Managing environments

- Creating an environment with commands
- Creating an environment from an environment.yml file
- Cloning an environment
- Building identical conda environments
- Activating an environment
- Deactivating an environment
- Determining your current environment
- Viewing a list of your environments
- Viewing a list of the packages in an environment
- Using pip in an environment
- Saving environment variables
- Sharing an environment
- Removing an environment

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.

NOTE: There are many options available for the commands described on this page. For details, see Command reference.

Creating an environment with commands

TIP: By default, environments are installed into the `envs` directory in your conda directory. Run `conda create --help` for information on specifying a different path.

1. To create an environment, run:

   ```
   conda create --name myenv
   ```

   NOTE: Replace `myenv` with the environment name.
2. When conda asks you to proceed, type `y`:

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

This creates the myenv environment in `/envs/`. This environment uses the same version of Python that you are currently using, because you did not specify a version.

To create an environment with a specific version of Python:

```
conda create -n myenv python=3.4
```

To create an environment with a specific package:

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

OR:

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

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

```
conda create -n myenv scipy=0.15.0
```

OR:

```
conda create -n myenv python
conda install -n myenv scipy=0.15.0
```

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

```
conda create -n myenv python=3.4 scipy=0.15.0 astroid babel
```

TIP: Install all the programs that you want in this environment at the same time. Installing 1 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

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

```
conda env create -f environment.yml
```

2. Activate the new environment:

   - Windows: `activate myenv`
   - macOS and Linux: `source activate myenv`

   NOTE: Replace `myenv` with the name of the environment.

3. Verify that the new environment was installed correctly:

```
conda list
```

Cloning an environment

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

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.

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.continuum.io/pkgs/free/osx-64/mkl-11.3.3-0.tar.bz2
https://repo.continuum.io/pkgs/free/osx-64/numpy-1.11.1-py35_0.tar.bz2
https://repo.continuum.io/pkgs/free/osx-64/openssl-1.0.2h-1.tar.bz2
https://repo.continuum.io/pkgs/free/osx-64/pip-8.1.2-py35_0.tar.bz2
https://repo.continuum.io/pkgs/free/osx-64/python-3.5.2-0.tar.bz2
https://repo.continuum.io/pkgs/free/osx-64/readline-6.2-2.tar.bz2
https://repo.continuum.io/pkgs/free/osx-64/setuptools-25.1.6-py35_0.tar.bz2
https://repo.continuum.io/pkgs/free/osx-64/sqlite-3.13.0-0.tar.bz2
https://repo.continuum.io/pkgs/free/osx-64/tk-8.5.18-0.tar.bz2
https://repo.continuum.io/pkgs/free/osx-64/wheel-0.29.0-py35_0.tar.bz2
https://repo.continuum.io/pkgs/free/osx-64/xz-5.2.2-0.tar.bz2
https://repo.continuum.io/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
```

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

To activate an environment:

- Windows: `activate myenv`
- macOS and Linux: `source activate myenv`

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

Deactivating an environment

To deactivate an environment:

- Windows: `deactivate`
- macOS and Linux: `source deactivate`

Conda removes the path name `myenv` from your system command.

TIP: In Windows, it is good practice to deactivate one environment before activating another.

Determining your current environment

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:

```
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
```

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:

  ```
  conda list -n myenv
  ```

- If the environment is activated:

```
conda list
```

To see if a specific package is installed in an environment:

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

Using pip in an environment

To use pip in your environment:

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

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, such as `C:\Users\jsmith\Anaconda3\envs\analytics`.
2. Enter that directory and create these subdirectories and files:

```
cd C:\Users\jsmith\Anaconda3\envs\analytics
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 `activate analytics`, the environment variables MY_KEY and MY_FILE are set to the values you wrote into the file. When you run `deactivate`, those variables are erased.

macOS and Linux

1. Locate the directory for the conda environment, such as `/home/jsmith/anaconda3/envs/analytics`.
2. Enter that directory and create these subdirectories and files:

```
cd /home/jsmith/anaconda3/envs/analytics
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 `source activate analytics`, the environment variables MY_KEY and MY_FILE are set to the values you wrote into the file. When you run `source 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.yml file`.

Exporting the environment file

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:
   - Windows: `activate myenv`
   - macOS and Linux: `source 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.

Creating an environment file manually

You can create an environment file 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=3.4   # or 2.7
  - bokeh=0.9.2
  - numpy=1.9.*
  - nodejs=0.10.*
  - flask
  - pip:
    - Flask-Testing
```

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.

Removing an environment

To remove an environment:

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

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

To verify that the environment was removed:

```
conda info --envs
```

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