.Mapping[str, collections.abc.Iterable[str]] | None = None*)

Set up queues and threads for shard traversal with a configurable network_worker. Called by reachable_pipelined()

_pipelined_traversal(*root_packages*, *cache_in_queue:* *queue.SimpleQueue[list[NodeId] | None]*, *shard_out_queue:* *queue.SimpleQueue[list[tuple[NodeId, conda._private.shards.typing.ShardDict]] | Exception]*, *cache_thread:* *threading.Thread*, *network_thread:* *threading.Thread*, *root_extras:* *collections.abc.Mapping[str, collections.abc.Iterable[str]] | None = None*)

Run reachability algorithm given queues to submit and receive shards.

`_visit_node`(*parent_node:* Node, *mentioned_packages:* *collections.abc.Iterable[tuple[str, frozenset[str]]]*, *extras_by_package:* *collections.abc.Mapping[str, collections.abc.Iterable[str]]* | *None* = *None*) → collections.abc.Iterable[*NodeId*]

Broadcast mentioned packages across channels. yield pending NodeId's.

`mentioned_packages` are `(name, extras)` pairs; extras are the CEP 44 groups requested for that package by the spec that mentions it. A package that already has a node only yields again if the extras add groups to a node whose shard was already processed.

`_drain_pending`(*pending:* *set[*NodeId*]*, *shardlikes_by_url:* *dict[str,* conda._private.shards.shards.ShardBase*]*) → tuple[list[tuple[*NodeId*, *conda._private.shards.typing.ShardDict*]], list[*NodeId*]]

Check pending for in-memory shards. Clear pending.

Return a list of shards we have and shards we need to fetch.

`build_repodata_subset`(*root_packages:* *collections.abc.Iterable[str]*, *channels:* *dict[str,* conda.models.channel.Channel*]*, *algorithm:* *Literal['bfs', 'pipelined']* = *RepodataSubset.DEFAULT_STRATEGY*, *spec_to_package_name_func:* *collections.abc.Callable[[str], str]* = *spec_to_package_name*, *repodata_version:* *int* = *1*, *depth:* *int* = *sys.maxsize*, *root_extras:* *collections.abc.Mapping[str, collections.abc.Iterable[str]]* | *None* = *None*) → dict[str, *conda._private.shards.shards.ShardBase*] | None

Retrieve all necessary information to build a repodata subset.

This function implements the conda.gateways.shards.BuildRepodataSubset protocol, allowing it to be passed to solvers that support sharded repodata optimization.

Params:

root_packages: iterable of installed and requested package names channels: Channel objects; dict form preferred. algorithm: desired traversal algorithm ("bfs" or "pipelined") spec_to_package_name_func: callable to convert package specs to names.

Defaults to the standard spec_to_package_name.

repodata_version: repodata format version (1 = classic, 3 = v3). depth: the maximum depth of dependant packages to include in the repodata

subset.

root_extras: optional mapping of root package name to an iterable of

CEP 44 extras requested for it (e.g. *{"package": ["extras_group"]}*). Extras on dependency specs are followed transitively; see *RepodataSubset.reachable*.

Returns

None if there are no shards available, or a mapping of channel URL's to ShardBase objects where build_repodata() returns the computed subset.

`class BuildRepodataSubset`

Bases: `Protocol`

Protocol for build_repodata_subset callable.

This function is used by solvers to construct a minimal subset of repodata based on the root packages that might be installed and the available channels. It traverses package dependencies to discover all reachable (channel, package) tuples, which are then used by the solver to reduce search space.

__call__(*root_packages: collections.abc.Iterable[str], channels: dict[str, Any], algorithm: Literal['bfs', 'pipelined'] = 'pipelined', repodata_version: int = 1, root_extras: collections.abc.Mapping[str, collections.abc.Iterable[str]] | None = None*) → dict[str, *Shards*] | None

Retrieve a minimal subset of repodata based on root packages.

Parameters

- **root_packages** -- Iterable of installed and requested package names
- **channels** -- Dictionary mapping channel URLs to Channel objects
- **algorithm** -- Traversal algorithm to use ("bfs" or "pipelined")
- **repodata_version** -- repodata format version (1 = classic, 3 = v3).
- **root_extras** -- Optional mapping of root package name to an iterable of CEP 44 extras requested for it (e.g. *{"httpx": ["cli"]}*), so that shards for its requested *extra_depends* groups are also fetched. Extras explicitly requested by dependency specs of fetched packages are followed transitively.

Returns

A dictionary mapping channel URLs to Shards objects containing the subset of packages needed, or None if shards are unavailable

streams

Helper functions for streaming output to stdout and stderr.

Functions

redact_token_urls(message) → str	Redact channel tokens in URLs.
stdout(text, **kwargs) → None	
stderr(text, **kwargs) → None	
stdoutlog(text, **kwargs) → None	
stderrlog(text, **kwargs) → None	

Attributes

_TOKEN_URL_PATTERN

_TOKEN_URL_PATTERN

redact_token_urls(*message: str*) → str

Redact channel tokens in URLs.

stdout(*text: str*, ***kwargs*) → None

stderr(*text: str*, ***kwargs*) → None

stdoutlog(*text: str*, ***kwargs*) → None

stderrlog(*text: str*, ***kwargs*) → None

subprocess

Helpler functions for subprocess.

Classes

Response

Functions

_format_output(command_str, cwd, rc, stdout, stderr)	
any_subprocess(args, prefix[, env, cwd])	
subprocess_call(command[, env, path, stdin, ...])	This utility function should be preferred for all conda subprocessing.
_subprocess_clean_env(env[, clean_python, clean_conda])	
subprocess_call_with_clean_env(command[, path, stdin, ...])	

class Response

Bases: tuple

stdout

stderr

rc

_format_output(*command_str*, *cwd*, *rc*, *stdout*, *stderr*)

any_subprocess(*args: collections.abc.Sequence[str]*, *prefix*, *env=None*, *cwd=None*)

subprocess_call(*command: str | os.PathLike | pathlib.Path | collections.abc.Sequence[str | os.PathLike | pathlib.Path]*, *env: dict[str, str] | None = None*, *path: str | os.PathLike | pathlib.Path | None = None*, *stdin: str | None = None*, *raise_on_error: bool = True*, *capture_output: bool = True*)

This utility function should be preferred for all conda subprocessing. It handles multiple tricky details.

_subprocess_clean_env(*env*, *clean_python=True*, *clean_conda=True*)

subprocess_call_with_clean_env(*command*, *path=None*, *stdin=None*, *raise_on_error=True*, *clean_python=True*, *clean_conda=True*)

history

Tools interfacing with conda's history file.

Classes

History

Functions

write_head(fo)	
is_diff(content)	
pretty_diff(diff)	
pretty_content(content)	

exception CondaHistoryWarning

Bases: Warning

Base class for warning categories.

Initialize self. See help(type(self)) for accurate signature.

write_head(*fo*)

is_diff(*content*)

pretty_diff(*diff*)

pretty_content(*content*)

class History(*prefix: conda.common.path.PathType*)

com_pat

spec_pat

conda_v_pat

prefix

meta_dir

path

__enter__()

__exit__(*exc_type*, *exc_value*, *traceback*)

init_log_file()

file_is_empty()

update() → None

Update the history file (creating a new one if necessary).

parse() → list[tuple[str, set[str], list[str]]]

Parse the history file.

Return a list of tuples(datetime strings, set of distributions/diffs, comments).

Comments appearing before the first section header (e.g. `==> 2024-01-01 00:00:00 <==`) in the history file will be ignored.

static _parse_old_format_specs_string(*specs_string*)

Parse specifications string that use conda<4.5 syntax.

Examples

- "param >=1.5.1,<2.0'"
- "python>=3.5.1,jupyter >=1.0.0,<2.0,matplotlib >=1.5.1,<2.0"

classmethod _parse_comment_line(*line*)

Parse comment lines in the history file.

These lines can be of command type or action type.

Examples

- "# cmd: /scratch/mc3/bin/conda install -c conda-forge param>=1.5.1,<2.0"
- "# install specs: python>=3.5.1,jupyter >=1.0.0,<2.0,matplotlib >=1.5.1,<2.0"

get_user_requests()

Return a list of user requested items.

Each item is a dict with the following keys: 'date': the date and time running the command 'cmd': a list of argv of the actual command which was run 'action': install/remove/update 'specs': the specs being used

get_requested_specs_map()

construct_states()

Return a list of tuples(datetime strings, set of distributions).

get_state(*rev=-1*)

Return the state, i.e. the set of distributions, for a given revision.

Defaults to latest (which is the same as the current state when the log file is up-to-date).

Returns a list of dist_strs.

print_log()

object_log()

write_changes(*last_state*, *current_state*)

write_specs(*remove_specs=()*, *update_specs=()*, *neutered_specs=()*)

instructions

Define the instruction set (constants) for conda operations.

Functions

PRINT_CMD(state, arg)	
FETCH_CMD(state, package_cache_entry)	
EXTRACT_CMD(state, arg)	
PROGRESSIVEFETCHEXTRACT_CMD(state, ...)	
UNLINKLINKTRANSACTION_CMD(state, arg)	
check_files_in_package(source_dir, files)	

Attributes

CHECK_FETCH
FETCH
CHECK_EXTRACT
EXTRACT
RM_EXTRACTED
RM_FETCHED
PRINT
PROGRESS
SYMLINK_CONDA
UNLINK
LINK
UNLINKLINKTRANSACTION
PROGRESSIVEFETCHEXTRACT
PROGRESS_COMMANDS
ACTION_CODES
commands
OP_ORDER

CHECK_FETCH = 'CHECK_FETCH'

FETCH = 'FETCH'

CHECK_EXTRACT = 'CHECK_EXTRACT'

EXTRACT = 'EXTRACT'

RM_EXTRACTED = 'RM_EXTRACTED'

RM_FETCHED = 'RM_FETCHED'

PRINT = 'PRINT'

PROGRESS = 'PROGRESS'

SYMLINK_CONDA = 'SYMLINK_CONDA'

UNLINK = 'UNLINK'

LINK = 'LINK'

UNLINKLINKTRANSACTION = 'UNLINKLINKTRANSACTION'

PROGRESSIVEFETCHEXTRACT = 'PROGRESSIVEFETCHEXTRACT'

PROGRESS_COMMANDS

ACTION_CODES

PRINT_CMD(*state*, *arg*)

FETCH_CMD(*state*, *package_cache_entry*)

EXTRACT_CMD(*state*, *arg*)

PROGRESSIVEFETCHEXTRACT_CMD(*state*, *progressive_fetch_extract*)

UNLINKLINKTRANSACTION_CMD(*state*, *arg*)

check_files_in_package(*source_dir*, *files*)

commands

OP_ORDER

misc

Miscellaneous utility functions.

Functions

conda_installed_files(prefix[, exclude_self_build])	Return the set of files which have been installed (using conda) into
_get_url_pattern() → re.Pattern	Build URL pattern dynamically based on registered package extensions.
_match_specs_from_explicit(specs) → collections.abc.Iterable[conda.models.match_spec.MatchSpec]	
explicit(specs, prefix[, verbose, force_extract, ...]) → None	
install_explicit_packages(package_cache_records prefix)	Install a list of PackageRecords into a prefix
_get_package_record_from_specs(specs) → collections.abc.Iterable[conda.models.records.PackageCache	Given a list of specs, find the corresponding PackageCacheRecord. If
get_package_records_from_explicit(lines) → collections.abc.Iterable[conda.models.records.PackageCa	Given the lines from an explicit.txt, create the PackageRecords for each of the
walk_prefix(prefix[, ignore_predefined_files, ...])	Return the set of all files in a given prefix directory.
untracked(prefix[, exclude_self_build])	Return (the set) of all untracked files for a given prefix.
clone_env(prefix1, prefix2[, verbose, quiet, index_args])	Clone existing prefix1 into new prefix2.
_get_best_prec_match(precs)	

conda_installed_files(*prefix*, *exclude_self_build=False*)

Return the set of files which have been installed (using conda) into a given prefix.

_get_url_pattern() → re.Pattern

Build URL pattern dynamically based on registered package extensions.

_match_specs_from_explicit(*specs: collections.abc.Iterable[str]*) → collections.abc.Iterable[*conda.models.match_spec.MatchSpec*]

explicit(*specs: collections.abc.Iterable[str]*, *prefix: str*, *verbose: bool = False*, *force_extract: bool = True*, *index: Any = None*, *requested_specs: collections.abc.Sequence[str] | None = None*) → None

install_explicit_packages(*package_cache_records: list[*conda.models.records.PackageRecord*]*, *prefix: str*, *requested_specs: collections.abc.Sequence[str] | None = None*)

Install a list of PackageRecords into a prefix

_get_package_record_from_specs(*specs: list[str]*) → collections.abc.Iterable[*conda.models.records.PackageCacheRecord*]

Given a list of specs, find the corresponding PackageCacheRecord. If some PackageCacheRecords are missing, raise an error.

get_package_records_from_explicit(*lines: list[str]*) → collections.abc.Iterable[*conda.models.records.PackageCacheRecord*]

Given the lines from an explicit.txt, create the PackageRecords for each of the specified packages. This may require downloading the package, if it does not already exist in the package cache.

walk_prefix(*prefix*, *ignore_predefined_files=True*, *windows_forward_slashes=True*)

Return the set of all files in a given prefix directory.

untracked(*prefix*, *exclude_self_build=False*)

Return (the set) of all untracked files for a given prefix.

clone_env(*prefix1*, *prefix2*, *verbose=True*, *quiet=False*, *index_args=None*)

Clone existing prefix1 into new prefix2.

_get_best_prec_match(*precs*)

models

Models are data transfer objects or "light-weight" domain objects with no appreciable logic other than their own validation. Models are used to pass data between layers of the stack. In many ways they are similar to ORM objects. Unlike ORM objects, they are NOT themselves allowed to load data from a remote resource. Thought of another way, they cannot import from `conda.gateways`, but rather `conda.gateways` imports from `conda.models` as appropriate to create model objects from remote resources.

Conda modules importable from `conda.models` are

- `conda.common`
- `conda.models`

build_string

Sortable conda build string data model.

Derived from reference implementation provided by Travis Hathaway in conda/conda#16849. Analyzes conda build strings to support ordering by build number, variant suffix, and string tie-breaker, adhering to CEP 26 conventions.

Classes

BuildString	A `str` that sorts conda build strings by build number.

Functions

_split(value) → tuple[int | None, str]	Return `(build_number, tail)` for `value`.

Attributes

_CEP26_RE

continues on next page

Table 236 – continued from previous page

_CEP26_MAX_LEN
_NUMBER_RE

_CEP26_RE

_CEP26_MAX_LEN = 64

_NUMBER_RE

_split(*value: str*) → tuple[int | None, str]

Return `(build_number, tail)` for `value`.

`build_number` is the last all-digit _ token, or `None`. `tail` is everything after that token (the variant suffix such as `cuda` or `openblas`), or `""`.

One special case: `python` / `cross-python_*` PyPy builds are `<build_number>_<pypy_abi>_pypy` (`14_73_pypy`). The trailing `73` is the PyPy ABI version, constant across builds, so the number is the token *before* it.

Bounded cache: caches up to 4096 distinct build strings to accelerate sorting while preventing unbounded memory growth.

Parameters
value -- The build string to parse.

Returns
A tuple of `(build_number, tail)`.

class BuildString

Bases: `str`

A `str` that sorts conda build strings by build number.

It is a `str`: hashing, == against plain strings, slicing, formatting, and JSON serialization are unchanged. Ordering (<, <=, >, >=) is overridden so `sorted()`, `max()`, and `min()` order by parsed build number, variant suffix, and raw string tie-breaker.

Ordering key, most significant first: 1. Strings without any numeric token sort before all numbered ones. 2. The build number, compared as an `int`. 3. The variant suffix after the number (e.g. `""` < `"cpu"` < `"cuda"`). 4. The raw string, ensuring deterministic total ordering when builds tie.

Initialize self. See help(type(self)) for accurate signature.

__slots__ = ()

property build_number: int | None

The build number parsed from the string, or `None`.

property suffix: str

Variant text following the build number (e.g. `cuda`, `openblas`).

property is_cep26_valid: bool

Whether the string satisfies CEP 26 (charset and <= 64 chars).

property sort_key: tuple[int, int, str, str]

Total-order key; usable as `key=` for plain strings too.

classmethod key(*value: str*) → tuple[int, int, str, str]

Convenience function for `sorted(strings, key=BuildString.key)`.

_other_key(*other: Any*) → tuple[int, int, str, str] | None

__lt__(*other: Any*) → bool

Return self<value.

__le__(*other: Any*) → bool

Return self<=value.

__gt__(*other: Any*) → bool

Return self>value.

__ge__(*other: Any*) → bool

Return self>=value.

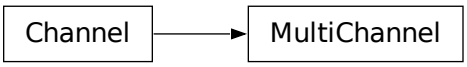
__hash__

Return hash(self).

channel

Defines Channel and MultiChannel objects and other channel-related functions.

Object inheritance:



Classes

Channel	Channel:
MultiChannel	Channel:

Functions

tokenized_startswith(test_iterable, startswith_iterable) → bool	
tokenized_conda_url_startswith(test_url, startswith_url) → bool	
_get_channel_for_name(channel_name) → Channel	
_read_channel_configuration(scheme, host, port, path) → tuple[str | None, str | None, str | None, str | None, str | None]	
parse_conda_channel_url(url) → Channel	
get_conda_build_local_url() → tuple[conda.common.path.PathType]	
prioritize_channels(channels[, with_credentials, subdirs]) → dict[str, tuple[str, int]]	Make a dictionary of channel priorities.

continues on next page

Table 238 – continued from previous page

all_channel_urls(channels[, subdirs, with_credentials]) → tuple[Channel, ...]	
offline_keep(url) → bool	
get_channel_objs(ctx) → tuple[Channel, ...]	Return current channels as Channel objects

class Channel(*scheme: str | None = None*, *auth: str | None = None*, *location: str | None = None*, *token: str | None = None*, *name: str | None = None*, *platform: str | None = None*, *package_filename: str | None = None*)

Channel: scheme <> auth <> location <> token <> channel <> subchannel <> platform <> package_filename

Package Spec: channel <> subchannel <> namespace <> package_name

static _reset_state() → None

scheme = None

auth = None

location = None

token = None

name = ''

platform = None

package_filename = None

property channel_location: str | None

property channel_name: str

property subdir: str | None

static from_url(*url: str*) → *Channel*

static from_channel_name(*channel_name: str*) → *Channel*

static from_value(*value: str | None*) → *Channel*

Construct a new *Channel* from a single value.

Parameters

value -- Anyone of the following forms:

***None*, or one of the special strings "<unknown>", "None:///<unknown>", or "None":**
represents the unknown channel, used for packages with unknown origin.

A URL including a scheme like `file://` or `https://`:
represents a channel URL.

A local directory path:
represents a local channel; relative paths must start with `./`.

A package file (i.e. the path to a file ending in `.conda` or `.tar.bz2`):
represents a channel for a single package

A known channel name:
represents a known channel, e.g. from the users `.condarc` file or the global configuration.

Returns
A channel object.

static make_simple_channel(*channel_alias: Channel*, *channel_url: str*, *name: str | None = None*) → *Channel*

property canonical_name: str

urls(*with_credentials: bool = False*, *subdirs: collections.abc.Iterable[str] | None = None*) → list[str]

Generate URLs for this channel across specified platforms.

Parameters

- **with_credentials** -- If True, include authentication credentials (token, auth) in URLs.
- **subdirs** -- Specific platform subdirs to generate URLs for. If None, uses the channel's platform (if defined) or falls back to *context.subdirs*. If this is explicitly provided, overrides any platform defined in the channel.

Examples

```
>>> channel = Channel("conda-forge")
>>> channel.urls()  # Uses context.subdirs
['https://conda.anaconda.org/conda-forge/linux-64',
'https://conda.anaconda.org/conda-forge/noarch']
```

```
>>> channel = Channel("conda-forge/linux-aarch64")
>>> channel.urls()  # Uses channel's platform
['https://conda.anaconda.org/conda-forge/linux-aarch64',
'https://conda.anaconda.org/conda-forge/noarch']
```

```
>>> channel.urls(subdirs=("osx-64", "noarch"))
['https://conda.anaconda.org/conda-forge/osx-64',
'https://conda.anaconda.org/conda-forge/noarch']
```

Returns
List of URLs for accessing this channel's specified subdirectories.

Return type
list[str]

url(*with_credentials: bool = False*) → str | None

property base_url: str | None

property base_urls: tuple[str | None, ...]

property subdir_url: str

property channels: tuple[*Channel*, ...]

__str__() → str

__repr__() → str

__eq__(*other: Any*) → bool

__hash__() → int

__nonzero__() → bool

__bool__() → bool

__json__() → dict[str, Any]

property url_channel_wtf: tuple[str | None, str]

dump() → dict[str, Any]

class MultiChannel(*name: str*, *channels: collections.abc.Iterable[*Channel*] | None = None*, *platform: str | None = None*)

Bases: *Channel*

Channel: scheme <> auth <> location <> token <> channel <> subchannel <> platform <> package_filename

Package Spec: channel <> subchannel <> namespace <> package_name

name

location = None

scheme = None

auth = None

token = None

platform = None

package_filename = None

property canonical_name: str

urls(*with_credentials: bool = False*, *subdirs: collections.abc.Iterable[str] | None = None*) → list[str]

Generate URLs for this channel across specified platforms.

Parameters

- **with_credentials** -- If True, include authentication credentials (token, auth) in URLs.
- **subdirs** -- Specific platform subdirs to generate URLs for. If None, uses the channel's platform (if defined) or falls back to *context.subdirs*. If this is explicitly provided, overrides any platform defined in the channel.

Examples

```
>>> channel = Channel("conda-forge")
>>> channel.urls()  # Uses context.subdirs
['https://conda.anaconda.org/conda-forge/linux-64',
'https://conda.anaconda.org/conda-forge/noarch']
```

```
>>> channel = Channel("conda-forge/linux-aarch64")
>>> channel.urls()  # Uses channel's platform
['https://conda.anaconda.org/conda-forge/linux-aarch64',
'https://conda.anaconda.org/conda-forge/noarch']
```

```
>>> channel.urls(subdirs=("osx-64", "noarch"))
['https://conda.anaconda.org/conda-forge/osx-64',
'https://conda.anaconda.org/conda-forge/noarch']
```

Returns
List of URLs for accessing this channel's specified subdirectories.

Return type
list[str]

property base_url: None

property base_urls: tuple[str | None, ...]

url(*with_credentials: bool = False*) → None

property channels: tuple[*Channel*, ...]

dump() → dict[str, Any]

tokenized_startswith(*test_iterable: collections.abc.Iterable[Any]*, *startswith_iterable: collections.abc.Iterable[Any]*) → bool

tokenized_conda_url_startswith(*test_url: collections.abc.Iterable[str]*, *startswith_url: collections.abc.Iterable[str]*) → bool

_get_channel_for_name(*channel_name: str*) → *Channel*

_read_channel_configuration(*scheme: str | None*, *host: str | None*, *port: str | None*, *path: str | None*) → tuple[str | None, str | None, str | None, str | None, str | None]

parse_conda_channel_url(*url: str*) → *Channel*

get_conda_build_local_url() → tuple[conda.common.path.PathType]

prioritize_channels(*channels: collections.abc.Iterable[Channel | str]*, *with_credentials: bool = True*, *subdirs: collections.abc.Iterable[str] | None = None*) → dict[str, tuple[str, int]]

Make a dictionary of channel priorities.

Maps channel names to priorities, e.g.:

```
>>> prioritize_channels(["conda-canary", "defaults", "conda-forge"])
{
    'https://conda.anaconda.org/conda-canary/osx-arm64': ('conda-canary', 0),
    'https://conda.anaconda.org/conda-canary/noarch': ('conda-canary', 0),
    'https://repo.anaconda.com/pkgs/main/osx-arm64': ('defaults', 1),
    'https://repo.anaconda.com/pkgs/main/noarch': ('defaults', 1),
    'https://repo.anaconda.com/pkgs/r/osx-arm64': ('defaults', 2),
    'https://repo.anaconda.com/pkgs/r/noarch': ('defaults', 2),
    'https://conda.anaconda.org/conda-forge/osx-arm64': ('conda-forge', 3),
    'https://conda.anaconda.org/conda-forge/noarch': ('conda-forge', 3),
}
```

Compare with `conda.resolve.Resolve._make_channel_priorities`.

all_channel_urls(*channels: collections.abc.Iterable[str | Channel]*, *subdirs: collections.abc.Iterable[str] | None = None*, *with_credentials: bool = True*) → tuple[*Channel*, ...]

offline_keep(*url: Any*) → bool

get_channel_objs(*ctx:* conda.base.context.Context) → tuple[*Channel*, ...]

Return current channels as Channel objects

dist

(Legacy) Low-level implementation of a Channel.

Classes

DistDetails
DistType
Dist

Functions

strip_extension(original_dist) → str
split_extension(original_dist) → tuple[str, str]
dist_str_to_quad(dist_str)

class DistDetails

Bases: NamedTuple

name: str

version: str

build_string: str

build_number: str

dist_name: str

fmt: str

class DistType(*name*, *bases*, *attr*)

Bases: conda.auxlib.entity.EntityType

__call__(**args*, ***kwargs*)

strip_extension(*original_dist: str*) → str

split_extension(*original_dist: str*) → tuple[str, str]

class Dist(*channel*, *dist_name=None*, *name=None*, *version=None*, *build_string=None*, *build_number=None*, *base_url=None*, *platform=None*, *fmt='.tar.bz2'*)

Bases: *conda.auxlib.entity.Entity*

cache

_lazy_validate = True

channel

dist_name

name

fmt

version

build_string

build_number

base_url

platform

to_package_ref()

property full_name

property build

property subdir

property pair

property quad

__str__()

property is_feature_package

property is_channel

to_filename(*extension=None*)

to_matchspec()

to_match_spec()

classmethod from_string(*string*, *channel_override=NULL*)

static parse_dist_name(*string*)

classmethod from_url(*url*)

to_url()

__key__()

__lt__(*other*)

__gt__(*other*)

__le__(*other*)

__ge__(*other*)

__hash__()

__eq__(*other*)

__ne__(*other*)

split(*sep=None*, *maxsplit=-1*)

rsplit(*sep=None*, *maxsplit=-1*)

startswith(*match*)

__contains__(*item*)

property fn

dist_str_to_quad(*dist_str*)

enums

Collection of enums used throughout conda.

Classes

Arch	Create a collection of name/value pairs.
Platform	Create a collection of name/value pairs.
FileMode	Create a collection of name/value pairs.
LinkType	Create a collection of name/value pairs.
PathEnum	Refers to if the file in question is hard linked or soft linked. Originally designed to be used
PackageType	Create a collection of name/value pairs.
NoarchType	Create a collection of name/value pairs.

class Arch

Bases: enum.Enum

Create a collection of name/value pairs.

Example enumeration:

```
>>> class Color(Enum):
...     RED = 1
...     BLUE = 2
...     GREEN = 3
```

Access them by:

- attribute access:

```
>>> Color.RED
<Color.RED: 1>
```

- value lookup:

```
>>> Color(1)
<Color.RED: 1>
```

- name lookup:

```
>>> Color['RED']
<Color.RED: 1>
```

Enumerations can be iterated over, and know how many members they have:

```
>>> len(Color)
3
```

```
>>> list(Color)
[<Color.RED: 1>, <Color.BLUE: 2>, <Color.GREEN: 3>]
```

Methods can be added to enumerations, and members can have their own attributes -- see the documentation for details.

x86 = 'x86'

x86_64 = 'x86_64'

arm64 = 'arm64'

armv6l = 'armv6l'

armv7l = 'armv7l'

aarch64 = 'aarch64'

ppc64 = 'ppc64'

ppc64le = 'ppc64le'

riscv64 = 'riscv64'

s390x = 's390x'

wasm32 = 'wasm32'

z = 'z'

classmethod from_sys()

__json__()

class Platform

Bases: enum.Enum

Create a collection of name/value pairs.

Example enumeration:

```
>>> class Color(Enum):
...     RED = 1
...     BLUE = 2
...     GREEN = 3
```

Access them by:

- attribute access:

```
>>> Color.RED
<Color.RED: 1>
```

- value lookup:

```
>>> Color(1)
<Color.RED: 1>
```

- name lookup:

```
>>> Color['RED']
<Color.RED: 1>
```

Enumerations can be iterated over, and know how many members they have:

```
>>> len(Color)
3
```

```
>>> list(Color)
[<Color.RED: 1>, <Color.BLUE: 2>, <Color.GREEN: 3>]
```

Methods can be added to enumerations, and members can have their own attributes -- see the documentation for details.

freebsd = 'freebsd'

linux = 'linux'

win = 'win32'

openbsd = 'openbsd5'

osx = 'darwin'

zos = 'zos'

emscripten = 'emscripten'

wasi = 'wasi'

classmethod from_sys()

__json__()

class FileMode

Bases: enum.Enum

Create a collection of name/value pairs.

Example enumeration:

```
>>> class Color(Enum):
...     RED = 1
...     BLUE = 2
...     GREEN = 3
```

Access them by:

- attribute access:

```
>>> Color.RED
<Color.RED: 1>
```

- value lookup:

```
>>> Color(1)
<Color.RED: 1>
```

- name lookup:

```
>>> Color['RED']
<Color.RED: 1>
```

Enumerations can be iterated over, and know how many members they have:

```
>>> len(Color)
3
```

```
>>> list(Color)
[<Color.RED: 1>, <Color.BLUE: 2>, <Color.GREEN: 3>]
```

Methods can be added to enumerations, and members can have their own attributes -- see the documentation for details.

text = 'text'

binary = 'binary'

__str__()

class LinkType

Bases: enum.Enum

Create a collection of name/value pairs.

Example enumeration:

```
>>> class Color(Enum):
...     RED = 1
...     BLUE = 2
...     GREEN = 3
```

Access them by:

- attribute access:

```
>>> Color.RED
<Color.RED: 1>
```

- value lookup:

```
>>> Color(1)
<Color.RED: 1>
```

- name lookup:

```
>>> Color['RED']
<Color.RED: 1>
```

Enumerations can be iterated over, and know how many members they have:

```
>>> len(Color)
3
```

```
>>> list(Color)
[<Color.RED: 1>, <Color.BLUE: 2>, <Color.GREEN: 3>]
```

Methods can be added to enumerations, and members can have their own attributes -- see the documentation for details.

hardlink = 1

softlink = 2

copy = 3

directory = 4

__int__()

__str__()

__json__()

class PathEnum

Bases: enum.Enum

Refers to if the file in question is hard linked or soft linked. Originally designed to be used in paths.json

hardlink = 'hardlink'

softlink = 'softlink'

directory = 'directory'

linked_package_record = 'linked_package_record'

pyc_file = 'pyc_file'

unix_python_entry_point = 'unix_python_entry_point'

windows_python_entry_point_script = 'windows_python_entry_point_script'

windows_python_entry_point_exe = 'windows_python_entry_point_exe'

basic_types()

__str__()

__json__()

class PackageType

Bases: enum.Enum

Create a collection of name/value pairs.

Example enumeration:

```
>>> class Color(Enum):
...     RED = 1
...     BLUE = 2
...     GREEN = 3
```

Access them by:

- attribute access:

```
>>> Color.RED
<Color.RED: 1>
```

- value lookup:

```
>>> Color(1)
<Color.RED: 1>
```

- name lookup:

```
>>> Color['RED']
<Color.RED: 1>
```

Enumerations can be iterated over, and know how many members they have:

```
>>> len(Color)
3
```

```
>>> list(Color)
[<Color.RED: 1>, <Color.BLUE: 2>, <Color.GREEN: 3>]
```

Methods can be added to enumerations, and members can have their own attributes -- see the documentation for details.

NOARCH_GENERIC = 'noarch_generic'

NOARCH_PYTHON = 'noarch_python'

VIRTUAL_PRIVATE_ENV = 'virtual_private_env'

VIRTUAL_PYTHON_WHEEL = 'virtual_python_wheel'

VIRTUAL_PYTHON_EGG_MANAGEABLE = 'virtual_python_egg_manageable'

VIRTUAL_PYTHON_EGG_UNMANAGEABLE = 'virtual_python_egg_unmanageable'

VIRTUAL_PYTHON_EGG_LINK = 'virtual_python_egg_link'

VIRTUAL_SYSTEM = 'virtual_system'

static conda_package_types()

static unmanageable_package_types()

class NoarchType

Bases: enum.Enum

Create a collection of name/value pairs.

Example enumeration:

```
>>> class Color(Enum):
...     RED = 1
...     BLUE = 2
...     GREEN = 3
```

Access them by:

- attribute access:

```
>>> Color.RED
<Color.RED: 1>
```

- value lookup:

```
>>> Color(1)
<Color.RED: 1>
```

- name lookup:

```
>>> Color['RED']
<Color.RED: 1>
```

Enumerations can be iterated over, and know how many members they have:

```
>>> len(Color)
3
```

```
>>> list(Color)
[<Color.RED: 1>, <Color.BLUE: 2>, <Color.GREEN: 3>]
```

Methods can be added to enumerations, and members can have their own attributes -- see the documentation for details.

generic = 'generic'

python = 'python'

static coerce(*val*)

environment

EXPERIMENTAL Conda environment data model

Classes

EnvironmentConfig	**Experimental** While experimental, expect both major and minor changes across minor releases.
Environment	**Experimental** While experimental, expect both major and minor changes across minor releases.

Attributes

T
EXTERNAL_PACKAGES_PYPI_KEY

T

EXTERNAL_PACKAGES_PYPI_KEY: Final = 'pip'

class EnvironmentConfig

Experimental While experimental, expect both major and minor changes across minor releases.

Data model for a conda environment config.

aggressive_update_packages: tuple[str, ...] = ()

channel_priority: *conda.base.constants.ChannelPriority* | None = None

channels: tuple[str, ...] = ()

channel_settings: tuple[dict[str, str], ...] = ()

deps_modifier: *conda.base.constants.DepsModifier* | None = None

disallowed_packages: tuple[str, ...] = ()

pinned_packages: tuple[str, ...] = ()

repodata_fns: tuple[str, ...] = ()

sat_solver: *conda.base.constants.SatSolverChoice* | None = None

solver: str | None = None

track_features: tuple[str, ...] = ()

update_modifier: *conda.base.constants.UpdateModifier* | None = None

use_only_tar_bz2: bool | None = None

_append_without_duplicates(*first: collections.abc.Iterable[T]*, *second: collections.abc.Iterable[T]*) → tuple[*T*, ...]

_merge_channel_settings(*first: tuple[dict[str, str], ...]*, *second: tuple[dict[str, str], ...]*) → tuple[dict[str, str], ...]

Merge channel settings.

An individual channel setting is a dict that may have the key "channels". Settings with matching "channels" should be merged together.

`_merge`(*other:* EnvironmentConfig) → *EnvironmentConfig*

Experimental While experimental, expect both major and minor changes across minor releases.

Merges an EnvironmentConfig into this one. Merging rules are: * Primitive types get clobbered if subsequent configs have a value, otherwise keep the last set value * Lists get appended to and deduplicated * Dicts get updated * Special cases:

- channel settings is a list of dicts, it merges inner dicts, keyed on "channel"
- channels: last wins for priority (prepended so later config's channels take precedence)

`classmethod from_context`() → *EnvironmentConfig*

Experimental While experimental, expect both major and minor changes across minor releases.

Create an EnvironmentConfig from the current context

`classmethod from_cli_channels`(*args:* *argparse.Namespace*) → *EnvironmentConfig*

Build a sparse EnvironmentConfig from CLI args for channels only.

Used to override channels from the CLI when files are provided.

`classmethod merge`(**configs:* EnvironmentConfig) → *EnvironmentConfig*

Experimental While experimental, expect both major and minor changes across minor releases.

Merges a list of EnvironmentConfigs into a single one. Merging rules are: * Primitive types get clobbered if subsequent configs have a value, otherwise keep the last set value * Lists get appended to and deduplicated * Dicts get updated

`class Environment`

Experimental While experimental, expect both major and minor changes across minor releases.

Data model for a conda environment.

`platform: str`

The platform this environment may be installed on (required).

`config: EnvironmentConfig`

Environment level configuration, eg. channels, solver options, etc.

TODO: may need to think more about the type of this field and how conda should be merging configs between environments.

`external_packages: dict[str, list[str]]`

Map of other package types that conda can install. For example pypi packages.

`explicit_packages: list[conda.models.records.PackageRecord] = []`

The complete list of specs for the environment.

E.g. after a solve, or from an explicit environment spec.

`name: str | None = None`

Environment name.

`prefix: str | None = None`

Prefix the environment is installed into.

`requested_packages: list[conda.models.match_spec.MatchSpec] = []`

User requested specs for this environment.

`variables: dict[str, str]`

Environment variables to be applied to the environment.

virtual_packages: list[*conda.models.records.PackageRecord*] = []

__post_init__()

classmethod merge(**environments*)

Experimental While experimental, expect both major and minor changes across minor releases.

Merges multiple environments into a single environment following the rules: * name, prefix: last wins (later env overrides earlier). * platform: must match across all envs. * requested_packages, explicit_packages, virtual_packages: union, deduplicated. * variables: merged dict, last value wins per key. * external_packages: concatenation per key (union of lists, deduplicated). * config: EnvironmentConfig.merge (last wins for primitives, append for lists).

classmethod from_prefix(*prefix: str*, *name: str*, *platform: str*, ***, *from_history: bool = False*, *no_builds: bool = False*, *ignore_channels: bool = False*, *channels: list[str] | None = None*) → *Environment*

Create an Environment model from an existing conda prefix.

This method analyzes an installed conda environment and creates an Environment model that can be used for exporting or other operations.

Parameters

- **prefix** -- Path to the conda environment prefix
- **name** -- Name for the environment
- **platform** -- Target platform (e.g., 'linux-64', 'osx-64')
- **from_history** -- Use explicit specs from history instead of installed packages
- **no_builds** -- Exclude build strings from package specs
- **ignore_channels** -- Don't include channel information in package specs

Returns
Environment model representing the prefix

classmethod from_cli(*args: argparse.Namespace*, *add_default_packages: bool = False*) → *Environment*

Create an Environment model from command-line arguments.

This method will parse command-line arguments and create an Environment object. This includes: reading files provided as cli arguments, and pulling EnvironmentConfig from the context.

Parameters
args -- argparse Namespace containing command-line arguments

Returns
An Environment object representing the cli

classmethod from_cli_with_file_envs(*args: argparse.Namespace*, *add_default_packages: bool = False*) → tuple[*Environment*, dict[str, *Environment*]]

Create an Environment model from command-line arguments, with a map of file path to Environment for each environment file specified.

Parameters
args -- argparse Namespace containing command-line arguments

Returns
Tuple of (merged Environment, dict mapping file path to Environment)

static from_history(*prefix: conda.common.path.PathType*) → list[*conda.models.match_spec.MatchSpec*]

extrapolate(*platform: str*) → *Environment*

Given the current environment, solve for a comparable environment on a different platform.

match_spec

Implements the query language for conda packages (a.k.a, MatchSpec).

The MatchSpec is the conda package specification (e.g. *conda==23.3*, *python<3.7*, *cryptography * *_0*) and is used to communicate the desired packages to install.

Classes

MatchSpecType	
MatchSpec	The query language for conda packages.
MatchInterface	
_StrMatchMixin	
ExactStrMatch	
ExactLowerStrMatch	
GlobStrMatch	
GlobLowerStrMatch	
SplitStrMatch	
FeatureMatch	
ListOfStrMatch	
ChannelMatch	
CaseInsensitiveStrMatch	

Functions

_validate_package_name(name, original_spec_str) → None	Validate a package name against CEP-26 rules.
_parse_version_plus_build(v_plus_b)	This should reliably pull the build string out of a version + build string combo.
_parse_legacy_dist(dist_str)	
_is_non_alias_url_channel(chn) → bool	Return `True` if `chn` was constructed from a URL whose location
_parse_channel(channel_val)	
_sanitize_version_str(version, build) → str	Sanitize version strings for MatchSpec parsing.
_parse_spec_str_dispatcher(spec_str) → dict[str, object]	Temporary dispatcher while we introduce the new v3 features
_parse_spec_str(spec_str)	
_parse_list_of_str(inner, *, list_syntax) → tuple[str, ...]	Parse names without converting scalar spelling or accepting typed lists.
_parse_spec_str_v3(spec_str)	New parser engine only used
_validate_when_spec(spec) → None	The value of a *when* expression is a MatchSpec or a boolean

Attributes

_BRACKETS_RE

continues on next page

Table 246 – continued from previous page

_BRACKETS_KV_RE
_BRACKETS_RE_V3
_LIST_EMPTY_ITEM_RE
_BRACKETS_KV_RE_V3
_PARENS_RE
_NAME_VERSION_RE
_CEP26_NAME_RE
_CEP26_VIRTUAL_NAME_RE
_VERSION_BUILD_RE
_PARSE_CACHE
_LIST_NAME_RE
_PLAIN_LIST_ITEM_RE
_PLAIN_SCALAR_ITEM_RE
_YAML_SCALAR_WORDS
_MATCH_SPEC_YAML
_implementors

_BRACKETS_RE: re.Pattern[str]

_BRACKETS_KV_RE: re.Pattern[str]

_BRACKETS_RE_V3: re.Pattern[str]

_LIST_EMPTY_ITEM_RE: re.Pattern[str]

_BRACKETS_KV_RE_V3: re.Pattern[str]

_PARENS_RE: re.Pattern[str]

_NAME_VERSION_RE: re.Pattern[str]

_CEP26_NAME_RE: re.Pattern[str]

_CEP26_VIRTUAL_NAME_RE: re.Pattern[str]

_VERSION_BUILD_RE: re.Pattern[str]

_validate_package_name(*name: str*, *original_spec_str: str*) → None

Validate a package name against CEP-26 rules.

Glob patterns (containing *) are not validated. Names are lowercased before checking since conda normalizes case.

class MatchSpecType

Bases: type

__call__(*spec_arg=None*, ***kwargs*)

class MatchSpec(*optional=False*, *target=None*, ***kwargs*)

The query language for conda packages.

Any of the fields that comprise a PackageRecord can be used to compose a *MatchSpec*.

MatchSpec can be composed with keyword arguments, where keys are any of the attributes of PackageRecord. Values for keyword arguments are the exact values the attribute should match against. Many fields can also be matched against non-exact values--by including wildcard * and >/< ranges--where supported. Any non-specified field is the equivalent of a full wildcard match.

MatchSpec can also be composed using a single positional argument, with optional keyword arguments. Keyword arguments also override any conflicting information provided in the positional argument. The positional argument can be either an existing *MatchSpec* instance or a string. Conda has historically supported more than one string representation for equivalent *MatchSpec* queries. This *MatchSpec* should accept any existing valid spec string, and correctly compose a *MatchSpec* instance.

A series of rules are now followed for creating the canonical string representation of a *MatchSpec* instance. The canonical string representation can generically be represented by

(channel(/subdir):(namespace):)name(version(build))[key1=value1,key2=value2]

where *()* indicate optional fields. The rules for constructing a canonical string representation are:

1. *name* (i.e. "package name") is required, but its value can be '*'. Its position is always outside the key-value brackets.
2. If *version* is an exact version, it goes outside the key-value brackets and is prepended by ==. If *version* is a "fuzzy" value (e.g. *1.11.**), it goes outside the key-value brackets with the *.** left off and is prepended by =. Otherwise *version* is included inside key-value brackets.
3. If *version* is an exact version, and *build* is an exact value, *build* goes outside key-value brackets prepended by a =. Otherwise, *build* goes inside key-value brackets. *build_string* is an alias for *build*.
4. The *namespace* position is being held for a future conda feature.
5. If *channel* is included and is an exact value, a *::* separator is ued between *channel* and *name*. *channel* can either be a canonical channel name or a channel url. In the canonical string representation, the canonical channel name will always be used.
6. If *channel* is an exact value and *subdir* is an exact value, *subdir* is appended to *channel* with a */* separator. Otherwise, *subdir* is included in the key-value brackets.
7. Key-value brackets can be delimited by comma, space, or comma+space. Value can optionally be wrapped in single or double quotes, but must be wrapped if *value* contains a comma, space, or equal sign. The canonical format uses comma delimiters and single quotes.
8. When constructing a *MatchSpec* instance from a string, any key-value pair given inside the key-value brackets overrides any matching parameter given outside the brackets.

When *MatchSpec* attribute values are simple strings, the are interpreted using the following conventions:

- If the string begins with ^ and ends with *$*, it is converted to a regex.
- If the string contains an asterisk (*), it is transformed from a glob to a regex.
- Otherwise, an exact match to the string is sought.

Some fields (*extras*, *flags*) allow list of strings. They must be passed with Python syntax for lists. Quotes are optional like in YAML.

The *when* field expects a string that can be parsed as a *MatchSpec*. This inner spec cannot contain another *when* field.

Examples

```
>>> str(MatchSpec(name='foo', build='py2*', channel='conda-forge'))
'conda-forge::foo[build=py2*]'
>>> str(MatchSpec('foo 1.0 py27_0'))
'foo==1.0=py27_0'
>>> str(MatchSpec('foo=1.0=py27_0'))
'foo==1.0=py27_0'
>>> str(MatchSpec('conda-forge::foo[version=1.0.*]'))
```

(continues on next page)

(continued from previous page)

```
'conda-forge::foo=1.0'
>>> str(MatchSpec('conda-forge/linux-64::foo>=1.0'))
"conda-forge/linux-64::foo[version='>=1.0']"
>>> str(MatchSpec('*/linux-64::foo>=1.0'))
"foo[subdir=linux-64,version='>=1.0']"
>>> str(MatchSpec('package[extras=[a,b]]'))
"package[extras=['a', 'b']]"
>>> str(MatchSpec('package[when=__unix]'))
'package[when=__unix]'
```

To fully-specify a package with a full, exact spec, the fields

- channel
- subdir
- name
- version
- build

must be given as exact values. In the future, the namespace field will be added to this list. Alternatively, an exact spec is given by '*[md5=12345678901234567890123456789012]' or '*[sha256=f453db4ffe2271ec492a2913af4e61d4a6c118201f07de757df0eff769b65d2e]'.

FIELD_NAMES = ('channel', 'subdir', 'name', 'version', 'build', 'build_number', 'track_features', 'features',...

FIELD_NAMES_SET

_MATCHER_CACHE

_optional = False

_target = None

_original_spec_str

_match_components

classmethod from_dist_str(*dist_str*)

get_exact_value(*field_name*)

get_raw_value(*field_name*)

get(*field_name*, *default=None*)

property is_name_only_spec

dist_str()

property optional

property target

property original_spec_str

match(*rec:* conda.models.records.PackageRecord | *dict[str, Any]*) → bool

Accepts a *PackageRecord* or a dict, and matches can pull from any field in that record. Returns True for a match, and False for no match.

_match_individual(*record*, *field_name*, *match_component*)

_is_simple()

_is_single()

_to_filename_do_not_use()

__repr__()

__str__()

__json__()

conda_build_form()

conda_env_form()

Return the package specification in conda environment export format.

This produces the format used by *conda env export*: name=version=build (single equals), without channel prefixes and without .* patterns.

Examples

```
>>> MatchSpec("numpy==1.21.0=py39h1234567_0").conda_env_form()
'numpy=1.21.0=py39h1234567_0'
>>> MatchSpec("numpy=1.21.0").conda_env_form()  # no-builds case
'numpy=1.21.0'
>>> MatchSpec("conda-forge::numpy==1.21.0=py39h1234567_0").conda_env_form()
'numpy=1.21.0=py39h1234567_0'
>>> # channel prefix removed
```

Returns

Package specification in conda env export format

Return type

str

__eq__(*other*)

__hash__()

_hash_key()

__contains__(*field*)

_build_components(***kwargs*)

static _make_component(*field_name*, *value*)

property name

property strictness

property spec

property version

property fn

classmethod merge(*match_specs*, *union=False*)

classmethod union(*match_specs*)

_merge(*other*, *union=False*)

_parse_version_plus_build(*v_plus_b*)

This should reliably pull the build string out of a version + build string combo. .. rubric:: Examples

```
>>> _parse_version_plus_build("=1.2.3 0")
('=1.2.3', '0')
>>> _parse_version_plus_build("1.2.3=0")
('1.2.3', '0')
>>> _parse_version_plus_build(">=1.0 , < 2.0 py34_0")
('>=1.0,<2.0', 'py34_0')
>>> _parse_version_plus_build(">=1.0 , < 2.0 =py34_0")
('>=1.0,<2.0', 'py34_0')
>>> _parse_version_plus_build("=1.2.3 ")
('=1.2.3', None)
>>> _parse_version_plus_build(">1.8,<2|==1.7")
('>1.8,<2|==1.7', None)
>>> _parse_version_plus_build("* openblas_0")
('*', 'openblas_0')
>>> _parse_version_plus_build("* *")
('*', '*')
```

_parse_legacy_dist(*dist_str*)

Examples

```
>>> _parse_legacy_dist("_license-1.1-py27_1.tar.bz2")
('_license', '1.1', 'py27_1')
>>> _parse_legacy_dist("_license-1.1-py27_1")
('_license', '1.1', 'py27_1')
```

_is_non_alias_url_channel(*chn:* conda.models.channel.Channel) → bool

Return `True` if `chn` was constructed from a URL whose location is not `channel_alias` (or a migrated alias).

This is used to decide whether to preserve the explicit URL (`base_url`) of a channel rather than collapsing it through `canonical_name`. The latter flattens URL-form multichannel members such as "https://repo.anaconda.com/pkgs/main" into the multichannel name ("defaults").

_parse_channel(*channel_val*)

_PARSE_CACHE

_sanitize_version_str(*version: str*, *build: str | None*) → str

Sanitize version strings for MatchSpec parsing.

Handles edge cases and translates version patterns for proper MatchSpec processing.

Empty operators like "==" are passed through so existing error handling code can treat them like other incomplete operators ("<=" or ">="). This is necessary because downstream translation code would mangle "==" into an empty string, resulting in an empty version field that breaks logic expecting missing versions to be represented as operators like "==", "<=", and ">=".

These missing version cases result from match specs like "numpy==", "numpy<=", "numpy>=", "numpy= " (with trailing space), which should be treated as errors. Note: "numpy=" (no trailing space) is treated as valid.

For simple versions starting with "=", translates patterns like "=1.2.3" to "1.2.3*" when appropriate conditions are met.

Parameters

- **version** -- The version string to sanitize
- **build** -- Optional build string that affects sanitization behavior

Returns

Sanitized version string

Examples

"==" or "=" -> passed through for error handling "==1.2.3" -> "1.2.3" (when build is None) "=1.2.3" -> "1.2.3*" (when build is None and no special chars) ">=1.0" -> ">=1.0" (unchanged, doesn't start with "=")

_parse_spec_str_dispatcher(*spec_str*) → dict[str, object]

Temporary dispatcher while we introduce the new v3 features

_parse_spec_str(*spec_str*)

_LIST_NAME_RE

_PLAIN_LIST_ITEM_RE

_PLAIN_SCALAR_ITEM_RE

_YAML_SCALAR_WORDS

_MATCH_SPEC_YAML = None

_parse_list_of_str(*inner: str*, *, *list_syntax: bool*) → tuple[str, ...]

Parse names without converting scalar spelling or accepting typed lists.

_parse_spec_str_v3(*spec_str*)

New parser engine only used

_validate_when_spec(*spec*) → None

The value of a *when* expression is a MatchSpec or a boolean expression of MatchSpecs. Parentheses override precedence. We can use Python's parser for this but we need to wrap the MatchSpec as strings first.

```
>>> _validate_when_spec("python")
>>> _validate_when_spec("(python)")
>>> _validate_when_spec("(python and __unix)")
>>> _validate_when_spec("(python and __unix>=3)")
>>> _validate_when_spec("(python and __unix>=3 or __win)")
```

class MatchInterface(*value*)

_raw_value

abstractmethod match(*other*)

matches(*value*)

property raw_value

abstract property exact_value

If the match value is an exact specification, returns the value. Otherwise returns None.

merge(*other*)

union(*other*)

class _StrMatchMixin

__str__()

__repr__()

__eq__(*other*)

__hash__()

property exact_value

class ExactStrMatch(*value*)

Bases: *_StrMatchMixin*, *MatchInterface*

__slots__ = ('_raw_value',)

match(*other*)

class ExactLowerStrMatch(*value*)

Bases: *ExactStrMatch*

match(*other*)

class GlobStrMatch(*value*)

Bases: *_StrMatchMixin*, *MatchInterface*

__slots__ = ('_raw_value', '_re_match')

_re_match = None

match(*other*)

property exact_value

If the match value is an exact specification, returns the value. Otherwise returns None.

property matches_all

merge(*other*)

class GlobLowerStrMatch(*value*)

Bases: *GlobStrMatch*

class SplitStrMatch(*value*)

Bases: *MatchInterface*

__slots__ = ('_raw_value',)

_convert(*value*)

match(*other*)

__repr__()

__str__()

__eq__(*other*)

__hash__()

property exact_value

If the match value is an exact specification, returns the value. Otherwise returns None.

class FeatureMatch(*value*)

Bases: *MatchInterface*

__slots__ = ('_raw_value',)

_convert(*value*)

match(*other*)

__repr__()

__str__()

__eq__(*other*)

__hash__()

property exact_value

If the match value is an exact specification, returns the value. Otherwise returns None.

class ListOfStrMatch(*value*)

Bases: *MatchInterface*

__slots__ = ('_raw_value',)

_convert(*value*)

match(*other*)

__repr__()

__str__

__eq__(*other*)

__hash__()

property exact_value

If the match value is an exact specification, returns the value. Otherwise returns None.

class ChannelMatch(*value*)

Bases: *GlobStrMatch*

_re_match = None

match(*other*)

__str__()

__repr__()

class CaseInsensitiveStrMatch(*value*)

Bases: *GlobLowerStrMatch*

match(*other*)

_implementors

package_info

(Legacy) Low-level implementation of a PackageRecord.

Classes

NoarchField	Fields are doing something very similar to boxing and unboxing
Noarch	
PreferredEnv	
PackageMetadata	
PackageInfo	

class NoarchField(*enum_class*, *default=NULL*, *required=True*, *validation=None*, *in_dump=True*, *default_in_dump=True*, *nullable=False*, *immutable=False*, *aliases=()*)

Bases: *conda.auxlib.entity.EnumField*

Fields are doing something very similar to boxing and unboxing of c#/java primitives. __set__ should take a "primitive" or "raw" value and create a "boxed" or "programmatically usable" value of it. While __get__ should return the boxed value, dump in turn should unbox the value into a primitive or raw value.

Parameters

- **types** (*primitive literal or type or sequence of types*)
- **default** (*any, callable,* optional) -- If default is callable, it's guaranteed to return a valid value at the time of Entity creation.
- **required** (*boolean,* optional)
- **validation** (*callable,* optional)
- **dump** (*boolean,* optional)

box(*instance*, *instance_type*, *val*)

class Noarch(***kwargs*)

Bases: *conda.auxlib.entity.Entity*

type

entry_points

class PreferredEnv(***kwargs*)

Bases: *conda.auxlib.entity.Entity*

name

executable_paths

softlink_paths

class PackageMetadata(***kwargs*)

Bases: *conda.auxlib.entity.Entity*

package_metadata_version

noarch

preferred_env

class PackageInfo(***kwargs*)

Bases: *conda.auxlib.entity.ImmutableEntity*

extracted_package_dir

package_tarball_full_path

channel

repodata_record

url

icondata

package_metadata

paths_data

dist_str()

property name

property version

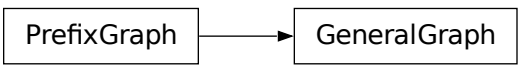
property build

property build_number

prefix_graph

Implements directed graphs to sort and manipulate packages within a prefix.

Object inheritance:



Classes

PrefixGraph	A directed graph structure used for sorting packages (prefix_records) in prefixes and
GeneralGraph	Compared with PrefixGraph, this class takes in more than one record of a given name,

class PrefixGraph(*records: collections.abc.Iterable[*conda.models.records.PrefixRecord*], specs: collections.abc.Iterable[*conda.models.match_spec.MatchSpec*] = ()*)

A directed graph structure used for sorting packages (prefix_records) in prefixes and manipulating packages within prefixes (e.g. removing and pruning).

The terminology used for edge direction is "parents" and "children" rather than "successors" and "predecessors". The parent nodes of a record are those records in the graph that match the record's "depends" field. E.g. NodeA depends on NodeB, then NodeA is a child of NodeB, and NodeB is a parent of NodeA. Nodes can have zero parents, or more than two parents.

Most public methods mutate the graph.

graph: dict[*conda.models.records.PrefixRecord*, dict[*conda.models.records.PrefixRecord*, None]]

spec_matches: dict[*conda.models.records.PrefixRecord*, dict[*conda.models.match_spec.MatchSpec*, None]]

remove_spec(*spec*)

Remove all matching nodes, and any associated child nodes.

Parameters
spec (MatchSpec)

Returns
The removed nodes.

Return type
tuple[*PrefixRecord*]

remove_youngest_descendant_nodes_with_specs()

A specialized method used to determine only dependencies of requested specs.

Returns
The removed nodes.

Return type
tuple[*PrefixRecord*]

property records

prune()

Prune back all packages until all child nodes are anchored by a spec.

Returns
The pruned nodes.

Return type
tuple[*PrefixRecord*]

get_node_by_name(*name*)

all_descendants(*node*)

all_ancestors(*node*)

_remove_node(*node*)

Removes this node and all edges referencing it.

_toposort()

classmethod _toposort_raise_on_cycles(*graph*)

classmethod _topo_sort_handle_cycles(*graph*)

static _toposort_pop_key(*graph*)

Pop an item from the graph that has the fewest parents. In the case of a tie, use the node with the alphabetically-first package name.

static _toposort_prepare_graph(*graph*)

class GeneralGraph(*records*, *specs=()*)

Bases: *PrefixGraph*

Compared with PrefixGraph, this class takes in more than one record of a given name, and operates on that graph from the higher view across any matching dependencies. It is not a Prefix thing, but more like a "graph of all possible candidates" thing, and is used for unsatisfiability analysis

specs_by_name

graph_by_name

breadth_first_search_by_name(*root_spec*, *target_spec*)

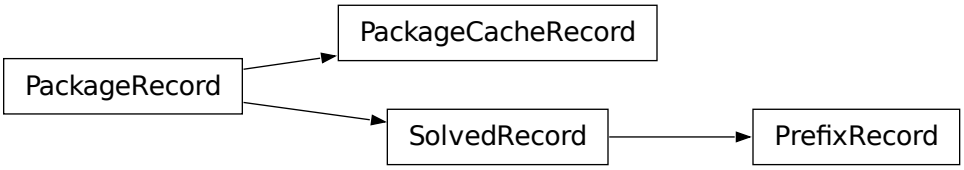
Return shorted path from root_spec to spec_name

records

Implements the data model for conda packages.

A PackageRecord is the record of a package present in a channel. A PackageCache is the record of a downloaded and cached package. A PrefixRecord is the record of a package installed into a conda environment.

Object inheritance:



Classes

LinkTypeField	Fields are doing something very similar to boxing and unboxing
NoarchField	Fields are doing something very similar to boxing and unboxing
TimestampField	Fields are doing something very similar to boxing and unboxing
IndexedTimestampField	Fields are doing something very similar to boxing and unboxing
Link	
_FeaturesField	Fields are doing something very similar to boxing and unboxing
ChannelField	Fields are doing something very similar to boxing and unboxing
SubdirField	Fields are doing something very similar to boxing and unboxing
FilenameField	Fields are doing something very similar to boxing and unboxing
PackageTypeField	Fields are doing something very similar to boxing and unboxing
PathData	
PathDataV1	
PathsData	
PackageRecord	Representation of a concrete package archive (tarball or .conda file).
Md5Field	Fields are doing something very similar to boxing and unboxing
PackageCacheRecord	Representation of a package that has been downloaded or unpacked in the local package cache.
SolvedRecord	Representation of a package that has been returned as part of a solver solution.
PrefixRecord	Representation of a package that is installed in a local conda environmnet.

Functions

_safe_opt_str(val) → tuple[int, str]

Attributes

EMPTY_LINK

_safe_opt_str(*val: Any*) → tuple[int, str]

class LinkTypeField(*enum_class*, *default=NULL*, *required=True*, *validation=None*, *in_dump=True*, *default_in_dump=True*, *nullable=False*, *immutable=False*, *aliases=()*)

Bases: *conda.auxlib.entity.EnumField*

Fields are doing something very similar to boxing and unboxing of c#/java primitives. __set__ should take a

"primitive" or "raw" value and create a "boxed" or "programmatically usable" value of it. While __get__ should return the boxed value, dump in turn should unbox the value into a primitive or raw value.

Parameters

- **types** (*primitive literal or type or sequence of types*)
- **default** (*any, callable,* optional) -- If default is callable, it's guaranteed to return a valid value at the time of Entity creation.
- **required** (*boolean,* optional)
- **validation** (*callable,* optional)
- **dump** (*boolean,* optional)

box(*instance*, *instance_type*, *val*)

class NoarchField(*enum_class*, *default=NULL*, *required=True*, *validation=None*, *in_dump=True*, *default_in_dump=True*, *nullable=False*, *immutable=False*, *aliases=()*)

Bases: *conda.auxlib.entity.EnumField*

Fields are doing something very similar to boxing and unboxing of c#/java primitives. __set__ should take a "primitive" or "raw" value and create a "boxed" or "programmatically usable" value of it. While __get__ should return the boxed value, dump in turn should unbox the value into a primitive or raw value.

Parameters

- **types** (*primitive literal or type or sequence of types*)
- **default** (*any, callable,* optional) -- If default is callable, it's guaranteed to return a valid value at the time of Entity creation.
- **required** (*boolean,* optional)
- **validation** (*callable,* optional)
- **dump** (*boolean,* optional)

box(*instance*, *instance_type*, *val*)

class TimestampField

Bases: *conda.auxlib.entity.NumberField*

Fields are doing something very similar to boxing and unboxing of c#/java primitives. __set__ should take a "primitive" or "raw" value and create a "boxed" or "programmatically usable" value of it. While __get__ should return the boxed value, dump in turn should unbox the value into a primitive or raw value.

Parameters

- **types** (*primitive literal or type or sequence of types*)
- **default** (*any, callable,* optional) -- If default is callable, it's guaranteed to return a valid value at the time of Entity creation.
- **required** (*boolean,* optional)
- **validation** (*callable,* optional)
- **dump** (*boolean,* optional)

static _make_seconds(*val*)

static _make_milliseconds(*val*)

box(*instance*, *instance_type*, *val*)

dump(*instance*, *instance_type*, *val*)

__get__(*instance*, *instance_type*)

class IndexedTimestampField

Bases: *TimestampField*

Fields are doing something very similar to boxing and unboxing of c#/java primitives. __set__ should take a "primitive" or "raw" value and create a "boxed" or "programmatically usable" value of it. While __get__ should return the boxed value, dump in turn should unbox the value into a primitive or raw value.

Parameters

- **types** (*primitive literal or type or sequence of types*)
- **default** (*any, callable,* optional) -- If default is callable, it's guaranteed to return a valid value at the time of Entity creation.
- **required** (*boolean,* optional)
- **validation** (*callable,* optional)
- **dump** (*boolean,* optional)

__get__(*instance*, *instance_type*)

class Link(***kwargs*)

Bases: conda.auxlib.entity.DictSafeMixin, *conda.auxlib.entity.Entity*

source

type

EMPTY_LINK

class _FeaturesField(***kwargs*)

Bases: *conda.auxlib.entity.ListField*

Fields are doing something very similar to boxing and unboxing of c#/java primitives. __set__ should take a "primitive" or "raw" value and create a "boxed" or "programmatically usable" value of it. While __get__ should return the boxed value, dump in turn should unbox the value into a primitive or raw value.

Parameters

- **types** (*primitive literal or type or sequence of types*)
- **default** (*any, callable,* optional) -- If default is callable, it's guaranteed to return a valid value at the time of Entity creation.
- **required** (*boolean,* optional)
- **validation** (*callable,* optional)
- **dump** (*boolean,* optional)

box(*instance*, *instance_type*, *val*)

dump(*instance*, *instance_type*, *val*)

class ChannelField(*aliases=()*)

Bases: *conda.auxlib.entity.ComposableField*

Fields are doing something very similar to boxing and unboxing of c#/java primitives. __set__ should take a "primitive" or "raw" value and create a "boxed" or "programmatically usable" value of it. While __get__ should return the boxed value, dump in turn should unbox the value into a primitive or raw value.

Parameters

- **types** (*primitive literal or type or sequence of types*)
- **default** (*any, callable,* optional) -- If default is callable, it's guaranteed to return a valid value at the time of Entity creation.
- **required** (*boolean,* optional)
- **validation** (*callable,* optional)
- **dump** (*boolean,* optional)

dump(*instance*, *instance_type*, *val*)

__get__(*instance*, *instance_type*)

class SubdirField

Bases: *conda.auxlib.entity.StringField*

Fields are doing something very similar to boxing and unboxing of c#/java primitives. __set__ should take a "primitive" or "raw" value and create a "boxed" or "programmatically usable" value of it. While __get__ should return the boxed value, dump in turn should unbox the value into a primitive or raw value.

Parameters

- **types** (*primitive literal or type or sequence of types*)
- **default** (*any, callable,* optional) -- If default is callable, it's guaranteed to return a valid value at the time of Entity creation.
- **required** (*boolean,* optional)
- **validation** (*callable,* optional)
- **dump** (*boolean,* optional)

__get__(*instance*, *instance_type*)

class FilenameField(*aliases=()*)

Bases: *conda.auxlib.entity.StringField*

Fields are doing something very similar to boxing and unboxing of c#/java primitives. __set__ should take a "primitive" or "raw" value and create a "boxed" or "programmatically usable" value of it. While __get__ should return the boxed value, dump in turn should unbox the value into a primitive or raw value.

Parameters

- **types** (*primitive literal or type or sequence of types*)
- **default** (*any, callable,* optional) -- If default is callable, it's guaranteed to return a valid value at the time of Entity creation.
- **required** (*boolean,* optional)
- **validation** (*callable,* optional)
- **dump** (*boolean,* optional)

__get__(*instance*, *instance_type*)

class PackageTypeField

Bases: *conda.auxlib.entity.EnumField*

Fields are doing something very similar to boxing and unboxing of c#/java primitives. __set__ should take a "primitive" or "raw" value and create a "boxed" or "programmatically usable" value of it. While __get__ should return the boxed value, dump in turn should unbox the value into a primitive or raw value.

Parameters

- **types** (*primitive literal or* type *or sequence of types*)
- **default** (*any, callable,* optional) -- If default is callable, it's guaranteed to return a valid value at the time of Entity creation.
- **required** (*boolean,* optional)
- **validation** (*callable,* optional)
- **dump** (*boolean,* optional)

__get__(*instance*, *instance_type*)

class PathData(***kwargs*)

Bases: *conda.auxlib.entity.Entity*

_path

prefix_placeholder

file_mode

no_link

path_type

property path

class PathDataV1(***kwargs*)

Bases: *PathData*

sha256

size_in_bytes

inode_paths

sha256_in_prefix

class PathsData(***kwargs*)

Bases: *conda.auxlib.entity.Entity*

paths_version

paths

class PackageRecord(*args, **kwargs)

Bases: conda.auxlib.entity.DictSafeMixin, conda.auxlib.entity.Entity

Representation of a concrete package archive (tarball or .conda file).

It captures all the relevant information about a given package archive, including its source, in the following attributes.

Note that there are three subclasses, SolvedRecord, PrefixRecord and PackageCacheRecord. These capture the same information, but are augmented with additional information relevant for these sources of packages.

Further note that PackageRecord makes use of its _pkey for comparison and hash generation. This means that for common operations, like comparisons between PackageRecord s and reference of PackageRecord s in mappings, _different_ objects appear identical. The fields taken into account are marked in the following list of attributes. The subclasses do not add further attributes to the _pkey.

name

The name of the package.

Part of the _pkey.

version

The version of the package.

Part of the _pkey.

build

The build string of the package.

Part of the _pkey.

build_number

The build number of the package.

Part of the _pkey.

channel

The channel where the package can be found.

subdir

The subdir, i.e. `noarch` or a platform (`linux-64` or similar).

Part of the _pkey.

fn

The filename of the package.

Only part of the _pkey if *separate_format_cache* is `true` (default: `false`).

md5

The md5 checksum of the package.

legacy_bz2_md5

If this is a `.conda` package and a corresponding `.tar.bz2` package exists, this may contain the md5 checksum of that package.

legacy_bz2_size

If this is a `.conda` package and a corresponding `.tar.bz2` package exists, this may contain the size of that package.

url

The download url of the package.

sha256

The sha256 checksum of the package.

property channel_name: str | None

The canonical name of the channel of this package.

Part of the *_pkey*.

Type
str

property _pkey

The components of the PackageRecord that are used for comparison and hashing.

The *_pkey* is a tuple made up of the following fields of the *PackageRecord*. Two *PackageRecord* s test equal if their respective *_pkey* s are equal. The hash of the *PackageRecord* (important for dictionary access) is the hash of the *_pkey*.

The included fields are:

- *channel_name*
- *subdir*
- *name*
- *version*
- *build_number*
- *build*
- *fn* only if *separate_format_cache* is set to true (default: false)

Type
tuple

__hash__()

__eq__(*other*)

property _sort_key: tuple

The components used for sorting PackageRecord instances.

Includes (in precedence order): - package name - VersionOrder(version) - build_number (authoritative integer) - BuildString(build) (variant suffix and numeric tokens) - raw version string (deterministic tie-breaker for equivalent VersionOrder) - channel canonical_name (preserving None vs empty string distinction) - subdir (preserving None vs empty string distinction) - fn (if separate_format_cache is enabled)

Type
tuple

__lt__(*other: Any*) → bool

__le__(*other: Any*) → bool

__gt__(*other: Any*) → bool

__ge__(*other: Any*) → bool

dist_str(*canonical_name: bool = True*) → str

dist_fields_dump()

arch

platform

depends

constrains

flags

extra_depends

track_features

features

noarch

preferred_env

python_site_packages_path

license

license_family

package_type

property is_unmanageable

timestamp

indexed_timestamp

property combined_depends

date

size

__str__()

to_match_spec()

to_simple_match_spec()

property namekey

property spec

name=version=build

Type

Return package spec

property spec_no_build

name=version

Type
Return package spec without build

record_id()

metadata: set[str]

classmethod feature(*feature_name*) → *PackageRecord*

classmethod virtual_package(*name: str*, *version: str | None = None*, *build_string: str | None = None*) → *PackageRecord*

Create a virtual package record.

Parameters

- **name** -- The name of the virtual package.
- **version** -- The version of the virtual package, defaults to "0".
- **build_string** -- The build string of the virtual package, defaults to "0".

Returns
A PackageRecord representing the virtual package.

class Md5Field

Bases: *conda.auxlib.entity.StringField*

Fields are doing something very similar to boxing and unboxing of c#/java primitives. __set__ should take a "primitive" or "raw" value and create a "boxed" or "programmatically usable" value of it. While __get__ should return the boxed value, dump in turn should unbox the value into a primitive or raw value.

Parameters

- **types** (*primitive literal or* type *or sequence of types*)
- **default** (*any, callable,* optional) -- If default is callable, it's guaranteed to return a valid value at the time of Entity creation.
- **required** (*boolean,* optional)
- **validation** (*callable,* optional)
- **dump** (*boolean,* optional)

__get__(*instance*, *instance_type*)

class PackageCacheRecord(**args*, ***kwargs*)

Bases: *PackageRecord*

Representation of a package that has been downloaded or unpacked in the local package cache.

Specialization of *PackageRecord* that adds information for packages that exist in the local package cache, either as the downloaded package file, or unpacked in its own package dir, or both.

Note that this class does not add new fields to the *PackageRecord._pkey* so that a pure *PackageRecord* object that has the same _pkey fields as a different *PackageCacheRecord* object (or, indeed, a *PrefixRecord* object) will be considered equal and will produce the same hash.

package_tarball_full_path

Full path to the local package file.

extracted_package_dir

Full path to the local extracted package.

md5

The md5 checksum of the package.

If the package file exists locally, this class can calculate a missing checksum on-the-fly.

property is_fetched

Whether the package file exists locally.

Type
bool

property is_extracted

Whether the package has been extracted locally.

Type
bool

property tarball_basename

The basename of the local package file.

Type
str

matches_metadata(*package_ref:* PackageRecord | conda.models.match_spec.MatchSpec) → bool

Match available size and MD5 against the requested package.

Accept missing values and the requested package's legacy archive metadata.

_calculate_md5sum()

class SolvedRecord(**args*, ***kwargs*)

Bases: *PackageRecord*

Representation of a package that has been returned as part of a solver solution.

This sits between *PackageRecord* and *PrefixRecord*, simply adding `requested_spec` (and `requested_specs`) so they can be used in lockfiles without requiring the artifact on disk.

requested_spec

The `MatchSpec` that the user requested or `None` if the package it was installed as a dependency.

_requested_specs

The `MatchSpec` objects that the user requested or `()` if the package was installed as a dependency. See also *.requested_specs* property.

property requested_specs: tuple[str, ...]

This property will use 'requested_spec' as the source for 'requested_specs' for old *conda-meta/*.json* files that did not define the plural version.

dump()

We need to expose _requested_specs as its public counterpart, and also expose single specs as a plural list for backwards compatibility.

class PrefixRecord(**args*, ***kwargs*)

Bases: *SolvedRecord*

Representation of a package that is installed in a local conda environmnet.

Specialization of *PackageRecord* that adds information for packages that are installed in a local conda environment or prefix.

Note that this class does not add new fields to the *PackageRecord._pkey* so that a pure *PackageRecord* object that has the same `_pkey` fields as a different *PrefixRecord* object (or, indeed, a *PackageCacheRecord* object) will be considered equal and will produce the same hash.

Objects of this class are generally constructed from metadata in json files inside *$prefix/conda-meta*.

package_tarball_full_path

The path to the originating package file, usually in the local cache.

extracted_package_dir

The path to the extracted package directory, usually in the local cache.

files

The list of all files comprising the package as relative paths from the prefix root.

paths_data

List with additional information about the files, e.g. checksums and link type.

link

Information about how the package was linked into the prefix.

auth

Authentication information.

_get_json_fn() → str

package_size(*prefix_path: pathlib.Path*) → int

Compute the installed size of this package within a prefix.

This sums up the size_in_bytes of all non-softlink paths in paths_data, and stats the files on disk if size_in_bytes is missing and for the package's conda-meta JSON manifest.

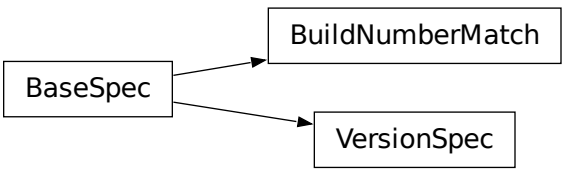
Returns

Total size in bytes of all files installed by this package.

version

Implements the version spec with parsing and comparison logic.

Object inheritance:



Classes

SingleStrArgCachingType	
VersionOrder	Implement an order relation between version strings.
BaseSpec	
VersionSpec	
BuildNumberMatch	

Functions

normalized_version(version) → VersionOrder	Parse a version string and return VersionOrder object.
ver_eval(vtest, spec)	
treeify(spec_str)	
untreeify(spec[, _inand, depth])	
compatible_release_operator(x, y)	

Attributes

version_check_re
version_split_re
version_cache
VSPEC_TOKENS
version_relation_re
regex_split_re
OPERATOR_MAP
OPERATOR_START
VersionMatch

normalized_version(*version: str*) → *VersionOrder*

Parse a version string and return VersionOrder object.

ver_eval(*vtest*, *spec*)

version_check_re

version_split_re

version_cache

class SingleStrArgCachingType

Bases: `type`

__call__(*arg*)

class VersionOrder(*vstr: str*)

Implement an order relation between version strings.

Version strings can contain the usual alphanumeric characters (A-Za-z0-9), separated into components by dots and underscores. Empty segments (i.e. two consecutive dots, a leading/trailing underscore) are not permitted. An optional epoch number - an integer followed by '!' - can proceed the actual version string (this is useful to indicate a change in the versioning scheme itself). Version comparison is case-insensitive.

Conda supports six types of version strings: * Release versions contain only integers, e.g. '1.0', '2.3.5'. * Pre-release versions use additional letters such as 'a' or 'rc',

for example '1.0a1', '1.2.beta3', '2.3.5rc3'.

- Development versions are indicated by the string 'dev', for example '1.0dev42', '2.3.5.dev12'.
- Post-release versions are indicated by the string 'post', for example '1.0post1', '2.3.5.post2'.
- Tagged versions have a suffix that specifies a particular property of interest, e.g. '1.1.parallel'. Tags can be added to any of the preceding four types. As far as sorting is concerned, tags are treated like strings in pre-release versions.
- An optional local version string separated by '+' can be appended to the main (upstream) version string. It is only considered in comparisons when the main versions are equal, but otherwise handled in exactly the same manner.

To obtain a predictable version ordering, it is crucial to keep the version number scheme of a given package consistent over time. Specifically, * version strings should always have the same number of components

(except for an optional tag suffix or local version string),

- letters/strings indicating non-release versions should always occur at the same position.

Before comparison, version strings are parsed as follows: * They are first split into epoch, version number, and local version

number at '!' and '+' respectively. If there is no '!', the epoch is set to 0. If there is no '+', the local version is empty.

- The version part is then split into components at '.' and '_'.
- Each component is split again into runs of numerals and non-numerals
- Subcomponents containing only numerals are converted to integers.
- Strings are converted to lower case, with special treatment for 'dev' and 'post'.
- When a component starts with a letter, the fillvalue 0 is inserted to keep numbers and strings in phase, resulting in '1.1.a1' == 1.1.0a1'.
- The same is repeated for the local version part.

Examples

1.2g.beta15.rc => [[0], [1], [2, 'g'], [0, 'beta', 15], [0, 'rc']] 1!2.15.1_ALPHA => [[1], [2], [15], [1, '_alpha']]

The resulting lists are compared lexicographically, where the following rules are applied to each pair of corresponding subcomponents: * integers are compared numerically * strings are compared lexicographically, case-insensitive * strings are smaller than integers, except * 'dev' versions are smaller than all corresponding versions of other types * 'post' versions are greater than all corresponding versions of other types * if a subcomponent has no correspondent, the missing correspondent is

treated as integer 0 to ensure '1.1' == '1.1.0'.

The resulting order is:

0.4

< 0.4.0 < 0.4.1.rc

== 0.4.1.RC # case-insensitive comparison
< 0.4.1 < 0.5a1 < 0.5b3 < 0.5C1 # case-insensitive comparison < 0.5 < 0.9.6 < 0.960923 < 1.0 < 1.1dev1 # special case 'dev' < 1.1_ # appended underscore is special case for openssl-like versions < 1.1a1 < 1.1.0dev1 # special case 'dev'

== 1.1.dev1 # 0 is inserted before string
< 1.1.a1 < 1.1.0rc1 < 1.1.0

== 1.1
< 1.1.0post1 # special case 'post'

== 1.1.post1 # 0 is inserted before string
< 1.1post1 # special case 'post' < 1996.07.12 < 1!0.4.1 # epoch increased < 1!3.1.1.6 < 2!0.4.1 # epoch increased again

Some packages (most notably openssl) have incompatible version conventions. In particular, openssl interprets letters as version counters rather than pre-release identifiers. For openssl, the relation

1.0.1 < 1.0.1a => False # should be true for openssl

holds, whereas conda packages use the opposite ordering. You can work-around this problem by appending an underscore to plain version numbers:

1.0.1_ < 1.0.1a => True # ensure correct ordering for openssl

cache

norm_version

fillvalue = 0

version

__str__() → str

__repr__() → str

_eq(*t1: list[str]*, *t2: list[str]*) → bool

__eq__(*other: object*) → bool

startswith(*other: object*) → bool

__ne__(*other: object*) → bool

__lt__(*other: object*) → bool

__gt__(*other: object*) → bool

__le__(*other: object*) → bool

__ge__(*other: object*) → bool

VSPEC_TOKENS = '\\s*\\^[^$]*[$]|\\s*[()|,]|[^()|,]+'

treeify(*spec_str*)

Examples

```
>>> treeify("1.2.3")
'1.2.3'
>>> treeify("1.2.3,>4.5.6")
(',', '1.2.3', '>4.5.6')
>>> treeify("1.2.3,4.5.6|<=7.8.9")
('|', (',', '1.2.3', '4.5.6'), '<=7.8.9')
>>> treeify("(1.2.3|4.5.6),<=7.8.9")
(',', ('|', '1.2.3', '4.5.6'), '<=7.8.9')
>>> treeify("((1.5|((1.6|1.7), 1.8), 1.9 |2.0))|2.1")
('|', '1.5', (',', ('|', '1.6', '1.7'), '1.8', '1.9'), '2.0', '2.1')
>>> treeify("1.5|(1.6|1.7),1.8,1.9|2.0|2.1")
('|', '1.5', (',', ('|', '1.6', '1.7'), '1.8', '1.9'), '2.0', '2.1')
```

untreeify(*spec*, *_inand=False*, *depth=0*)

Examples

```
>>> untreeify('1.2.3')
'1.2.3'
>>> untreeify((',', '1.2.3', '>4.5.6'))
'1.2.3,>4.5.6'
>>> untreeify(('|', (',', '1.2.3', '4.5.6'), '<=7.8.9'))
'(1.2.3,4.5.6)|<=7.8.9'
>>> untreeify((',', ('|', '1.2.3', '4.5.6'), '<=7.8.9'))
'(1.2.3|4.5.6),<=7.8.9'
>>> untreeify(('|', '1.5', (',', ('|', '1.6', '1.7'), '1.8', '1.9'), '2.0', '2.1'))
'1.5|((1.6|1.7),1.8,1.9)|2.0|2.1'
```

compatible_release_operator(*x*, *y*)

version_relation_re

regex_split_re

OPERATOR_MAP

OPERATOR_START

class BaseSpec(*spec_str*, *matcher*, *is_exact*)

spec_str

_is_exact

match

property spec

is_exact()

__eq__(*other*)

__ne__(*other*)

__hash__()

__str__()

__repr__()

property raw_value

property exact_value

abstractmethod merge(*other*)

regex_match(*spec_str*)

operator_match(*spec_str*)

any_match(*spec_str*)

all_match(*spec_str*)

exact_match(*spec_str*)

always_true_match(*spec_str*)

class VersionSpec(*vspec*)

Bases: *BaseSpec*

cache

get_matcher(*vspec*)

merge(*other*)

union(*other*)

VersionMatch

class BuildNumberMatch(*vspec*)

Bases: *BaseSpec*

cache

get_matcher(*vspec*)

merge(*other*)

union(*other*)

property exact_value: int | None

__str__()

__repr__()

notices

cache

Handles all caching logic including:

- Retrieving from cache
- Saving to cache
- Determining whether not certain items have expired and need to be refreshed

Functions

cached_response(func)	
is_notice_response_cache_expired(channel_notic → bool	This checks the contents of the cache response to see if it is expired.
get_notices_cache_dir() → pathlib.Path	Returns the location of the notices cache directory as a Path object
get_notices_cache_file() → pathlib.Path	Return path of notices cache
get_notice_response_from_cache(url, name, cache_dir) → conda.notices.types.ChannelNoticeResponse | None	Retrieves a notice response object from cache if it exists.
write_notice_response_to_cache(...) → None	Writes our notice data to our local cache location.
mark_channel_notices_as_viewed(cache_file, channel_notices) → None	Insert channel notice into our database marking it as read.
get_viewed_channel_notice_ids(cache_file, channel_notices) → set[str]	Return the ids of the channel notices which have already been seen.
clear_cache() → None	Invalidate the notices cache so notices will be retrieved and displayed
_clear_cache_file(cache_file) → None	

Attributes

logger

logger

cached_response(*func*)

is_notice_response_cache_expired(*channel_notice_response:* conda.notices.types.ChannelNoticeResponse) → bool

This checks the contents of the cache response to see if it is expired.

If for whatever reason we encounter an exception while parsing the individual messages, we assume an invalid cache and return true.

get_notices_cache_dir() → pathlib.Path

Returns the location of the notices cache directory as a Path object

get_notices_cache_file() → pathlib.Path

Return path of notices cache

If the file does not exist, we create it with natural filesystem timestamps, then set only the modification time to be in the past. This ensures notices are checked and displayed immediately rather than waiting for the full display

interval.

get_notice_response_from_cache(*url: str*, *name: str*, *cache_dir: pathlib.Path*) → *conda.notices.types.ChannelNoticeResponse* | None

Retrieves a notice response object from cache if it exists.

write_notice_response_to_cache(*channel_notice_response:* conda.notices.types.ChannelNoticeResponse, *cache_dir: pathlib.Path*) → None

Writes our notice data to our local cache location.

mark_channel_notices_as_viewed(*cache_file: pathlib.Path*, *channel_notices:* *collections.abc.Sequence*[conda.notices.types.ChannelNotice]) → None

Insert channel notice into our database marking it as read.

get_viewed_channel_notice_ids(*cache_file: pathlib.Path*, *channel_notices:* *collections.abc.Sequence*[conda.notices.types.ChannelNotice]) → set[str]

Return the ids of the channel notices which have already been seen.

clear_cache() → None

Invalidate the notices cache so notices will be retrieved and displayed on the next command invocation.

For the `notices.cache` file we rewind its mtime into the past instead of removing it. On Windows the file can be transiently locked (e.g. by antivirus or lingering file handles), in which case `unlink()` raises `PermissionError` and the cache would otherwise remain with a recent `touch()`-ed mtime, suppressing notices on subsequent runs.

Per-channel cached response files are still removed when possible, but any `OSError` raised while unlinking them is ignored for the same reason.

_clear_cache_file(*cache_file: conda.common.path.PathType*) → None

core

Core conda notices logic.

Functions

retrieve_notices([limit, always_show_viewed, silent]) → conda.notices.types.ChannelNoticeResultSet	Function used for retrieving notices. This is called by the "notices" decorator as well
display_notices(channel_notice_set) → None	Prints the channel notices to std out.
notices(func)	Wrapper for "execute" entry points for subcommands.
get_channel_name_and_urls(channels) → list[tuple[ChannelUrl, ChannelName]]	Return a sequence of Channel URL and name tuples.
flatten_notice_responses(channel_notice_response → collections.abc.Sequence[conda.notices.types.Channel	
filter_notices(channel_notices[, limit, exclude]) → collections.abc.Sequence[conda.notices.types.ChannelNotice]	Perform filtering actions for the provided sequence of ChannelNotice objects.
is_channel_notices_enabled(ctx) → bool	Determines whether channel notices are enabled and therefore displayed when
is_channel_notices_cache_expired() → bool	Checks to see if the notices cache file we use to keep track of

Attributes

ChannelName
ChannelUrl
logger

ChannelName

ChannelUrl

logger

retrieve_notices(*limit: int | None = None*, *always_show_viewed: bool = True*, *silent: bool = False*) → *conda.notices.types.ChannelNoticeResultSet*

Function used for retrieving notices. This is called by the "notices" decorator as well as the sub-command "notices"

Parameters

- **limit** -- Limit the number of notices to show (defaults to None).
- **always_show_viewed** -- Whether all notices should be shown, not only the unread ones (defaults to True).
- **silent** -- Whether to use a spinner when fetching and caching notices.

display_notices(*channel_notice_set:* conda.notices.types.ChannelNoticeResultSet) → None

Prints the channel notices to std out.

notices(*func*)

Wrapper for "execute" entry points for subcommands.

If channel notices need to be fetched, we do that first and then run the command normally. We then display these notices at the very end of the command output so that the user is more likely to see them.

This ordering was specifically done to address the following bug report:

- https://github.com/conda/conda/issues/11847

Parameters

func -- Function to be decorated

get_channel_name_and_urls(*channels: collections.abc.Sequence[*conda.models.channel.Channel | conda.models.channel.MultiChannel*]*) → list[tuple[*ChannelUrl*, *ChannelName*]]

Return a sequence of Channel URL and name tuples.

This function handles both Channel and MultiChannel object types.

flatten_notice_responses(*channel_notice_responses: collections.abc.Sequence[*conda.notices.types.ChannelNoticeResponse*]*) → collections.abc.Sequence[*conda.notices.types.ChannelNotice*]

filter_notices(*channel_notices: collections.abc.Sequence[*conda.notices.types.ChannelNotice*]*, *limit: int | None = None*, *exclude: set[str] | None = None*) → collections.abc.Sequence[*conda.notices.types.ChannelNotice*]

Perform filtering actions for the provided sequence of ChannelNotice objects.

is_channel_notices_enabled(*ctx:* conda.base.context.Context) → bool

Determines whether channel notices are enabled and therefore displayed when invoking the *notices* command decorator.

This only happens when:

- offline is False
- number_channel_notices is greater than 0

Parameters
ctx -- The conda context object

is_channel_notices_cache_expired() → bool

Checks to see if the notices cache file we use to keep track of displayed notices is expired. This involves checking the mtime attribute of the file. Anything older than what is specified as the NOTICES_DECORATOR_DISPLAY_INTERVAL_NS is considered expired.

fetch

Notices network fetch logic.

Functions

get_notice_responses(url_and_names[, silent, max_workers]) → collections.abc.Sequence[conda.notices.types.ChannelNoticeR	Provided a list of channel notification url/name tuples, return a sequence of
get_channel_notice_response(url, name) → conda.notices.types.ChannelNoticeResponse | None	Return a channel response object. We use this to wrap the response with

Attributes

logger

logger

get_notice_responses(*url_and_names: collections.abc.Sequence[tuple[str, str]]*, *silent: bool = False*, *max_workers: int = 10*) → collections.abc.Sequence[*conda.notices.types.ChannelNoticeResponse*]

Provided a list of channel notification url/name tuples, return a sequence of ChannelNoticeResponse objects.

Parameters

- **url_and_names** -- channel url and the channel name
- **silent** -- turn off "loading animation" (defaults to False)
- **max_workers** -- increase worker number in thread executor (defaults to 10)

Returns
Sequence[ChannelNoticeResponse]

get_channel_notice_response(*url: str*, *name: str*) → *conda.notices.types.ChannelNoticeResponse* | None

Return a channel response object. We use this to wrap the response with additional channel information to use. If the response was invalid we suppress/log and error message.

types

Implements all conda.notices types.

Classes

ChannelNotice	Represents an individual channel notice.
ChannelNoticeResultSet	Represents a list of a channel notices, plus some accompanying
ChannelNoticeResponse	

Attributes

UNDEFINED_MESSAGE_ID	Value to use for message ID when it is not provided.

UNDEFINED_MESSAGE_ID = 'undefined'

Value to use for message ID when it is not provided.

class ChannelNotice

Bases: NamedTuple

Represents an individual channel notice.

id: str

channel_name: str | None

message: str | None

level: *conda.base.constants.NoticeLevel*

created_at: datetime.datetime | None

expired_at: datetime.datetime | None

interval: int | None

to_dict()

class ChannelNoticeResultSet

Bases: NamedTuple

Represents a list of a channel notices, plus some accompanying metadata such as *viewed_channel_notices*.

channel_notices: collections.abc.Sequence[*ChannelNotice*]

Channel notices that are included in this particular set.

total_number_channel_notices: int

Total number of channel notices; not just the ones that will be displayed.

viewed_channel_notices: int

The number of channel notices that have already been viewed.

class ChannelNoticeResponse

Bases: NamedTuple

url: str

name: str

json_data: dict | None

property notices: collections.abc.Sequence[*ChannelNotice*]

static _parse_notice_level(*level: str | None*) → *conda.base.constants.NoticeLevel*

We use this to validate notice levels and provide reasonable defaults if any are invalid.

static _parse_iso_timestamp(*iso_timestamp: str | None*) → datetime.datetime | None

Parse ISO timestamp and fail over to a default value of none.

classmethod get_cache_key(*url: str*, *cache_dir: pathlib.Path*) → pathlib.Path

Returns where this channel response will be cached by hashing the URL.

views

Handles all display/view logic.

Functions

print_notices(channel_notices)	Accepts a list of channel notice responses and prints a display.
print_notice_message(notice[, indent]) → None	Prints a single channel notice.
print_more_notices_message(total_notices, ...) → None	Conditionally shows a message informing users how many more message there are.

print_notices(*channel_notices: collections.abc.Sequence[*conda.notices.types.ChannelNotice*]*)

Accepts a list of channel notice responses and prints a display.

Parameters
channel_notices -- A sequence of ChannelNotice objects.

print_notice_message(*notice:* conda.notices.types.ChannelNotice, *indent: str = ' '*) → None

Prints a single channel notice.

print_more_notices_message(*total_notices: int*, *displayed_notices: int*, *viewed_notices: int*) → None

Conditionally shows a message informing users how many more message there are.

Functions

notices(func)	Wrapper for "execute" entry points for subcommands.

notices(*func*)

Wrapper for "execute" entry points for subcommands.

If channel notices need to be fetched, we do that first and then run the command normally. We then display these notices at the very end of the command output so that the user is more likely to see them.

This ordering was specifically done to address the following bug report:

- https://github.com/conda/conda/issues/11847

Parameters
func -- Function to be decorated

plugins

In this module, you will find everything relevant to conda's plugin system. It contains all of the code that plugin authors will use to write plugins, as well as conda's internal implementations of plugins.

Modules relevant for plugin authors

- *conda.plugins.hookspec*: all available hook specifications are listed here, including examples of how to use them
- *conda.plugins.types*: important types to use when defining plugin hooks

Modules relevant for internal development

- *conda.plugins.manager*: includes our custom subclass of pluggy's PluginManager class

Modules with internal plugin implementations

- *conda.plugins.previews*: aggregator for opt-in preview features
- *conda.plugins.solvers*: implementation of the "classic" solver
- *conda.plugins.subcommands.doctor*: `conda doctor` and `conda check` subcommands (with `--fix` support)
- *conda.plugins.virtual_packages*: registers virtual packages in conda

config

Handles plugin configuration functionality including:

- Managing plugin-specific settings
- Processing configuration data from various sources
- Dynamically adding and removing plugin settings
- Providing a standardized interface for plugins to access their configurations

Classes

PluginConfig	Class used to hold settings for conda plugins.

class PluginConfig(*data*)

Bases: *conda.common.configuration.Configuration*

Class used to hold settings for conda plugins.

The object created by this class should only be accessed via *conda.base.context.Context.plugins*.

When this class is updated via the *add_plugin_setting()* function it adds new setting properties which can be accessed later via the context object.

We currently call that function in *conda.plugins.manager.CondaPluginManager.load_settings()*. because `CondaPluginManager` has access to all registered plugin settings via the settings plugin hook.

parameter_names: `tuple[str, ...] = ()`

parameter_names_and_aliases: `tuple[str, ...] = ()`

classmethod add_plugin_setting(*name: str*, *parameter:* conda.common.configuration.Parameter, *aliases: tuple[str, ...] = ()*)

Adds a setting to the *PluginConfig* class

classmethod remove_all_plugin_settings() → None

Removes all attached settings from the *PluginConfig* class

cache

_data

property raw_data: dict[pathlib.Path, dict[str, *conda.common.configuration.RawParameter*]]

This is used to move everything under the key "plugins" from the provided dictionary to the top level of the returned dictionary. The returned dictionary is then passed to *PluginConfig*.

property category_map: dict[str, tuple[str, ...]]

get_descriptions() → dict[str, str]

describe_parameter(*parameter_name*) → dict[str, Any]

Returns the description of a parameter.

We add to this method in order to change the "name" key that is returned to prepend "plugins." to it.

environment_exporters

Built-in conda environment exporter plugins.

environment_yml

Built-in conda standard environment exporter plugins (YAML and JSON).

Functions

to_dict(env) → dict[str, Any]	Convert Environment to standard dictionary format used by YAML and JSON exporters.
export_yaml(env) → str	Export Environment to YAML format.
export_json(env) → str	Export Environment to JSON format.
conda_environment_exporters()	Register the built-in YAML and JSON environment exporters.

Attributes

ENVIRONMENT_JSON_FORMAT	The name of the environment-json format.
ENVIRONMENT_YAML_FORMAT	The name of the environment-yaml format.
EnvironmentFormatType	Type alias for format validation.

ENVIRONMENT_JSON_FORMAT: Final = 'environment-json'

The name of the environment-json format.

ENVIRONMENT_YAML_FORMAT: Final = 'environment-yaml'

The name of the environment-yaml format.

EnvironmentFormatType

Type alias for format validation.

to_dict(*env:* conda.models.environment.Environment) → dict[str, Any]

Convert Environment to standard dictionary format used by YAML and JSON exporters.

This represents the common dictionary structure that both YAML and JSON environment formats use.

Parameters
env -- Environment model to convert

Returns
Dictionary with standard environment fields

export_yaml(*env:* conda.models.environment.Environment) → str

Export Environment to YAML format.

export_json(*env:* conda.models.environment.Environment) → str

Export Environment to JSON format.

conda_environment_exporters()

Register the built-in YAML and JSON environment exporters.

explicit

Built-in conda explicit environment exporter plugin.

This module implements the explicit format defined in CEP 23: Files with @EXPLICIT marker and package URLs for reproducible installs.

Functions

export_explicit(env) → str	Export Environment to explicit format with @EXPLICIT and URLs (CEP 23 compliant).
conda_environment_exporters()	Environment exporter plugin for explicit format.

Attributes

EXPLICIT_FORMAT	The name of the explicit format.

EXPLICIT_FORMAT: Final = 'explicit'

The name of the explicit format.

export_explicit(*env:* conda.models.environment.Environment) → str

Export Environment to explicit format with @EXPLICIT and URLs (CEP 23 compliant).

conda_environment_exporters()

Environment exporter plugin for explicit format.

requirements_txt

Built-in conda requirements environment exporter plugin.

This module implements the requirements format defined in CEP 23: Files with MatchSpec strings (no @EXPLICIT marker) for flexible package specifications.

Functions

export_requirements(env) → str	Export Environment to requirements format with MatchSpecs (CEP 23 compliant).
conda_environment_exporters()	Environment exporter plugin for requirements format.

Attributes

REQUIREMENTS_FORMAT	The name of the requirements format.

REQUIREMENTS_FORMAT: Final = 'requirements'

The name of the requirements format.

export_requirements(*env:* conda.models.environment.Environment) → str

Export Environment to requirements format with MatchSpecs (CEP 23 compliant).

conda_environment_exporters()

Environment exporter plugin for requirements format.

plugins

The list of environment exporter plugins for easier registration with pluggy.

environment_specifiers

Register the built-in environment specifier hook implementations.

cep_24

EXPERIMENTAL Register the conda env spec for environment.yml files.

Functions

conda_environment_specifiers()	

conda_environment_specifiers()

environment_yml

EXPERIMENTAL Register the conda env spec for environment.yml files.

Functions

conda_environment_specifiers()	

conda_environment_specifiers()

explicit

EXPERIMENTAL Register the conda env spec for explicit files.

Functions

conda_environment_specifiers()	

conda_environment_specifiers()

requirements_txt

EXPERIMENTAL Register the conda env spec for requirements.txt files.

Functions

conda_environment_specifiers()	

conda_environment_specifiers()

plugins

The list of environment specifier plugins for easier registration with pluggy.

hookspec

Pluggy hook specifications ("hookspecs") to register conda plugins.

Each hookspec defined in *CondaSpecs* contains an example of how to use it.

Classes

CondaSpecs	Collection of all supported conda plugin hookspecs.

Attributes

_hookspec	Decorator to mark conda plugin hook specifications, used to register plugin hook types.
hookimpl	Decorator to mark plugin hook implementations, used to register plugins.

_hookspec

Decorator to mark conda plugin hook specifications, used to register plugin hook types.

hookimpl

Decorator to mark plugin hook implementations, used to register plugins.

class CondaSpecs

Collection of all supported conda plugin hookspecs.

conda_solvers() → collections.abc.Iterable[*conda.plugins.types.CondaSolver*]

Register solvers in conda.

Example:

```
import logging

from conda import plugins
from conda.core import solve

log = logging.getLogger(__name__)


class VerboseSolver(solve.Solver):
    def solve_final_state(self, *args, **kwargs):
        log.info("My verbose solver!")
        return super().solve_final_state(*args, **kwargs)


@plugins.hookimpl
def conda_solvers():
    yield plugins.types.CondaSolver(
        name="verbose-classic",
        backend=VerboseSolver,
    )
```

Returns

An iterable of solver entries.

conda_subcommands() → collections.abc.Iterable[*conda.plugins.types.CondaSubcommand*]

Register external subcommands in conda.

Example:

```
from conda import plugins
```

(continues on next page)

(continued from previous page)

```
def example_command(args):
    print("This is an example command!")


@plugins.hookimpl
def conda_subcommands():
    yield plugins.types.CondaSubcommand(
        name="example",
        aliases=("example-alias",),
        summary="example command",
        action=example_command,
    )
```

Returns
An iterable of subcommand entries.

conda_virtual_packages() → collections.abc.Iterable[*conda.plugins.types.CondaVirtualPackage*]

Register virtual packages in Conda.

Example:

```
from conda import plugins


@plugins.hookimpl
def conda_virtual_packages():
    yield plugins.types.CondaVirtualPackage(
        name="my_custom_os",
        version="1.2.3",
        build="x86_64",
    )
```

Returns
An iterable of virtual package entries.

conda_pre_commands() → collections.abc.Iterable[*conda.plugins.types.CondaPreCommand*]

Register pre-command functions in conda.

Shell activation commands avoid loading the plugin manager for startup performance. These hooks run for `conda shell.* activate`, `deactivate`, `reactivate`, and `hook` only if the plugin manager was already loaded by another code path.

Example:

```
from conda import plugins


def example_pre_command(command):
    print("pre-command action")


@plugins.hookimpl
```

(continues on next page)

(continued from previous page)

```
def conda_pre_commands():
    yield plugins.types.CondaPreCommand(
        name="example-pre-command",
        action=example_pre_command,
        run_for={"install", "create"},
    )
```

conda_post_commands() → collections.abc.Iterable[*conda.plugins.types.CondaPostCommand*]

Register post-command functions in conda.

Shell activation commands avoid loading the plugin manager for startup performance. These hooks run for `conda shell.* activate`, `deactivate`, `reactivate`, and `hook` only if the plugin manager was already loaded by another code path.

Example:

```
from conda import plugins


def example_post_command(command):
    print("post-command action")


@plugins.hookimpl
def conda_post_commands():
    yield plugins.types.CondaPostCommand(
        name="example-post-command",
        action=example_post_command,
        run_for={"install", "create"},
    )
```

conda_auth_handlers() → collections.abc.Iterable[*conda.plugins.types.CondaAuthHandler*]

Register a conda auth handler derived from the requests API.

This plugin hook allows attaching requests auth handler subclasses, e.g. when authenticating requests against individual channels hosted at HTTP/HTTPS services.

Example:

```
import os
from conda import plugins
from requests.auth import AuthBase


class EnvironmentHeaderAuth(AuthBase):
    def __init__(self, *args, **kwargs):
        self.username = os.environ["EXAMPLE_CONDA_AUTH_USERNAME"]
        self.password = os.environ["EXAMPLE_CONDA_AUTH_PASSWORD"]

    def __call__(self, request):
        request.headers["X-Username"] = self.username
        request.headers["X-Password"] = self.password
        return request
```

(continues on next page)

(continued from previous page)

```
@plugins.hookimpl
def conda_auth_handlers():
    yield plugins.types.CondaAuthHandler(
        name="environment-header-auth",
        handler=EnvironmentHeaderAuth,
    )
```

conda_health_checks() → collections.abc.Iterable[*conda.plugins.types.CondaHealthCheck*]

Register health checks for conda doctor.

This plugin hook allows you to add more "health checks" to conda doctor that you can write to diagnose problems in your conda environment. Check out the health checks already shipped with conda for inspiration.

Health checks can optionally provide a `fixer` callable that is invoked via `conda doctor --fix` or `conda doctor --fix <check-name>`.

Example (check only):

```
from conda import plugins


def example_health_check(prefix: str, verbose: bool):
    print("This is an example health check!")


@plugins.hookimpl
def conda_health_checks():
    yield plugins.types.CondaHealthCheck(
        name="example-health-check",
        action=example_health_check,
    )
```

Example (check with fix):

```
from conda import plugins


def my_health_check(prefix: str, verbose: bool):
    # Check and report issues
    print("Checking for issues...")


def my_health_fix(prefix: str, args) -> int:
    # Fix the issues
    print("Fixing issues...")
    return 0  # exit code


@plugins.hookimpl
def conda_health_checks():
    yield plugins.types.CondaHealthCheck(
        name="my-check",
```

(continues on next page)

(continued from previous page)

```
        action=my_health_check,
        fixer=my_health_fix,
        summary="Check for common issues",
        fix="Repair detected issues",
    )
```

Returns
An iterable of health check entries.

conda_pre_transaction_actions() → collections.abc.Iterable[*conda.plugins.types.CondaPreTransactionAction*]

Register pre-transaction hooks.

Pre-transaction hooks run before all other actions run in a UnlinkLinkTransaction. For information about the Action class, see *Action*.

Example:

```
from conda import plugins
from conda.plugins.types import CondaPreTransactionAction
from conda.core.path_actions import Action


class PrintAction(Action):
    def verify(self):
        print("Performing verification...")
        self._verified = True

    def execute(self):
        print(
            self.transaction_context,
            self.target_prefix,
            self.unlink_precs,
            self.link_precs,
            self.remove_specs,
            self.update_specs,
            self.neutered_specs,
        )

    def reverse(self):
        print("Reversing only happens when `execute` raises an exception.")

    def cleanup(self):
        print("Carrying out cleanup...")


class PrintActionPlugin:
    @plugins.hookimpl
    def conda_pre_transaction_actions(
        self,
    ) -> Iterable[CondaPreTransactionAction]:
        yield CondaPreTransactionAction(
```

(continues on next page)

(continued from previous page)

```
            name="example-pre-transaction-action",
            action=PrintAction,
        )
```

conda_post_transaction_actions() → collections.abc.Iterable[*conda.plugins.types.CondaPostTransactionAction*]

Register post-transaction hooks.

Post-transaction hooks run after all other actions run in a UnlinkLinkTransaction. For information about the Action class, see *Action*.

Example:

```
from conda import plugins
from conda.plugins.types import CondaPostTransactionAction
from conda.core.path_actions import Action


class PrintAction(Action):
    def verify(self):
        print("Performing verification...")
        self._verified = True

    def execute(self):
        print(
            self.transaction_context,
            self.target_prefix,
            self.unlink_precs,
            self.link_precs,
            self.remove_specs,
            self.update_specs,
            self.neutered_specs,
        )

    def reverse(self):
        print("Reversing only happens when `execute` raises an exception.")

    def cleanup(self):
        print("Carrying out cleanup...")


class PrintActionPlugin:
    @plugins.hookimpl
    def conda_post_transaction_actions(
        self,
    ) -> Iterable[CondaPostTransactionAction]:
        yield CondaPostTransactionAction(
            name="example-post-transaction-action",
            action=PrintAction,
        )
```

conda_pre_solves() → collections.abc.Iterable[*conda.plugins.types.CondaPreSolve*]

Register pre-solve functions in conda that are used in the general solver API, before the solver processes the package specs in search of a solution.

Example:

```
from conda import plugins
from conda.models.match_spec import MatchSpec


def example_pre_solve(
    specs_to_add: frozenset[MatchSpec],
    specs_to_remove: frozenset[MatchSpec],
):
    print(f"Adding {len(specs_to_add)} packages")
    print(f"Removing {len(specs_to_remove)} packages")


@plugins.hookimpl
def conda_pre_solves():
    yield plugins.types.CondaPreSolve(
        name="example-pre-solve",
        action=example_pre_solve,
    )
```

conda_post_solves() → collections.abc.Iterable[*conda.plugins.types.CondaPostSolve*]

Register post-solve functions in conda that are used in the general solver API, after the solver has provided the package records to add or remove from the conda environment.

Example:

```
from conda import plugins
from conda.models.records import PackageRecord


def example_post_solve(
    repodata_fn: str,
    unlink_precs: tuple[PackageRecord, ...],
    link_precs: tuple[PackageRecord, ...],
):
    print(f"Uninstalling {len(unlink_precs)} packages")
    print(f"Installing {len(link_precs)} packages")


@plugins.hookimpl
def conda_post_solves():
    yield plugins.types.CondaPostSolve(
        name="example-post-solve",
        action=example_post_solve,
    )
```

conda_settings() → collections.abc.Iterable[*conda.plugins.types.CondaSetting*]

Register new setting

The example below defines a simple string type parameter

Example:

```
from conda import plugins
from conda.common.configuration import PrimitiveParameter, SequenceParameter


@plugins.hookimpl
def conda_settings():
    yield plugins.types.CondaSetting(
        name="example_option",
        description="This is an example option",
        parameter=PrimitiveParameter("default_value", element_type=str),
        aliases=("example_option_alias",),
    )
```

conda_reporter_backends() → collections.abc.Iterable[*conda.plugins.types.CondaReporterBackend*]

Register new reporter backend

The example below defines a reporter backend that uses the `pprint` module in Python.

Example:

```
from pprint import pformat

from conda import plugins
from conda.plugins.types import (
    CondaReporterBackend,
    ReporterRendererBase,
    ProgressBarBase,
)


class PprintReporterRenderer(ReporterRendererBase):
    "Implementation of the ReporterRendererBase"

    def detail_view(self, data):
        return pformat(data)

    def envs_list(self, data):
        formatted_data = pformat(data)
        return f"Environments: {formatted_data}"

    def progress_bar(self, description, io_context_manager) -> ProgressBarBase:
        "Returns our custom progress bar implementation"
        return PprintProgressBar(description, io_context_manager)


class PprintProgressBar(ProgressBarBase):
    "Blank implementation of ProgressBarBase which does nothing"

    def update_to(self, fraction) -> None:
        pass

    def refresh(self) -> None:
        pass
```

(continues on next page)

(continued from previous page)

```
    def close(self) -> None:
        pass


@plugins.hookimpl
def conda_reporter_backends():
    yield CondaReporterBackend(
        name="pprint",
        description="Reporter backend based on the pprint module",
        renderer=PprintReporterRenderer,
    )
```

conda_session_headers(*host: str*) → collections.abc.Iterable[*conda.plugins.types.CondaRequestHeader*]

Register new HTTP request headers

The example below defines how to add HTTP headers for all requests with the hostname of `example.com`.

Example:

```
from conda import plugins

HOSTS = {"example.com", "sub.example.com"}


@plugins.hookimpl
def conda_session_headers(host: str):
    if host in HOSTS:
        yield plugins.types.CondaRequestHeader(
            name="Example-Header",
            value="example",
        )
```

conda_request_headers(*host: str*, *path: str*) → collections.abc.Iterable[*conda.plugins.types.CondaRequestHeader*]

Register new HTTP request headers

The example below defines how to add HTTP headers for all requests with the hostname of `example.com` and a `path/to/endpoint.json` path.

Example:

```
from conda import plugins

HOSTS = {"example.com", "sub.example.com"}
ENDPOINT = "/path/to/endpoint.json"


@plugins.hookimpl
def conda_request_headers(host: str, path: str):
    if host in HOSTS and path == ENDPOINT:
        yield plugins.types.CondaRequestHeader(
            name="Example-Header",
            value="example",
        )
```

conda_error_hints(*error:* conda.CondaError) → collections.abc.Iterable[*conda.plugins.types.CondaErrorHint*]

Register user-facing hints for expected conda errors.

This hook is invoked while rendering a *conda.CondaError*. Plugins receive the current error and yield structured *CondaErrorHint* objects that conda appends to the error's existing guidance. Plugins should not print directly from this hook.

When a *CondaMultiError* is printed, conda invokes this hook once per nested leaf error (for example `RemoveError`), not on the multi-error container itself. Implementations should match concrete exception types; `isinstance(error, CondaMultiError)` will not see wrapped failures.

Conda invokes implementations in deterministic plugin-name order and preserves the hint order yielded by each plugin. Existing core guidance hints win when a plugin yields the same `hint_code`. Hook wrappers are skipped so conda can isolate failures per implementation.

Use this hook for user-facing next steps. Use *conda_exception_observers()* for telemetry, logging, demand tracking, or other side effects.

Example:

```
from conda import plugins
from conda.exceptions import PackagesNotFoundInChannelsError


@plugins.hookimpl
def conda_error_hints(error):
    if isinstance(error, PackagesNotFoundInChannelsError):
        yield plugins.types.CondaErrorHint(
            text="Check whether the package exists on your expected channel.",
            hint_code="check_expected_channel",
        )
```

Returns

An iterable of *CondaErrorHint* entries.

conda_prefix_data_loaders() → collections.abc.Iterable[*conda.plugins.types.CondaPrefixDataLoader*]

Register new loaders for PrefixData

The example below defines how to expose the packages installed by a hypothetical 'penguin' tool as conda packages.

Example:

```
from pathlib import Path

from conda import plugins
from conda.common.path import PathType
from conda.models.records import PrefixRecord
from conda.plugins.types import CondaPrefixDataLoader


@plugins.hookimpl
def conda_prefix_data_loaders():
    yield CondaPrefixDataLoader(
        "hypothetical",
```

(continues on next page)

(continued from previous page)

```
        load_hypothetical_packages,
    )


def load_hypothetical_packages(
    prefix: PathType, records: dict[str, PrefixRecord]
) -> dict[str, PrefixRecord]:
    penguin_records = {}
    for info in Path(prefix).glob("lib/penguin/*.penguin-info"):
        name, version = info.name.rsplit("-", 1)
        kwargs = {}  # retrieve extra fields here
        penguin_records[name] = PrefixRecord(
            name=name, version=version, build_number=0, build="0", **kwargs
        )
    records.update(penguin_records)
    return penguin_records
```

conda_environment_specifiers() → collections.abc.Iterable[*conda.plugins.types.CondaEnvironmentSpecifier*]

EXPERIMENTAL Register new conda env spec type

The example below defines a type of conda env file called "random". It can parse a file with the file extension *.random*. This plugin will ignore whatever is in the input environment file and produce an environment with a random name and with random packages.

Example:

```
import json
import random
from pathlib import Path
from subprocess import run
from conda import plugins
from conda.plugins.types import EnvironmentSpecBase
from conda.base.context import context
from conda.models.environment import Environment
from conda.models.match_spec import MatchSpec

packages = ["python", "numpy", "scipy", "matplotlib", "pandas", "scikit-learn"]


class RandomSpec(EnvironmentSpecBase):
    extensions = {".random"}

    def __init__(self, filename: str):
        self.filename = filename

    def can_handle(self):
        # Return early if no filename was provided
        if self.filename is None:
            raise FileNotFoundError

        # The can_handle method should check the contents of the file and
        # not be reliant on file name or extension.  This allows a user to
```

(continues on next page)

(continued from previous page)

```
        # select a plugin by name, but also allows a plugin to handle a file
        # with a different extension.

        # Some quick checks to ensure the file is in the proper format here.
        # NOTE: It should be good enough to determine if the file is the proper
        # format in order to not be greedy and cause issues with other plugins.
        # If more than one plugin can handle the file, the user will need to go
        # through an additional step to select the proper plugin.

    @property
    def env(self):
        return Environment(
            platform=context.subdir,
            name="".join(random.choice("0123456789abcdef") for i in range(6)),
            requested_packages=[MatchSpec(random.choice(packages)) for i in␣
↪range(6)],
        )


@plugins.hookimpl
def conda_environment_specifiers():
    yield plugins.CondaEnvSpec(
        name="random",
        environment_spec=RandomSpec,
    )
```

conda_environment_exporters() → collections.abc.Iterable[*conda.plugins.types.CondaEnvironmentExporter*]

Register new conda environment exporter

Environment exporters serialize conda Environment objects to different output formats (JSON, TOML, XML, etc.) for the 'conda export' command. This is separate from environment specifiers which parse input files.

Example:

```
import tomlkit
from conda import plugins
from conda.exceptions import CondaValueError
from conda.models.environment import Environment
from conda.plugins.types import CondaEnvironmentExporter


def export_toml(env: Environment) -> str:
    # Export Environment to TOML format
    # For formats that use the standard dictionary structure,
    # you can use the shared utility:
    from conda.plugins.environment_exporters.standard import to_dict

    env_dict = to_dict(env)

    # Create TOML document
    toml_doc = tomlkit.document()
```

(continues on next page)

(continued from previous page)

```
    if env_dict.get("name"):
        toml_doc["name"] = env_dict["name"]

    if env_dict.get("channels"):
        toml_doc["channels"] = env_dict["channels"]

    if env_dict.get("dependencies"):
        toml_doc["dependencies"] = env_dict["dependencies"]

    if env_dict.get("variables"):
        toml_doc["variables"] = env_dict["variables"]

    return tomlkit.dumps(toml_doc)


@plugins.hookimpl
def conda_environment_exporters():
    yield CondaEnvironmentExporter(
        name="environment-toml",
        aliases=("toml",),
        default_filenames=("environment.toml",),
        export=export_toml,
    )
```

conda_package_extractors() → collections.abc.Iterable[*conda.plugins.types.CondaPackageExtractor*]

Register package extractors for different archive formats.

Package extractors handle the unpacking of package archives. Each extractor specifies which file extensions it supports (e.g., `.conda`, `.tar.bz2`) and provides an extraction function that takes the source archive path and destination directory.

Example:

```
from conda import plugins
from conda.common.path import PathType


def extract_custom(source_path: PathType, destination_directory: PathType) ->␣
↪None:
    # Custom extraction logic for a hypothetical package format
    ...


@plugins.hookimpl
def conda_package_extractors():
    yield plugins.types.CondaPackageExtractor(
        name="custom-package",
        extensions=[".custom"],
        extract=extract_custom,
    )
```

Returns

An iterable of *CondaPackageExtractor* entries.

conda_exception_observers() → collections.abc.Iterable[*conda.plugins.types.CondaExceptionObserver*]

Register exception observer callbacks in conda.

Exception observers are invoked when any exception is handled by the `ExceptionHandler`. They are **purely observational**: they cannot suppress, modify, or redirect the exception. Their return value is ignored. This follows the same model as CPython's `sys.excepthook`.

Any exception raised by an observer is caught at the `BaseException` level, logged at DEBUG, and swallowed -- a buggy plugin can never disrupt conda's error reporting path.

Observers run synchronously on the error path. They **must return promptly**. Network I/O, large file operations, or any potentially blocking call should be deferred to a daemon thread or subprocess.

Observers receive a frozen *CondaExceptionEvent* dataclass. The exception triple (`exc_type`, `exc_value`, `exc_traceback`) is always populated. Conda runtime fields (`argv`, `conda_version`, `return_code`, `active_prefix`, `target_prefix`, `channels`, `subdir`, `offline`, `dry_run`, `quiet`, `json`) are `None` when the runtime isn't initialized.

`watch_for` controls which exceptions trigger the observer via MRO matching: `{"CondaError"}` catches all conda errors, `{"BaseException"}` catches everything, `{"MemoryError"}` catches only OOM, etc. See *CondaExceptionObserver*.

Example:

```
import logging

from conda import plugins

log = logging.getLogger(__name__)


def report_missing(event):
    log.info("Missing packages: %s", event.exc_value.packages)
    log.info("Command was: %s", " ".join(event.argv))


@plugins.hookimpl
def conda_exception_observers():
    yield plugins.types.CondaExceptionObserver(
        name="missing-package-reporter",
        hook=report_missing,
        watch_for={"PackagesNotFoundInChannelsError"},
    )
```

Returns

An iterable of *CondaExceptionObserver* entries.

manager

This module contains a subclass implementation of pluggy's PluginManager.

Additionally, it contains a function we use to construct the `PluginManager` object and register all plugins during conda's startup process.

Classes

_HookImplWrapper	
CondaPluginManager	The conda plugin manager to implement behavior additional to pluggy's default plugin manager.

Functions

get_plugin_manager() → CondaPluginManager	Get a cached version of the *CondaPluginManager* instance,

Attributes

P
BUILTIN_PLUGIN_PREFIX

P

BUILTIN_PLUGIN_PREFIX = 'conda.plugins.'

class _HookImplWrapper

impl: pluggy.HookImpl

function(**args*)

__getattr__(*name*)

static _isiterable(*obj*)

static _set_impl(*result*, *impl*)

class CondaPluginManager(**args*, ***kwargs*)

Bases: pluggy.PluginManager

The conda plugin manager to implement behavior additional to pluggy's default plugin manager.

get_cached_solver_backend: collections.abc.Callable[[str | None], *type*[*conda.core.solve.Solver*]]

Cached version of the *get_solver_backend()* method.

get_cached_session_headers: collections.abc.Callable[[str], dict[str, str]]

Cached version of the *get_session_headers()* method.

get_cached_request_headers: collections.abc.Callable[[str, str], dict[str, str]]

Cached version of the *get_request_headers()* method.

plugin_aliases: dict[str, set[str]]

_distribution_aliases: dict[str, set[str]]

get_canonical_name(*plugin: object*) → str

Return a canonical name for a plugin object.

Note that a plugin may be registered under a different name specified by the caller of *register(plugin, name)*. To obtain the name of a registered plugin use `get_name(plugin)` instead.

get_plugin_source(*conda_plugin:* conda.plugins.types.CondaPlugin) → str | None

Return a human-readable source for a registered plugin, or `None` if `plugin` is `None`.

get_plugin_hook_names(*plugin: object*) → list[str]

Return short hook names implemented by a plugin.

is_conda_plugin_package(*package:* conda.models.package_info.PackageInfo | conda.models.records.PrefixRecord, *prefix: conda.common.path.PathType | None = None*) → bool

Return whether package contents declare conda plugin entry points.

get_installed_plugins() → list[PluginInfo]

Return metadata for installed entry-point conda plugins.

get_installed_plugin_info(*name: str*) → PluginDetails

Return detailed metadata for an installed entry-point conda plugin.

register(*plugin*, *name: str | None = None*) → str | None

Call `pluggy.PluginManager.register()` and return the result or ignore errors raised, except `ValueError`, which means the plugin had already been registered.

load_plugins(**plugins*) → int

Load the provided list of plugins and fail gracefully on error. The provided list of plugins can either be classes or modules with *hookimpl*.

load_entrypoints(*group: str*, *name: str | None = None*) → int

Load modules from querying the specified setuptools `group`.

Parameters

- **group** -- Entry point group to load plugins.
- **name** -- If given, loads only plugins with the given `name`.

Returns

The number of plugins loaded by this call.

_hookexec(*hook_name: str*, *methods: collections.abc.Sequence[pluggy.HookImpl]*, *kwargs: collections.abc.Mapping[str, object]*, *firstresult: bool*) → object | list[object]

get_hook_results(*name: Literal['subcommands']*) → list[*conda.plugins.types.CondaSubcommand*]

get_hook_results(*name: Literal['virtual_packages']*) → list[*conda.plugins.types.CondaVirtualPackage*]

get_hook_results(*name: Literal['solvers']*) → list[*conda.plugins.types.CondaSolver*]

get_hook_results(*name: Literal['pre_commands']*) → list[*conda.plugins.types.CondaPreCommand*]

get_hook_results(*name: Literal['post_commands']*) → list[*conda.plugins.types.CondaPostCommand*]

get_hook_results(*name: Literal['auth_handlers']*) → list[*conda.plugins.types.CondaAuthHandler*]

get_hook_results(*name: Literal['health_checks']*) → list[*conda.plugins.types.CondaHealthCheck*]

get_hook_results(*name: Literal['pre_solves']*) → list[*conda.plugins.types.CondaPreSolve*]

get_hook_results(*name: Literal['post_solves']*) → list[*conda.plugins.types.CondaPostSolve*]

get_hook_results(*name: Literal['session_headers']*, ***, *host: str*) → list[*conda.plugins.types.CondaRequestHeader*]

get_hook_results(*name: Literal['request_headers']*, ***, *host: str*, *path: str*) → list[*conda.plugins.types.CondaRequestHeader*]

get_hook_results(*name: Literal['settings']*) → list[*conda.plugins.types.CondaSetting*]

get_hook_results(*name: Literal['reporter_backends']*) → list[*conda.plugins.types.CondaReporterBackend*]

get_hook_results(*name: Literal['pre_transaction_actions']*) → list[*conda.plugins.types.CondaPreTransactionAction*]

get_hook_results(*name: Literal['post_transaction_actions']*) → list[*conda.plugins.types.CondaPostTransactionAction*]

get_hook_results(*name: Literal['prefix_data_loaders']*) → list[*conda.plugins.types.CondaPrefixDataLoader*]

get_hook_results(*name: Literal['environment_specifiers']*) → list[*conda.plugins.types.CondaEnvironmentSpecifier*]

get_hook_results(*name: Literal['package_extractors']*) → list[*conda.plugins.types.CondaPackageExtractor*]

get_hook_results(*name: Literal['environment_exporters']*) → list[*conda.plugins.types.CondaEnvironmentExporter*]

get_hook_results(*name: Literal['exception_observers']*) → list[*conda.plugins.types.CondaExceptionObserver*]

Return results of the plugin hooks with the given name and raise an error if there is a conflict.

get_solvers() → dict[str, *conda.plugins.types.CondaSolver*]

Return a mapping from solver name to solver class.

get_solver_backend(*name: str | None = None*) → *type*[*conda.core.solve.Solver*]

Get the solver backend with the given name (or fall back to the name provided in the context).

See `context.solver` for more details.

Please use the cached version of this method called *`get_cached_solver_backend()`* for high-throughput code paths which is set up as a instance-specific LRU cache.

get_auth_handler(*name: str*) → *type*[requests.auth.AuthBase] | None

Get the auth handler with the given name or None

get_settings() → dict[str, *conda.plugins.types.CondaSetting*]

Return a mapping of plugin setting name to CondaSetting objects.

This method intentionally overwrites any duplicates that may be present

invoke_pre_commands(*command: str*) → None

Invokes `CondaPreCommand.action` functions registered with `conda_pre_commands`.

Parameters
command -- name of the command that is currently being invoked

invoke_post_commands(*command: str*) → None

Invokes `CondaPostCommand.action` functions registered with `conda_post_commands`.

Parameters
command -- name of the command that is currently being invoked

invoke_exception_observers(*exc_val: BaseException*, *exc_tb: types.TracebackType*) → None

Invokes `CondaExceptionObserver.hook` functions registered with `conda_exception_observers`.

Observers are purely observational (following CPython's `sys.excepthook` model) and cannot suppress, modify, or redirect the exception. Return values are ignored.

Any exception raised by an observer is caught at the `BaseException` level and logged -- a buggy plugin raising `SystemExit` or `KeyboardInterrupt` cannot kill conda's error reporting.

Parameters

- **exc_val** -- the exception instance being handled
- **exc_tb** -- the associated traceback

disable_external_plugins(*, *except_plugins: collections.abc.Iterable[str] = ()*) → None

Disable external plugins except those explicitly selected.

disable_plugins(*names: collections.abc.Iterable[str]*, *, *except_plugins: collections.abc.Iterable[str] = ()*) → None

Disable specific plugins by name.

Accepts canonical names (e.g. `conda_self.plugin`), distribution names (e.g. `conda-self`), or entry point names. Raises `PluginError` for built-in plugins and logs a warning for unrecognized names. Explicit exceptions take precedence and must match registered plugins.

_resolve_plugin_names(*target: str*) → set[str]

get_subcommands() → dict[str, *conda.plugins.types.CondaSubcommand*]

get_health_checks() → dict[str, *conda.plugins.types.CondaHealthCheck*]

Return a mapping of health check name to health check.

get_reporter_backends() → tuple[*conda.plugins.types.CondaReporterBackend*, ...]

get_reporter_backend(*name: str*) → *conda.plugins.types.CondaReporterBackend*

Attempts to find a reporter backend while providing a fallback option if it is not found.

This method must return a valid `CondaReporterBackend` object or else it will raise an exception.

get_virtual_package_records() → tuple[*conda.models.records.PackageRecord*, ...]

get_session_headers(*host: str*) → dict[str, str]

get_request_headers(*host: str*, *path: str*) → dict[str, str]

get_error_hints(*error:* conda.CondaError) → tuple[*conda._private.exception_guidance.GuidanceHint*, ...]

Return plugin-provided guidance hints for a conda error.

Each hook implementation is invoked independently so a faulty plugin cannot prevent other plugins from contributing hints during error rendering. Implementations are invoked in deterministic plugin-name order, while preserving the hint order yielded by each plugin.

get_prefix_data_loaders() → collections.abc.Iterable[*conda.plugins.types.CondaPrefixDataLoaderCallable*]

invoke_pre_solves(*specs_to_add: frozenset[*conda.models.match_spec.MatchSpec*]*, *specs_to_remove: frozenset[*conda.models.match_spec.MatchSpec*]*) → None

Invokes `CondaPreSolve.action` functions registered with `conda_pre_solves`.

Parameters

- **specs_to_add**

- **specs_to_remove**

invoke_post_solves(*repodata_fn: str*, *unlink_precs: tuple[*conda.models.records.PackageRecord*, ...]*, *link_precs: tuple[*conda.models.records.PackageRecord*, ...]*) → None

Invokes `CondaPostSolve.action` functions registered with `conda_post_solves`.

Parameters

- **repodata_fn**
- **unlink_precs**
- **link_precs**

load_settings() → None

Iterates through all registered settings and adds them to the `conda.common.configuration.PluginConfig` class.

get_config(*data*) → *conda.plugins.config.PluginConfig*

Retrieve the configuration for the plugin.

Returns
The configuration object for the plugin, initialized with raw data from the context.

Return type
PluginConfig

get_environment_specifiers(*, *supports_detection: bool | None = None*, *with_aliases: bool = True*) → dict[str, *conda.plugins.types.CondaEnvironmentSpecifier*]

Returns a mapping from environment specifier name to environment specifier.

Parameters

- **supports_detection** -- ternary value that returns either everything, only supporting detection or not supporting detection.
- **with_aliases** -- whether to include aliased values of environment specifiers.

_get_name_and_alias_mapping(*plugins: collections.abc.Iterable[*P*]*) → dict[str, *P*]

Get a mapping from plugin names (including aliases) to plugin.

Parameters
plugins -- Plugins that expose a `name` and `aliases`.

Returns
Mapping from each canonical name and alias to the corresponding plugin.

Raises
PluginError -- If multiple plugins use the same name or alias.

get_environment_specifier_by_name(*source: str*, *name: str*) → *conda.plugins.types.CondaEnvironmentSpecifier*

Get an environment specifier plugin by name

Parameters

- **source** -- full path to the environment spec file/source
- **name** -- name of the environment plugin to load

Returns
An environment specifier plugin that matches the provided plugin name, or can handle the provided file

Raises

- ***CondaValueError*** -- if the requested plugin is not available.
- ***PluginError*** -- if the requested plugin is unable to handle the provided file.

_detect_filename_env_spec(*source: str*, *basename: str*, *hooks: dict[str, conda.plugins.types.CondaEnvironmentSpecifier]*) → list[*conda.plugins.types.CondaEnvironmentSpecifier*]

Detect environment specifier by filename pattern matching.

Parameters

- **basename** -- basename of the source file
- **hooks** -- mapping of environment specifier plugins

Returns
List of matching plugins, or None if no filename matches

static _detect_content_env_spec(*source: str*, *hooks: dict[str, conda.plugins.types.CondaEnvironmentSpecifier]*) → list[*conda.plugins.types.CondaEnvironmentSpecifier*]

Detect environment specifier by content-based autodetection.

Parameters

- **source** -- full path to the environment spec file or source
- **hooks** -- mapping of environment specifier plugins

Returns
Tuple of (found plugins, autodetect disabled plugin names)

detect_environment_specifier(*source: str*) → *conda.plugins.types.CondaEnvironmentSpecifier*

Detect the environment specifier plugin for a given spec source

Uses two-phase detection: 1. Filename-based filtering using fnmatch patterns 2. Fallback to content-based autodetection (can_handle())

Raises PluginError if more than one environment_spec plugin is found to be able to handle the file. Raises EnvironmentSpecPluginNotDetected if no plugins were found.

Parameters
source -- full path to the environment spec file or source

Returns
An environment specifier plugin that can handle the provided file

get_environment_specifier(*source: str*, *name: str | None = None*) → *conda.plugins.types.CondaEnvironmentSpecifier*

Get the environment specifier plugin for a given spec source, or given a plugin name. Raises PluginError if more than one environment_spec plugin is found to be able to handle the file. Raises EnvironmentSpecPluginNotDetected if no plugins were found. Raises CondaValueError if the requested plugin is not available.

Parameters

- **source** -- full path to the environment spec file/source
- **name** -- name of the environment plugin to load

Returns
An environment specifier plugin that matches the provided plugin name, or can handle the provided file

get_environment_specifiers_grouped() → dict[*conda.plugins.types.EnvironmentFormat*, list[*conda.plugins.types.CondaEnvironmentSpecifier*]]

Group environment specifiers by *EnvironmentFormat*.

get_environment_exporters() → collections.abc.Iterable[*conda.plugins.types.CondaEnvironmentExporter*]

Yields all detected environment exporters.

get_exporter_format_mapping() → dict[str, *conda.plugins.types.CondaEnvironmentExporter*]

Get a mapping from format names (including aliases) to environment exporters.

Returns
Dict mapping format name to CondaEnvironmentExporter

Raises
PluginError -- If multiple exporters use the same format name or alias

detect_environment_exporter(*filename: str*) → *conda.plugins.types.CondaEnvironmentExporter*

Detect an environment exporter based on filename matching against default_filenames.

Uses fnmatch pattern matching for flexible filename patterns (e.g., *.conda-lock.yml).

Parameters
filename -- Filename to find an exporter for (basename is used for detection)

Returns
CondaEnvironmentExporter that supports the filename

Raises

- *EnvironmentExporterNotDetected* -- If no exporter supports the filename
- *PluginError* -- If multiple exporters claim to support the same filename

get_environment_exporter_by_format(*format_name: str*) → *conda.plugins.types.CondaEnvironmentExporter*

Get an environment exporter based on the format name.

Parameters
format_name -- Format name to find an exporter for (e.g., 'yaml', 'json', 'environment-yaml')

Returns
CondaEnvironmentExporter that supports the format

Raises
CondaValueError -- If no exporter is found for the given format

get_environment_exporters_grouped() → dict[*conda.plugins.types.EnvironmentFormat*, list[*conda.plugins.types.CondaEnvironmentExporter*]]

Group environment exporters by *EnvironmentFormat*.

get_pre_transaction_actions(*transaction_context: dict[str, str] | None = None*, *target_prefix: str | None = None*, *unlink_precs: collections.abc.Iterable[conda.models.records.PackageRecord] | None = None*, *link_precs: collections.abc.Iterable[conda.models.records.PackageRecord] | None = None*, *remove_specs: collections.abc.Iterable[conda.models.match_spec.MatchSpec] | None = None*, *update_specs: collections.abc.Iterable[conda.models.match_spec.MatchSpec] | None = None*, *neutered_specs: collections.abc.Iterable[conda.models.match_spec.MatchSpec] | None = None*) → list[*conda.core.path_actions.Action*]

Get the plugin-defined pre-transaction actions.

Parameters

- **transaction_context** -- Mapping between target prefixes and PrefixActionGroup instances
- **target_prefix** -- Target prefix for the action
- **unlink_precs** -- Package records to be unlinked
- **link_precs** -- Package records to link
- **remove_specs** -- Specs to be removed
- **update_specs** -- Specs to be updated
- **neutered_specs** -- Specs to be neutered

Returns

The plugin-defined pre-transaction actions

get_post_transaction_actions(*transaction_context: dict[str, str] | None = None*, *target_prefix: str | None = None*, *unlink_precs: collections.abc.Iterable[conda.models.records.PackageRecord] | None = None*, *link_precs: collections.abc.Iterable[conda.models.records.PackageRecord] | None = None*, *remove_specs: collections.abc.Iterable[conda.models.match_spec.MatchSpec] | None = None*, *update_specs: collections.abc.Iterable[conda.models.match_spec.MatchSpec] | None = None*, *neutered_specs: collections.abc.Iterable[conda.models.match_spec.MatchSpec] | None = None*) → list[*conda.core.path_actions.Action*]

Get the plugin-defined post-transaction actions.

Parameters

- **transaction_context** -- Mapping between target prefixes and PrefixActionGroup instances
- **target_prefix** -- Target prefix for the action
- **unlink_precs** -- Package records to be unlinked
- **link_precs** -- Package records to link
- **remove_specs** -- Specs to be removed
- **update_specs** -- Specs to be updated

- **neutered_specs** -- Specs to be neutered

Returns
The plugin-defined post-transaction actions

get_package_extractors() → dict[str, *conda.plugins.types.CondaPackageExtractor*]

Return a mapping from file extension to package extractor plugin.

Extensions are lowercased for case-insensitive matching.

Returns
Dictionary mapping lowercased extensions (e.g., ".conda") to their *CondaPackageExtractor* plugins.

get_package_extractor(*source_full_path: conda.common.path.PathType*) → *conda.plugins.types.CondaPackageExtractor*

Get the package extractor plugin for a given package path.

Searches through registered package extractor plugins to find one that handles the file extension of the provided package path.

Parameters
source_full_path -- Full path to the package archive file.

Returns
The matching *CondaPackageExtractor* plugin.

Raises
PluginError -- If no registered extractor handles the file extension.

extract_package(*source_full_path: conda.common.path.PathType*, *destination_directory: conda.common.path.PathType*) → None

Extract a package archive to a destination directory.

Finds the appropriate extractor plugin based on the file extension and extracts the package.

Parameters

- **source_full_path** -- Full path to the package archive file.
- **destination_directory** -- Directory to extract the package contents to.

Raises
PluginError -- If no registered extractor handles the file extension.

has_package_extension(*path: conda.common.path.PathType*) → str | None

Check if a path has a supported package file extension.

Parameters
path -- Path to check.

Returns
The matched extension (lowercased) if found, None otherwise.

get_plugin_manager() → *CondaPluginManager*

Get a cached version of the *CondaPluginManager* instance, with the built-in and entrypoints provided by the plugins loaded.

package_extractors

Package extractor plugins.

conda

Package extractor plugin for .conda and .tar.bz2 formats.

Functions

extract_conda_or_tarball(tarball_full_path, ...) → None	Extract a .conda or .tar.bz2 package to the specified destination.
conda_package_extractors()	

extract_conda_or_tarball(*tarball_full_path: conda.common.path.PathType*, *destination_directory: conda.common.path.PathType*) → None

Extract a .conda or .tar.bz2 package to the specified destination.

Parameters

- **tarball_full_path** -- Path to the package archive.
- **destination_directory** -- Directory to extract the package contents to.

conda_package_extractors()

plugins

post_solves

Register the built-in post_solves hook implementations.

plugins = []

The list of post-solve plugins for easier registration with pluggy.

prefix_data_loaders

pypi

Reads PyPI packages in a conda prefix that have been installed with non-conda tools.

pkg_format

Common Python package format utilities.

Classes

PySpec	
PythonDistribution	Base object describing a python distribution based on path to anchor file.
PythonInstalledDistribution	Python distribution installed via distutils.
PythonEggInfoDistribution	Python distribution installed via setuptools.
PythonEggLinkDistribution	Python distribution installed via setuptools.

continues on next page

Table 282 – continued from previous page

PythonDistributionMetadata	Object representing the metada of a Python Distribution given by anchor
Evaluator	This class is used to evaluate marker expressions.

Functions

norm_package_name(name)	
pypi_name_to_conda_name(pypi_name)	
norm_package_version(version)	Normalize a version by removing extra spaces and parentheses.
split_spec(spec, sep)	Split a spec by separator and return stripped start and end parts.
parse_specification(spec)	Parse a requirement from a python distribution metadata and return a
get_site_packages_anchor_files(site_packages_pa ...)	Get all the anchor files for the site packages directory.
get_dist_file_from_egg_link(egg_link_file, prefix_path)	Return the egg info file path following an egg link.
parse_marker(marker_string)	Parse marker string and return a dictionary containing a marker expression.
_is_literal(o)	
get_default_marker_context()	Return the default context dictionary to use when parsing markers.
interpret(marker[, execution_context])	Interpret a marker and return a result depending on environment.
read_python_record(prefix_path, anchor_file, ...)	Convert a python package defined by an anchor file (Metadata information)

Attributes

PYPI_TO_CONDA
PYPI_CONDA_DEPS
PARTIAL_PYPI_SPEC_PATTERN
PY_FILE_RE
IDENTIFIER
VERSION_IDENTIFIER
COMPARE_OP
MARKER_OP
OR
AND
NON_SPACE
STRING_CHUNK
DEFAULT_MARKER_CONTEXT
evaluator

PYPI_TO_CONDA

PYPI_CONDA_DEPS

PARTIAL_PYPI_SPEC_PATTERN

PY_FILE_RE

class PySpec

Bases: tuple

name

extras

constraints

marker

url

exception MetadataWarning

Bases: Warning

Base class for warning categories.

Initialize self. See help(type(self)) for accurate signature.

class PythonDistribution(*anchor_full_path*, *python_version*)

Base object describing a python distribution based on path to anchor file.

MANIFEST_FILES = ()

REQUIRES_FILES = ()

MANDATORY_FILES = ()

ENTRY_POINTS_FILES = ('entry_points.txt',)

static init(*prefix_path*, *anchor_file*, *python_version*)

anchor_full_path

python_version

_metadata

_provides_file_data = ()

_requires_file_data = ()

_check_files()

Check the existence of mandatory files for a given distribution.

_check_path_data(*path*, *checksum*, *size*)

Normalizes record data content and format.

static _parse_requires_file_data(*data*, *global_section='__global__'*)

static _parse_entries_file_data(*data*)

_load_requires_provides_file()

manifest_full_path()

get_paths()

Read the list of installed paths from record or source file.

Example

[(u'skdata/__init__.py', u'sha256=47DEQpj8HBSa-_TImW-5JCeuQeRkm5NMpJWZG3hSuFU', 0),
(u'skdata/diabetes.py', None, None), ...

]

get_dist_requirements()

get_python_requirements()

get_external_requirements()

get_extra_provides()

get_conda_dependencies()

Process metadata fields providing dependency information.

This includes normalizing fields, and evaluating environment markers.

abstractmethod get_optional_dependencies()

get_entry_points()

property name

property norm_name

property conda_name

property version

class PythonInstalledDistribution(*prefix_path*, *anchor_file*, *python_version*)

Bases: *PythonDistribution*

Python distribution installed via distutils.

Notes

- https://www.python.org/dev/peps/pep-0376/

MANIFEST_FILES = ('RECORD',)

REQUIRES_FILES = ()

MANDATORY_FILES = ('METADATA',)

ENTRY_POINTS_FILES = ()

is_manageable = True

sp_reference

class PythonEggInfoDistribution(*anchor_full_path*, *python_version*, *sp_reference*)

Bases: *PythonDistribution*

Python distribution installed via setuptools.

Notes

- http://peak.telecommunity.com/DevCenter/EggFormats

MANIFEST_FILES = ('installed-files.txt', 'SOURCES', 'SOURCES.txt')

REQUIRES_FILES = ('requires.txt', 'depends.txt')

MANDATORY_FILES = ()

ENTRY_POINTS_FILES = ('entry_points.txt',)

sp_reference

property is_manageable

class PythonEggLinkDistribution(*prefix_path*, *anchor_file*, *python_version*)

Bases: *PythonEggInfoDistribution*

Python distribution installed via setuptools.

Notes

- http://peak.telecommunity.com/DevCenter/EggFormats

is_manageable = False

class PythonDistributionMetadata(*path*)

Object representing the metada of a Python Distribution given by anchor file (or directory) path.

This metadata is extracted from a single file. Python distributions might create additional files that complement this metadata information, but that is handled at the python distribution level.

Notes

- https://packaging.python.org/specifications/core-metadata/
- Metadata 2.1: https://www.python.org/dev/peps/pep-0566/
- Metadata 2.0: https://www.python.org/dev/peps/pep-0426/ (Withdrawn)
- Metadata 1.2: https://www.python.org/dev/peps/pep-0345/
- Metadata 1.1: https://www.python.org/dev/peps/pep-0314/
- Metadata 1.0: https://www.python.org/dev/peps/pep-0241/

FILE_NAMES = ('METADATA', 'PKG-INFO')

SINGLE_USE_KEYS

MULTIPLE_USE_KEYS

_path

_data

static _process_path(*path*, *metadata_filenames*)

Find metadata file inside dist-info folder, or check direct file.

`classmethod _message_to_dict(message)`

Convert the RFC-822 headers data into a dictionary.

message is an email.parser.Message instance.

The canonical method to transform metadata fields into such a data structure is as follows:

- The original key-value format should be read with email.parser.HeaderParser
- All transformed keys should be reduced to lower case. Hyphens should be replaced with underscores, but otherwise should retain all other characters
- The transformed value for any field marked with "(Multiple-use") should be a single list containing all the original values for the given key
- The Keywords field should be converted to a list by splitting the original value on whitespace characters
- The message body, if present, should be set to the value of the description key.
- The result should be stored as a string-keyed dictionary.

`classmethod _read_metadata(fpath)`

Read the original format which is stored as RFC-822 headers.

`_get_multiple_data(keys)`

Helper method to get multiple data values by keys.

Keys is an iterable including the preferred key in order, to include values of key that might have been replaced (deprecated), for example keys can be ['requires_dist', 'requires'], where the key 'requires' is deprecated and replaced by 'requires_dist'.

`get_dist_requirements()`

Changed in version 2.1: The field format specification was relaxed to accept the syntax used by popular publishing tools.

Each entry contains a string naming some other distutils project required by this distribution.

The format of a requirement string contains from one to four parts:

- A project name, in the same format as the Name: field. The only mandatory part.
- A comma-separated list of 'extra' names. These are defined by the required project, referring to specific features which may need extra dependencies.
- A version specifier. Tools parsing the format should accept optional parentheses around this, but tools generating it should not use parentheses.
- An environment marker after a semicolon. This means that the requirement is only needed in the specified conditions.

This field may be followed by an environment marker after a semicolon.

Example

frozenset(['pkginfo', 'PasteDeploy', 'zope.interface (>3.5.0)',
'pywin32 >1.0; sys_platform == "win32"'])

Return 'Requires' if 'Requires-Dist' is empty.

`get_python_requirements()`

New in version 1.2.

This field specifies the Python version(s) that the distribution is guaranteed to be compatible with. Installation tools may look at this when picking which version of a project to install.

The value must be in the format specified in Version specifiers.

This field may be followed by an environment marker after a semicolon.

Example

frozenset(['>=3', '>2.6,!=3.0.*,!=3.1.*', '~=2.6',
'>=3; sys_platform == "win32"'])

`get_external_requirements()`

Changed in version 2.1: The field format specification was relaxed to accept the syntax used by popular publishing tools.

Each entry contains a string describing some dependency in the system that the distribution is to be used. This field is intended to serve as a hint to downstream project maintainers, and has no semantics which are meaningful to the distutils distribution.

The format of a requirement string is a name of an external dependency, optionally followed by a version declaration within parentheses.

This field may be followed by an environment marker after a semicolon.

Because they refer to non-Python software releases, version numbers for this field are not required to conform to the format specified in PEP 440: they should correspond to the version scheme used by the external dependency.

Notice that there's is no particular rule on the strings to be used!

Example

frozenset(['C', 'libpng (>=1.5)', 'make; sys_platform != "win32"'])

`get_extra_provides()`

New in version 2.1.

A string containing the name of an optional feature. Must be a valid Python identifier. May be used to make a dependency conditional on hether the optional feature has been requested.

Example

frozenset(['pdf', 'doc', 'test'])

`get_dist_provides()`

New in version 1.2.

Changed in version 2.1: The field format specification was relaxed to accept the syntax used by popular publishing tools.

Each entry contains a string naming a Distutils project which is contained within this distribution. This field must include the project identified in the Name field, followed by the version : Name (Version).

A distribution may provide additional names, e.g. to indicate that multiple projects have been bundled together. For instance, source distributions of the ZODB project have historically included the transaction project, which is now available as a separate distribution. Installing such a source distribution satisfies requirements for both ZODB and transaction.

A distribution may also provide a "virtual" project name, which does not correspond to any separately-distributed project: such a name might be used to indicate an abstract capability which could be supplied by one of multiple projects. E.g., multiple projects might supply RDBMS bindings for use by a given ORM: each project might declare that it provides ORM-bindings, allowing other projects to depend only on having at most one of them installed.

A version declaration may be supplied and must follow the rules described in Version specifiers. The distribution's version number will be implied if none is specified.

This field may be followed by an environment marker after a semicolon.

Return *Provides* in case *Provides-Dist* is empty.

`get_dist_obsolete()`

New in version 1.2.

Changed in version 2.1: The field format specification was relaxed to accept the syntax used by popular publishing tools.

Each entry contains a string describing a distutils project's distribution which this distribution renders obsolete, meaning that the two projects should not be installed at the same time.

Version declarations can be supplied. Version numbers must be in the format specified in Version specifiers [1].

The most common use of this field will be in case a project name changes, e.g. Gorgon 2.3 gets subsumed into Torqued Python 1.0. When you install Torqued Python, the Gorgon distribution should be removed.

This field may be followed by an environment marker after a semicolon.

Return *Obsoletes* in case *Obsoletes-Dist* is empty.

Example

frozenset(['Gorgon', "OtherProject (<3.0) ; python_version == '2.7'"])

Notes

- [1] https://packaging.python.org/specifications/version-specifiers/

`get_classifiers()`

Classifiers are described in PEP 301, and the Python Package Index publishes a dynamic list of currently defined classifiers.

This field may be followed by an environment marker after a semicolon.

Example

frozenset(['Development Status :: 4 - Beta',
"Environment :: Console (Text Based) ; os_name == "posix"])

`property name`

`property version`

`norm_package_name(`*name*`)`

`pypi_name_to_conda_name(`*pypi_name*`)`

`norm_package_version(`*version*`)`

Normalize a version by removing extra spaces and parentheses.

`split_spec(`*spec*, *sep*`)`

Split a spec by separator and return stripped start and end parts.

parse_specification(*spec*)

Parse a requirement from a python distribution metadata and return a namedtuple with name, extras, constraints, marker and url components.

This method does not enforce strict specifications but extracts the information which is assumed to be *correct*. As such no errors are raised.

Example

PySpec(name='requests', extras=['security'], constraints='>=3.3.0',
marker='foo >= 2.7 or bar == 1', url='')

get_site_packages_anchor_files(*site_packages_path*, *site_packages_dir*)

Get all the anchor files for the site packages directory.

get_dist_file_from_egg_link(*egg_link_file*, *prefix_path*)

Return the egg info file path following an egg link.

parse_marker(*marker_string*)

Parse marker string and return a dictionary containing a marker expression.

The dictionary will contain keys "op", "lhs" and "rhs" for non-terminals in the expression grammar, or strings. A string contained in quotes is to be interpreted as a literal string, and a string not contained in quotes is a variable (such as os_name).

IDENTIFIER

VERSION_IDENTIFIER

COMPARE_OP

MARKER_OP

OR

AND

NON_SPACE

STRING_CHUNK

_is_literal(*o*)

class Evaluator

This class is used to evaluate marker expressions.

operations

evaluate(*expr*, *context*)

Evaluate a marker expression returned by the `parse_requirement()` function in the specified context.

get_default_marker_context()

Return the default context dictionary to use when parsing markers.

DEFAULT_MARKER_CONTEXT

evaluator

interpret(*marker*, *execution_context=None*)

Interpret a marker and return a result depending on environment.

Parameters

- **marker** -- The marker to interpret.
- **execution_context** -- The context used for name lookup.

read_python_record(*prefix_path*, *anchor_file*, *python_version*)

Convert a python package defined by an anchor file (Metadata information) into a conda prefix record object.

Classes

PrefixGraph	A directed graph structure used for sorting packages (prefix_records) in prefixes and
CondaPrefixDataLoader	Define new loaders to expose non-conda packages in a given prefix

Functions

get_python_site_packages_short_path(python_ve	
win_path_ok(path)	
get_conda_anchor_files_and_records(...) → dict[conda.common.path.PathType, conda.models.records.PrefixRecord]	Return the anchor files for the conda records of python packages.
rm_rf(path[, clean_empty_parents]) → bool	Completely delete path
get_site_packages_anchor_files(site_packages_pa ...)	Get all the anchor files for the site packages directory.
read_python_record(prefix_path, anchor_file, ...)	Convert a python package defined by an anchor file (Metadata information)
resolved_short_path(short_path, prefix_path) → str	Return short_path with any symlinks resolved.
load_site_packages(prefix, records) → dict[str, conda.models.records.PrefixRecord]	Load non-conda-installed python packages in the site-packages of the prefix.
conda_prefix_data_loaders()	

Attributes

hookimpl	Decorator to mark plugin hook implementations, used to register plugins.

exception ValidationError(*key*, *value=None*, *valid_types=None*, *msg=None*)

Bases: `AuxlibError`, `TypeError`

Mixin to identify exceptions associated with the auxlib package.

Initialize self. See help(type(self)) for accurate signature.

__cause__ = None

get_python_site_packages_short_path(*python_version*)

win_path_ok(*path*)

get_conda_anchor_files_and_records(*site_packages_short_path: conda.common.path.PathType*, *python_records: collections.abc.Iterable[*conda.models.records.PrefixRecord*]*) → dict[conda.common.path.PathType, *conda.models.records.PrefixRecord*]

Return the anchor files for the conda records of python packages.

rm_rf(*path: str | os.PathLike*, *clean_empty_parents: bool = False*) → bool

Completely delete path max_retries is the number of times to retry on failure. The default is 5. This only applies to deleting a directory. If removing path fails and trash is True, files will be moved to the trash directory.

class PrefixGraph(*records: collections.abc.Iterable[*conda.models.records.PrefixRecord*]*, *specs: collections.abc.Iterable[*conda.models.match_spec.MatchSpec*] = ()*)

A directed graph structure used for sorting packages (prefix_records) in prefixes and manipulating packages within prefixes (e.g. removing and pruning).

The terminology used for edge direction is "parents" and "children" rather than "successors" and "predecessors". The parent nodes of a record are those records in the graph that match the record's "depends" field. E.g. NodeA depends on NodeB, then NodeA is a child of NodeB, and NodeB is a parent of NodeA. Nodes can have zero parents, or more than two parents.

Most public methods mutate the graph.

graph: dict[*conda.models.records.PrefixRecord*, dict[*conda.models.records.PrefixRecord*, None]]

spec_matches: dict[*conda.models.records.PrefixRecord*, dict[*conda.models.match_spec.MatchSpec*, None]]

remove_spec(*spec*)

Remove all matching nodes, and any associated child nodes.

Parameters
spec (MatchSpec)

Returns
The removed nodes.

Return type
tuple[*PrefixRecord*]

remove_youngest_descendant_nodes_with_specs()

A specialized method used to determine only dependencies of requested specs.

Returns
The removed nodes.

Return type
tuple[*PrefixRecord*]

property records

prune()

Prune back all packages until all child nodes are anchored by a spec.

Returns
The pruned nodes.

Return type
tuple[*PrefixRecord*]

`get_node_by_name`(*name*)

`all_descendants`(*node*)

`all_ancestors`(*node*)

`_remove_node`(*node*)

Removes this node and all edges referencing it.

`_toposort`()

`classmethod _toposort_raise_on_cycles`(*graph*)

`classmethod _topo_sort_handle_cycles`(*graph*)

`static _toposort_pop_key`(*graph*)

Pop an item from the graph that has the fewest parents. In the case of a tie, use the node with the alphabetically-first package name.

`static _toposort_prepare_graph`(*graph*)

`hookimpl`

Decorator to mark plugin hook implementations, used to register plugins.

`class CondaPrefixDataLoader`

Bases: `CondaPlugin`

Define new loaders to expose non-conda packages in a given prefix as `PrefixRecord` objects.

Parameters

- **name** -- name of the loader
- **loader** -- a function that takes a prefix and a dictionary that maps package names to `PrefixRecord` objects. The newly loaded packages must be inserted in the passed dictionary accordingly, and also returned as a separate dictionary.

`name: str`

User-facing name of the plugin used for selecting & filtering plugins and error messages.

`loader: CondaPrefixDataLoaderCallable`

`get_site_packages_anchor_files`(*site_packages_path*, *site_packages_dir*)

Get all the anchor files for the site packages directory.

`read_python_record`(*prefix_path*, *anchor_file*, *python_version*)

Convert a python package defined by an anchor file (Metadata information) into a conda prefix record object.

`resolved_short_path`(*short_path: str*, *prefix_path: pathlib.Path*) → str

Return short_path with any symlinks resolved.

`load_site_packages`(*prefix: conda.common.path.PathType*, *records: dict[str, conda.models.records.PrefixRecord]*) → dict[str, *conda.models.records.PrefixRecord*]

Load non-conda-installed python packages in the site-packages of the prefix.

Python packages not handled by conda are installed via other means, like using pip or using python setup.py develop for local development.

Packages found that are not handled by conda are converted into a prefix record and handled in memory.

Packages clobbering conda packages (i.e. the conda-meta record) are removed from the in memory representation.

conda_prefix_data_loaders()

plugins

previews

Built-in plugin hooks for opt-in preview features.

Functions

conda_subcommands()	
is_preview_subcommand(plugin_subcommand) → bool	Return whether *plugin_subcommand* was yielded by `conda.plugins.previews`.

Attributes

PREVIEW_PLUGIN_NAME

PREVIEW_PLUGIN_NAME = 'conda.plugins.previews'

conda_subcommands()

is_preview_subcommand(*plugin_subcommand:* conda.plugins.types.CondaSubcommand) → bool

Return whether *plugin_subcommand* was yielded by `conda.plugins.previews`.

Checks `plugin_subcommand.impl.plugin_name` (hook provenance set by the plugin manager).

reporter_backends

console

Defines a "console" reporter backend.

This reporter backend provides the default output for conda.

Classes

QuietProgressBar	Progress bar class used when no output should be printed
TQDMProgressBar	Progress bar class used for tqdm progress bars
Spinner	Helper class that provides a standard way to create an ABC using
QuietSpinner	Helper class that provides a standard way to create an ABC using
ConsoleReporterRenderer	Default implementation for console reporting in conda

Functions

conda_reporter_backends()	Reporter backend for console

class QuietProgressBar(*description: str*, ***kwargs*)

Bases: conda.plugins.types.ProgressBarBase

Progress bar class used when no output should be printed

update_to(*fraction*) → None

refresh() → None

close() → None

class TQDMProgressBar(*description: str*, *position=None*, *leave=True*, ***kwargs*)

Bases: conda.plugins.types.ProgressBarBase

Progress bar class used for tqdm progress bars

enabled = True

update_to(*fraction*) → None

close() → None

refresh() → None

static _tqdm(**args*, ***kwargs*)

Deferred import so it doesn't hit the *conda activate* paths.

class Spinner(*message*, *fail_message='failed\n'*)

Bases: conda.plugins.types.SpinnerBase

Helper class that provides a standard way to create an ABC using inheritance.

spinner_cycle

show_spin: bool = True

_stop_running

_spinner_thread

_indicator_length

fh

start()

stop()

_start_spinning()

__enter__()

__exit__(*exc_type*, *exc_val*, *exc_tb*)

class QuietSpinner(*message: str*, *fail_message: str = 'failed\n'*)

Bases: *conda.plugins.types.SpinnerBase*

Helper class that provides a standard way to create an ABC using inheritance.

__enter__()

__exit__(*exc_type*, *exc_val*, *exc_tb*)

class ConsoleReporterRenderer

Bases: *conda.plugins.types.ReporterRendererBase*

Default implementation for console reporting in conda

render(*data: Any*, ***kwargs*) → str

detail_view(*data: dict[str, str | int | bool]*, ***kwargs*) → str

Render the output in a "tabular" format.

static envs_list(*prefixes: collections.abc.Iterable[conda.common.path.PathType | conda.core.prefix_data.PrefixData]*, ***kwargs*) → str

Render a list of environments

property animations_disabled: bool

True when progress bars/spinners should be suppressed.

progress_bar(*description: str*, ***kwargs*) → *conda.plugins.types.ProgressBarBase*

Determines whether to return a TQDMProgressBar or QuietProgressBar.

Animations are suppressed when: * `context.quiet` is set, * the output is not a TTY (e.g. piped to a file), or * `TERM=dumb` or `TERM=unknown` is set.

spinner(*message: str*, *fail_message: str = 'failed\n'*) → *conda.plugins.types.SpinnerBase*

Determines whether to return a Spinner or QuietSpinner.

Animations are suppressed under the same conditions as *progress_bar()*.

prompt(*message='Proceed'*, *choices=('yes', 'no')*, *default='yes'*) → str

Implementation of a prompt dialog

conda_reporter_backends()

Reporter backend for console

json

Defines a JSON reporter backend

This reporter backend is used to provide JSON strings for output rendering. It is essentially just a wrapper around `conda.common.serialize.json.dumps`.

Classes

JSONProgressBar	Progress bar that outputs JSON to stdout
JSONReporterRenderer	Default implementation for JSON reporting in conda
JSONSpinner	This class for a JSONSpinner does nothing because we do not want to include this output.

Functions

conda_reporter_backends()	Reporter backend for JSON

class JSONProgressBar(*description: str*, *enabled: bool = True*, ***kwargs*)

Bases: conda.plugins.types.ProgressBarBase

Progress bar that outputs JSON to stdout

enabled = True

update_to(*fraction*) → None

refresh()

close()

classmethod get_lock()

Used for our own sys.stdout.write/flush calls

class JSONReporterRenderer

Bases: conda.plugins.types.ReporterRendererBase

Default implementation for JSON reporting in conda

render(*data: Any*, ***kwargs*) → str

detail_view(*data: dict[str, str | int | bool]*, ***kwargs*) → str

Render the output in a "tabular" format.

envs_list(*data: list[str] | dict[str, dict[str, str | bool | None]]*, ***kwargs*) → str

Render a list of environments

progress_bar(*description: str*, ***kwargs*) → *conda.plugins.types.ProgressBarBase*

Return a ProgressBarBase~ object to use as a progress bar

spinner(*message: str*, *fail_message: str = 'failed\n'*) → *conda.plugins.types.SpinnerBase*

Return a SpinnerBase~ object to use as a spinner (i.e. loading dialog)

prompt(*message: str = 'Proceed'*, *choices=('yes', 'no')*, *default: str = 'yes'*) → str

For this class, we want this method to do nothing

class JSONSpinner(*message: str*, *fail_message: str = 'failed\n'*)

Bases: conda.plugins.types.SpinnerBase

This class for a JSONSpinner does nothing because we do not want to include this output.

__enter__()

__exit__(*exc_type*, *exc_val*, *exc_tb*)

conda_reporter_backends()

Reporter backend for JSON

This is the default reporter backend that returns objects as JSON strings.

plugins

solvers

Register the classic conda solver.

Functions

conda_solvers()	The classic solver as shipped by default in conda.

conda_solvers()

The classic solver as shipped by default in conda.

subcommands

doctor

Implementation for *conda doctor* subcommand. Adds various environment and package checks to detect issues or possible environment corruption.

health_checks

Health checks for *conda doctor*.

This package contains individual health check modules that are registered via the conda_health_checks plugin hook.

altered_files

Health check: Altered files in packages.

Functions

find_altered_packages(prefix) → dict[str, list[str]]	Finds packages with altered files (checksum mismatch).
altered_files(prefix, verbose) → None	Health check action: Report packages with altered files.
fix_altered_files(prefix, args, confirm) → int	Fix altered files by reinstalling affected packages.
conda_health_checks() → collections.abc.Iterable[conda.plugins.types.CondaHealthChecl	Register the altered files health check.

Attributes

logger

logger

find_altered_packages(*prefix: str | pathlib.Path*) → dict[str, list[str]]

Finds packages with altered files (checksum mismatch).

altered_files(*prefix: str*, *verbose: bool*) → None

Health check action: Report packages with altered files.

fix_altered_files(*prefix: str*, *args: argparse.Namespace*, *confirm: conda.plugins.types.ConfirmCallback*) → int

Fix altered files by reinstalling affected packages.

conda_health_checks() → collections.abc.Iterable[*conda.plugins.types.CondaHealthCheck*]

Register the altered files health check.

consistency

Health check: Environment consistency (dependencies).

Functions

find_inconsistent_packages(prefix_data) → tuple[dict[str, dict[str, list]], set[str]]	Find packages with missing or inconsistent dependencies.
consistent_env_check(prefix, verbose) → None	Health check action: Check environment consistency.
fix_inconsistent_packages(prefix, args, confirm) → int	Fix inconsistent packages by updating the environment.
conda_health_checks() → collections.abc.Iterable[conda.plugins.types.CondaHealthChecl	Register the consistency health check.

find_inconsistent_packages(*prefix_data:* conda.core.prefix_data.PrefixData) → tuple[dict[str, dict[str, list]], set[str]]

Find packages with missing or inconsistent dependencies.

Returns a tuple of (issues dict, missing dependency names set). The issues dict maps package name -> {"missing": [...], "inconsistent": [...]}.

consistent_env_check(*prefix: str*, *verbose: bool*) → None

Health check action: Check environment consistency.

fix_inconsistent_packages(*prefix: str*, *args: argparse.Namespace*, *confirm: conda.plugins.types.ConfirmCallback*) → int

Fix inconsistent packages by updating the environment.

conda_health_checks() → collections.abc.Iterable[*conda.plugins.types.CondaHealthCheck*]

Register the consistency health check.

environment_txt

Health check: Environment listed in environments.txt.

Functions

check_envs_txt_file(prefix) → bool	Checks whether the environment is listed in the environments.txt file.
env_txt_check(prefix, verbose) → None	Health check action: Check if environment is in environments.txt.
fix_env_txt(prefix, args, confirm) → int	Register environment in environments.txt.
conda_health_checks() → collections.abc.Iterable[conda.plugins.types.CondaHealthChecl	Register the environment.txt health check.

Attributes

logger

logger

check_envs_txt_file(*prefix: str | os.PathLike | pathlib.Path*) → bool

Checks whether the environment is listed in the environments.txt file.

env_txt_check(*prefix: str*, *verbose: bool*) → None

Health check action: Check if environment is in environments.txt.

fix_env_txt(*prefix: str*, *args: argparse.Namespace*, *confirm: conda.plugins.types.ConfirmCallback*) → int

Register environment in environments.txt.

conda_health_checks() → collections.abc.Iterable[*conda.plugins.types.CondaHealthCheck*]

Register the environment.txt health check.

file_locking

Health check: File locking support.

Functions

file_locking_check(prefix, verbose) → None	Health check action: Report if file locking is supported.
conda_health_checks() → collections.abc.Iterable[conda.plugins.types.CondaHealthChecl	Register the file locking health check.

file_locking_check(*prefix: str*, *verbose: bool*) → None

Health check action: Report if file locking is supported.

conda_health_checks() → collections.abc.Iterable[*conda.plugins.types.CondaHealthCheck*]

Register the file locking health check.

missing_files

Health check: Missing files in packages.

Functions

excluded_files_check(filename) → bool	Check if a file should be excluded from health checks.
find_packages_with_missing_files(prefix) → dict[str, list[str]]	Finds packages listed in conda-meta which have missing files.
missing_files(prefix, verbose) → None	Health check action: Report packages with missing files.
fix_missing_files(prefix, args, confirm) → int	Fix missing files by reinstalling affected packages.
conda_health_checks() → collections.abc.Iterable[conda.plugins.types.CondaHealthChecl	Register the missing files health check.

Attributes

logger

logger

excluded_files_check(*filename: str*) → bool

Check if a file should be excluded from health checks.

find_packages_with_missing_files(*prefix: str | pathlib.Path*) → dict[str, list[str]]

Finds packages listed in conda-meta which have missing files.

missing_files(*prefix: str*, *verbose: bool*) → None

Health check action: Report packages with missing files.

fix_missing_files(*prefix: str*, *args: argparse.Namespace*, *confirm: conda.plugins.types.ConfirmCallback*) → int

Fix missing files by reinstalling affected packages.

conda_health_checks() → collections.abc.Iterable[*conda.plugins.types.CondaHealthCheck*]

Register the missing files health check.

pinned

Health check: Pinned file format.

Functions

find_malformed_pinned_specs(prefix_data) → list[conda.models.match_spec.MatchSpec]	Find pinned specs that reference packages not installed in the environment.
pinned_well_formatted_check(prefix, verbose) → None	Health check action: Check pinned file format.
fix_malformed_pinned(prefix, args, confirm) → int	Clean up malformed specs in pinned file.
conda_health_checks() → collections.abc.Iterable[conda.plugins.types.CondaHealthChec	Register the pinned file health check.

find_malformed_pinned_specs(*prefix_data: conda.core.prefix_data.PrefixData*) → list[*conda.models.match_spec.MatchSpec*]

Find pinned specs that reference packages not installed in the environment.

Returns a list of MatchSpec objects for packages that might be typos.

pinned_well_formatted_check(*prefix: str*, *verbose: bool*) → None

Health check action: Check pinned file format.

fix_malformed_pinned(*prefix: str*, *args: argparse.Namespace*, *confirm: conda.plugins.types.ConfirmCallback*) → int

Clean up malformed specs in pinned file.

conda_health_checks() → collections.abc.Iterable[*conda.plugins.types.CondaHealthCheck*]

Register the pinned file health check.

requests_ca_bundle

Health check: REQUESTS_CA_BUNDLE environment variable.

Functions

requests_ca_bundle_check(prefix, verbose) → None	Health check action: Verify REQUESTS_CA_BUNDLE environment variable.
conda_health_checks() → collections.abc.Iterable[conda.plugins.types.CondaHealthChecl	Register the REQUESTS_CA_BUNDLE health check.

requests_ca_bundle_check(*prefix: str*, *verbose: bool*) → None

Health check action: Verify REQUESTS_CA_BUNDLE environment variable.

conda_health_checks() → collections.abc.Iterable[*conda.plugins.types.CondaHealthCheck*]

Register the REQUESTS_CA_BUNDLE health check.

utils

Health check: Helpful shared functions for performing health checks.

Functions

get_spec_str_from_conda_meta(prefix, packages) → tuple[list[str], list[str]]	Get a list of specs (specified with name, version, and build) from the prefix
get_name_from_conda_meta_file_name(conda_meta → str	Given the name of a package conda_meta file, extract the name of the package

Attributes

logger

logger

get_spec_str_from_conda_meta(*prefix: str*, *packages: list[str]*) → tuple[list[str], list[str]]

Get a list of specs (specified with name, version, and build) from the prefix matching the name of the requested packages. The conda-meta is generally assumed to be intact. However, if the name, version or build metadata is incomplete, the package will be skipped.

get_name_from_conda_meta_file_name(*conda_meta_filename: str*) → str

Given the name of a package conda_meta file, extract the name of the package

plugins

Classes

PrefixData	The PrefixData class aims to be the representation of the state

continues on next page

Table 307 – continued from previous page

CondaSubcommand	Return type to use when defining a conda subcommand plugin hook.

Functions

add_output_and_prompt_options(p) → argparse._ArgumentGroup	
add_parser_frozen_env(p)	
add_parser_help(p) → None	So we can use consistent capitalization and periods in the help. You must
add_parser_prefix(p[, prefix_required]) → argparse._MutuallyExclusiveGroup	
confirm_yn([message, default, dry_run]) → bool	Display a "yes/no" confirmation input
configure_parser(parser)	
execute(args) → int	Run registered health_check plugins and optionally fix issues.
conda_subcommands()	

Attributes

context	
hookimpl	Decorator to mark plugin hook implementations, used to register plugins.

context

add_output_and_prompt_options(*p: argparse.ArgumentParser*) → argparse._ArgumentGroup

add_parser_frozen_env(*p: argparse.ArgumentParser*)

add_parser_help(*p: argparse.ArgumentParser*) → None

So we can use consistent capitalization and periods in the help. You must use the add_help=False argument to ArgumentParser or add_parser to use this. Add this first to be consistent with the default argparse output.

add_parser_prefix(*p: argparse.ArgumentParser*, *prefix_required: bool = False*) → argparse._MutuallyExclusiveGroup

class PrefixData(*prefix_path: conda.common.path.PathType*, *interoperability: bool | None = None*)

The PrefixData class aims to be the representation of the state of a conda environment on disk. The directory where the environment lives is called prefix.

This class supports different types of tasks:

- Reading and querying *conda-meta/*.json* files as *PrefixRecord* objects
- Reading and writing environment-specific configuration (env vars, state file, nonadmin markers, etc)
- Existence checks and validations of name, path, and magic files / markers
- Exposing non-conda packages installed in prefix as *PrefixRecord*, via the plugin system

cache: dict[tuple[pathlib.Path, bool | None], *PrefixData*]

prefix_path: pathlib.Path

_magic_file: pathlib.Path

_frozen_file: pathlib.Path

_last_activated_file: pathlib.Path

__prefix_records: dict[str, *conda.models.records.PrefixRecord*] | None = None

__is_writable: bool | None | conda.auxlib._Null

interoperability: bool

classmethod from_name(*name: str*, ***kwargs*) → *PrefixData*

Creates a PrefixData instance from an environment name.

The name will be validated with *PrefixData.validate_name()* if it does not exist.

Parameters
name -- The name of the environment. Must not contain path separators (/,).

Raises
CondaValueError -- If *name* contains a path separator.

classmethod from_context(*validate: bool = False*, ***kwargs*) → *PrefixData*

Creates a PrefixData instance from the path specified by *context.target_prefix*.

The path and name will be validated with *PrefixData.validate_path()* and *PrefixData.validate_name()*, respectively, if *validate* is *True*.

Parameters

- **validate** -- Whether the path and name should be validated. Useful for environments about to be created.
- **kwargs** -- Additional keyword arguments to pass to the constructor.

property name: str

Returns the name of the environment, if available.

If the environment doesn't live in one the configured *envs_dirs*, an empty string is returned. The construct *prefix_data.name or prefix_data.prefix_path* can be helpful in those cases.

__eq__(*other: Any*) → bool

exists() → bool

Check whether the PrefixData path exists and is a directory.

is_environment() → bool

Check whether the PrefixData path is a valid conda environment.

This is assessed by checking if *conda-meta/history* marker file exists.

is_frozen() → bool

Check whether the environment is marked as frozen, as per CEP 22.

This is assessed by checking if *conda-meta/frozen* marker file exists.

is_base() → bool

Check whether the configured path refers to the *base* environment.

property is_active: bool

Whether this prefix is the currently active environment.

Though is_active isn't an on disk characteristic of an environment, it is a nice to have when assessing the state of an environment.

property is_writable: bool | None | conda.auxlib._Null

Check whether the configured path is writable. This is assessed by checking whether *conda-meta/history* is writable. It if is, it is assumed that the rest of the directory tree is writable too.

Note: The value is cached in the instance. Use *.assert_writable()* for a non- cached check.

assert_exists() → None

Check whether the environment path exists.

Raises
EnvironmentLocationNotFound -- If the check returns False.

assert_environment() → None

Check whether the environment path exists and is a valid conda environment.

Raises
DirectoryNotACondaEnvironmentError -- If the check returns False.

assert_writable() → None

Check whether the environment path is a valid conda environment and is writable.

Raises
EnvironmentNotWritableError -- If the check returns False.

assert_not_frozen() → None

Check whether the environment path is a valid conda environment and is not marked as frozen (as per CEP 22).

Raises
EnvironmentIsFrozenError -- If the environment is marked as frozen.

validate_path(*expand_path: bool = False*) → None

Validate the path of the environment.

It runs the following checks:

- Make sure the path does not contain *:* or *;* (OS-dependent).
- Disallow immediately nested environments (e.g. *$CONDA_ROOT* and *$CONDA_ROOT/my-env*).
- Warn if there are spaces in the path.

Parameters
expand_path -- Whether to process ~ and environment variables in the string. The expanded value will replace *.prefix_path*.

Raises
CondaValueError -- If the environment contains *:*, *;*, or is nested.

validate_name(*allow_base: bool = False*) → None

Validate the name of the environment.

Parameters
allow_base -- Whether to allow *base* as a valid name.

Raises

- *CondaValueError* -- If the name is protected, or if it contains disallowed characters
- (/, ` `, ` -- , `#).

property created: datetime.datetime | None

Returns the time when the environment was created, as evidenced by the *conda-meta* directory creation time (if available). Falls back to *conda-meta/created_at*.

This may return None in the following cases:

- If the environment doesn't exist.
- If the creation time cannot be determined. This can happen for existing environments created with conda versions before 25.11 in Linux systems, where the 'birthtime' metadata is only available in kernels 4.11+, and not implemented in Python as of Python 3.14.

property last_modified: datetime.datetime | None

Returns the time when the environment was last modified, as evidenced by the *conda-meta/history* file modification time. If the environment does not exist, returns None.

property last_activated: datetime.datetime | None

Returns the time when the environment was last activated, as evidenced by the *conda-meta/last_activated* filesystem modification time. If the environment or file doesn't exist, returns None.

size() → int

Compute the total size of a conda environment prefix.

This aggregates the installed size of all packages in the environment, plus the size of non-manifest files under conda-meta (history, markers, etc.)

Returns
Total size in bytes.

load() → None

reload() → Self

_get_json_fn(*prefix_record:* conda.models.records.PrefixRecord) → str

insert(*prefix_record:* conda.models.records.PrefixRecord, *remove_auth: bool = True*) → None

remove(*package_name: str*) → None

get(*package_name: str*, *default:* T = *NULL*) → *conda.models.records.PrefixRecord* | *T*

iter_records() → collections.abc.Iterable[*conda.models.records.PrefixRecord*]

iter_records_sorted() → collections.abc.Iterable[*conda.models.records.PrefixRecord*]

map_records() → frozendict.frozendict[str, *conda.models.records.PrefixRecord*]

Map the records to a frozendict of name -> record.

Returns
A mapping of name -> record.

all_subdir_urls() → set[str]

query(*package_ref_or_match_spec:* conda.models.records.PackageRecord | conda.models.match_spec.MatchSpec | *str*) → collections.abc.Iterable[*conda.models.records.PrefixRecord*]

get_conda_packages() → list[*conda.models.records.PrefixRecord*]

Get conda packages sorted alphabetically by name.

Returns
Sorted conda package records.

get_python_packages() → list[*conda.models.records.PrefixRecord*]

Get Python packages (installed via pip) sorted alphabetically by name.

Returns
Sorted Python package records.

property _prefix_records: dict[str, *conda.models.records.PrefixRecord*] | None

_load_single_record(*prefix_record_json_path: conda.common.path.PathType*) → None

_get_environment_state_file() → dict[str, dict[str, str]]

_write_environment_state_file(*state: dict[str, dict[str, str]]*) → None

_ensure_no_reserved_env_vars(*env_vars_names: collections.abc.Iterable[str]*) → None

Ensure that the set of env_var_names does not contain any reserved env vars. Will raise an OperationNotAllowed if a reserved env var is found.

get_environment_env_vars() → dict[str, str] | dict[bytes, bytes]

set_environment_env_vars(*env_vars: dict[str, str]*) → dict[str, str] | None

unset_environment_env_vars(*env_vars: list[str]*) → dict[str, str] | None

set_creation_time() → None

Writes a .creation-time file in conda-meta with the current timestamp, meant to be used by .created property as a fallback.

set_nonadmin() → None

Creates $PREFIX/.nonadmin if sys.prefix/.nonadmin exists (on Windows).

get_pinned_specs() → tuple[*conda.models.match_spec.MatchSpec*]

Find pinned specs from file and return a tuple of MatchSpec.

exception CondaSystemExit(**args*)

Bases: *conda.CondaExitZero*, SystemExit

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception DryRunExit

Bases: *conda.CondaExitZero*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

confirm_yn(*message: str = 'Proceed'*, *default='yes'*, *dry_run=None*) → bool

Display a "yes/no" confirmation input

hookimpl

Decorator to mark plugin hook implementations, used to register plugins.

class CondaSubcommand(*, *name: str*, *summary: str*, *action: collections.abc.Callable[[argparse.Namespace], int | None]*, *configure_parser: collections.abc.Callable[[argparse.ArgumentParser], None]*, *aliases: str | collections.abc.Iterable[str] = ()*)

class CondaSubcommand(*, *name: str*, *summary: str*, *action: collections.abc.Callable[[tuple[str, ...]], int | None]*, *configure_parser: None = None*, *aliases: str | collections.abc.Iterable[str] = ()*)

Bases: `CondaPlugin`

Return type to use when defining a conda subcommand plugin hook.

For details on how this is used, see *conda_subcommands()*.

Subcommands support two shapes, distinguished by `configure_parser`:

- If `configure_parser` is set, `action` receives the parsed `argparse.Namespace`.
- If `configure_parser` is omitted, `action` receives the remaining argv as `tuple[str, ...]`.

Parameters

- **name** -- Subcommand name (e.g., `conda my-subcommand-name`).
- **summary** -- Subcommand summary, will be shown in `conda --help`.
- **action** -- Callable that will be run when the subcommand is invoked.
- **aliases** -- Alternative name or names for the subcommand.
- **configure_parser** -- Callable that will be run when the subcommand parser is initialized.

summary: `str`

action: `collections.abc.Callable[[argparse.Namespace], int | None] | collections.abc.Callable[[tuple[str, ...]], int | None]`

aliases: `tuple[str, ...] = ()`

configure_parser: `collections.abc.Callable[[argparse.ArgumentParser], None] | None = None`

configure_parser(*parser: argparse.ArgumentParser*)

execute(*args: argparse.Namespace*) → int

Run registered health_check plugins and optionally fix issues.

conda_subcommands()

plugins

Implementation for *conda plugins* subcommand.

info

Implementation for *conda plugins info* subcommand.

Functions

configure_parser(parser) → None
execute(args) → int

Attributes

HELP

HELP = 'Show detailed information about an installed conda plugin.'

configure_parser(*parser: argparse.ArgumentParser*) → None

execute(*args: argparse.Namespace*) → int

install

Implementation for *conda plugins install* subcommand.

Functions

configure_parser(parser) → None
execute(args) → int

Attributes

HELP

HELP = 'Install conda plugin packages into an environment.'

configure_parser(*parser: argparse.ArgumentParser*) → None

execute(*args: argparse.Namespace*) → int

list

Implementation for *conda plugins list* subcommand.

Functions

configure_parser(parser) → None
execute(args) → int

Attributes

HELP

HELP = 'List installed conda plugins.'

configure_parser(*parser: argparse.ArgumentParser*) → None

execute(*args: argparse.Namespace*) → int

package_validation

Validation for plugin package operations.

Functions

get_installed_plugin_package_names(prefix) → frozenset[str]	Return conda plugin package names installed in a prefix.
require_installed_plugin_specs(specs, prefix, *, command) → None	
require_plugin_update_environment(environment) → None	Validate requested plugin updates after merging file input.
require_plugin_transaction(unlink_link_transactic *) → None	Validate requested plugin packages in a transaction.
require_explicit_plugin_packages(package_recor → None	

get_installed_plugin_package_names(*prefix: conda.common.path.PathType*) → frozenset[str]

Return conda plugin package names installed in a prefix.

require_installed_plugin_specs(*specs: collections.abc.Iterable[str | conda.models.match_spec.MatchSpec]*, *prefix: conda.common.path.PathType*, *, *command: str*) → None

require_plugin_update_environment(*environment: conda.models.environment.Environment*) → None

Validate requested plugin updates after merging file input.

require_plugin_transaction(*unlink_link_transaction: conda.core.link.UnlinkLinkTransaction*, *, *prefetch_link_precs: bool = False*, *command: str = 'install'*) → None

Validate requested plugin packages in a transaction.

require_explicit_plugin_packages(*package_records: collections.abc.Iterable[conda.models.records.PackageCacheRecord]*) → None

remove

Implementation for *conda plugins remove* subcommand.

Functions

configure_parser(parser) → None	
execute(args) → int	

Attributes

HELP

HELP = 'Remove conda plugin packages from an environment.'

configure_parser(*parser: argparse.ArgumentParser*) → None

execute(*args: argparse.Namespace*) → int

update

Implementation for *conda plugins update* subcommand.

Functions

configure_parser(parser) → None	
execute(args) → int	

Attributes

HELP	

HELP = 'Update conda plugin packages in an environment.'

configure_parser(*parser: argparse.ArgumentParser*) → None

execute(*args: argparse.Namespace*) → int

Classes

CondaSubcommand	Return type to use when defining a conda subcommand plugin hook.

Functions

configure_parser(parser) → None	
execute(args) → int	
conda_subcommands()	

Attributes

hookimpl	Decorator to mark plugin hook implementations, used to register plugins.
SUMMARY	

hookimpl

Decorator to mark plugin hook implementations, used to register plugins.

class CondaSubcommand(**, name: str, summary: str, action: collections.abc.Callable[[argparse.Namespace], int | None], configure_parser: collections.abc.Callable[[argparse.ArgumentParser], None], aliases: str | collections.abc.Iterable[str] = ()*)

class CondaSubcommand(**, name: str, summary: str, action: collections.abc.Callable[[tuple[str, ...]], int | None], configure_parser: None = None, aliases: str | collections.abc.Iterable[str] = ()*)

Bases: CondaPlugin

Return type to use when defining a conda subcommand plugin hook.

For details on how this is used, see *conda_subcommands()*.

Subcommands support two shapes, distinguished by `configure_parser`:

- If `configure_parser` is set, `action` receives the parsed `argparse.Namespace`.
- If `configure_parser` is omitted, `action` receives the remaining argv as `tuple[str, ...]`.

Parameters

- **name** -- Subcommand name (e.g., `conda my-subcommand-name`).
- **summary** -- Subcommand summary, will be shown in `conda --help`.
- **action** -- Callable that will be run when the subcommand is invoked.
- **aliases** -- Alternative name or names for the subcommand.
- **configure_parser** -- Callable that will be run when the subcommand parser is initialized.

summary: `str`

action: `collections.abc.Callable[[argparse.Namespace], int | None] | collections.abc.Callable[[tuple[str, ...]], int | None]`

aliases: `tuple[str, ...] = ()`

configure_parser: `collections.abc.Callable[[argparse.ArgumentParser], None] | None = None`

SUMMARY = 'Manage conda plugins.'

configure_parser(*parser: argparse.ArgumentParser*) → None

execute(*args: argparse.Namespace*) → int

conda_subcommands()

plugins

types

Definition of specific return types for use when defining a conda plugin hook.

Each type corresponds to the plugin hook for which it is used.

Classes

CondaPlugin	Base class for all conda plugins.
CondaSubcommand	Return type to use when defining a conda subcommand plugin hook.
CondaVirtualPackage	Return type to use when defining a conda virtual package plugin hook.
CondaSolver	Return type to use when defining a conda solver plugin hook.

continues on next page

Table 324 – continued from previous page

CondaPreCommand	Return type to use when defining a conda pre-command plugin hook.
CondaPostCommand	Return type to use when defining a conda post-command plugin hook.
ChannelNameMixin	Class mixin to make all plugin implementations compatible, e.g. when they
ChannelAuthBase	Base class that we require all plugin implementations to use to be compatible.
CondaAuthHandler	Return type to use when the defining the conda auth handlers hook.
CondaHealthCheck	Return type to use when defining conda health checks plugin hook.
CondaPreSolve	Return type to use when defining a conda pre-solve plugin hook.
CondaPostSolve	Return type to use when defining a conda post-solve plugin hook.
CondaSetting	Return type to use when defining a conda setting plugin hook.
ProgressBarBase	Helper class that provides a standard way to create an ABC using
SpinnerBase	Helper class that provides a standard way to create an ABC using
ReporterRendererBase	Base class for all reporter renderers.
CondaReporterBackend	Return type to use when defining a conda reporter backend plugin hook.
CondaRequestHeader	Define vendor specific headers to include HTTP requests
CondaErrorHint	Return type to use when defining a conda error hints plugin hook.
CondaPreTransactionAction	Return type to use when defining a pre-transaction action hook.
CondaPostTransactionAction	Return type to use when defining a post-transaction action hook.
CondaPrefixDataLoader	Define new loaders to expose non-conda packages in a given prefix
EnvironmentSpecBase	**EXPERIMENTAL**
EnvironmentFormat	Represents supported environment formats.
CondaEnvironmentSpecifier	**EXPERIMENTAL**
CondaEnvironmentExporter	**EXPERIMENTAL**
CondaPackageExtractor	Return type to use when defining a conda package extractor plugin hook.
CondaExceptionEvent	Structured exception event passed to exception observer plugin callbacks.
CondaExceptionObserver	Return type to use when defining a conda exception observer plugin hook.

Attributes

CondaPrefixDataLoaderCallable

```
type CondaPrefixDataLoaderCallable = Callable[[PathType, dict[str, PrefixRecord]],
dict[str, PrefixRecord]]
```

class CondaPlugin

Base class for all conda plugins.

name: str

User-facing name of the plugin used for selecting & filtering plugins and error messages.

__post_init__()

class CondaSubcommand(*, *name: str*, *summary: str*, *action: collections.abc.Callable[[argparse.Namespace], int | None]*, *configure_parser: collections.abc.Callable[[argparse.ArgumentParser], None]*, *aliases: str | collections.abc.Iterable[str] = ()*)

class CondaSubcommand(*, *name: str*, *summary: str*, *action: collections.abc.Callable[[tuple[str, ...]], int | None]*, *configure_parser: None = None*, *aliases: str | collections.abc.Iterable[str] = ()*)

Bases: *CondaPlugin*

Return type to use when defining a conda subcommand plugin hook.

For details on how this is used, see *conda_subcommands()*.

Subcommands support two shapes, distinguished by `configure_parser`:

- If `configure_parser` is set, `action` receives the parsed `argparse.Namespace`.
- If `configure_parser` is omitted, `action` receives the remaining argv as `tuple[str, ...]`.

Parameters

- **name** -- Subcommand name (e.g., `conda my-subcommand-name`).
- **summary** -- Subcommand summary, will be shown in `conda --help`.
- **action** -- Callable that will be run when the subcommand is invoked.
- **aliases** -- Alternative name or names for the subcommand.
- **configure_parser** -- Callable that will be run when the subcommand parser is initialized.

summary: str

action: collections.abc.Callable[[argparse.Namespace], int | None] | collections.abc.Callable[[tuple[str, ...]], int | None]

aliases: tuple[str, ...] = ()

configure_parser: collections.abc.Callable[[argparse.ArgumentParser], None] | None = None

class CondaVirtualPackage

Bases: *CondaPlugin*

Return type to use when defining a conda virtual package plugin hook.

For details on how this is used, see *conda_virtual_packages()*.

Note

The `version` and `build` parameters can be provided in two ways:

1. Direct values: a string or `None` (where `None` translates to `0`)

2. Deferred callables: functions that return either a string, `None` (translates to `0`), or `NULL` (indicates the virtual package should not be exported)

Parameters

- **name** -- Virtual package name (e.g., `my_custom_os`).
- **version** -- Virtual package version (e.g., `1.2.3`).
- **build** -- Virtual package build string (e.g., `x86_64`).
- **override_entity** -- Can be set to either to "version" or "build", the corresponding value will be overridden if the environment variable `CONDA_OVERRIDE_<name>` is set.
- **empty_override** -- Value to use for version or build if the override environment variable is set to an empty string. By default, this is `NULL`.
- **version_validation** -- Optional version validation function to ensure that the override version follows a certain pattern.

name: *str*

User-facing name of the plugin used for selecting & filtering plugins and error messages.

version: *str | None | collections.abc.Callable[[], str | None | conda.auxlib._Null]*

build: *str | None | collections.abc.Callable[[], str | None | conda.auxlib._Null]*

override_entity: *Literal['version', 'build'] | None = None*

empty_override: *None | conda.auxlib._Null*

version_validation: *collections.abc.Callable[[str], str | None] | None = None*

to_virtual_package() → *conda.models.records.PackageRecord* | conda.auxlib._Null

class CondaSolver

Bases: *CondaPlugin*

Return type to use when defining a conda solver plugin hook.

For details on how this is used, see *conda_solvers()*.

Parameters

- **name** -- Solver name (e.g., `custom-solver`).
- **backend** -- Type that will be instantiated as the solver backend.

name: *str*

User-facing name of the plugin used for selecting & filtering plugins and error messages.

backend: *type[conda.core.solve.Solver]*

class CondaPreCommand

Bases: *CondaPlugin*

Return type to use when defining a conda pre-command plugin hook.

For details on how this is used, see *conda_pre_commands()*.

Parameters

- **name** -- Pre-command name (e.g., `custom_plugin_pre_commands`).

- **action** -- Callable which contains the code to be run.
- **run_for** -- Represents the command(s) this will be run on (e.g. `install` or `create`).

name: *str*

User-facing name of the plugin used for selecting & filtering plugins and error messages.

action: *collections.abc.Callable[[str], None]*

run_for: *set[str]*

class CondaPostCommand

Bases: *CondaPlugin*

Return type to use when defining a conda post-command plugin hook.

For details on how this is used, see *conda_post_commands()*.

Parameters

- **name** -- Post-command name (e.g., `custom_plugin_post_commands`).
- **action** -- Callable which contains the code to be run.
- **run_for** -- Represents the command(s) this will be run on (e.g. `install` or `create`).

name: *str*

User-facing name of the plugin used for selecting & filtering plugins and error messages.

action: *collections.abc.Callable[[str], None]*

run_for: *set[str]*

class ChannelNameMixin(*channel_name: str*, **args*, ***kwargs*)

Class mixin to make all plugin implementations compatible, e.g. when they use an existing (e.g. 3rd party) requests authentication handler.

Please use the concrete *ChannelAuthBase* in case you're creating an own implementation.

channel_name

class ChannelAuthBase(*channel_name: str*, **args*, ***kwargs*)

Bases: *ChannelNameMixin*, `requests.auth.AuthBase`

Base class that we require all plugin implementations to use to be compatible.

Authentication is tightly coupled with individual channels. Therefore, an additional `channel_name` property must be set on the `requests.auth.AuthBase` based class.

class CondaAuthHandler

Bases: *CondaPlugin*

Return type to use when the defining the conda auth handlers hook.

Parameters

- **name** -- Name (e.g., `basic-auth`). This name should be unique and only one may be registered at a time.
- **handler** -- Type that will be used as the authentication handler during network requests.

name: *str*

User-facing name of the plugin used for selecting & filtering plugins and error messages.

handler: *type*[*ChannelAuthBase*]

class CondaHealthCheck

Bases: *CondaPlugin*

Return type to use when defining conda health checks plugin hook.

Health checks are diagnostic actions that report on the state of a conda environment. They are invoked via `conda doctor`.

Health checks can optionally provide a fix capability, which is invoked via `conda doctor --fix` or `conda doctor --fix <name>`.

Fixer guidelines:

Fixers receive a `confirm` function that handles user confirmation and dry-run mode automatically. Simply call it with your message:

- In normal mode: Prompts the user for confirmation (default: no).
- In dry-run mode: Raises `DryRunExit` (handled by the framework).
- If user declines: Raises `CondaSystemExit` (handled by the framework).

Example

from conda.plugins.types import ConfirmCallback

def my_fixer(prefix: str, args: Namespace, confirm: ConfirmCallback) -> int:
issues = find_issues(prefix) if not issues:

print("No issues found.") return 0

print(f"Found {len(issues)} issues") confirm("Fix these issues?") # ... perform fix ... return 0

For details on how this is used, see *conda_health_checks()*.

Parameters

- **name** -- Health check identifier (e.g., `missing-files`).
- **action** -- Callable that performs the check: `action(prefix, verbose) -> None`.
- **fixer** -- Optional callable that fixes issues: `fixer(prefix, args, confirm) -> int`. The `confirm` parameter is a function to call for user confirmation. It raises an exception if the user declines or in dry-run mode.
- **summary** -- Short description of what the check detects (shown in `--list`).
- **fix** -- Short description of what the fix does (shown in `--list`).

name: str

User-facing name of the plugin used for selecting & filtering plugins and error messages.

action: collections.abc.Callable[[str, bool], None]

fixer: collections.abc.Callable[[str, argparse.Namespace, ConfirmCallback], int] | None = None

summary: str | None = None

fix: str | None = None

class CondaPreSolve

Bases: *CondaPlugin*

Return type to use when defining a conda pre-solve plugin hook.

For details on how this is used, see *conda_pre_solves()*.

Parameters

- **name** -- Pre-solve name (e.g., `custom_plugin_pre_solve`).
- **action** -- Callable which contains the code to be run.

name: str

User-facing name of the plugin used for selecting & filtering plugins and error messages.

action: collections.abc.Callable[[frozenset[*conda.models.match_spec.MatchSpec*], frozenset[*conda.models.match_spec.MatchSpec*]], None]

class CondaPostSolve

Bases: *CondaPlugin*

Return type to use when defining a conda post-solve plugin hook.

For details on how this is used, see *conda_post_solves()*.

Parameters

- **name** -- Post-solve name (e.g., `custom_plugin_post_solve`).
- **action** -- Callable which contains the code to be run.

name: str

User-facing name of the plugin used for selecting & filtering plugins and error messages.

action: collections.abc.Callable[[str, tuple[*conda.models.records.PackageRecord*, ...], tuple[*conda.models.records.PackageRecord*, ...]], None]

class CondaSetting

Bases: *CondaPlugin*

Return type to use when defining a conda setting plugin hook.

For details on how this is used, see *conda_settings()*.

Parameters

- **name** -- name of the setting (e.g., `config_param`)
- **description** -- description of the setting that should be targeted towards users of the plugin
- **parameter** -- Parameter instance containing the setting definition
- **aliases** -- alternative names of the setting

name: str

User-facing name of the plugin used for selecting & filtering plugins and error messages.

description: str

parameter: *conda.common.configuration.Parameter*

aliases: tuple[str, ...] = ()

class ProgressBarBase(*description: str*, ***kwargs*)

Bases: abc.ABC

Helper class that provides a standard way to create an ABC using inheritance.

description

abstractmethod update_to(*fraction*) → None

abstractmethod refresh() → None

abstractmethod close() → None

finish()

classmethod get_lock()

class SpinnerBase(*message: str*, *fail_message: str = 'failed\n'*)

Bases: abc.ABC

Helper class that provides a standard way to create an ABC using inheritance.

message

fail_message = Multiline-String

```
"""failed
"""
```

abstractmethod __enter__()

abstractmethod __exit__(*exc_type*, *exc_val*, *exc_tb*)

class ReporterRendererBase

Bases: abc.ABC

Base class for all reporter renderers.

render(*data: Any*, ***kwargs*) → str

abstractmethod detail_view(*data: dict[str, str | int | bool]*, ***kwargs*) → str

Render the output in a "tabular" format.

abstractmethod envs_list(*data: collections.abc.Iterable[str] | dict[str, dict[str, str | bool | None]]*, ***kwargs*) → str

Render a list of environments

abstractmethod progress_bar(*description: str*, ***kwargs*) → *ProgressBarBase*

Return a ProgressBarBase~ object to use as a progress bar

classmethod progress_bar_context_manager() → contextlib.AbstractContextManager

Returns a null context by default but allows plugins to define their own if necessary

abstractmethod spinner(*message*, *failed_message*) → *SpinnerBase*

Return a SpinnerBase~ object to use as a spinner (i.e. loading dialog)

abstractmethod prompt(*message: str = 'Proceed'*, *choices=('yes', 'no')*, *default: str = 'yes'*) → str

Allows for defining an implementation of a "yes/no" confirmation function

class CondaReporterBackend

Bases: *CondaPlugin*

Return type to use when defining a conda reporter backend plugin hook.

For details on how this is used, see: *conda_reporter_backends()*.

Parameters

- **name** -- name of the reporter backend (e.g., `email_reporter`) This is how the reporter backend will be referenced in configuration files.
- **description** -- short description of what the reporter handler does
- **renderer** -- implementation of `ReporterRendererBase` that will be used as the reporter renderer

name: *str*

User-facing name of the plugin used for selecting & filtering plugins and error messages.

description: *str*

renderer: *type*[*ReporterRendererBase*]

class CondaRequestHeader

Bases: *CondaPlugin*

Define vendor specific headers to include HTTP requests

For details on how this is used, see *conda_request_headers()* and *conda_session_headers()*.

Parameters

- **name** -- name of the header used in the HTTP request
- **value** -- value of the header used in the HTTP request

name: *str*

User-facing name of the plugin used for selecting & filtering plugins and error messages.

value: *str*

class CondaErrorHint

Bases: *conda._private.exception_guidance.GuidanceHint*

Return type to use when defining a conda error hints plugin hook.

For details on how this is used, see *conda_error_hints()*.

Parameters

- **text** -- Human-readable description of the action to take.
- **hint_code** -- Stable machine-readable identifier. Use snake_case.

text: *str*

Human-readable description of the action to take.

hint_code: *str*

Stable machine-readable identifier. Use snake_case.

class CondaPreTransactionAction

Bases: *CondaPlugin*

Return type to use when defining a pre-transaction action hook.

For details on how this is used, see *conda_pre_transaction_actions()*.

Parameters

- **name** -- Pre transaction name (this is just a label)
- **action** -- Action class which implements plugin behavior. See *Action* for implementation details

name: str

User-facing name of the plugin used for selecting & filtering plugins and error messages.

action: *type*[*conda.core.path_actions.Action*]

class CondaPostTransactionAction

Bases: *CondaPlugin*

Return type to use when defining a post-transaction action hook.

For details on how this is used, see *conda_post_transaction_actions()*.

Parameters

- **name** -- Post transaction name (this is just a label)
- **action** -- Action class which implements plugin behavior. See *Action* for implementation details

name: str

User-facing name of the plugin used for selecting & filtering plugins and error messages.

action: *type*[*conda.core.path_actions.Action*]

class CondaPrefixDataLoader

Bases: *CondaPlugin*

Define new loaders to expose non-conda packages in a given prefix as `PrefixRecord` objects.

Parameters

- **name** -- name of the loader
- **loader** -- a function that takes a prefix and a dictionary that maps package names to `PrefixRecord` objects. The newly loaded packages must be inserted in the passed dictionary accordingly, and also returned as a separate dictionary.

name: str

User-facing name of the plugin used for selecting & filtering plugins and error messages.

loader: *CondaPrefixDataLoaderCallable*

class EnvironmentSpecBase

Bases: abc.ABC

EXPERIMENTAL

Base class for all environment specifications.

Environment specs parse different types of environment definition files (environment.yml, requirements.txt, pyproject.toml, etc.) into a common Environment object model.

detection_supported: ClassVar[bool] = True

abstractmethod can_handle() → bool

Determines if the EnvSpec plugin can read and operate on the environment described by the *filename*.

Returns
True, if the plugin can interpret the file.

Raises
Exception -- raises an exception if it can not handle the file. The exception should describe why the file can not be handled.

property env: *conda.models.environment.Environment*

Abstractmethod

Express the provided environment file as a conda environment object.

Returns
the conda environment represented by the file.

property available_platforms: tuple[str, ...]

Platforms this spec can produce an `Environment` for.

Defaults to `(context.subdir,)`. Multi-platform specs (`conda-lock.yml`, `pixi.lock`) override to return every platform declared in the input file.

env_for(*platform: str*) → *conda.models.environment.Environment*

Return the `Environment` for a specific platform.

Defaults to returning *env* when `platform` matches `context.subdir`, and raising `ValueError` otherwise. Multi-platform specs override this method to build the `Environment` directly from the parsed input file without constructing one per platform.

To iterate every platform a spec covers:

envs = (spec.env_for(p) for p in spec.available_platforms)

class EnvironmentFormat

Bases: `enum.StrEnum`

Represents supported environment formats.

Initialize self. See help(type(self)) for accurate signature.

lockfile = 'lockfile'

environment = 'environment'

property label: str

class CondaEnvironmentSpecifier

Bases: *CondaPlugin*

EXPERIMENTAL

Return type to use when defining a conda env spec plugin hook.

For details on how this is used, see *conda_environment_specifiers()*.

Parameters

- **name** -- name of the spec (e.g., `environment_yaml`)
- **aliases** -- user-friendly format aliases (e.g., ("yaml",)). Defaults to an empty list.

- **environment_spec** -- EnvironmentSpecBase subclass handler
- **default_filenames** -- default filename patterns this specifier handles (e.g., ("environment.yml", "*.conda-lock.yml"))
- **description** -- user-friendly description of what the format does. Defaults to the name if not provided.
- **environment_format** -- EnvironmentFormat category. Defaults to EnvironmentFormat.environment.

name: *str*

User-facing name of the plugin used for selecting & filtering plugins and error messages.

environment_spec: *type*[*EnvironmentSpecBase*]

default_filenames: tuple[str, ...] = ()

aliases: tuple[str, ...] = ()

description: str | None = None

environment_format: *EnvironmentFormat*

__post_init__()

class CondaEnvironmentExporter

Bases: *CondaPlugin*

EXPERIMENTAL

Return type to use when defining a conda environment exporter plugin hook supporting a single platform.

Parameters

- **name** -- name of the exporter (e.g., `environment-yaml`)
- **aliases** -- user-friendly format aliases (e.g., ("yaml",))
- **default_filenames** -- default filenames this exporter handles (e.g., ("environment.yml", "environment.yaml"))
- **export** -- callable that exports an Environment to string format for a single platform
- **multiplatform_export** -- callable that exports an Environment to string format for multiple platforms
- **description** -- user-friendly description of what the format does. Defaults to the name if not provided.
- **environment_format** -- EnvironmentFormat category. Defaults to EnvironmentFormat.environment.

name: *str*

User-facing name of the plugin used for selecting & filtering plugins and error messages.

aliases: tuple[str, ...]

default_filenames: tuple[str, ...]

export: SinglePlatformEnvironmentExport | None = None

multiplatform_export: MultiPlatformEnvironmentExport | None = None

description: *str | None = None*

environment_format: *EnvironmentFormat*

__post_init__()

class CondaPackageExtractor

Bases: *CondaPlugin*

Return type to use when defining a conda package extractor plugin hook.

Package extractors handle the extraction of different package archive formats. Each extractor specifies which file extensions it supports and provides an extraction function to unpack the archive.

For details on how this is used, see *conda_package_extractors()*.

Parameters

- **name** -- Extractor name (e.g., `conda-package`, `wheel-package`).
- **extensions** -- List of file extensions this extractor handles (e.g., `[".conda", ".tar.bz2"]` or `[".whl"]`).
- **extract** -- Callable that extracts the package archive. Takes the source archive path and the destination directory where the package contents should be extracted.

name: *str*

User-facing name of the plugin used for selecting & filtering plugins and error messages.

extensions: *list[str]*

extract: *PackageExtract*

class CondaExceptionEvent

Structured exception event passed to exception observer plugin callbacks.

Frozen to prevent plugins from mutating exception state. Structured args follow the `threading.ExceptHookArgs` / `sys.UnraisableHookArgs` pattern for forward compatibility.

The exception triple (`exc_type`, `exc_value`, `exc_traceback`) is always populated. The remaining fields describe the conda runtime state and default to `None` when the runtime isn't initialized (e.g. `MemoryError` during early startup). Runtime fields are populated all-or-nothing: if `conda_version` is not `None`, the runtime was available and all other fields are populated (`active_prefix` may still be `None` when no environment is active).

> **Warning**
>
> Do not store references to `exc_value` or `exc_traceback` beyond the lifetime of the callback. This can create reference cycles and prevent garbage collection.

Parameters

- **exc_type** -- The exception class.
- **exc_value** -- The exception instance.
- **exc_traceback** -- The traceback object.
- **argv** -- The command-line arguments at the time of error (frozen copy of `sys.argv`). `None` if unavailable.
- **conda_version** -- The conda version string. `None` if unavailable.

- **return_code** -- The exit code conda will return for this error. `None` if unavailable.
- **active_prefix** -- The currently active conda environment prefix, or `None` if no environment is active (also `None` when the runtime is unavailable).
- **target_prefix** -- The prefix the command was operating on.
- **channels** -- The configured channel names at the time of error (canonical names, e.g. `defaults`, `conda-forge`).
- **subdir** -- The platform subdirectory (e.g., `linux-64`, `osx-arm64`).
- **offline** -- Whether conda is running in offline mode (`--offline`).
- **dry_run** -- Whether conda is running in dry-run mode (`--dry-run`).
- **quiet** -- Whether conda is running in quiet mode (`--quiet`).
- **json** -- Whether conda is running in JSON output mode (`--json`).

exc_type: *type*[BaseException]

exc_value: BaseException

exc_traceback: types.TracebackType

argv: tuple[str, ...] | None = None

conda_version: str | None = None

return_code: int | None = None

active_prefix: str | None = None

target_prefix: str | None = None

channels: tuple[str, ...] | None = None

subdir: str | None = None

offline: bool | None = None

dry_run: bool | None = None

quiet: bool | None = None

json: bool | None = None

class CondaExceptionObserver

Bases: *CondaPlugin*

Return type to use when defining a conda exception observer plugin hook.

Exception observers are purely observational, modelled after CPython's `sys.excepthook`. They cannot suppress, modify, or redirect the exception. Their return value is ignored.

For details on how this is used, see *conda_exception_observers()*.

> **Warning**
>
> Do not store references to `exc_value` or `exc_traceback` beyond the lifetime of the callback. This can create reference cycles and prevent garbage collection.

Parameters

- **name** -- Observer name (e.g., `missing-package-reporter`).
- **hook** -- Callable invoked with a *CondaExceptionEvent* instance. Must not raise; any exception is caught and logged.
- **watch_for** -- Set of exception class names this observer watches for. Matches against the full MRO. Examples:
  - `{"BaseException"}` — fires for every exception.
  - `{"Exception"}` — all standard exceptions (excludes `KeyboardInterrupt`, `SystemExit`).
  - `{"CondaError"}` — all conda errors and subclasses.
  - `{"PackagesNotFoundError"}` — a specific error and its subclasses (e.g. `PackagesNotFoundInChannelsError`).
  - `{"MemoryError"}`, `{"KeyboardInterrupt"}`, `{"SystemExit"}` — specific non-conda exceptions.
  - `{"CondaError", "MemoryError"}` — combine scopes.

  For non-`CondaError` exceptions the conda-specific fields on *CondaExceptionEvent* may be `None`.

name: `str`

User-facing name of the plugin used for selecting & filtering plugins and error messages.

hook: `collections.abc.Callable[[CondaExceptionEvent], None]`

watch_for: `set[str]`

virtual_packages

archspec

Detect archspec name.

Functions

archspec_build()	
conda_virtual_packages()	→ collections.abc.Iterable[conda.plugins.types.CondaVirtualPack

archspec_build()

conda_virtual_packages() → collections.abc.Iterable[*conda.plugins.types.CondaVirtualPackage*]

conda

Expose conda version.

Functions

conda_virtual_packages() → collections.abc.Iterable[conda.plugins.types.CondaVirtualPack	

conda_virtual_packages() → collections.abc.Iterable[*conda.plugins.types.CondaVirtualPackage*]

cuda

Detect CUDA version.

Functions

cuda_version()	Attempt to detect the version of CUDA present in the operating system.
cached_cuda_version()	A cached version of the cuda detection system.
conda_virtual_packages() → collections.abc.Iterable[conda.plugins.types.CondaVirtualPack	
_cuda_driver_version_detector_target(queue)	Attempt to detect the version of CUDA present in the operating system in a

cuda_version()

Attempt to detect the version of CUDA present in the operating system.

On Windows and Linux, the CUDA library is installed by the NVIDIA driver package, and is typically found in the standard library path, rather than with the CUDA SDK (which is optional for running CUDA apps).

On macOS, the CUDA library is only installed with the CUDA SDK, and might not be in the library path.

Returns: version string (e.g., '9.2') or None if CUDA is not found.

cached_cuda_version()

A cached version of the cuda detection system.

conda_virtual_packages() → collections.abc.Iterable[*conda.plugins.types.CondaVirtualPackage*]

_cuda_driver_version_detector_target(*queue*)

Attempt to detect the version of CUDA present in the operating system in a subprocess.

On Windows and Linux, the CUDA library is installed by the NVIDIA driver package, and is typically found in the standard library path, rather than with the CUDA SDK (which is optional for running CUDA apps).

On macOS, the CUDA library is only installed with the CUDA SDK, and might not be in the library path.

Returns: version string (e.g., '9.2') or None if CUDA is not found.
The result is put in the queue rather than a return value.

freebsd

Detect whether this is FeeBSD.

Functions

conda_virtual_packages()	→ collections.abc.Iterable[conda.plugins.types.CondaVirtualPack

conda_virtual_packages() → collections.abc.Iterable[*conda.plugins.types.CondaVirtualPackage*]

linux

Detect whether this is Linux.

Functions

linux_version()	→ str | None
linux_version_validate(version)	→ str | None
conda_virtual_packages()	→ collections.abc.Iterable[conda.plugins.types.CondaVirtualPack

Attributes

LINUX_VERSION_PATTERN

LINUX_VERSION_PATTERN

linux_version() → str | None

linux_version_validate(*version: str*) → str | None

conda_virtual_packages() → collections.abc.Iterable[*conda.plugins.types.CondaVirtualPackage*]

osx

Detect whether this is macOS.

Functions

osx_version()	→ str | None
conda_virtual_packages()	→ collections.abc.Iterable[conda.plugins.types.CondaVirtualPack

osx_version() → str | None

conda_virtual_packages() → collections.abc.Iterable[*conda.plugins.types.CondaVirtualPackage*]

windows

Detect whether this is Windows.

Functions

win_version() → str | None	
conda_virtual_packages() → collections.abc.Iterable[conda.plugins.types.CondaVirtualPack	

win_version() → str | None

conda_virtual_packages() → collections.abc.Iterable[*conda.plugins.types.CondaVirtualPackage*]

plugins

The list of virtual package plugins for easier registration with pluggy.

hookimpl

Decorator to mark plugin hook implementations, used to register plugins.

reporters

Holds functions for output rendering in conda

Functions

_get_render_func([style]) → collections.abc.Callable	Retrieves the render function to use
render(data[, style]) → None	Used to render output in conda
get_progress_bar(description, **kwargs) → conda.plugins.types.ProgressBarBase	Retrieve the progress bar for the currently configured reporter backend
get_progress_bar_context_manager() → contextlib.AbstractContextManager	Retrieve progress bar context manager to use with registered reporter
get_spinner(message[, fail_message]) → conda.plugins.types.SpinnerBase	Retrieve spinner to use with registered reporter
confirm_yn([message, default, dry_run]) → bool	Display a "yes/no" confirmation input

Attributes

logger

logger

_get_render_func(*style: str | None = None*) → collections.abc.Callable

Retrieves the render function to use

render(*data*, *style: str | None = None*, ***kwargs*) → None

Used to render output in conda

The output will either be rendered as "json" or normal "console" output to stdout. This function allows us to configure different reporter backends for these two types of output.

get_progress_bar(*description: str*, ***kwargs*) → *conda.plugins.types.ProgressBarBase*

Retrieve the progress bar for the currently configured reporter backend

get_progress_bar_context_manager() → contextlib.AbstractContextManager

Retrieve progress bar context manager to use with registered reporter

get_spinner(*message: str*, *fail_message: str = 'failed\n'*) → *conda.plugins.types.SpinnerBase*

Retrieve spinner to use with registered reporter

confirm_yn(*message: str = 'Proceed'*, *default='yes'*, *dry_run=None*) → bool

Display a "yes/no" confirmation input

resolve

Low-level SAT solver wrapper/interface for the classic solver.

See conda.core.solver.Solver for the high-level API.

Classes

Resolve

Functions

_get_sat_solver_cls([sat_solver_choice])	
exactness_and_number_of_deps(resolve_obj, ms)	Sorting key to emphasize packages that have more strict

Attributes

Unsatisfiable
ResolvePackageNotFound
_sat_solvers

Unsatisfiable

ResolvePackageNotFound

_sat_solvers

_get_sat_solver_cls(*sat_solver_choice=SatSolverChoice.PYCOSAT*)

exactness_and_number_of_deps(*resolve_obj*, *ms*)

Sorting key to emphasize packages that have more strict requirements. More strict means the reduced index can be reduced more, so we want to consider these more constrained deps earlier in reducing the index.

class Resolve(*index*, *processed=False*, *channels=()*)

index

channels = ()

_channel_priorities_map

_channel_priority

_solver_ignore_timestamps

groups

trackers

_cached_find_matches

ms_depends_

_reduced_index_cache

_pool_cache

_strict_channel_cache

_system_precs

__hash__()

default_filter(*features=None*, *filter=None*)

valid(*spec_or_prec*, *filter*, *optional=True*)

Tests if a package, MatchSpec, or a list of both has satisfiable dependencies, assuming cyclic dependencies are always valid.

Parameters

- **spec_or_prec** -- a package record, a MatchSpec, or an iterable of these.
- **filter** -- a dictionary of (fkey,valid) pairs, used to consider a subset of dependencies, and to eliminate repeated searches.
- **optional** -- if True (default), do not enforce optional specifications when considering validity. If False, enforce them.

Returns

True if the full set of dependencies can be satisfied; False otherwise. If filter is supplied and update is True, it will be updated with the search results.

valid2(*spec_or_prec*, *filter_out*, *optional=True*)

invalid_chains(*spec*, *filter*, *optional=True*)

Constructs a set of 'dependency chains' for invalid specs.

A dependency chain is a tuple of MatchSpec objects, starting with the requested spec, proceeding down the dependency tree, ending at a specification that cannot be satisfied.

Parameters

- **spec** -- a package key or MatchSpec
- **filter** -- a dictionary of (prec, valid) pairs to be used when testing for package validity.

Returns

A tuple of tuples, empty if the MatchSpec is valid.

verify_specs(*specs*)

Perform a quick verification that specs and dependencies are reasonable.

Parameters
specs -- An iterable of strings or MatchSpec objects to be tested.

Returns
Nothing, but if there is a conflict, an error is thrown.

Note that this does not attempt to resolve circular dependencies.

_classify_bad_deps(*bad_deps: list[list[*conda.models.match_spec.MatchSpec*]]*, *specs_to_add*, *history_specs*, *strict_channel_priority*) → conda.exceptions.UnsatisfiableConflictMap

find_matches_with_strict(*ms*, *strict_channel_priority*)

find_conflicts(*specs*, *specs_to_add=None*, *history_specs=None*)

breadth_first_search_for_dep_graph(*root_spec*, *target_name*, *dep_graph*, *num_targets=1*)

Return shorted path from root_spec to target_name

build_graph_of_deps(*spec*)

build_conflict_map(*specs: collections.abc.Iterable[*conda.models.match_spec.MatchSpec*]*, *specs_to_add=None*, *history_specs=None*) → conda.exceptions.UnsatisfiableConflictMap

Perform a deeper analysis on conflicting specifications, by attempting to find the common dependencies that might be the cause of conflicts.

Parameters

- **specs** -- An iterable of strings or MatchSpec objects to be tested.
- **conflict.** (*It is assumed that the specs*)

Returns
Classified conflict map for *UnsatisfiableError*.

Strategy:
If we're here, we know that the specs conflict. This could be because: - One spec conflicts with another; e.g.

['numpy 1.5*', 'numpy >=1.6']

- **One spec conflicts with a dependency of another; e.g.**
  ['numpy 1.5*', 'scipy 0.12.0b1']
- **Each spec depends on *the same package* but in a different way; e.g.,**
  ['A', 'B'] where A depends on numpy 1.5, and B on numpy 1.6.

Technically, all three of these cases can be boiled down to the last one if we treat the spec itself as one of the "dependencies". There might be more complex reasons for a conflict, but this code only considers the ones above.

The purpose of this code, then, is to identify packages (like numpy above) that all of the specs depend on *but in different ways*. We then identify the dependency chains that lead to those packages.

_get_strict_channel(*package_name*)

_broader(*ms*, *specs_by_name*)

Prevent introduction of matchspecs that broaden our selection of choices.

_get_package_pool(*specs*)

get_reduced_index(*explicit_specs*, *sort_by_exactness=True*, *exit_on_conflict=False*)

match_any(*mss*, *prec*)

find_matches(*spec:* conda.models.match_spec.MatchSpec) → tuple[*conda.models.records.PackageRecord*]

ms_depends(*prec:* conda.models.records.PackageRecord) → list[*conda.models.match_spec.MatchSpec*]

version_key(*prec*, *vtype=None*)

static _make_channel_priorities(*channels: collections.abc.Iterable[*conda.models.channel.Channel | *str]*) → dict[str, int]

get_pkgs(*ms*, *emptyok=False*)

static to_sat_name(*val*)

static to_feature_metric_id(*prec_dist_str*, *feat*)

push_MatchSpec(*C*, *spec*)

gen_clauses()

generate_spec_constraints(*C*, *specs*)

generate_feature_count(*C*)

generate_update_count(*C*, *specs*)

generate_feature_metric(*C*)

generate_removal_count(*C*, *specs*)

generate_install_count(*C*, *specs*)

generate_package_count(*C*, *missing*)

generate_version_metrics(*C*, *specs*, *include0=False*)

dependency_sort(*must_have: dict[str,* conda.models.records.PackageRecord*]*) → list[*conda.models.records.PackageRecord*]

environment_is_consistent(*installed*)

get_conflicting_specs(*specs*, *explicit_specs*)

bad_installed(*installed*, *new_specs*)

restore_bad(*pkgs*, *preserve*)

install_specs(*specs*, *installed*, *update_deps=True*)

install(*specs*, *installed=None*, *update_deps=True*, *returnall=False*)

remove_specs(*specs*, *installed*)

remove(*specs*, *installed*)

solve(*specs: list*, *returnall: bool = False*, *_remove=False*, *specs_to_add=None*, *history_specs=None*, *should_retry_solve=False*) → list[*conda.models.records.PackageRecord*]

testing

cases

Extends unittest.TestCase to include select pytest fixtures.

Classes

BaseTestCase	A class whose instances are single test cases.

class BaseTestCase(*methodName='runTest'*)

Bases: unittest.TestCase

A class whose instances are single test cases.

By default, the test code itself should be placed in a method named 'runTest'.

If the fixture may be used for many test cases, create as many test methods as are needed. When instantiating such a TestCase subclass, specify in the constructor arguments the name of the test method that the instance is to execute.

Test authors should subclass TestCase for their own tests. Construction and deconstruction of the test's environment ('fixture') can be implemented by overriding the 'setUp' and 'tearDown' methods respectively.

If it is necessary to override the __init__ method, the base class __init__ method must always be called. It is important that subclasses should not change the signature of their __init__ method, since instances of the classes are instantiated automatically by parts of the framework in order to be run.

When subclassing TestCase, you can set these attributes: * failureException: determines which exception will be raised when

> the instance's assertion methods fail; test methods raising this exception will be deemed to have 'failed' rather than 'errored'.

- **longMessage: determines whether long messages (including repr of**
  objects used in assert methods) will be printed on failure in *addition* to any explicit message passed.
- **maxDiff: sets the maximum length of a diff in failure messages**
  by assert methods using difflib. It is looked up as an instance attribute so can be configured by individual tests if required.

Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

fixture_names = ('tmpdir',)

auto_injector_fixture(*request*)

fixtures

Collection of pytest fixtures used in conda tests.

Classes

CondaCLIFixture	
PipCLIFixture	Fixture for calling pip in specific conda environments.
PathFactoryFixture	
TmpEnvFixture	
TmpRepodataChannelFixture	
TmpChannelFixture	
HttpTestServerFixture	Fixture providing HTTP test server for serving local files.

Functions

suppress_resource_warning()	Suppress *Unclosed Socket Warning*
suppress_pytest_bencher_deprecation_warning(	Suppress *PendingDeprecationWarning: FileType is deprecated. Simply open files after parsing arguments.*
tmpdir(tmpdir, request)	
clear_subdir_cache()	
clear_plugin_manager_cache() → collections.abc.Iterable[None]	
reset_conda_context()	Resets the context object after each test function is run.
temp_package_cache(tmp_path_factory, monkeypatch) → collections.abc.Iterator[pathlib.Path]	Used to isolate package or index cache from other tests.
parametrized_solver_fixture(request) → collections.abc.Iterable[Literal['rattler', 'libmamba', 'classic']]	A parameterized fixture that runs each test with rattler, libmamba,
solver_classic(request) → collections.abc.Iterable[Literal['classic']]	
solver_libmamba(request) → collections.abc.Iterable[Literal['libmamba']]	
solver_rattler(request) → collections.abc.Iterable[Literal['rattler']]	
_solver_helper(request, solver) → collections.abc.Iterable[Solver]	
conda_cli(capsys) → collections.abc.Iterator[CondaCLIFixture]	A function scoped fixture returning CondaCLIFixture instance.
session_conda_cli() → collections.abc.Iterator[CondaCLIFixture]	A session scoped fixture returning CondaCLIFixture instance.
pip_cli() → collections.abc.Iterator[PipCLIFixture]	A session scoped fixture returning PipCLIFixture instance.
path_factory(tmp_path) → collections.abc.Iterator[PathFactoryFixture]	A function scoped fixture returning PathFactoryFixture instance.
tmp_env(path_factory, conda_cli) → collections.abc.Iterator[TmpEnvFixture]	A function scoped fixture returning TmpEnvFixture instance.
empty_env(tmp_env) → pathlib.Path	A function scoped fixture returning an empty environment.

continues on next page

Table 341 – continued from previous page

session_tmp_env(tmp_path_factory, session_conda_cli) → collections.abc.Iterator[TmpEnvFixture]	A session scoped fixture returning TmpEnvFixture instance.
tmp_repodata_channel(path_factory) → collections.abc.Iterator[TmpRepodataChannelFixture]	Return a function scoped factory for local metadata-only channels.
tmp_channel(path_factory, conda_cli) → collections.abc.Iterator[TmpChannelFixture]	A function scoped fixture returning TmpChannelFixture instance.
context_aware_monkeypatch(monkeypatch) → pytest.MonkeyPatch	A monkeypatch fixture that resets context after each test.
tmp_pkgs_dir(path_factory, mocker) → collections.abc.Iterator[pathlib.Path]	A function scoped fixture returning a temporary package cache directory.
tmp_envs_dir(path_factory, mocker) → collections.abc.Iterator[pathlib.Path]	A function scoped fixture returning a temporary environment directory.
PYTHONPATH()	We need to set this so Python loads the dev version of 'conda', usually taken
context_testdata() → None	
http_test_server(request, path_factory) → collections.abc.Iterator[HttpTestServerFixture]	Function-scoped HTTP test server for serving local files.

Attributes

TEST_CONDARC	
Solver	
TMP_REPODATA_CHANNEL_DEFAULTS	Defaults for synthetic records, overridable per channel or per record.

TEST_CONDARC = ''

suppress_resource_warning()

Suppress *Unclosed Socket Warning*

It seems urllib3 keeps a socket open to avoid costly recreation costs.

xref: https://github.com/kennethreitz/requests/issues/1882

suppress_pytest_bencher_deprecation_warning()

Suppress *PendingDeprecationWarning: FileType is deprecated. Simply open files after parsing arguments.*

Raised from pytest-bencher.

xref: https://github.com/ionelmc/pytest-benchmark/issues/283

tmpdir(*tmpdir*, *request*)

clear_subdir_cache()

clear_plugin_manager_cache() → collections.abc.Iterable[None]

reset_conda_context()

Resets the context object after each test function is run.

temp_package_cache(*tmp_path_factory*, *monkeypatch: pytest.MonkeyPatch*) → collections.abc.Iterator[pathlib.Path]

Used to isolate package or index cache from other tests.

parametrized_solver_fixture(*request: pytest.FixtureRequest*) → collections.abc.Iterable[Literal['rattler', 'libmamba', 'classic']]

A parameterized fixture that runs each test with rattler, libmamba, and classic. Not autouse: activate it with *pytestmark = pytest.mark.usefixtures("parametrized_solver_fixture")* in modules that need it (or CI can narrow the backends via *CONDA_TEST_SOLVERS*).

Note that skips and xfails need to be done _inside_ the test body. Decorators can't be used because they are evaluated before the fixture has done its work!

So, instead of:

@pytest.mark.skipif(context.solver == "libmamba", reason="...") def test_foo():

...

Do:

def test_foo():

if context.solver == "libmamba":
pytest.skip("...")

...

solver_classic(*request: pytest.FixtureRequest*) → collections.abc.Iterable[Literal['classic']]

solver_libmamba(*request: pytest.FixtureRequest*) → collections.abc.Iterable[Literal['libmamba']]

solver_rattler(*request: pytest.FixtureRequest*) → collections.abc.Iterable[Literal['rattler']]

Solver

_solver_helper(*request: pytest.FixtureRequest*, *solver:* Solver) → collections.abc.Iterable[*Solver*]

class CondaCLIFixture

capsys: pytest.CaptureFixture | None

__call__(**argv: conda.common.path.PathType*, *raises:* type*[BaseException]* | *tuple[*type*[BaseException], ...]*) → tuple[str, str, pytest.ExceptionInfo]

__call__(**argv: conda.common.path.PathType*) → tuple[str, str, int]

Test conda CLI. Mimic what is done in *conda.cli.main.main*.

conda ... == conda_cli(...)

Parameters

- **argv** -- Arguments to parse.
- **raises** -- Expected exception to intercept. If provided, the raised exception will be returned instead of exit code (see pytest.raises and pytest.ExceptionInfo).

Returns

Command results (stdout, stderr, exit code or pytest.ExceptionInfo).

static _cast_args(*argv: tuple[conda.common.path.PathType, ...]*) → collections.abc.Iterable[str]

Cast args to string.

conda_cli(*capsys: pytest.CaptureFixture*) → collections.abc.Iterator[*CondaCLIFixture*]

A function scoped fixture returning CondaCLIFixture instance.

Use this for any commands that are local to the current test (e.g., creating a conda environment only used in the test).

`session_conda_cli()` → collections.abc.Iterator[*CondaCLIFixture*]

A session scoped fixture returning CondaCLIFixture instance.

Use this for any commands that are global to the test session (e.g., creating a conda environment shared across tests, *conda info*, etc.).

`class PipCLIFixture`

Fixture for calling pip in specific conda environments.

`__call__`(**argv: conda.common.path.PathType*, *prefix: conda.common.path.PathType*, *raises: type[Exception] | tuple[type[Exception], ...]*) → tuple[str, str, pytest.ExceptionInfo]

`__call__`(**argv: conda.common.path.PathType*, *prefix: conda.common.path.PathType*) → tuple[str, str, int]

Test pip CLI in a specific conda environment.

pip ... in environment == *pip_cli(..., prefix=env_path)*

Parameters

- **argv** -- Arguments to pass to pip.
- **prefix** -- Path to the conda environment containing pip.
- **raises** -- Expected exception to intercept. If provided, the raised exception will be returned instead of exit code (see pytest.raises and pytest.ExceptionInfo).

Returns

Command results (stdout, stderr, exit code or pytest.ExceptionInfo).

`pip_cli()` → collections.abc.Iterator[*PipCLIFixture*]

A session scoped fixture returning PipCLIFixture instance.

Use this for calling pip commands in specific conda environments during tests. Uses *python -m pip* for reliable cross-platform execution.

Example

def test_pip_install(tmp_env, pip_cli):

with tmp_env("python=3.10", "pip") as prefix:
stdout, stderr, code = pip_cli("install", "requests", prefix=prefix) assert code == 0

`class PathFactoryFixture`

tmp_path: `pathlib.Path`

`__call__`(*name: str | None = None*, ***, *prefix: str | None = None*, *infix: str | None = None*, *suffix: str | None = None*) → pathlib.Path

Unique, non-existent path factory.

Extends pytest's *tmp_path* fixture with a new unique, non-existent path for usage in cases where we need a temporary path that doesn't exist yet.

Default behavior (no arguments):
`path_factory()` → `tmp_path/ab12cd34ef56` (12-char UUID)

Two modes of operation (mutually exclusive):

1. Name mode: Pass a complete path name. `path_factory("myfile.txt")` → `tmp_path/myfile.txt`
2. Parts mode: Pass prefix/infix/suffix; unspecified parts get UUID defaults. `path_factory(infix="!")` → `tmp_path/ab12!ef56` `path_factory(suffix=".yml")` → `tmp_path/ab12cd34.yml`

Parameters

- **name** -- Complete path name (mutually exclusive with prefix/infix/suffix)
- **prefix** -- Prefix for generated name (mutually exclusive with name param)
- **infix** -- Infix for generated name (mutually exclusive with name param)
- **suffix** -- Suffix for generated name (mutually exclusive with name param)

Returns

A new unique path

path_factory(*tmp_path: pathlib.Path*) → collections.abc.Iterator[*PathFactoryFixture*]

A function scoped fixture returning PathFactoryFixture instance.

Use this to generate any number of temporary paths for the test that are unique and do not exist yet.

class TmpEnvFixture

path_factory: *PathFactoryFixture* | pytest.TempPathFactory

conda_cli: *CondaCLIFixture*

get_path(*name: str | None = None*, *prefix: str | None = None*, *infix: str | None = None*, *suffix: str | None = None*) → pathlib.Path

__call__(**args: conda.common.path.PathType*, *prefix: str | os.PathLike | None = None*, *name: str | None = None*, *path_prefix: str | None = None*, *path_infix: str | None = None*, *path_suffix: str | None = None*, *shallow: bool | None = None*) → collections.abc.Iterator[pathlib.Path]

Generate a conda environment with the provided packages.

Path customization (mutually exclusive options):

1. Auto-generated path (default): Unique path in tmp_path. `tmp_env()` → `tmp_path/ab12cd34ef56` (12-char UUID)
2. Custom prefix: Specify exact location. `tmp_env(prefix="/path/to/env")` → `/path/to/env`
3. Name mode: Specify env name directly. `tmp_env(name="my-test-env")` → `tmp_path/my-test-env`
4. Parts mode: Customize path name generation (useful for special char testing). `tmp_env(path_infix="!")` → `tmp_path/ab12!ef56`

Parameters

- **args** -- Arguments to pass to conda create (e.g., packages, flags)
- **prefix** -- Exact prefix path (mutually exclusive with name/path_* params)
- **name** -- Env name (mutually exclusive with prefix/path_* params)
- **path_prefix** -- Prefix for path name (mutually exclusive with prefix/name params)
- **path_infix** -- Infix for path name (mutually exclusive with prefix/name params)
- **path_suffix** -- Suffix for path name (mutually exclusive with prefix/name params)
- **shallow** -- If True, create env on disk only without conda create

Returns

The conda environment's prefix

tmp_env(*path_factory:* PathFactoryFixture, *conda_cli:* CondaCLIFixture) → collections.abc.Iterator[*TmpEnvFixture*]

A function scoped fixture returning TmpEnvFixture instance.

Use this when creating a conda environment that is local to the current test.

empty_env(*tmp_env:* TmpEnvFixture) → pathlib.Path

A function scoped fixture returning an empty environment.

Use this when creating a conda environment that is empty.

session_tmp_env(*tmp_path_factory: pytest.TempPathFactory*, *session_conda_cli:* CondaCLIFixture) → collections.abc.Iterator[*TmpEnvFixture*]

A session scoped fixture returning TmpEnvFixture instance.

Use this when creating a conda environment that is shared across tests.

TMP_REPODATA_CHANNEL_DEFAULTS: dict[str, object]

Defaults for synthetic records, overridable per channel or per record.

class TmpRepodataChannelFixture

path_factory: *PathFactoryFixture*

__call__(*packages:* *collections.abc.Iterable[collections.abc.Mapping[str, Any] |* conda.models.records.PackageRecord*] = ()*, **, subdirs:* *collections.abc.Iterable[str] |* *None* = *None*, ***record_defaults: Any*) → tuple[pathlib.Path, str]

Create a local channel from in-memory package metadata.

Only `repodata.json` files are written, without package archives. Use the channel for solving and `--dry-run` transactions.

Parameters

- **packages** -- Mappings (`name` required) or `PackageRecord` objects. Record fields override channel defaults, which override `TMP_REPODATA_CHANNEL_DEFAULTS`. Records in `noarch` also default to `noarch: generic`. `fn` sets the repodata key and defaults to `{name}-{version}-{build}.tar.bz2`.
- **subdirs** -- Subdirs to write, in addition to those used by records. Defaults to `noarch` and the native subdir.
- ****record_defaults** -- Defaults applied to every record in the channel.

Returns

The channel path and its `file://` URL.

tmp_repodata_channel(*path_factory:* PathFactoryFixture) → collections.abc.Iterator[*TmpRepodataChannelFixture*]

Return a function scoped factory for local metadata-only channels.

class TmpChannelFixture

path_factory: *PathFactoryFixture*

conda_cli: *CondaCLIFixture*

__call__(**specs: str*) → collections.abc.Iterator[tuple[pathlib.Path, str]]

tmp_channel(*path_factory:* PathFactoryFixture, *conda_cli:* CondaCLIFixture) → collections.abc.Iterator[*TmpChannelFixture*]

A function scoped fixture returning TmpChannelFixture instance.

context_aware_monkeypatch(*monkeypatch: pytest.MonkeyPatch*) → pytest.MonkeyPatch

A monkeypatch fixture that resets context after each test.

tmp_pkgs_dir(*path_factory:* PathFactoryFixture, *mocker: pytest_mock.MockerFixture*) → collections.abc.Iterator[pathlib.Path]

A function scoped fixture returning a temporary package cache directory.

tmp_envs_dir(*path_factory:* PathFactoryFixture, *mocker: pytest_mock.MockerFixture*) → collections.abc.Iterator[pathlib.Path]

A function scoped fixture returning a temporary environment directory.

PYTHONPATH()

We need to set this so Python loads the dev version of 'conda', usually taken from *conda/* in the root of the cloned repo. This root is usually the working directory when we run *pytest*. Otherwise, it will import the one installed in the base environment, which might have not been overwritten with *pip install -e . --no-deps*. This doesn't happen in other tests because they run with the equivalent of *python -m conda*. However, some tests directly run *conda (shell function) which calls `conda* (Python entry point). When a script is called this way, it bypasses the automatic "working directory is first on sys.path" behavior you find in *python -m* style calls. See https://docs.python.org/3/library/sys_path_init.html for details.

context_testdata() → None

class HttpTestServerFixture

Fixture providing HTTP test server for serving local files.

server: http.server.ThreadingHTTPServer

host: str

port: int

url: str

directory: pathlib.Path

__post_init__()

Log server startup for debugging.

get_url(*path: str = ''*) → str

Get full URL for a given path on the server.

Parameters
path -- Relative path on the server (e.g., "subdir/package.tar.bz2")

Returns
Full URL

http_test_server(*request: pytest.FixtureRequest*, *path_factory:* PathFactoryFixture) → collections.abc.Iterator[*HttpTestServerFixture*]

Function-scoped HTTP test server for serving local files.

This fixture starts an HTTP server on a random port and serves files from a directory. The server supports both IPv4 and IPv6.

Usage without parametrize (dynamic content):

def test_dynamic(http_test_server):
Server uses temporary directory automatically (http_test_server.directory / "file.txt").write_text("content") url = http_test_server.get_url("file.txt") response = requests.get(url) assert response.status_code == 200

Usage with parametrize (pre-existing directory):
@pytest.mark.parametrize("http_test_server", ["tests/data/mock-channel"], indirect=True) def test_existing(http_test_server):

url = http_test_server.get_url("file.txt") response = requests.get(url) assert response.status_code == 200

Use `None` in parametrize to mix pre-existing directories with dynamic content:
@pytest.mark.parametrize("http_test_server", ["tests/data", None], indirect=True)

Parameters

- **`request`** -- pytest fixture request object
- **`path_factory`** -- path_factory fixture for creating unique temp directories

Returns
HttpTestServerFixture with server, host, port, url, and directory attributes

Raises
`ValueError` -- If parametrized directory is invalid

`gateways`

`fixtures`

Collection of pytest fixtures used in conda.gateways tests.

Functions

minio_s3_server(xprocess, tmp_path)	Mock a local S3 server using *minio*

Attributes

MINIO_EXE

`MINIO_EXE = None`

`minio_s3_server`(*xprocess*, *tmp_path*)
Mock a local S3 server using *minio*

This requires: - pytest-xprocess: runs the background process - minio: the executable must be in PATH

Note, the given S3 server will be EMPTY! The test function needs to populate it. You can use *conda.testing.helpers.populate_s3_server* for that.

helpers

Collection of helper functions used in conda tests.

Functions

strip_expected(stderr)	
raises(exception, func[, string])	
deprecated_import(module)	Assert a module's import-time deprecation warning, but only on first import.
captured([disallow_stderr])	
assert_equals(a, b[, output])	
assert_not_in(a, b[, output])	
assert_in(a, b[, output])	
add_subdir(dist_string)	
add_subdir_to_iter(iterable)	
tempdir()	
supplement_index_with_repodata(index, repodata, ...)	
add_feature_records_legacy(index)	
_export_subdir_data_to_repodata(subdir_data)	This function is only temporary and meant to patch wrong / undesirable
_sync_channel_to_disk(subdir_data)	This function is only temporary and meant to patch wrong / undesirable
*_alias_canonical_channel_name_cache_to_file_*	This function is only temporary and meant to patch wrong / undesirable
_patch_for_local_exports(name, subdir_data)	This function is only temporary and meant to patch wrong / undesirable
_get_index_r_base(json_filename_or_packages, channel_name)	
get_index_r_1([subdir, add_pip, merge_noarch])	
get_index_r_2([subdir, add_pip, merge_noarch])	
get_index_r_4([subdir, add_pip, merge_noarch])	
get_index_r_5([subdir, add_pip, merge_noarch])	
get_index_must_unfreeze([subdir, add_pip, merge_noarch])	
get_index_cuda([subdir, add_pip, merge_noarch])	
record([name, version, depends, build, build_number, ...])	
_get_solver_base(channel_id, tmpdir[, specs_to_add, ...])	
get_solver(tmpdir[, specs_to_add, specs_to_remove, ...])	
get_solver_2(tmpdir[, specs_to_add, specs_to_remove, ...])	
get_solver_4(tmpdir[, specs_to_add, specs_to_remove, ...])	
get_solver_5(tmpdir[, specs_to_add, specs_to_remove, ...])	
get_solver_aggregate_1(tmpdir[, specs_to_add, ...])	

continues on next page

Table 345 – continued from previous page

get_solver_aggregate_2(tmpdir[, specs_to_add, ...])	
get_solver_must_unfreeze(tmpdir[, specs_to_add, ...])	
get_solver_cuda(tmpdir[, specs_to_add, ...])	
convert_to_dist_str(solution)	
solver_class()	
in_subprocess()	
forward_to_subprocess(request, *cli_args, ...) → subprocess.CompletedProcess | None	

Attributes

TEST_DATA_DIR	
CHANNEL_DIR_V2	
EXPORTED_CHANNELS_DIR	
expected_error_prefix	

TEST_DATA_DIR

CHANNEL_DIR_V2 = b'.'

EXPORTED_CHANNELS_DIR = b'.'

expected_error_prefix = 'Using Anaconda Cloud api site https://api.anaconda.org'

strip_expected(*stderr*)

raises(*exception*, *func*, *string=None*)

deprecated_import(*module: str*)

Assert a module's import-time deprecation warning, but only on first import.

A module-level `deprecated.module(...)` warning fires once, on first import. If something else already imported the module first (e.g. test collection or an earlier test), asserting `pytest.deprecated_call()` again would fail. Use this instead of bare `pytest.deprecated_call()` around deprecated module imports.

captured(*disallow_stderr=True*)

assert_equals(*a*, *b*, *output=''*)

assert_not_in(*a*, *b*, *output=''*)

assert_in(*a*, *b*, *output=''*)

add_subdir(*dist_string*)

add_subdir_to_iter(*iterable*)

tempdir()

supplement_index_with_repodata(*index*, *repodata*, *channel*, *priority*)

add_feature_records_legacy(*index*)

_export_subdir_data_to_repodata(*subdir_data:* conda.core.subdir_data.SubdirData)

This function is only temporary and meant to patch wrong / undesirable testing behaviour. It should end up being replaced with the new class-based, backend-agnostic solver tests.

_sync_channel_to_disk(*subdir_data:* conda.core.subdir_data.SubdirData)

This function is only temporary and meant to patch wrong / undesirable testing behaviour. It should end up being replaced with the new class-based, backend-agnostic solver tests.

_alias_canonical_channel_name_cache_to_file_prefixed(*name*, *subdir_data=None*)

This function is only temporary and meant to patch wrong / undesirable testing behaviour. It should end up being replaced with the new class-based, backend-agnostic solver tests.

_patch_for_local_exports(*name*, *subdir_data*)

This function is only temporary and meant to patch wrong / undesirable testing behaviour. It should end up being replaced with the new class-based, backend-agnostic solver tests.

_get_index_r_base(*json_filename_or_packages*, *channel_name*, *subdir=context.subdir*, *add_pip=False*, *merge_noarch=False*)

get_index_r_1(*subdir=context.subdir*, *add_pip=True*, *merge_noarch=False*)

get_index_r_2(*subdir=context.subdir*, *add_pip=True*, *merge_noarch=False*)

get_index_r_4(*subdir=context.subdir*, *add_pip=True*, *merge_noarch=False*)

get_index_r_5(*subdir=context.subdir*, *add_pip=False*, *merge_noarch=False*)

get_index_must_unfreeze(*subdir=context.subdir*, *add_pip=True*, *merge_noarch=False*)

get_index_cuda(*subdir=context.subdir*, *add_pip=True*, *merge_noarch=False*)

record(*name='a'*, *version='1.0'*, *depends=None*, *build='0'*, *build_number=0*, *timestamp=0*, *channel=None*, ***kwargs*)

_get_solver_base(*channel_id*, *tmpdir*, *specs_to_add=()*, *specs_to_remove=()*, *prefix_records=()*, *history_specs=()*, *add_pip=False*, *merge_noarch=False*)

get_solver(*tmpdir*, *specs_to_add=()*, *specs_to_remove=()*, *prefix_records=()*, *history_specs=()*, *add_pip=False*, *merge_noarch=False*)

get_solver_2(*tmpdir*, *specs_to_add=()*, *specs_to_remove=()*, *prefix_records=()*, *history_specs=()*, *add_pip=False*, *merge_noarch=False*)

get_solver_4(*tmpdir*, *specs_to_add=()*, *specs_to_remove=()*, *prefix_records=()*, *history_specs=()*, *add_pip=False*, *merge_noarch=False*)

get_solver_5(*tmpdir*, *specs_to_add=()*, *specs_to_remove=()*, *prefix_records=()*, *history_specs=()*, *add_pip=False*, *merge_noarch=False*)

get_solver_aggregate_1(*tmpdir*, *specs_to_add=()*, *specs_to_remove=()*, *prefix_records=()*, *history_specs=()*, *add_pip=False*, *merge_noarch=False*)

get_solver_aggregate_2(*tmpdir*, *specs_to_add=()*, *specs_to_remove=()*, *prefix_records=()*, *history_specs=()*, *add_pip=False*, *merge_noarch=False*)

get_solver_must_unfreeze(*tmpdir*, *specs_to_add=()*, *specs_to_remove=()*, *prefix_records=()*, *history_specs=()*, *add_pip=False*, *merge_noarch=False*)

get_solver_cuda(*tmpdir*, *specs_to_add=()*, *specs_to_remove=()*, *prefix_records=()*, *history_specs=()*, *add_pip=False*, *merge_noarch=False*)

convert_to_dist_str(*solution*)

solver_class()

in_subprocess()

forward_to_subprocess(*request*, **cli_args*, ***subprocess_kwargs*) → subprocess.CompletedProcess | None

http_test_server

HTTP test server for serving local files during tests.

This module provides a local HTTP server that can be used in pytest fixtures to serve files from a specified directory. The server runs on a random port and supports both IPv4 and IPv6.

The server is commonly used for testing: - Mock conda channels with packages and repodata - Remote environment files (environment.yml) - Remote configuration files - Any scenario where conda needs to fetch files from HTTP URLs

Example usage:

```
from conda.testing import http_test_server

def test_something():
    server = http_test_server.run_test_server("/path/to/files")
    host, port = server.socket.getsockname()[:2]
    url = f"http://{host}:{port}/file.txt"

    # Make HTTP requests to url...

    server.shutdown()
```

For pytest fixtures that wrap this functionality, see: - *http_test_server* - function-scoped fixture

The fixture can be configured via *@pytest.mark.parametrize("http_test_server", ["<directory>"], indirect=True)* to specify the directory to serve, or used without parametrize (or with *None*) for dynamic content generation.

Functions

run_test_server(directory[, finish_request_action]) → http.server.ThreadingHTTPServer	Run a test server on a random port. Inspect returned server to get port,

run_test_server(*directory: str*, *finish_request_action: collections.abc.Callable[[], None] | None = None*) → http.server.ThreadingHTTPServer

Run a test server on a random port. Inspect returned server to get port, shutdown etc.

Parameters

- **directory** -- Directory to serve files from.
- **finish_request_action** -- Optional callable invoked before each request is handled (e.g. to inject latency or failures).

integration

These helpers were originally defined in tests/test_create.py, but were refactored here so downstream projects can benefit from them too.

Classes

Commands

Functions

escape_for_winpath(p)
package_is_installed(prefix, spec[, reload_records]) → conda.models.records.PrefixRecord | None
get_shortcut_dir([prefix_for_unix])

Attributes

TEST_LOG_LEVEL
PYTHON_BINARY
which_or_where
env_or_set

TEST_LOG_LEVEL = 10

PYTHON_BINARY = 'bin/python'

which_or_where = 'which'

env_or_set = 'env'

escape_for_winpath(*p*)

class Commands

    COMPARE = 'compare'

    CONFIG = 'config'

    CLEAN = 'clean'

    CREATE = 'create'

    INFO = 'info'

    INSTALL = 'install'

    LIST = 'list'

    REMOVE = 'remove'

SEARCH = 'search'

UPDATE = 'update'

RUN = 'run'

package_is_installed(*prefix: str | os.PathLike | pathlib.Path*, *spec: str | conda.models.match_spec.MatchSpec*, *reload_records: bool = True*) → *conda.models.records.PrefixRecord* | None

get_shortcut_dir(*prefix_for_unix=sys.prefix*)

notices

fixtures

Collection of pytest fixtures used in conda.notices tests.

Functions

notices_cache_dir(mocker, tmp_path) → collections.abc.Iterator[pathlib.Path]	Fixture that creates the notices cache dir while also mocking
notices_mock_fetch_get_session(mocker) → unittest.mock.MagicMock	
conda_notices_args_n_parser()	

notices_cache_dir(*mocker: pytest_mock.MockerFixture*, *tmp_path: pathlib.Path*) → collections.abc.Iterator[pathlib.Path]

Fixture that creates the notices cache dir while also mocking out a call to user_cache_dir.

notices_mock_fetch_get_session(*mocker: pytest_mock.MockerFixture*) → unittest.mock.MagicMock

conda_notices_args_n_parser()

helpers

Collection of helper functions used in conda.notices tests.

Classes

DummyArgs	Dummy object that sets all kwargs as object properties.
MockResponse	

Functions

get_test_notices(messages[, level, created_at, expired_at]) → dict	
add_resp_to_mock(mock_session, status_code, messages_json) → None	Adds any number of MockResponse to MagicMock object as side_effects
create_notice_cache_files(cache_dir, cache_files, ...) → None	Creates the cache files that we use in tests

continues on next page

Table 353 – continued from previous page

offset_cache_file_mtime(mtime_offset) → None	Allows for offsetting the mtime of the notices cache file. This is often
notices_decorator_assert_message_in_stdout(c ...)	Tests a run of notices decorator where we expect to see the messages
get_notice_cache_filenames(ctx) → tuple[str]	Returns the filenames of the cache files that will be searched for

Attributes

DEFAULT_NOTICE_MESG

DEFAULT_NOTICE_MESG = 'Here is an example message that will be displayed to users'

get_test_notices(*messages: collections.abc.Sequence[str]*, *level: str | None = 'info'*, *created_at: datetime.datetime | None = None*, *expired_at: datetime.datetime | None = None*) → dict

add_resp_to_mock(*mock_session: unittest.mock.MagicMock*, *status_code: int*, *messages_json: dict*, *raise_exc: bool = False*, *count: int = 1*) → None

Adds any number of MockResponse to MagicMock object as side_effects

create_notice_cache_files(*cache_dir: pathlib.Path*, *cache_files: collections.abc.Sequence[str]*, *messages_json_seq: collections.abc.Sequence[dict]*) → None

Creates the cache files that we use in tests

offset_cache_file_mtime(*mtime_offset*) → None

Allows for offsetting the mtime of the notices cache file. This is often used to mock an older creation time the cache file.

class DummyArgs(***kwargs*)

Dummy object that sets all kwargs as object properties.

no_ansi_colors = True

notices_decorator_assert_message_in_stdout(*captured*, *messages: collections.abc.Sequence[str]*, *dummy_mesg: str | None = None*, *not_in: bool = False*)

Tests a run of notices decorator where we expect to see the messages print to stdout.

class MockResponse(*status_code*, *json_data*, *raise_exc=False*)

status_code

json_data

raise_exc = False

json()

get_notice_cache_filenames(*ctx: conda.base.context.Context*) → tuple[str]

Returns the filenames of the cache files that will be searched for

solver_helpers

Helpers for testing the solver.

Classes

SimpleEnvironment	Helper environment object.
SolverTests	Tests for *conda.core.solve.Solver* implementations.

Functions

index_packages(num)	Get the index data of the `helpers.get_index_r_*` helpers.
package_string(record)	
package_string_set(packages)	Transforms package container in package string set.
package_dict(packages)	Transforms package container into a dictionary.
empty_prefix()	
temp_simple_env([solver_class]) → SimpleEnvironment	

index_packages(*num*)

Get the index data of the `helpers.get_index_r_*` helpers.

package_string(*record*)

package_string_set(*packages*)

Transforms package container in package string set.

package_dict(*packages*)

Transforms package container into a dictionary.

class SimpleEnvironment(*path*, *solver_class*, *subdirs=context.subdirs*)

Helper environment object.

REPO_DATA_KEYS = ('build', 'build_number', 'depends', 'license', 'md5', 'name', 'sha256', 'size', 'subdir',...

_path

_prefix_path

_channels_path

_solver_class

subdirs

installed_packages = []

repo_packages: list[str] | dict[str, list[str]] = []

solver(*add*, *remove*)

Writes `repo_packages` to the disk and creates a solver instance.

solver_transaction(*add=()*, *remove=()*, *as_specs=False*)

install(**specs*, *as_specs=False*)

remove(**specs*, *as_specs=False*)

property _channel_packages

Helper that unfolds the `repo_packages` into a dictionary.

_package_data(*record*)

Turn record into data, to be written in the JSON environment/repo files.

_write_installed_packages()

_write_repo_packages(*channel_name*, *packages*)

Write packages to the channel path.

empty_prefix()

temp_simple_env(*solver_class=Solver*) → *SimpleEnvironment*

class SolverTests

Tests for *conda.core.solve.Solver* implementations.

property solver_class: *type*[*conda.core.solve.Solver*]

Abstractmethod

Class under test.

property tests_to_skip

skip_tests(*request*)

env()

find_package_in_list(*packages*, ***kwargs*)

find_package(***kwargs*)

assert_unsatisfiable(*exc_info*, *entries*)

Helper to assert that a *conda.exceptions.UnsatisfiableError* instance as a the specified set of unsatisfiable specifications.

test_empty(*env*)

test_iopro_mkl(*env*)

test_iopro_nomkl(*env*)

test_mkl(*env*)

test_accelerate(*env*)

test_scipy_mkl(*env*)

test_anaconda_nomkl(*env*)

test_pseudo_boolean(*env*)

test_unsat_from_r1(*env*)

test_unsat_simple(*env*)

test_get_dists(*env*)

test_unsat_shortest_chain_1(*env*)

test_unsat_shortest_chain_2(*env*)

test_unsat_shortest_chain_3(*env*)

test_unsat_shortest_chain_4(*env*)

test_unsat_chain(*env*)

test_unsat_any_two_not_three(*env*)

test_unsat_expand_single(*env*)

test_unsat_missing_dep(*env*)

test_nonexistent(*env*)

test_timestamps_and_deps(*env: SimpleEnvironment*)

test_nonexistent_deps(*env*)

test_install_package_with_feature(*env*)

test_unintentional_feature_downgrade(*env*)

test_circular_dependencies(*env*)

test_irrational_version(*env*)

test_no_features(*env*)

test_channel_priority_1(*monkeypatch*, *env*)

test_unsat_channel_priority(*monkeypatch*, *env*)

test_remove(*env*)

test_surplus_features_1(*env*)

test_surplus_features_2(*env*)

test_get_reduced_index_broadening_with_unsatisfiable_early_dep(*env*)

test_get_reduced_index_broadening_preferred_solution(*env*)

test_arch_preferred_over_noarch_when_otherwise_equal(*env*)

test_noarch_preferred_over_arch_when_version_greater(*env*)

test_noarch_preferred_over_arch_when_version_greater_dep(*env*)

test_noarch_preferred_over_arch_when_build_greater(*env*)

test_noarch_preferred_over_arch_when_build_greater_dep(*env*)

Functions

conda_move_to_front_of_PATH()

conda_move_to_front_of_PATH()

utils

Utility functions.

Functions

human_bytes(n)	Return the number of bytes n in more human readable form.
sys_prefix_unfollowed()	Since conda is installed into non-root environments as a symlink only
quote_for_shell(*arguments)	Properly quote arguments for command line passing.
massage_arguments(arguments[, errors])	
wrap_subprocess_call(root_prefix, prefix, dev_mode, ...)	
get_comspec()	Returns COMSPEC from envvars.
ensure_dir_exists(func)	Ensures that the directory exists for functions returning

Attributes

_RE_UNSAFE

human_bytes(*n*)

Return the number of bytes n in more human readable form.

Note: Uses SI prefixes (KB, MB, GB) instead of binary prefixes (KiB, MiB, GiB).

Examples

```
>>> human_bytes(42)
'42 B'
>>> human_bytes(1042)
'1 KB'
>>> human_bytes(10004242)
'9.5 MB'
>>> human_bytes(100000004242)
'93.13 GB'
```

sys_prefix_unfollowed()

Since conda is installed into non-root environments as a symlink only and because sys.prefix follows symlinks, this function can be used to get the 'unfollowed' sys.prefix.

This value is usually the same as the prefix of the environment into which conda has been symlinked. An example of when this is necessary is when conda looks for external sub-commands in find_commands.py

quote_for_shell(**arguments*)

Properly quote arguments for command line passing.

For POSIX uses *shlex.join*, for Windows uses a custom implementation to properly escape metacharacters.

Parameters
arguments -- Arguments to quote.

Returns
Quoted arguments.

_RE_UNSAFE

massage_arguments(*arguments*, *errors='assert'*)

wrap_subprocess_call(*root_prefix*, *prefix*, *dev_mode*, *debug_wrapper_scripts*, *arguments: collections.abc.Sequence[str]*)

get_comspec()

Returns COMSPEC from envvars.

Ensures COMSPEC envvar is set to cmd.exe, if not attempt to find it.

Raises
KeyError -- COMSPEC is undefined and cannot be found.

Returns
COMSPEC value.

ensure_dir_exists(*func*)

Ensures that the directory exists for functions returning a Path object containing a directory

Functions

conda_signal_handler(signum, frame)	

Attributes

__version__	
__name__	
__author__	
__email__	
__license__	
__copyright__	
__summary__	
__url__	
CONDA_PACKAGE_ROOT	The conda package directory.

__version__

__name__ = 'conda'

__author__ = 'Anaconda, Inc.'

__email__ = 'conda@continuum.io'

__license__ = 'BSD-3-Clause'

__copyright__ = 'Copyright (c) 2012, Anaconda, Inc.'

__summary__ = 'OS-agnostic, system-level binary package manager.'

__url__ = 'https://github.com/conda/conda'

CONDA_PACKAGE_ROOT = b'.'

The conda package directory.

exception CondaError(*message: str | None*, **args: Any*, *caused_by: Any = None*, *guidance:* _private.exception_guidance.ErrorGuidance | *_private.exception_guidance.ErrorGuidanceTypedDict | None = None*, ***kwargs*)

Bases: Exception

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

return_code: int = 1

reportable: bool = False

_kwargs: dict[str, Any] | None = None

_caused_by: Any = None

_guidance: *_private.exception_guidance.ErrorGuidance* | None = None

message

property guidance: *_private.exception_guidance.ErrorGuidance* | None

Structured guidance for this exception, if provided.

Returns an ErrorGuidance instance or None. The result is always the canonical type after dict coercion; callers do not need to handle raw dicts.

__repr__() → str

Return repr(self).

__str__() → str

Return str(self).

dump_map() → dict[str, Any]

exception CondaMultiError(*errors: collections.abc.Iterable[*CondaError*]*)

Bases: *CondaError*

Container for multiple conda errors raised together.

This is a control-flow artifact, not a user-facing error type. It must not carry its own guidance; printers apply guidance and conda_error_hints to each nested leaf error instead.

Initialize self. See help(type(self)) for accurate signature.

errors

__repr__() → str

Return repr(self).

__str__() → str

Return str(self).

dump_map() → dict[str, str | tuple[str, ...]]

contains(*exception_class: BaseException | tuple[BaseException]*) → bool

exception CondaExitZero(*message: str | None*, **args: Any*, *caused_by: Any = None*, *guidance:* _private.exception_guidance.ErrorGuidance | *_private.exception_guidance.ErrorGuidanceTypedDict | None = None*, ***kwargs*)

Bases: *CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

return_code = 0

conda_signal_handler(*signum: int*, *frame: Any*)

PYTHON MODULE INDEX

C

conda, 553
conda.__main__, 553
conda._preview, 553
conda._preview.env_setup, 553
conda._preview.env_setup.cli, 554
conda._preview.env_setup.cli.main_create, 554
conda._preview.env_setup.cli.main_install, 554
conda._private, 554
conda._private.exception_guidance, 554
conda._private.extract, 556
conda._private.shards, 556
conda._private.shards.cache, 556
conda._private.shards.decompression, 558
conda._private.shards.misc, 558
conda._private.shards.shards, 561
conda._private.shards.subset, 566
conda._private.shards.typing, 572
conda._private.zstd, 577
conda._version, 577
conda.activate, 577
conda.api, 585
conda.auxlib, 591
conda.auxlib.collection, 591
conda.auxlib.compat, 592
conda.auxlib.decorators, 593
conda.auxlib.entity, 595
conda.auxlib.exceptions, 607
conda.auxlib.ish, 607
conda.auxlib.logz, 608
conda.auxlib.type_coercion, 609
conda.base, 611
conda.base.constants, 611
conda.base.context, 619
conda.cli, 629
conda.cli.actions, 629
conda.cli.common, 631
conda.cli.conda_argparse, 632
conda.cli.condarc, 636
conda.cli.find_commands, 641
conda.cli.formats, 641
conda.cli.helpers, 642
conda.cli.install, 645
conda.cli.main, 648
conda.cli.main_clean, 648
conda.cli.main_commands, 649
conda.cli.main_compare, 649
conda.cli.main_config, 650
conda.cli.main_create, 651
conda.cli.main_env, 652
conda.cli.main_env_config, 652
conda.cli.main_env_create, 652
conda.cli.main_env_list, 653
conda.cli.main_env_remove, 653
conda.cli.main_env_update, 653
conda.cli.main_env_vars, 654
conda.cli.main_export, 654
conda.cli.main_info, 655
conda.cli.main_init, 657
conda.cli.main_install, 658
conda.cli.main_list, 658
conda.cli.main_mock_activate, 659
conda.cli.main_mock_deactivate, 659
conda.cli.main_notices, 659
conda.cli.main_package, 660
conda.cli.main_pip, 661
conda.cli.main_remove, 661
conda.cli.main_rename, 661
conda.cli.main_run, 662
conda.cli.main_search, 662
conda.cli.main_update, 663
conda.common, 663
conda.common._logic, 663
conda.common._os, 667
conda.common._os.linux, 667
conda.common._os.osx, 667
conda.common._os.unix, 668
conda.common._os.windows, 668
conda.common.compat, 670
conda.common.configuration, 671
conda.common.constants, 685
conda.common.datetime, 686
conda.common.io, 686

conda.common.iterators, 691
conda.common.logic, 692
conda.common.path, 694
conda.common.path._cygpath, 695
conda.common.path.directories, 696
conda.common.path.python, 696
conda.common.path.windows, 697
conda.common.serialize, 700
conda.common.serialize.json, 700
conda.common.serialize.yaml, 702
conda.common.signals, 704
conda.common.terminal, 704
conda.common.toposort, 706
conda.common.url, 706
conda.core, 711
conda.core.envs_manager, 711
conda.core.exclude_newer, 713
conda.core.index, 715
conda.core.initialize, 719
conda.core.launchers, 724
conda.core.link, 725
conda.core.package_cache_data, 729
conda.core.path_actions, 732
conda.core.portability, 751
conda.core.prefix_data, 755
conda.core.solve, 759
conda.core.subdir_data, 765
conda.deprecations, 770
conda.env, 773
conda.env.env, 773
conda.env.installers, 776
conda.env.installers.base, 776
conda.env.installers.conda, 777
conda.env.installers.pip, 778
conda.env.pip_util, 779
conda.env.specs, 779
conda.env.specs.cep_24, 779
conda.env.specs.explicit, 780
conda.env.specs.requirements, 781
conda.env.specs.yaml_file, 781
conda.exception_handler, 783
conda.exceptions, 784
conda.exports, 798
conda.gateways, 800
conda.gateways.anaconda_client, 800
conda.gateways.connection, 801
conda.gateways.connection.adapters, 801
conda.gateways.connection.adapters.ftp, 801
conda.gateways.connection.adapters.http, 802
conda.gateways.connection.adapters.localfs, 803
conda.gateways.connection.adapters.s3, 804
conda.gateways.connection.download, 805
conda.gateways.connection.session, 806
conda.gateways.disk, 809
conda.gateways.disk.create, 809
conda.gateways.disk.delete, 811
conda.gateways.disk.link, 812
conda.gateways.disk.lock, 812
conda.gateways.disk.permissions, 813
conda.gateways.disk.read, 813
conda.gateways.disk.test, 815
conda.gateways.disk.update, 815
conda.gateways.logging, 816
conda.gateways.repodata, 818
conda.gateways.repodata.lock, 818
conda.gateways.repodata.zstd, 818
conda.gateways.shards, 830
conda.gateways.shards.typing, 830
conda.gateways.streams, 835
conda.gateways.subprocess, 836
conda.history, 836
conda.instructions, 838
conda.misc, 840
conda.models, 841
conda.models.build_string, 841
conda.models.channel, 843
conda.models.dist, 848
conda.models.enums, 850
conda.models.environment, 856
conda.models.match_spec, 860
conda.models.package_info, 869
conda.models.prefix_graph, 870
conda.models.records, 872
conda.models.version, 883
conda.notices, 889
conda.notices.cache, 889
conda.notices.core, 890
conda.notices.fetch, 892
conda.notices.types, 893
conda.notices.views, 894
conda.plugins, 895
conda.plugins.config, 895
conda.plugins.environment_exporters, 896
conda.plugins.environment_exporters.environment_yml, 896
conda.plugins.environment_exporters.explicit, 897
conda.plugins.environment_exporters.requirements_txt, 898
conda.plugins.environment_specifiers, 898
conda.plugins.environment_specifiers.cep_24, 898
conda.plugins.environment_specifiers.environment_yml, 899
conda.plugins.environment_specifiers.explicit, 899

conda.plugins.environment_specifiers.requirements_txt, 899
conda.plugins.hookspec, 899
conda.plugins.manager, 913
conda.plugins.package_extractors, 923
conda.plugins.package_extractors.conda, 923
conda.plugins.post_solves, 923
conda.plugins.prefix_data_loaders, 923
conda.plugins.prefix_data_loaders.pypi, 923
conda.plugins.prefix_data_loaders.pypi.pkg_format, 923
conda.plugins.previews, 935
conda.plugins.reporter_backends, 935
conda.plugins.reporter_backends.console, 935
conda.plugins.reporter_backends.json, 937
conda.plugins.solvers, 939
conda.plugins.subcommands, 939
conda.plugins.subcommands.doctor, 939
conda.plugins.subcommands.doctor.health_checks, 939
conda.plugins.subcommands.doctor.health_checks.altered_files, 939
conda.plugins.subcommands.doctor.health_checks.consistency, 940
conda.plugins.subcommands.doctor.health_checks.environment_txt, 940
conda.plugins.subcommands.doctor.health_checks.file_locking, 941
conda.plugins.subcommands.doctor.health_checks.missing_files, 941
conda.plugins.subcommands.doctor.health_checks.pinned, 942
conda.plugins.subcommands.doctor.health_checks.requests_ca_bundle, 943
conda.plugins.subcommands.doctor.health_checks.utils, 943
conda.plugins.subcommands.plugins, 949
conda.plugins.subcommands.plugins.info, 949
conda.plugins.subcommands.plugins.install, 950
conda.plugins.subcommands.plugins.list, 950
conda.plugins.subcommands.plugins.package_validation, 951
conda.plugins.subcommands.plugins.remove, 951
conda.plugins.subcommands.plugins.update, 952
conda.plugins.types, 953
conda.plugins.virtual_packages, 967
conda.plugins.virtual_packages.archspec, 967
conda.plugins.virtual_packages.conda, 968
conda.plugins.virtual_packages.cuda, 968
conda.plugins.virtual_packages.freebsd, 969
conda.plugins.virtual_packages.linux, 969
conda.plugins.virtual_packages.osx, 969
conda.plugins.virtual_packages.windows, 970
conda.reporters, 970
conda.resolve, 971
conda.testing, 975
conda.testing.cases, 975
conda.testing.fixtures, 976
conda.testing.gateways, 983
conda.testing.gateways.fixtures, 983
conda.testing.helpers, 984
conda.testing.http_test_server, 987
conda.testing.integration, 988
conda.testing.notices, 989
conda.testing.notices.fixtures, 989
conda.testing.notices.helpers, 989
conda.testing.solver_helpers, 991
conda.utils, 994

INDEX

Symbols

_Activator (*class in conda.activate*), 578
_BRACKETS_KV_RE (*in module conda.models.match_spec*), 861
_BRACKETS_KV_RE_V3 (*in module conda.models.match_spec*), 861
_BRACKETS_RE (*in module conda.models.match_spec*), 861
_BRACKETS_RE_V3 (*in module conda.models.match_spec*), 861
_CEP26_MAX_LEN (*in module conda.models.build_string*), 842
_CEP26_NAME_RE (*in module conda.models.match_spec*), 861
_CEP26_RE (*in module conda.models.build_string*), 842
_CEP26_VIRTUAL_NAME_RE (*in module conda.models.match_spec*), 861
_COMPACT_DATE_ONLY_RE (*in module conda.common.datetime*), 686
_COMPACT_DURATION_RE (*in module conda.common.datetime*), 686
_ChannelCutoff (*class in conda.core.exclude_newer*), 713
_ChannelSelector (*class in conda.core.exclude_newer*), 713
_ClauseArray (*class in conda.common._logic*), 664
_ClauseList (*class in conda.common._logic*), 664
_ConstantDeprecationRegistry (*class in conda.deprecations*), 773
_CutoffParser (*class in conda.core.exclude_newer*), 713
_DATE_ONLY_RE (*in module conda.common.datetime*), 686
_FORMATTER (*in module conda.common.io*), 688
_FeaturesField (*class in conda.models.records*), 875
_GreedySubParsersAction (*class in conda.cli.conda_argparse*), 634
_HookImplWrapper (*class in conda.plugins.manager*), 914
_ISO8601_DURATION_RE (*in module conda.common.datetime*), 686
_LAZY_CONCURRENCY (*in module conda.common.io*), 691
_LIST_EMPTY_ITEM_RE (*in module conda.models.match_spec*), 861
_LIST_NAME_RE (*in module conda.models.match_spec*), 866
_LazyParserMap (*class in conda.cli.conda_argparse*), 634
_LazySubParsersAction (*class in conda.cli.conda_argparse*), 635
_MATCHER_CACHE (*MatchSpec attribute*), 863
_MATCH_SPEC_YAML (*in module conda.models.match_spec*), 866
_MENU_RE (*in module conda.core.path_actions*), 733
_MSYS2_CYGWIN_PREFIX_RE (*in module conda.activate*), 578
_MissingSentinel (*class in conda.cli.condarc*), 637
_NAME_VERSION_RE (*in module conda.models.match_spec*), 861
_NUMBER_RE (*in module conda.models.build_string*), 842
_PARENS_RE (*in module conda.models.match_spec*), 861
_PARSE_CACHE (*in module conda.models.match_spec*), 865
_PLAIN_LIST_ITEM_RE (*in module conda.models.match_spec*), 866
_PLAIN_SCALAR_ITEM_RE (*in module conda.models.match_spec*), 866
_PLUGIN_FREE_BUILTIN_COMMANDS (*in module conda.cli.conda_argparse*), 633
_PaddingError, 752
_PyCryptoSatSolver (*class in conda.common._logic*), 665
_PySatSolver (*class in conda.common._logic*), 666
_PycoSatSolver (*class in conda.common._logic*), 665
_RE_CUSTOM_EXPANDVARS (*in module conda.common.configuration*), 683
_RE_UNSAFE (*in module conda.utils*), 995
_SSLContextAdapterMixin (*class in conda.gateways.connection.adapters.http*), 803
_SatSolver (*class in conda.common._logic*), 665
_ShardDictBase (*in module conda._private.shards.typing*), 572
_SplitUrlParts (*class in conda.common.url*), 710

_StrMatchMixin (*class in conda.models.match_spec*), 867
_T (*in module conda._private.shards.misc*), 559
_T (*in module conda._private.shards.subset*), 570
_TOKEN_URL_PATTERN (*in module conda.gateways.streams*), 835
_UNSET (*in module conda.deprecations*), 770
_URLJOIN_SAFE_SCHEMES (*in module conda._private.shards.misc*), 559
_VERBOSITY_LEVELS (*in module conda.gateways.logging*), 817
_VERSION_BUILD_RE (*in module conda.models.match_spec*), 861
_VERSION_REGEX (*in module conda.common.path.python*), 697
_ValidatePackages (*class in conda.cli.helpers*), 644
_YAML_SCALAR_WORDS (*in module conda.models.match_spec*), 866
__author__ (*in module conda*), 995
__author__ (*in module conda.auxlib*), 610
__bool__() (*Channel method*), 826, 846
__bool__() (*_MissingSentinel method*), 637
__call__() (*BuildRepodataSubset method*), 831, 834
__call__() (*ChannelPriorityMeta method*), 617
__call__() (*CondaCLIFixture method*), 978
__call__() (*CondaHttpAuth method*), 809
__call__() (*CondaSessionType method*), 808
__call__() (*ConfigurationType method*), 683
__call__() (*ContextDecorator method*), 688
__call__() (*DeprecationHandler method*), 771
__call__() (*DistType method*), 848
__call__() (*ExceptionHandler method*), 783
__call__() (*ExtendConstAction method*), 630
__call__() (*LazyChoicesAction method*), 644
__call__() (*MatchSpecType method*), 861
__call__() (*NullCountAction method*), 630
__call__() (*PackageCacheType method*), 729
__call__() (*PathFactoryFixture method*), 979
__call__() (*PipCLIFixture method*), 979
__call__() (*PrefixDataType method*), 755
__call__() (*SingleStrArgCachingType method*), 884
__call__() (*SubdirDataType method*), 765
__call__() (*TmpChannelFixture method*), 981
__call__() (*TmpEnvFixture method*), 980
__call__() (*TmpRepodataChannelFixture method*), 981
__call__() (*_ConstantDeprecationRegistry method*), 773
__call__() (*_GreedySubParsersAction method*), 634
__call__() (*_LazySubParsersAction method*), 636
__call__() (*_ValidatePackages method*), 644
__call__() (*time_recorder method*), 691
__cause__ (*ValidationError attribute*), 607, 932
__commit_id__ (*in module conda._version*), 577
__contains__() (*Dist method*), 850
__contains__() (*Index method*), 718
__contains__() (*MatchSpec method*), 864
__contains__() (*RepodataState method*), 828
__contains__() (*ShardBase method*), 563, 574
__contains__() (*Shards method*), 831
__contains__() (*UrlsData method*), 731
__contains__() (*_LazyParserMap method*), 635
__copy__() (*Index method*), 718
__copyright__ (*in module conda*), 996
__copyright__ (*in module conda.auxlib*), 610
__del__() (*TemporaryDirectory method*), 810
__delattr__() (*ImmutableEntity method*), 606
__delete__() (*Field method*), 601
__dict__ (*AttrDict attribute*), 592
__doc__ (*classproperty attribute*), 595
__dump_fields() (*Entity class method*), 606
__email__ (*in module conda*), 995
__email__ (*in module conda.auxlib*), 610
__enter__() (*ConfigurationFile method*), 639
__enter__() (*History method*), 837
__enter__() (*JSONSpinner method*), 938
__enter__() (*QueueCache method*), 570
__enter__() (*QuietSpinner method*), 937
__enter__() (*ShardCache method*), 557
__enter__() (*Spinner method*), 936
__enter__() (*SpinnerBase method*), 960
__enter__() (*SwallowBrokenPipe method*), 688
__enter__() (*TemporaryDirectory method*), 810
__enter__() (*TmpDownload method*), 806
__enter__() (*TryRepodata method*), 646
__enter__() (*time_recorder method*), 691
__eq__() (*BaseSpec method*), 887
__eq__() (*Channel method*), 825, 845
__eq__() (*Dist method*), 850
__eq__() (*Entity method*), 606
__eq__() (*FeatureMatch method*), 868
__eq__() (*ListOfStrMatch method*), 868
__eq__() (*LoadedParameter method*), 676
__eq__() (*MatchSpec method*), 864
__eq__() (*PackageRecord method*), 879
__eq__() (*PrefixData method*), 756, 945
__eq__() (*PrimitiveLoadedParameter method*), 677
__eq__() (*ProgressiveFetchExtract method*), 731
__eq__() (*SplitStrMatch method*), 868
__eq__() (*VersionOrder method*), 886
__eq__() (*_MissingSentinel method*), 637
__eq__() (*_StrMatchMixin method*), 867
__exit__() (*ConfigurationFile method*), 639
__exit__() (*History method*), 837
__exit__() (*JSONSpinner method*), 938
__exit__() (*QueueCache method*), 570
__exit__() (*QuietSpinner method*), 937
__exit__() (*ShardCache method*), 557

__exit__() (*Spinner method*), 936
__exit__() (*SpinnerBase method*), 960
__exit__() (*SwallowBrokenPipe method*), 688
__exit__() (*TemporaryDirectory method*), 810
__exit__() (*TmpDownload method*), 806
__exit__() (*TryRepodata method*), 646
__exit__() (*time_recorder method*), 691
__fields__ (*Entity attribute*), 606
__ge__() (*BuildString method*), 843
__ge__() (*Dist method*), 849
__ge__() (*PackageRecord method*), 879
__ge__() (*VersionOrder method*), 886
__get (*classproperty attribute*), 595
__get__() (*ChannelField method*), 876
__get__() (*Field method*), 601
__get__() (*FilenameField method*), 876
__get__() (*IndexedTimestampField method*), 875
__get__() (*Md5Field method*), 881
__get__() (*PackageTypeField method*), 877
__get__() (*ParameterLoader method*), 683
__get__() (*SubdirField method*), 876
__get__() (*TimestampField method*), 875
__get__() (*classproperty method*), 595
__getattr__() (*StdStreamHandler method*), 817
__getattr__() (*_HookImplWrapper method*), 914
__getattr__() (*in module conda.common.io*), 691
__getattr__() (*in module conda.common.serialize.yaml*), 703
__getitem__() (*Index method*), 718
__getitem__() (*PackageRecordList method*), 766
__getitem__() (*PrefixRecordDict method*), 755
__getitem__() (*RepodataState method*), 828
__getitem__() (*_LazyParserMap method*), 635
__gt__() (*BuildString method*), 843
__gt__() (*Dist method*), 849
__gt__() (*PackageRecord method*), 879
__gt__() (*VersionOrder method*), 886
__hash__ (*BuildString attribute*), 843
__hash__() (*BaseSpec method*), 887
__hash__() (*Channel method*), 826, 845
__hash__() (*Dist method*), 849
__hash__() (*Entity method*), 606
__hash__() (*FeatureMatch method*), 868
__hash__() (*ListOfStrMatch method*), 868
__hash__() (*LoadedParameter method*), 676
__hash__() (*MatchSpec method*), 864
__hash__() (*NodeId method*), 567
__hash__() (*PackageRecord method*), 879
__hash__() (*PrimitiveLoadedParameter method*), 677
__hash__() (*ProgressiveFetchExtract method*), 731
__hash__() (*Resolve method*), 972
__hash__() (*SplitStrMatch method*), 868
__hash__() (*_MissingSentinel method*), 637
__hash__() (*_StrMatchMixin method*), 867
__important_split_value (*EnvRawParameter property*), 675
__init_subclass__() (*BaseSolver class method*), 760
__int__() (*LinkType method*), 854
__is_writable (*PackageCacheData attribute*), 730, 822
__is_writable (*PrefixData attribute*), 756, 945
__iter__() (*PrefixActionGroup method*), 727
__iter__() (*Repodatas method*), 646
__iter__() (*UrlsData method*), 731
__iter__() (*_LazyParserMap method*), 635
__json__() (*Arch method*), 851
__json__() (*Channel method*), 826, 846
__json__() (*ErrorGuidance method*), 555
__json__() (*LinkType method*), 854
__json__() (*MatchSpec method*), 864
__json__() (*PathEnum method*), 854
__json__() (*Platform method*), 852
__key__() (*Dist method*), 849
__le__() (*BuildString method*), 843
__le__() (*Dist method*), 849
__le__() (*PackageRecord method*), 879
__le__() (*VersionOrder method*), 886
__len__() (*_LazyParserMap method*), 635
__license__ (*in module conda*), 995
__license__ (*in module conda.auxlib*), 610
__lt__() (*BuildString method*), 843
__lt__() (*Dist method*), 849
__lt__() (*PackageRecord method*), 879
__lt__() (*VersionOrder method*), 886
__module__ (*classproperty attribute*), 595
__name__ (*ChannelPriority attribute*), 618
__name__ (*classproperty attribute*), 595
__name__ (*in module conda*), 995
__ne__() (*BaseSpec method*), 887
__ne__() (*Dist method*), 850
__ne__() (*VersionOrder method*), 886
__nonzero__() (*Channel method*), 826, 846
__package_cache_records (*PackageCacheData attribute*), 730, 822
__post_init__() (*CondaEnvironmentExporter method*), 965
__post_init__() (*CondaEnvironmentSpecifier method*), 964
__post_init__() (*CondaPlugin method*), 955
__post_init__() (*Environment method*), 859
__post_init__() (*ErrorGuidance method*), 555
__post_init__() (*HttpTestServerFixture method*), 982
__prefix_records (*PrefixData attribute*), 756, 945
__register__() (*Entity class method*), 606
__repr__() (*Action method*), 734
__repr__() (*BaseSpec method*), 888
__repr__() (*BuildNumberMatch method*), 888
__repr__() (*Channel method*), 825, 845
__repr__() (*ChannelMatch method*), 869

__repr__() (*ClobberError method*), 786
__repr__() (*CondaError method*), 821, 996
__repr__() (*CondaMultiError method*), 996
__repr__() (*Entity method*), 606
__repr__() (*FeatureMatch method*), 868
__repr__() (*Index method*), 717
__repr__() (*ListOfStrMatch method*), 868
__repr__() (*MatchSpec method*), 864
__repr__() (*PackageCacheData method*), 730, 823
__repr__() (*RawParameter method*), 675
__repr__() (*ReducedIndex method*), 719
__repr__() (*ShardLike method*), 564, 575
__repr__() (*SplitStrMatch method*), 868
__repr__() (*TemporaryDirectory method*), 810
__repr__() (*VersionOrder method*), 886
__repr__() (*_MissingSentinel method*), 637
__repr__() (*_StrMatchMixin method*), 867
__set (*classproperty attribute*), 595
__set__() (*Field method*), 601
__set__() (*classproperty method*), 595
__setattr__() (*ImmutableEntity method*), 606
__setitem__() (*RepodataState method*), 828
__slots__ (*BuildString attribute*), 842
__slots__ (*ExactStrMatch attribute*), 867
__slots__ (*FeatureMatch attribute*), 868
__slots__ (*GlobStrMatch attribute*), 867
__slots__ (*ListOfStrMatch attribute*), 868
__slots__ (*SplitStrMatch attribute*), 867
__str__ (*ListOfStrMatch attribute*), 868
__str__() (*BaseSpec method*), 888
__str__() (*BuildNumberMatch method*), 888
__str__() (*CacheUrlAction method*), 750
__str__() (*Channel method*), 825, 845
__str__() (*ChannelMatch method*), 869
__str__() (*CondaError method*), 821, 996
__str__() (*CondaMultiError method*), 996
__str__() (*DepsModifier method*), 616
__str__() (*Dist method*), 849
__str__() (*ExtractPackageAction method*), 751
__str__() (*FeatureMatch method*), 868
__str__() (*FileMode method*), 853
__str__() (*LinkType method*), 854
__str__() (*MatchSpec method*), 864
__str__() (*PackageRecord method*), 880
__str__() (*ParameterFlag method*), 674
__str__() (*PathConflict method*), 616
__str__() (*PathEnum method*), 854
__str__() (*SafetyChecks method*), 615
__str__() (*SplitStrMatch method*), 868
__str__() (*UpdateModifier method*), 617
__str__() (*Url method*), 708
__str__() (*ValueEnum method*), 618
__str__() (*VersionOrder method*), 886
__str__() (*_StrMatchMixin method*), 867
__summary__ (*in module conda*), 996
__summary__ (*in module conda.auxlib*), 610
__url__ (*in module conda*), 996
__url__ (*in module conda.auxlib*), 610
__version__ (*in module conda*), 995
__version__ (*in module conda._version*), 577
__version__ (*in module conda.auxlib*), 610
__version_tuple__ (*in module conda._version*), 577
_action (*_LazyParserMap attribute*), 635
_add_http_value_to_dict() (*in module conda.gateways.repodata*), 827
_add_info_dir() (*in module conda.cli.main_package*), 660
_add_pip_as_python_dependency (*RepodataSubset attribute*), 568, 832
_add_prefix_to_path() (*_Activator method*), 580
_add_specs() (*Solver method*), 764
_aggressive_update_packages (*Context attribute*), 622
_alias_canonical_channel_name_cache_to_file_prefixed() (*in module conda.testing.helpers*), 986
_aliased (*RepodataState attribute*), 827
_aliases (*Field attribute*), 601
_append_without_duplicates() (*EnvironmentConfig method*), 857
_apply_basic_auth() (*CondaHttpAuth static method*), 809
_arch_names (*in module conda.base.context*), 620
_array_append (*_ClauseArray attribute*), 664
_array_extend (*_ClauseArray attribute*), 664
_ask_upload() (*ExceptionHandler method*), 784
_assign() (*Clauses method*), 693
_base_component() (*InfoRenderer method*), 657
_base_url (*ShardBase attribute*), 563, 574
_base_url (*Shards attribute*), 564, 576
_base_url (*SubdirData attribute*), 767
_base_url_w_credentials (*SubdirData attribute*), 767
_bashrc_content() (*in module conda.core.initialize*), 724
_bashrc_content_to_add_condabin_to_path() (*in module conda.core.initialize*), 724
_broader() (*Resolve method*), 973
_build_activate_stack() (*_Activator method*), 580
_build_activator_cls() (*in module conda.activate*), 584
_build_components() (*MatchSpec method*), 864
_building (*_LazySubParsersAction attribute*), 636
_cache (*ZstdRepoInterface attribute*), 819
cache (*BuildNumberMatch attribute*), 888
cache (*Configuration attribute*), 684
cache (*Dist attribute*), 848
cache (*PackageCacheData attribute*), 730, 822
cache (*PluginConfig attribute*), 896

cache (*PrefixData attribute*), 755, 944
cache (*SubdirData attribute*), 766
cache (*VersionOrder attribute*), 886
cache (*VersionSpec attribute*), 888
_cached_choices (*LazyChoicesAction attribute*), 644
_cached_find_matches (*Resolve attribute*), 972
_calculate_change_report() (*UnlinkLinkTransaction static method*), 728
_calculate_md5sum() (*PackageCacheRecord method*), 882
_cast_args() (*CondaCLIFixture static method*), 978
_caused_by (*CondaError attribute*), 821, 996
_change_report_str() (*UnlinkLinkTransaction method*), 728
_channel_alias (*Context attribute*), 623
_channel_packages (*SimpleEnvironment property*), 992
_channel_priorities_map (*Resolve attribute*), 971
_channel_priority (*Resolve attribute*), 971
_channels (*BaseSolver attribute*), 761
_channels (*Context attribute*), 623
_channels (*Index attribute*), 716
_channels_component() (*InfoRenderer method*), 657
_channels_path (*SimpleEnvironment attribute*), 991
_check_files() (*PythonDistribution method*), 925
_check_literal() (*Clauses method*), 693
_check_path_data() (*PythonDistribution method*), 925
_check_value() (*ArgumentParser method*), 634
_check_variable() (*Clauses method*), 693
_check_writable() (*PackageCacheData method*), 730, 823
_classify_bad_deps() (*Resolve method*), 973
_clause_array (*_ClauseArray attribute*), 664
_clause_list (*_ClauseList attribute*), 664
_clauses (*Clauses attribute*), 693
_clauses (*_SatSolver attribute*), 665
_clean_environments_txt() (*in module conda.core.envs_manager*), 712
_clean_tarball_path_and_get_md5sum() (*PackageCacheData static method*), 730, 823
_cleanup_transaction_artifacts() (*UnlinkLinkTransaction method*), 728
_clear_cache_file() (*in module conda.notices.cache*), 890
_closed (*TemporaryDirectory attribute*), 810
_codesign_batch_state (*in module conda.core.portability*), 753
_codesign_or_enqueue() (*in module conda.core.portability*), 753
_collect_all_metadata() (*Solver method*), 763
_collect_validation_error() (*Configuration static method*), 685
_command (*BaseSolver attribute*), 761
_component_style_map (*InfoRenderer attribute*), 656
_conda_build (*Context attribute*), 625
_config_fish_content() (*in module conda.core.initialize*), 723
_config_fish_content_to_add_condabin_to_path() (*in module conda.core.initialize*), 724
_config_xonsh_content() (*in module conda.core.initialize*), 724
_config_xonsh_content_to_add_condabin_to_path() (*in module conda.core.initialize*), 724
_configure_builtin_subcommand() (*in module conda.cli.conda_argparse*), 633
_console (*Context attribute*), 624
_content (*ConfigurationFile attribute*), 638
_context (*ConfigurationFile attribute*), 638
_context (*InfoRenderer attribute*), 656
_context_params (*ConfigurationFile attribute*), 638
_convert() (*Clauses method*), 693
_convert() (*FeatureMatch method*), 868
_convert() (*ListOfStrMatch method*), 868
_convert() (*SplitStrMatch method*), 867
_create_default_packages (*Context attribute*), 621
_croot (*Context attribute*), 625
_cuda_driver_version_detector_target() (*in module conda.plugins.virtual_packages.cuda*), 968
_custom_channels (*Context attribute*), 623
_custom_multichannels (*Context attribute*), 623
_cutoff_for_record() (*ExcludeNewerPolicy method*), 714
_data (*PluginConfig attribute*), 896
_data (*PythonDistributionMetadata attribute*), 927
_debug (*Context attribute*), 623
_dedupe_pkgs_dir_contents() (*PackageCacheData static method*), 731, 823
_default (*Parameter attribute*), 680
_default_activation_env (*Context attribute*), 621
_default_channels (*Context attribute*), 623
_default_env() (*_Activator method*), 580
_default_in_dump (*Field attribute*), 601
_default_threads (*Context attribute*), 622
_derive_reduced_index() (*ReducedIndex method*), 719
_detail_component() (*InfoRenderer method*), 657
_detect_content_env_spec() (*CondaPluginManager static method*), 919
_detect_filename_env_spec() (*CondaPluginManager method*), 919
_dev (*Context attribute*), 623
_distribution_aliases (*CondaPluginManager attribute*), 914
_do_copy() (*in module conda.gateways.disk.create*), 811
_do_softlink() (*in module conda.gateways.disk.create*), 811

_drain_pending() (*RepodataSubset method*), 570, 834
_element_type (*ListField attribute*), 604
_element_type (*LoadedParameter attribute*), 676
_element_type (*MapLoadedParameter attribute*), 678
_element_type (*MapParameter attribute*), 681
_element_type (*ObjectLoadedParameter attribute*), 679
_element_type (*ObjectParameter attribute*), 682
_element_type (*Parameter attribute*), 679
_element_type (*PrimitiveLoadedParameter attribute*), 677
_element_type (*PrimitiveParameter attribute*), 680
_element_type (*SequenceLoadedParameter attribute*), 678
_element_type (*SequenceParameter attribute*), 681
_ensure_dir() (*time_recorder method*), 691
_ensure_hex_hash() (*in module conda._private.shards.misc*), 560
_ensure_loaded() (*_LazySubParsersAction method*), 636
_ensure_no_reserved_env_vars() (*PrefixData method*), 759, 948
_ensure_plugins_loaded() (*_LazySubParsersAction method*), 636
_ensure_value() (*NullCountAction static method*), 630
_environment (*Cep24YamlFileSpec attribute*), 779
_environment (*YamlFileSpec attribute*), 782
_envs_component() (*InfoRenderer method*), 657
_envs_dirs (*Context attribute*), 622
_eq() (*VersionOrder method*), 886
_error_upload_url (*Context attribute*), 624
_escaped_sys_rc_path() (*in module conda.cli.conda_argparse*), 633
_escaped_user_rc_path() (*in module conda.cli.conda_argparse*), 633
_eval() (*Clauses method*), 693
_exec() (*in module conda.cli.conda_argparse*), 636
_exec_unix() (*in module conda.cli.conda_argparse*), 636
_exec_win() (*in module conda.cli.conda_argparse*), 636
_execute() (*UnlinkLinkTransaction method*), 728
_execute() (*in module conda.cli.main_clean*), 649
_execute_actions() (*UnlinkLinkTransaction static method*), 728
_execute_channel() (*CacheUrlAction method*), 750
_execute_local() (*CacheUrlAction method*), 750
_execute_post_link_actions() (*UnlinkLinkTransaction static method*), 728
_execute_successful (*CompileMultiPycAction attribute*), 740
_execute_successful (*CreatePrefixRecordAction attribute*), 743
_execute_successful (*CreatePythonEntryPointAction attribute*), 742
_execute_successful (*LinkPathAction attribute*), 738
_execute_successful (*MakeMenuAction attribute*), 739
_execute_successful (*RegisterEnvironmentLocationAction attribute*), 744
_execute_successful (*UnregisterEnvironmentLocationAction attribute*), 748
_execute_successful (*UpdateHistoryAction attribute*), 744
_execute_threads (*Context attribute*), 622
_execute_upload() (*ExceptionHandler method*), 784
_executed (*ProgressiveFetchExtract attribute*), 731
_expand_channels() (*in module conda.env.env*), 775
_expand_search_path() (*Configuration static method*), 684
_expandvars (*ParameterLoader attribute*), 683
_expansion_specs() (*RepodataSubset method*), 568, 832
_experimental (*Context attribute*), 624
_export_platforms (*Context attribute*), 622
_export_subdir_data_to_repodata() (*in module conda.testing.helpers*), 985
_fetch_from_shards() (*ShardFetch method*), 562
_fetch_shards_impl() (*ShardFetch method*), 562
_fetch_threads (*Context attribute*), 622
_fetched (*ShardFetch attribute*), 562
_finalize() (*JSONFormatMixin method*), 584
_finalize() (*_Activator method*), 579
_find_inconsistent_packages() (*Solver method*), 763
_find_launcher() (*in module conda.core.launchers*), 725
_finish_extract() (*ExtractPackageAction method*), 751
_first_important_matches() (*LoadedParameter static method*), 677
_format_chain_str() (*UnsatisfiableError method*), 792
_format_exc() (*in module conda.exceptions*), 797
_format_leaf_errors() (*in module conda.exceptions*), 797
_format_output() (*in module conda.gateways.subprocess*), 836
_formatted_chains (*ResolvePackageNotFound attribute*), 785
_frozen_file (*PrefixData attribute*), 756, 945
_generate_message() (*DeprecationHandler method*), 772
_get_RE_WIN_ROOT() (*in module conda.common.path._cygpath*), 695
_get_activate_scripts() (*_Activator method*), 580
_get_attr() (*in module conda.auxlib.ish*), 607
_get_base_url() (*SubdirData method*), 769

_get_best_prec_match() (*in module conda.misc*), 841
_get_binstar_token_directory() (*in module conda.gateways.anaconda_client*), 800
_get_channel_for_name() (*in module conda.models.channel*), 847
_get_deactivate_scripts() (*_Activator method*), 580
_get_environment_env_vars() (*_Activator method*), 580
_get_environment_state_file() (*PrefixData method*), 759, 948
_get_errors() (*in module conda.exceptions*), 797
_get_guidance() (*UnsatisfiableError method*), 792
_get_guidance() (*in module conda.exceptions*), 797
_get_index_r_base() (*in module conda.testing.helpers*), 986
_get_json_fn() (*PrefixData method*), 758, 947
_get_json_fn() (*PrefixRecord method*), 883
_get_key() (*in module conda.cli.main_config*), 651
_get_module() (*DeprecationHandler method*), 772
_get_multiple_data() (*PythonDistributionMetadata method*), 928
_get_name_and_alias_mapping() (*CondaPluginManager method*), 918
_get_package_pool() (*Resolve method*), 974
_get_package_record_from_specs() (*in module conda.misc*), 840
_get_parameter_loader() (*Configuration method*), 684
_get_path_dirs() (*_Activator method*), 580
_get_pfe() (*UnlinkLinkTransaction method*), 728
_get_pkgs_dirs() (*in module conda.cli.main_clean*), 649
_get_python_info() (*UnlinkLinkTransaction static method*), 728
_get_python_info() (*in module conda.core.initialize*), 724
_get_render_func() (*in module conda.reporters*), 970
_get_root() (*in module conda.common.path._cygpath*), 695
_get_sat_solver_cls() (*in module conda.resolve*), 971
_get_size() (*in module conda.cli.main_clean*), 649
_get_solver_base() (*in module conda.testing.helpers*), 986
_get_starting_path_list() (*_Activator method*), 580
_get_strict_channel() (*Resolve method*), 973
_get_subactions() (*_GreedySubParsersAction method*), 634
_get_subactions() (*_LazySubParsersAction method*), 636
_get_timestamp() (*in module conda.core.exclude_newer*), 714
_get_total_size() (*in module conda.cli.main_clean*), 649
_get_url_pattern() (*in module conda.misc*), 840
_get_value() (*in module conda.core.exclude_newer*), 714
_get_version_tuple() (*DeprecationHandler static method*), 771
_get_yaml_key_comment() (*YamlRawParameter static method*), 675
_get_yaml_list_comment_item() (*YamlRawParameter static method*), 675
_get_yaml_list_comments() (*YamlRawParameter class method*), 675
_get_yaml_map_comments() (*YamlRawParameter static method*), 675
_grouped_parameter (*ParameterTypeGroups attribute*), 637
_guidance (*CondaError attribute*), 821, 996
_hash_key() (*MatchSpec method*), 864
_hook_postamble() (*PowerShellActivator method*), 584
_hook_postamble() (*_Activator method*), 579
_hook_preamble() (*CmdExeActivator method*), 583
_hook_preamble() (*JSONFormatMixin method*), 584
_hook_preamble() (*PowerShellActivator method*), 584
_hook_preamble() (*_Activator method*), 579
_hookexec() (*CondaPluginManager method*), 915
_hookspec (*in module conda.plugins.hookspec*), 900
_immutable (*Field attribute*), 601
_implementors (*in module conda.models.match_spec*), 869
_in_dump (*Field attribute*), 601
_index (*BaseSolver attribute*), 760
_indicator_length (*Spinner attribute*), 936
_individuals (*AggregateCompileMultiPycAction attribute*), 741
_info_dict (*InfoRenderer property*), 656
_info_dict_envs (*InfoRenderer property*), 657
_info_dict_envs_details (*InfoRenderer property*), 657
_init_solution_precs() (*SolverStateContainer method*), 764
_initd (*Entity property*), 606
_initialize_dev_bash() (*in module conda.core.initialize*), 722
_initialize_dev_cmdexe() (*in module conda.core.initialize*), 722
_install_file() (*in module conda.core.initialize*), 723
_internal (*PackageCacheData attribute*), 588
_internal (*PrefixData attribute*), 590
_internal (*Solver attribute*), 585
_internal (*SubdirData attribute*), 587
_is_exact (*BaseSpec attribute*), 887
_is_http_error_most_400_codes() (*in module*

conda._private.shards.misc), 560
_is_http_error_most_400_codes() (*in module conda.gateways.repodata.zstd*), 819
_is_literal() (*in module conda.plugins.prefix_data_loaders.pypi.pkg_format*), 931
_is_non_alias_url_channel() (*in module conda.models.match_spec*), 865
_is_simple() (*MatchSpec method*), 864
_is_single() (*MatchSpec method*), 864
_is_unix_executable_using_ORIGIN() (*in module conda.gateways.disk.create*), 811
_isatty (*ExceptionHandler property*), 784
_isiterable() (*_HookImplWrapper static method*), 914
_iter_records_by_name() (*SubdirData method*), 768
_iter_shard_records() (*in module conda._private.shards.misc*), 560
_json_all_component() (*InfoRenderer method*), 657
_json_error_map() (*in module conda.exceptions*), 797
_key_comment (*YamlRawParameter attribute*), 675
_key_exists() (*in module conda.cli.main_config*), 651
_key_mgr (*SubdirData attribute*), 767
_known_names() (*_LazyParserMap method*), 635
_kwargs (*CondaError attribute*), 821, 996
_last_activated_file (*PrefixData attribute*), 756, 945
_last_argparse_args (*ContextStack attribute*), 628
_last_search_path (*ContextStack attribute*), 628
_lazy_aliases (*_LazySubParsersAction attribute*), 636
_lazy_loaders (*_LazySubParsersAction attribute*), 636
_lazy_validate (*Dist attribute*), 848
_lazy_validate (*Entity attribute*), 606
_load() (*SubdirData method*), 768
_load_concurrency() (*in module conda.common.io*), 691
_load_requires_provides_file() (*PythonDistribution method*), 925
_load_search_path() (*Configuration class method*), 684
_load_single_record() (*PrefixData method*), 759, 948
_loaded (*SubdirData attribute*), 767
_loading_name (*_LazySubParsersAction attribute*), 636
_lock_impl() (*in module conda.gateways.disk.lock*), 813
_lock_noop() (*in module conda.gateways.disk.lock*), 813
_log (*ZstdRepoInterface property*), 819
_magic_file (*PrefixData attribute*), 756, 945
_make_channel_priorities() (*Resolve static method*), 974
_make_compile_actions() (*UnlinkLinkTransaction static method*), 728
_make_component() (*MatchSpec static method*), 864
_make_entry_point_actions() (*UnlinkLinkTransaction static method*), 728
_make_legacy_action_groups() (*UnlinkLinkTransaction method*), 728
_make_link_actions() (*UnlinkLinkTransaction static method*), 728
_make_milliseconds() (*TimestampField static method*), 874
_make_seconds() (*TimestampField static method*), 874
_make_single_record() (*PackageCacheData method*), 731, 823
_match_components (*MatchSpec attribute*), 863
_match_individual() (*MatchSpec method*), 864
_match_key_is_important() (*LoadedParameter static method*), 677
_match_specs_from_explicit() (*in module conda.misc*), 840
_md5_not_for_security() (*in module conda.gateways.repodata*), 830
_merge() (*EnvironmentConfig method*), 857
_merge() (*MatchSpec method*), 865
_merge_channel_settings() (*EnvironmentConfig method*), 857
_message_to_dict() (*PythonDistributionMetadata class method*), 927
_metadata (*PythonDistribution attribute*), 925
_migrated_channel_aliases (*Context attribute*), 623
_name (*LoadedParameter attribute*), 676
_name (*ParameterLoader attribute*), 682
_name_parser_map (*_LazySubParsersAction attribute*), 635
_names (*ParameterLoader attribute*), 682
_native_subdir() (*Context method*), 625
_neighbors() (*RepodataSubset method*), 568, 832
_new_makepasv() (*in module conda.gateways.connection.adapters.ftp*), 801
_nodes (*RepodataSubset attribute*), 568, 832
_nodes_from_packages() (*in module conda._private.shards.subset*), 568
_notify_conda_outdated() (*BaseSolver method*), 762
_notify_pip_as_python_deprecation() (*BaseSolver method*), 762
_nullable (*Field attribute*), 601
_old_makepasv (*in module conda.gateways.connection.adapters.ftp*), 801
_optional (*MatchSpec attribute*), 863
_order_helper (*Field attribute*), 601
_original_spec_str (*MatchSpec attribute*), 863
_other_key() (*BuildString method*), 842
_outgoing() (*RepodataSubset method*), 568, 833
_override() (*Context method*), 627

_override_virtual_packages (*Context attribute*), 625
_package_cache_records (*PackageCacheData property*), 730, 822
_package_data() (*SimpleEnvironment method*), 992
_package_has_updates() (*Solver method*), 764
_parameter_loaders (*ConfigurationType property*), 683
_parse_and_set_args() (*_Activator method*), 579
_parse_channel() (*in module conda.models.match_spec*), 865
_parse_comment_line() (*History class method*), 838
_parse_entries_file_data() (*PythonDistribution static method*), 925
_parse_iso_timestamp() (*ChannelNoticeResponse static method*), 894
_parse_legacy_dist() (*in module conda.models.match_spec*), 865
_parse_list_of_str() (*in module conda.models.match_spec*), 866
_parse_notice_level() (*ChannelNoticeResponse static method*), 894
_parse_old_format_specs_string() (*History static method*), 837
_parse_requires_file_data() (*PythonDistribution static method*), 925
_parse_spec_str() (*in module conda.models.match_spec*), 866
_parse_spec_str_dispatcher() (*in module conda.models.match_spec*), 866
_parse_spec_str_v3() (*in module conda.models.match_spec*), 866
_parse_version_plus_build() (*in module conda.models.match_spec*), 865
_patch_for_local_exports() (*in module conda.testing.helpers*), 986
_path (*ConfigurationFile attribute*), 638
_path (*PathData attribute*), 877
_path (*PythonDistributionMetadata attribute*), 927
_path (*SimpleEnvironment attribute*), 991
_path_to() (*in module conda.common.path.windows*), 698
_pfe (*UnlinkLinkTransaction attribute*), 727
_pickle_me() (*SubdirData method*), 768
_pickle_valid_checks() (*SubdirData method*), 768
_pipelined_traversal() (*RepodataSubset method*), 569, 833
_pkey (*PackageRecord property*), 879
_pkgs_dirs (*Context attribute*), 622
_platform_map (*in module conda.base.context*), 620
_plugins_loaded (*_LazySubParsersAction attribute*), 636
_pool_cache (*BaseSolver attribute*), 761
_pool_cache (*Resolve attribute*), 972
_post_sat_handling() (*Solver method*), 764
_post_upload() (*ExceptionHandler method*), 784
_powershell_profile_content() (*in module conda.core.initialize*), 724
_powershell_profile_content_to_add_condabin_to_path() (*in module conda.core.initialize*), 724
_prefix_path (*SimpleEnvironment attribute*), 991
_prefix_records (*PrefixData property*), 759, 948
_prepare() (*Solver method*), 764
_prepare() (*UnlinkLinkTransaction class method*), 728
_prepare_extract() (*ExtractPackageAction method*), 751
_prepared (*BaseSolver attribute*), 761
_prepared (*ProgressiveFetchExtract attribute*), 731
_prepared (*UnlinkLinkTransaction attribute*), 727
_pretty_record_format() (*in module conda.cli.main_search*), 662
_print_conda_exception() (*ExceptionHandler method*), 783
_process_fetch_result() (*ShardFetch method*), 562
_process_path() (*PythonDistributionMetadata static method*), 927
_process_raw_repodata() (*SubdirData method*), 769
_process_raw_repodata_str() (*SubdirData method*), 769
_progress_bar() (*ProgressiveFetchExtract static method*), 731
_prompt_modifier() (*_Activator method*), 580
_provided_index (*BaseSolver attribute*), 760
_provides_file_data (*PythonDistribution attribute*), 925
_r (*BaseSolver attribute*), 760
_raw_arguments (*_Activator attribute*), 579
_raw_parameters_from_single_source() (*ParameterLoader method*), 683
_raw_value (*MatchInterface attribute*), 866
_re_match (*ChannelMatch attribute*), 868
_re_match (*GlobStrMatch attribute*), 867
_reachable_bfs() (*RepodataSubset method*), 569, 833
_reachable_pipelined() (*RepodataSubset method*), 569, 833
_read_channel_configuration() (*in module conda.models.channel*), 847
_read_local_repodata() (*SubdirData method*), 768
_read_metadata() (*PythonDistributionMetadata class method*), 928
_read_pickled() (*SubdirData method*), 769
_read_rc() (*in module conda.cli.main_config*), 651
_read_state (*ConfigurationFile attribute*), 638
_read_windows_registry() (*in module conda.core.initialize*), 724
_ready (*_LazyParserMap property*), 635
_realize() (*Index method*), 718
_reduced_index_cache (*Resolve attribute*), 972

_register_enum_representers() (*in module conda.cli.condarc*), 637
_remove_item() (*in module conda.cli.main_config*), 651
_remove_key() (*in module conda.cli.main_config*), 651
_remove_node() (*PrefixGraph method*), 872, 934
_remove_prefix_from_path() (*_Activator method*), 580
_remove_specs() (*Solver method*), 763
_replace_prefix_in_path() (*_Activator method*), 580
_repo (*RepodataFetch property*), 829
_repo (*SubdirData property*), 767
_repodata_fn (*BaseSolver attribute*), 761
_repodata_fn (*CondaRepoInterface attribute*), 827
_repodata_fn (*Index attribute*), 716
_repodata_fn (*ZstdRepoInterface attribute*), 819
_repodata_shards() (*in module conda._private.shards.shards*), 565
_repodata_state_copy() (*ZstdRepoInterface method*), 820
_repodata_threads (*Context attribute*), 622
_repodata_version (*RepodataSubset attribute*), 568, 832
_report_errors (*Context attribute*), 624
_representer() (*in module conda.common.serialize.yaml*), 703
_requested_specs (*SolvedRecord attribute*), 882
_required (*Field attribute*), 601
_requires_file_data (*PythonDistribution attribute*), 925
_reset_cache() (*Configuration method*), 685
_reset_callbacks (*Configuration attribute*), 684
_reset_state() (*Channel static method*), 824, 844
_resolve() (*_LazyParserMap method*), 635
_resolve() (*in module conda.exports*), 799
_resolve_all() (*_LazyParserMap method*), 635
_resolve_path() (*in module conda.common.path._cygpath*), 696
_resolve_plugin_names() (*CondaPluginManager method*), 917
_resolve_prefix() (*CmdExeActivator method*), 583
_resolve_prefix() (*_Activator method*), 580
_retrieve_all_from_channels() (*Index method*), 718
_retrieve_from_channels() (*Index method*), 718
_reverse_actions() (*UnlinkLinkTransaction static method*), 728
_rewrite_environments_txt() (*in module conda.core.envs_manager*), 712
_rm_rf() (*in module conda.cli.main_clean*), 649
_root_prefix (*Context attribute*), 622
_run_codesign() (*in module conda.core.portability*), 753
_run_kwargs (*_SatSolver attribute*), 665
_run_sat() (*Clauses method*), 667
_run_sat() (*Solver method*), 764
_safe_opt_str() (*in module conda.models.records*), 873
_safe_toposort() (*in module conda.common.toposort*), 706
_safe_urljoin_with_slash() (*in module conda._private.shards.misc*), 559
_sanitize_version_str() (*in module conda.models.match_spec*), 865
_sat_solver (*Clauses attribute*), 666
_sat_solver_cls_to_str (*in module conda.common._logic*), 666
_sat_solver_str_to_cls (*in module conda.common._logic*), 666
_sat_solvers (*in module conda.resolve*), 971
_scan_for_dist_no_channel() (*PackageCacheData method*), 730, 823
_search_package() (*in module conda.core.subdir_data*), 769
_search_package_via_shards() (*in module conda.core.subdir_data*), 769
_send_boto3() (*S3Adapter method*), 805
_set_argparse_args() (*Configuration method*), 684
_set_entry_name() (*time_recorder method*), 691
_set_env_vars() (*Configuration method*), 684
_set_impl() (*_HookImplWrapper static method*), 914
_set_key() (*in module conda.cli.main_config*), 651
_set_name() (*ParameterLoader method*), 683
_set_parameter_names_and_aliases() (*Configuration class method*), 684
_set_raw_data() (*Configuration method*), 684
_set_search_path() (*Configuration method*), 684
_shard (*ShardFetch attribute*), 561
_shard_index_exists() (*in module conda.gateways.repodata*), 827
_shard_url_cache (*Shards attribute*), 564, 576
_shards_base_url (*Shards attribute*), 564, 576
_shards_base_url() (*in module conda._private.shards.shards*), 564
_shards_connections() (*in module conda._private.shards.misc*), 559
_shards_from_bytes() (*in module conda._private.shards.shards*), 565
_shards_known() (*RepodataCache method*), 829
_should_freeze() (*Solver method*), 764
_show_size (*InfoRenderer attribute*), 656
_signing_metadata_url_base (*Context attribute*), 622
_solve() (*in module conda.env.installers.conda*), 777
_solver_class (*SimpleEnvironment attribute*), 991
_solver_helper() (*in module conda.testing.fixtures*), 978

_solver_ignore_timestamps (*Resolve attribute*), 972
_sort_key (*PackageRecord property*), 879
_spec_to_package_name (*RepodataSubset attribute*), 568, 832
_spinner_thread (*Spinner attribute*), 936
_split() (*in module conda.models.build_string*), 842
_split_platform_re() (*in module conda.common.url*), 710
_ssl_context (*_SSLContextAdapterMixin attribute*), 803
_stack (*ContextStack attribute*), 628
_stack_idx (*ContextStack attribute*), 628
_start_spinning() (*Spinner method*), 936
_stderrlog (*ZstdRepoInterface property*), 819
_stop_running (*Spinner attribute*), 936
_stream_is_tty() (*in module conda.common.terminal*), 705
_strict_channel_cache (*Resolve attribute*), 972
_strings (*RepodataState attribute*), 827
_subdir (*Context attribute*), 622
_subdir_is_win() (*in module conda.core.portability*), 752
_subdirs (*Context attribute*), 622
_subdirs (*Index attribute*), 716
_subprocess_clean_env() (*in module conda.gateways.subprocess*), 836
_supplement_index_dict_with_cache() (*Index method*), 718
_supplement_index_dict_with_prefix() (*Index method*), 718
_sync_channel_to_disk() (*in module conda.testing.helpers*), 986
_sys_rc_path() (*in module conda.cli.conda_argparse*), 633
_system_component() (*InfoRenderer method*), 657
_system_precs (*Resolve attribute*), 972
_target (*MatchSpec attribute*), 863
_to_filename_do_not_use() (*MatchSpec method*), 864
_to_unix_drive() (*in module conda.common.path._cygpath*), 695
_to_unix_mount() (*in module conda.common.path._cygpath*), 695
_to_unix_root() (*in module conda.common.path._cygpath*), 695
_to_win_drive() (*in module conda.common.path._cygpath*), 696
_to_win_mount() (*in module conda.common.path._cygpath*), 696
_to_win_root() (*in module conda.common.path._cygpath*), 696
_topo_sort_handle_cycles() (*PrefixGraph class method*), 872, 934
_toposort() (*PrefixGraph method*), 872, 934
_toposort() (*in module conda.common.toposort*), 706
_toposort_pop_key() (*PrefixGraph static method*), 872, 934
_toposort_prepare_graph() (*PrefixGraph static method*), 872, 934
_toposort_raise_on_cycles() (*PrefixGraph class method*), 872, 934
_touch_last_activated() (*in module conda.activate*), 578
_tqdm() (*TQDMProgressBar static method*), 936
_trace (*Context attribute*), 623
_type (*BooleanField attribute*), 602
_type (*ComposableField attribute*), 605
_type (*DateField attribute*), 603
_type (*EnumField attribute*), 604
_type (*IntegerField attribute*), 602
_type (*ListField attribute*), 604
_type (*LoadedParameter attribute*), 676
_type (*MapField attribute*), 605
_type (*MapLoadedParameter attribute*), 678
_type (*MapParameter attribute*), 681
_type (*NumberField attribute*), 603
_type (*ObjectLoadedParameter attribute*), 679
_type (*ObjectParameter attribute*), 682
_type (*Parameter attribute*), 679
_type (*PrimitiveLoadedParameter attribute*), 677
_type (*PrimitiveParameter attribute*), 680
_type (*SequenceLoadedParameter attribute*), 678
_type (*SequenceParameter attribute*), 681
_type (*StringField attribute*), 603
_typify_data_structure() (*LoadedParameter static method*), 677
_update_from_cache() (*Index method*), 718
_update_from_prefix() (*Index method*), 718
_update_prompt() (*CmdExeActivator method*), 583
_update_prompt() (*CshActivator method*), 581
_update_prompt() (*PosixActivator method*), 581
_update_prompt() (*_Activator method*), 580
_upload() (*ExceptionHandler method*), 784
_url (*CondaRepoInterface attribute*), 827
_url (*ZstdRepoInterface attribute*), 819
_urls_data (*PackageCacheData attribute*), 730, 822
_use_only_tar_bz2 (*Context attribute*), 623
_use_only_tar_bz2 (*RepodataSubset attribute*), 568, 832
_user_rc_path() (*in module conda.cli.conda_argparse*), 633
_validate_exclude_newer_link_precs() (*BaseSolver method*), 761
_validate_exclude_newer_support() (*BaseSolver method*), 761
_validate_no_denylist_channels() (*_ValidatePackages static method*), 644

_validate_package_name() (*in module conda.models.match_spec*), 861
_validate_plugin_headers() (*in module conda.gateways.connection.session*), 807
_validate_provided_parameters() (*in module conda.cli.main_config*), 651
_validate_when_spec() (*in module conda.models.match_spec*), 866
_validation (*Field attribute*), 601
_validation (*LoadedParameter attribute*), 676
_validation (*Parameter attribute*), 680
_validation_errors (*Configuration attribute*), 684
_verbosity (*Context attribute*), 624
_verified (*Action attribute*), 733
_verified (*UnlinkLinkTransaction attribute*), 727
_verify() (*UnlinkLinkTransaction method*), 728
_verify_individual_level() (*UnlinkLinkTransaction static method*), 728
_verify_pre_link_message() (*UnlinkLinkTransaction method*), 728
_verify_prefix_level() (*UnlinkLinkTransaction static method*), 728
_verify_threads (*Context attribute*), 622
_verify_transaction_level() (*UnlinkLinkTransaction static method*), 728
_version (*DeprecationHandler attribute*), 771
_version_less_than() (*DeprecationHandler method*), 771
_version_object (*DeprecationHandler attribute*), 771
_version_tuple (*DeprecationHandler attribute*), 771
_visit_node() (*RepodataSubset method*), 569, 834
_wait_and_close_handle() (*in module conda.common._os.windows*), 670
_win_sysconfig_map (*in module conda.base.context*), 621
_write_environment_state_file() (*PrefixData method*), 759, 948
_write_installed_packages() (*SimpleEnvironment method*), 992
_write_rc() (*in module conda.cli.main_config*), 651
_write_repo_packages() (*SimpleEnvironment method*), 992
_write_tempfile() (*S3Adapter method*), 805
_write_windows_registry() (*in module conda.core.initialize*), 724
_yaml() (*in module conda.common.serialize.yaml*), 703
_yaml_round_trip() (*in module conda.common.serialize*), 703
_yaml_safe() (*in module conda.common.serialize*), 703
_yield_commands() (*JSONFormatMixin method*), 584
_yield_commands() (*_Activator method*), 579

A

aarch64 (*Arch attribute*), 851
ACCESS_DENIED (*ERROR attribute*), 669
Action (*class in conda.core.path_actions*), 733
action (*CondaHealthCheck attribute*), 535, 958
action (*CondaPostCommand attribute*), 537, 957
action (*CondaPostSolve attribute*), 959
action (*CondaPostTransactionAction attribute*), 541, 962
action (*CondaPreCommand attribute*), 538, 957
action (*CondaPreSolve attribute*), 959
action (*CondaPreTransactionAction attribute*), 539, 962
action (*CondaSubcommand attribute*), 549, 949, 953, 955
action (*LinkScriptError attribute*), 788
action() (*DeprecationHandler method*), 771
ACTION_CODES (*in module conda.instructions*), 839
ActionGroup (*class in conda.core.link*), 726
actions (*ActionGroup attribute*), 726
activate() (*_Activator method*), 579
ActivateHelp, 785
activator_map (*in module conda.activate*), 584
active (*ExcludeNewerPolicy property*), 714
active_prefix (*CondaExceptionEvent attribute*), 530, 966
active_prefix (*Context property*), 626
add() (*ConfigurationFile method*), 640
add() (*Dependencies method*), 776
add_anaconda_token (*Context attribute*), 623
add_back_map (*SolverStateContainer attribute*), 764
add_binstar_token() (*CondaHttpAuth static method*), 809
add_channels() (*EnvironmentYaml method*), 776
add_clause (*_SatSolver attribute*), 665
add_clause (*Clauses attribute*), 666
add_clause() (*Clauses method*), 693
add_clauses (*_SatSolver attribute*), 665
add_clauses (*Clauses attribute*), 666
add_clauses() (*Clauses method*), 693
add_condabin_to_path() (*in module conda.core.initialize*), 722
add_condabin_to_path_registry() (*in module conda.core.initialize*), 724
add_feature_records_legacy() (*in module conda.testing.helpers*), 985
add_lazy_subcommand() (*_LazySubParsersAction method*), 636
add_output_and_prompt_options() (*in module conda.cli.helpers*), 644
add_output_and_prompt_options() (*in module conda.plugins.subcommands.doctor*), 944
add_parser() (*_LazySubParsersAction method*), 636
add_parser_channels() (*in module conda.cli.helpers*), 644
add_parser_create_install_update() (*in module conda.cli.helpers*), 644

add_parser_default_packages() (*in module conda.cli.helpers*), 645
add_parser_environment_specifier() (*in module conda.cli.helpers*), 645
add_parser_frozen_env() (*in module conda.cli.helpers*), 644
add_parser_frozen_env() (*in module conda.plugins.subcommands.doctor*), 944
add_parser_help() (*in module conda.cli.helpers*), 644
add_parser_help() (*in module conda.plugins.subcommands.doctor*), 944
add_parser_json() (*in module conda.cli.helpers*), 644
add_parser_known() (*in module conda.cli.helpers*), 645
add_parser_networking() (*in module conda.cli.helpers*), 645
add_parser_package_install_options() (*in module conda.cli.helpers*), 645
add_parser_platform() (*in module conda.cli.helpers*), 645
add_parser_prefix() (*in module conda.cli.helpers*), 644
add_parser_prefix() (*in module conda.plugins.subcommands.doctor*), 944
add_parser_prefix_to_group() (*in module conda.cli.helpers*), 644
add_parser_prune() (*in module conda.cli.helpers*), 644
add_parser_pscheck() (*in module conda.cli.helpers*), 644
add_parser_show_channel_urls() (*in module conda.cli.helpers*), 644
add_parser_solver() (*in module conda.cli.helpers*), 644
add_parser_solver_mode() (*in module conda.cli.helpers*), 644
add_parser_update_modifiers() (*in module conda.cli.helpers*), 644
add_parser_verbose() (*in module conda.cli.helpers*), 645
add_pip_as_python_dependency (*Context attribute*), 621
add_plugin_setting() (*PluginConfig class method*), 895
add_resp_to_mock() (*in module conda.testing.notices.helpers*), 990
add_subdir() (*in module conda.testing.helpers*), 985
add_subdir_to_iter() (*in module conda.testing.helpers*), 985
add_url() (*UrlsData method*), 731
add_username_and_password() (*in module conda.common.url*), 710
Administering a multi-user conda installation, **84**
AggregateCompileMultiPycAction (*class in conda.core.path_actions*), 741
aggressive_update_packages (*Context property*), 626
aggressive_update_packages (*EnvironmentConfig attribute*), 857
aliases (*CondaEnvironmentExporter attribute*), 513, 964
aliases (*CondaEnvironmentSpecifier attribute*), 964
aliases (*CondaSetting attribute*), 547, 959
aliases (*CondaSubcommand attribute*), 549, 949, 953, 955
aliases (*ParameterLoader attribute*), 683
All() (*Clauses method*), 667, 694
all_ancestors() (*PrefixGraph method*), 872, 934
all_caches_writable_first() (*PackageCacheData class method*), 730, 822
all_channel_urls() (*in module conda.models.channel*), 847
all_descendants() (*PrefixGraph method*), 871, 934
all_link_path_actions (*CreatePrefixRecordAction attribute*), 743
all_match() (*BaseSpec method*), 888
all_subdir_urls() (*PrefixData method*), 758, 947
allow_conda_downgrades (*Context attribute*), 621
allow_cycles (*Context attribute*), 621
allow_non_channel_urls (*Context attribute*), 623
allow_softlinks (*Context attribute*), 621
allowed_errors (*Repodatas attribute*), 646
allowed_errors (*TryRepodata attribute*), 646
allowlist_channels (*Context attribute*), 623
altered_files() (*in module conda.plugins.subcommands.doctor.health_checks.altered_files*), 939
ALTERNATIVE_CONDARC_FILENAME (*in module conda.common.configuration*), 683
always_copy (*Context attribute*), 623
always_softlink (*Context attribute*), 623
always_true_match() (*BaseSpec method*), 888
always_yes (*Context attribute*), 623
AmbiguousEnvironmentSpecPlugin, 796
Anaconda Distribution, **12**
anaconda_upload (*Context attribute*), 625
anchor_full_path (*PythonDistribution attribute*), 925
AND (*in module conda.plugins.prefix_data_loaders.pypi.pkg_format*), 931
And() (*Clauses method*), 666, 694
animations_disabled (*ConsoleReporterRenderer property*), 937
AnnotatedRawShard (*class in conda._private.shards.cache*), 557
Any() (*Clauses method*), 667, 694
any_match() (*BaseSpec method*), 888

any_subprocess() (*in module conda.gateways.subprocess*), 836
API reference, **417**
APP_NAME (*in module conda.base.constants*), 613
append (*_ClauseList attribute*), 664
append() (*_ClauseArray method*), 664
application_entry_point_template (*in module conda.gateways.disk.create*), 811
apply() (*ContextStack method*), 628
apply() (*ContextStackObject method*), 628
Arch (*class in conda.models.enums*), 850
arch (*PackageRecord attribute*), 880
arch_name (*Context property*), 625
arch_name (*in module conda.exports*), 799
Architecture, **416**
archspec_build() (*in module conda.plugins.virtual_packages.archspec*), 967
ArgParseRawParameter (*class in conda.common.configuration*), 675
argument() (*DeprecationHandler method*), 771
ArgumentError, 785
ArgumentParser (*class in conda.cli.conda_argparse*), 634
argv (*CondaExceptionEvent attribute*), 530, 966
argv() (*in module conda.common.io*), 690
arm64 (*Arch attribute*), 851
armv6l (*Arch attribute*), 851
armv7l (*Arch attribute*), 851
as_array() (*_ClauseArray method*), 665
as_array() (*_ClauseList method*), 664
as_dict() (*Url method*), 708
as_list() (*_ClauseArray method*), 665
as_list() (*_ClauseList method*), 664
as_list() (*_SatSolver method*), 665
as_list() (*Clauses method*), 666, 693
assert_environment() (*PrefixData method*), 757, 946
assert_equals() (*in module conda.testing.helpers*), 985
assert_exists() (*PrefixData method*), 757, 946
assert_in() (*in module conda.testing.helpers*), 985
assert_not_frozen() (*PrefixData method*), 757, 946
assert_not_in() (*in module conda.testing.helpers*), 985
assert_unsatisfiable() (*SolverTests method*), 992
assert_writable() (*PrefixData method*), 757, 946
assign() (*Clauses method*), 666
ASSOC_INCOMPLETE (*ERROR attribute*), 669
AtMostOne() (*Clauses method*), 694
AtMostOne_BDD() (*Clauses method*), 667, 694
AtMostOne_NSQ() (*Clauses method*), 667, 694
attach_stderr() (*in module conda.auxlib.logz*), 608
attach_stderr_handler() (*in module conda.common.io*), 690
attestations_sha256 (*PackageRecordDict attribute*), 572
AttrDict (*class in conda.auxlib.collection*), 591
auth (*_SplitUrlParts attribute*), 710
auth (*Channel attribute*), 824, 844
auth (*CondaSession attribute*), 808
auth (*MultiChannel attribute*), 846
auth (*PrefixRecord attribute*), 883
auth (*Url property*), 708
AuthenticationError, 791
auto_activate (*Context attribute*), 621
auto_injector_fixture() (*BaseTestCase method*), 975
auto_stack (*Context attribute*), 621
auto_update_conda (*Context attribute*), 621
AuxlibError (*class in conda.auxlib.exceptions*), 607
av_data_dir (*Context property*), 626
available_platforms (*EnvironmentSpecBase property*), 963

B

backend (*CondaSolver attribute*), 547, 956
backoff_rename() (*in module conda.gateways.disk.update*), 816
backoff_rmdir() (*in module conda.gateways.disk.delete*), 812
backslash_to_forwardslash() (*in module conda.activate*), 580
bad_deps (*ResolvePackageNotFound attribute*), 785
BAD_FORMAT (*ERROR attribute*), 669
bad_installed() (*Resolve method*), 974
base (*ShardCache attribute*), 557
base_url (*Channel property*), 825, 845
base_url (*Dist attribute*), 849
base_url (*MultiChannel property*), 847
base_url (*RepodataInfoDict attribute*), 573
base_url (*ShardBase property*), 563, 574
base_url (*Shards attribute*), 830
base_urls (*Channel property*), 825, 845
base_urls (*MultiChannel property*), 847
BaseSolver (*class in conda.core.solve*), 760
BaseSpec (*class in conda.models.version*), 887
BaseTestCase (*class in conda.testing.cases*), 975
basic_types() (*PathEnum method*), 854
BasicClobberError, 786
batch_codesign_calls() (*in module conda.core.portability*), 753
batch_retrieve_from_cache() (*in module conda._private.shards.shards*), 565
batch_retrieve_from_network() (*in module conda._private.shards.shards*), 565
BDD() (*Clauses method*), 667
binary (*FileMode attribute*), 853

binary_replace() (*in module conda.core.portability*), 753
BinaryPrefixReplacementError, 794
binstar_upload (*Context property*), 626
binstar_upload (*in module conda.exports*), 799
bits (*Context property*), 626
bits (*in module conda.exports*), 799
bld_path (*Context attribute*), 624
BooleanField (*class in conda.auxlib.entity*), 602
BoolField (*in module conda.auxlib.entity*), 602
boolify() (*in module conda.auxlib.type_coercion*), 609
bottom (*ParameterFlag attribute*), 674
box() (*_FeaturesField method*), 875
box() (*BooleanField method*), 602
box() (*ComposableField method*), 605
box() (*DateField method*), 603
box() (*EnumField method*), 604
box() (*Field method*), 601
box() (*LinkTypeField method*), 874
box() (*ListField method*), 604
box() (*MapField method*), 605
box() (*NoarchField method*), 869, 874
box() (*StringField method*), 603
box() (*TimestampField method*), 874
breadth_first_search_by_name() (*GeneralGraph method*), 872
breadth_first_search_for_dep_graph() (*Resolve method*), 973
Breaking changes, **417**
build (*CondaVirtualPackage attribute*), 550, 956
build (*Dist property*), 849
build (*PackageInfo property*), 870
build (*PackageRecord attribute*), 878
build (*PackageRecordDict attribute*), 572
build_activate() (*_Activator method*), 579
build_binary_response() (*in module conda.gateways.connection.adapters.ftp*), 802
build_conflict_map() (*Resolve method*), 973
build_deactivate() (*_Activator method*), 580
build_graph_of_deps() (*Resolve method*), 973
build_headers() (*in module conda.gateways.repodata.zstd*), 819
build_number (*BuildString property*), 842
build_number (*Dist attribute*), 849
build_number (*DistDetails attribute*), 848
build_number (*PackageInfo property*), 870
build_number (*PackageRecord attribute*), 878
build_number (*PackageRecordDict attribute*), 572
build_reactivate() (*_Activator method*), 580
build_repodata() (*ShardBase method*), 563, 575
build_repodata_subset() (*in module conda._private.shards.subset*), 570
build_repodata_subset() (*in module conda.gateways.shards*), 834
build_response() (*in module conda.gateways.connection.adapters.ftp*), 802
build_stack() (*_Activator method*), 580
build_string (*Dist attribute*), 849
build_string (*DistDetails attribute*), 848
build_text_response() (*in module conda.gateways.connection.adapters.ftp*), 802
BuildNumberMatch (*class in conda.models.version*), 888
BuildRepodataSubset (*class in conda.gateways.shards*), 834
BuildRepodataSubset (*class in conda.gateways.shards.typing*), 831
BuildString (*class in conda.models.build_string*), 842
BUILTIN_COMMANDS (*in module conda.activate*), 578
BUILTIN_COMMANDS (*in module conda.cli.conda_argparse*), 633
BUILTIN_PLUGIN_PREFIX (*in module conda.plugins.manager*), 914
BUILTIN_SUBCOMMANDS (*in module conda.cli.conda_argparse*), 633

C

cache_actions (*ProgressiveFetchExtract property*), 731
cache_clear() (*CondaSession class method*), 808
cache_clear() (*YamlRawParameter class method*), 676
cache_control (*RepodataState property*), 827
CACHE_CONTROL_KEY (*in module conda.gateways.repodata*), 826
cache_dir (*RepodataCache attribute*), 828
cache_entries (*Index property*), 716
cache_fetch_thread() (*in module conda._private.shards.subset*), 570
cache_fn_url() (*in module conda.gateways.repodata*), 830
cache_path_base (*RepodataFetch attribute*), 829
cache_path_base (*SubdirData property*), 767
cache_path_json (*RepodataCache property*), 828
cache_path_json (*RepodataFetch property*), 829
cache_path_json (*RepodataState attribute*), 827
cache_path_json (*SubdirData property*), 768
cache_path_pickle (*SubdirData property*), 768
cache_path_shards (*RepodataCache property*), 828
cache_path_state (*RepodataCache property*), 828
cache_path_state (*RepodataFetch property*), 829
cache_path_state (*RepodataState attribute*), 827
cache_path_state (*SubdirData property*), 768
CACHE_STATE_SUFFIX (*in module conda.gateways.repodata*), 826
cached_cuda_version() (*in module conda.plugins.virtual_packages.cuda*), 968

cached_response() (*in module conda.notices.cache*), 889
CacheUrlAction (*class in conda.core.path_actions*), 749
can_handle() (*Cep24YamlFileSpec method*), 779
can_handle() (*EnvironmentSpecBase method*), 963
can_handle() (*ExplicitSpec method*), 780
can_handle() (*RequirementsSpec method*), 781
can_handle() (*YamlFileSpec method*), 782
CancelOperation, 815
canonical_name (*Channel property*), 825, 845
canonical_name (*MultiChannel property*), 846
capped_decompress() (*in module conda._private.zstd*), 577
capsys (*CondaCLIFixture attribute*), 978
captured() (*in module conda.common.io*), 689
captured() (*in module conda.testing.helpers*), 985
CaptureTarget (*class in conda.common.io*), 688
CaseInsensitiveStrMatch (*class in conda.models.match_spec*), 869
category_map (*Context attribute*), 628
category_map (*PluginConfig property*), 896
cause (*ErrorGuidance attribute*), 555
Cep24YamlFileSpec (*class in conda.env.specs.cep_24*), 779
changeps1 (*Context attribute*), 621
ChangeReport (*class in conda.core.link*), 727
Channel (*class in conda.gateways.repodata*), 824
Channel (*class in conda.models.channel*), 844
channel (*Dist attribute*), 849
channel (*Node attribute*), 567
channel (*NodeId attribute*), 567
channel (*PackageInfo attribute*), 870
channel (*PackageRecord attribute*), 878
channel (*RepodataFetch attribute*), 829
channel (*SubdirData attribute*), 767
channel_alias() (*Context method*), 626
channel_alias_validation() (*in module conda.base.context*), 621
channel_cutoffs (*ExcludeNewerPolicy attribute*), 714
CHANNEL_DIR_V2 (*in module conda.testing.helpers*), 985
channel_from() (*_ChannelSelector static method*), 713
channel_location (*Channel property*), 824, 844
channel_name (*Channel property*), 824, 844
channel_name (*ChannelNameMixin attribute*), 957
channel_name (*ChannelNotice attribute*), 893
channel_name (*PackageRecord property*), 879
channel_notices (*ChannelNoticeResultSet attribute*), 893
channel_priority (*Context attribute*), 623
channel_priority (*EnvironmentConfig attribute*), 857
channel_settings (*Context attribute*), 623
channel_settings (*EnvironmentConfig attribute*), 857
channel_urls (*PackagesNotFoundError attribute*), 791
ChannelAuthBase (*class in conda.plugins.types*), 957
ChannelDenied, 789
ChannelError, 789
ChannelField (*class in conda.models.records*), 875
ChannelMatch (*class in conda.models.match_spec*), 868
ChannelName (*in module conda.notices.core*), 891
ChannelNameMixin (*class in conda.plugins.types*), 957
ChannelNotAllowed, 789
ChannelNotice (*class in conda.notices.types*), 893
ChannelNoticeResponse (*class in conda.notices.types*), 893
ChannelNoticeResultSet (*class in conda.notices.types*), 893
ChannelPriority (*class in conda.base.constants*), 618
ChannelPriorityMeta (*class in conda.base.constants*), 617
Channels, **54**, **116**
channels (*BaseSolver attribute*), 761
channels (*Channel property*), 825, 845
channels (*CondaExceptionEvent attribute*), 530, 966
channels (*Context property*), 626
channels (*EnvironmentConfig attribute*), 857
channels (*EnvironmentYaml attribute*), 776
channels (*Index attribute*), 716
channels (*MultiChannel property*), 847
channels (*Resolve attribute*), 971
channels_validation() (*in module conda.env.env*), 774
ChannelUrl (*in module conda.notices.core*), 891
Cheat sheet, **137**
CHECK_ALTERNATE_FORMAT_INTERVAL (*in module conda.gateways.repodata*), 826
check_envs_txt_file() (*in module conda.plugins.subcommands.doctor.health_checks.environment_t*, 941
CHECK_EXTRACT (*in module conda.instructions*), 839
CHECK_FETCH (*in module conda.instructions*), 839
check_files_in_package() (*in module conda.instructions*), 840
check_non_admin() (*in module conda.cli.common*), 631
check_source() (*Configuration method*), 685
ChecksumMismatchError, 790
choices (*_LazySubParsersAction attribute*), 636
choices (*LazyChoicesAction property*), 644
choices_func (*LazyChoicesAction attribute*), 644
CHUNK_SIZE (*in module conda.gateways.connection.download*), 806
CLASSIC_SOLVER (*in module conda.base.constants*), 618
classproperty (*class in conda.auxlib.decorators*), 595
Clauses (*class in conda.common._logic*), 666
Clauses (*class in conda.common.logic*), 693
CLEAN (*Commands attribute*), 988
cleanup() (*Action method*), 734

cleanup() (*CacheUrlAction method*), 750
cleanup() (*CompileMultiPycAction method*), 741
cleanup() (*CreateInPrefixPathAction method*), 737
cleanup() (*ExtractPackageAction method*), 751
cleanup() (*RegisterEnvironmentLocationAction method*), 745
cleanup() (*RemoveMenuAction method*), 747
cleanup() (*TemporaryDirectory method*), 810
cleanup() (*UnlinkPathAction method*), 746
cleanup() (*UnregisterEnvironmentLocationAction method*), 749
cleanup() (*UpdateHistoryAction method*), 744
clear() (*PackageCacheData class method*), 730, 822
clear_cache() (*in module conda.notices.cache*), 890
clear_cache() (*ShardCache method*), 558
clear_cached_local_channel_data() (*SubdirData class method*), 766
clear_has_format() (*RepodataState method*), 828
clear_key() (*ConfigurationFile method*), 641
clear_memoized_methods() (*in module conda.auxlib.decorators*), 594
clear_plugin_manager_cache() (*in module conda.testing.fixtures*), 977
clear_subdir_cache() (*in module conda.testing.fixtures*), 977
client_ssl_cert (*Context attribute*), 623
client_ssl_cert_key (*Context attribute*), 623
clobber (*Context attribute*), 621
clobber (*PathConflict attribute*), 616
ClobberError, 786
clone() (*in module conda.cli.install*), 646
clone_env() (*in module conda.misc*), 841
close() (*EnforceUnusedAdapter method*), 807
close() (*FTPAdapter method*), 801
close() (*JSONProgressBar method*), 938
close() (*LocalFSAdapter method*), 804
close() (*ProgressBarBase method*), 960
close() (*QuietProgressBar method*), 936
close() (*S3Adapter method*), 805
close() (*ShardCache method*), 557
close() (*TQDMProgressBar method*), 936
CMD_LINE_SOURCE (*in module conda.base.constants*), 619
CmdExeActivator (*class in conda.activate*), 582
coerce() (*ErrorGuidance class method*), 556
coerce() (*NoarchType static method*), 856
collect_all() (*Configuration method*), 685
collect_errors() (*LoadedParameter method*), 676
collect_errors() (*MapLoadedParameter method*), 678
collect_errors() (*ObjectLoadedParameter method*), 679
collect_errors() (*SequenceLoadedParameter method*), 678
com_pat (*History attribute*), 837
Combine() (*Clauses method*), 666
combine_batches_until_none() (*in module conda._private.shards.misc*), 560
combined_depends (*PackageRecord property*), 880
comma_separated_stripped() (*in module conda.cli.helpers*), 645
command_join (*_Activator attribute*), 578
command_join (*CmdExeActivator attribute*), 582
command_join (*CshActivator attribute*), 581
command_join (*FishActivator attribute*), 583
command_join (*JSONFormatMixin attribute*), 584
command_join (*PosixActivator attribute*), 580
command_join (*PowerShellActivator attribute*), 584
command_join (*XonshActivator attribute*), 582
CommandNotFoundError, 787
Commands, **116**
Commands (*class in conda.testing.integration*), 988
commands (*in module conda.instructions*), 840
commit_id (*in module conda._version*), 577
COMPACT_DURATION_UNITS (*in module conda.common.datetime*), 686
COMPARE (*Commands attribute*), 988
COMPARE_OP (*in module conda.plugins.prefix_data_loaders.pypi.pkg_format*), 931
compare_packages() (*in module conda.cli.main_compare*), 650
compatible_release_operator() (*in module conda.models.version*), 887
COMPATIBLE_SHELLS (*in module conda.base.constants*), 614
compile_action_groups (*PrefixActionGroup attribute*), 727
compile_multiple_pyc() (*in module conda.gateways.disk.create*), 811
CompileMultiPycAction (*class in conda.core.path_actions*), 740
ComposableField (*class in conda.auxlib.entity*), 605
compressed_shard (*AnnotatedRawShard attribute*), 557
compute_sum() (*in module conda.gateways.disk.read*), 814
conda
    module, 553
conda config and context, **484**
Conda for data scientists, **116**
conda init and conda activate, **484**
conda install, **484**
conda.__main__
    module, 553
conda._preview
    module, 553
conda._preview.env_setup
    module, 553

conda._preview.env_setup.cli
    module, 554
conda._preview.env_setup.cli.main_create
    module, 554
conda._preview.env_setup.cli.main_install
    module, 554
conda._private
    module, 554
conda._private.exception_guidance
    module, 554
conda._private.extract
    module, 556
conda._private.shards
    module, 556
conda._private.shards.cache
    module, 556
conda._private.shards.decompression
    module, 558
conda._private.shards.misc
    module, 558
conda._private.shards.shards
    module, 561
conda._private.shards.subset
    module, 566
conda._private.shards.typing
    module, 572
conda._private.zstd
    module, 577
conda._version
    module, 577
conda.activate
    module, 577
conda.api
    module, 585
conda.auxlib
    module, 591
conda.auxlib.collection
    module, 591
conda.auxlib.compat
    module, 592
conda.auxlib.decorators
    module, 593
conda.auxlib.entity
    module, 595
conda.auxlib.exceptions
    module, 607
conda.auxlib.ish
    module, 607
conda.auxlib.logz
    module, 608
conda.auxlib.type_coercion
    module, 609
conda.base
    module, 611
conda.base.constants
    module, 611
conda.base.context
    module, 619
conda.cli
    module, 629
conda.cli.actions
    module, 629
conda.cli.common
    module, 631
conda.cli.conda_argparse
    module, 632
conda.cli.condarc
    module, 636
conda.cli.find_commands
    module, 641
conda.cli.formats
    module, 641
conda.cli.helpers
    module, 642
conda.cli.install
    module, 645
conda.cli.main
    module, 648
conda.cli.main_clean
    module, 648
conda.cli.main_commands
    module, 649
conda.cli.main_compare
    module, 649
conda.cli.main_config
    module, 650
conda.cli.main_create
    module, 651
conda.cli.main_env
    module, 652
conda.cli.main_env_config
    module, 652
conda.cli.main_env_create
    module, 652
conda.cli.main_env_list
    module, 653
conda.cli.main_env_remove
    module, 653
conda.cli.main_env_update
    module, 653
conda.cli.main_env_vars
    module, 654
conda.cli.main_export
    module, 654
conda.cli.main_info
    module, 655
conda.cli.main_init
    module, 657

conda.cli.main_install
    module, 658
conda.cli.main_list
    module, 658
conda.cli.main_mock_activate
    module, 659
conda.cli.main_mock_deactivate
    module, 659
conda.cli.main_notices
    module, 659
conda.cli.main_package
    module, 660
conda.cli.main_pip
    module, 661
conda.cli.main_remove
    module, 661
conda.cli.main_rename
    module, 661
conda.cli.main_run
    module, 662
conda.cli.main_search
    module, 662
conda.cli.main_update
    module, 663
conda.common
    module, 663
conda.common._logic
    module, 663
conda.common._os
    module, 667
conda.common._os.linux
    module, 667
conda.common._os.osx
    module, 667
conda.common._os.unix
    module, 668
conda.common._os.windows
    module, 668
conda.common.compat
    module, 670
conda.common.configuration
    module, 671
conda.common.constants
    module, 685
conda.common.datetime
    module, 686
conda.common.io
    module, 686
conda.common.iterators
    module, 691
conda.common.logic
    module, 692
conda.common.path
    module, 694
conda.common.path._cygpath
    module, 695
conda.common.path.directories
    module, 696
conda.common.path.python
    module, 696
conda.common.path.windows
    module, 697
conda.common.serialize
    module, 700
conda.common.serialize.json
    module, 700
conda.common.serialize.yaml
    module, 702
conda.common.signals
    module, 704
conda.common.terminal
    module, 704
conda.common.toposort
    module, 706
conda.common.url
    module, 706
conda.core
    module, 711
conda.core.envs_manager
    module, 711
conda.core.exclude_newer
    module, 713
conda.core.index
    module, 715
conda.core.initialize
    module, 719
conda.core.launchers
    module, 724
conda.core.link
    module, 725
conda.core.package_cache_data
    module, 729
conda.core.path_actions
    module, 732
conda.core.portability
    module, 751
conda.core.prefix_data
    module, 755
conda.core.solve
    module, 759
conda.core.subdir_data
    module, 765
conda.deprecations
    module, 770
conda.env
    module, 773
conda.env.env
    module, 773

conda.env.installers
    module, 776
conda.env.installers.base
    module, 776
conda.env.installers.conda
    module, 777
conda.env.installers.pip
    module, 778
conda.env.pip_util
    module, 779
conda.env.specs
    module, 779
conda.env.specs.cep_24
    module, 779
conda.env.specs.explicit
    module, 780
conda.env.specs.requirements
    module, 781
conda.env.specs.yaml_file
    module, 781
conda.exception_handler
    module, 783
conda.exceptions
    module, 784
conda.exports
    module, 798
conda.gateways
    module, 800
conda.gateways.anaconda_client
    module, 800
conda.gateways.connection
    module, 801
conda.gateways.connection.adapters
    module, 801
conda.gateways.connection.adapters.ftp
    module, 801
conda.gateways.connection.adapters.http
    module, 802
conda.gateways.connection.adapters.localfs
    module, 803
conda.gateways.connection.adapters.s3
    module, 804
conda.gateways.connection.download
    module, 805
conda.gateways.connection.session
    module, 806
conda.gateways.disk
    module, 809
conda.gateways.disk.create
    module, 809
conda.gateways.disk.delete
    module, 811
conda.gateways.disk.link
    module, 812
conda.gateways.disk.lock
    module, 812
conda.gateways.disk.permissions
    module, 813
conda.gateways.disk.read
    module, 813
conda.gateways.disk.test
    module, 815
conda.gateways.disk.update
    module, 815
conda.gateways.logging
    module, 816
conda.gateways.repodata
    module, 818
conda.gateways.repodata.lock
    module, 818
conda.gateways.repodata.zstd
    module, 818
conda.gateways.shards
    module, 830
conda.gateways.shards.typing
    module, 830
conda.gateways.streams
    module, 835
conda.gateways.subprocess
    module, 836
conda.history
    module, 836
conda.instructions
    module, 838
conda.misc
    module, 840
conda.models
    module, 841
conda.models.build_string
    module, 841
conda.models.channel
    module, 843
conda.models.dist
    module, 848
conda.models.enums
    module, 850
conda.models.environment
    module, 856
conda.models.match_spec
    module, 860
conda.models.package_info
    module, 869
conda.models.prefix_graph
    module, 870
conda.models.records
    module, 872
conda.models.version
    module, 883

conda.notices
    module, 889
conda.notices.cache
    module, 889
conda.notices.core
    module, 890
conda.notices.fetch
    module, 892
conda.notices.types
    module, 893
conda.notices.views
    module, 894
conda.plugins
    module, 895
conda.plugins.config
    module, 895
conda.plugins.environment_exporters
    module, 896
conda.plugins.environment_exporters.environment_yaml
    module, 896
conda.plugins.environment_exporters.explicit
    module, 897
conda.plugins.environment_exporters.requirements_txt
    module, 898
conda.plugins.environment_specifiers
    module, 898
conda.plugins.environment_specifiers.cep_24
    module, 898
conda.plugins.environment_specifiers.environment_yaml
    module, 899
conda.plugins.environment_specifiers.explicit
    module, 899
conda.plugins.environment_specifiers.requirements_txt
    module, 899
conda.plugins.hookspec
    module, 899
conda.plugins.manager
    module, 913
conda.plugins.package_extractors
    module, 923
conda.plugins.package_extractors.conda
    module, 923
conda.plugins.post_solves
    module, 923
conda.plugins.prefix_data_loaders
    module, 923
conda.plugins.prefix_data_loaders.pypi
    module, 923
conda.plugins.prefix_data_loaders.pypi.pkg_format
    module, 923
conda.plugins.previews
    module, 935
conda.plugins.reporter_backends
    module, 935
conda.plugins.reporter_backends.console
    module, 935
conda.plugins.reporter_backends.json
    module, 937
conda.plugins.solvers
    module, 939
conda.plugins.subcommands
    module, 939
conda.plugins.subcommands.doctor
    module, 939
conda.plugins.subcommands.doctor.health_checks
    module, 939
conda.plugins.subcommands.doctor.health_checks.altered_fil
    module, 939
conda.plugins.subcommands.doctor.health_checks.consistency
    module, 940
conda.plugins.subcommands.doctor.health_checks.environment
    module, 940
conda.plugins.subcommands.doctor.health_checks.file_lockin
    module, 941
conda.plugins.subcommands.doctor.health_checks.missing_fil
    module, 941
conda.plugins.subcommands.doctor.health_checks.pinned
    module, 942
conda.plugins.subcommands.doctor.health_checks.requests_ca
    module, 943
conda.plugins.subcommands.doctor.health_checks.utils
    module, 943
conda.plugins.subcommands.plugins
    module, 949
conda.plugins.subcommands.plugins.info
    module, 949
conda.plugins.subcommands.plugins.install
    module, 950
conda.plugins.subcommands.plugins.list
    module, 950
conda.plugins.subcommands.plugins.package_validation
    module, 951
conda.plugins.subcommands.plugins.remove
    module, 951
conda.plugins.subcommands.plugins.update
    module, 952
conda.plugins.types
    module, 953
conda.plugins.virtual_packages
    module, 967
conda.plugins.virtual_packages.archspec
    module, 967
conda.plugins.virtual_packages.conda
    module, 968
conda.plugins.virtual_packages.cuda
    module, 968
conda.plugins.virtual_packages.freebsd
    module, 969

conda.plugins.virtual_packages.linux
    module, 969
conda.plugins.virtual_packages.osx
    module, 969
conda.plugins.virtual_packages.windows
    module, 970
conda.reporters
    module, 970
conda.resolve
    module, 971
conda.testing
    module, 975
conda.testing.cases
    module, 975
conda.testing.fixtures
    module, 976
conda.testing.gateways
    module, 983
conda.testing.gateways.fixtures
    module, 983
conda.testing.helpers
    module, 984
conda.testing.http_test_server
    module, 987
conda.testing.integration
    module, 988
conda.testing.notices
    module, 989
conda.testing.notices.fixtures
    module, 989
conda.testing.notices.helpers
    module, 989
conda.testing.solver_helpers
    module, 991
conda.utils
    module, 994
conda_auth_handlers() (*CondaSpecs method*), 902
conda_auth_handlers() (*in module conda.plugins.hookspec.CondaSpecs*), 510
conda_build (*Context property*), 625
conda_build (*in module conda.exports*), 799
conda_build_form() (*MatchSpec method*), 864
conda_build_local_paths (*Context property*), 625
conda_build_local_urls (*Context property*), 625
conda_cli (*TmpChannelFixture attribute*), 981
conda_cli (*TmpEnvFixture attribute*), 980
conda_cli() (*in module conda.testing.fixtures*), 978
conda_env_form() (*MatchSpec method*), 864
CONDA_ENV_VARS_UNSET_VAR (*in module conda.base.constants*), 619
conda_environment_exporters() (*CondaSpecs method*), 911
conda_environment_exporters() (*in module conda.plugins.environment_exporters.environment_yml*), 897
conda_environment_exporters() (*in module conda.plugins.environment_exporters.explicit*), 897
conda_environment_exporters() (*in module conda.plugins.environment_exporters.requirements_txt*), 898
conda_environment_exporters() (*in module conda.plugins.hookspec.CondaSpecs*), 514
conda_environment_specifiers() (*CondaSpecs method*), 910
conda_environment_specifiers() (*in module conda.plugins.environment_specifiers.cep_24*), 898
conda_environment_specifiers() (*in module conda.plugins.environment_specifiers.environment_yml*), 899
conda_environment_specifiers() (*in module conda.plugins.environment_specifiers.explicit*), 899
conda_environment_specifiers() (*in module conda.plugins.environment_specifiers.requirements_txt*), 899
conda_environment_specifiers() (*in module conda.plugins.hookspec.CondaSpecs*), 521
conda_error_hints() (*CondaSpecs method*), 908
conda_error_hints() (*in module conda.plugins.hookspec.CondaSpecs*), 512
conda_exception_handler() (*in module conda.exception_handler*), 784
conda_exception_observers() (*CondaSpecs method*), 913
conda_exception_observers() (*in module conda.plugins.hookspec.CondaSpecs*), 531
conda_exe_vars_dict (*Context property*), 626
conda_health_checks() (*CondaSpecs method*), 903
conda_health_checks() (*in module conda.plugins.hookspec.CondaSpecs*), 535
conda_health_checks() (*in module conda.plugins.subcommands.doctor.health_checks.altered_files*), 939
conda_health_checks() (*in module conda.plugins.subcommands.doctor.health_checks.consistency*), 940
conda_health_checks() (*in module conda.plugins.subcommands.doctor.health_checks.environment_t*
941
conda_health_checks() (*in module conda.plugins.subcommands.doctor.health_checks.file_locking*), 941
conda_health_checks() (*in module conda.plugins.subcommands.doctor.health_checks.missing_files*), 942
conda_health_checks() (*in module*

conda.plugins.subcommands.doctor.health_checks.pinned), *method*), 904
942
conda_health_checks() (*in module conda.plugins.subcommands.doctor.health_checks*), 943
CONDA_HOMEPAGE_URL (*in module conda.base.constants*), 613
CONDA_HOMEPAGE_URL (*in module conda.gateways.repodata*), 821
conda_http_errors() (*in module conda.gateways.repodata*), 827
CONDA_INITIALIZE_PS_RE_BLOCK (*in module conda.core.initialize*), 722
CONDA_INITIALIZE_RE_BLOCK (*in module conda.core.initialize*), 722
conda_installed_files() (*in module conda.misc*), 840
CONDA_LIST_FIELDS (*in module conda.base.constants*), 618
CONDA_LOGS_DIR (*in module conda.base.constants*), 614
conda_move_to_front_of_PATH() (*in module conda.testing*), 994
conda_name (*PythonDistribution property*), 926
conda_notices_args_n_parser() (*in module conda.testing.notices.fixtures*), 989
CONDA_PACKAGE_EXTENSION_V1 (*in module conda.base.constants*), 614
CONDA_PACKAGE_EXTENSION_V2 (*in module conda.base.constants*), 614
CONDA_PACKAGE_EXTRACTOR_NAME (*in module conda.base.constants*), 614
conda_package_extractors() (*CondaSpecs method*), 912
conda_package_extractors() (*in module conda.plugins.hookspec.CondaSpecs*), 536
conda_package_extractors() (*in module conda.plugins.package_extractors.conda*), 923
CONDA_PACKAGE_ROOT (*in module conda*), 996
conda_package_types() (*PackageType static method*), 855
conda_post_commands() (*CondaSpecs method*), 902
conda_post_commands() (*in module conda.plugins.hookspec.CondaSpecs*), 537
conda_post_solves() (*CondaSpecs method*), 906
conda_post_transaction_actions() (*CondaSpecs method*), 905
conda_post_transaction_actions() (*in module conda.plugins.hookspec.CondaSpecs*), 541
conda_pre_commands() (*CondaSpecs method*), 901
conda_pre_commands() (*in module conda.plugins.hookspec.CondaSpecs*), 538
conda_pre_solves() (*CondaSpecs method*), 905
conda_pre_transaction_actions() (*CondaSpecs method*), 904
conda_pre_transaction_actions() (*in module conda.plugins.hookspec.CondaSpecs*), 539
conda_prefix (*Context property*), 626
conda_prefix_data_loaders() (*CondaSpecs method*), 909
conda_prefix_data_loaders() (*in module conda.plugins.hookspec.CondaSpecs*), 542
conda_prefix_data_loaders() (*in module conda.plugins.prefix_data_loaders.pypi*), 935
conda_reporter_backends() (*CondaSpecs method*), 907
conda_reporter_backends() (*in module conda.plugins.reporter_backends.console*), 937
conda_reporter_backends() (*in module conda.plugins.reporter_backends.json*), 938
conda_request_headers() (*CondaSpecs method*), 908
conda_request_headers() (*in module conda.plugins.hookspec.CondaSpecs*), 545
conda_session_headers() (*CondaSpecs method*), 908
conda_session_headers() (*in module conda.plugins.hookspec.CondaSpecs*), 546
CONDA_SESSION_SCHEMES (*in module conda.gateways.connection.session*), 807
conda_settings() (*CondaSpecs method*), 906
conda_settings() (*in module conda.plugins.hookspec.CondaSpecs*), 547
conda_signal_handler() (*in module conda*), 997
conda_solvers() (*CondaSpecs method*), 900
conda_solvers() (*in module conda.plugins.hookspec.CondaSpecs*), 547
conda_solvers() (*in module conda.plugins.solvers*), 939
conda_subcommands() (*CondaSpecs method*), 900
conda_subcommands() (*in module conda._preview.env_setup.cli.main_create*), 554
conda_subcommands() (*in module conda._preview.env_setup.cli.main_install*), 554
conda_subcommands() (*in module conda.plugins.hookspec.CondaSpecs*), 549
conda_subcommands() (*in module conda.plugins.previews*), 935
conda_subcommands() (*in module conda.plugins.subcommands.doctor*), 949
conda_subcommands() (*in module conda.plugins.subcommands.plugins*), 953
CONDA_TARBALL_EXTENSION (*in module conda.base.constants*), 614
CONDA_TEMP_EXTENSION (*in module conda.base.constants*), 614

CONDA_TEMP_EXTENSIONS (*in module conda.base.constants*), 614
conda_tests_ctxt_mgmt_def_pol (*in module conda.base.context*), 629
conda_v_pat (*History attribute*), 837
conda_version (*CondaExceptionEvent attribute*), 530, 966
conda_virtual_packages() (*CondaSpecs method*), 901
conda_virtual_packages() (*in module conda.plugins.hookspec.CondaSpecs*), 550
conda_virtual_packages() (*in module conda.plugins.virtual_packages.archspec*), 967
conda_virtual_packages() (*in module conda.plugins.virtual_packages.conda*), 968
conda_virtual_packages() (*in module conda.plugins.virtual_packages.cuda*), 968
conda_virtual_packages() (*in module conda.plugins.virtual_packages.freebsd*), 969
conda_virtual_packages() (*in module conda.plugins.virtual_packages.linux*), 969
conda_virtual_packages() (*in module conda.plugins.virtual_packages.osx*), 969
conda_virtual_packages() (*in module conda.plugins.virtual_packages.windows*), 970
CondaAuthHandler (*class in conda.plugins.types*), 509, 957
CondaCLIFixture (*class in conda.testing.fixtures*), 978
CondaDependencyError, 794, 823
CondaEnvException, 795
CondaEnvironmentError, 787
CondaEnvironmentExporter (*class in conda.plugins.types*), 513, 964
CondaEnvironmentSpecifier (*class in conda.plugins.types*), 963
CondaError, 821, 996
CondaErrorHint (*class in conda.plugins.types*), 512, 961
CondaExceptionEvent (*class in conda.plugins.types*), 529, 965
CondaExceptionObserver (*class in conda.plugins.types*), 531, 966
CondaExitZero, 997
CondaExportWarning, 654
CondaFileIOError, 789
CondaHealthCheck (*class in conda.plugins.types*), 534, 958
CondaHistoryError, 793
CondaHistoryWarning, 837
CondaHttpAuth (*class in conda.gateways.connection.session*), 808
CondaHTTPError, 790, 823
CondaImportError, 790
CondaIndexError, 793
CondaIOError, 789
CondaJSONEncoder (*class in conda.common.serialize.json*), 701
CondaKeyError, 789
CondaMemoryError, 794
CondaMultiError, 996
CondaOSError, 788
CondaOSError (*in module conda.exports*), 799
CondaPackageExtractor (*class in conda.plugins.types*), 536, 965
CondaPlugin (*class in conda.plugins.types*), 955
CondaPluginManager (*class in conda.plugins.manager*), 914
CondaPostCommand (*class in conda.plugins.types*), 537, 957
CondaPostSolve (*class in conda.plugins.types*), 959
CondaPostTransactionAction (*class in conda.plugins.types*), 540, 962
CondaPreCommand (*class in conda.plugins.types*), 538, 956
CondaPrefixDataLoader (*class in conda.plugins.prefix_data_loaders.pypi*), 934
CondaPrefixDataLoader (*class in conda.plugins.types*), 542, 962
CondaPrefixDataLoaderCallable (*in module conda.plugins.types*), 954
CondaPreSolve (*class in conda.plugins.types*), 958
CondaPreTransactionAction (*class in conda.plugins.types*), 539, 961
CONDARC_FILENAMES (*in module conda.common.configuration*), 683
CondaRepoInterface (*class in conda.gateways.repodata*), 827
CondaReporterBackend (*class in conda.plugins.types*), 960
CondaRequestHeader (*class in conda.plugins.types*), 546, 961
CondaSession (*class in conda.gateways.connection.session*), 808
CondaSessionType (*class in conda.gateways.connection.session*), 808
CondaSetting (*class in conda.plugins.types*), 546, 959
CondaSignalInterrupt, 785
CondaSolver (*class in conda.plugins.types*), 547, 956
CondaSpecs (*class in conda.plugins.hookspec*), 900
CondaSSLError, 790, 823
CondaSubcommand (*class in conda.plugins.subcommands.doctor*), 948
CondaSubcommand (*class in conda.plugins.subcommands.plugins*), 952

CondaSubcommand (*class in conda.plugins.types*), 548, 955
CondaSystemExit, 788, 948
CondaUpdatePackageError, 798
CondaUpgradeError, 793
CondaValueError, 793
CondaVerificationError, 793
CondaVirtualPackage (*class in conda.plugins.types*), 549, 955
CONFIG (*Commands attribute*), 988
config (*Environment attribute*), 858
config_files (*Context property*), 626
Configuration (*class in conda.common.configuration*), 684
CONFIGURATION_SOURCES (*in module conda.base.constants*), 619
ConfigurationError, 672
ConfigurationFile (*class in conda.cli.condarc*), 638
ConfigurationLoadError, 673
ConfigurationObject (*class in conda.common.configuration*), 679
ConfigurationType (*class in conda.common.configuration*), 683
configure_parser (*CondaSubcommand attribute*), 549, 949, 953, 955
configure_parser() (*in module conda.cli.main_clean*), 648
configure_parser() (*in module conda.cli.main_commands*), 649
configure_parser() (*in module conda.cli.main_compare*), 650
configure_parser() (*in module conda.cli.main_config*), 650
configure_parser() (*in module conda.cli.main_create*), 652
configure_parser() (*in module conda.cli.main_env*), 652
configure_parser() (*in module conda.cli.main_env_config*), 652
configure_parser() (*in module conda.cli.main_env_create*), 653
configure_parser() (*in module conda.cli.main_env_list*), 653
configure_parser() (*in module conda.cli.main_env_remove*), 653
configure_parser() (*in module conda.cli.main_env_update*), 654
configure_parser() (*in module conda.cli.main_env_vars*), 654
configure_parser() (*in module conda.cli.main_export*), 654
configure_parser() (*in module conda.cli.main_info*), 655
configure_parser() (*in module conda.cli.main_init*), 658
configure_parser() (*in module conda.cli.main_install*), 658
configure_parser() (*in module conda.cli.main_list*), 658
configure_parser() (*in module conda.cli.main_mock_activate*), 659
configure_parser() (*in module conda.cli.main_mock_deactivate*), 659
configure_parser() (*in module conda.cli.main_notices*), 659
configure_parser() (*in module conda.cli.main_package*), 660
configure_parser() (*in module conda.cli.main_remove*), 661
configure_parser() (*in module conda.cli.main_rename*), 662
configure_parser() (*in module conda.cli.main_run*), 662
configure_parser() (*in module conda.cli.main_search*), 662
configure_parser() (*in module conda.cli.main_update*), 663
configure_parser() (*in module conda.plugins.subcommands.doctor*), 949
configure_parser() (*in module conda.plugins.subcommands.plugins*), 953
configure_parser() (*in module conda.plugins.subcommands.plugins.info*), 950
configure_parser() (*in module conda.plugins.subcommands.plugins.install*), 950
configure_parser() (*in module conda.plugins.subcommands.plugins.list*), 950
configure_parser() (*in module conda.plugins.subcommands.plugins.remove*), 951
configure_parser() (*in module conda.plugins.subcommands.plugins.update*), 952
configure_parser_plugins() (*in module conda.cli.conda_argparse*), 636
Configuring conda, **137**
Configuring Temporary File Locations, **84**
confirm_yn() (*in module conda.plugins.subcommands.doctor*), 948
confirm_yn() (*in module conda.reporters*), 971
connect() (*in module conda._private.shards.cache*), 557
connect() (*ShardCache method*), 557
consistent_env_check() (*in module conda.plugins.subcommands.doctor.health_checks.consistency*), 940

console (*Context property*), 627
ConsoleReporterRenderer (*class in conda.plugins.reporter_backends.console*), 937
constant() (*DeprecationHandler method*), 772
constrains (*PackageRecord attribute*), 880
constrains (*PackageRecordDict attribute*), 572
constraints (*PySpec attribute*), 925
construct_states() (*History method*), 838
contains() (*CondaMultiError method*), 997
content (*ConfigurationFile property*), 639
ContentTypeOptions (*in module conda.core.initialize*), 722
Context (*class in conda.base.context*), 621
context (*ConfigurationFile property*), 639
context (*in module conda.base.context*), 629
context (*in module conda.env.specs*), 782
context (*in module conda.gateways.repodata*), 822
context (*in module conda.plugins.subcommands.doctor*), 944
context_aware_monkeypatch() (*in module conda.testing.fixtures*), 982
context_params (*ConfigurationFile property*), 639
context_stack (*in module conda.base.context*), 628
context_testdata() (*in module conda.testing.fixtures*), 982
ContextDecorator (*class in conda.common.io*), 688
ContextStack (*class in conda.base.context*), 628
ContextStackObject (*class in conda.base.context*), 628
Contributing 101, **416**
convert_to_dist_str() (*in module conda.testing.helpers*), 987
copy (*LinkType attribute*), 854
copy() (*in module conda.gateways.disk.create*), 811
copy() (*Index method*), 718
copy() (*QueueCache method*), 570
copy() (*ShardCache method*), 557
copy_acl() (*in module conda.common._os.osx*), 668
CorruptedEnvironmentError, 793
CouldntParseError, 790
CREATE (*Commands attribute*), 988
create_actions() (*CompileMultiPycAction class method*), 740
create_actions() (*CreatePrefixRecordAction class method*), 743
create_actions() (*CreatePythonEntryPointAction class method*), 742
create_actions() (*MakeMenuAction class method*), 739
create_actions() (*RemoveMenuAction class method*), 747
create_actions() (*UpdateHistoryAction class method*), 743
create_cache_dir() (*in module conda.gateways.repodata*), 830
create_conda_pkg() (*in module conda.cli.main_package*), 660
create_default_packages (*Context property*), 627
create_directory_actions() (*LinkPathAction class method*), 738
create_envs_directory() (*in module conda.gateways.disk.create*), 811
create_file_link_actions() (*LinkPathAction class method*), 738
create_hard_link_or_copy() (*in module conda.gateways.disk.create*), 811
create_info() (*in module conda.cli.main_package*), 660
create_link() (*in module conda.gateways.disk.create*), 811
create_notice_cache_files() (*in module conda.testing.notices.helpers*), 990
create_package_cache_directory() (*in module conda.gateways.disk.create*), 811
create_python_entry_point() (*in module conda.gateways.disk.create*), 811
create_python_entry_point_windows_exe_action() (*LinkPathAction class method*), 738
created (*PrefixData property*), 758, 947
created_at (*ChannelNotice attribute*), 893
CreateInPrefixPathAction (*class in conda.core.path_actions*), 736
CreatePrefixRecordAction (*class in conda.core.path_actions*), 742
CreatePythonEntryPointAction (*class in conda.core.path_actions*), 741
Creating custom channels, **57**
Creating projects with conda, **57**
CRITICAL (*NoticeLevel attribute*), 618
croot (*Context property*), 625
CshActivator (*class in conda.activate*), 581
cuda_version() (*in module conda.plugins.virtual_packages.cuda*), 968
custom_channels() (*Context method*), 626
custom_expandvars() (*in module conda.common.configuration*), 683
custom_multichannels() (*Context method*), 626
CustomValidationError, 673
cutoff (*_ChannelCutoff attribute*), 714
cutoff() (*_CutoffParser method*), 713
CyclicalDependencyError, 793

D

dals() (*in module conda.auxlib.ish*), 607
dashlist() (*in module conda.common.io*), 688
data (*Index property*), 717

data_callback_factory() (*in module conda.gateways.connection.adapters.ftp*), 802
date (*PackageRecord attribute*), 880
DateField (*class in conda.auxlib.entity*), 603
DDE_BUSY (*ERROR attribute*), 669
DDE_FAIL (*ERROR attribute*), 669
DDE_TIMEOUT (*ERROR attribute*), 669
deactivate() (*_Activator method*), 579
DeactivateHelp, 785
debug (*Context property*), 627
DEBUG_FORMATTER (*in module conda.auxlib.logz*), 608
decompress_shard() (*in module conda._private.shards.decompression*), 558
Deep dives, **416**
default (*Field property*), 601
default (*Parameter property*), 680
default() (*CondaJSONEncoder method*), 702
default_activation_env (*Context property*), 627
default_activation_prefix (*Context property*), 627
DEFAULT_AGGRESSIVE_UPDATE_PACKAGES (*in module conda.base.constants*), 614
DEFAULT_CHANNEL_ALIAS (*in module conda.base.constants*), 613
DEFAULT_CHANNELS (*in module conda.base.constants*), 613
default_channels() (*Context method*), 626
DEFAULT_CHANNELS_UNIX (*in module conda.base.constants*), 613
DEFAULT_CHANNELS_WIN (*in module conda.base.constants*), 613
DEFAULT_CONDA_LIST_FIELDS (*in module conda.base.constants*), 618
DEFAULT_CONDARC_FILENAME (*in module conda.common.configuration*), 683
DEFAULT_CONSOLE_REPORTER_BACKEND (*in module conda.base.constants*), 618
DEFAULT_CUSTOM_CHANNELS (*in module conda.base.constants*), 613
default_filenames (*CondaEnvironmentExporter attribute*), 513, 964
default_filenames (*CondaEnvironmentSpecifier attribute*), 964
default_filter() (*Resolve method*), 972
default_in_dump (*Field property*), 601
DEFAULT_JSON_REPORTER_BACKEND (*in module conda.base.constants*), 618
DEFAULT_MARKER_CONTEXT (*in module conda.plugins.prefix_data_loaders.pypi.pkg_format*), 931
DEFAULT_NOTICE_MESG (*in module conda.testing.notices.helpers*), 990
default_prefix (*Context property*), 626
default_prefix (*in module conda.exports*), 799
default_python (*Context attribute*), 621
default_python (*in module conda.exports*), 799
default_python_default() (*in module conda.base.context*), 621
default_python_validation() (*in module conda.base.context*), 621
DEFAULT_SOLVER (*in module conda.base.constants*), 618
DEFAULT_STRATEGY (*RepodataSubset attribute*), 568, 832
default_threads (*Context property*), 625
DEFAULTS_CHANNEL_NAME (*in module conda.base.constants*), 613
DefaultValueRawParameter (*class in conda.common.configuration*), 676
delete_prefix_from_linked_data() (*in module conda.core.prefix_data*), 759
delete_trash() (*in module conda.gateways.disk.delete*), 812
DeltaSecondsFormatter (*class in conda.common.io*), 687
denylist_channels (*Context attribute*), 623
Dependencies, **54**
Dependencies (*class in conda.env.env*), 776
dependencies (*EnvironmentYaml attribute*), 776
dependencies_validation() (*in module conda.env.env*), 774
dependency_sort() (*Resolve method*), 974
depends (*PackageRecord attribute*), 880
depends (*PackageRecordDict attribute*), 572
deprecated (*in module conda.common.serialize*), 703
deprecated (*in module conda.deprecations*), 773
deprecated (*in module conda.gateways.repodata*), 823
deprecated_import() (*in module conda.testing.helpers*), 985
DeprecatedError, 770
DeprecationHandler (*class in conda.deprecations*), 770
Deprecations, **417**
deprecations (*_ConstantDeprecationRegistry attribute*), 773
deps_modifier (*Context attribute*), 624
deps_modifier (*EnvironmentConfig attribute*), 857
deps_modifier (*SolverStateContainer attribute*), 764
DepsModifier (*class in conda.base.constants*), 616
DepsModifier (*in module conda.api*), 585
depth (*RepodataSubset attribute*), 568, 832
describe_all_parameters() (*in module conda.cli.main_config*), 651
describe_parameter() (*Configuration method*), 685
describe_parameter() (*PluginConfig method*), 896
description (*CondaEnvironmentExporter attribute*), 513, 964
description (*CondaEnvironmentSpecifier attribute*), 964

description (*CondaReporterBackend attribute*), 961
description (*CondaSetting attribute*), 547, 959
description (*ProgressBarBase attribute*), 960
description_map() (*Context method*), 628
detach_stderr() (*in module conda.auxlib.logz*), 608
detail_view() (*ConsoleReporterRenderer method*), 937
detail_view() (*JSONReporterRenderer method*), 938
detail_view() (*ReporterRendererBase method*), 960
detect() (*in module conda.env.specs*), 783
detect_environment_exporter() (*CondaPluginManager method*), 920
detect_environment_specifier() (*CondaPluginManager method*), 919
detection_supported (*EnvironmentSpecBase attribute*), 962
detection_supported (*YamlFileSpec attribute*), 782
determine_constricting_specs() (*Solver method*), 763
determine_link_type() (*in module conda.core.link*), 725
determine_target_prefix() (*in module conda.base.context*), 629
dev (*Context property*), 625
Development environment, **416**
diff_for_unlink_link_precs() (*in module conda.core.solve*), 764
direct_download() (*DirectDownloadAdapter method*), 809
direct_download() (*S3Adapter method*), 805
DirectDownloadAdapter (*class in conda.gateways.connection*), 809
directory (*HttpTestServerFixture attribute*), 982
directory (*LinkType attribute*), 854
directory (*PathEnum attribute*), 854
DirectoryNotACondaEnvironmentError, 787
DirectoryNotFoundError, 787
Disable SSL Verification, **84**
disable_external_plugins() (*CondaPluginManager method*), 917
disable_logger() (*in module conda.common.io*), 690
disable_plugins() (*CondaPluginManager method*), 917
disable_ssl_verify_warning() (*in module conda.gateways.connection.download*), 806
DISABLED (*ChannelPriority attribute*), 618
disabled (*SafetyChecks attribute*), 615
disallowed_packages (*Context attribute*), 622
disallowed_packages (*EnvironmentConfig attribute*), 857
DisallowedPackageError, 792
disp_features() (*in module conda.cli.common*), 631
display_notices() (*in module conda.notices.core*), 891
Dist (*class in conda.models.dist*), 848
dist_fields_dump() (*PackageRecord method*), 880
dist_name (*Dist attribute*), 849
dist_name (*DistDetails attribute*), 848
dist_str() (*MatchSpec method*), 863
dist_str() (*PackageInfo method*), 870
dist_str() (*PackageRecord method*), 879
dist_str_in_index() (*in module conda.core.index*), 719
dist_str_to_quad() (*in module conda.models.dist*), 850
distance (*Node attribute*), 567
DistDetails (*class in conda.models.dist*), 848
DistType (*class in conda.models.dist*), 848
DLL_NOT_FOUND (*ERROR attribute*), 669
do_cache_action() (*in module conda.core.package_cache_data*), 731
do_call() (*in module conda.cli.conda_argparse*), 633
do_cleanup() (*in module conda.core.package_cache_data*), 731
do_extract_action() (*in module conda.core.package_cache_data*), 731
do_reverse() (*in module conda.core.package_cache_data*), 731
done_callback() (*in module conda.core.package_cache_data*), 732
downgraded_precs (*ChangeReport attribute*), 727
download() (*in module conda.exports*), 800
download() (*in module conda.gateways.connection.download*), 806
download_and_extract() (*UnlinkLinkTransaction method*), 727
download_http_errors() (*in module conda.gateways.connection.download*), 806
download_inner() (*in module conda.gateways.connection.download*), 806
download_only (*Context attribute*), 622
download_partial_file() (*in module conda.gateways.connection.download*), 806
download_repodata() (*in module conda.gateways.repodata.zstd*), 819
download_text() (*in module conda.gateways.connection.download*), 806
dry_run (*CondaExceptionEvent attribute*), 530, 966
dry_run (*Context attribute*), 624
dry_run() (*in module conda.env.installers.conda*), 777
DRY_RUN_PREFIX (*in module conda.base.constants*), 614
DryRunExit, 787, 948
DummyArgs (*class in conda.testing.notices.helpers*), 990
dump() (*_FeaturesField method*), 875
dump() (*Channel method*), 826, 846
dump() (*ChannelField method*), 876
dump() (*ComposableField method*), 605
dump() (*DateField method*), 603

dump() (*Entity method*), 606
dump() (*EnumField method*), 604
dump() (*Field method*), 601
dump() (*in module conda.common.serialize.json*), 702
dump() (*in module conda.common.serialize.yaml*), 703
dump() (*ListField method*), 604
dump() (*MultiChannel method*), 847
dump() (*SolvedRecord method*), 882
dump() (*TimestampField method*), 875
dump_map() (*CondaError method*), 821, 996
dump_map() (*CondaMultiError method*), 997
dump_record() (*in module conda.cli.main_info*), 656
dumps() (*in module conda.common.serialize.json*), 702
dumps() (*in module conda.common.serialize.yaml*), 703
duration_cutoff() (*_CutoffParser method*), 713

E

emit() (*StdStreamHandler method*), 817
empty_env() (*in module conda.testing.fixtures*), 981
EMPTY_LINK (*in module conda.models.records*), 875
EMPTY_MAP (*in module conda.common.configuration*), 672
empty_override (*CondaVirtualPackage attribute*), 550, 956
empty_prefix() (*in module conda.testing.solver_helpers*), 992
emscripten (*Platform attribute*), 852
enable_private_envs (*Context attribute*), 622
enabled (*JSONProgressBar attribute*), 938
enabled (*SafetyChecks attribute*), 615
enabled (*TQDMProgressBar attribute*), 936
ENCODE_ENVIRONMENT (*in module conda.common.compat*), 671
encode_environment() (*in module conda.common.compat*), 671
encode_for_env_var() (*in module conda.common.compat*), 671
EncodingError, 795
EnforceUnusedAdapter (*class in conda.gateways.connection.session*), 807
ensure_binary() (*in module conda.common.compat*), 671
ensure_dir_exists() (*in module conda.utils*), 995
ensure_hex_hashes() (*in module conda._private.shards.misc*), 560
ensure_text_type() (*in module conda.common.compat*), 671
ensure_update_specs_exist() (*in module conda.cli.install*), 647
ensure_utf8_encoding() (*in module conda.common.compat*), 671
Entity (*class in conda.auxlib.entity*), 605
entry_name (*time_recorder attribute*), 691
entry_point_action_groups (*PrefixActionGroup attribute*), 727
entry_points (*Noarch attribute*), 869
ENTRY_POINTS_FILES (*PythonDistribution attribute*), 925
ENTRY_POINTS_FILES (*PythonEggInfoDistribution attribute*), 927
ENTRY_POINTS_FILES (*PythonInstalledDistribution attribute*), 926
EnumField (*class in conda.auxlib.entity*), 604
env (*Cep24YamlFileSpec property*), 780
env (*EnvironmentSpecBase property*), 963
env (*ExplicitSpec property*), 780
env (*RequirementsSpec property*), 781
env (*YamlFileSpec property*), 782
env() (*SolverTests method*), 992
env_for() (*EnvironmentSpecBase method*), 963
env_name() (*in module conda.base.context*), 629
env_or_set (*in module conda.testing.integration*), 988
env_prompt (*Context attribute*), 621
env_txt_check() (*in module conda.plugins.subcommands.doctor.health_checks.environment_t*, 941
env_unmodified() (*in module conda.common.io*), 689
env_var() (*in module conda.common.io*), 689
env_vars() (*in module conda.common.io*), 689
ENV_VARS_SOURCE (*in module conda.base.constants*), 619
Environment (*class in conda.models.environment*), 858
environment (*EnvironmentFormat attribute*), 963
environment_context_keys (*Context property*), 628
environment_format (*CondaEnvironmentExporter attribute*), 513, 965
environment_format (*CondaEnvironmentSpecifier attribute*), 964
environment_is_consistent() (*Resolve method*), 974
ENVIRONMENT_JSON_FORMAT (*in module conda.plugins.environment_exporters.environment_yml*), 896
environment_settings (*Context property*), 628
environment_spec (*CondaEnvironmentSpecifier attribute*), 964
environment_specifier (*Context attribute*), 621
ENVIRONMENT_YAML_FORMAT (*in module conda.plugins.environment_exporters.environment_yml*), 896
EnvironmentConfig (*class in conda.models.environment*), 857
EnvironmentExporterNotDetected, 797
EnvironmentFileEmpty, 796
EnvironmentFileExtensionNotValid, 795
EnvironmentFileInvalid, 795
EnvironmentFileNotDownloaded, 796

EnvironmentFileNotFound, 795
EnvironmentFileTypeMismatchError, 796
EnvironmentFormat (*class in conda.plugins.types*), 963
EnvironmentFormatType (*in module conda.plugins.environment_exporters.environment_yml*), 896
EnvironmentIsFrozenError, 794
EnvironmentLocationNotFound, 787
EnvironmentNameNotFound, 787
EnvironmentNotWritableError, 794
Environments, **116**
EnvironmentSpecBase (*class in conda.plugins.types*), 962
EnvironmentSpecPluginNotDetected, 782, 797
EnvironmentSpecPluginSelectionError, 797
EnvironmentYaml (*class in conda.env.env*), 776
EnvRawParameter (*class in conda.common.configuration*), 675
envs_dirs (*Context property*), 626
envs_dirs (*in module conda.exports*), 799
envs_list() (*ConsoleReporterRenderer static method*), 937
envs_list() (*JSONReporterRenderer method*), 938
envs_list() (*ReporterRendererBase method*), 960
envvars_force_uppercase (*Context attribute*), 624
epilog() (*in module conda.cli.main_create*), 652
epilog() (*in module conda.cli.main_env_create*), 653
epilog() (*in module conda.cli.main_export*), 654
errno (*NotWritableError attribute*), 794, 823
ERROR (*class in conda.common._os.windows*), 669
ERROR_SNIPPET_LENGTH (*in module conda.gateways.repodata*), 826
error_upload_url (*Context property*), 625
error_upload_url (*ExceptionHandler property*), 783
ErrorGuidance (*class in conda._private.exception_guidance*), 555
errors (*CondaMultiError attribute*), 996
escape_for_winpath() (*in module conda.testing.integration*), 988
etag (*RepodataState property*), 827
ETAG_KEY (*in module conda.gateways.repodata*), 826
Eval() (*Clauses method*), 666
evaluate() (*Evaluator method*), 931
Evaluator (*class in conda.plugins.prefix_data_loaders.pypi.pkg_format*), 931
evaluator (*in module conda.plugins.prefix_data_loaders.pypi.pkg_format*), 931
exact_match() (*BaseSpec method*), 888
exact_value (*_StrMatchMixin property*), 867
exact_value (*BaseSpec property*), 888
exact_value (*BuildNumberMatch property*), 888
exact_value (*FeatureMatch property*), 868
exact_value (*GlobStrMatch property*), 867
exact_value (*ListOfStrMatch property*), 868
exact_value (*MatchInterface property*), 867
exact_value (*SplitStrMatch property*), 868
ExactLowerStrMatch (*class in conda.models.match_spec*), 867
ExactlyOne() (*Clauses method*), 694
ExactlyOne_BDD() (*Clauses method*), 667, 694
ExactlyOne_NSQ() (*Clauses method*), 667, 694
exactness_and_number_of_deps() (*in module conda.resolve*), 971
ExactStrMatch (*class in conda.models.match_spec*), 867
exc_traceback (*CondaExceptionEvent attribute*), 530, 966
exc_type (*CondaExceptionEvent attribute*), 530, 966
exc_value (*CondaExceptionEvent attribute*), 530, 966
exception_to_queue() (*in module conda._private.shards.misc*), 560
ExceptionHandler (*class in conda.exception_handler*), 783
exclude_newer (*Context attribute*), 622
exclude_newer_package (*Context attribute*), 622
exclude_newer_policy (*BaseSolver attribute*), 761
exclude_newer_policy (*Context property*), 625
exclude_newer_policy (*Index attribute*), 716
excluded_files_check() (*in module conda.plugins.subcommands.doctor.health_checks.missing_files*), 942
excluded_records() (*ExcludeNewerPolicy method*), 714
ExcludeNewerPolicy (*class in conda.core.exclude_newer*), 714
executable_paths (*PreferredEnv attribute*), 870
execute() (*_Activator method*), 579
execute() (*Action method*), 734
execute() (*AggregateCompileMultiPycAction method*), 741
execute() (*CacheUrlAction method*), 750
execute() (*CompileMultiPycAction method*), 741
execute() (*CreatePrefixRecordAction method*), 743
execute() (*CreatePythonEntryPointAction method*), 742
execute() (*ExtractPackageAction method*), 751
execute() (*in module conda._preview.env_setup.cli.main_create*), 554
execute() (*in module conda._preview.env_setup.cli.main_install*), 554
execute() (*in module conda.cli.main_clean*), 649
execute() (*in module conda.cli.main_commands*), 649
execute() (*in module conda.cli.main_compare*), 650
execute() (*in module conda.cli.main_config*), 650
execute() (*in module conda.cli.main_create*), 652

execute() (*in module conda.cli.main_env*), 652
execute() (*in module conda.cli.main_env_config*), 652
execute() (*in module conda.cli.main_env_update*), 654
execute() (*in module conda.cli.main_export*), 655
execute() (*in module conda.cli.main_info*), 657
execute() (*in module conda.cli.main_init*), 658
execute() (*in module conda.cli.main_install*), 658
execute() (*in module conda.cli.main_list*), 659
execute() (*in module conda.cli.main_mock_activate*), 659
execute() (*in module conda.cli.main_mock_deactivate*), 659
execute() (*in module conda.cli.main_notices*), 659
execute() (*in module conda.cli.main_package*), 660
execute() (*in module conda.cli.main_remove*), 661
execute() (*in module conda.cli.main_rename*), 662
execute() (*in module conda.cli.main_run*), 662
execute() (*in module conda.cli.main_search*), 662
execute() (*in module conda.cli.main_update*), 663
execute() (*in module conda.plugins.subcommands.doctor*), 949
execute() (*in module conda.plugins.subcommands.plugins*), 953
execute() (*in module conda.plugins.subcommands.plugins.info*), 950
execute() (*in module conda.plugins.subcommands.plugins.install*), 950
execute() (*in module conda.plugins.subcommands.plugins.list*), 950
execute() (*in module conda.plugins.subcommands.plugins.remove*), 951
execute() (*in module conda.plugins.subcommands.plugins.update*), 952
execute() (*LinkPathAction method*), 738
execute() (*MakeMenuAction method*), 740
execute() (*PrefixReplaceLinkAction method*), 739
execute() (*ProgressiveFetchExtract method*), 731
execute() (*RegisterEnvironmentLocationAction method*), 745
execute() (*RemoveLinkedPackageRecordAction method*), 748
execute() (*RemoveMenuAction method*), 747
execute() (*UnlinkLinkTransaction method*), 728
execute() (*UnlinkPathAction method*), 746
execute() (*UnregisterEnvironmentLocationAction method*), 749
execute() (*UpdateHistoryAction method*), 744
execute_config() (*in module conda.cli.main_config*), 651
execute_executor (*UnlinkLinkTransaction attribute*), 727
execute_list() (*in module conda.cli.main_env_vars*), 654
execute_set() (*in module conda.cli.main_env_vars*), 654
execute_threads (*Context property*), 625
execute_unset() (*in module conda.cli.main_env_vars*), 654
exists() (*PrefixData method*), 756, 945
exp_backoff_fn() (*in module conda.gateways.disk*), 816
expand() (*in module conda.activate*), 580
expand() (*LoadedParameter method*), 677
expand_environment_variables() (*in module conda.common.configuration*), 672
expanded (*Node attribute*), 567
expanded_channels (*Index attribute*), 716
expanded_extras (*Node attribute*), 567
expected (*TooManyArgumentsError attribute*), 786
expected_error_prefix (*in module conda.testing.helpers*), 985
experimental (*Context property*), 625
expired_at (*ChannelNotice attribute*), 893
explicit() (*in module conda.misc*), 840
EXPLICIT_FORMAT (*in module conda.plugins.environment_exporters.explicit*), 897
EXPLICIT_MARKER (*in module conda.base.constants*), 619
explicit_packages (*Environment attribute*), 858
ExplicitSpec (*class in conda.env.specs.explicit*), 780
explode_directories() (*in module conda.common.path*), 699
explode_directories() (*in module conda.common.path.directories*), 696
export (*CondaEnvironmentExporter attribute*), 513, 964
export_explicit() (*in module conda.plugins.environment_exporters.explicit*), 897
export_json() (*in module conda.plugins.environment_exporters.environment_yml*), 897
export_platforms (*Context property*), 626
export_requirements() (*in module conda.plugins.environment_exporters.requirements_txt*), 898
export_var_tmpl (*_Activator attribute*), 579
export_var_tmpl (*CmdExeActivator attribute*), 582
export_var_tmpl (*CshActivator attribute*), 581
export_var_tmpl (*FishActivator attribute*), 583
export_var_tmpl (*PosixActivator attribute*), 581
export_var_tmpl (*PowerShellActivator attribute*), 584
export_var_tmpl (*XonshActivator attribute*), 582

export_yaml() (*in module conda.plugins.environment_exporters.environment_yaml*), 897
EXPORTED_CHANNELS_DIR (*in module conda.testing.helpers*), 985
extend (*_ClauseList attribute*), 664
extend() (*_ClauseArray method*), 664
ExtendConstAction (*class in conda.cli.actions*), 630
extensions (*CondaPackageExtractor attribute*), 536, 965
external_packages (*Environment attribute*), 858
EXTERNAL_PACKAGES_PYPI_KEY (*in module conda.models.environment*), 857
extra_depends (*PackageRecord attribute*), 880
extra_depends (*PackageRecordDict attribute*), 572
extra_safety_checks (*Context attribute*), 622
extract (*CondaPackageExtractor attribute*), 536, 965
EXTRACT (*in module conda.instructions*), 839
extract_actions (*ProgressiveFetchExtract property*), 731
EXTRACT_CMD() (*in module conda.instructions*), 839
extract_conda_or_tarball() (*in module conda.plugins.package_extractors.conda*), 923
extract_conda_package_archive() (*in module conda._private.extract*), 556
extract_package() (*CondaPluginManager method*), 922
EXTRACT_PROCESS_EXTENSIONS (*in module conda.core.package_cache_data*), 729
EXTRACT_PROCESSES (*in module conda.core.package_cache_data*), 729
extract_tarball() (*in module conda.gateways.disk.create*), 811
EXTRACT_THREADS (*in module conda.core.package_cache_data*), 729
extracted_package_dir (*PackageCacheRecord attribute*), 881
extracted_package_dir (*PackageInfo attribute*), 870
extracted_package_dir (*PrefixRecord attribute*), 883
ExtractPackageAction (*class in conda.core.path_actions*), 750
extrapolate() (*Environment method*), 859
extras (*PySpec attribute*), 925

F

fail_message (*SpinnerBase attribute*), 960
fallback (*_ConstantDeprecationRegistry attribute*), 773
FALSE (*in module conda.common._logic*), 664
FALSE (*in module conda.common.logic*), 693
feature() (*PackageRecord class method*), 881
FeatureMatch (*class in conda.models.match_spec*), 868
features (*Index property*), 717
features (*PackageRecord attribute*), 880
FETCH (*in module conda.instructions*), 839
fetch() (*ShardFetch method*), 562
fetch_batch() (*ShardFetch static method*), 562
fetch_channels() (*in module conda._private.shards*), 576
fetch_channels() (*in module conda._private.shards.shards*), 565
FETCH_CMD() (*in module conda.instructions*), 839
fetch_latest() (*RepodataFetch method*), 829
fetch_latest_parsed() (*RepodataFetch method*), 829
fetch_latest_path() (*RepodataFetch method*), 829
fetch_precs (*ChangeReport attribute*), 727
fetch_shards_index() (*in module conda._private.shards*), 577
fetch_shards_index() (*in module conda._private.shards.shards*), 565
fetch_threads (*Context property*), 625
fh (*Spinner attribute*), 936
Field (*class in conda.auxlib.entity*), 600
FIELD_NAMES (*MatchSpec attribute*), 863
FIELD_NAMES_SET (*MatchSpec attribute*), 863
field_type_validation() (*in module conda.env.env*), 774
file_is_empty() (*History method*), 837
file_locking_check() (*in module conda.plugins.subcommands.doctor.health_checks.file_locking*), 941
file_mode (*PathData attribute*), 877
file_mode (*PrefixReplaceLinkAction attribute*), 739
FILE_NAMES (*PythonDistributionMetadata attribute*), 927
FILE_NOT_FOUND (*ERROR attribute*), 669
file_path_is_writable() (*in module conda.gateways.disk.test*), 815
file_scheme (*in module conda.common.url*), 707
FileMode (*class in conda.models.enums*), 852
filename (*Cep24YamlFileSpec attribute*), 779
filename (*EnvironmentExporterNotDetected attribute*), 797
filename (*EnvironmentFileEmpty attribute*), 796
filename (*EnvironmentFileExtensionNotValid attribute*), 796
filename (*EnvironmentFileNotFound attribute*), 795
filename (*EnvironmentYaml attribute*), 776
filename (*ExplicitSpec attribute*), 780
filename (*RequirementsSpec attribute*), 781
filename (*YamlFileSpec attribute*), 782
FilenameField (*class in conda.models.records*), 876
FileNotFoundError (*in module conda.core.package_cache_data*), 729
FileNotFoundError (*in module conda.core.path_actions*), 733
filepath (*CondaFileIOError attribute*), 789
files (*PrefixRecord attribute*), 883

FileSpecTypes (*in module conda.env.specs*), 783
fillvalue (*VersionOrder attribute*), 886
filter() (*TokenURLFilter method*), 817
filter_notices() (*in module conda.notices.core*), 891
filter_records() (*ExcludeNewerPolicy method*), 714
filter_redundant_packages() (*in module conda._private.shards.misc*), 560
final (*ParameterFlag attribute*), 674
final_action_groups (*PrefixActionGroup attribute*), 727
final_environment_specs (*SolverStateContainer attribute*), 764
find_altered_packages() (*in module conda.plugins.subcommands.doctor.health_checks.altered_files*), 939
find_builtin_commands() (*in module conda.cli.conda_argparse*), 634
find_commands() (*in module conda.cli.find_commands*), 641
find_conflicts() (*Resolve method*), 973
find_executable() (*in module conda.cli.find_commands*), 641
find_inconsistent_packages() (*in module conda.plugins.subcommands.doctor.health_checks.consistency*), 940
find_index_cache() (*in module conda.cli.main_clean*), 649
find_logfiles() (*in module conda.cli.main_clean*), 649
find_malformed_pinned_specs() (*in module conda.plugins.subcommands.doctor.health_checks.pinned*), 942
find_matches() (*Resolve method*), 974
find_matches_with_strict() (*Resolve method*), 973
find_or_none() (*in module conda.auxlib.ish*), 607
find_or_raise() (*in module conda.auxlib.ish*), 608
find_package() (*SolverTests method*), 992
find_package_in_list() (*SolverTests method*), 992
find_packages_with_missing_files() (*in module conda.plugins.subcommands.doctor.health_checks.missing_files*), 942
find_pkgs() (*in module conda.cli.main_clean*), 649
find_pkgs_dirs() (*in module conda.cli.main_clean*), 649
find_tarballs() (*in module conda.cli.main_clean*), 649
find_tempfiles() (*in module conda.cli.main_clean*), 649
finish() (*ProgressBarBase method*), 960
first() (*in module conda.auxlib.collection*), 592
first_writable() (*PackageCacheData class method*), 730, 822
first_writable() (*PackageCacheData static method*), 589
first_writable_envs_dir() (*in module conda.gateways.disk.create*), 811
FishActivator (*class in conda.activate*), 583
fix (*CondaHealthCheck attribute*), 535, 958
fix_altered_files() (*in module conda.plugins.subcommands.doctor.health_checks.altered_files*), 939
fix_env_txt() (*in module conda.plugins.subcommands.doctor.health_checks.environment_t*, 941
fix_inconsistent_packages() (*in module conda.plugins.subcommands.doctor.health_checks.consistency*), 940
fix_malformed_pinned() (*in module conda.plugins.subcommands.doctor.health_checks.pinned*), 942
fix_missing_files() (*in module conda.plugins.subcommands.doctor.health_checks.missing_files*), 942
fix_shebang() (*in module conda.cli.main_package*), 660
fixer (*CondaHealthCheck attribute*), 535, 958
fixture_names (*BaseTestCase attribute*), 975
flags (*PackageRecord attribute*), 880
flatten_notice_responses() (*in module conda.notices.core*), 891
FLEXIBLE (*ChannelPriority attribute*), 618
fmt (*Dist attribute*), 849
fmt (*DistDetails attribute*), 848
fn (*Dist property*), 850
fn (*MatchSpec property*), 865
fn (*PackageRecord attribute*), 878
FORBIDDEN_HEADERS (*in module conda.gateways.connection.session*), 807
FORBIDDEN_HTTP_METHODS (*in module conda.gateways.connection.session*), 807
FORBIDDEN_METHOD_OVERRIDE_HEADERS (*in module conda.gateways.connection.session*), 807
force (*Context attribute*), 624
force_32bit (*Context attribute*), 622
force_color() (*in module conda.common.terminal*), 705
force_reinstall (*Context attribute*), 624
force_remove (*Context attribute*), 624
force_remove (*SolverStateContainer attribute*), 764
format() (*DeltaSecondsFormatter method*), 688
format() (*ErrorGuidance method*), 556
format_dict() (*in module conda.cli.main_config*), 650
format_env_spec_available_plugins() (*in module conda.exceptions*), 796
FORMAT_JSON (*in module conda.gateways.repodata*), 826
FORMAT_SHARDS (*in module conda.gateways.repodata*), 826
FORMAT_ZST (*in module conda.gateways.repodata*), 826

formatter_map (*in module conda.activate*), 584
forward_to_subprocess() (*in module conda.testing.helpers*), 987
freebsd (*Platform attribute*), 852
FREEZE_INSTALLED (*UpdateModifier attribute*), 617
fresh_context() (*in module conda.base.context*), 628
from_channel_name() (*Channel static method*), 824, 844
from_cli() (*Environment class method*), 859
from_cli_channels() (*EnvironmentConfig class method*), 858
from_cli_with_file_envs() (*Environment class method*), 859
from_context() (*EnvironmentConfig class method*), 858
from_context() (*PrefixData class method*), 756, 945
from_dist_str() (*MatchSpec class method*), 863
from_env_condarc() (*ConfigurationFile class method*), 639
from_environment() (*in module conda.env.env*), 775
from_file() (*in module conda.env.env*), 775
from_hints() (*ErrorGuidance class method*), 556
from_history() (*Environment static method*), 859
from_index() (*Clauses method*), 693
from_json() (*Entity class method*), 606
from_name() (*Clauses method*), 693
from_name() (*ParameterFlag class method*), 674
from_name() (*PrefixData class method*), 756, 945
from_objects() (*Entity class method*), 606
from_parse_result() (*Url class method*), 708
from_prefix() (*Environment class method*), 859
from_settings() (*_ChannelCutoff class method*), 714
from_string() (*Dist class method*), 849
from_string() (*ParameterFlag class method*), 674
from_sys() (*Arch class method*), 851
from_sys() (*Platform class method*), 852
from_system_condarc() (*ConfigurationFile class method*), 638
from_url() (*Channel static method*), 824, 844
from_url() (*Dist class method*), 849
from_user_condarc() (*ConfigurationFile class method*), 638
from_value() (*_ChannelSelector class method*), 713
from_value() (*Channel static method*), 824, 844
from_value() (*ParameterFlag class method*), 674
from_values() (*ExcludeNewerPolicy class method*), 714
from_yaml() (*in module conda.env.env*), 775
FTPAdapter (*class in conda.gateways.connection.adapters.ftp*), 801
full_name (*Dist property*), 849
fullname (*_ConstantDeprecationRegistry attribute*), 773
fullname() (*in module conda.auxlib.logz*), 609
func (*CreatePythonEntryPointAction attribute*), 742
func_table (*FTPAdapter attribute*), 801
function() (*_HookImplWrapper method*), 914

G

gen_clauses() (*Resolve method*), 974
GeneralGraph (*class in conda.models.prefix_graph*), 872
generate_feature_count() (*Resolve method*), 974
generate_feature_metric() (*Resolve method*), 974
generate_install_count() (*Resolve method*), 974
generate_package_count() (*Resolve method*), 974
generate_parser() (*in module conda.cli.conda_argparse*), 633
generate_pre_parser() (*in module conda.cli.conda_argparse*), 633
generate_removal_count() (*Resolve method*), 974
generate_shebang_for_entry_point() (*in module conda.core.portability*), 754
generate_spec_constraints() (*Resolve method*), 974
generate_update_count() (*Resolve method*), 974
generate_version_metrics() (*Resolve method*), 974
generic (*NoarchType attribute*), 856
GenericHelp, 785
get() (*_LazyParserMap method*), 635
get() (*MatchSpec method*), 863
get() (*PackageCacheData method*), 588, 730, 822
get() (*PrefixData method*), 590, 758, 947
get_all_directories() (*in module conda.common.path*), 699
get_all_directories() (*in module conda.common.path.directories*), 696
get_all_extracted_entries() (*PackageCacheData class method*), 730, 822
get_all_matches() (*MapParameter method*), 681
get_all_matches() (*ObjectParameter method*), 682
get_all_matches() (*Parameter method*), 680
get_all_matches() (*SequenceParameter method*), 681
get_archspec_name() (*in module conda.core.index*), 719
get_auth_handler() (*CondaPluginManager method*), 916
get_available_environment_formats() (*in module conda.cli.formats*), 642
get_cache_control_max_age() (*in module conda.gateways.repodata*), 830
get_cache_key() (*ChannelNoticeResponse class method*), 894
get_cached_request_headers (*CondaPluginManager attribute*), 914
get_cached_session_headers (*CondaPluginManager attribute*), 914
get_cached_solver_backend (*CondaPluginManager attribute*), 914

get_canonical_name() (*CondaPluginManager method*), 914
get_channel_name_and_urls() (*in module conda.notices.core*), 891
get_channel_name_from_url() (*in module conda.gateways.connection.session*), 807
get_channel_notice_response() (*in module conda.notices.fetch*), 892
get_channel_objs() (*in module conda.models.channel*), 848
get_classifiers() (*PythonDistributionMetadata method*), 930
get_clause_count() (*_ClauseArray method*), 664
get_clause_count() (*_ClauseList method*), 664
get_clause_count() (*_SatSolver method*), 665
get_clause_count() (*Clauses method*), 666, 693
get_comspec() (*in module conda.utils*), 995
get_conda_anchor_files_and_records() (*in module conda.core.prefix_data*), 759
get_conda_anchor_files_and_records() (*in module conda.plugins.prefix_data_loaders.pypi*), 933
get_conda_build_local_url() (*in module conda.models.channel*), 847
get_conda_dependencies() (*PythonDistribution method*), 926
get_conda_packages() (*PrefixData method*), 758, 947
get_config() (*CondaPluginManager method*), 918
get_conflicting_specs() (*Resolve method*), 974
get_constrained_packages() (*Solver method*), 763
get_default_marker_context() (*in module conda.plugins.prefix_data_loaders.pypi.pkg_format*), 931
get_default_urls() (*in module conda.exports*), 799
get_descriptions() (*Configuration method*), 685
get_descriptions() (*Context method*), 628
get_descriptions() (*PluginConfig method*), 896
get_dist_file_from_egg_link() (*in module conda.plugins.prefix_data_loaders.pypi.pkg_format*), 931
get_dist_obsolete() (*PythonDistributionMetadata method*), 930
get_dist_provides() (*PythonDistributionMetadata method*), 929
get_dist_requirements() (*PythonDistribution method*), 926
get_dist_requirements() (*PythonDistributionMetadata method*), 928
get_entry_points() (*PythonDistribution method*), 926
get_entry_to_link() (*PackageCacheData class method*), 730, 822
get_env_vars_str() (*in module conda.cli.main_info*), 656
get_environment_env_vars() (*PrefixData method*), 759, 948
get_environment_exporter_by_format() (*CondaPluginManager method*), 920
get_environment_exporters() (*CondaPluginManager method*), 920
get_environment_exporters_grouped() (*CondaPluginManager method*), 920
get_environment_specifier() (*CondaPluginManager method*), 919
get_environment_specifier_by_name() (*CondaPluginManager method*), 918
get_environment_specifiers() (*CondaPluginManager method*), 918
get_environment_specifiers_grouped() (*CondaPluginManager method*), 920
get_error_hints() (*CondaPluginManager method*), 917
get_error_report() (*ExceptionHandler method*), 784
get_exact_value() (*MatchSpec method*), 863
get_export_unset_vars() (*_Activator method*), 579
get_exporter_format_mapping() (*CondaPluginManager method*), 920
get_external_requirements() (*PythonDistribution method*), 926
get_external_requirements() (*PythonDistributionMetadata method*), 929
get_extra_provides() (*PythonDistribution method*), 926
get_extra_provides() (*PythonDistributionMetadata method*), 929
get_free_space_on_unix() (*in module conda.common._os.unix*), 668
get_free_space_on_windows() (*in module conda.common._os.windows*), 669
get_health_checks() (*CondaPluginManager method*), 917
get_hook_results() (*CondaPluginManager method*), 915
get_host_and_path_from_url() (*FTPAdapter method*), 802
get_index() (*in module conda.exports*), 799
get_index_args() (*in module conda.cli.install*), 646
get_index_cuda() (*in module conda.testing.helpers*), 986
get_index_must_unfreeze() (*in module conda.testing.helpers*), 986
get_index_r_1() (*in module conda.testing.helpers*), 986
get_index_r_2() (*in module conda.testing.helpers*), 986
get_index_r_4() (*in module conda.testing.helpers*), 986
get_index_r_5() (*in module conda.testing.helpers*),

986
get_info_dict() (*in module conda.cli.main_info*), 656
get_installed_plugin_info() (*CondaPluginManager method*), 915
get_installed_plugin_package_names() (*in module conda.plugins.subcommands.plugins.package_utils*), 951
get_installed_plugins() (*CondaPluginManager method*), 915
get_installed_version() (*in module conda.cli.main_package*), 660
get_installer() (*in module conda.env.installers.base*), 777
get_installer_info() (*in module conda.cli.main_info*), 655
get_instrumentation_record_file() (*in module conda.common.io*), 691
get_key() (*ConfigurationFile method*), 640
get_leaf_directories() (*in module conda.common.path*), 699
get_leaf_directories() (*in module conda.common.path.directories*), 696
get_local_urls (*in module conda.exports*), 799
get_lock() (*JSONProgressBar class method*), 938
get_lock() (*ProgressBarBase class method*), 960
get_main_info_display() (*in module conda.cli.main_info*), 656
get_main_info_str() (*in module conda.cli.main_info*), 656
get_major_minor_version() (*in module conda.common.path*), 699
get_major_minor_version() (*in module conda.common.path.python*), 697
get_matcher() (*BuildNumberMatch method*), 888
get_matcher() (*VersionSpec method*), 888
get_multiplatform_lockfile() (*in module conda.cli.formats*), 642
get_name_from_conda_meta_file_name() (*in module conda.plugins.subcommands.doctor.health_checks.unlinked*), 943
get_name_prefix_from_env_file() (*in module conda.cli.common*), 632
get_node_by_name() (*PrefixGraph method*), 871, 934
get_notice_cache_filenames() (*in module conda.testing.notices.helpers*), 990
get_notice_response_from_cache() (*in module conda.notices.cache*), 890
get_notice_responses() (*in module conda.notices.fetch*), 892
get_notices_cache_dir() (*in module conda.notices.cache*), 889
get_notices_cache_file() (*in module conda.notices.cache*), 889
get_optional_dependencies() (*PythonDistribution method*), 926
get_package_extractor() (*CondaPluginManager method*), 922
get_package_extractors() (*CondaPluginManager method*), 922
get_package_records_from_explicit() (*in module conda.misc*), 841
get_packages() (*in module conda.cli.main_list*), 658
get_path() (*TmpEnvFixture method*), 980
get_paths() (*PythonDistribution method*), 925
get_pinned_specs() (*in module conda.core.solve*), 764
get_pinned_specs() (*PrefixData method*), 759, 948
get_pip_installed_packages() (*in module conda.env.pip_util*), 779
get_pip_workdir() (*in module conda.env.pip_util*), 779
get_pkgs() (*Resolve method*), 974
get_plugin_hook_names() (*CondaPluginManager method*), 915
get_plugin_manager() (*in module conda.plugins.manager*), 922
get_plugin_source() (*CondaPluginManager method*), 915
get_post_transaction_actions() (*CondaPluginManager method*), 921
get_pre_transaction_actions() (*CondaPluginManager method*), 920
get_prefix_data_loaders() (*CondaPluginManager method*), 917
get_progress_bar() (*in module conda.reporters*), 970
get_progress_bar_context_manager() (*in module conda.reporters*), 971
get_proxy_username_and_pass() (*in module conda.common.url*), 710
get_python_noarch_target_path() (*in module conda.common.path*), 699
get_python_noarch_target_path() (*in module conda.common.path.python*), 697
get_python_packages() (*PrefixData method*), 759, 948
get_python_requirements() (*PythonDistribution method*), 926
get_python_requirements() (*PythonDistributionMetadata method*), 928
get_python_short_path() (*in module conda.common.path*), 699
get_python_short_path() (*in module conda.common.path.python*), 697
get_python_site_packages_short_path() (*in module conda.common.path*), 699
get_python_site_packages_short_path() (*in module conda.common.path.python*), 697
get_python_site_packages_short_path() (*in module conda.plugins.prefix_data_loaders.pypi*),

932
get_raw_value() (*MatchSpec method*), 863
get_rc_urls (*in module conda.exports*), 799
get_reduced_index() (*Index method*), 717
get_reduced_index() (*Resolve method*), 974
get_repo_interface() (*in module conda.gateways.repodata*), 827
get_reporter_backend() (*CondaPluginManager method*), 917
get_reporter_backends() (*CondaPluginManager method*), 917
get_request_headers() (*CondaPluginManager method*), 917
get_request_package_in_solution() (*Solver method*), 763
get_requested_specs_map() (*History method*), 838
get_revision() (*in module conda.cli.install*), 646
get_schema_errors() (*in module conda.env.env*), 775
get_session() (*in module conda.gateways.connection.session*), 808
get_session() (*in module conda.gateways.repodata*), 826
get_session_headers() (*CondaPluginManager method*), 917
get_session_storage_key() (*in module conda.gateways.connection.session*), 808
get_settings() (*CondaPluginManager method*), 916
get_shortcut_dir() (*in module conda.testing.integration*), 989
get_signal_name() (*in module conda.common.signals*), 704
get_site_packages_anchor_files() (*in module conda.plugins.prefix_data_loaders.pypi*), 934
get_site_packages_anchor_files() (*in module conda.plugins.prefix_data_loaders.pypi.pkg_format*), 931
get_softlinked_package_dirs() (*in module conda.core.package_cache_data*), 729
get_solver() (*in module conda.testing.helpers*), 986
get_solver_2() (*in module conda.testing.helpers*), 986
get_solver_4() (*in module conda.testing.helpers*), 986
get_solver_5() (*in module conda.testing.helpers*), 986
get_solver_aggregate_1() (*in module conda.testing.helpers*), 986
get_solver_aggregate_2() (*in module conda.testing.helpers*), 986
get_solver_backend() (*CondaPluginManager method*), 916
get_solver_cuda() (*in module conda.testing.helpers*), 986
get_solver_must_unfreeze() (*in module conda.testing.helpers*), 986
get_solvers() (*CondaPluginManager method*), 916
get_spec_str_from_conda_meta() (*in module conda.plugins.subcommands.doctor.health_checks.utils*), 943
get_spinner() (*in module conda.reporters*), 971
get_state() (*History method*), 838
get_status_code_from_code_response() (*in module conda.gateways.connection.adapters.ftp*), 802
get_subcommands() (*CondaPluginManager method*), 917
get_test_notices() (*in module conda.testing.notices.helpers*), 990
get_url() (*HttpTestServerFixture method*), 982
get_url() (*UrlsData method*), 731
get_user_environments_txt_file() (*in module conda.core.envs_manager*), 712
get_user_requests() (*History method*), 838
get_user_site() (*in module conda.cli.main_info*), 655
get_username_password_from_header() (*FTPAdapter method*), 802
get_viewed_channel_notice_ids() (*in module conda.notices.cache*), 890
get_virtual_package_records() (*CondaPluginManager method*), 917
get_windows_launcher_stub() (*in module conda.core.launchers*), 725
Getting started, **137**
global_cutoff (*ExcludeNewerPolicy attribute*), 714
GlobLowerStrMatch (*class in conda.models.match_spec*), 867
GlobStrMatch (*class in conda.models.match_spec*), 867
graph (*PrefixGraph attribute*), 871, 933
graph_by_name (*GeneralGraph attribute*), 872
groupby_to_dict() (*in module conda.common.iterators*), 692
groups (*Resolve attribute*), 972
guidance (*CondaError property*), 821, 996
GuidanceHint (*class in conda._private.exception_guidance*), 555
GuidanceHintTypedDict (*class in conda._private.exception_guidance*), 555

H

handle_407() (*CondaHttpAuth static method*), 809
handle_application_exception() (*ExceptionHandler method*), 783
handle_exception() (*ExceptionHandler method*), 783
handle_reportable_application_exception() (*ExceptionHandler method*), 784
handle_txn() (*in module conda.cli.install*), 647
handle_unexpected_exception() (*ExceptionHandler method*), 784
handler (*CondaAuthHandler attribute*), 510, 957
hardlink (*LinkType attribute*), 854
hardlink (*PathEnum attribute*), 854

hardlink_supported() (*in module conda.gateways.disk.test*), 815
has_channel_overrides (*ExcludeNewerPolicy property*), 714
has_format() (*RepodataState method*), 827
has_global_cutoff (*ExcludeNewerPolicy property*), 714
has_package_extension() (*CondaPluginManager method*), 922
has_package_overrides (*ExcludeNewerPolicy property*), 714
has_platform() (*in module conda.common.url*), 710
has_pyzzer_entry_point() (*in module conda.core.portability*), 754
has_scheme() (*in module conda.common.url*), 709
has_strategy() (*RepodataSubset class method*), 568, 832
HASH_FIELDS (*in module conda._private.shards.misc*), 560
Help, 785
HELP (*in module conda.plugins.subcommands.plugins.info*), 950
HELP (*in module conda.plugins.subcommands.plugins.install*), 950
HELP (*in module conda.plugins.subcommands.plugins.list*), 950
HELP (*in module conda.plugins.subcommands.plugins.remove*), 951
HELP (*in module conda.plugins.subcommands.plugins.update*), 952
HIDE (*SW attribute*), 669
hint_code (*CondaErrorHint attribute*), 512, 961
hint_code (*GuidanceHint attribute*), 555
hint_code (*GuidanceHintTypedDict attribute*), 555
hint_codes (*ErrorGuidance property*), 555
hints (*ErrorGuidance attribute*), 555
History (*class in conda.history*), 837
hold_path (*CacheUrlAction attribute*), 750
hold_path (*ExtractPackageAction attribute*), 751
hold_path (*UpdateHistoryAction attribute*), 744
holding_full_path (*UnlinkPathAction attribute*), 746
holding_short_path (*UnlinkPathAction attribute*), 746
hook (*CondaExceptionObserver attribute*), 531, 967
hook() (*_Activator method*), 579
hook_source_path (*_Activator attribute*), 579
hook_source_path (*CmdExeActivator attribute*), 583
hook_source_path (*CshActivator attribute*), 581
hook_source_path (*FishActivator attribute*), 583
hook_source_path (*PosixActivator attribute*), 581
hook_source_path (*PowerShellActivator attribute*), 584
hook_source_path (*XonshActivator attribute*), 582
hookimpl (*in module conda.plugins*), 970
hookimpl (*in module conda.plugins.hookspec*), 900
hookimpl (*in module conda.plugins.prefix_data_loaders.pypi*), 934
hookimpl (*in module conda.plugins.subcommands.doctor*), 948
hookimpl (*in module conda.plugins.subcommands.plugins*), 952
host (*HttpTestServerFixture attribute*), 982
hostname (*_SplitUrlParts attribute*), 710
http_test_server() (*in module conda.testing.fixtures*), 982
http_timeout (*ExceptionHandler property*), 783
HTTPAdapter (*class in conda.gateways.connection.adapters.http*), 803
HttpTestServerFixture (*class in conda.testing.fixtures*), 982
human_bytes() (*in module conda.utils*), 994

I

icondata (*PackageInfo attribute*), 870
id (*ChannelNotice attribute*), 893
IDENTIFIER (*in module conda.plugins.prefix_data_loaders.pypi.pkg_format*), 931
IGNORE_FIELDS (*in module conda.cli.main_info*), 656
ignore_pinned (*Context attribute*), 624
ignore_pinned (*SolverStateContainer attribute*), 764
immutable (*Field property*), 602
ImmutableEntity (*class in conda.auxlib.entity*), 606
impl (*_HookImplWrapper attribute*), 914
in_dump (*Field property*), 601
in_subprocess() (*in module conda.testing.helpers*), 987
Index (*class in conda.core.index*), 715
index (*Resolve attribute*), 971
index (*SolverStateContainer attribute*), 764
index_args (*Repodatas attribute*), 646
index_args (*TryRepodata attribute*), 646
index_packages() (*in module conda.testing.solver_helpers*), 991
indexed_timestamp (*PackageRecord attribute*), 880
IndexedTimestampField (*class in conda.models.records*), 875
indices (*Clauses attribute*), 693
INFO (*Commands attribute*), 988
info (*Context property*), 627
INFO (*NoticeLevel attribute*), 619
info (*RepodataDict attribute*), 573
info (*ShardsIndexDict attribute*), 573
INFO_FORMATTER (*in module conda.auxlib.logz*), 608
InfoComponents (*in module conda.cli.main_info*), 656
InfoRenderer (*class in conda.cli.main_info*), 656
init() (*PythonDistribution static method*), 925

init_cmd_exe_registry() (*in module conda.core.initialize*), 724
init_fish_user() (*in module conda.core.initialize*), 724
init_log_file() (*History method*), 837
init_loggers() (*in module conda.cli.main*), 648
init_long_path() (*in module conda.core.initialize*), 724
init_poolmanager() (*_SSLContextAdapterMixin method*), 803
init_powershell_user() (*in module conda.core.initialize*), 724
init_sh_system() (*in module conda.core.initialize*), 724
init_sh_user() (*in module conda.core.initialize*), 724
init_xonsh_user() (*in module conda.core.initialize*), 724
initial_action_groups (*PrefixActionGroup attribute*), 727
initialize() (*in module conda.core.initialize*), 722
initialize_dev() (*in module conda.core.initialize*), 722
initialize_logging() (*in module conda.auxlib.logz*), 609
initialize_logging() (*in module conda.gateways.logging*), 818
initialize_std_loggers() (*in module conda.gateways.logging*), 818
inline_hook_source (*_Activator attribute*), 579
inline_hook_source (*CmdExeActivator attribute*), 583
inline_hook_source (*CshActivator attribute*), 581
inline_hook_source (*FishActivator attribute*), 583
inline_hook_source (*PosixActivator attribute*), 581
inline_hook_source (*PowerShellActivator attribute*), 584
inline_hook_source (*XonshActivator attribute*), 582
inode_paths (*PathDataV1 attribute*), 877
insert() (*PackageCacheData method*), 730, 822
insert() (*PrefixData method*), 758, 947
insert() (*QueueCache method*), 570
insert() (*ShardCache method*), 557
INSTALL (*Commands attribute*), 988
install() (*in module conda.cli.install*), 647
install() (*in module conda.core.initialize*), 722
install() (*in module conda.env.installers.conda*), 778
install() (*in module conda.env.installers.pip*), 778
install() (*Resolve method*), 974
install() (*SimpleEnvironment method*), 992
install_activate() (*in module conda.core.initialize*), 723
install_activate_bat() (*in module conda.core.initialize*), 723
install_anaconda_prompt() (*in module conda.core.initialize*), 723
install_clone() (*in module conda.cli.install*), 647
install_conda_csh() (*in module conda.core.initialize*), 723
install_conda_fish() (*in module conda.core.initialize*), 723
install_conda_hook_ps1() (*in module conda.core.initialize*), 723
install_conda_psm1() (*in module conda.core.initialize*), 723
install_conda_sh() (*in module conda.core.initialize*), 723
install_conda_xsh() (*in module conda.core.initialize*), 723
install_condabin_conda_activate_bat() (*in module conda.core.initialize*), 723
install_condabin_conda_auto_activate_bat() (*in module conda.core.initialize*), 723
install_condabin_conda_bat() (*in module conda.core.initialize*), 723
install_condabin_hook_bat() (*in module conda.core.initialize*), 723
install_condabin_rename_tmp_bat() (*in module conda.core.initialize*), 723
install_deactivate() (*in module conda.core.initialize*), 723
install_deactivate_bat() (*in module conda.core.initialize*), 723
install_explicit_packages() (*in module conda.misc*), 840
install_library_bin_conda_bat() (*in module conda.core.initialize*), 723
install_revision() (*in module conda.cli.install*), 647
install_Scripts_activate_bat() (*in module conda.core.initialize*), 723
install_specs() (*Resolve method*), 974
installed_packages (*SimpleEnvironment attribute*), 991
INSTALLER_INFO_FIELDS (*in module conda.cli.main_info*), 655
Installing conda, **137**
Installing packages from PyPI, **57**
Installing with conda, **116**
IntegerField (*class in conda.auxlib.entity*), 602
intermediate_path (*PrefixReplaceLinkAction attribute*), 739
interoperability (*PrefixData attribute*), 756, 945
interpret() (*in module conda.plugins.prefix_data_loaders.pypi.pkg_format*), 931
INTERRUPT_SIGNALS (*in module conda.common.signals*), 704
interval (*ChannelNotice attribute*), 893
IntField (*in module conda.auxlib.entity*), 602
invalid() (*_CutoffParser static method*), 713

invalid_chains() (*Resolve method*), 972
InvalidInstaller, 797
InvalidMatchSpec, 795
InvalidSpec, 795
InvalidTypeError, 673
InvalidVersionSpec, 795
invoke() (*_PycoSatSolver method*), 665
invoke() (*_PyCryptoSatSolver method*), 666
invoke() (*_PySatSolver method*), 666
invoke() (*_SatSolver method*), 665
invoke_exception_observers() (*CondaPluginManager method*), 916
invoke_post_commands() (*CondaPluginManager method*), 916
invoke_post_solves() (*CondaPluginManager method*), 918
invoke_pre_commands() (*CondaPluginManager method*), 916
invoke_pre_solves() (*CondaPluginManager method*), 917
is_active (*PrefixData property*), 756, 945
is_active_prefix() (*in module conda.cli.common*), 631
is_admin_on_unix() (*in module conda.common._os.unix*), 668
is_admin_on_windows() (*in module conda.common._os.windows*), 669
is_base() (*PrefixData method*), 756, 945
is_cep26_valid (*BuildString property*), 842
is_channel (*Dist property*), 849
is_channel_notices_cache_expired() (*in module conda.notices.core*), 892
is_channel_notices_enabled() (*in module conda.notices.core*), 891
is_conda_plugin_package() (*CondaPluginManager method*), 915
is_diff() (*in module conda.history*), 837
is_environment() (*PrefixData method*), 756, 945
is_exact() (*BaseSpec method*), 887
is_executable() (*in module conda.gateways.disk.permissions*), 813
is_extracted (*PackageCacheRecord property*), 882
is_false_override() (*_CutoffParser static method*), 713
is_feature_package (*Dist property*), 849
is_fetched (*PackageCacheRecord property*), 882
is_frozen() (*PrefixData method*), 756, 945
is_interactive_tty() (*in module conda.common.terminal*), 705
is_ip_address() (*in module conda.common.url*), 709
is_ipv4_address() (*in module conda.common.url*), 708
is_ipv6_address() (*in module conda.common.url*), 708
is_linked() (*in module conda.exports*), 799
is_manageable (*PythonEggInfoDistribution property*), 927
is_manageable (*PythonEggLinkDistribution attribute*), 927
is_manageable (*PythonInstalledDistribution attribute*), 926
is_name_only_spec (*MatchSpec property*), 863
is_notice_response_cache_expired() (*in module conda.notices.cache*), 889
is_nullable (*Field property*), 601
is_preview_subcommand() (*in module conda.plugins.previews*), 935
is_stderr_tty() (*in module conda.common.terminal*), 705
is_stdin_tty() (*in module conda.common.terminal*), 705
is_tty() (*in module conda.common.terminal*), 705
is_unmanageable (*PackageRecord property*), 880
is_url() (*in module conda.common.url*), 708
is_valid_import_path() (*in module conda.common.path.python*), 697
is_writable (*PackageCacheData property*), 589, 730, 822
is_writable (*PrefixData property*), 590, 757, 946
isiterable() (*in module conda.common.compat*), 671
islink (*in module conda.gateways.disk.link*), 812
ITE() (*Clauses method*), 667, 694
items() (*_LazyParserMap method*), 635
iter_info_components() (*in module conda.cli.main_info*), 657
iter_records() (*PackageCacheData method*), 589, 730, 822
iter_records() (*PrefixData method*), 590, 758, 947
iter_records() (*ShardBase method*), 563, 575
iter_records() (*Shards method*), 831
iter_records() (*SubdirData method*), 588, 768
iter_records_sorted() (*PrefixData method*), 758, 947
iter_records_v3() (*ShardBase method*), 563, 575
iter_records_v3() (*Shards method*), 831
itersolve() (*Clauses method*), 694
itervalues() (*PackageCacheData method*), 730, 823

J

join() (*in module conda.common.url*), 709
join_url (*in module conda.common.url*), 709
join_url (*in module conda.gateways.repodata*), 822
json (*CondaExceptionEvent attribute*), 530, 966
json (*Context attribute*), 624
json() (*Entity method*), 606
json() (*MockResponse method*), 990
json_data (*ChannelNoticeResponse attribute*), 894
json_data (*MockResponse attribute*), 990

JSONDecodeError (*in module conda.common.serialize.json*), 702
JSONFormatMixin (*class in conda.activate*), 584
JSONProgressBar (*class in conda.plugins.reporter_backends.json*), 938
JSONReporterRenderer (*class in conda.plugins.reporter_backends.json*), 938
JSONSpinner (*class in conda.plugins.reporter_backends.json*), 938

K

key (*CondaKeyError attribute*), 789
key (*RawParameter attribute*), 674
key() (*BuildString class method*), 842
key_exists() (*ConfigurationFile method*), 640
key_flag (*LoadedParameter attribute*), 676
keyflag() (*ArgParseRawParameter method*), 675
keyflag() (*DefaultValueRawParameter method*), 676
keyflag() (*EnvRawParameter method*), 675
keyflag() (*RawParameter method*), 675
keyflag() (*YamlRawParameter method*), 675
keys (*_ChannelSelector attribute*), 713
keys (*MultipleKeysError attribute*), 673
keys() (*_LazyParserMap method*), 635
keys_for() (*_ChannelSelector class method*), 713
keys_for_record() (*_ChannelSelector class method*), 713
KNOWN_SUBDIRS (*in module conda.base.constants*), 613
known_subdirs() (*Context method*), 626
KnownPackageClobberError, 786

L

label (*EnvironmentFormat property*), 963
last() (*in module conda.auxlib.collection*), 592
last_activated (*PrefixData property*), 758, 947
LAST_CHANNEL_URLS (*in module conda.core.index*), 715
last_modified (*PrefixData property*), 758, 947
LAST_MODIFIED_KEY (*in module conda.gateways.repodata*), 826
last_repodata (*TryRepodata attribute*), 646
LazyChoicesAction (*class in conda.cli.helpers*), 643
LB_Preprocess() (*Clauses method*), 667
lchmod (*in module conda.gateways.disk.link*), 812
legacy_bz2_md5 (*PackageRecord attribute*), 878
legacy_bz2_md5 (*PackageRecordDict attribute*), 572
legacy_bz2_size (*PackageRecord attribute*), 878
level (*ChannelNotice attribute*), 893
libc_family_version() (*Context method*), 627
license (*PackageRecord attribute*), 880
license_family (*PackageRecord attribute*), 880
LinearBound() (*Clauses method*), 667, 694
Link (*class in conda.models.records*), 875
link (*in module conda.gateways.disk.link*), 812
LINK (*in module conda.instructions*), 839
link (*PrefixRecord attribute*), 883
link_action_groups (*PrefixActionGroup attribute*), 726
link_precs (*Action attribute*), 734
link_precs (*PrefixSetup attribute*), 726
link_precs (*ProgressiveFetchExtract attribute*), 731
link_type (*LinkPathAction attribute*), 738
link_type (*UnlinkPathAction attribute*), 746
linked() (*in module conda.exports*), 799
linked_data() (*in module conda.exports*), 799
linked_package_data (*RemoveFromPrefixPathAction attribute*), 745
linked_package_record (*PathEnum attribute*), 854
LinkError, 788
LinkError (*in module conda.exports*), 799
LinkPathAction (*class in conda.core.path_actions*), 737
LinkScriptError, 788
LinkSourceNotFoundError, 788
LinkType (*class in conda.models.enums*), 853
LinkTypeField (*class in conda.models.records*), 873
linux (*Platform attribute*), 852
linux_get_libc_version() (*in module conda.common._os.linux*), 667
linux_version() (*in module conda.plugins.virtual_packages.linux*), 969
LINUX_VERSION_PATTERN (*in module conda.plugins.virtual_packages.linux*), 969
linux_version_validate() (*in module conda.plugins.virtual_packages.linux*), 969
LIST (*Commands attribute*), 988
list() (*FTPAdapter method*), 801
list_all_known_prefixes() (*in module conda.core.envs_manager*), 712
list_fields (*Context attribute*), 624
list_fields_validation() (*in module conda.base.context*), 621
list_packages() (*in module conda.cli.main_list*), 658
list_parameters() (*Configuration method*), 685
listdir (*in module conda.gateways.disk.read*), 814
ListField (*class in conda.auxlib.entity*), 604
ListOfStrMatch (*class in conda.models.match_spec*), 868
load() (*Entity class method*), 606
load() (*in module conda.common.serialize.json*), 702
load() (*in module conda.common.serialize.yaml*), 703
load() (*MapParameter method*), 681
load() (*ObjectParameter method*), 682
load() (*PackageCacheData method*), 730, 822
load() (*Parameter method*), 680
load() (*PrefixData method*), 758, 947
load() (*PrimitiveParameter method*), 680
load() (*RepodataCache method*), 828
load() (*SequenceParameter method*), 682

load() (*SubdirData method*), 768
load_condarc (*in module conda.exports*), 799
load_entrypoints() (*CondaPluginManager method*), 915
load_file() (*in module conda.env.env*), 775
load_plugins() (*CondaPluginManager method*), 915
load_settings() (*CondaPluginManager method*), 918
load_site_packages() (*in module conda.plugins.prefix_data_loaders.pypi*), 934
load_state() (*RepodataCache method*), 828
LoadedParameter (*class in conda.common.configuration*), 676
loader (*CondaPrefixDataLoader attribute*), 542, 934, 962
loads() (*in module conda.common.serialize.json*), 702
loads() (*in module conda.common.serialize.yaml*), 703
local_build_root (*Context property*), 625
local_repodata_ttl (*Context attribute*), 622
LocalFSAdapter (*class in conda.gateways.connection.adapters.localfs*), 804
locate_prefix_by_name() (*in module conda.base.context*), 629
location (*Channel attribute*), 824, 844
location (*MultiChannel attribute*), 846
lock() (*in module conda.gateways.disk.lock*), 813
lock() (*in module conda.gateways.repodata*), 826
lock() (*RepodataCache method*), 828
LOCK_ATTEMPTS (*in module conda.gateways.disk.lock*), 812
LOCK_BYTE (*in module conda.gateways.disk.lock*), 812
LOCK_SLEEP (*in module conda.gateways.disk.lock*), 813
LockError, 785
lockfile (*EnvironmentFormat attribute*), 963
locking_supported() (*in module conda.gateways.disk.lock*), 813
log_level (*Context property*), 627
log_totals() (*time_recorder class method*), 691
logger (*in module conda.notices.cache*), 889
logger (*in module conda.notices.core*), 891
logger (*in module conda.notices.fetch*), 892
logger (*in module conda.plugins.subcommands.doctor.health_checks.altered_files*), 939
logger (*in module conda.plugins.subcommands.doctor.health_checks.environment_txt*), 941
logger (*in module conda.plugins.subcommands.doctor.health_checks.missing_files*), 942
logger (*in module conda.plugins.subcommands.doctor.health_checks.utils*), 943
logger (*in module conda.reporters*), 970
Logging, **484**

M

m (*Clauses attribute*), 666
m (*Clauses property*), 693
mac_ver() (*in module conda.common._os.osx*), 668
machine_bits (*in module conda.base.constants*), 613
main() (*in module conda.cli*), 663
main() (*in module conda.cli.main*), 648
main() (*in module conda.cli.main_pip*), 661
main_sourced() (*in module conda.cli.main*), 648
main_subshell() (*in module conda.cli.main*), 648
make_actions_for_record() (*ProgressiveFetchExtract static method*), 731
make_conda_egg_link() (*in module conda.core.initialize*), 724
make_condabin_plan() (*in module conda.core.initialize*), 722
make_dev_egg_info_file() (*in module conda.core.initialize*), 724
make_diff() (*in module conda.core.initialize*), 724
make_entry_point() (*in module conda.core.initialize*), 723
make_entry_point_exe() (*in module conda.core.initialize*), 723
make_executable() (*in module conda.gateways.disk.permissions*), 813
make_initialize_plan() (*in module conda.core.initialize*), 722
make_install_plan() (*in module conda.core.initialize*), 722
make_menu() (*in module conda.gateways.disk.create*), 811
make_menu_action_groups (*PrefixActionGroup attribute*), 727
make_raw_parameters() (*ArgParseRawParameter class method*), 675
make_raw_parameters() (*EnvRawParameter class method*), 675
make_raw_parameters() (*RawParameter class method*), 675
make_raw_parameters() (*YamlRawParameter class method*), 675
make_raw_parameters_from_file() (*YamlRawParameter class method*), 675
make_read_only() (*in module conda.gateways.disk.permissions*), 813
make_simple_channel() (*Channel static method*), 825, 845
make_tarbz2() (*in module conda.cli.main_package*), 660
make_unlink_actions() (*in module conda.core.link*), 725
make_writable() (*in module conda.gateways.disk.permissions*), 813

MakeMenuAction (*class in conda.core.path_actions*), 739
Managing channels, 57
Managing conda, 57
Managing environments, 57
Managing packages, 57
Managing python, 57
Managing virtual packages, 57
MANDATORY_FILES (*PythonDistribution attribute*), 925
MANDATORY_FILES (*PythonEggInfoDistribution attribute*), 927
MANDATORY_FILES (*PythonInstalledDistribution attribute*), 926
MANIFEST_FILES (*PythonDistribution attribute*), 925
MANIFEST_FILES (*PythonEggInfoDistribution attribute*), 927
MANIFEST_FILES (*PythonInstalledDistribution attribute*), 926
manifest_full_path() (*PythonDistribution method*), 925
map_parameters (*ParameterTypeGroups property*), 637
map_records() (*PrefixData method*), 758, 947
MapField (*class in conda.auxlib.entity*), 605
MapLoadedParameter (*class in conda.common.configuration*), 677
MapParameter (*class in conda.common.configuration*), 680
mark_channel_notices_as_viewed() (*in module conda.notices.cache*), 890
marker (*PySpec attribute*), 925
MARKER_OP (*in module conda.plugins.prefix_data_loaders.pypi.pkg_format*), 931
mask_anaconda_token() (*in module conda.common.url*), 709
massage_arguments() (*in module conda.utils*), 995
match (*BaseSpec attribute*), 887
match() (*CaseInsensitiveStrMatch method*), 869
match() (*ChannelMatch method*), 868
match() (*ExactLowerStrMatch method*), 867
match() (*ExactStrMatch method*), 867
match() (*FeatureMatch method*), 868
match() (*GlobStrMatch method*), 867
match() (*ListOfStrMatch method*), 868
match() (*MatchInterface method*), 866
match() (*MatchSpec method*), 863
match() (*SplitStrMatch method*), 868
match_any() (*Resolve method*), 974
match_specs_to_dists() (*in module conda.core.link*), 726
matches() (*_ChannelCutoff method*), 714
matches() (*_ChannelSelector method*), 713
matches() (*MatchInterface method*), 867
matches_all (*GlobStrMatch property*), 867
matches_metadata() (*PackageCacheRecord method*), 882
MatchInterface (*class in conda.models.match_spec*), 866
MatchSpec (*class in conda.models.match_spec*), 861
MatchSpecType (*class in conda.models.match_spec*), 861
MAX_CHANNEL_PRIORITY (*in module conda.base.constants*), 614
MAX_REPODATA_VERSION (*in module conda.core.subdir_data*), 765
MAX_SECONDS_TIMESTAMP (*in module conda.common.datetime*), 686
MAX_SHEBANG_LENGTH (*in module conda.core.portability*), 752
MAX_TRIES (*in module conda.gateways.disk*), 816
MAXIMIZE (*SW attribute*), 669
maybe_add_auth() (*in module conda.common.url*), 710
maybe_raise() (*in module conda.exceptions*), 797
maybe_unquote() (*in module conda.common.url*), 711
maybe_unquote() (*in module conda.gateways.repodata*), 822
maybecall() (*in module conda.auxlib.type_coercion*), 610
md5 (*CacheUrlAction attribute*), 750
md5 (*ExtractPackageAction attribute*), 751
md5 (*PackageCacheRecord attribute*), 882
md5 (*PackageRecord attribute*), 878
md5 (*PackageRecordDict attribute*), 572
Md5Field (*class in conda.models.records*), 881
memoizedproperty() (*in module conda.auxlib.decorators*), 594
memoizemethod() (*in module conda.auxlib.decorators*), 593
merge() (*BaseSpec method*), 888
merge() (*BuildNumberMatch method*), 888
merge() (*Environment class method*), 859
merge() (*EnvironmentConfig class method*), 858
merge() (*GlobStrMatch method*), 867
merge() (*LoadedParameter method*), 677
merge() (*MapLoadedParameter method*), 678
merge() (*MatchInterface method*), 867
merge() (*MatchSpec class method*), 865
merge() (*ObjectLoadedParameter method*), 679
merge() (*PrimitiveLoadedParameter method*), 677
merge() (*SequenceLoadedParameter method*), 678
merge() (*VersionSpec method*), 888
message (*ChannelNotice attribute*), 893
message (*CondaError attribute*), 821, 996
message (*SpinnerBase attribute*), 960
messages() (*in module conda.core.link*), 728
meta_dir (*History attribute*), 837
metadata (*PackageRecord attribute*), 881
MetadataWarning, 925

migrated_channel_aliases (*Context property*), 626
migrated_custom_channels (*Context attribute*), 623
Miniconda, **12**
Miniforge, **12**
minimal_unsatisfiable_subset() (*in module conda.common.logic*), 694
MINIMIZE (*SW attribute*), 669
minimize() (*Clauses method*), 667, 694
MINIO_EXE (*in module conda.testing.gateways.fixtures*), 983
minio_s3_server() (*in module conda.testing.gateways.fixtures*), 983
Mirroring channels, **84**
MISSING (*in module conda.cli.condarc*), 637
missing_files() (*in module conda.plugins.subcommands.doctor.health_checks.missing_files*), 942
missing_pyc_files() (*in module conda.common.path*), 699
missing_pyc_files() (*in module conda.common.path.python*), 697
mkdir_p (*in module conda.gateways.disk.create*), 810
mkdir_p() (*in module conda.gateways.disk*), 816
mkdir_p_sudo_safe() (*in module conda.gateways.disk*), 816
mkdir_p_sudo_safe() (*in module conda.gateways.repodata*), 826
mockable_context_envs_dirs() (*in module conda.base.context*), 621
MockResponse (*class in conda.testing.notices.helpers*), 990
mod (*RepodataState property*), 827
MODIFIED (*Result attribute*), 722
modify_easy_install_pth() (*in module conda.core.initialize*), 724
module
    conda, 553
    conda.__main__, 553
    conda._preview, 553
    conda._preview.env_setup, 553
    conda._preview.env_setup.cli, 554
    conda._preview.env_setup.cli.main_create, 554
    conda._preview.env_setup.cli.main_install, 554
    conda._private, 554
    conda._private.exception_guidance, 554
    conda._private.extract, 556
    conda._private.shards, 556
    conda._private.shards.cache, 556
    conda._private.shards.decompression, 558
    conda._private.shards.misc, 558
    conda._private.shards.shards, 561
    conda._private.shards.subset, 566
    conda._private.shards.typing, 572
    conda._private.zstd, 577
    conda._version, 577
    conda.activate, 577
    conda.api, 585
    conda.auxlib, 591
    conda.auxlib.collection, 591
    conda.auxlib.compat, 592
    conda.auxlib.decorators, 593
    conda.auxlib.entity, 595
    conda.auxlib.exceptions, 607
    conda.auxlib.ish, 607
    conda.auxlib.logz, 608
    conda.auxlib.type_coercion, 609
    conda.base, 611
    conda.base.constants, 611
    conda.base.context, 619
    conda.cli, 629
    conda.cli.actions, 629
    conda.cli.common, 631
    conda.cli.conda_argparse, 632
    conda.cli.condarc, 636
    conda.cli.find_commands, 641
    conda.cli.formats, 641
    conda.cli.helpers, 642
    conda.cli.install, 645
    conda.cli.main, 648
    conda.cli.main_clean, 648
    conda.cli.main_commands, 649
    conda.cli.main_compare, 649
    conda.cli.main_config, 650
    conda.cli.main_create, 651
    conda.cli.main_env, 652
    conda.cli.main_env_config, 652
    conda.cli.main_env_create, 652
    conda.cli.main_env_list, 653
    conda.cli.main_env_remove, 653
    conda.cli.main_env_update, 653
    conda.cli.main_env_vars, 654
    conda.cli.main_export, 654
    conda.cli.main_info, 655
    conda.cli.main_init, 657
    conda.cli.main_install, 658
    conda.cli.main_list, 658
    conda.cli.main_mock_activate, 659
    conda.cli.main_mock_deactivate, 659
    conda.cli.main_notices, 659
    conda.cli.main_package, 660
    conda.cli.main_pip, 661
    conda.cli.main_remove, 661
    conda.cli.main_rename, 661
    conda.cli.main_run, 662
    conda.cli.main_search, 662
    conda.cli.main_update, 663

conda.common, 663
conda.common._logic, 663
conda.common._os, 667
conda.common._os.linux, 667
conda.common._os.osx, 667
conda.common._os.unix, 668
conda.common._os.windows, 668
conda.common.compat, 670
conda.common.configuration, 671
conda.common.constants, 685
conda.common.datetime, 686
conda.common.io, 686
conda.common.iterators, 691
conda.common.logic, 692
conda.common.path, 694
conda.common.path._cygpath, 695
conda.common.path.directories, 696
conda.common.path.python, 696
conda.common.path.windows, 697
conda.common.serialize, 700
conda.common.serialize.json, 700
conda.common.serialize.yaml, 702
conda.common.signals, 704
conda.common.terminal, 704
conda.common.toposort, 706
conda.common.url, 706
conda.core, 711
conda.core.envs_manager, 711
conda.core.exclude_newer, 713
conda.core.index, 715
conda.core.initialize, 719
conda.core.launchers, 724
conda.core.link, 725
conda.core.package_cache_data, 729
conda.core.path_actions, 732
conda.core.portability, 751
conda.core.prefix_data, 755
conda.core.solve, 759
conda.core.subdir_data, 765
conda.deprecations, 770
conda.env, 773
conda.env.env, 773
conda.env.installers, 776
conda.env.installers.base, 776
conda.env.installers.conda, 777
conda.env.installers.pip, 778
conda.env.pip_util, 779
conda.env.specs, 779
conda.env.specs.cep_24, 779
conda.env.specs.explicit, 780
conda.env.specs.requirements, 781
conda.env.specs.yaml_file, 781
conda.exception_handler, 783
conda.exceptions, 784
conda.exports, 798
conda.gateways, 800
conda.gateways.anaconda_client, 800
conda.gateways.connection, 801
conda.gateways.connection.adapters, 801
conda.gateways.connection.adapters.ftp, 801
conda.gateways.connection.adapters.http, 802
conda.gateways.connection.adapters.localfs, 803
conda.gateways.connection.adapters.s3, 804
conda.gateways.connection.download, 805
conda.gateways.connection.session, 806
conda.gateways.disk, 809
conda.gateways.disk.create, 809
conda.gateways.disk.delete, 811
conda.gateways.disk.link, 812
conda.gateways.disk.lock, 812
conda.gateways.disk.permissions, 813
conda.gateways.disk.read, 813
conda.gateways.disk.test, 815
conda.gateways.disk.update, 815
conda.gateways.logging, 816
conda.gateways.repodata, 818
conda.gateways.repodata.lock, 818
conda.gateways.repodata.zstd, 818
conda.gateways.shards, 830
conda.gateways.shards.typing, 830
conda.gateways.streams, 835
conda.gateways.subprocess, 836
conda.history, 836
conda.instructions, 838
conda.misc, 840
conda.models, 841
conda.models.build_string, 841
conda.models.channel, 843
conda.models.dist, 848
conda.models.enums, 850
conda.models.environment, 856
conda.models.match_spec, 860
conda.models.package_info, 869
conda.models.prefix_graph, 870
conda.models.records, 872
conda.models.version, 883
conda.notices, 889
conda.notices.cache, 889
conda.notices.core, 890
conda.notices.fetch, 892
conda.notices.types, 893
conda.notices.views, 894
conda.plugins, 895
conda.plugins.config, 895

conda.plugins.environment_exporters, 896
conda.plugins.environment_exporters.environment_yml, 896
conda.plugins.environment_exporters.explicit, 897
conda.plugins.environment_exporters.requirements_txt, 898
conda.plugins.environment_specifiers, 898
conda.plugins.environment_specifiers.cep_24, 898
conda.plugins.environment_specifiers.environment_yaml, 899
conda.plugins.environment_specifiers.explicit, 899
conda.plugins.environment_specifiers.requirements_txt, 899
conda.plugins.hookspec, 899
conda.plugins.manager, 913
conda.plugins.package_extractors, 923
conda.plugins.package_extractors.conda, 923
conda.plugins.post_solves, 923
conda.plugins.prefix_data_loaders, 923
conda.plugins.prefix_data_loaders.pypi, 923
conda.plugins.prefix_data_loaders.pypi.pkg_format, 923
conda.plugins.previews, 935
conda.plugins.reporter_backends, 935
conda.plugins.reporter_backends.console, 935
conda.plugins.reporter_backends.json, 937
conda.plugins.solvers, 939
conda.plugins.subcommands, 939
conda.plugins.subcommands.doctor, 939
conda.plugins.subcommands.doctor.health_checks, 939
conda.plugins.subcommands.doctor.health_checks.altered_files, 939
conda.plugins.subcommands.doctor.health_checks.conda_envs, 940
conda.plugins.subcommands.doctor.health_checks.environment_txt, 940
conda.plugins.subcommands.doctor.health_checks.find_pkg_level_conflicts, 941
conda.plugins.subcommands.doctor.health_checks.missing_files, 941
conda.plugins.subcommands.doctor.health_checks.multi_channel, 942
conda.plugins.subcommands.doctor.health_checks.requests_ca_bundle, 943
conda.plugins.subcommands.doctor.health_checks.utils, 943
conda.plugins.subcommands.plugins, 949
conda.plugins.subcommands.plugins.info, 949
conda.plugins.subcommands.plugins.install, 950
conda.plugins.subcommands.plugins.list, 950
conda.plugins.subcommands.plugins.package_validation, 951
conda.plugins.subcommands.plugins.remove, 951
conda.plugins.subcommands.plugins.update, 952
conda.plugins.types, 953
conda.plugins.virtual_packages, 967
conda.plugins.virtual_packages.archspec, 967
conda.plugins.virtual_packages.conda, 968
conda.plugins.virtual_packages.cuda, 968
conda.plugins.virtual_packages.freebsd, 969
conda.plugins.virtual_packages.linux, 969
conda.plugins.virtual_packages.osx, 969
conda.plugins.virtual_packages.windows, 970
conda.reporters, 970
conda.resolve, 971
conda.testing, 975
conda.testing.cases, 975
conda.testing.fixtures, 976
conda.testing.gateways, 983
conda.testing.gateways.fixtures, 983
conda.testing.helpers, 984
conda.testing.http_test_server, 987
conda.testing.integration, 988
conda.testing.notices, 989
conda.testing.notices.fixtures, 989
conda.testing.notices.helpers, 989
conda.testing.solver_helpers, 991
conda.utils, 994
module (*CrossPythonEntryPointAction attribute*), 742
module() (*DeprecationHandler method*), 771
module_environment_vars (*Context attribute*), 691
ms_depends() (*Resolve method*), 974
mtime_lock (*PackageCacheData attribute*), 972
msg (*CondaKeyError attribute*), 789
msg (*MissingSpec attribute*), 781
msg (*YamlFileSpec attribute*), 782
MultiChannel (*class in conda.models.channel*), 846
MultiPathAction (*class in conda.core.path_actions*), 735
multiplatform_export (*CondaEnvironmentExporter attribute*), 513, 964
MULTIPLE_USE_KEYS (*PythonDistributionMetadata attribute*), 927

MultipleKeysError, 673
MultiValidationError, 673

N

Name, **54**
name (*Channel attribute*), 824, 844
name (*ChannelNoticeResponse attribute*), 894
name (*CondaAuthHandler attribute*), 510, 957
name (*CondaEnvironmentExporter attribute*), 513, 964
name (*CondaEnvironmentSpecifier attribute*), 964
name (*CondaExceptionObserver attribute*), 531, 967
name (*CondaHealthCheck attribute*), 535, 958
name (*CondaPackageExtractor attribute*), 536, 965
name (*CondaPlugin attribute*), 955
name (*CondaPostCommand attribute*), 537, 957
name (*CondaPostSolve attribute*), 959
name (*CondaPostTransactionAction attribute*), 541, 962
name (*CondaPreCommand attribute*), 538, 957
name (*CondaPrefixDataLoader attribute*), 542, 934, 962
name (*CondaPreSolve attribute*), 959
name (*CondaPreTransactionAction attribute*), 539, 962
name (*CondaReporterBackend attribute*), 961
name (*CondaRequestHeader attribute*), 546, 961
name (*CondaSetting attribute*), 547, 959
name (*CondaSolver attribute*), 547, 956
name (*CondaVirtualPackage attribute*), 550, 956
name (*Dist attribute*), 849
name (*DistDetails attribute*), 848
name (*Environment attribute*), 858
name (*EnvironmentYaml attribute*), 776
name (*Field property*), 601
name (*MatchSpec property*), 864
name (*MultiChannel attribute*), 846
name (*PackageInfo property*), 870
name (*PackageRecord attribute*), 878
name (*PackageRecordDict attribute*), 572
name (*ParameterLoader property*), 683
name (*PreferredEnv attribute*), 869
name (*PrefixData property*), 756, 945
name (*PySpec attribute*), 925
name (*PythonDistribution property*), 926
name (*PythonDistributionMetadata property*), 930
name (*RepodataCache attribute*), 828
name (*TemporaryDirectory attribute*), 810
name_for_alias() (*Configuration method*), 684
name_validation() (*in module conda.env.env*), 774
name_var() (*Clauses method*), 693
namekey (*PackageRecord property*), 880
names (*Clauses attribute*), 693
names (*ParameterLoader property*), 683
names_in_specs() (*in module conda.cli.common*), 631
NAMESPACE_PACKAGE_NAMES (*in module conda.base.constants*), 619
NAMESPACES (*in module conda.base.constants*), 619
NAMESPACES_MAP (*in module conda.base.constants*), 619
needs_line_ending_fix (*_Activator attribute*), 579
needs_line_ending_fix (*CmdExeActivator attribute*), 582
needs_line_ending_fix (*CshActivator attribute*), 581
needs_line_ending_fix (*FishActivator attribute*), 583
needs_line_ending_fix (*PosixActivator attribute*), 580
needs_line_ending_fix (*PowerShellActivator attribute*), 584
needs_line_ending_fix (*XonshActivator attribute*), 582
NEEDS_SUDO (*Result attribute*), 722
netloc (*Url property*), 708
network_fetch_thread() (*in module conda._private.shards.subset*), 571
neutered_specs (*Action attribute*), 734
neutered_specs (*BaseSolver attribute*), 761
neutered_specs (*PrefixSetup attribute*), 726
neutered_specs (*UpdateHistoryAction attribute*), 744
new_precs (*ChangeReport attribute*), 727
new_var() (*Clauses method*), 666, 693
nlst() (*FTPAdapter method*), 802
no_ansi_colors (*DummyArgs attribute*), 990
NO_ASSOC (*ERROR attribute*), 669
NO_CHANGE (*Result attribute*), 722
no_color() (*in module conda.common.terminal*), 705
NO_DEPS (*DepsModifier attribute*), 616
no_link (*PathData attribute*), 877
no_lock (*Context attribute*), 624
no_plugins (*Context attribute*), 625
NO_PLUGINS (*in module conda.base.constants*), 619
Noarch (*class in conda.models.package_info*), 869
noarch (*PackageMetadata attribute*), 870
noarch (*PackageRecord attribute*), 880
noarch (*PackageRecordDict attribute*), 572
NOARCH_GENERIC (*PackageType attribute*), 855
NOARCH_PYTHON (*PackageType attribute*), 855
NoarchField (*class in conda.models.package_info*), 869
NoarchField (*class in conda.models.records*), 874
NoarchType (*class in conda.models.enums*), 856
NoBaseEnvironmentError, 787
NoChannelsConfiguredError, 792
Node (*class in conda._private.shards.subset*), 567
node_count (*RepodataSubset property*), 568, 832
NodeId (*class in conda._private.shards.subset*), 567
non_admin_enabled (*Context attribute*), 622
NON_SPACE (*in module conda.plugins.prefix_data_loaders.pypi.pkg_format*), 931
non_x86_linux_machines (*in module conda.exports*), 799
non_x86_machines (*in module conda.base.context*), 620
NoneType (*in module conda.common.compat*), 671

norm_name (*PythonDistribution property*), 926
norm_package_name() (*in module conda.plugins.prefix_data_loaders.pypi.pkg_format*), 930
norm_package_version() (*in module conda.plugins.prefix_data_loaders.pypi.pkg_format*), 930
norm_version (*VersionOrder attribute*), 886
normalize() (*_ChannelSelector static method*), 713
normalize_timestamp_seconds() (*in module conda.common.datetime*), 686
normalized_version() (*in module conda.models.version*), 884
NoSpaceLeftError, 795
Not() (*Clauses method*), 666, 694
NOT_SET (*DepsModifier attribute*), 616
nothing_to_do (*UnlinkLinkTransaction property*), 727
NoticeLevel (*class in conda.base.constants*), 618
notices (*ChannelNoticeResponse property*), 894
notices() (*in module conda.notices*), 894
notices() (*in module conda.notices.core*), 891
notices_cache_dir() (*in module conda.testing.notices.fixtures*), 989
NOTICES_CACHE_FN (*in module conda.base.constants*), 614
NOTICES_CACHE_SUBDIR (*in module conda.base.constants*), 614
notices_decorator_assert_message_in_stdout() (*in module conda.testing.notices.helpers*), 990
NOTICES_DECORATOR_DISPLAY_INTERVAL (*in module conda.base.constants*), 614
NOTICES_DECORATOR_DISPLAY_INTERVAL_NS (*in module conda.base.constants*), 614
NOTICES_FN (*in module conda.base.constants*), 614
notices_mock_fetch_get_session() (*in module conda.testing.notices.fixtures*), 989
notify_outdated_conda (*Context attribute*), 621
notify_success (*TryRepodata attribute*), 646
NotWritableError, 794, 823
now (*_CutoffParser attribute*), 713
now (*ExcludeNewerPolicy attribute*), 714
NoWritableEnvsDirError, 794
NoWritablePkgsDirError, 794
nt_to_posix() (*in module conda.common.path._cygpath*), 695
NULL (*in module conda.common.constants*), 685
nullable (*Field property*), 601
NullCountAction (*class in conda.cli.actions*), 629
NullHandler (*in module conda.auxlib.logz*), 608
number_channel_notices (*Context attribute*), 624
NumberField (*class in conda.auxlib.entity*), 602
numberify() (*in module conda.auxlib.type_coercion*), 609

O

object_log() (*History method*), 838
ObjectLoadedParameter (*class in conda.common.configuration*), 679
ObjectParameter (*class in conda.common.configuration*), 682
offending_arguments (*TooManyArgumentsError attribute*), 786
offline (*CondaExceptionEvent attribute*), 530, 966
offline (*Context attribute*), 624
offline_keep() (*in module conda.models.channel*), 848
offline_nofetch_thread() (*in module conda._private.shards.subset*), 571
OfflineError, 797
offset_cache_file_mtime() (*in module conda.testing.notices.helpers*), 990
OK_MARK (*in module conda.base.constants*), 619
on_linux (*in module conda.common.compat*), 670
on_mac (*in module conda.common.compat*), 670
on_win (*in module conda.common._os*), 670
on_win (*in module conda.common.compat*), 670
on_win (*in module conda.gateways.disk*), 816
ONLY_DEPS (*DepsModifier attribute*), 616
OOM (*ERROR attribute*), 669
OP_ORDER (*in module conda.instructions*), 840
open_utf8() (*in module conda.common.compat*), 671
openbsd (*Platform attribute*), 852
OperationNotAllowed, 790
operations (*Evaluator attribute*), 931
OPERATOR_MAP (*in module conda.models.version*), 887
operator_match() (*BaseSpec method*), 888
OPERATOR_START (*in module conda.models.version*), 887
optional (*MatchSpec property*), 863
optional_cutoff() (*_CutoffParser method*), 713
OPTIONAL_KEYS (*in module conda.env.env*), 774
optional_message (*TooManyArgumentsError attribute*), 786
OR (*in module conda.plugins.prefix_data_loaders.pypi.pkg_format*), 931
Or() (*Clauses method*), 666, 694
original_spec_str (*MatchSpec property*), 863
os_distribution_name_version() (*Context method*), 627
osx (*Platform attribute*), 852
osx_version() (*in module conda.plugins.virtual_packages.osx*), 969
override_channels_enabled (*Context attribute*), 623
override_entity (*CondaVirtualPackage attribute*), 550, 956
override_virtual_packages (*Context property*), 627

P

P (*in module conda.plugins.manager*), 914

package (*AnnotatedRawShard attribute*), 557
package (*Node attribute*), 567
package (*NodeId attribute*), 567
package (*ShardFetch attribute*), 561
Package search and install, **116**
Package specification, **116**
package_cache() (*in module conda.exports*), 799
PACKAGE_CACHE_MAGIC_FILE (*in module conda.base.constants*), 619
package_cutoffs (*ExcludeNewerPolicy attribute*), 714
package_dict() (*in module conda.testing.solver_helpers*), 991
PACKAGE_ENV_VARS_DIR (*in module conda.base.constants*), 619
package_filename (*_SplitUrlParts attribute*), 710
package_filename (*Channel attribute*), 824, 844
package_filename (*MultiChannel attribute*), 846
package_info (*CompileMultiPycAction attribute*), 740
package_info (*CreateInPrefixPathAction attribute*), 737
package_is_installed() (*in module conda.testing.integration*), 989
package_metadata (*PackageInfo attribute*), 870
package_metadata_version (*PackageMetadata attribute*), 870
package_names (*ShardBase property*), 563, 574
package_names (*ShardLike property*), 564, 575
package_names (*Shards property*), 565, 576, 831
package_size() (*PrefixRecord method*), 883
package_string() (*in module conda.testing.solver_helpers*), 991
package_string_set() (*in module conda.testing.solver_helpers*), 991
package_tarball_full_path (*PackageCacheRecord attribute*), 881
package_tarball_full_path (*PackageInfo attribute*), 870
package_tarball_full_path (*PrefixRecord attribute*), 883
package_type (*PackageRecord attribute*), 880
PackageCacheData (*class in conda.api*), 588
PackageCacheData (*class in conda.core.package_cache_data*), 730
PackageCacheData (*class in conda.gateways.repodata*), 822
PackageCacheRecord (*class in conda.models.records*), 881
PackageCacheType (*class in conda.core.package_cache_data*), 729
PackageInfo (*class in conda.models.package_info*), 870
PackageMetadata (*class in conda.models.package_info*), 870
packagename (*EnvironmentFileNotDownloaded attribute*), 796
PackageNotInstalledError, 791
PackageRecord (*class in conda.models.records*), 877
PackageRecordDict (*class in conda._private.shards.typing*), 572
PackageRecordList (*class in conda.core.subdir_data*), 766
Packages, **116**
packages (*PackagesNotFoundError attribute*), 791
packages_index (*Shards property*), 565, 576
PackagesNotFoundError, 791
PackagesNotFoundInChannelsError, 791
PackagesNotFoundInPrefixError, 791
PackageType (*class in conda.models.enums*), 854
PackageTypeField (*class in conda.models.records*), 877
PaddingError, 788
PaddingError (*in module conda.exports*), 799
pair (*Dist property*), 849
paired_actions (*ProgressiveFetchExtract attribute*), 731
Parameter (*class in conda.common.configuration*), 679
parameter (*CondaSetting attribute*), 547, 959
parameter_description_builder() (*in module conda.cli.main_config*), 650
parameter_name (*ValidationError attribute*), 673
parameter_names (*PluginConfig attribute*), 895
parameter_names_and_aliases (*PluginConfig attribute*), 895
parameter_value (*ValidationError attribute*), 673
ParameterFlag (*class in conda.common.configuration*), 673
ParameterLoader (*class in conda.common.configuration*), 682
ParameterTypeGroups (*class in conda.cli.condarc*), 637
parametrized_solver_fixture() (*in module conda.testing.fixtures*), 977
parse() (*Dependencies method*), 776
parse() (*History method*), 837
parse_args() (*ArgumentParser method*), 634
parse_conda_channel_url() (*in module conda.models.channel*), 847
parse_datetime_to_timestamp() (*in module conda.common.datetime*), 686
parse_dist_name() (*Dist static method*), 849
parse_duration() (*in module conda.common.datetime*), 686
parse_entry_point_def() (*in module conda.common.path*), 699
parse_entry_point_def() (*in module conda.common.path.python*), 697
parse_marker() (*in module conda.plugins.prefix_data_loaders.pypi.pkg_format*), 931
parse_specification() (*in module*

conda.plugins.prefix_data_loaders.pypi.pkg_format), 930
ParseError, 790
PARTIAL_EXTENSION (*in module conda.base.constants*), 614
PARTIAL_PYPI_SPEC_PATTERN (*in module conda.plugins.prefix_data_loaders.pypi.pkg_format*), 924
path (*_SplitUrlParts attribute*), 710
path (*ConfigurationFile property*), 639
path (*History attribute*), 837
path (*LinkScriptError attribute*), 788
path (*PathData property*), 877
path_conflict (*ClobberError attribute*), 786
path_conflict (*Context attribute*), 622
path_conversion (*_Activator attribute*), 578
path_conversion (*CmdExeActivator attribute*), 582
path_conversion (*CshActivator attribute*), 581
path_conversion (*FishActivator attribute*), 583
path_conversion (*PosixActivator attribute*), 580
path_conversion (*PowerShellActivator attribute*), 583
path_conversion (*XonshActivator attribute*), 581
path_factory (*TmpChannelFixture attribute*), 981
path_factory (*TmpEnvFixture attribute*), 980
path_factory (*TmpRepodataChannelFixture attribute*), 981
path_factory() (*in module conda.testing.fixtures*), 980
path_is_clean() (*in module conda.gateways.disk.delete*), 812
PATH_NOT_FOUND (*ERROR attribute*), 669
path_to_url() (*in module conda.common.url*), 707
path_type (*PathData attribute*), 877
path_var_tmpl (*_Activator attribute*), 579
path_var_tmpl (*CmdExeActivator attribute*), 583
path_var_tmpl (*CshActivator attribute*), 581
path_var_tmpl (*FishActivator attribute*), 583
path_var_tmpl (*PosixActivator attribute*), 581
path_var_tmpl (*PowerShellActivator attribute*), 584
path_var_tmpl (*XonshActivator attribute*), 582
PathAction (*class in conda.core.path_actions*), 734
PathConflict (*class in conda.base.constants*), 615
PathData (*class in conda.models.records*), 877
PathDataV1 (*class in conda.models.records*), 877
PathEnum (*class in conda.models.enums*), 854
PathFactoryFixture (*class in conda.testing.fixtures*), 979
PathNotFoundError, 787
paths (*PathsData attribute*), 877
paths_data (*PackageInfo attribute*), 870
paths_data (*PrefixRecord attribute*), 883
paths_on_same_device() (*in module conda.gateways.disk.test*), 815
paths_version (*PathsData attribute*), 877
PathsData (*class in conda.models.records*), 877
pathsep_join (*_Activator attribute*), 578
pathsep_join (*CmdExeActivator attribute*), 582
pathsep_join (*CshActivator attribute*), 581
pathsep_join (*FishActivator attribute*), 583
pathsep_join (*JSONFormatMixin attribute*), 584
pathsep_join (*PosixActivator attribute*), 580
pathsep_join (*PowerShellActivator attribute*), 583
pathsep_join (*XonshActivator attribute*), 581
percent_decode() (*in module conda.common.url*), 707
Performance, **116**
PHANDLE (*in module conda.common._os.windows*), 668
pinned_packages (*Context attribute*), 622
pinned_packages (*EnvironmentConfig attribute*), 857
pinned_specs() (*SolverStateContainer method*), 764
pinned_well_formatted_check() (*in module conda.plugins.subcommands.doctor.health_checks.pinned*), 942
Pip interoperability (*experimental*), **84**
pip_cli() (*in module conda.testing.fixtures*), 979
pip_installed_post_parse_hook() (*in module conda.cli.main_pip*), 661
pip_subprocess() (*in module conda.env.pip_util*), 779
PipCLIFixture (*class in conda.testing.fixtures*), 979
pkg_data (*ActionGroup attribute*), 726
pkgs_dir (*PackageCacheData attribute*), 730, 822
pkgs_dir (*UrlsData attribute*), 731
pkgs_dirs (*Context property*), 626
pkgs_dirs (*in module conda.exports*), 799
platform (*_SplitUrlParts attribute*), 710
platform (*Channel attribute*), 824, 844
Platform (*class in conda.models.enums*), 851
platform (*Context property*), 625
platform (*Dist attribute*), 849
platform (*Environment attribute*), 858
platform (*in module conda.exports*), 799
platform (*MultiChannel attribute*), 846
platform (*PackageRecord attribute*), 880
PLATFORM_DIRECTORIES (*in module conda.base.constants*), 613
platform_system_release() (*Context method*), 627
PlatformMismatchError, 793
PLATFORMS (*in module conda.base.constants*), 613
plugin_aliases (*CondaPluginManager attribute*), 914
plugin_manager (*Context property*), 625
plugin_map_parameters (*ParameterTypeGroups property*), 638
plugin_sequence_parameters (*ParameterTypeGroups property*), 637
PluginConfig (*class in conda.plugins.config*), 895
PluginError, 796
Plugins, **116**, **417**
plugins (*Context property*), 625
plugins (*in module conda.plugins.environment_exporters*), 898

plugins (*in module conda.plugins.environment_specifiers*), 899
plugins (*in module conda.plugins.package_extractors*), 923
plugins (*in module conda.plugins.post_solves*), 923
plugins (*in module conda.plugins.prefix_data_loaders*), 935
plugins (*in module conda.plugins.reporter_backends*), 938
plugins (*in module conda.plugins.subcommands*), 953
plugins (*in module conda.plugins.subcommands.doctor.health_checks*), 943
plugins (*in module conda.plugins.virtual_packages*), 970
pop() (*ContextStack method*), 628
pop_key() (*in module conda.common.toposort*), 706
POPULAR_ENCODINGS (*in module conda.core.portability*), 752
port (*_SplitUrlParts attribute*), 710
port (*HttpTestServerFixture attribute*), 982
posix_to_nt() (*in module conda.common.path._cygpath*), 695
PosixActivator (*class in conda.activate*), 580
post_build_validation() (*Configuration method*), 685
post_build_validation() (*Context method*), 625
PowerShellActivator (*class in conda.activate*), 583
ppc64 (*Arch attribute*), 851
ppc64le (*Arch attribute*), 851
prec (*LinkScriptError attribute*), 788
preferred_env (*PackageMetadata attribute*), 870
preferred_env (*PackageRecord attribute*), 880
PreferredEnv (*class in conda.models.package_info*), 869
prefix (*BaseSolver attribute*), 761
prefix (*ChangeReport attribute*), 727
prefix (*Environment attribute*), 858
prefix (*EnvironmentYaml attribute*), 776
prefix (*History attribute*), 837
prefix (*SolverStateContainer attribute*), 764
prefix_action_groups (*UnlinkLinkTransaction attribute*), 727
PREFIX_CREATION_TIMESTAMP_FILE (*in module conda.base.constants*), 619
prefix_data() (*SolverStateContainer method*), 764
prefix_data_interoperability (*Context attribute*), 622
PREFIX_FROZEN_FILE (*in module conda.base.constants*), 619
PREFIX_LAST_ACTIVATED_FILE (*in module conda.base.constants*), 619
PREFIX_MAGIC_FILE (*in module conda.base.constants*), 619
PREFIX_NAME_DISALLOWED_CHARS (*in module conda.base.constants*), 614
prefix_path (*PrefixData attribute*), 755, 944
prefix_path_data (*CompileMultiPycAction attribute*), 740
prefix_path_data (*CreatePythonEntryPointAction attribute*), 742
prefix_path_data (*LinkPathAction attribute*), 738
prefix_paths_data (*CompileMultiPycAction attribute*), 740
PREFIX_PINNED_FILE (*in module conda.base.constants*), 619
PREFIX_PLACEHOLDER (*in module conda.base.constants*), 613
prefix_placeholder (*PathData attribute*), 877
prefix_placeholder (*PrefixReplaceLinkAction attribute*), 739
prefix_record_groups (*PrefixActionGroup attribute*), 727
prefix_setups (*UnlinkLinkTransaction attribute*), 727
prefix_specified (*Context property*), 626
PREFIX_STATE_FILE (*in module conda.base.constants*), 619
prefix_validation() (*in module conda.env.env*), 774
PrefixActionGroup (*class in conda.core.link*), 726
PrefixData (*class in conda.api*), 589
PrefixData (*class in conda.core.prefix_data*), 755
PrefixData (*class in conda.plugins.subcommands.doctor*), 944
PrefixDataType (*class in conda.core.prefix_data*), 755
PrefixGraph (*class in conda.models.prefix_graph*), 871
PrefixGraph (*class in conda.plugins.prefix_data_loaders.pypi*), 933
PrefixPathAction (*class in conda.core.path_actions*), 735
PrefixRecord (*class in conda.models.records*), 882
PrefixRecordDict (*class in conda.core.prefix_data*), 755
PrefixReplaceLinkAction (*class in conda.core.path_actions*), 738
PrefixSetup (*class in conda.core.link*), 726
prepare() (*ProgressiveFetchExtract method*), 731
prepare() (*UnlinkLinkTransaction method*), 727
prepare_request() (*CondaSession method*), 808
pretty_content() (*in module conda.history*), 837
pretty_diff() (*in module conda.history*), 837
pretty_json() (*Entity method*), 606
pretty_list() (*in module conda.common.configuration*), 672
pretty_map() (*in module conda.common.configuration*), 672
pretty_package() (*in module conda.cli.main_info*), 656
pretty_record() (*in module conda.cli.main_search*),

662
prev_time (*DeltaSecondsFormatter attribute*), 688
prevent (*PathConflict attribute*), 616
Prevent() (*Clauses method*), 666, 693
preview (*Context attribute*), 624
preview_enabled() (*Context method*), 627
PREVIEW_LABEL (*in module conda._preview.env_setup*), 554
PREVIEW_PLUGIN_NAME (*in module conda.plugins.previews*), 935
primitive_types (*in module conda.common.compat*), 671
PrimitiveLoadedParameter (*class in conda.common.configuration*), 677
PrimitiveParameter (*class in conda.common.configuration*), 680
PRINT (*in module conda.instructions*), 839
print_activate() (*in module conda.cli.common*), 632
PRINT_CMD() (*in module conda.instructions*), 839
print_conda_exception() (*in module conda.exceptions*), 797
print_config_item() (*in module conda.cli.main_config*), 651
print_expected_error_report() (*ExceptionHandler method*), 784
print_explicit() (*in module conda.cli.main_list*), 658
print_export_header() (*in module conda.cli.main_list*), 658
print_instrumentation_data() (*in module conda.common.io*), 691
print_log() (*History method*), 838
print_more_notices_message() (*in module conda.notices.views*), 894
print_notice_message() (*in module conda.notices.views*), 894
print_notices() (*in module conda.notices.views*), 894
print_packages() (*in module conda.cli.main_list*), 658
print_plan_results() (*in module conda.core.initialize*), 723
print_result() (*in module conda.env.env*), 776
print_transaction_summary() (*UnlinkLinkTransaction method*), 728
print_unexpected_error_report() (*ExceptionHandler method*), 784
prioritize_channels() (*in module conda.models.channel*), 847
process_solution() (*_PycoSatSolver method*), 665
process_solution() (*_PyCryptoSatSolver method*), 666
process_solution() (*_PySatSolver method*), 666
process_solution() (*_SatSolver method*), 665
Programmatic .condarc API, **484**
PROGRESS (*in module conda.instructions*), 839
progress_bar() (*ConsoleReporterRenderer method*), 937
progress_bar() (*JSONReporterRenderer method*), 938
progress_bar() (*ReporterRendererBase method*), 960
progress_bar_context_manager() (*ReporterRendererBase class method*), 960
PROGRESS_COMMANDS (*in module conda.instructions*), 839
ProgressBarBase (*class in conda.plugins.types*), 959
ProgressiveFetchExtract (*class in conda.core.package_cache_data*), 731
PROGRESSIVEFETCHEXTRACT (*in module conda.instructions*), 839
PROGRESSIVEFETCHEXTRACT_CMD() (*in module conda.instructions*), 839
prompt() (*ConsoleReporterRenderer method*), 937
prompt() (*JSONReporterRenderer method*), 938
prompt() (*ReporterRendererBase method*), 960
protect_frozen_envs (*Context attribute*), 621
proxy_manager_for() (*_SSLContextAdapterMixin method*), 803
proxy_servers (*Context attribute*), 623
ProxyError, 789, 823
prune (*SolverStateContainer attribute*), 764
prune() (*PrefixGraph method*), 871, 933
push() (*ContextStack method*), 628
push_MatchSpec() (*Resolve method*), 974
PY_FILE_RE (*in module conda.plugins.prefix_data_loaders.pypi.pkg_format*), 925
pyc_file (*PathEnum attribute*), 854
pyc_path() (*in module conda.common.path*), 699
pyc_path() (*in module conda.common.path.python*), 697
PYCOSAT (*SatSolverChoice attribute*), 618
PycoSatSolver (*in module conda.common.logic*), 693
PYCRYPTOSAT (*SatSolverChoice attribute*), 618
PyCryptoSatSolver (*in module conda.common.logic*), 693
PYPI_CONDA_DEPS (*in module conda.plugins.prefix_data_loaders.pypi.pkg_format*), 924
pypi_name_to_conda_name() (*in module conda.plugins.prefix_data_loaders.pypi.pkg_format*), 930
PYPI_TO_CONDA (*in module conda.plugins.prefix_data_loaders.pypi.pkg_format*), 924
PYSAT (*SatSolverChoice attribute*), 618
PySatSolver (*in module conda.common.logic*), 693
PySpec (*class in conda.plugins.prefix_data_loaders.pypi.pkg_format*), 925
python (*NoarchType attribute*), 856
PYTHON_BINARY (*in module conda.testing.integration*), 988

python_entry_point_template (*in module conda.gateways.disk.create*), 811
python_implementation_name_version() (*Context method*), 627
python_site_packages_path (*PackageRecord attribute*), 880
python_version (*PythonDistribution attribute*), 925
PythonDistribution (*class in conda.plugins.prefix_data_loaders.pypi.pkg_format*), 925
PythonDistributionMetadata (*class in conda.plugins.prefix_data_loaders.pypi.pkg_format*), 927
PythonEggInfoDistribution (*class in conda.plugins.prefix_data_loaders.pypi.pkg_format*), 926
PythonEggLinkDistribution (*class in conda.plugins.prefix_data_loaders.pypi.pkg_format*), 927
PythonInstalledDistribution (*class in conda.plugins.prefix_data_loaders.pypi.pkg_format*), 926
PYTHONPATH() (*in module conda.testing.fixtures*), 982

Q

quad (*Dist property*), 849
query (*_SplitUrlParts attribute*), 710
query() (*PackageCacheData method*), 588, 730, 822
query() (*PrefixData method*), 590, 758, 947
query() (*SubdirData method*), 587, 767
query_all() (*in module conda.core.subdir_data*), 770
query_all() (*PackageCacheData class method*), 730, 822
query_all() (*PackageCacheData static method*), 589
query_all() (*SubdirData static method*), 587, 766
query_all_prefixes() (*in module conda.core.envs_manager*), 712
queue (*QueueCache attribute*), 570
QUEUE_TIMEOUT (*in module conda._private.shards.subset*), 567
QueueCache (*class in conda._private.shards.subset*), 570
quiet (*CondaExceptionEvent attribute*), 530, 966
quiet (*Context attribute*), 624
QuietProgressBar (*class in conda.plugins.reporter_backends.console*), 936
QuietSpinner (*class in conda.plugins.reporter_backends.console*), 936
quote_for_shell() (*in module conda.utils*), 994

R

r (*SolverStateContainer attribute*), 764
Raise() (*in module conda.auxlib.exceptions*), 607
raise_errors() (*in module conda.common.configuration*), 673
raise_exc (*MockResponse attribute*), 990
raises() (*in module conda.testing.helpers*), 985
raw (*Dependencies attribute*), 776
raw_data (*Configuration attribute*), 684
raw_data (*PluginConfig property*), 896
raw_parameters_from_single_source() (*ParameterLoader static method*), 683
raw_value (*BaseSpec property*), 888
raw_value (*MatchInterface property*), 867
RawParameter (*class in conda.common.configuration*), 674
rc (*Response attribute*), 836
RE_UNIX_DRIVE (*in module conda.common.path._cygpath*), 696
RE_UNIX_MOUNT (*in module conda.common.path._cygpath*), 696
RE_UNIX_ROOT (*in module conda.common.path._cygpath*), 696
RE_WIN_DRIVE (*in module conda.common.path._cygpath*), 695
RE_WIN_MOUNT (*in module conda.common.path._cygpath*), 695
reachable() (*RepodataSubset method*), 568, 833
reachable_bfs() (*RepodataSubset method*), 569, 833
reachable_pipelined() (*RepodataSubset method*), 569, 833
REACHABLE_PIPELINED_MAX_TIMEOUTS (*in module conda._private.shards.subset*), 567
reactivate() (*_Activator method*), 579
read() (*ConfigurationFile method*), 639
read() (*in module conda.common.serialize.json*), 702
read() (*in module conda.common.serialize.yaml*), 703
read_binstar_tokens() (*in module conda.gateways.anaconda_client*), 800
read_cache() (*RepodataFetch method*), 829
read_has_prefix() (*in module conda.gateways.disk.read*), 814
read_icondata() (*in module conda.gateways.disk.read*), 814
read_index_json() (*in module conda.gateways.disk.read*), 814
read_index_json_from_tarball() (*in module conda.gateways.disk.read*), 814
read_no_link() (*in module conda.gateways.disk.read*), 814
read_only_caches() (*PackageCacheData class method*), 730, 822
read_package_info() (*in module conda.gateways.disk.read*), 814
read_package_metadata() (*in module conda.gateways.disk.read*), 814
read_paths_json() (*in module*

conda.gateways.disk.read), 814
read_python_record() (*in module conda.plugins.prefix_data_loaders.pypi*), 934
read_python_record() (*in module conda.plugins.prefix_data_loaders.pypi.pkg_format*), 932
read_repodata_json() (*in module conda.gateways.disk.read*), 814
read_soft_links() (*in module conda.gateways.disk.read*), 814
reason (*CouldntParseError attribute*), 790
received (*TooManyArgumentsError attribute*), 786
RECOGNIZED_URL_SCHEMES (*in module conda.base.constants*), 613
record() (*in module conda.testing.helpers*), 986
record_file (*time_recorder attribute*), 691
record_id() (*PackageRecord method*), 881
record_or_spec (*ExtractPackageAction attribute*), 751
records (*PrefixGraph property*), 871, 933
recursive_make_writable() (*in module conda.gateways.disk.permissions*), 813
redact_token_urls() (*in module conda.gateways.streams*), 835
ReducedIndex (*class in conda.core.index*), 718
refresh() (*JSONProgressBar method*), 938
refresh() (*ProgressBarBase method*), 960
refresh() (*QuietProgressBar method*), 936
refresh() (*RepodataCache method*), 828
refresh() (*TQDMProgressBar method*), 936
regex_match() (*BaseSpec method*), 888
regex_split_re (*in module conda.models.version*), 887
register() (*_ConstantDeprecationRegistry method*), 773
register() (*CondaPluginManager method*), 915
register_action_groups (*PrefixActionGroup attribute*), 727
register_env() (*in module conda.core.envs_manager*), 712
register_envs (*Context attribute*), 621
register_reset_callback() (*Configuration method*), 685
register_reset_callback() (*Configuration method*), 685
RegisterEnvironmentLocationAction (*class in conda.core.path_actions*), 744
reinstall_packages() (*in module conda.cli.install*), 646
Releasing, **417**
reload() (*Index method*), 717
reload() (*PackageCacheData method*), 589, 730, 822
reload() (*PrefixData method*), 590, 758, 947
reload() (*SubdirData method*), 588, 767
remote_backoff_factor (*Context attribute*), 623
remote_connect_timeout_secs (*Context attribute*), 623
remote_max_retries (*Context attribute*), 623
remote_read_timeout_secs (*Context attribute*), 623
REMOVE (*Commands attribute*), 988
remove() (*in module conda.cli.main_package*), 660
remove() (*PackageCacheData method*), 730, 822
remove() (*PrefixData method*), 758, 947
remove() (*Resolve method*), 974
remove() (*SimpleEnvironment method*), 992
remove_all_plugin_settings() (*PluginConfig class method*), 896
remove_auth() (*in module conda.common.url*), 711
remove_binstar_token() (*in module conda.gateways.anaconda_client*), 800
remove_cache() (*ShardCache method*), 558
remove_channels() (*EnvironmentYaml method*), 776
remove_conda_in_sp_dir() (*in module conda.core.initialize*), 724
remove_empty_parent_paths() (*in module conda.gateways.disk.delete*), 812
remove_item() (*ConfigurationFile method*), 640
remove_key() (*ConfigurationFile method*), 641
remove_menu_action_groups (*PrefixActionGroup attribute*), 726
remove_spec() (*PrefixGraph method*), 871, 933
remove_specs (*Action attribute*), 734
remove_specs (*PrefixSetup attribute*), 726
remove_specs (*UpdateHistoryAction attribute*), 743
remove_specs() (*Resolve method*), 974
remove_youngest_descendant_nodes_with_specs() (*PrefixGraph method*), 871, 933
removed_precs (*ChangeReport attribute*), 727
RemoveError, 792
RemoveFromPrefixPathAction (*class in conda.core.path_actions*), 745
RemoveLinkedPackageRecordAction (*class in conda.core.path_actions*), 747
RemoveMenuAction (*class in conda.core.path_actions*), 746
rename() (*in module conda.gateways.disk.update*), 815
rename_context() (*in module conda.gateways.disk.update*), 816
render() (*ConsoleReporterRenderer method*), 937
render() (*in module conda.reporters*), 970
render() (*InfoRenderer method*), 657
render() (*JSONReporterRenderer method*), 938
render() (*ReporterRendererBase method*), 960
renderer (*CondaReporterBackend attribute*), 961
Rendering output, **484**
replace() (*ContextStack method*), 628
replace() (*RepodataCache method*), 828
replace() (*Url method*), 708

replace_context() (*in module conda.base.context*), 628
replace_context_default() (*in module conda.base.context*), 629
replace_first_api_with_conda() (*in module conda.gateways.anaconda_client*), 800
replace_long_shebang() (*in module conda.core.portability*), 754
replace_prefix() (*in module conda.core.portability*), 753
replace_pyzzer_entry_point_shebang() (*in module conda.core.portability*), 754
repo_cache (*RepodataFetch property*), 829
repo_cache (*SubdirData property*), 767
REPO_DATA_KEYS (*SimpleEnvironment attribute*), 991
repo_fetch (*SubdirData property*), 767
repo_interface_cls (*RepodataFetch attribute*), 829
repo_packages (*SimpleEnvironment attribute*), 991
repodata (*TryRepodata attribute*), 646
repodata() (*CondaRepoInterface method*), 827
repodata() (*RepoInterface method*), 826
repodata() (*ZstdRepoInterface method*), 819
REPODATA_FN (*in module conda.base.constants*), 614
REPODATA_FN (*in module conda.gateways.repodata*), 822
repodata_fn (*RepodataCache attribute*), 828
repodata_fn (*RepodataFetch attribute*), 829
repodata_fn (*RepodataState attribute*), 827
repodata_fn (*SubdirData attribute*), 767
repodata_fns (*Context attribute*), 623
repodata_fns (*EnvironmentConfig attribute*), 857
repodata_fns (*Repodatas attribute*), 646
REPODATA_HEADER_RE (*in module conda.core.subdir_data*), 765
repodata_no_packages (*ShardBase attribute*), 563, 574
repodata_no_packages (*ShardLike attribute*), 564, 575
repodata_no_packages (*Shards attribute*), 564, 576
repodata_parsed() (*ZstdRepoInterface method*), 820
REPODATA_PICKLE_VERSION (*in module conda.core.subdir_data*), 765
repodata_record (*PackageInfo attribute*), 870
REPODATA_SHARDS_FN (*in module conda.base.constants*), 614
REPODATA_SHARDS_FN (*in module conda.gateways.repodata*), 822
repodata_threads (*Context property*), 625
repodata_use_shards (*Context attribute*), 624
repodata_use_zst (*Context attribute*), 624
repodata_version (*RepodataDict attribute*), 573
RepodataCache (*class in conda.gateways.repodata*), 828
RepodataDict (*class in conda._private.shards.typing*), 573
RepodataFetch (*class in conda.gateways.repodata*), 829
RepodataInfoDict (*class in conda._private.shards.typing*), 573
RepodataIsEmpty, 826
RepodataOnDisk, 826
Repodatas (*class in conda.cli.install*), 646
RepodataState (*class in conda.gateways.repodata*), 827
RepodataSubset (*class in conda._private.shards.subset*), 568
RepodataSubset (*class in conda.gateways.shards*), 832
RepoInterface (*class in conda.gateways.repodata*), 826
RepoInterface (*SubdirData attribute*), 767
report_errors (*Context property*), 625
reportable (*CondaError attribute*), 821, 996
ReporterRendererBase (*class in conda.plugins.types*), 960
REPR_IGNORE_KWARGS (*in module conda.core.path_actions*), 733
request_header_sort_dict (*in module conda.auxlib.logz*), 609
request_header_sort_key() (*in module conda.auxlib.logz*), 609
request_url_zstd_state() (*in module conda.gateways.repodata.zstd*), 819
requested_extras (*Node attribute*), 567
requested_link_type (*CreatePrefixRecordAction attribute*), 743
requested_packages (*Environment attribute*), 858
requested_spec (*CreatePrefixRecordAction attribute*), 743
requested_spec (*SolvedRecord attribute*), 882
requested_specs (*SolvedRecord property*), 882
requests_ca_bundle_check() (*in module conda.plugins.subcommands.doctor.health_checks.requests_ca_b*
943
requests_version() (*Context method*), 627
Require() (*Clauses method*), 666, 693
require_explicit_plugin_packages() (*in module conda.plugins.subcommands.plugins.package_validation*), 951
require_installed_plugin_specs() (*in module conda.plugins.subcommands.plugins.package_validation*), 951
require_plugin_transaction() (*in module conda.plugins.subcommands.plugins.package_validation*), 951
require_plugin_update_environment() (*in module conda.plugins.subcommands.plugins.package_validation*), 951
required (*Field property*), 601
REQUIRED_KEYS (*in module conda.env.env*), 774
REQUIREMENTS_FORMAT (*in module conda.plugins.environment_exporters.requirements_txt*), 898
RequirementsSpec (*class in*

conda.env.specs.requirements), 781
REQUIRES_FILES (*PythonDistribution attribute*), 925
REQUIRES_FILES (*PythonEggInfoDistribution attribute*), 927
REQUIRES_FILES (*PythonInstalledDistribution attribute*), 926
RESERVED_ENV_NAMES (*in module conda.base.constants*), 613
RESERVED_ENV_VARS (*in module conda.base.constants*), 619
reset_conda_context() (*in module conda.testing.fixtures*), 977
reset_context() (*in module conda.base.context*), 628
Resolve (*class in conda.resolve*), 971
resolve_paths() (*in module conda.common.path._cygpath*), 696
resolved_short_path() (*in module conda.plugins.prefix_data_loaders.pypi*), 934
ResolvePackageNotFound, 784
ResolvePackageNotFound (*in module conda.resolve*), 971
Response (*class in conda.gateways.subprocess*), 836
Response304ContentUnchanged, 827
response_header_sort_dict (*in module conda.auxlib.logz*), 609
response_header_sort_key() (*in module conda.auxlib.logz*), 609
RESTORE (*SW attribute*), 669
restore_bad() (*Resolve method*), 974
restore_state() (*_ClauseArray method*), 664
restore_state() (*_ClauseList method*), 664
restore_state() (*_SatSolver method*), 665
Result (*class in conda.core.initialize*), 722
retr() (*FTPAdapter method*), 802
RETRIES (*in module conda.gateways.connection.session*), 807
retrieve() (*ShardCache method*), 558
retrieve_multiple() (*ShardCache method*), 558
retrieve_notices() (*in module conda.notices.core*), 891
return_code (*ArgumentError attribute*), 785
return_code (*CondaError attribute*), 821, 996
return_code (*CondaExceptionEvent attribute*), 530, 966
return_code (*CondaExitZero attribute*), 997
return_code (*Help attribute*), 785
reverse() (*Action method*), 734
reverse() (*CacheUrlAction method*), 750
reverse() (*CompileMultiPycAction method*), 741
reverse() (*CreatePrefixRecordAction method*), 743
reverse() (*CreatePythonEntryPointAction method*), 742
reverse() (*ExtractPackageAction method*), 751
reverse() (*LinkPathAction method*), 738
reverse() (*MakeMenuAction method*), 740
reverse() (*RegisterEnvironmentLocationAction method*), 745
reverse() (*RemoveLinkedPackageRecordAction method*), 748
reverse() (*RemoveMenuAction method*), 747
reverse() (*UnlinkPathAction method*), 746
reverse() (*UnregisterEnvironmentLocationAction method*), 749
reverse() (*UpdateHistoryAction method*), 744
revert_actions() (*in module conda.cli.install*), 647
revised_precs (*ChangeReport attribute*), 727
riscv64 (*Arch attribute*), 851
RM_EXTRACTED (*in module conda.instructions*), 839
RM_FETCHED (*in module conda.instructions*), 839
rm_items() (*in module conda.cli.main_clean*), 649
rm_pkgs() (*in module conda.cli.main_clean*), 649
rm_rf() (*in module conda.exports*), 799
rm_rf() (*in module conda.gateways.disk.delete*), 812
rm_rf() (*in module conda.plugins.prefix_data_loaders.pypi*), 933
rmtree() (*in module conda.gateways.disk.delete*), 811
rollback_enabled (*Context attribute*), 622
root_dir (*in module conda.exports*), 799
ROOT_ENV_NAME (*in module conda.base.constants*), 613
root_log (*in module conda.auxlib.logz*), 608
ROOT_NO_RM (*in module conda.base.constants*), 614
root_prefix() (*Context method*), 626
root_writable (*in module conda.exports*), 799
root_writable() (*Context method*), 626
rsplit() (*Dist method*), 850
RUN (*Commands attribute*), 989
run() (*_SatSolver method*), 665
run_as_admin() (*in module conda.common._os.windows*), 670
run_for (*CondaPostCommand attribute*), 537, 957
run_for (*CondaPreCommand attribute*), 538, 957
run_plan() (*in module conda.core.initialize*), 722
run_plan_elevated() (*in module conda.core.initialize*), 722
run_plan_from_stdin() (*in module conda.core.initialize*), 723
run_plan_from_temp_file() (*in module conda.core.initialize*), 723
run_script() (*in module conda.core.link*), 728
run_script_tmpl (*_Activator attribute*), 579
run_script_tmpl (*CmdExeActivator attribute*), 583
run_script_tmpl (*CshActivator attribute*), 581
run_script_tmpl (*FishActivator attribute*), 583
run_script_tmpl (*PosixActivator attribute*), 581
run_script_tmpl (*PowerShellActivator attribute*), 584
run_script_tmpl (*XonshActivator attribute*), 582
run_test_server() (*in module conda.testing.http_test_server*), 987

S

s390x (*Arch attribute*), 851
S3Adapter (*class in conda.gateways.connection.adapters.s3*), 804
safety_checks (*Context attribute*), 622
SafetyChecks (*class in conda.base.constants*), 614
SafetyError, 793
sat() (*Clauses method*), 667, 694
sat_solver (*Context attribute*), 624
sat_solver (*EnvironmentConfig attribute*), 857
SatSolverChoice (*class in conda.base.constants*), 618
save() (*EnvironmentYaml method*), 776
save() (*RepodataCache method*), 828
save_state() (*_ClauseArray method*), 664
save_state() (*_ClauseList method*), 664
save_state() (*_SatSolver method*), 665
SCHEMA_VALIDATORS (*in module conda.env.env*), 775
scheme (*_SplitUrlParts attribute*), 710
scheme (*Channel attribute*), 824, 844
scheme (*MultiChannel attribute*), 846
script_extension (*_Activator attribute*), 578
script_extension (*CmdExeActivator attribute*), 582
script_extension (*CshActivator attribute*), 581
script_extension (*FishActivator attribute*), 583
script_extension (*PosixActivator attribute*), 580
script_extension (*PowerShellActivator attribute*), 584
script_extension (*XonshActivator attribute*), 582
SEARCH (*Commands attribute*), 988
SEARCH_PATH (*in module conda.base.constants*), 613
selector (*_ChannelCutoff attribute*), 714
send() (*EnforceUnusedAdapter method*), 807
send() (*FTPAdapter method*), 801
send() (*LocalFSAdapter method*), 804
send() (*S3Adapter method*), 804
sep (*_Activator attribute*), 578
sep (*CmdExeActivator attribute*), 582
sep (*CshActivator attribute*), 581
sep (*FishActivator attribute*), 583
sep (*PosixActivator attribute*), 580
sep (*PowerShellActivator attribute*), 583
sep (*XonshActivator attribute*), 581
separate_format_cache (*Context attribute*), 622
sequence_parameters (*ParameterTypeGroups property*), 637
SequenceLoadedParameter (*class in conda.common.configuration*), 678
SequenceParameter (*class in conda.common.configuration*), 681
server (*HttpTestServerFixture attribute*), 982
session (*Shards attribute*), 564, 576
session_conda_cli() (*in module conda.testing.fixtures*), 978
session_tmp_env() (*in module conda.testing.fixtures*), 981
set_all_logger_level() (*in module conda.gateways.logging*), 818
set_binstar_token() (*in module conda.gateways.anaconda_client*), 800
set_conda_log_level() (*in module conda.gateways.logging*), 818
set_creation_time() (*PrefixData method*), 759, 948
set_environment_env_vars() (*PrefixData method*), 759, 948
set_file_logging() (*in module conda.gateways.logging*), 818
set_has_format() (*RepodataState method*), 828
set_key() (*ConfigurationFile method*), 640
set_keys() (*in module conda.cli.main_config*), 651
set_log_level() (*in module conda.gateways.logging*), 818
set_name() (*Field method*), 601
set_nonadmin() (*PrefixData method*), 759, 948
set_repository_metadata() (*SolverStateContainer method*), 764
set_root_level() (*in module conda.auxlib.logz*), 608
set_value() (*ContextStackObject method*), 628
set_var_tmpl (*_Activator attribute*), 579
set_var_tmpl (*CmdExeActivator attribute*), 583
set_var_tmpl (*CshActivator attribute*), 581
set_var_tmpl (*FishActivator attribute*), 583
set_var_tmpl (*PosixActivator attribute*), 581
set_var_tmpl (*PowerShellActivator attribute*), 584
set_var_tmpl (*XonshActivator attribute*), 582
setter() (*classproperty method*), 595
SETTING_KEYS (*_ChannelCutoff attribute*), 714
Settings, **84**
setup() (*_PycoSatSolver method*), 665
setup() (*_PyCryptoSatSolver method*), 665
setup() (*_PySatSolver method*), 666
setup() (*_SatSolver method*), 665
sha256 (*CacheUrlAction attribute*), 749
sha256 (*ExtractPackageAction attribute*), 751
sha256 (*PackageRecord attribute*), 879
sha256 (*PackageRecordDict attribute*), 572
sha256 (*PathDataV1 attribute*), 877
sha256_in_prefix (*PathDataV1 attribute*), 877
shard_cache (*ShardFetch attribute*), 561
SHARD_CACHE_NAME (*in module conda._private.shards.cache*), 557
shard_loaded() (*ShardBase method*), 563, 574
shard_loaded() (*ShardLike method*), 564, 575
shard_loaded() (*Shards method*), 565, 576
shard_mentioned_package_extras() (*in module conda._private.shards.shards*), 562
shard_mentioned_packages() (*in module conda._private.shards.shards*), 562
shard_url (*Node attribute*), 567
shard_url (*NodeId attribute*), 567

shard_url() (*ShardBase method*), 563, 574
shard_url() (*ShardLike method*), 564, 575
shard_url() (*Shards method*), 565, 576
ShardBase (*class in conda._private.shards*), 574
ShardBase (*class in conda._private.shards.shards*), 562
shardbase (*ShardFetch attribute*), 561
ShardCache (*class in conda._private.shards.cache*), 557
ShardDict (*class in conda._private.shards.typing*), 572
Sharded repodata, **484**
ShardFetch (*class in conda._private.shards.shards*), 561
ShardLike (*class in conda._private.shards*), 575
ShardLike (*class in conda._private.shards.shards*), 563
shardlikes (*RepodataSubset attribute*), 568, 832
Shards (*class in conda._private.shards*), 576
Shards (*class in conda._private.shards.shards*), 564
Shards (*class in conda.gateways.shards.typing*), 830
shards (*ShardLike attribute*), 564, 575
shards (*ShardsIndexDict attribute*), 574
shards_base_url (*RepodataInfoDict attribute*), 573
shards_base_url (*Shards property*), 565, 576
SHARDS_CONNECTIONS_DEFAULT (*in module conda._private.shards.misc*), 559
shards_index (*Shards attribute*), 564, 576
ShardsIndexDict (*class in conda._private.shards.typing*), 573
SHARE (*ERROR attribute*), 669
SharedLinkPathClobberError, 786
shebang_pat (*in module conda.cli.main_package*), 660
SHEBANG_REGEX (*in module conda.core.portability*), 752
SHEBANG_REGEX (*in module conda.gateways.disk.update*), 815
shlex_split_unicode() (*in module conda.auxlib.compat*), 592
shlvl (*Context property*), 626
shortcuts (*Context attribute*), 624
shortcuts_only (*Context attribute*), 624
should_check_format() (*RepodataState method*), 828
should_include() (*ExcludeNewerPolicy method*), 714
should_retry_solve (*SolverStateContainer attribute*), 764
should_use_color() (*in module conda.common.terminal*), 705
SHOW (*SW attribute*), 669
show_channel_urls (*Context attribute*), 623
show_spin (*Spinner attribute*), 936
SHOWDEFAULT (*SW attribute*), 669
SHOWMAXIMIZED (*SW attribute*), 669
SHOWMINIMIZED (*SW attribute*), 669
SHOWMINNOACTIVE (*SW attribute*), 669
SHOWNA (*SW attribute*), 669
SHOWNOACTIVATE (*SW attribute*), 669
SHOWNORMAL (*SW attribute*), 669
signal_handler() (*in module conda.common.signals*), 704
signing_metadata_url_base (*Context property*), 626
SimpleEnvironment (*class in conda.testing.solver_helpers*), 991
SINGLE_USE_KEYS (*PythonDistributionMetadata attribute*), 927
SingleStrArgCachingType (*class in conda.models.version*), 884
size (*CacheUrlAction attribute*), 750
size (*ExtractPackageAction attribute*), 751
size (*PackageRecord attribute*), 880
size() (*PrefixData method*), 758, 947
size_in_bytes (*PathDataV1 attribute*), 877
SKIP_FIELDS (*in module conda.cli.main_info*), 656
skip_tests() (*SolverTests method*), 992
softlink (*LinkType attribute*), 854
softlink (*PathEnum attribute*), 854
softlink_paths (*PreferredEnv attribute*), 870
softlink_supported() (*in module conda.gateways.disk.test*), 815
solution_precs (*SolverStateContainer attribute*), 764
solve() (*Resolve method*), 975
solve_final_state() (*BaseSolver method*), 762
solve_final_state() (*Solver method*), 585, 763
solve_for_diff() (*BaseSolver method*), 761
solve_for_diff() (*Solver method*), 586
solve_for_transaction() (*BaseSolver method*), 761
solve_for_transaction() (*Solver method*), 586
SolvedRecord (*class in conda.models.records*), 882
Solver (*class in conda.api*), 585
Solver (*class in conda.core.solve*), 762
solver (*Context attribute*), 624
solver (*EnvironmentConfig attribute*), 857
Solver (*in module conda.testing.fixtures*), 978
Solver state, **484**
solver() (*SimpleEnvironment method*), 991
solver_class (*SolverTests property*), 992
solver_class() (*in module conda.testing.helpers*), 987
solver_classic() (*in module conda.testing.fixtures*), 978
solver_ignore_timestamps (*Context attribute*), 624
solver_libmamba() (*in module conda.testing.fixtures*), 978
solver_rattler() (*in module conda.testing.fixtures*), 978
solver_transaction() (*SimpleEnvironment method*), 991
solver_user_agent() (*Context method*), 627
Solvers, **116**, **484**
SolverStateContainer (*class in conda.core.solve*), 764
SolverTests (*class in conda.testing.solver_helpers*), 992
sort_key (*BuildString property*), 842
source (*ArgParseRawParameter attribute*), 675

source (*EnvRawParameter attribute*), 675
source (*Link attribute*), 875
source (*MultipleKeysError attribute*), 673
source (*RawParameter attribute*), 674
source (*ValidationError attribute*), 673
source_full_path (*CreateInPrefixPathAction property*), 737
source_full_path (*ExtractPackageAction attribute*), 751
source_full_paths (*CompileMultiPycAction property*), 741
source_path_data (*LinkPathAction attribute*), 738
source_prefix (*CreateInPrefixPathAction attribute*), 737
source_short_path (*CreateInPrefixPathAction attribute*), 737
source_short_paths (*CompileMultiPycAction attribute*), 740
sp_reference (*PythonEggInfoDistribution attribute*), 927
sp_reference (*PythonInstalledDistribution attribute*), 926
spec (*BaseSpec property*), 887
spec (*MatchSpec property*), 864
spec (*PackageRecord property*), 880
spec_from_line() (*in module conda.cli.common*), 631
spec_matches (*PrefixGraph attribute*), 871, 933
spec_no_build (*PackageRecord property*), 880
spec_pat (*History attribute*), 837
spec_pat (*in module conda.cli.common*), 631
spec_str (*BaseSpec attribute*), 887
spec_to_package_extras() (*in module conda._private.shards.misc*), 560
spec_to_package_name() (*in module conda._private.shards.misc*), 560
SpecNotFound, 783, 796
SpecNotFoundInPackageCache, 797
specs (*ReducedIndex attribute*), 719
specs_by_name (*GeneralGraph attribute*), 872
specs_from_args() (*in module conda.cli.common*), 631
specs_from_history_map() (*SolverStateContainer method*), 764
specs_from_url() (*in module conda.cli.common*), 631
specs_map (*SolverStateContainer attribute*), 764
SPECS_SATISFIED_SKIP_SOLVE (*UpdateModifier attribute*), 617
specs_to_add (*BaseSolver attribute*), 761
specs_to_add (*ChangeReport attribute*), 727
specs_to_add_names (*BaseSolver attribute*), 761
specs_to_remove (*BaseSolver attribute*), 761
specs_to_remove (*ChangeReport attribute*), 727
SpecsConfigurationConflictError, 792
Spinner (*class in conda.plugins.reporter_backends.console*), 936
spinner() (*ConsoleReporterRenderer method*), 937
spinner() (*JSONReporterRenderer method*), 938
spinner() (*ReporterRendererBase method*), 960
spinner_cycle (*Spinner attribute*), 936
SpinnerBase (*class in conda.plugins.types*), 960
split() (*Dist method*), 850
split_anaconda_token() (*in module conda.common.url*), 709
split_conda_url_easy_parts() (*in module conda.common.url*), 710
split_extension() (*in module conda.models.dist*), 848
split_platform() (*in module conda.common.url*), 709
split_scheme_auth_token() (*in module conda.common.url*), 710
split_spec() (*in module conda.plugins.prefix_data_loaders.pypi.pkg_format*), 930
SplitStrMatch (*class in conda.models.match_spec*), 867
src (*LinkSourceNotFoundError attribute*), 788
ssl_verify (*Context attribute*), 622
ssl_verify_validation() (*in module conda.base.context*), 621
stack_context() (*in module conda.base.context*), 628
stack_context_default() (*in module conda.base.context*), 628
stale() (*RepodataCache method*), 829
start() (*Spinner method*), 936
start_time (*time_recorder attribute*), 691
startswith() (*Dist method*), 850
startswith() (*VersionOrder method*), 886
state (*RepodataCache attribute*), 828
status_code (*MockResponse attribute*), 990
status_code (*UnavailableInvalidChannel attribute*), 790, 824
stderr (*Response attribute*), 836
stderr() (*in module conda.gateways.streams*), 835
stderr_log_level() (*in module conda.common.io*), 690
stderrlog() (*in module conda.gateways.streams*), 835
STDOUT (*CaptureTarget attribute*), 689
stdout (*Response attribute*), 836
stdout() (*in module conda.gateways.streams*), 835
stdout_json() (*in module conda.cli.common*), 631
stdout_json_success() (*in module conda.cli.common*), 631
stdoutlog() (*in module conda.gateways.streams*), 835
StdStreamHandler (*class in conda.gateways.logging*), 817
stop() (*Spinner method*), 936
STRICT (*ChannelPriority attribute*), 618

strictness (*MatchSpec property*), 864
STRING (*CaptureTarget attribute*), 689
STRING_CHUNK (*in module conda.plugins.prefix_data_loaders.pypi.pkg_format*), 931
string_delimiter (*SequenceParameter attribute*), 681
StringField (*class in conda.auxlib.entity*), 603
stringify() (*in module conda.auxlib.logz*), 609
stringify() (*in module conda.gateways.repodata*), 821
strip_comment() (*in module conda.cli.common*), 631
strip_expected() (*in module conda.testing.helpers*), 985
strip_extension() (*in module conda.models.dist*), 848
strip_scheme() (*in module conda.common.url*), 709
subdir (*Channel property*), 824, 844
subdir (*CondaExceptionEvent attribute*), 530, 966
subdir (*Context property*), 625
subdir (*Dist property*), 849
subdir (*in module conda.exports*), 799
subdir (*PackageRecord attribute*), 878
subdir (*RepodataInfoDict attribute*), 573
subdir_url (*Channel property*), 825, 845
SubdirData (*class in conda.api*), 587
SubdirData (*class in conda.core.subdir_data*), 766
SubdirDataType (*class in conda.core.subdir_data*), 765
SubdirField (*class in conda.models.records*), 876
subdirs (*BaseSolver attribute*), 761
subdirs (*Context property*), 626
subdirs (*SimpleEnvironment attribute*), 991
subprocess_call() (*in module conda.gateways.subprocess*), 836
subprocess_call_with_clean_env() (*in module conda.gateways.subprocess*), 836
succeed() (*Repodatas method*), 646
success (*Repodatas attribute*), 646
suffix (*BuildString property*), 842
summary (*CondaHealthCheck attribute*), 535, 958
summary (*CondaSubcommand attribute*), 549, 949, 953, 955
summary (*ErrorGuidance attribute*), 555
SUMMARY (*in module conda.plugins.subcommands.plugins*), 953
superseded_precs (*ChangeReport attribute*), 727
supplement_index_with_repodata() (*in module conda.testing.helpers*), 985
supports_exclude_newer_channel (*BaseSolver attribute*), 760
supports_exclude_newer_channel (*Solver attribute*), 763
supports_exclude_newer_global (*BaseSolver attribute*), 760
supports_exclude_newer_global (*Solver attribute*), 763
supports_exclude_newer_package (*BaseSolver attribute*), 760
supports_exclude_newer_package (*Solver attribute*), 763
suppress_pytest_bencher_deprecation_warning() (*in module conda.testing.fixtures*), 977
suppress_resource_warning() (*in module conda.testing.fixtures*), 977
SW (*class in conda.common._os.windows*), 668
swallow_broken_pipe (*in module conda.common.io*), 688
SwallowBrokenPipe (*class in conda.common.io*), 688
SYMLINK_CONDA (*in module conda.instructions*), 839
sys_prefix_unfollowed() (*in module conda.utils*), 994
sys_rc_path (*in module conda.base.context*), 621
sys_stream (*StdStreamHandler attribute*), 817
system_packages (*Index property*), 717

T

T (*in module conda.common.iterators*), 692
T (*in module conda.common.toposort*), 706
T (*in module conda.core.prefix_data*), 755
T (*in module conda.deprecations*), 770
T (*in module conda.exception_handler*), 783
T (*in module conda.models.environment*), 857
tarball_basename (*PackageCacheRecord property*), 882
tarball_file_in_cache() (*PackageCacheData class method*), 730, 822
tarball_file_in_this_cache() (*PackageCacheData method*), 730, 822
target (*MatchSpec property*), 863
target_extracted_dirname (*ExtractPackageAction attribute*), 751
target_full_path (*CacheUrlAction property*), 750
target_full_path (*ExtractPackageAction property*), 751
target_full_path (*PathAction property*), 735
target_full_path (*PrefixPathAction property*), 736
target_full_path (*RegisterEnvironmentLocationAction property*), 745
target_full_path (*UnregisterEnvironmentLocationAction property*), 749
target_full_paths (*CompileMultiPycAction property*), 741
target_full_paths (*MultiPathAction property*), 735
target_package_basename (*CacheUrlAction attribute*), 749
target_pkgs_dir (*CacheUrlAction attribute*), 749
target_pkgs_dir (*ExtractPackageAction attribute*), 751
target_prefix (*Action attribute*), 733
target_prefix (*ActionGroup attribute*), 726

target_prefix (*CompileMultiPycAction attribute*), 740
target_prefix (*CondaExceptionEvent attribute*), 531, 966
target_prefix (*Context property*), 626
target_prefix (*PrefixPathAction attribute*), 736
target_prefix (*PrefixSetup attribute*), 726
target_prefix (*RegisterEnvironmentLocationAction attribute*), 744
target_prefix (*UnregisterEnvironmentLocationAction attribute*), 748
target_prefix_override (*Context attribute*), 624
target_short_path (*PrefixPathAction attribute*), 736
target_short_paths (*CompileMultiPycAction attribute*), 740
target_short_paths (*PrefixPathAction property*), 736
temp_package_cache() (*in module conda.testing.fixtures*), 977
temp_simple_env() (*in module conda.testing.solver_helpers*), 992
tempdir() (*in module conda.testing.helpers*), 985
tempfile_extension (*_Activator attribute*), 578
tempfile_extension (*CmdExeActivator attribute*), 582
tempfile_extension (*CshActivator attribute*), 581
tempfile_extension (*FishActivator attribute*), 583
tempfile_extension (*JSONFormatMixin attribute*), 584
tempfile_extension (*PosixActivator attribute*), 580
tempfile_extension (*PowerShellActivator attribute*), 584
tempfile_extension (*XonshActivator attribute*), 582
template_export_var() (*_Activator method*), 579
template_path_var() (*_Activator method*), 579
template_path_var() (*XonshActivator method*), 582
template_run_script() (*_Activator method*), 579
template_set_var() (*_Activator method*), 579
template_unset_var() (*_Activator method*), 579
TemporaryDirectory (*class in conda.gateways.disk.create*), 810
term_dumb() (*in module conda.common.terminal*), 705
terminator (*StdStreamHandler attribute*), 817
test_accelerate() (*SolverTests method*), 992
test_anaconda_nomkl() (*SolverTests method*), 992
test_arch_preferred_over_noarch_when_otherwise_equal() (*SolverTests method*), 993
test_channel_priority_1() (*SolverTests method*), 993
test_circular_dependencies() (*SolverTests method*), 993
TEST_CONDARC (*in module conda.testing.fixtures*), 977
TEST_DATA_DIR (*in module conda.testing.helpers*), 985
test_empty() (*SolverTests method*), 992
test_get_dists() (*SolverTests method*), 993
test_get_reduced_index_broadening_preferred_solution() (*SolverTests method*), 993
test_get_reduced_index_broadening_with_unsatisfiable_early (*SolverTests method*), 993
test_install_package_with_feature() (*SolverTests method*), 993
test_iopro_mkl() (*SolverTests method*), 992
test_iopro_nomkl() (*SolverTests method*), 992
test_irrational_version() (*SolverTests method*), 993
TEST_LOG_LEVEL (*in module conda.testing.integration*), 988
test_mkl() (*SolverTests method*), 992
test_no_features() (*SolverTests method*), 993
test_noarch_preferred_over_arch_when_build_greater() (*SolverTests method*), 993
test_noarch_preferred_over_arch_when_build_greater_dep() (*SolverTests method*), 993
test_noarch_preferred_over_arch_when_version_greater() (*SolverTests method*), 993
test_noarch_preferred_over_arch_when_version_greater_dep() (*SolverTests method*), 993
test_nonexistent() (*SolverTests method*), 993
test_nonexistent_deps() (*SolverTests method*), 993
test_pseudo_boolean() (*SolverTests method*), 992
test_remove() (*SolverTests method*), 993
test_scipy_mkl() (*SolverTests method*), 992
test_surplus_features_1() (*SolverTests method*), 993
test_surplus_features_2() (*SolverTests method*), 993
test_timestamps_and_deps() (*SolverTests method*), 993
test_unintentional_feature_downgrade() (*SolverTests method*), 993
test_unsat_any_two_not_three() (*SolverTests method*), 993
test_unsat_chain() (*SolverTests method*), 993
test_unsat_channel_priority() (*SolverTests method*), 993
test_unsat_expand_single() (*SolverTests method*), 993
test_unsat_from_r1() (*SolverTests method*), 992
test_unsat_missing_dep() (*SolverTests method*), 993
test_unsat_shortest_chain_1() (*SolverTests method*), 993
test_unsat_shortest_chain_2() (*SolverTests method*), 993
test_unsat_shortest_chain_3() (*SolverTests method*), 993
test_unsat_shortest_chain_4() (*SolverTests method*), 993
test_unsat_simple() (*SolverTests method*), 992
tests_to_skip (*SolverTests property*), 992
text (*CondaErrorHint attribute*), 512, 961
text (*FileMode attribute*), 853

text (*GuidanceHint attribute*), 555
text (*GuidanceHintTypedDict attribute*), 555
ThisShouldNeverHappenError, 607
THREAD_WAIT_TIMEOUT (*in module conda._private.shards.subset*), 567
THREADSAFE_EXTRACT (*in module conda.core.package_cache_data*), 729
time_recorder (*class in conda.common.io*), 691
timeme() (*in module conda.gateways.repodata.zstd*), 818
timeout() (*in module conda.common.io*), 690
timeout() (*RepodataCache method*), 829
timestamp (*PackageRecord attribute*), 880
TimestampField (*class in conda.models.records*), 874
tmp_channel() (*in module conda.testing.fixtures*), 981
tmp_env() (*in module conda.testing.fixtures*), 980
tmp_envs_dir() (*in module conda.testing.fixtures*), 982
tmp_path (*PathFactoryFixture attribute*), 979
tmp_pkgs_dir() (*in module conda.testing.fixtures*), 982
tmp_repodata_channel() (*in module conda.testing.fixtures*), 981
TMP_REPODATA_CHANNEL_DEFAULTS (*in module conda.testing.fixtures*), 981
TmpChannelFixture (*class in conda.testing.fixtures*), 981
tmpdir() (*in module conda.testing.fixtures*), 977
TmpDownload (*class in conda.gateways.connection.download*), 806
TmpEnvFixture (*class in conda.testing.fixtures*), 980
TmpRepodataChannelFixture (*class in conda.testing.fixtures*), 981
to_dict() (*ChannelNotice method*), 893
to_dict() (*EnvironmentYaml method*), 776
to_dict() (*in module conda.plugins.environment_exporters.environment_yaml*), 897
to_environment_model() (*EnvironmentYaml method*), 776
to_feature_metric_id() (*Resolve static method*), 974
to_filename() (*Dist method*), 849
to_id() (*Node method*), 567
to_json() (*ConfigurationObject method*), 679
to_match_spec() (*Dist method*), 849
to_match_spec() (*PackageRecord method*), 880
to_matchspec() (*Dist method*), 849
to_package_ref() (*Dist method*), 849
to_sat_name() (*Resolve static method*), 974
to_simple_match_spec() (*PackageRecord method*), 880
to_url() (*Dist method*), 849
to_virtual_package() (*CondaVirtualPackage method*), 550, 956
to_yaml() (*EnvironmentYaml method*), 776
token (*_SplitUrlParts attribute*), 710
token (*Channel attribute*), 824, 844
token (*MultiChannel attribute*), 846
TOKEN_REPLACE (*TokenURLFilter attribute*), 817
TOKEN_URL_PATTERN (*TokenURLFilter attribute*), 817
tokenized_conda_url_startswith() (*in module conda.models.channel*), 847
tokenized_startswith() (*in module conda.common.path*), 699
tokenized_startswith() (*in module conda.common.path.directories*), 696
tokenized_startswith() (*in module conda.models.channel*), 847
TokenURLFilter (*class in conda.gateways.logging*), 817
TooManyArgumentsError, 785
top (*ParameterFlag attribute*), 674
topic() (*DeprecationHandler method*), 772
toposort() (*in module conda.common.toposort*), 706
total_call_num (*time_recorder attribute*), 691
total_number_channel_notices (*ChannelNoticeResultSet attribute*), 893
total_run_time (*time_recorder attribute*), 691
touch() (*in module conda.gateways.disk.update*), 816
TQDMProgressBar (*class in conda.plugins.reporter_backends.console*), 936
trace (*Context property*), 626
TRACE (*in module conda.common.constants*), 685
TRACE (*in module conda.gateways.disk*), 816
track_features (*Context attribute*), 622
track_features (*EnvironmentConfig attribute*), 857
track_features (*PackageRecord attribute*), 880
track_features_specs() (*SolverStateContainer method*), 764
trackers (*Resolve attribute*), 972
transaction_context (*Action attribute*), 733
transaction_context (*CompileMultiPycAction attribute*), 740
transaction_context (*PrefixPathAction attribute*), 736
transaction_context (*RegisterEnvironmentLocationAction attribute*), 744
transaction_context (*UnregisterEnvironmentLocationAction attribute*), 748
trash_dir() (*Context method*), 626
treeify() (*in module conda.models.version*), 886
Troubleshooting, **137**
TRUE (*in module conda.common._logic*), 664
TRUE (*in module conda.common.logic*), 693
TryRepodata (*class in conda.cli.install*), 646
type (*ActionGroup attribute*), 726
type (*Field property*), 601
type (*Link attribute*), 875
type (*Noarch attribute*), 869
type (*ParameterLoader attribute*), 683

Type hinting, 417
typify() (*in module conda.auxlib.type_coercion*), 610
typify() (*LoadedParameter method*), 677
typify() (*Parameter method*), 680
typify_parameter() (*Configuration method*), 685

U

UnavailableInvalidChannel, 790, 823
unbox() (*Field method*), 601
unbox() (*ListField method*), 604
UNDEFINED_MESSAGE_ID (*in module conda.notices.types*), 893
union() (*BuildNumberMatch method*), 888
union() (*MatchInterface method*), 867
union() (*MatchSpec class method*), 865
union() (*VersionSpec method*), 888
unique() (*in module conda.common.iterators*), 692
unique_sequence_map() (*in module conda.common.configuration*), 685
unix_path_to_win() (*in module conda.common.path*), 699
unix_path_to_win() (*in module conda.common.path.windows*), 698
unix_python_entry_point (*PathEnum attribute*), 854
UNKNOWN_CHANNEL (*in module conda.base.constants*), 614
UnknownPackageClobberError, 786
UNLINK (*in module conda.instructions*), 839
unlink_action_groups (*PrefixActionGroup attribute*), 726
unlink_or_rename_to_trash() (*in module conda.gateways.disk.delete*), 811
unlink_precs (*Action attribute*), 734
unlink_precs (*PrefixSetup attribute*), 726
UnlinkLinkTransaction (*class in conda.core.link*), 727
UNLINKLINKTRANSACTION (*in module conda.instructions*), 839
UNLINKLINKTRANSACTION_CMD() (*in module conda.instructions*), 840
UnlinkPathAction (*class in conda.core.path_actions*), 746
unmanageable_package_types() (*PackageType static method*), 855
unmerged_specs_to_add (*BaseSolver attribute*), 761
unmerged_specs_to_remove (*BaseSolver attribute*), 761
unregister_action_groups (*PrefixActionGroup attribute*), 726
unregister_env() (*in module conda.core.envs_manager*), 712
UnregisterEnvironmentLocationAction (*class in conda.core.path_actions*), 748
unsat (*Clauses attribute*), 666
unsat (*Clauses property*), 693
Unsatisfiable (*in module conda.resolve*), 971
unsatisfiable (*UnsatisfiableError attribute*), 792
unsatisfiable_hints (*Context attribute*), 624
unsatisfiable_hints_check_depth (*Context attribute*), 624
UnsatisfiableConflictEntry (*in module conda.exceptions*), 784
UnsatisfiableError, 792
unset_environment_env_vars() (*PrefixData method*), 759, 948
unset_var_tmpl (*_Activator attribute*), 579
unset_var_tmpl (*CmdExeActivator attribute*), 582
unset_var_tmpl (*CshActivator attribute*), 581
unset_var_tmpl (*FishActivator attribute*), 583
unset_var_tmpl (*PosixActivator attribute*), 580
unset_var_tmpl (*PowerShellActivator attribute*), 584
unset_var_tmpl (*XonshActivator attribute*), 582
untracked() (*in module conda.misc*), 841
untreeify() (*in module conda.models.version*), 887
UNUSED_ENV_NAME (*in module conda.base.constants*), 613
UPDATE (*Commands attribute*), 989
update() (*History method*), 837
UPDATE_ALL (*UpdateModifier attribute*), 617
UPDATE_DEPS (*UpdateModifier attribute*), 617
update_file_in_place_as_binary() (*in module conda.gateways.disk.update*), 815
update_modifier (*Context attribute*), 624
update_modifier (*EnvironmentConfig attribute*), 857
update_modifier (*SolverStateContainer attribute*), 764
update_prefix() (*in module conda.core.portability*), 752
update_specs (*Action attribute*), 734
update_specs (*PrefixSetup attribute*), 726
update_specs (*UpdateHistoryAction attribute*), 743
UPDATE_SPECS (*UpdateModifier attribute*), 617
update_to() (*JSONProgressBar method*), 938
update_to() (*ProgressBarBase method*), 960
update_to() (*QuietProgressBar method*), 936
update_to() (*TQDMProgressBar method*), 936
updated_precs (*ChangeReport attribute*), 727
UpdateHistoryAction (*class in conda.core.path_actions*), 743
UpdateModifier (*class in conda.base.constants*), 616
UpdateModifier (*in module conda.api*), 585
url (*AnnotatedRawShard attribute*), 557
url (*CacheUrlAction attribute*), 749
url (*ChannelNoticeResponse attribute*), 893
Url (*class in conda.common.url*), 707
url (*HttpTestServerFixture attribute*), 982
url (*PackageInfo attribute*), 870
url (*PackageRecord attribute*), 878
url (*PySpec attribute*), 925

url (*ShardBase attribute*), 562, 574
url (*ShardFetch attribute*), 561
url (*ShardLike attribute*), 564, 575
url (*Shards attribute*), 564, 576
url (*TmpDownload attribute*), 806
url() (*Channel method*), 825, 845
url() (*MultiChannel method*), 847
url_attrs (*in module conda.common.url*), 707
url_channel_wtf (*Channel property*), 826, 846
URL_GLOB_CHARS (*_ChannelSelector attribute*), 713
URL_KEY (*in module conda.gateways.repodata*), 826
url_match_parts() (*_ChannelSelector class method*), 713
url_pattern (*_ChannelSelector attribute*), 713
url_to_s3_info() (*in module conda.common.url*), 708
url_w_credentials (*RepodataFetch attribute*), 829
url_w_credentials (*SubdirData attribute*), 767
url_w_repodata_fn (*RepodataFetch property*), 829
url_w_repodata_fn (*SubdirData property*), 767
url_w_subdir (*RepodataFetch attribute*), 829
url_w_subdir (*SubdirData attribute*), 767
urlparse() (*in module conda.common.url*), 708
urls() (*Channel method*), 825, 845
urls() (*MultiChannel method*), 846
UrlsData (*class in conda.core.package_cache_data*), 731
use_cache (*Index attribute*), 716
use_index_cache (*Context attribute*), 622
use_local (*Context attribute*), 623
use_only_tar_bz2 (*Context property*), 626
use_only_tar_bz2 (*EnvironmentConfig attribute*), 857
use_system (*Index attribute*), 716
user_agent (*ExceptionHandler property*), 783
user_agent() (*Context method*), 627
user_data_dir() (*in module conda.base.context*), 621
user_rc_path (*in module conda.base.context*), 621
username (*EnvironmentFileNotDownloaded attribute*), 796
Using authenticated channels, **57**
Using conda for your project, **137**
Using Custom Locations for Environment and Package Cache, **84**
Using non-standard certificates, **84**
Using the .condarc conda configuration file, **84**
Utf8NamedTemporaryFile() (*in module conda.auxlib.compat*), 593

V

v3 (*ShardDict attribute*), 573
valid() (*Resolve method*), 972
valid2() (*Resolve method*), 972
VALID_KEYS (*in module conda.env.env*), 774
valid_types (*InvalidTypeError attribute*), 673
validate() (*Entity method*), 606
validate() (*Field method*), 601
validate() (*ListField method*), 604
validate_all() (*Configuration method*), 685
validate_channels() (*in module conda.base.context*), 629
validate_configuration() (*Configuration method*), 685
validate_environment_files_consistency() (*in module conda.cli.common*), 632
validate_file_exists() (*in module conda.cli.common*), 632
validate_install_command() (*in module conda.cli.install*), 646
validate_keys() (*in module conda.env.env*), 775
validate_name() (*PrefixData method*), 757, 946
validate_path() (*PrefixData method*), 757, 946
validate_provided_parameters() (*in module conda.cli.condarc*), 638
validate_subdir_config() (*in module conda.cli.common*), 631
ValidationError, 607, 673, 932
value (*CondaRequestHeader attribute*), 546, 961
value (*LoadedParameter attribute*), 676
value() (*ArgParseRawParameter method*), 675
value() (*DefaultValueRawParameter method*), 676
value() (*EnvRawParameter method*), 675
value() (*RawParameter method*), 675
value() (*YamlRawParameter method*), 675
value_flags (*LoadedParameter attribute*), 676
ValueEnum (*class in conda.base.constants*), 618
valueflags() (*ArgParseRawParameter method*), 675
valueflags() (*DefaultValueRawParameter method*), 676
valueflags() (*EnvRawParameter method*), 675
valueflags() (*RawParameter method*), 675
valueflags() (*YamlRawParameter method*), 675
values() (*_LazyParserMap method*), 635
values() (*PackageCacheData method*), 730, 823
variables (*Environment attribute*), 858
variables (*EnvironmentYaml attribute*), 776
variables_validation() (*in module conda.env.env*), 775
ver_eval() (*in module conda.models.version*), 884
verbose (*Context property*), 627
verbose (*TmpDownload attribute*), 806
verbosity (*Context property*), 627
verified (*Action property*), 734
verify() (*Action method*), 734
verify() (*CacheUrlAction method*), 750
verify() (*CompileMultiPycAction method*), 741
verify() (*CreateInPrefixPathAction method*), 737
verify() (*ExtractPackageAction method*), 751
verify() (*LinkPathAction method*), 738

verify() (*PrefixReplaceLinkAction method*), 739
verify() (*RegisterEnvironmentLocationAction method*), 744
verify() (*RemoveFromPrefixPathAction method*), 745
verify() (*UnlinkLinkTransaction method*), 728
verify() (*UnregisterEnvironmentLocationAction method*), 748
verify_executor (*UnlinkLinkTransaction attribute*), 727
verify_specs() (*Resolve method*), 972
verify_threads (*Context property*), 625
verify_windows_launcher() (*in module conda.core.launchers*), 725
version (*CondaVirtualPackage attribute*), 550, 956
version (*Dist attribute*), 849
version (*DistDetails attribute*), 848
version (*in module conda._version*), 577
version (*MatchSpec property*), 865
version (*PackageInfo property*), 870
version (*PackageRecord attribute*), 878
version (*PackageRecordDict attribute*), 572
version (*PythonDistribution property*), 926
version (*PythonDistributionMetadata property*), 930
version (*ShardsIndexDict attribute*), 573
version (*VersionOrder attribute*), 886
version_cache (*in module conda.models.version*), 884
version_check_re (*in module conda.models.version*), 884
VERSION_IDENTIFIER (*in module conda.plugins.prefix_data_loaders.pypi.pkg_format*), 931
version_key() (*Resolve method*), 974
version_relation_re (*in module conda.models.version*), 887
version_split_re (*in module conda.models.version*), 884
version_tuple (*in module conda._version*), 577
version_validation (*CondaVirtualPackage attribute*), 550, 956
VersionMatch (*in module conda.models.version*), 888
VersionOrder (*class in conda.models.version*), 884
VersionSpec (*class in conda.models.version*), 888
View command line help, **57**
viewed_channel_notices (*ChannelNoticeResultSet attribute*), 893
virtual_package() (*PackageRecord class method*), 881
virtual_packages (*Environment attribute*), 858
VIRTUAL_PRIVATE_ENV (*PackageType attribute*), 855
VIRTUAL_PYTHON_EGG_LINK (*PackageType attribute*), 855
VIRTUAL_PYTHON_EGG_MANAGEABLE (*PackageType attribute*), 855
VIRTUAL_PYTHON_EGG_UNMANAGEABLE (*PackageType attribute*), 855
VIRTUAL_PYTHON_WHEEL (*PackageType attribute*), 855
VIRTUAL_SYSTEM (*PackageType attribute*), 855
visit_package() (*ShardBase method*), 563, 575
visit_package() (*ShardLike method*), 564, 575
visit_package() (*Shards method*), 565, 576
visit_shard() (*ShardBase method*), 563, 575
visited (*Node attribute*), 567
visited (*ShardBase attribute*), 563, 574
visited (*ShardLike attribute*), 564, 575
visited (*Shards attribute*), 564, 576
VSPEC_TOKENS (*in module conda.models.version*), 886

W

walk_prefix() (*in module conda.misc*), 841
warn (*PathConflict attribute*), 616
warn (*SafetyChecks attribute*), 615
warning (*ChannelDenied attribute*), 789
warning (*ChannelNotAllowed attribute*), 789
WARNING (*NoticeLevel attribute*), 618
warning_handler (*ConfigurationFile attribute*), 638
wasi (*Platform attribute*), 852
wasm32 (*Arch attribute*), 851
watch_for (*CondaExceptionObserver attribute*), 531, 967
which_or_where (*in module conda.testing.integration*), 988
which_package() (*in module conda.cli.main_package*), 660
which_prefix() (*in module conda.cli.main_package*), 661
win (*Platform attribute*), 852
win_path_backout() (*in module conda.common.path*), 700
win_path_backout() (*in module conda.common.path.windows*), 698
win_path_double_escape() (*in module conda.common.path*), 700
win_path_double_escape() (*in module conda.common.path.windows*), 698
win_path_ok() (*in module conda.common.path*), 700
win_path_ok() (*in module conda.common.path.windows*), 697
win_path_ok() (*in module conda.plugins.prefix_data_loaders.pypi*), 932
win_path_to_unix() (*in module conda.common.path*), 700
win_path_to_unix() (*in module conda.common.path.windows*), 698
win_version() (*in module conda.plugins.virtual_packages.windows*), 970

WINDOWS_LAUNCHER_STUB_PATH (*in module conda.base.constants*), 613
WINDOWS_PROBLEMATIC_CHARS (*in module conda.base.constants*), 614
windows_python_entry_point_exe (*PathEnum attribute*), 854
windows_python_entry_point_script (*PathEnum attribute*), 854
with_hints() (*ErrorGuidance method*), 556
withext() (*in module conda.gateways.repodata.zstd*), 818
Working with environments, **137**
Working with packages, **137**
working_state_reset() (*SolverStateContainer method*), 764
wrap_subprocess_call() (*in module conda.utils*), 995
writable_caches() (*PackageCacheData class method*), 730, 822
write() (*ConfigurationFile method*), 640
write() (*in module conda.common.serialize.json*), 702
write() (*in module conda.common.serialize.yaml*), 703
write_as_json_to_file() (*in module conda.gateways.disk.create*), 811
write_changes() (*History method*), 838
write_head() (*in module conda.history*), 837
write_notice_response_to_cache() (*in module conda.notices.cache*), 890
write_out() (*ExceptionHandler method*), 783
write_specs() (*History method*), 838
Writing tests, **417**
wrong_type (*InvalidTypeError attribute*), 673

X

x86 (*Arch attribute*), 851
x86_64 (*Arch attribute*), 851
X_MARK (*in module conda.base.constants*), 619
XonshActivator (*class in conda.activate*), 581
Xor() (*Clauses method*), 667, 694

Y

YAML_EXTENSIONS (*in module conda.common.configuration*), 683
yaml_round_trip_dump() (*in module conda.common.serialize*), 704
yaml_round_trip_load() (*in module conda.common.serialize*), 703
yaml_safe_dump() (*in module conda.common.serialize*), 704
yaml_safe_load() (*in module conda.common.serialize*), 703
YamlFileSpec (*class in conda.env.specs.yaml_file*), 782
YamlRawParameter (*class in conda.common.configuration*), 675
yield_lines() (*in module conda.gateways.disk.read*), 814

Z

z (*Arch attribute*), 851
ZERO (*ERROR attribute*), 669
zos (*Platform attribute*), 852
ZSTD_MAX_SHARD_INDEX_SIZE (*in module conda._private.shards.decompression*), 558
ZSTD_MAX_SHARD_SIZE (*in module conda._private.shards.decompression*), 558
ZstdRepoInterface (*class in conda.gateways.repodata.zstd*), 819
ZstSkip, 819