Cookiecutter Python Library Documentation

Release 0.1.0

Bernardo Martínez Garrido

October 12, 2016

Contents

This is the documentation for the Cookiecutter Python Library.

Created to fit my own tastes, this library offers a series of good practices, and a bunch of configurations I've found useful.

CHAPTER 1

Features

- Travis configuration file
- Customized setup.py module to minimize configuration, and using tox for the tests
- Docs using Sphinx and the Sphinx Docs Theme
- Prepared to run tests through tox
- Prepared to run tests on Python 2.6, 2.7, 3.2, 3.3, 3.4
- Prepared to run tests on pypy and pypy 3
- Prepared to run tests on Jython
- Prepared to run coverage tests and integrate with Coveralls
- Prepared to run tests for the Sphinx documentation

1.1 Acquiring the project

The project files are kept on Github, where they can be freely acquired, but this is unneeded because Cookiecutter will automatically take them from said repository, as shown in the usage section.

1.2 Usage

1.2.1 Requirements

This project requires the use of Cookiecutter, which can be installed with the help of pip:

```
$ pip install cookiecutter
```

1.2.2 Creating a new project

After that the template can be acquired and used directly from Github:

```
$ cookiecutter gh:bernardo-mg/cookiecutter-python-library
```

For reusing it there is no need to download the project again:

```
$ cookiecutter cookiecutter-python-library/
```

Once the command has been executed just follow the instructions which will appear on the screen, asking for some information to fill the project template.

1.3 Documentation site

The Sphinx folder contains basic files to generate a Sphinx documentation site. Of course, Sphinx can be used to create other kinds of docs, but combined with ReadTheDocs it is possible to have an always accessible documentation site, which will look similar to this one.

There is little to add which can't be found on the Sphinx site, apart from the fact that the Sphinx Docs Theme template has been used for the UI look.

1.4 The makefile

A makefile, and an equivalent make.bat file for windows, is offered as an easy way to control the project's flow.

It is possible to get all the available tasks with the command:

```
$ make help
```

But there is a small summary:

Command	Description
clean	Removes all the generated and distribution files
build	Creates the source distribution
install	Installs the project in the local repository
requirements	Installs the project requirements
register	Registers the project into Pypi
deploy	Deploys the project into Pypi
test	Runs the tox tests suite

1.4.1 Pypi commands

Pypi requires access information. Otherwise the commands making use of this service won't work.

For this a '.pypirc' file should be on the user folder, with the following data:

```
[distutils]
index-servers =
    pypi
    pypitest

[pypi]
username: username_pypi
password: password_pypi

[pypitest]
repository: https://testpypi.python.org/pypi
username: username_pypitest
password: password_pypitest
```

Where the usernames and passwords should be changed for the correct ones.

1.4.2 Deployment

Deployment is made with Twine, to make sure old versions of Python use HTTPS when deploying.

1.5 Tests

Projects created from the template come ready to run test suites, which include not only unit tests, but also documentation validation and coverage reports.

1.5.1 tox

The new project will come ready to run most of the test suites through tox. This framework helps isolating and reproducing the test environment and is completely compatible with Travis CI service, which is the recommended way to run said tests.

tox environments

Travis offers several Python interpreters, which allow testing using different test environments. The interpreters it has include not only various Python 2 and 3 releases, but also Pypy.

It is also possible to run the tests for a concrete release manually, but in that case the correct Python interpreter should have been installed locally.

For that the usual command can be used:

```
$ tox -e env_name
```

Where 'env_name' is the Python version code, such as 'py27' for Python 2.7.

Coverage

The included '.coveragerc' file allows using Coveralls for generating coverage reports.

This can be done through tox with the following command:

```
$ tox -e coverage
```

Which will generate and send the coverage information required for the report. If the job is done with Travis, something the included Travis configuration file already takes care of, no additional configuration is required.

Otherwise check the Coveralls page to find instructions in how to set up the coverage process.

Documentation validation

It is possible to validate the project's documentation with tox and the following command:

```
$ tox -e docs
```

This will run the Sphinx tests, and it is a good idea running it before deploying the docs, like the included Travis configuration file does.

Style validation

In a similar way to the documentation, the code's style can be validated. For this the following tox command can be used:

```
$ tox -e check
```

By default Travis won't run this environment, as it is too prone to failures. It will check the readme, the manifest and all the code, making sure they conform style standards.