**Contents**

[Python Package Manager 2](#_Toc28963916)

[Virtualenv and why you should use virtual environment? 3](#_Toc28963917)

[Virtual Environment with pip 3](#_Toc28963918)

[virtualenv 3](#_Toc28963919)

[pipenv 6](#_Toc28963920)

[Using pipenv 6](#_Toc28963921)

# Python Package Manager

if you type pip help it will show all commands you can use in pip.

we can type pip help install and it will show us the options for install.

* This is how you search a package:

pip search Pympler

* to install a package

pip install Pympler

* to list installed packages

pip uninstall Pympler

* how to know if a version installed is the latest version of the package

pip list -o # or

pip list --outdated

* to upgrade a package

pip install -U Pympler

* If there are hundreds or tens of **outdated** packages in your requirements.txt and you want to update all of them then here is a solution for that:

pip list --outdated --format=freeze | grep -v '^\-e' | cut -d = -f 1 | xargs -n1 pip install -U

<https://stackoverflow.com/questions/2720014/how-to-upgrade-all-python-packages-with-pip>

* if you want to give list of packages you are currently using to someone you can do

pip freeze

This will list all of packages installed in your project

Now if you want to export the name and version of those packages in a separate file you can do

pip freeze > requirements.txt

* how to install packages from requirements.txt

pip install -r requirements.txt

-r here means that hey we are using the requirements file.

Once you hit enter this will install all those packages with their exact version stated in requirements file.

# Virtualenv and why you should use virtual environment?

Before you go any further, make sure you have Python and that it’s available from your command line. You can check this by simply running:

$ python --version

Additionally, you’ll need to make sure you have [pip](https://pypi.org/project/pip/) available. You can check this by running:

$ pip --version

# Virtual Environment with pip

You can either work with one of these environments:

1. default python environment
2. python virtual environment

but it is always a good idea to separate different projects in their own virtual environment. virtualenv is a way that you can separate different Python environments for different projects.

**Why virtual environment?**

A Virtual Environment is an isolated working copy of Python which allows you to work on a specific project without worry of affecting other projects.

For example, you can work on a project which requires Django 1.3 while also maintaining a project which requires Django 1.0.

# virtualenv

virtualenv is a tool to create isolated Python environments.

* You can install virtualenv in your machine using this code:

> pip install virtualenv

* Use the list command to list all the packages installed on your system:

> pip list

* Let’s use virtualenv to make few different python environments. First make a directory and keeps all of these at one place:

> mkdir Environments

> cd Environments

To make our first virtual environment

> virtualenv project1\_env

In order to active this new virtualenv

> project1\_env\Scripts\activate

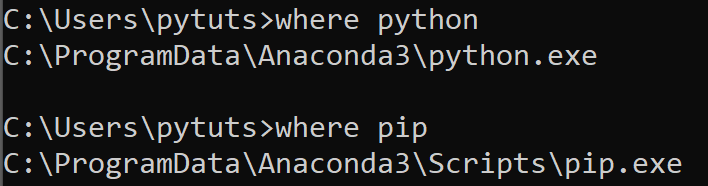
> source prject1\_env/bin/activate # in mac

# [How can I find where python is installed on Windows?](https://stackoverflow.com/questions/647515/how-can-i-find-where-python-is-installed-on-windows)

> where python  
C:\ProgramData\Anaconda3\python.exe

# [How can I find where pip is installed on Windows?](https://stackoverflow.com/questions/647515/how-can-i-find-where-python-is-installed-on-windows)

> where pip  
C:\ProgramData\Anaconda3\Scripts\pip.exe



* Now, let’s install a few packages into this new Python environment

> pip install numpy

> pip install pytz

> pip install psutil

* To export all these packages with their versions in another project

> pip freeze --local > requirement.txt

> type requirement

Numpy=1.16.12

Pip is a package manager for Python modules. The command pip freeze outputs all installed modules (including version numbers). The --local flag prevents Pip from printing globally installed packages in a virtual environment.

Usually, a Python program depends on other modules. You can put those required modules in a text file (requirements.txt by convention) so that other people can install those dependencies by running pip install -r requirements.txt. You can conveniently create such a file using pip freeze.

On a Linux machine, cat is used to output the contents of a file. You can use type on Windows.

The requirements format looks like this:

docutils==0.11

Jinja2==2.7.2

MarkupSafe==0.19

Pygments==1.6

Sphinx==1.2.2

* Type deactivate if you want to deactivate your current virtual environment.
* If you want to get rid of that virtual environment and delete it you can say

> rmdir /S newprj

* If you have multiple environment and just want to switch to another virtual environment from existing one, use:

> workon new\_venv\_name

* If you want to create a virtual environment using python version 2.6 you can do that:

virtualenv -p /usr/bin/python2.6 py26\_env

# pipenv

pipenv is basically combining the features of pip and virtualenv.

virtualenv is a specific environment for every project that we are making.

Before pipenv we used pip to handle package managements and virtualenv to handle virtual environments.

* Installing pipenv

pip install pipenv

Installing pipenv adds two files in your virtual environment.

1. pipfile - describes the environment of the project and how we can build it back from scratch. This is similar to requirements.txt file and it will be actually a replacement with requirements.txt
2. pipfile.lock - has exact hashes and versions of our current packages and dependencies. it has more information about our current environment.

# Using pipenv

* lets use pipenv and install ‘requests’ package:

pipenv install requests

Upon using pipenv, it does several things for us:

* It creates a virtual environment
* It installs stated package, in our case requests
* Now if you navigate to that directory. You will see that it created pipfile and pipfile.lock files for us

Pipfile looks like, and the format it is using is called **toml** which is key value pairs:

[[source]]

name = "pypi"

url = "https://pypi.org/simple" # site from wherein packages are installed

verify\_ssl = true

[dev-packages] # list of packages that you don’t need them in production

[packages] # installed packages’re listed here, in our case requests package without defining which version

requests = "\*" # asterisk here means the latest version

[requires]

python\_version = "3.7"

pipfile.lock is generated in virtual environment that has more information about our current environment, for example information about requests package that we just installed contains as below:

"requests": {

"hashes": [

"sha256:11e007a8a2aa0323f5a921e9e6a2d7e4e67d9877e85773fba9ba6419025cbeb4",

"sha256:9cf5292fcd0f598c671cfc1e0d7d1a7f13bb8085e9a590f48c010551dc6c4b31"

],

"index": "pypi",

"version": "==2.22.0"

},

* In order to activate this virtual environment

pipenv shell

* Alternatively, run a command inside the virtualenv with

pipenv run

* In order to deactivate the virtual environment

exit

* You can run a python file from within stated virtual environment even if it is deactivated with exit

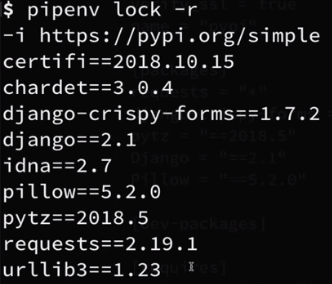
pipenv run python script.py

* Installing dependencies from requirements.txt file

pipenv install –r ../snippets/requirements.txt

* To list all dependencies of a project in pip use pip freeze, but in pipenv it is tad different:

pipenv lock –r



* To add dependencies that you don’t need them in production use this code and add them to dev packages. For example lets install pytest but as it is not going to be included in production we will install it as below:

pipenv install pytest --dev

Now, if you go to pipfile in your virtual environment directory then you should see this:

# code removed for brevity

[dev-packages]

pytest = "\*"

* lets go ahead and uninstall a dependency of our project

pipenv uninstall requests

* If you currently have **Python 3.7** installed in your virtual environment, and due to some unknown reasons you want to stick to previous version of **Python 3.6** , do follow these steps:

First change the version of python from pipfile

[requires]

python\_version = "3.7"

to

[requires]

python\_version = "3.6"

and execute the following command

pipenv --python 3.6

this command should delete your previous virtual environment and replace it with this new environment having python 3.6 installed.

* To remove a virtual environment completely

pipenv -rm

* the path to your virtual environment

$ pipenv --venv  
C:\Users\pytuts\.virtualenvs\pipenv-uptfPjJB

* Another good thing about pipenv is that you can check for security vulnerability of your currently installed dependencies using

pipenv check

* Assuming that you have django 2.1 installed in your virtual environment and you want to upgrade it to 2.1.2. Thence, follow these steps:

First replace below code from within pipfile

[packages]

Django = "==2.1"

with

[packages]

Django = "==2.1.2" # updated

from your Pipfile and navigate back to your current virtualenv directory and run this command:

pipenv install

this should replace Django2.1.2 with Django2.1

* To list packages along with their dependencies run pipenv graph, for example:

PS D:\SWDev\Python\py\pipenv> pipenv graph

Django==3.0.2

- asgiref [required: ~=3.2, installed: 3.2.3]

- pytz [required: Any, installed: 2019.3]

- sqlparse [required: >=0.2.2, installed: 0.3.0]

django-crispy-forms==1.8.1

requests==2.22.0

- certifi [required: >=2017.4.17, installed: 2019.11.28]

- chardet [required: >=3.0.2,<3.1.0, installed: 3.0.4]

- idna [required: >=2.5,<2.9, installed: 2.8]

- urllib3 [required: >=1.21.1,<1.26,!=1.25.1,!=1.25.0, installed: 1.25.7]

* To uninstall a package, like requests, from your virtual environment

pipenv uninstall requests

* How to add sensitive information of environment variables to our virtual environment.

Steps:

1. Create a .env file within your project directory and add those sensitive stuffs like following text in it

SECRET\_KEY = '23SDF8912NKJSDFSD912NKSNDFLKS'

1. to access that environment variables within python you can use the os module, so:

PS D:\SWDev\Python\py\pipenv> pipenv run python

Loading .env environment variables…

Python 3.7.1 (default, Dec 10 2018, 22:54:23) [MSC v.1915 64 bit (AMD64)] :: Anaconda, Inc. on win32

Type "help", "copyright", "credits" or "license" for more information.

>>> import os

>>> os.environ['SECRET\_KEY'] **# LearnedHardWay**: not (), it is [] ;p

'23SDF8912NKJSDFSD912NKSNDFLKS'

**Note**: Don’t add .env file in your GitHub repository, if you did you are most likely to add it within .gitignore file.

.env is only loaded from shell and run

* Check path from wherein pipenv is running

python

>>> import sys

**# python is using this location, the virtual envioronment created.**

>>> sys.executable   
quit()

* pipfile is not deterministic, but pipfile.lock is deterministic. Means that later has exact hashes and versions of our current packages and dependencies but former, pipfile , is asking us to grab the latest version of dependencies (this is good in development but not in production).

So, we should push the exact version of our packages in production. Now, you should prepare your pipfile.lock by saying:

pipenv lock

pipenv lock is used to create a Pipfile.lock, which declares **all** dependencies (and sub-dependencies) of your project, their latest available versions, and the current hashes for the downloaded files. This ensures repeatable, and most importantly *deterministic*, builds.

Once it is done and you’re in your server, you can run this command and it will install all of your project dependencies with stated version from pipfile.lock

pipenv install --ignore-pipfile

# [How to install dependencies from a copied pipfile inside a virtual environment?](https://stackoverflow.com/questions/52171593/how-to-install-dependencies-from-a-copied-pipfile-inside-a-virtual-environment)

The proper answer to this question is that pipenv install or pipenv install --dev (if there are dev dependencies) should be ran. This will install all the dependencies from Pipefile.

Putting the dependencies into a requirements.txt and then using pip will work but is not really necessary. The whole point of using pipenv for most people is to avoid the need to manage a requirements.txt or to use pip.

# Questions:

* lets use pipenv and install ‘requests’ package:
* In order to activate a virtual environment
* run a command inside the virtualenv with
* In order to deactivate the virtual environment
* How to run a python file from within stated virtual environment
* Installing dependencies from requirements.txt file
* To list all dependencies of a project in pip use pip freeze, but in pipenv it is tad different:
* To add dependencies that you don’t need them in production use this code and add them to dev packages. For example lets install pytest but as it is not going to be included in production we will install it as below:
* lets go ahead and uninstall a dependency of our project
* If you currently have **Python 3.7** installed in your virtual environment, and due to some reasons you want to stick to previous version of **Python 3.6** , do follow these steps:
* to remove a virtual environment completely
* to look at path of your virtual environment
* Another good thing about pipenv is that you can check for security vulnerability of your currently installed dependencies using
* assuming that you have django 2.1 installed in your virtual environment and you want to upgrade it to 2.1.2. Thence, follow these steps:
* To list packages along with their dependencies run
* How to add sensitive information of environment variables to our virtual environment.