Getting Started with virtualenv and virtualenvwrapper in Python

Posted At : July 24, 2012 9:52 AM | Posted By : Bob Silverberg
Related Categories: MozWebQA, Python

There are a couple of tools which can be extremely useful when developing with Python on your local system that I would encourage you to try. These are virtualenv and virtualenvwrapper, and this post will introduce you to them, including instructions for their installation and use.

What is virtualenv?

virtualenv is a tool that allows you to create isolated Python environments, which can be quite helpful when you have different projects with differing requirements. As this is the case with many Mozilla Web QA projects it can be indispensable when working on those.

What is virtualenvwrapper?

virtualenvwrapper is just that, a wrapper utility around virtualenv that makes it even easier to work with. I admit that I have never used virtualenv without virtualenvwrapper, and I do not intend to. For that reason this post will only cover working with virtualenv via virtualenvwrapper. Note also that virtualenvwrapper is a set of shell functions that are guaranteed to work in the following shells:

- bash
- ksh
- zsh

It may run under other shells, and there is a Windows version available called virtualenvwrapper-win

Installation

Both virtualenv and virtualenvwrapper can be installed via pip. Install virtualenv first and then virtualenvwrapper. Use the following commands to install them:

```
pip install virtualenv

        pip install virtualenvwrapper
```

Configuration

In order to use virtualenvwrapper you should add two lines to your shell startup file (e.g., .bash_profile):

```
export WORKON_HOME=$HOME/.virtualenvs

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

The first line tells virtualenvwrapper where to store the virtualenvs that will be created and used. The example above stores them in a folder called *.virtualenvs* inside your home folder. The first line runs the shell script to set up the virtualenvwrapper commands and should point to the location where virtualenvwrapper was installed.

Using virtualenvwrapper to Manage and Work with Virtual Environments

There are a lot of commands available with virtualenvwrapper, all of which are well documented. In my experience, the following are the commands you will use most often:

- *mkvirtualenv* - used to create a new virtual environment. When you create a new environment it automatically becomes the active environment.
- *rmvirtualenv* - used to remove an existing virtual environment. The environment must be deactivated (see below) before it can be removed.
- *workon* - used to activate a virtual environment. Will also list all existing virtual environments if no argument is passed.
- *deactivate* - used to deactivate the currently active virtual environment. Note that *workon* will automatically deactivate the current environment before activating a new one.

Here's an example of how one might use virtualenvwrapper to set up and configure a virtual environment for running tests for a Mozilla Web QA project (marketplace-tests). This code assumes that you already have the Git repo cloned to your local machine and you have navigated to the root folder of the project.

```
mkvirtualenv marketplace-tests

        pip install -Ur requirements.txt
```

The first line will create and activate a new virtual environment for marketplace-tests, while the second line will install all of the required packages into this virtual environment. Now, whenever you want to work on marketplace-tests you can simply type the command:

```
workon marketplace-tests
```

Using virtualenv and PyCharm

The instructions above are specific to using virtualenv when running Python from the command line. You can also use your virtual environments when running and debugging Python code from within PyCharm. First you have to add your virtual environments to PyCharm's list of Python interpreters:

1. Open the *Preferences* (*Settings* on Windows) dialog and choose **Project Interpreter**.
2. Click on the **Configure Interpreters** link, which will open up the *Python Interpreters* dialog. Here you will see all of the interpreters currently configured for PyCharm.
3. Click the **+** button at the bottom of the list and choose the path to the interpreter in your virtual environment. In my experience sometimes PyCharm is able to find and list these automatically, and other times you have to choose the **Local...** item and browse for the path yourself. After choosing the path PyCharm will do some setup, displaying a progress indicator and will finally ask you whether you want to *set this interpreter as Project Interpreter?* Generally you are adding an interpreter precisely because you *do* want to set it as the interpreter for the current project, in which case you would answer **Yes**
4. Your interpreter will now appear in the list of Python interpreters.

If ever you want to change the interpreter for a given project just access the *Project Interpreter* dialog again and simply select a different interpreter from the *Project Interpreter* select list. This option is even available to be changed for individual run configurations, as described in the previous post on Setting Up PyCharm to Run MozWebQA Tests:

1. Type **ctrl + alt + R** to open the *Run* configuration selector.
2. Type **0**, or choose the first option (*Edit configurations...*) from the select list.
3. In the *Run* dialog, beside *Interpreter* > *Python Interpreter*, choose your interpreter from the list.
4. Click **Run**.