venv — Create Virtual Environments

Purpose: Create isolated installation and execution contexts.

Python virtual environments, managed by venv, are set up for installing packages and running programs in a way that isolates them from other packages installed on the rest of the system. Because each environment has its own interpreter executable and directory for installing packages, it is easy to create environments configured with various combinations of Python and package versions all on the same computer.

Creating Environments

The primary command line interface to venv relies on Python's ability to run a "main" function in a module using the `-m` option.

```
$ python3 -m venv /tmp/demoenv
```

A separate pyvenv command line application may be installed, depending on how the Python interpreter was built and packaged. The following command has the same effect as the previous example.

```
$ pyvenv /tmp/demoenv
```

Using `-m venv` is preferred because it requires explicitly selecting a Python interpreter, so there can be no confusion about the version number or import path assocated with the resulting virtual environment.

Contents of a Virtual Environment

Each virtual environment contains a `bin` directory, where the local interpreter and any executable scripts are installed, an `include` directory for files related to building C extensions, and a `lib` directory, with a separate `site-packages` location for installing packages.

```
$ ls -F /tmp/demoenv

bin/
include/
lib/
pyvenv.cfg
```

The default `bin` directory contains "activation" scripts for several Unix shell variants. These can be used to install the virtual environment on the shell's search path to ensure the shell picks up programs installed in the environment. It's not necessary to activate an environment to use programs installed into it, but it can be more convenient.

```
$ ls -F /tmp/demoenv/bin

activate
activate.csh
activate.fish
easy_install*
easy_install-3.6*
pip*
pip3*
pip3.6*
python@
python3@
```

On platforms that support them, symbolic links are used rather than copying the executables like the Python interpreter. In this environment, `pip` is installed as a local copy but the interpreter is a symlink.

Finally, the environment includes a `pyvenv.cfg` file with settings describing how the environment is configured and should behave. The `home` variable points to the location of the Python interpreter where venv was run to create the environment. `include-system-site-packages` is a boolean indicating whether or not the packages installed outside of the virtual environment, at the system level, should be visible inside the virtual environment. And `version` is the Python version used to create the environment.

```
# pyvenv.cfg

home = /Library/Frameworks/Python.framework/Versions/3.6/bin
include-system-site-packages = false
version = 3.6.4
```

A virtual environment is more useful with tools like `pip` and `setuptools` available to install other packages, so pyvenv installs them by default. To create a bare environment without these tools, pass `--without-pip` on the command line.

Using Virtual Environments

Virtual environments are commonly used to run different versions of programs or to test a given version of a program with different versions of its dependencies. For example, before upgrading from one version of Sphinx to another, it is useful to test the input documentation files using both the old and new versions. To start, create two virtual environments.

```
$ python3 -m venv /tmp/sphinx1
$ python3 -m venv /tmp/sphinx2
```

Then install the versions of the tools to test.

```
$ /tmp/sphinx1/bin/pip install Sphinx==1.3.6

Collecting Sphinx==1.3.6
  Using cached Sphinx-1.3.6-py2.py3-none-any.whl
Collecting Pygments>=2.0 (from Sphinx==1.3.6)
  Using cached Pygments-2.2.0-py2.py3-none-any.whl
Collecting sphinx-rtd-theme<2.0,>=0.1 (from Sphinx==1.3.6)
  Using cached sphinx_rtd_theme-0.2.4-py2.py3-none-any.whl
Collecting babel!=2.0,>=1.3 (from Sphinx==1.3.6)
  Using cached Babel-2.5.3-py2.py3-none-any.whl
Collecting alabaster<0.8,>=0.7 (from Sphinx==1.3.6)
  Using cached alabaster-0.7.10-py2.py3-none-any.whl
Collecting Jinja2>=2.3 (from Sphinx==1.3.6)
  Using cached Jinja2-2.10-py2.py3-none-any.whl
Collecting docutils>=0.11 (from Sphinx==1.3.6)
  Using cached docutils-0.14-py3-none-any.whl
Collecting snowballstemmer>=1.1 (from Sphinx==1.3.6)
  Using cached snowballstemmer-1.2.1-py2.py3-none-any.whl
Collecting six>=1.4 (from Sphinx==1.3.6)
  Using cached six-1.11.0-py2.py3-none-any.whl
Collecting pytz>=0a (from babel!=2.0,>=1.3->Sphinx==1.3.6)
  Using cached pytz-2018.3-py2.py3-none-any.whl
Collecting MarkupSafe>=0.23 (from Jinja2>=2.3->Sphinx==1.3.6)
  Using cached MarkupSafe-1.0.tar.gz
Installing collected packages: Pygments, sphinx-rtd-theme, pytz,
babel, alabaster, MarkupSafe, Jinja2, docutils, snowballstemmer,
six, Sphinx
  Running setup.py install for MarkupSafe: started
    Running setup.py install for MarkupSafe: finished with
status 'done'
Successfully installed Jinja2-2.10 MarkupSafe-1.0 Pygments-2.2.0
Sphinx-1.3.6 alabaster-0.7.10 babel-2.5.3 docutils-0.14
pytz-2018.3 six-1.11.0 snowballstemmer-1.2.1 sphinx-rtd-
theme-0.2.4

$ /tmp/sphinx2/bin/pip install Sphinx==1.4.4

Collecting Sphinx==1.4.4
  Using cached Sphinx-1.4.4-py2.py3-none-any.whl
Collecting imagesize (from Sphinx==1.4.4)
  Using cached imagesize-1.0.0-py2.py3-none-any.whl
Collecting Pygments>=2.0 (from Sphinx==1.4.4)
  Using cached Pygments-2.2.0-py2.py3-none-any.whl
Collecting snowballstemmer>=1.1 (from Sphinx==1.4.4)
  Using cached snowballstemmer-1.2.1-py2.py3-none-any.whl
Collecting alabaster<0.8,>=0.7 (from Sphinx==1.4.4)
  Using cached alabaster-0.7.10-py2.py3-none-any.whl
Collecting Jinja2>=2.3 (from Sphinx==1.4.4)
```

```
  Using cached Jinja2-2.10-py2.py3-none-any.whl
Collecting docutils>=0.11 (from Sphinx==1.4.4)
  Using cached docutils-0.14-py3-none-any.whl
Collecting babel!=2.0,>=1.3 (from Sphinx==1.4.4)
  Using cached Babel-2.5.3-py2.py3-none-any.whl
Collecting six>=1.4 (from Sphinx==1.4.4)
  Using cached six-1.11.0-py2.py3-none-any.whl
Collecting MarkupSafe>=0.23 (from Jinja2>=2.3->Sphinx==1.4.4)
  Using cached MarkupSafe-1.0.tar.gz
Collecting pytz>=0a (from babel!=2.0,>=1.3->Sphinx==1.4.4)
  Using cached pytz-2018.3-py2.py3-none-any.whl
Installing collected packages: imagesize, Pygments,
snowballstemmer, alabaster, MarkupSafe, Jinja2, docutils, pytz,
babel, six, Sphinx
  Running setup.py install for MarkupSafe: started
    Running setup.py install for MarkupSafe: finished with
status 'done'
Successfully installed Jinja2-2.10 MarkupSafe-1.0 Pygments-2.2.0
Sphinx-1.4.4 alabaster-0.7.10 babel-2.5.3 docutils-0.14
imagesize-1.0.0 pytz-2018.3 six-1.11.0 snowballstemmer-1.2.1
```

Then it is possible to run the different versions of Sphinx from the virtual environments separately, to test them with the same input files.

```
$ /tmp/sphinx1/bin/sphinx-build --version

Sphinx (sphinx-build) 1.3.6

$ /tmp/sphinx2/bin/sphinx-build --version

Sphinx (sphinx-build) 1.4.4
```

See also

- Standard library documentation for venv
- **PEP 405** – Python Virtual Environments
- virtualenv – A version of Python virtual environments that works for Python 2 and 3.
- virtualenvwrapper – A set of shell wrappers for virtualenv to make it easier to manage a large number of environments.
- Sphinx – Tool for converting reStructuredText input files to HTML, LaTeX, and other formats for consumption.

pyclbr — Class Browser

ensurepip — Install the Python Package Installer

Quick Links

Creating Environments
Contents of a Virtual Environment
Using Virtual Environments

This page was last updated 2018-03-18.

Navigation

pyclbr — Class Browser
ensurepip — Install the Python Package Installer

The output from all the example programs from PyMOTW-3 has been generated with Python 3.7.1, unless otherwise noted. Some of the features described here may not be available in earlier versions of Python.

Looking for examples for Python 2?

This Site

Module Index
Index

Other Writing

Blog
The Python Standard Library By Example