VirtualEnv

1. VirtualEnv and VirtualEnvWrapper

VirtualEnv allows us to create an isolated working copy of Python which allows you to work on a specific project without worry of affecting other projects. It enables multiple side-by-side installations of Python, one for each project. It doesn't actually install separate copies of Python, but it does provide a clever way to keep different project environments isolated.

1.1 Install VirtualEnv and VirtualEnvWrapper

We will be installing our Django projects in their own virtual environments to isolate the requirements for each. To do this, we will be installing virtualenv, which can create Python virtual environments, and virtualenvwrapper, which adds some usability improvements to the virtualenv work flow.

We will be installing both of these components using pip, the Python package manager. We can install this utility from the Ubuntu repositories. If you are building your Django projects with Python 2, type:

```
sudo apt-get update

sudo apt-get install python-pip
```

If you are using Python 3, type:

```
sudo apt-get update

sudo apt-get install python3-pip
```

Now that you have pip installed, we can install virtualenv and virtualenvwrapper globally. If you are using Python 2, type:

```
sudo pip install virtualenv virtualenvwrapper
```

If you are using Python 3, type:

```
sudo pip3 install virtualenv virtualenvwrapper

sudo apt-get install python3-venv
```

1.2 Configure VirtualEnv and VirtualEnvWrapper

With these components installed, we can now configure our shell with the information it needs to work with the virtualenvwrapper script. Our virtual environments will all be placed within a directory in our home folder called Env for easy access. This is configured through an environmental variable called WORKON_HOME. We can add this to our shell initialization script and can source the virtual environment wrapper script.

If you are using Python 3 and the pip3 command, you will have to add an additional line to your shell initialization script as well:

```
echo "export VIRTUALENVWRAPPER_PYTHON=/usr/bin/python3" >> ~/.bashrc
```

Regardless of which version of Python you are using, you need to run the following commands:

```
echo "export WORKON_HOME=~/Env" >> ~/.bashrc

echo "source /usr/local/bin/virtualenvwrapper.sh" >> ~/.bashrc
```

Now, source your shell initialization script so that you can use this functionality in your current session:

```
source ~/.bashrc
```

You should now have directory called Env in your home folder which will hold virtual environment information.

2. Create a Virtual Enviroment in Python 3.3+

Create a directory that will “hold” our virtualen environment at your home directory:

```
mkdir virtualenv-iquote
```

Creation of virtual environments is done by executing the command **venv**:

```
python3 -m venv /path/to/new/virtual/environment
```

Running this command creates the target directory (creating any parent directories that don’t exist already) and places a pyvenv.cfg file in it with a home key pointing to the Python installation from which the command was run. It also creates a bin (or Scripts on Windows) subdirectory containing a copy of the python binary (or binaries, in the case of Windows). It also creates an (initially empty) lib/pythonX.Y/site-packages subdirectory (on Windows, this is Lib\site-packages).

On Windows, invoke the venv command as follows:

```
c:\>c:\Python35\python -m venv c:\path\to\myenv
```

Alternatively, if you configured the PATH and PATHEXT variables for your Python installation:

```
c:\>python -m venv myenv c:\path\to\myenv
```