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.

Note

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

Note

`conda activate` and `conda deactivate` only work on conda 4.6 and later versions. For conda versions prior to 4.6, run:

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

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.

Use the terminal or an Anaconda Prompt for the following steps:

1. To create an environment:

   ```
   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.

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

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

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=0.15.0
```

OR:

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

6. 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

Use the terminal or an Anaconda Prompt for the following steps:

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

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

   The first line of the `yml` file sets the new environment's name. For details see Creating an environment file manually.
2. Activate the new environment: `conda activate myenv`

Note

Replace `myenv` with the environment name.

3. Verify that the new environment was installed correctly:

```
conda list
```

Cloning an environment

Use the terminal or an Anaconda Prompt 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

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 or an Anaconda Prompt 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.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

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.

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.

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 standard library search order, defined at https://docs.microsoft.com/en-us/previous-versions/7d83bc18(v=vs.140). If environments are not active, libraries won't get 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.

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 base 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.

Determining your current environment

Use the terminal or an Anaconda Prompt 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 or an Anaconda Prompt, 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
```

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 or an Anaconda Prompt, run:

  ```
  conda list -n myenv
  ```

- If the environment is activated, in your terminal window or an Anaconda Prompt, run:

  ```
  conda list
  ```

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

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

Using pip in an environment

To use pip in your environment, in your terminal window or an Anaconda Prompt, run:

```
conda install -n myenv pip
conda 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 in your Anaconda Prompt 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.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: `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.

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.

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

Removing an environment

To remove an environment, in your terminal window or an Anaconda Prompt, 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 or an Anaconda Prompt, run:

```
conda info --envs
```

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