Anaconda Crash Course

March 21, 2022

1 Introduction

When writing python code, it is considered good practice to separate your python environments for different projects. The goal is maintain environments with minimal dependencies for running your projects. The reason for that is to expedite package loading times by reducing the size of the search space, and to more easily adapt projects across multiple platforms.

Anaconda is a popular, open-source python and R environment manager that aims to simplify package management and product deployment for data science and machine learning projects. It allows the user to install multiple python versions in isolated environments, as well as third-party python libraries and other software required for the project. These environments are easily installed, updated, and removed via terminal commands or a graphical user interface. Environments can also be exported and used on a different machine, sometimes even across different OS platforms.

2 Installation

The following are instructions on how to install miniconda, a minimal version of anaconda that is controlled via a shell. To install miniconda, follow your platform-specific installation instructions found here. After installation, run `conda --version` to check that the program was installed successfully.

3 Basic usage

To use anaconda, the main commands we need to know are:

```
conda env list
conda create -n <env_name> python=<python_version>
conda activate <env_name>
conda install -c conda-forge <package_name>
conda remove <package_name>
conda env export > <environment_file>
conda env update -f <environment_file>
```

Anaconda is already very well documented, but we offer below a crash course on anaconda environment and package management, highlighting some of the most useful anaconda features and commands. To better understand how anaconda works and how to best take advantage of its features run the code blocks in your terminal and observe the results.

4 Environments

An environment is a directory of isolated software within the file system that, when "active", runs software directly from the directory. Anaconda enables us to manage different environments and switch between them with ease.

4.1 Listing

We can see our installed environments using the command

```
conda env list
```

A freshly installed version of miniconda should display the base environment only, accompanied by the environment's location in the file system and a star '*' character indicating that it is the active environment. `base` is an anaconda environment that contains the basic anaconda functionality and a version of python.

4.2 Creating

We can create a new environment called `empty1` with the command

```
conda create -n empty1
```

and input `y` when prompted (or alternatively append the `-y` flag to the end of the command). This creates an empty environment into which we can install all kinds of software. If we list the environments once more with `conda env list`, we will see the newly created environment has been added to the list. However, it has not yet been activated and is therefor not in use at the moment.

4.3 Activating

The active environment name is visible to the left of the command prompt in parentheses. Anaconda maintains a stack of environments for us. When anaconda is launched, the base environment is active. We can activate an environment with the

```
conda activate <env_name>
```

command. For example, running

```
conda activate empty1
```

will activate our previously created environment. Let us create another environment using `conda create -n empty2`, and then run `conda activate empty2` to activate the newly created environment. Running

```
conda deactivate
```

will deactivate the active environment and pop the environment stack, meaning that now `empty1` should now be the active environment.

4.4 Removing

To completely remove an environment from anaconda, we must run the command

```
conda env remove -n <env_name>
```

However, the current active environment cannot be removed, so we must deactivate it before removing. For example, to remove the `empty2` we must first make sure it is not the active environment, and if it is we must run `conda deactivate` to pop the environment stack or `conda activate` to empty the entire stack and return to `base`. Then we can remove it with

```
conda env remove -n empty2
```

Warning: once an environment has been removed, it cannot be recovered unless it was exported and saved to an external file beforehand (see "Environment files" section).

5 Managing Packages

Our environments are meaningless without any third-party software. The `conda list` command shows a table of all installed packages in the current active environment. If we activate `empty1` and list its packages, we will see an empty table.

5.1 Install in active environment

We can install packages in the active environment via the

```
conda install <package_name>
```

command and responding y when prompted for confirmation (or using the `-y` flag). For example, run

```
conda install git
```

to install the latest available version of git in the conda repository that is compatible with the environment's previously installed packages, along with all of its dependencies. We can install an older version of a package by specifying it explicitly, e.g.,

```
conda install git=2.32
```

To remove git, run

```
conda remove git
```

which will uninstall git and any of its dependencies that are not needed for other packages.

5.2 Install in target environment

Note that for all the above commands, we can specify the target environment by appending `-n <env_name>` to the command, e.g.,

```
conda env install -n empty1 git
```

will install git in environment `empty1`. We can also install packages in a newly created environment simply by appending the packages to the end of the `create` command, e.g.,

```
conda create -n git_env git
```

5.3 Channels

Anaconda maintains a channels list to specify where to look for packages. The list is ordered from highest to lowest priority channels. The deafault channels list contains the single channel `defaults`

(which is actually a small collection of channels). However, many useful packages are available only on other channels. For example, the official version of the `ninja` package (an efficient build system) is found in the conda-forge channel. We can add a channel to the top of the list (highest priority) for a single install command using the `-c` option like so:

```
conda install -c conda-forge ninja
```

Note that we do not need to remember in which channel a package resides since we can search for packages in the anaconda package repository where packages from many different channels can be found with the install commands required for installation, including the channel. For example, see the nija page. This way, however, the requested channel is not permanently added to the channels list for future installs. Since conda-forge is a very popular channel for anaconda package distribution, we should consider adding it to the top of the list with the command

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

To view the channels list, run

```
conda config --show channels
```

We can search for packages in the channels list with the command

```
conda search <package_name>
```

6 Managing python

It is considred best practice for python project to have minimal environments to prevent dependency issues and inscrutable bugs. Anaconda is designed to support python environments of different versions. We can create multiple python environemnts for different projects and have them isolated and maintained effortlessly by the package manager.

6.1 Installing python

In anaconda, python is simply another package to be installed by the package manager. For example, we can turn our `empty1` environment into a python environment simply by installing python with `conda install python`. This will install the latest available python version for anaconda. We can specify a version in any granularity, e.g., `conda install ptyhon=2`, `conda install python=3.8`, `conda install 3.7.12`. However, the easiest way to create python enviroments is by installing them directly at the environment's creation like so:

```
conda create -n my_py38_env python=3.8
```

6.2 Installing libraries with conda

Once our python environment has been activated, e.g., `conda activate my_py38_env`, we can install third-party python libraries and packages as we would any other package using

```
conda install <package_name>
```

For example, we can install the official numpy package with the command

```
conda install -c conda-forge numpy
```

and we can now import the numpy library into our python scripts. Note that if we try to install numpy in an empty environment, anaconda will install the latest available python version and then numpy within it. Other versions of python on your computer will be unaffected.

6.3 installing libraries with pip

AnaIonda works seemlessly with pip, the Package Installer for Python. Once a python version has been installed, we can run the pip command from the environment. For example, when the `my_py38_env` environment we created in a previous section is active, we can run `pip --version` and see that pip is indeed installed (if not, then `conda install pip`). We can run

```
pip install numpy
```

to install numpy directly. `pip install` does not collide with `conda install` since conda is aware of installation with pip and adds the newly installed packages to `conda list` from channel `pypi`. We can overwrite that package with either `pip install` or `conda install` at any time. In our environment, we can freely use one installation method or the other, or even both simultaneously. This is incredibly useful since some packages can only be installed with `pip` and others can only be installed `conda`.

7 Environment files

Anaconda offers easy depolyment of environments by providing an API for exporting environments into and installing environments from environment files. Environment files are written in YAML, a data serialization language much like XML and JSON.

7.0.1 File structure

The file accepts an environment name field, a channels list, and a dependencies list. The dependencies list can have sublists for installing packages with another package manager, e.g., pip. Below is an example environment yaml file. The environment name is `236606-tut01`. The cahnnels are, in order of priority: pytorch, conda-forge, defaults. The python version is 3.8 and has several packages installed. `jupyterlab`, `pytorch` and `pip` are installed via `conda install` and the packages `pettingzoo[mpe]` and `multi_taxi` (installed directly from git) are installed via `pip install`.

```yaml
name: 236606-tut01
channels:
  - pytorch
  - conda-forge
  - defaults
dependencies:
  - python=3.8
  - jupyterlab
  - pytorch
  - pip
  - pip:
    - pettingzoo[mpe]
    - git+https://github.com/sarah-keren/multi_taxi
```

7.1 Export to file

To generate a YAML file representing the active environment, run

```
conda env export
```

Usually, we want to save this output to a file, so we simply redirect like so:

```
conda env export > my_env.yml
```

This command also outputs `pip` dependencies as they appear in `pip freeze` (the pip version of `conda env export`). This environment file is perfect for saving the environment as a checkpoint, a backup, or for transfering the environment to other identical machines. However, since this includes exact versions of all installed packages, and since dependencies may differ across platforms, we should try to specify minimal dependencies with loose versioning. To do this, we can run

```
conda env export --from-history > my_env.yml
```

which will specify only packages that were explicitly installed by the user, without their dependencies, as they were installed with `conda install`.

Note that using the `from-history` flag omits the pip dependencies. This is fine if all packages were installed through conda, but problematic if some were installed with pip. We can always list pip packages with `pip freeze` and add it manually to our environment file, like `export`, `freeze` outputs a full list of packages and dependencies. We can achieve a similar effect to the `--from-history` flag with the pip-chill package (installed with pip). Further note that we may need to manually edit this file in special cases, e.g., installing with pip directly form git as we did with `multi_taxi` in the above example.

Finally, notice that when using the `export` command to generate an environment file, the last line contains the `prefix` field, which shows where this environment was installed in the exporting machine. But according to this forum post, this field is ignored when installing an environment from the file.

7.2 Install from file

Given a path to some environment file, we simply run

```
conda env update -f <environment_file>
```

For example, this tutorial contains a file called `environment.yaml`. If we run

```
conda env update -f environment.yaml
```

a new environment will be created called `236606-tut01` which should contain all the packages required to run the tutorial. The `update` function creates an environment if it does not exist and updates it if there are differences between the environment and the YAML file.

8 Conclusion

Congradulations! If you have made it here and internalized all the above information, you are now a black-belt in anaconda. As we have seen throughout this document, anaconda can help us manage our python work environments for easy environment installation, deletion, control, isolation, and

deployment. For these reasons, it is a common choice for sceintific programming in both academic research and the industry.