Python Package Installer (PIP)

Python was created as open-source software, and this also works as an invitation for all coders to maintain the whole Python ecosystem as an open, friendly and free environment. To make this happen, some tools to help creators to publish, maintain and take care of their code need to be provided.

Two basic entities have to be established and kept in action, a centralized **repository** for all available software packages, and a tool allowing users to **access** the repository.

PyPI

What is the repository then? Its named **PyPI** (short for **Py**thon **P**ackage **I**ndex), it is maintained by a workgroup named Packaging Working Group, a part of the Python Software Foundation, whose main task is to support Python developers in efficient code dissemination. Here is their website: PackagingWG - PSF Wiki (python.org).

You can find the PyPI website here at PyPI · The Python Package Index.

Until now (1/4/2022 **AND THIS IS NOT AN APRIL FOOL'S JOKE**), there are 366897 projects, consisting 5836784 files managed by 582865 users. These three numbers clearly show the potency if the Python community and the importance of developer cooperation.

Keep in mind that PyPI is **not** the only existing Python repository, there are a lot of them, but for sure PyPI is the most important one. Its likely that someday you and your colleagues may want to create your own repository.

PIP

PyPI is completely free, meaning that you can just pick a code and use it. There are a lot of software inside and is available 24/7.

There is a specific tool you'll need to use what PyPI offers, and that is named `pip`, which of course, is an acronym, but its more complex than the previously mentioned PyPI.

Why? pip means "pip installs packages" and the pip inside "pip installs packages" means "pip installs packages" and so on, ITS AN INFINITE RECURSION!!! (It's called recursive acronyms if you'd like to know)

Off-topic fun fact: Another famous recursive acronym is *Linux*, which can be interpreted as "*Linux is Not Unix*".

How to get *pip* ready? You may ask. Some Python installations comes with *pip*, some don't. It doesn't only depend on the OS you use, although this is a very important factor.

Microsoft Windows

The MS Windows Python installer already contains *pip*, and so no other steps are needed to install it. Unfortunately, if the PATH variable I misconfigured, *pip* may be unavailable.

To check whether your *pip* is running fine, try this

1. Open a windows console (Command Prompt or PowerShell, whatever you'd like)
2. Type and execute (press ENTER) the following command: `pip --version`.
3. In the optimal situation you'll get this.

```
C:\Users\user>pip --version
pip 21.2.4 from C:\program files\Python3xx\lib\site-packages\pip (python 3.10)
```

Yellow words are names subject to the change by the name of the user and file location.

Red words are subject to change due to the version of your Python and *pip*.

If the response is absent, this may mean the PATH variable either incorrectly points to the location of the Python binaries, or doesn't point at all.

The easiest way to reconfigure the PATH variable is to **reinstall** Python.

Linux

Different Linux distributions may behave differently when it comes to using *pip*. Some of them (for example *Gentoo*) closely bound to Python may use it internally, may have *pip* already preinstalled and are instantly ready to work for you.

Some distributions of Linux may have **both** Python 2 and Python 3 coexisting, such systems may launch Python 2 as the default version. In this case, there may be two different *pip*s identified as *pip* (**or** *pip2*) and *pip3*.

In such case typing `pip --version` summons the *pip* from Python 2, typing `pip3 –version` can check the version of *pip3*.

In some other cases (like some versions of Ubuntu), they don't have *pip* preinstalled for you. You have two possibilities.

1. Install *pip* as a system package using a dedicated package manager
2. Install *pip* using internal Python mechanisms.

I recommend using the first choice, type the following command.

```
sudo apt install python3-pip
```

MacOS

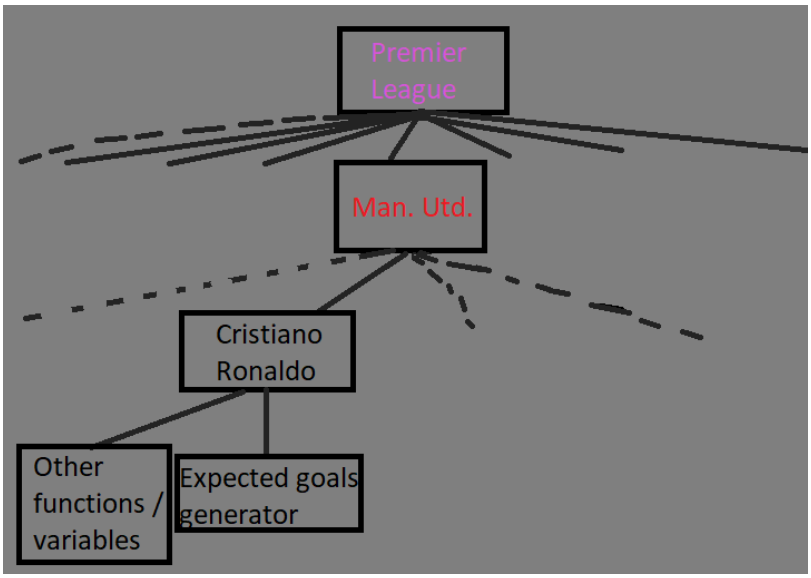
You've installed *pip* is you used the *brew* installer, check by the command:

```
pip3 –version
```

Now we've got *pip* ready, so I am going to limit the focus to Microsoft Windows only, as its behaviours (should be) the same in all OSs.

Dependencies

Dependency is a phenomenon that appears every time you're going to use a piece of software that relies on other software, this may include more than 1 level of software development.



Here is an example of dependency, let's say you've created a Python application called *Premier League*, able to predict future Premier League results with 99% (if you actually do that, please contact me **IMMEDIATELY**). You've used some existing code to achieve that (for example TensorFlow for Machine Learning). Does this mean that a potential package user is obliged to trace all dependencies and manually install all the needed packages? That would be horrible, wouldn't it?

Definitely yes, don't be surprised that this has its own name called "dependency hell".

But how do we deal with that? Is everyone just doomed to visited “hell” in order to run the code?

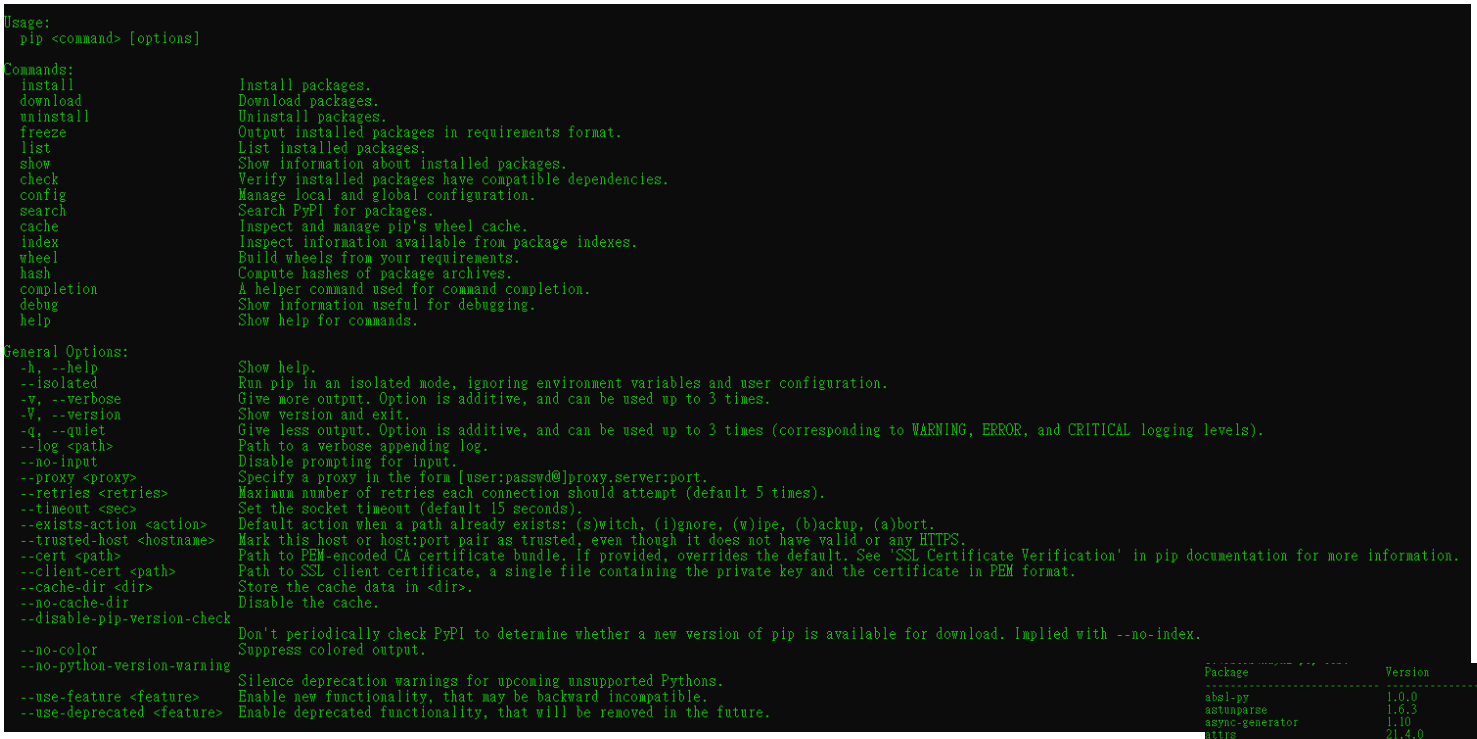
Fortunately, *pip* can do all these for us, it can discover, identify and resolve all dependencies. Moreover, it can do it optimally, meaning that it can avoid any unnecessary downloads and reinstalls.

How to use *pip*

Help

Type the following command in Command Prompt or PowerShell.

```
pip help
```



You should’ve got this long list of commands; how do we specify the listed operations?

Type this in the Command Prompt / PowerShell

```
pip help operation
```

Where operation is yet to be specified.

For example, we type

```
pip help install
```

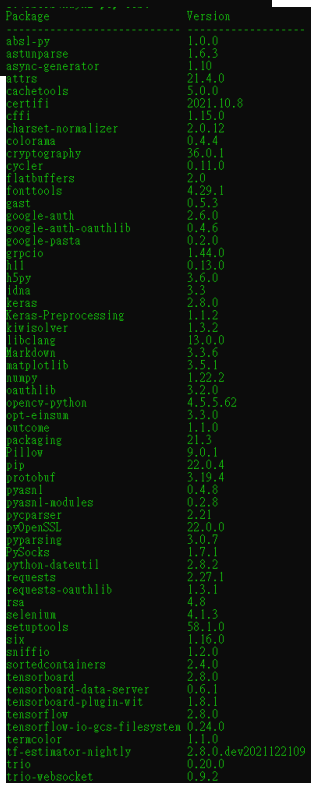
This will show you the detailed information about using and parameterizing the *install* command (and yes, the list is even longer).

List

If you would like to have a glance at all the Python packages you’ve installed so far, you can use the *list* function, like this:

```
pip list
```

At the right side is a part of the list I’ve got, what you’ve got is totally up to do. However, for sure you will see two lines on the list: *pip* and *setuptools*. This



happens because the OS is convinced that a user wanting *pip* will very likely need the *setuptools* soon, and it's not wrong.

Show

The *pip list* isn't very informative, and it may not satisfy your curiosity. There is a command that can tell you more about any of the **installed** packages.

```
pip show package_name
```

An example will be

```
pip show pip
```

```
Name: pip
Version: 22.0.4
Summary: The PyPA recommended tool for installing Python packages.
Home-page: https://pip.pypa.io/
Author: The pip developers
Author-email: distutils-sig@python.org
License: MIT
Location: c:\users\                                    \python310\lib\site-packages
Requires:
Required-by:
```

This is what you should've get.

You may ask where this data comes from? The information appearing on the prompt is taken from inside the package being shown. In other words, the package's creator is obliged to equip it with all the needed data (or more precisely, metadata).

The power of *pip* comes from the fact that it's actually a gateway to the Python software universe. You can browse and install any of the thousand's projects and packages in the PyPI repositories. Keep in mind that *pip* does **not** store all PyPI content locally in your computer as it is unnecessary and uneconomical.

So how does it work? Pip uses the **internet** to query PyPI and to download the requested data.

Search

One example of this is when you want to search through PyPI. The command would be:

```
pip search any_string
```

The *any_string* parameter provided by the user (you) will be searched on the **names** of all the packages and the **summary strings** of all the packages.

There might be a LOT of found results, so try to be as specific as possible. For example:

```
pip search pip
```

This produces more than 100 lines of results.

If you're not a fan of console reading, you can use this link to search: Search results · PyPI

Install

Assuming now you've found your targeted package and wants to install it onto your computer.

The command would simply be:

```
pip install package_name
```

By default, this will install the package system-wide, if you're not using those or don't want to install the package system-wide, you can install for yourself only using the command:

```
pip install --user package_name
```

Pip will show a textual animation of the installation progress, after finishing, you can use:

```
pip show package_name
```

and

```
pip list
```

to get more information about what just happened.

Now the package you just installed is fully accessible, a simple

```
import
```

would've done the job.

Update

```
pip install
```

has two important additional abilities:

1. Update a locally installed package
   Command:
   ```
   pip install -U package_name
   ```
2. Install a user-specified version of a package (it installs the newest available version by default)
   Command:
   ```
   pip install package_name==package_version
   ```
   Note the double equal sign.

Uninstall

Let's say you no longer need a package and decided to get rid of it. The pip command *uninstall* can help too! It will execute all the needed steps. Command:

pip uninstall *package_name*

Pip will want to know if you're sure about the choice you're making

```
Proceed (y/n)?
```

'y' stands for Yes and 'n' stands for No.

The user (you) has to type either 'y' or 'n' to complete the process.

Key notes

1. A repository us designed to collect and share free Python code exists and works under the name Python Package Index (PyPI), the website: https://pypi.org/
2. A specialized tool has been created and its name is *pip* (pip installs packages…) As *pip* may not be deployed as a part of standard Python installation, it is possible that you will have to install manually. Pip is a console tool.
3. Commands to check pip's version

   ```
   pip --version
   ```

   or

   ```
   pip3 --version
   ```

4. The list of main *pip* activities/commands looks as follows.

   ```
   pip help operation
   pip list
   pip show package_name
   pip search any_string
   pip install package_name
   pip install --user package_name
   pip install -U package_name
   pip uninstall package_name
   ```

   Explanation of all these commands can be found above.