Managing Packages / Modules and Python versions

PyPA: Python packaging authority

maintains relevant projects and packaged version

provide the tool we use for installing packages: pip

Repository: pypa/pip on Github

How to install pip?

→ It is already installed if you installed Python from python.org

- requires: Python ≥ 2.7.9 or Python ≥ 3.4

→ May not be installed on some Linux distros (install with `sudo apt-get install python3-pip`)

* Update pip version:

Linux / MacOS: pip install -U pip

Windows: python -m pip install -U pip (or py)

} the better way is: python3 -m pip install -U pip

Using pip:

pip is a program, used via CLI

usually when installed it is added to the operating system environmental variable PATH

PATH: contains a list of folders where the executable programs are located

to display the PATH:

Windows 10+: $Env:Path

MacOS/Linux: echo $PATH

executables are searched in the folders listed in the PATH variable

for example, when you type `python` in the CLI, then we search for an executable in the folders listed in the order they appear from first to last

in my case:

echo $PATH

the first folder contains the python 3.7 installation files

the folder has these files: python3, python3.7, pip3, pip3.7, ...

if on CLI we run `python3` or `python3.7` we will load this python version

observe there isn't a file named 'python', this is because MacOS comes with a python 2.7 bundled with the OS, and some of the system processes may use it; if we overwrite it the OS may run into trouble

now if we call python3 we will end up with python3.7

the second folder contains the python3.6 installation files

the folder has these files: python3, python3.6

if on CLI we run 'python3' we will get `python3.7` because of the path

if we run 'python3.6' then we will get the actual 3.6 version

if two executables have the same name then we execute the one that first shows up in the path

How do I know which executable will be run when we call it?

MacOS/Linux: which name_of_executable

Windows: Get-Command name_of_executable

{ New-Alias which Get-Command
"`n New-Alias which Get-Command" | add-content $profile

In my case:

which python /usr/local/bin/python

which python3 /Library/Frameworks/Python.framework/Versions/3.6/bin/python3.6

If you cannot run pip or pip3 directly it could be because it is not in the PATH. Then you can either add it to the path or run it by giving the entire path to the executable.

You can use pip from within Python (if you are on Windows, for example).

Ways to use pip:

```
pip <pip arguments>
python -m pip <pip arguments>
```

↑ depends on which python you are calling (what is the difference between calling { python -m pip --version / python3 -m pip --version }

Where pip finds its packages?

Mainly from PyPI (Python Package Index)

url: pypi.org

can search for packages developed by the community

you can generate your own package and share on PyPI

→ see packaging.python.org/tutorials/packaging-projects/

can also install from other sources: version control, local projects, distribution files

How to install?

```
pip install PackageName
pip install PackageName==1.0.4        specify the exact version
pip install 'PackageName>=1.0.4'      specify the minimum version (notice the use of quotation marks)
```

Where is the package installed ?

In general the packages are installed in a folder called:

/Library/Frameworks/↑/<python version number>/lib/python3.7/site-packages/

Python.framework/Versions

If you want to find where a module is installed you can use

```
pip show <module-name>
python3 -m pip show <module-name>
```

What happens when I have multiple Python and pip versions?

pip is usually related to Python 2

pip3 is usually related to Python 3

If we also have Python 3.6 installed then we may get confused on what is installed where

to make sure the right package is installed for the right version of python run pip via python:

Linux/Mac OS:

```
python3.7 -m pip <pip arguments>
python3.6 -m pip <pip arguments>
```

Windows:

```
py -3.7 -m pip <pip arguments>
py -3.6 -m pip <pip arguments>
```

this all depends on what is installed and what is in your path ↓

example: If you have Python 3.6 and Python 3.7 and want to install tensorflow for Python3.6 you would run `python3.6 -m pip install tensorflow`. Running `pip3 install tensorflow` would in my case install tensorflow for Python3.7

Useful commands:	explanation
pip install <package name>	
pip uninstall <package name>	
pip list	lists all installed packages and their version numbers
pip list --outdated	shows packages that are outdated and the version number of the most recent release
pip show <package name>	details the package, where it is installed, version, requirements

Managing Modules and their Versions for different projects: virtualenv

(virtualenv.pypa.io/en/stable)

pip installs modules/packages globally

if one applications depends on a given version and another app. depends on some other version things will crash

virtualenv creates an environment with its own folders for installing modules and does not share/depend on other folders

it also keeps track of the python version you want

Installing virtualenv:

choose the python version, here we will use python 3.6

```
python3.6 -m pip install virtualenv
```

check that you are running the correct version

```
which virtualenv
```

should be associated with Python 3.6

If it is not we just need to be careful to let virtualenv know what Python version we want

Using virtualenv:

virtualenv ENV ← or virtual --python=python3.6 ENV

creates a folder with name and inside that folder adds:

ENV/lib/

ENV/include/

ENV/lib/python3.6/site-packages/ → packages will be installed here

ENV/bin/ → has all executables, including a new python and pip.
↳ Windows uses the folder ENV\Scripts\

now we cd into the project folder and run the activate script:

cd ENV

MacOS/Linux } source bin/activate → this changes the PATH so that the first folder in it is ENV/bin/

Windows } .\Scripts\activate → check PATH with $Env:Path

if you need to undo the changes you can run

All OS's } deactivate

also the changes are only for your current terminal session, if you close it they will go away

on older Windows you may need to change the execution policy of scripts by:

Set-ExecutionPolicy AllSigned

or Set-ExecutionPolicy RemoteSigned

to remove ENV first `deactivate` and then remove the folder

If you have multiple Pythons installed and you want to specify the version you can do so when creating the environment:

```
virtualenv --python=python3.7 ENV
```

- `virtualenv` ↑ this was installed with Python 3.6
- `python3.7` ↑ name of the executable that can be found in the PATH
- `ENV` ↑ folder name (new folder is created)

Why is this useful?

We can create a folder for a specific python version and packages that work with it

If we want to create a project with a specific version then we just need to activate the script from that folder

Now:

```
virtualenv --python=python3.6 HFFE
cd HFFE
source bin/activate
```

and now we can all use python/pip and not have to worry about PATH and other complications

After activating, let's install the packages we want:

```
pip install numpy
pip install matplotlib
pip install jupyter
pip install tensorflow
```

Inside python we can import any of these modules.

Observations on Python versions:

version numbers usually follow: x.y.z

- x: major version
- y: minor changes
- z: bug fixes

sometimes you will see a 3.6.7rc or 3.6.7a or 3.6.7b

- rc: release candidate
- a: alpha
- b: beta

more about versioning on PEP440