Creating Python packages

All Versions

Examples

Introduction

Every package requires a setup.py file which describes the package.

Consider the following directory structure for a simple package:

```
+-- package_name
|       |
|       +-- __init__.py
|
+-- setup.py
```

The __init__.py contains only the line def foo(): return 100 .

The following setup.py will define the package:

```
from setuptools import setup


setup(
    name='package_name',                    # package name
    version='0.1',                          # version
    description='Package Description',      # short description
    url='http://example.com',               # package URL
    install_requires=[],                    # list of packages this package depends
                                            # on.
    packages=['package_name'],              # List of module names that installing
                                            # this package will provide.
)
```

virtualenv is great to test package installs without modifying your other Python environments:

```
$ virtualenv .virtualenv
...
$ source .virtualenv/bin/activate
$ python setup.py install
running install
...
Installed .../package_name-0.1-....egg
...
$ python
>>> import package_name
>>> package_name.foo()
100
```

Uploading to PyPI

Once your setup.py is fully functional (see Introduction), it is very easy to upload your package to PyPI .

Setup a .pypirc File

This file stores logins and passwords to authenticate your accounts. It is typically stored in your home directory.

```
# .pypirc file

[distutils]
index-servers =
  pypi
  pypitest

[pypi]
repository=https://pypi.python.org/pypi
username=your_username
password=your_password

[pypitest]
repository=https://testpypi.python.org/pypi
username=your_username
password=your_password
```

It is safer to use twine for uploading packages, so make sure that is installed.

```
$ pip install twine
```

Register and Upload to testpypi (optional)

Note : PyPI does not allow overwriting uploaded packages , so it is prudent to first test your deployment on a dedicated test server, e.g. testpypi. This option will be discussed. Consider a versioning scheme for your package prior to uploading such as calendar versioning or semantic versioning .

Either log in, or create a new account at testpypi . Registration is only required the first time, although registering more than once is not harmful.

```
$ python setup.py register -r pypitest
```

While in the root directory of your package:

```
$ twine upload dist/* -r pypitest
```

Your package should now be accessible through your account.

Testing

Make a test virtual environment. Try to pip install your package from either testpypi or PyPI.

```
# Using virtualenv
$ mkdir testenv
$ cd testenv
$ virtualenv .virtualenv
...
$ source .virtualenv/bin/activate
# Test from testpypi
(.virtualenv)  pip install --verbose --extra-index-url https://testpypi.python.org/pypi package_
...
# Or test from PyPI
(.virtualenv) $ pip install package_name
...

(.virtualenv) $ python
Python 3.5.1 (default, Jan 27 2016, 19:16:39)
[GCC 4.2.1 Compatible Apple LLVM 7.0.2 (clang-700.1.81)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import package_name
>>> package_name.foo()
100
```

If successful, your package is least importable. You might consider testing your API as well before your final upload to PyPI. If you package failed during testing, do not worry. You can still fix it, re-upload to testpypi and test again.

Register and Upload to PyPI

Make sure twine is installed:

```
$ pip install twine
```

Either log in, or create a new account at PyPI .

```
$ python setup.py register -r pypi
$ twine upload dist/*
```

That's it! Your package is now live .

If you discover a bug, simply upload a new version of your package.

Documentation

Don't forget to include at least some kind of documentation for your package. PyPi takes as the default formatting language reStructuredText .

Readme

If your package doesn't have a big documentation, include what can help other users in README.rst file. When the file is ready, another one is needed to tell PyPi to show it.

Create setup.cfg file and put these two lines in it:

```
[metadata]
description-file = README.rst
```

Note that if you try to put Markdown file into your package, PyPi will read it as a pure text file without any formatting.

Licensing

It's often more than welcome to put a LICENSE.txt file in your package with one of the OpenSource licenses to tell users if they can use your package for example in commercial projects or if your code is usable with their license.

In more readable way some licenses are explained at TL;DR .

Making package executable

If your package isn't only a library, but has a piece of code that can be used either as a showcase or a standalone application when your package is installed, put that piece of code into `__main__.py` file.

Put the `__main__.py` in the `package_name` folder. This way you will be able to run it directly from console:

```
python -m package_name
```

If there's no `__main__.py` file available, the package won't run with this command and this error will be printed:

> python: No module named `package_name.__main__`; 'package_name' is a package and cannot be directly executed.

Syntax

Parameters

Remarks

The pypa sample project contains a complete, easily modifiable template `setup.py` that demonstrates a large range of capabilities setup-tools has to offer.