A Brief Guide to Python Packaging

Christopher R. Genovese

16 Nov 2017

1 Installing Packages

1.1 Key Tools: pip and setuptools

For current versions of python, pip and setuptools are installed automatically. To upgrade to the latest version do the following.

Mac OS X and Linux:

```
pip install -U pip setuptools
```

Windows:

```
python -m pip install -U pip setuptools
```

1.2 Optional Tool: Virtual Environments

- virtualenv

  ```
  pip install virtualenv
  ```

- venv (python3 only)

  Available by default for python >= 3.3

1.3 Optional Tools: pipenv

pipenv is a dependency manager for python that provides a higher level tool than pip with some convenient features.

See the section on Pipenv at `http://docs.python-guide.org/en/latest/dev/virtualenvs/`.

1.4 Installing Packages with pip

- Install latest version: `pip install 'apackage'`
- Install specific version: `pip install 'apackage==2.1'`
- Install version within range: `pip install 'apackage>=2.1,<3'`
- Install version compatible with specific version: `pip install 'apackage~=2.1.2'`
- Upgrade package: `pip install --upgrade 'apackage'`
- Install list from a requirements file: `pip install -r requirements.txt`
- More on requirements files: `https://pip.pypa.io/en/latest/user_guide/#requirements-files`

1.5 Using Virtual Environments

Python virtual environments give isolated locations in which to install packags used by applications or packages you are developing.

For instance, suppose your application or package uses package foobar version 2. In your global package library, you might have foobar installed, but different applications might demand different versions, and changing the version to upgrade foobar could break your application. A virtual environment lets you build your application with its own version of the libraries it needs and even the python executables. You can thus leave it and use it as needed.

There are a variety of tools now available for managing virtual environments. Two are worth focusing on: `virtualenv` is a low-level tool, the first to be commonly used, and `venv` is a package included by default with python 3 that provides the same functionality without an additional install.

To create a virtual environment do one of:

```
virtualenv DIR
python3 -m venv DIR
```

You then *activate* the virtual environment with

```
source DIR/bin/activate
```

so that new installs (e.g., with pip) will go in the virtual environment. This DIR (replace "DIR" with any name you like) will also contain versions of the python and pip executables at least.

When you are done, type

```
DIR/bin/deactivate
```

and installs will once again go into your global or user environment.

See

- virtualenv documentation: `https://virtualenv.pypa.io/en/stable/`
- venv documentation: `https://docs.python.org/3/library/venv.html`

for more details.

2 Creating Packages

A python package is, at heart, relatively straightforward: any directory with an `__init__.py` file in it is considered a package.

For example, a directory `package` containing files `__init__.py` and `module.py` is a package, and the latter is imported by writing `import package.module` in python. This will first look at `package/__init__.py` and execute its top-level statements, and then read `package/module.py` and execute its top-level statements.

It is fine to keep `__init__.py` empy. Indeed, if the modules and sub-modules in the package do not share any code, keeping the file empty is considered good practice. Note: the file `__init__.py` should be included even if it is empty, although in python 3.3 or greater the file is optional. See `http://ncoghlan-devs-python-notes.readthedocs.io/en/latest/python_concepts/import_traps.html` for some subtleties.

While the package structure can be simple, there is a recommended format that is commonly used in the python community:

```
README.rst
LICENSE
setup.py
requirements.txt
packagename/__init__.py
packagename/corecode.py
packagename/helpercode.py
docs/conf.py
docs/index.rst
tests/
```

where `packagename`, `corecode`, and `helpercode` are placeholders for whatever your package and code files are called.

See this git repository (`https://github.com/kennethreitz/samplemod`) for the structure and this post (`https://www.kennethreitz.org/essays/repository-structure-and-python`) for an explanation of the structure and contents.

See `https://packaging.python.org/tutorials/distributing-packages/` for another description.

3 Resources

- `https://packaging.python.org/`
- Hitchhiker's guide to Python `http://docs.python-guide.org/en/latest/`

  See in particular `http://docs.python-guide.org/en/latest/dev/virtualenvs/`.