Managing Python Packages with PIP

Dr. Saad Laouadi

Econometrics and Data Science Academy

Outline

Introduction

In this small manual, we will show how to use:

- Basic PIP Commands on:
  1. Windows OS
  2. Unix Based systems (Linux and Mac OS)
- Creating and working with python virtual environment.

Introduction to PIP

Definition

- **PIP** Stands for **Package Installer for Python**.
- **PIP** gets installed automatically when installing python from the official website.
- **PIP** is available on Python virtual environment.

Checking the PIP version:

```
root$ pip --version
```

You can also use **pip -V** (capital V).

Upgrading PIP

1. **Upgrading PIP version for Linux:**

```
root$ python -m pip install --upgrade pip
```

2. **Upgrading PIP version for Windows:**

```
C:\Users\Name> py -m pip install --upgrade pip
```

Note: You can leave out python -m or py -m and PIP will be upgraded. But it is best to use it.

PIP Syntax

PIP manager has the following syntax:

pip <command> [options]

- **Command** or argument is required to tell PIP which program or process to run.
- **Options** are additional optional options that can be given to the command to change how its default behavior. The options are written using **dash option (-U)** or **dash dash option** for example *(- -help)*
- Here is an example of pip installing a package in verbose mode.

```
root$ pip install -v
```

Getting PIP Help

When using the command line, *help* is your friend.. To get the help of PIP use the one of the following commands.

```
root$ pip -h
```

or

```
root$ pip --help
```

This command will give the whole manual, which you need to scroll up to get to the beginning of the help manual.
Combining pip -h with | **less** allows you to page through the manual line by line by using **enter** or page by page using **space bar**. Press **q** to **quit**.

```
root$ pip --help | less
```

```
Usage:
  pip <command> [options]

Commands:
  install                     Install packages.
  download                    Download packages.
  uninstall                   Uninstall packages.
  freeze                      Output installed packages in requirements format.
  list                        List installed packages.
  show                        Show information about installed packages.
  check                       Verify installed packages have compatible dependencies.
  config                      Manage local and global configuration.
  search                      Search PyPI for packages.
  cache                       Inspect and manage pip's wheel cache.
  index                       Inspect information available from package indexes.
  wheel                       Build wheels from your requirements.
  hash                        Compute hashes of package archives.
  completion                  A helper command used for command completion.
  debug                       Show information useful for debugging.
  help                        Show help for commands.
```

Figure: PIP Help

Installing New Package

The general syntax for command is like this:

1. Linux and Mac: **python -m pip install [options] <...> [other options]**
2. Windows: **py -m pip install [options] <...> [other options]**

For example, installing numpy:

```
root$ python -m pip install numpy
```

or [1]

```
root$ pip install numpy
```

Note that when install a package, all the dependencies will be installed as well. [2]

[1] replace python -m with py -m for windows

[2] You'd better use the first syntax because to explicitly indicate where the package will be installed.

Installing a Specific Version

You can specify which version you want to install using ($==$) or a minimum version using ($>=$)

- Install a specific version

```
root$ python -m pip install numpy==1.15.1
```

- Install minimum version

```
root$ python -m pip install "numpy>=1.15.4"
```

Upgradind Packages

Proving a **-U or –upgrade** option to the ***install command*** will install the latest available version of the package.

```
root$ python -m pip install -U numpy
```

Or

```
root$ python -m pip install --upgrade numpy
```

Installing Multiple Package

You can install or upgrade many packages at one by listing them one after another using space.

```
root$ python -m pip install -U numpy pandas requests
```

Removing Packages

Remove a package using uninstall command:

```
root$ python -m pip uninstall numpy
```

You can add the option (-y or - -yes) to stop the uninstallation confirmation.

```
root$  python -m pip uninstall --yes numpy
```

Listing Installed Packages

To see all the available install packages in the active environment. use ***pip list***.

```
root$ pip list
```

Here is an example

```
Package                 Version
----------              ----------
certifi                 2021.10.8
charset-normalizer      2.0.12
idna                    3.3
numpy                   1.22.3
pip                     22.0.4
requests                2.27.1
setuptools              58.1.0
urllib3                 1.26.9
```

Package Information

What if you want to see the information about a specific package. The *pip show package_name* is right one for you.

```
root$ pip show numpy
```

The results as follows:

```
Name: numpy
Version: 1.22.3
Summary: NumPy is the fundamental package for array computing with Python.
Home-page: https://www.numpy.org
Author: Travis E. Oliphant et al.
Author-email:
License: BSD
```

Freezing Packages

- In the scenario of developing projects, required packages versions are necessary
- How can you get the details about all the packages with their versions to install them in another machine so the project can run properly.
- The **pip freeze** command outputs the installed packages with their versions.

```
root$ python -m pip freeze
```

The results should look like this:

```
certifi==2021.10.8
charset-normalizer==2.0.12
idna==3.3
numpy==1.22.3
requests==2.27.1
urllib3==1.26.9
```

Creating Requirements File

- The **pip freeze** command output is necessary to be used later. Thus, you can use the command line to redirect the output to a file.
- The $>$ **operator** is a **redirection operator**
- Call the file the output is redirected to **requirements.txt**

Here is how to create a requirements.txt file

```
root$ python -m pip freeze > requirements.txt
```

Requirements file Creation steps Example

```
# Create a directory
mkdir myProject
# Change the directory
cd myProject
# Create the requirements file
python -m pip freeze > requirements.txt
# Check the file content, use TYPE command on windows and CAT command on Mac
cat requirements.txt
certifi==2021.10.8
charset-normalizer==2.0.12
idna==3.3
numpy==1.22.3
requests==2.27.1
urllib3==1.26.9
```

Using Requirements

- **Why did we create a requirements.txt file?**
- **The answer to use it to install the packages with their required versions**
- In order to use the requirements file you need to use the option **-r requirements** when installing.

Example (Installing packages with requirements option)

```
root$ python -m pip install -r requirements.txt
```

Check Command

- Some Python packages depend upon other packages. The latter are called **dependencies**.
- Sometimes you import a package then use it, you run into errors. Why?
- One of the reasons is its dependencies not installed or not the right version. (One of the advantages of using virtual environment)
- To check the dependencies are available, the command **pip check** comes in handy.

```
root$ python -m pip check
```

If everything is ok. you get this message

```
No broken requirements found
```

Download Command

- Developing packages needs to read source code of other packages
- Do you have to download each package manually using the mouse?
- This is a tedious task.
- The **pip download command** is useful is this case:
  1. The **pip download** command works like **pip install** but it does not install the package(s).
  2. The command **pip download** downloads the package(s) to the current directory unless specified otherwise.
  3. The command **pip download** works with requirements as well.

```
root$ python -m pip download numpy
```

Wheel Command

- The **pip wheel** command is useful when developing python packages
- This command builds **Wheel** archives for your requirements and dependencies.
  - Wheel is a built-package format, and offers the advantage of not recompiling your software during every install. For more details, see the wheel docs.
  - Requirements: setuptools>=0.8, and wheel.
  - "pip wheel" uses the bdist_wheel setuptools extension from the wheel package to build individual wheels.

Example

1. Build wheels for a requirement (and all its dependencies), and then install:

```
root$ pip wheel --wheel-dir=/mydir/here PackageName
root$ pip install --no-index --find-links=/mydir/here PackageName
```

2. Build a wheel for a package from source:

```
root$ python -m pip wheel --no-binary PackageName PackageName
```

Hash Command

The *pip hash* command computes the hash number of a downloaded package to make sure it's downloaded correctly.

```
root$ python -m pip download numpy
root$ python -m pip hash  numpy-1.22.3-cp39-cp39-macosx_\_.whl
```

The output will be something like this:

```
numpy-1.22.3-cp39-cp39-macosx_10_14_x86_64.whl:
--hash=sha256:2c10a93606e0b4b95c9b04b77dc349b398fdfbda382d2a39ba5a822f669a0123
```

Cache Command

- The **pip cache** command inspects and manages the **pip's wheel** cache.
- This command has several subcommands which are:
  1. **dir**: Show the cache directory.
  2. **info**: Show information about the cache.
  3. **list**: List filenames of packages stored in the cache.
  4. **remove**: Remove one or more package from the cache.
  5. **purge**: Remove all items from the cache.

```
root$ python -m pip cache dir
```

Config Command

- The **pip config** manages local and global configuration.
- This command has several subcommands which are:
  1. **list:** List the active configuration (or from the file specified).
  2. **edit:** Edit the configuration file in an editor.
  3. **get:** Get the value associated with name.
  4. **set:** Set the name=value.
  5. **unset:** Unset the value associated with name.
  6. **debug:** List the configuration files and values defined under them.
- If none of –user, –global and –site are passed, a virtual environment configuration file is used if one is active and the file exists. Otherwise, all modifications happen to the user file by default.

```
root$ python -m pip config edit
```

Debug Command

The debug command is used for debugging only, which displays the debugging information. And it has the following syntax:

python -m pip debug <options>

```
root$ python -m pip debug
```

Warning

This command is only meant for debugging. Its options and outputs are provisional and may change without notice.

Bonus: PIP-Check

You can install pip-check to check the installed packages

```
root$ python -m pip install pip-check
```

Use the help and check your environment

```
root$ pip-check --help
root$ pip-check --cmd==pip3 --hide-unchanged
```

Copyright Information

This slide show was prepared by Dr. Saad Laouadi. Independent Researcher . It is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License. Use any part of it as you like and share the result freely.