

Notes on using Anaconda

Anaconda is a program to manage (*install, upgrade, or uninstall*) packages and environments to use with Python. It's simple to install packages with Anaconda and create virtual environments to work on multiple projects conveniently.

Even if you already have Python installed, it will be beneficial to use Anaconda/Miniconda because:

1. Anaconda comes with a bunch of data science packages; you'll be all set to start working with data.
2. Using `conda` to manage your packages and environments will reduce future issues dealing with the various libraries you'll be using.

Python Packages

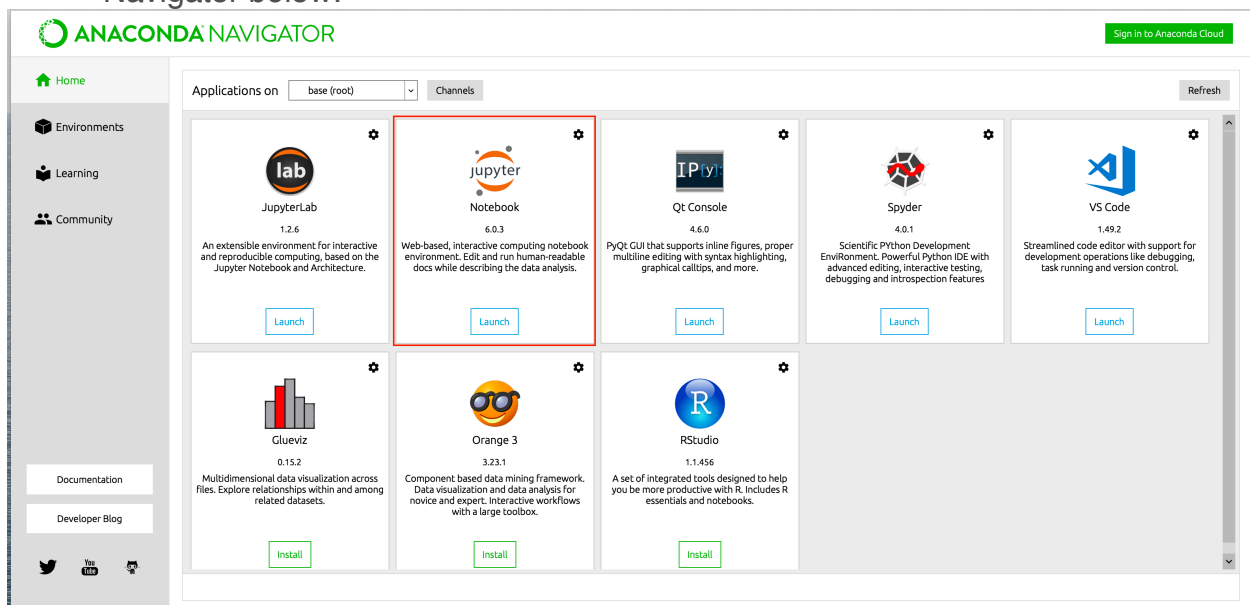
A package is a bunch of modules, where each module consists of a set of classes and function definitions. After installing a particular package, you can `import` and use the functions defined in that package.

If we install Anaconda, then a basic few packages are installed by default. However, you can install many more packages, if needed.

Anaconda Distribution

Anaconda is a fairly large download (~500 MB) because it comes with Python's most common data science packages. [Anaconda](#) is a software distribution that includes the following:

- **Anaconda Navigator** - It is a graphical user interface that helps open up any installed applications, such as Jupyter notebook or VS code editor. We will learn more about the notebook in the next lesson. See a snapshot of Anaconda Navigator below:



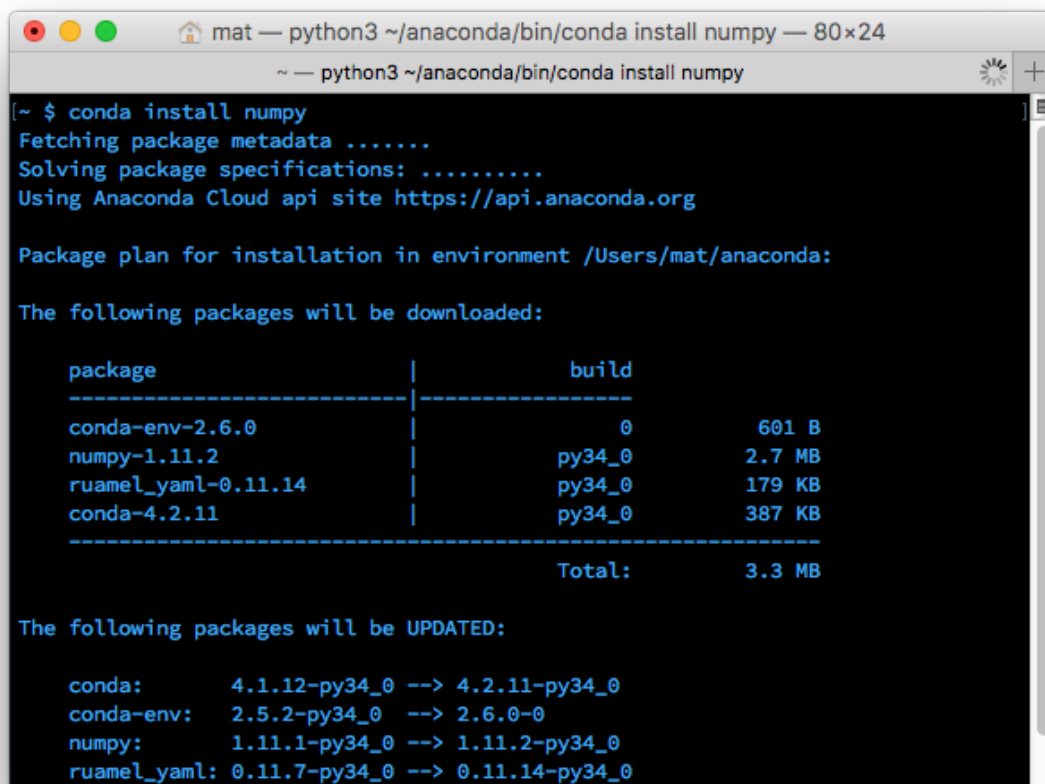
Anaconda Navigator GUI, same in both macOS/Linux and Windows. We will code in the Jupyter Notebook application

- **conda** - A command-line utility for package and environment management. Mac/Linux users can use the Terminal, and Windows users can use the "**Anaconda Prompt**" to execute **conda** commands. Windows users must run the Anaconda Prompt as an Administrator. Your first command should be `conda --version`

- **Python** - The latest version of Python gets installed as an individual package.
- Over 160 scientific packages and their dependencies are also installed.

If you don't need all the packages or need to conserve bandwidth or storage space, there is an option for you - **Miniconda**. Be aware that Miniconda will not come with NumPy and Pandas.

Overview - Managing Packages using either **pip** or **conda**



```
mat — python3 ~/anaconda/bin/conda install numpy — 80x24
~ — python3 ~/anaconda/bin/conda install numpy

[~ $ conda install numpy
Fetching package metadata .....
Solving package specifications: .....
Using Anaconda Cloud api site https://api.anaconda.org

Package plan for installation in environment /Users/mat/anaconda:

The following packages will be downloaded:

package | build
-----|-----
conda-env-2.6.0 | 0 601 B
numpy-1.11.2 | py34_0 2.7 MB
ruamel_yaml-0.11.14 | py34_0 179 KB
conda-4.2.11 | py34_0 387 KB
-----|-----
Total: 3.3 MB

The following packages will be UPDATED:

conda: 4.1.12-py34_0 --> 4.2.11-py34_0
conda-env: 2.5.2-py34_0 --> 2.6.0-0
numpy: 1.11.1-py34_0 --> 1.11.2-py34_0
ruamel_yaml: 0.11.7-py34_0 --> 0.11.14-py34_0
```

Installing numpy with conda

The **conda** and **pip** both are the Python package managers. Package managers are used to installing libraries and other software on your computer. **pip** is the default package manager for Python libraries, whereas **conda** focuses only on the packages that are available from the Anaconda distribution.

Which one should I prefer - `pip` or `conda`?

There are two points you can consider before making a choice:

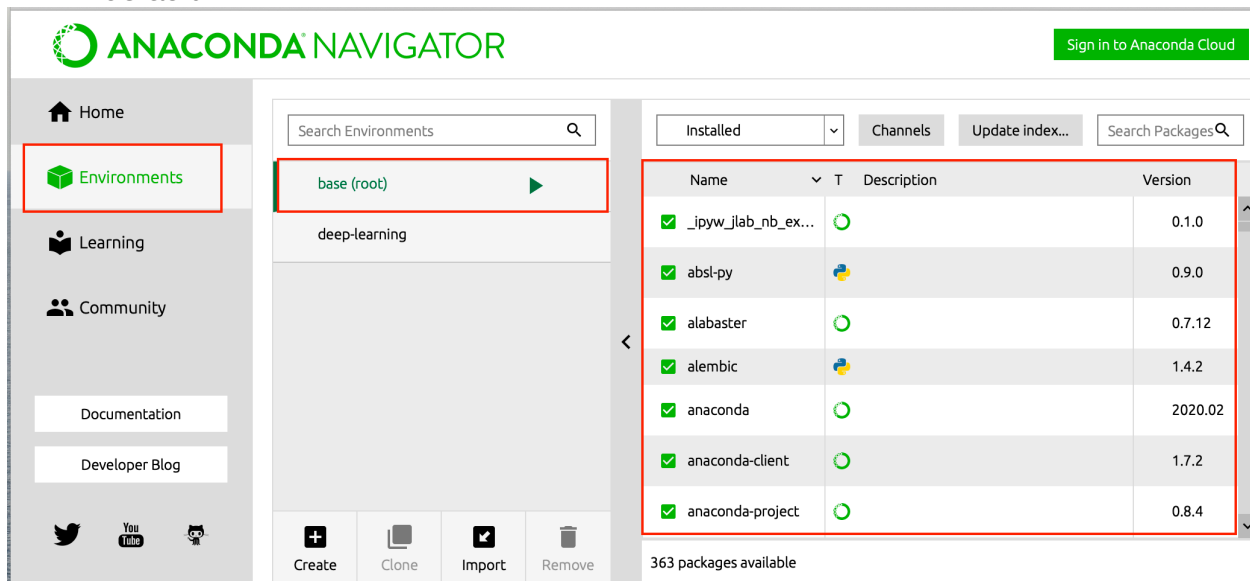
1. The available packages available from the Anaconda distribution in `conda` focus on data science, whereas `pip` is for general use. Conda installs precompiled packages. For example, the Anaconda distribution comes with Numpy, Scipy, and Scikit-learn compiled with the [MKL library](#), speeding up various math operations. **But, sometimes, you may need packages other than the ones listed on the Anaconda distribution.**
2. Pip can install both Python and non-Python packages. Pip can install any package listed on the [Python Package Index](#) (PyPI).

You can (and will) still use `pip` alongside `conda` to install packages.

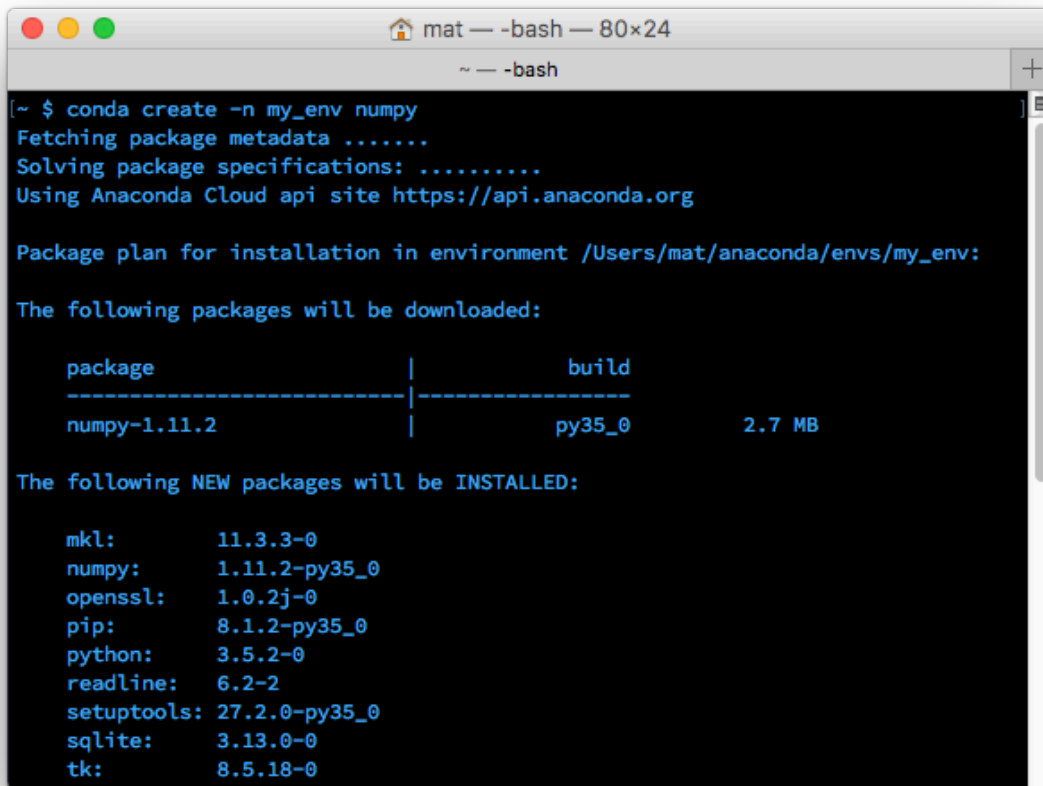
Environments

A Python environment comprises a particular version of each of the following:

- Python interpreter,
- Python-packages, and
- The utility scripts, such as `pip`.
- It is possible to have two or more environments residing on the same computer *virtually*. If you are using Anaconda, you are in the `base (root)` environment by default.



The default `base (root)` environment in Anaconda

A terminal window titled 'mat - bash - 80x24' showing the output of the command 'conda create -n my_env numpy'. The output includes progress bars for fetching metadata and solving specifications, the Anaconda Cloud API URL, a package plan for the environment, a list of packages to be downloaded (numpy-1.11.2, build py35_0, 2.7 MB), and a list of new packages to be installed (mkl, numpy, openssl, pip, python, readline, setuptools, sqlite, tk).

```
mat - bash - 80x24
~ - bash

[~ $ conda create -n my_env numpy
Fetching package metadata .....
Solving package specifications: .....
Using Anaconda Cloud api site https://api.anaconda.org

Package plan for installation in environment /Users/mat/anaconda/envs/my_env:

The following packages will be downloaded:

package | build
-----|-----
numpy-1.11.2 | py35_0 2.7 MB

The following NEW packages will be INSTALLED:

mkl:      11.3.3-0
numpy:    1.11.2-py35_0
openssl:  1.0.2j-0
pip:      8.1.2-py35_0
python:   3.5.2-0
readline: 6.2-2
setuptools: 27.2.0-py35_0
sqlite:   3.13.0-0
tk:       8.5.18-0
```

Creating an environment with conda

Along with managing packages, Conda is also a virtual environment manager. It's similar to [virtualenv](#) and [pyenv](#), other popular environment managers.

Why do you need a Virtual Environment?

Each virtual environment remains isolated from other virtual environments, and the default “system” environment. **Environments allow you to separate and isolate the packages you are using for different projects.** Often you'll be working with code that depends on different versions of some library. For example, you could have code that uses new features in NumPy, or code that uses old features that have been removed. It's practically impossible to have two versions of NumPy installed at once. Instead, you should make an environment for each version of Numpy then work in the appropriate environment for the project.

This issue also happens a lot when dealing with Python 2 and Python 3. You might be working with old code that doesn't run in Python 3 and new code that doesn't run in Python 2. Having both installed can lead to a lot of confusion and bugs. It's much better to have separate environments.

You can also export the list of packages in an environment to a file, then include that file with your code. This allows other people to easily load all the dependencies for your code. Pip has similar functionality with `pip freeze > requirements.txt`.

Installing Anaconda

Anaconda is available for Windows, Mac OS X, and Linux. Follow the links below to get started:

1. Download the installer from <https://www.anaconda.com/download/>. Choose the Python 3.7 or higher version, and the appropriate 64/32-bit installer. If you already have Python installed on your computer, this won't break anything. Instead, the default Python used by your scripts and programs will be the one that comes with Anaconda.

After installation, you're automatically in the default conda environment with all packages installed which you can see below. You can check out your own install by entering the following command into your terminal.

```
conda list
```

List of Applications Installed with Anaconda

As we read on the previous page, the following packages will get installed with Anaconda:

- **Anaconda Navigator** - a GUI for managing your environments and packages
- **conda** - a command-line utility
- **Python** - The latest version of Python gets installed as an individual package.
- **Anaconda Prompt** - [Only for Windows] a terminal where you can use the command-line interface to manage your environments and packages
- A bunch of applications, such as **Spyder**. It is an IDE geared toward scientific development. In total, over 160 scientific packages and their dependencies are also installed.

To avoid errors later, it's best to update all the packages in the default environment. Open the Terminal/ Anaconda Prompt application. In the prompt, run the following commands:

```
conda upgrade conda
```

```
conda upgrade --all
```

and answer yes when asked if you want to install the packages. The packages that come with the initial install tend to be out of date, so updating them now will prevent future errors from out of date software.

Note: In the previous step, running `conda upgrade conda` should not be necessary because `--all` includes the conda package itself, but some users have encountered errors without it.

Managing Packages

Install Packages

Once you have Anaconda installed, managing packages is fairly straightforward. To install a package, type the following command in your terminal.

```
conda install PACKAGE_NAME
```

Conda also automatically installs dependencies for you. For example `scipy` uses and requires `numpy`. If you install just `scipy` (`conda install scipy`), Conda will also install `numpy` if it isn't already installed.

Remove Packages

Most of the commands are pretty intuitive. To uninstall, use

```
conda remove PACKAGE_NAME
```

Update Packages

To update a package, use

```
conda update package_name
```

If you want to update all packages in an environment, which is often useful, use `conda update --all`. And finally, to list installed packages, it's `conda list` which you've seen before.

Search a Package to Install

If you don't know the exact name of the package you're looking for, you can try searching with `conda search *SEARCH_TERM*`. For example, I know I want to install [Beautiful Soup](#), but I'm not sure of the exact package name. So, I try `conda search *beautifulsoup*`. Note that your shell might expand the wildcard `*` before running the conda command. To fix this, wrap the search string in single or double quotes like `conda search '*beautifulsoup*'`.

Managing Environments

As you saw on the previous page, `conda` can be used to create environments to isolate your projects. To create an environment, use the following command in your Terminal/Anaconda Prompt.

```
conda create -n env_name [python=X.X] [LIST_OF_PACKAGES]
```

Here `-n env_name` sets the name of your environment (`-n` for name) and `LIST_OF_PACKAGES` is the list of packages you want to be installed in the environment. If you wish to install a specific version of Python to be installed, say 3.7, use `python=3.7`. For example, to create an environment named `my_env` with Python 3.7, and install NumPy and Keras in it, use the command below.

```
conda create -n my_env python=3.7 numpy Keras
```

```
mat — -bash — 80x24
~ — -bash

[~ $ conda create -n my_env numpy
Fetching package metadata .....
Solving package specifications: .....
Using Anaconda Cloud api site https://api.anaconda.org

Package plan for installation in environment /Users/mat/anaconda/envs/my_env:

The following packages will be downloaded:

package | build
-----|-----
numpy-1.11.2 | py35_0 2.7 MB

The following NEW packages will be INSTALLED:

mkl:      11.3.3-0
numpy:    1.11.2-py35_0
openssl:  1.0.2j-0
pip:      8.1.2-py35_0
python:   3.5.2-0
readline: 6.2-2
setuptools: 27.2.0-py35_0
sqlite:   3.13.0-0
tk:       8.5.18-0
```

Create `my_env` environment with the NumPy package in it.

When creating an environment, you can specify which version of Python to install in the environment. This is useful when you're working with code in both Python 2.x and Python 3.x. To create an environment with a specific Python version, use either of the following commands:

```
conda create -n py3_env python=3
```

or

```
conda create -n py2_env python=2
```

I actually have both of these environments on my personal computer. I use them as general environments not tied to any specific project, but rather for general work with each Python version easily accessible. These commands will install the most recent version of Python 3 and 2, respectively. To install a specific version, use `conda create -n py python=3.6` for Python 3.6.

Entering (Activate) an environment

Once you have an environment created, you can enter into it by using:

For conda 4.6 and later versions on Linux/macOS/Windows, use

```
conda activate my_env
```

#For conda versions prior to 4.6 on Linux/macOS, use

```
source activate my_env
```

```
#For conda versions prior to 4.6 on Windows, use  
activate my_env
```

When you're in the environment, you'll see the environment name in the terminal prompt. Something like `(my_env) ~ $`.

List the Installed Packages in the Current Environment

The environment has only a few packages installed by default, plus the ones you installed when creating it. You can check this out with

```
conda list
```

Installing packages in the environment is the same as before: `conda install package_name`. Only this time, the specific packages you install will only be available when you're in the environment.

Deactivate an Environment

To leave the environment, type `conda deactivate` (on OSX/Linux) or `deactivate` (Windows).

```
# For conda 4.6 and later versions on Linux/macOS/Windows, use  
conda deactivate
```

```
#For conda versions prior to 4.6 on Linux/macOS, use  
source deactivate
```

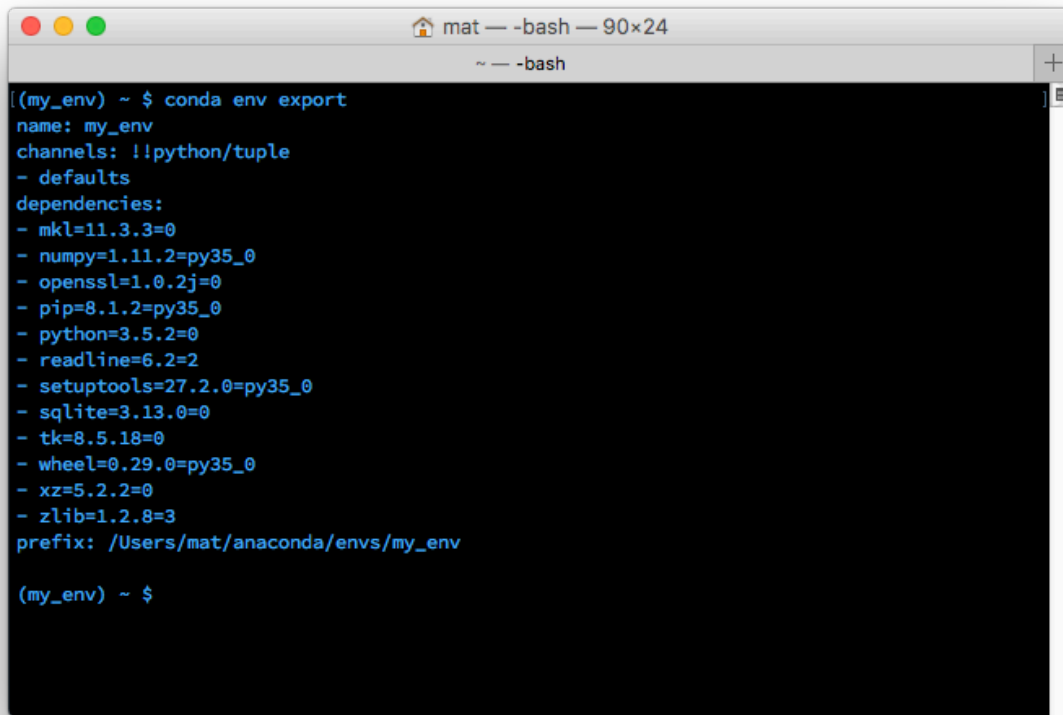
```
#For conda versions prior to 4.6 on Windows, use  
deactivate
```

```
conda create -n tea_facts python=3  
source activate tea_facts  
conda list  
conda install numpy pandas matplotlib
```

Saving and loading environments

A really useful feature is sharing environments so others can install all the packages used in your code, with the correct versions. Let's see all the package-names, including the Python version present in the current environment, using the command:

```
conda env export
```


A terminal window titled 'mat - bash - 90x24' with a subtitle '~ - bash'. The terminal shows the command '(my_env) ~ \$ conda env export' and its output. The output lists the environment name 'my_env', channels '!!python/tuple', a list of dependencies with their versions (mkl, numpy, openssl, pip, python, readline, setuptools, sqlite, tk, wheel, xz, zlib), and the prefix path '/Users/mat/anaconda/envs/my_env'. The prompt '(my_env) ~ \$' is shown at the bottom.

```
(my_env) ~ $ conda env export
name: my_env
channels: !!python/tuple
- defaults
dependencies:
- mkl=11.3.3=0
- numpy=1.11.2=py35_0
- openssl=1.0.2j=0
- pip=8.1.2=py35_0
- python=3.5.2=0
- readline=6.2=2
- setuptools=27.2.0=py35_0
- sqlite=3.13.0=0
- tk=8.5.18=0
- wheel=0.29.0=py35_0
- xz=5.2.2=0
- zlib=1.2.8=3
prefix: /Users/mat/anaconda/envs/my_env

(my_env) ~ $
```

Exported environment printed to the terminal

In the above image, you can see the name of the environment, and all the dependencies (*along with versions*) are listed. You can save all the above information to a [YAML](#) file, `environment.yaml`, and later share this file with other users over GitHub or other means. This file will get created (or overwritten) in your current directory.

`conda env export > environment.yaml`

The second part of the export command above, `> environment.yaml` writes the exported text to the `environment.yaml`. This file can now be shared using Github repository (or any other means), and others will be able to create the same environment you used for the project.

Create an environment

To create an environment from an environment file, use the following command:

`conda env create -f environment.yaml`

The above command will create a new environment with the same name listed in `environment.yaml`.

Listing environments

If you forget what your environments are named (happens to me sometimes), use *either* of the commands below to list out all the environments you've created.

`conda env list`

`conda info --envs`

You should see a list of environments, there will be an asterisk next to the environment you're currently in. The default environment is called `base`.

List the packages inside an environment

To view the list of packages, run the following command in your terminal / Anaconda Prompt,:

```
# If the environment is not activated  
conda list -n env_name
```

```
# If the environment is activated  
conda list
```

```
# To see if a specific package, say `scipy` is installed in an environment  
conda list -n env_name scipy
```

Removing an environment

If there are environments you don't use anymore, use the command below to remove the specified environment (here, named `env_name`).

```
conda env remove -n env_name
```

Sharing Environments

When sharing your code on GitHub, it's good practice to make an environment file and include it in the repository. You can do this using `conda` as:

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

Share the List of Dependencies

For users not using `conda`, you may want to share the list of packages installed in the current environment. You can use `pip` to generate such a list as `requirements.txt` file using:

```
pip freeze > requirements.txt
```

Later, you can share this `requirements.txt` file with other users over Github. Once a user (or yourself) switches to another environment, you can install all the packages mentioned in the `requirements.txt` file using:

```
pip install -r requirements.txt
```