cookiecutter-pypackage Documentation

Release 0.9.0

Audrey Roy Greenfeld

Oct 04, 2017

Contents

1 Getting Started **1**
- 1.1 cookiecutter-pypackage . . . 1
- 1.2 Tutorial . . . 3
- 1.3 PyPI Release Checklist . . . 4

2 Basics **7**
- 2.1 Prompts . . . 7

3 Advanced Features **9**
- 3.1 Travis/PyPI Setup . . . 9

4 Indices and tables **11**

CHAPTER 1

Getting Started

cookiecutter-pypackage

Cookiecutter template for a Python package.

- GitHub repo: https://github.com/audreyr/cookiecutter-pypackage/
- Free software: BSD license

Features

- Testing setup with `unittest` and `python setup.py test` or `py.test`
- Travis-CI: Ready for Travis Continuous Integration testing
- Tox testing: Setup to easily test for Python 2.6, 2.7, 3.3, 3.4, 3.5
- Sphinx docs: Documentation ready for generation with, for example, ReadTheDocs
- Bumpversion: Pre-configured version bumping with a single command
- Auto-release to PyPI when you push a new tag to master (optional)

Quickstart

Install cookiecutter if you haven't installed it yet):

```
pip install cookiecutter
```

Generate a Python package project:

```
cookiecutter https://github.com/audreyr/cookiecutter-pypackage.git
```

Then:

- Create a repo and put it there.
- Add the repo to your Travis-CI account.
- Install the dev requirements into a virtualenv. (`pip install -r requirements_dev.txt`)
- Run the script *travis_pypi_setup.py* to encrypt your PyPI password in Travis config and activate automated deployment on PyPI when you push a new tag to master branch.
- Add the repo to your ReadTheDocs account + turn on the ReadTheDocs service hook.
- Release your package by pushing a new tag to master.
- Add a *requirements.txt* file that specifies the packages you will need for your project and their versions. For more information see the pip documentation for requirements files.
- Activate your project on RequiresIO.

For more details, see the cookiecutter-pypackage tutorial.

Not Exactly What You Want?

Don't worry, you have options:

Similar Cookiecutter Templates

- Nekroze/cookiecutter-pypackage: A fork of this with a PyTest test runner, strict flake8 checking with Travis/Tox, and some docs and *setup.py* differences.
- tony/cookiecutter-pypackage-pythonic: Fork with py2.7+3.3 optimizations. Flask/Werkzeug-style test runner, `_compat` module and module/doc conventions. See `README.rst` or the github comparison view for exhaustive list of additions and modifications.
- ardydedase/cookiecutter-pypackage: A fork with separate requirements files rather than a requirements list in the `setup.py` file.
- Also see the network and family tree for this repo. (If you find anything that should be listed here, please add it and send a pull request!)

Fork This / Create Your Own

If you have differences in your preferred setup, I encourage you to fork this to create your own version. Or create your own; it doesn't strictly have to be a fork.

- Once you have your own version working, add it to the Similar Cookiecutter Templates list above with a brief description.
- It's up to you whether or not to rename your fork/own version. Do whatever you think sounds good.

Or Submit a Pull Request

I also accept pull requests on this, if they're small, atomic, and if they make my own packaging experience better.

Tutorial

Note: Did you find any of these instructions confusing? Edit this file and submit a pull request with your improvements!

Step 1: Install Cookiecutter

First, create a virtualenv for your new package:

```
virtualenv ~/.virtualenvs/mypackage
```

Here, *mypackage* is the name of the package that you'll create.

Activate your environment

```
source bin/activate
```

On Windows

```
> \path\to\env\Scripts\activate
```

Install cookiecutter

```
pip install cookiecutter
```

Step 2: Generate Your Package

Now it's time to generate your Python package.

Use cookiecutter, pointing it at the cookiecutter-pypackage repo:

```
cookiecutter https://github.com/audreyr/cookiecutter-pypackage.git
```

You'll be asked to enter a bunch of values to set the package up. If you don't know what to enter, stick with the defaults.

Step 3: Create a GitHub Repo

Create a repo and put your package there.

```
Be sure to add you environment folder into the .gitignore file.
```

This can be accomplished with:

```
git init .
git add .
git commit -m "Initial skeleton."
git remote add origin git@github.com:myusername/mypackage.git
git push -u origin master
```

Where *myusername* and *mypackage* are the adjusted for the specific project.

You'll need a ssh key to push the repo. You can Generate a key or Add existing one.

Step 4: Install Dev Requirements

Make sure you are in the folder containing the `requirements_dev.txt` file.

Install the new project's local development requirements into a virtualenv:

```
pip install -r requirements_dev.txt
```

If you have issues building the wheel for cryptography. Make sure that the required dependencies are installed. Follow the **'Cryptography Instruction'_** for your OS

Step 5: Set Up Travis CI

Travis CI org is a continuous integration tool used to prevent integration problems. Every commit to the master branch will trigger automated builds to create the necessary files to run the application.

Login using your Github credentials.

Add the public repo to your Travis CI account.

[#] For private projects got to Travis CI com

Go to your terminal and run the script *travis_pypi_setup.py*. It will:

- Encrypt your PyPI password in your Travis config.
- Activate automated deployment on PyPI when you push a new tag to master branch.

See *Travis/PyPI Setup*.

Step 6: Set Up ReadTheDocs

'ReadTheDocs'_ hosts documentation for the open source community. Think of it as Continuous Documentation.

Log into your account at **'ReadTheDocs'_** .

Import the repository

In your GitHub repo settings > Webhooks & services, turn on the ReadTheDocs service hook.

Step 7: Release on PyPI

The Python Package Index or **'PyPI'_** is the official third-party software repository for the Python programming language. Python developers intend it to be a comprehensive catalog of all open source Python packages.[1]

Release your package the standard Python way.

'PyPI Help'_ submitting a package.

Here's a release checklist: https://gist.github.com/audreyr/5990987

PyPI Release Checklist

Before Your First Release

1. Register the package on PyPI:

```
python setup.py register
```

2. Visit PyPI to make sure it registered.

For Every Release

1. Update HISTORY.rst
2. Commit the changes:

```
git add HISTORY.rst
git commit -m "Changelog for upcoming release 0.1.1."
```

3. Update version number (can also be patch or major)

```
bumpversion minor
```

4. Install the package again for local development, but with the new version number:

```
python setup.py develop
```

5. Run the tests:

```
tox
```

6. Push the commit:

```
git push
```

7. Push the tags, creating the new release on both GitHub and PyPI:

```
git push --tags
```

8. Check the PyPI listing page to make sure that the README, release notes, and roadmap display properly. If not, try one of these:
   (a) Copy and paste the RestructuredText into http://rst.ninjs.org/ to find out what broke the formatting.
   (b) Check your long_description locally:

```
pip install readme_renderer
python setup.py check -r -s
```

9. Edit the release on GitHub (e.g. https://github.com/audreyr/cookiecutter/releases). Paste the release notes into the release's release page, and come up with a title for the release.

About This Checklist

This checklist is adapted from:

- https://gist.github.com/audreyr/5990987
- https://gist.github.com/audreyr/9f1564ea049c14f682f4

It assumes that you are using all features of Cookiecutter PyPackage.

CHAPTER 2

Basics

Prompts

When you create a package, you are prompted to enter these values.

Templated Values

The following appear in various parts of your generated project.

full_name Your full name.

email Your email address.

github_username Your GitHub username.

project_name The name of your new Python package project. This is used in documentation, so spaces and any characters are fine here.

project_slug The namespace of your Python package. This should be Python import-friendly. Typically, it is the slugified version of project_name.

project_short_description A 1-sentence description of what your Python package does.

release_date The date of the first release.

pypi_username Your Python Package Index account username.

year The year of the initial package copyright in the license file.

version The starting version number of the package.

Options

The following package configuration options set up different features for your project.

use_pypi_deployment_with_travis Whether to use PyPI deployment with Travis.

CHAPTER 3

Advanced Features

Travis/PyPI Setup

Optionally, your package can automatically be released on PyPI whenever you push a new tag to the master branch.

How It Works

Your project comes with a script called *travis_pypi_setup.py*.

This script does the following:

- Encrypt your PyPI password and save it in your Travis config
- Activate automated deployment on PyPI when you push a new tag to master.

The encryption is done using RSA encryption, you can read more about Travis encryption here. In short, the encrypted password can only be decrypted by Travis, using the private key it associates with your repo.

Using the Travis command-line tool instead

If you have the *travis* command - line tool installed, instead of using the *travis_pypi_setup.py* script you can do:

```
travis encrypt --add deploy.password
```

Which does essentially the same thing.

Your Release Process

If you are using this feature, this is how you would do a patch release:

```
bumpversion patch
git push --tags
```

This will result in:

- mypackage 0.1.1 showing up in your GitHub tags/releases page
- mypackage 0.1.1 getting released on PyPI

You can also replace patch with *minor* or *major*.

More Details

You can read more about using Travis for PyPI deployment at: https://docs.travis-ci.com/user/deployment/pypi/

CHAPTER 4

Indices and tables

- genindex
- modindex
- search