Python Guide Documentation

Release 0.0.1

Kenneth Reitz

August 17, 2017

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 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
```

2. 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
```

3. 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 -rf my_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 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.)

For Windows, you can use the 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`.

2. 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.

3. Deactivating is still the same:

```
$ deactivate
```

4. 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.

virtualenv-burrito

With 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 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
```

Further Configuration of Pip and Virtualenv

Requiring an active virtual environment for `pip`

By now it should be clear that using virtual environments is a great way to keep your development environment clean and keeping different projects' requirements separate.

When you start working on many different projects, it can be hard to remember to activate the related virtual environment when you come back to a specific project. As a result of this, it is very easy to install packages globally while thinking that you are actually installing the package for the virtual environment of the project. Over time this can result in a messy global package list.

In order to make sure that you install packages to your active virtual environment when you use `pip install`, consider adding the following line to your `~/.bashrc` file:

```
export PIP_REQUIRE_VIRTUALENV=true
```

After saving this change and sourcing the `~/.bashrc` file with `source ~/.bashrc`, pip will no longer let you install packages if you are not in a virtual environment. If you try to use `pip install` outside of a virtual environment pip will gently remind you that an activated virtual environment is needed to install packages.

```
$ pip install requests
Could not find an activated virtualenv (required).
```

You can also do this configuration by editing your `pip.conf` or `pip.ini` file. `pip.conf` is used by Unix and Mac OS X operating systems and it can be found at:

```
$HOME/.pip/pip.conf
```

Similarly, the `pip.ini` file is used by Windows operating systems and it can be found at:

```
%HOME%\pip\pip.ini
```

If you don't have a `pip.conf` or `pip.ini` file at these locations, you can create a new file with the correct name for your operating system.

If you already have a configuration file, just add the following line under the `[global]` settings to require an active virtual environment:

```
require-virtualenv = true
```

If you did not have a configuration file, you will need to create a new one and add the following lines to this new file:

```
[global]
require-virtualenv = true
```

You will of course need to install some packages globally (usually ones that you use across different projects consistently) and this can be accomplished by adding the following to your `~/.bashrc` file:

```
gpip() {
    PIP_REQUIRE_VIRTUALENV="" pip "$@"
}
```

After saving the changes and sourcing your `~/.bashrc` file you can now install packages globally by running `gpip install`. You can change the name of the function to anything you like, just keep in mind that you will have to use that name when trying to install packages globally with pip.

Caching packages for future use

Every developer has preferred libraries and when you are working on a lot of different projects, you are bound to have some overlap between the libraries that you use. For example, you may be using the `requests` library in a lot of different projects.

It is surely unnecessary to re-download the same packages/libraries each time you start working on a new project (and in a new virtual environment as a result). Fortunately, you can configure pip in such a way that it tries to reuse already installed packages.

On UNIX systems, you can add the following line to your `.bashrc` or `.bash_profile` file.

```
export PIP_DOWNLOAD_CACHE=$HOME/.pip/cache
```

You can set the path to anywhere you like (as long as you have write access). After adding this line, `source` your `.bashrc` (or `.bash_profile`) file and you will be all set.

Another way of doing the same configuration is via the `pip.conf` or `pip.ini` files, depending on your system. If you are on Windows, you can add the following line to your `pip.ini` file under `[global]` settings:

```
download-cache = %HOME%\pip\cache
```

Similarly, on UNIX systems you should simply add the following line to your `pip.conf` file under `[global]` settings:

```
download-cache = $HOME/.pip/cache
```

Even though you can use any path you like to store your cache, it is recommended that you create a new folder *in* the folder where your `pip.conf` or `pip.ini` file lives. If you don't trust yourself with all of this path voodoo, just use the values provided here and you will be fine.