Virtualenv

Mailing list | Issues | Github | PyPI | User IRC: #pypa Dev IRC: #pypa-dev

Introduction

`virtualenv` is a tool to create isolated Python environments.

The basic problem being addressed is one of dependencies and versions, and indirectly permissions. Imagine you have an application that needs version 1 of LibFoo, but another application requires version 2. How can you use both these applications? If you install everything into `/usr/lib/python2.7/site-packages` (or whatever your platform's standard location is), it's easy to end up in a situation where you unintentionally upgrade an application that shouldn't be upgraded.

Or more generally, what if you want to install an application *and leave it be*? If an application works, any change in its libraries or the versions of those libraries can break the application.

Also, what if you can't install packages into the global `site-packages` directory? For instance, on a shared host.

In all these cases, `virtualenv` can help you. It creates an environment that has its own installation directories, that doesn't share libraries with other virtualenv environments (and optionally doesn't access the globally installed libraries either).

- 1.9.1 (2013-03-08)
- 1.9 (2013-03-07)
- 1.8.4 (2012-11-25)
- 1.8.3 (2012-11-21)
- 1.8.2 (2012-09-06)
- 1.8.1 (2012-09-03)
- 1.8 (2012-09-01)
- 1.7.2 (2012-06-22)
- 1.7.1.2 (2012-02-17)
- 1.7.1.1 (2012-02-16)
- 1.7.1 (2012-02-16)
- 1.7 (2011-11-30)
- 1.6.4 (2011-07-21)
- 1.6.3 (2011-07-16)
- 1.6.2 (2011-07-16)
- 1.6.1 (2011-04-30)
- 1.6
- 1.5.2
- 1.5.1
- 1.5
- 1.4.9
- 1.4.8
- 1.4.7
- 1.4.6
- 1.4.5
- 1.4.4
- 1.4.3
- 1.4.2
- 1.4.1
- 1.4
- 1.3.4
- 1.3.3
- 1.3.2
- 1.3.1
- 1.3
- 1.2
- 1.1.1
- 1.1
- 1.0
- 0.9.2
- 0.9.1
- 0.9

- 0.8.4
- 0.8.3
- 0.8.2
- 0.8.1
- 0.8

Warning

Python bugfix releases 2.6.8, 2.7.3, 3.1.5 and 3.2.3 include a change that will cause "import random" to fail with "cannot import name urandom" on any virtualenv created on a Unix host with an earlier release of Python 2.6/2.7/3.1/3.2, if the underlying system Python is upgraded. This is due to the fact that a virtualenv uses the system Python's standard library but contains its own copy of the Python interpreter, so an upgrade to the system Python results in a mismatch between the version of the Python interpreter and the version of the standard library. It can be fixed by removing `$ENV/bin/python` and re-running virtualenv on the same target directory with the upgraded Python.

Other Documentation and Links

- Blog announcement of virtualenv.
- James Gardner has written a tutorial on using virtualenv with Pylons.
- Chris Perkins created a showmedo video including virtualenv.
- Doug Hellmann's virtualenvwrapper is a useful set of scripts to make your workflow with many virtualenvs even easier. His initial blog post on it. He also wrote an example of using virtualenv to try IPython.
- Pew is another wrapper for virtualenv that makes use of a different activation technique.
- Using virtualenv with mod_wsgi.
- virtualenv commands for some more workflow-related tools around virtualenv.
- PyCon US 2011 talk: Reverse-engineering Ian Bicking's brain: inside pip and virtualenv. By the end of the talk, you'll have a good idea exactly how pip and virtualenv do their magic, and where to go looking in the source for particular behaviors or bug fixes.

Compare & Contrast with Alternatives

There are several alternatives that create isolated environments:

- `workingenv` (which I do not suggest you use anymore) is the predecessor to this library. It used the main Python interpreter, but relied on setting `$PYTHONPATH` to activate the environment. This causes problems when running Python scripts that aren't part of the environment (e.g., a globally installed `hg` or `bzr`). It also conflicted a lot with Setuptools.

- virtual-python is also a predecessor to this library. It uses only symlinks, so it couldn't work on Windows. It also symlinks over the *entire* standard library and global `site-packages`. As a result, it won't see new additions to the global `site-packages`.

  This script only symlinks a small portion of the standard library into the environment, and so on Windows it is feasible to simply copy these files over. Also, it creates a new/empty `site-packages` and also adds the global `site-packages` to the path, so updates are tracked separately. This script also installs Setuptools automatically, saving a step and avoiding the need for network access.
- zc.buildout doesn't create an isolated Python environment in the same style, but achieves similar results through a declarative config file that sets up scripts with very particular packages. As a declarative system, it is somewhat easier to repeat and manage, but more difficult to experiment with. `zc.buildout` includes the ability to setup non-Python systems (e.g., a database server or an Apache instance).

I *strongly* recommend anyone doing application development or deployment use one of these tools.