Virtual Environments[¶](http://docs.python-guide.org/en/latest/dev/virtualenvs/#virtual-environments)

A Virtual Environment is a tool to keep the dependencies required by different projects in separate places, by creating virtual Python environments for them. It solves the “Project X depends on version 1.x but, Project Y needs 4.x” dilemma, and keeps your global site-packages directory clean and manageable.

For example, you can work on a project which requires Django 1.10 while also maintaining a project which requires Django 1.8.

virtualenv

[virtualenv](http://pypi.python.org/pypi/virtualenv) is a tool to create isolated Python environments. virtualenv creates a folder which contains all the necessary executables to use the packages that a Python project would need.

Install virtualenv via pip:

$ pip install virtualenv

Test your installation

$ virtualenv --version

Basic Usage

1. Create a virtual environment for a project:

$ cd my\_project\_folder

$ virtualenv my\_project

virtualenv my\_project will create a folder in the current directory which will contain the Python executable files, and a copy of the pip library which you can use to install other packages. The name of the virtual environment (in this case, it was my\_project) can be anything; omitting the name will place the files in the current directory instead.

This creates a copy of Python in whichever directory you ran the command in, placing it in a folder named my\_project.

You can also use the Python interpreter of your choice (like python2.7).

$ virtualenv -p /usr/bin/python2.7 my\_project

or change the interpreter globally with an env variable in ~/.bashrc:

$ export VIRTUALENVWRAPPER\_PYTHON=/usr/bin/python2.7

1. To begin using the virtual environment, it needs to be activated:

$ source my\_project/bin/activate

The name of the current virtual environment will now appear on the left of the prompt (e.g. (my\_project)Your-Computer:your\_project UserName$) to let you know that it’s active. From now on, any package that you install using pip will be placed in the my\_project folder, isolated from the global Python installation.

Install packages as usual, for example:

$ pip install requests

1. If you are done working in the virtual environment for the moment, you can deactivate it:

$ deactivate

This puts you back to the system’s default Python interpreter with all its installed libraries.

To delete a virtual environment, just delete its folder. (In this case, it would be rm -rfmy\_project.)

After a while, though, you might end up with a lot of virtual environments littered across your system, and its possible you’ll forget their names or where they were placed.

Other Notes

Running virtualenv with the option --no-site-packages will not include the packages that are installed globally. This can be useful for keeping the package list clean in case it needs to be accessed later. [This is the default behavior for virtualenv 1.7 and later.]

In order to keep your environment consistent, it’s a good idea to “freeze” the current state of the environment packages. To do this, run

$ pip freeze > requirements.txt

This will create a requirements.txt file, which contains a simple list of all the packages in the current environment, and their respective versions. You can see the list of installed packages without the requirements format using “pip list”. Later it will be easier for a different developer (or you, if you need to re-create the environment) to install the same packages using the same versions:

$ pip install -r requirements.txt

This can help ensure consistency across installations, across deployments, and across developers.

Lastly, remember to exclude the virtual environment folder from source control by adding it to the ignore list.

virtualenvwrapper

[virtualenvwrapper](https://virtualenvwrapper.readthedocs.io/en/latest/index.html) provides a set of commands which makes working with virtual environments much more pleasant. It also places all your virtual environments in one place.

To install (make sure **virtualenv** is already installed):

$ pip install virtualenvwrapper

$ export WORKON\_HOME=~/Envs

$ source /usr/local/bin/virtualenvwrapper.sh

([Full virtualenvwrapper install instructions](https://virtualenvwrapper.readthedocs.io/en/latest/install.html).)

For Windows, you can use the [virtualenvwrapper-win](https://github.com/davidmarble/virtualenvwrapper-win/).

To install (make sure **virtualenv** is already installed):

$ pip install virtualenvwrapper-win

In Windows, the default path for WORKON\_HOME is %USERPROFILE%Envs

Basic Usage

1. Create a virtual environment:

$ mkvirtualenv my\_project

This creates the my\_project folder inside ~/Envs.

1. Work on a virtual environment:

$ workon my\_project

Alternatively, you can make a project, which creates the virtual environment, and also a project directory inside $PROJECT\_HOME, which is cd -ed into when you workon myproject.

$ mkproject myproject

**virtualenvwrapper** provides tab-completion on environment names. It really helps when you have a lot of environments and have trouble remembering their names.

workon also deactivates whatever environment you are currently in, so you can quickly switch between environments.

1. Deactivating is still the same:

$ deactivate

1. To delete:

$ rmvirtualenv venv

Other useful commands

lsvirtualenv

List all of the environments.

cdvirtualenv

Navigate into the directory of the currently activated virtual environment, so you can browse its site-packages, for example.

cdsitepackages

Like the above, but directly into site-packages directory.

lssitepackages

Shows contents of site-packages directory.

[Full list of virtualenvwrapper commands](https://virtualenvwrapper.readthedocs.io/en/latest/command_ref.html).

virtualenv-burrito

With [virtualenv-burrito](https://github.com/brainsik/virtualenv-burrito), you can have a working virtualenv + virtualenvwrapper environment in a single command.

autoenv

When you cd into a directory containing a .env, [autoenv](https://github.com/kennethreitz/autoenv) automagically activates the environment.

Install it on Mac OS X using brew:

$ brew install autoenv

And on Linux:

$ git clone git://github.com/kennethreitz/autoenv.git ~/.autoenv

$ echo 'source ~/.autoenv/activate.sh' >> ~/.bashrc