

---

# **Pycookiecutter Documentation**

***Release 0.1.0***

**Qiong X. Michaels**

**Jun 02, 2019**



---

## Contents

---

<b>1</b>	<b>Getting Started</b>	<b>1</b>
1.1	Pycookiecutter . . . . .	1
1.2	Tutorial . . . . .	2
1.3	PyPI Release Checklist . . . . .	4
1.4	Troubleshooting . . . . .	5
1.5	History . . . . .	6
1.6	Authors . . . . .	6
1.7	Contributing . . . . .	6
<b>2</b>	<b>Basics</b>	<b>9</b>
2.1	Prompts . . . . .	9
<b>3</b>	<b>Advanced Features</b>	<b>11</b>
3.1	Travis/PyPI Setup . . . . .	11
3.2	Console Script Setup . . . . .	12
<b>4</b>	<b>Indices and tables</b>	<b>13</b>



## 1.1 Pycookiecutter

### 1.1.1 Overview

Starting template for creating a Python package. It's built using the [Cookiecutter](#) and the [audreyr/cookiecutter-pypackage](#) packages.

- Free software: BSD license
- Documentation: <https://pycookiecutter.readthedocs.io>.

### 1.1.2 Features

- Travis CI configuration
- Tox configuration (Python3.5/3.6/3.7)
- Sphinx configuration
- Easy PyPI registration/distribution
- Badges and more options.

### 1.1.3 Install & Run

To generate a template, run this command in your terminal:

```
$ cookiecutter https://github.com/mqlab-dev/pycookiecutter
```

Or:

```
$ pip install pycookiecutter
$ cookiecutter pycookiecutter
```

## 1.2 Tutorial

---

**Note:** Did you find any of these instructions confusing? [Edit this file](#) and submit a pull request with your improvements!

---

To start with, you will need a [GitHub account](#) and an account on [PyPI](#). Create these before you get started on this tutorial. If you are new to Git and GitHub, you should probably spend a few minutes on some of the tutorials at the top of the page at [GitHub Help](#).

### 1.2.1 Step 1: Install Cookiecutter

First, you need to create and activate a virtualenv for the package project. Use your favorite method, or create a virtualenv for your new package like this (Recommend installation of Python3.5 or above):

```
python3 -m venv env
```

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

Activate your environment:

```
source env/bin/activate
```

On Windows, activate it like this. You may find that using a Command Prompt window works better than gitbash.

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

Install cookiecutter:

```
pip install cookiecutter
```

### 1.2.2 Step 2: Generate Your Package

Now it's time to generate your Python package.

Use cookiecutter, pointing it at the pycookiecutter repo:

```
cookiecutter https://github.com/mqlab-dev/pycookiecutter.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.

### 1.2.3 Step 3: Create a GitHub Repo

Go to your GitHub account and create a new repo named `mypackage`, where `mypackage` matches the `[project_slug]` from your answers to running `cookiecutter`. This is so that Travis CI and `pyup.io` can find it when we get to Step 5.

If your `virtualenv` folder is within your project folder, be sure to add the `virtualenv` folder name to your `.gitignore` file.

You will find one folder named after the `[project_slug]`. Move into this folder, and then setup git to use your GitHub repo and upload the code:

```
cd mypackage
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 adjusted for your username and package name.

You'll need a ssh key to push the repo. You can [Generate](#) a key or [Add](#) an existing one.

### 1.2.4 Step 4: Install Dev Requirements

You should still be in the folder containing the `requirements_dev.txt` file.

Your `virtualenv` should still be activated. If it isn't, activate it now. Install the new project's local development requirements:

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

### 1.2.5 Step 5: Set Up Travis CI

[Travis CI](#) is a continuous integration tool used to prevent integration problems. Every commit to the master branch will trigger automated builds of the application.

Login using your Github credentials. It may take a few minutes for Travis CI to load up a list of all your GitHub repos. They will be listed with boxes to the left of the repo name, where the boxes have an X in them, meaning it is not connected to Travis CI.

Add the public repo to your Travis CI account by clicking the X to switch it "on" in the box next to the `mypackage` repo. Do not try to follow the other instructions, that will be taken care of next.

In your terminal, your `virtualenv` should still be activated. If it isn't, activate it now. Run the Travis CLI tool to do your Travis CI setup:

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

This 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](#) for more information.

## 1.2.6 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](#) . If you don't have one, create one and log into it.

If you are not at your dashboard, choose the pull-down next to your username in the upper right, and select "My Projects". Choose the button to Import the repository and follow the directions.

In your GitHub repo, select Settings > Webhooks & Services, turn on the ReadTheDocs service hook.

Now your documentation will get rebuilt when you make documentation changes to your package.

## 1.2.7 Step 7: Set Up pyup.io

[pyup.io](#) is a service that helps you to keep your requirements files up to date. It sends you automated pull requests whenever there's a new release for one of your dependencies.

To use it, create a new account at [pyup.io](#) or log into your existing account.

Click on the green Add Repo button in the top left corner and select the repo you created in Step 3. A popup will ask you whether you want to pin your dependencies. Click on Pin to add the repo.

Once your repo is set up correctly, the pyup.io badge will show your current update status.

## 1.2.8 Step 8: 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.

When you are ready, release your package the standard Python way.

See [PyPI Help](#) for more information about submitting a package.

Here's a release checklist you can use: <https://gist.github.com/mqlab-dev/5990987>

## 1.2.9 Having problems?

Visit our [Troubleshooting](#) page for help. If that doesn't help, go to our [Issues](#) page and create a new Issue. Be sure to give as much information as possible.

# 1.3 PyPI Release Checklist

## 1.3.1 Before Your First Release

1. Register the package on PyPI:

```
python setup.py register
```

2. Visit PyPI to make sure it registered.



### 1.3.2 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)

```
git tag <new-version-number>
git push --tags
```

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:

1. Copy and paste the RestructuredText into <http://rst.ninjs.org/> to find out what broke the formatting.
2. 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/mqlab-dev/pycookiecutter/releases>). Paste the release notes into the release's release page, and come up with a title for the release.

### 1.3.3 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.

## 1.4 Troubleshooting

---

**Note:** Can you help improve this file? [Edit this file](#) and submit a pull request with your improvements!

---

### 1.4.1 Windows Issues

- Some people have reported issues using git bash; try using the Command Terminal instead.
- Virtual environments can sometimes be tricky on Windows. If you have Python 3.5 or above installed (recommended), this should get you a virtualenv named `myenv` created inside the current folder:

```
> c:\Python35\python -m venv myenv
```

Or:

```
> c:\Python35\python c:\Python35\Tools\Scripts\pyvenv.py myenv
```

- Some people have reported that they have to re-activate their virtualenv whenever they change directory, so you should remember the path to the virtualenv in case you need it.

## 1.5 History

### 1.5.1 0.1.0 (2019-05-29)

- First release on PyPI.

### 1.5.2 0.2.1 (2019-06-01)

## 1.6 Authors

### 1.6.1 Development Lead

- Qiong X. Michaels <[qiong@mqlab.dev](mailto:qiong@mqlab.dev)>

### 1.6.2 Contributors

## 1.7 Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given. You can contribute in many ways:

### 1.7.1 Types of Contributions

#### Report Bugs

Report bugs at <https://github.com/mqlab-dev/pycookiecutter/issues>.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

## Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with “bug” and “help wanted” is open to whoever wants to implement it.

## Implement Features

Look through the GitHub issues for features. Anything tagged with “enhancement” and “help wanted” is open to whoever wants to implement it.

## Write Documentation

Pycookiecutter could always use more documentation, whether as part of the official Pycookiecutter docs, in docstrings, or even on the web in blog posts, articles, and such.

## Submit Feedback

The best way to send feedback is to file an issue at <https://github.com/mqlab-dev/pycookiecutter/issues>.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

## 1.7.2 Get Started!

Ready to contribute? Here’s how to set up *pycookiecutter* for local development.

1. Fork the *pycookiecutter* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/pycookiecutter.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv pycookiecutter
$ cd pycookiecutter/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 pycookiecutter tests
$ python setup.py test or py.test
$ tox
```

To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

### 1.7.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 3.5, 3.6, and 3.7. Check [https://travis-ci.org/mqlab-dev/pycookiecutter/pull\\_requests](https://travis-ci.org/mqlab-dev/pycookiecutter/pull_requests) and make sure that the tests pass for all supported Python versions.

### 1.7.4 Tips

To run a subset of tests:

```
$ py.test tests.test_pycookiecutter
```

### 1.7.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ git tag
$ git push --tags
```

Travis will then deploy to PyPI if tests pass.

### 2.1 Prompts

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

#### 2.1.1 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.

**pypi\_username** Your Python Package Index account username.

**version** The starting version number of the package.

#### 2.1.2 Options

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

**use\_pytest** Whether to use Pytest.

**use\_pypi\_deployment\_with\_travis** Whether to use PyPI deployment with Travis.

**add\_pyup\_badge** Whether to add a Pyup badge.

**command\_line\_interface** Whether to create a console script using Click. Console script entry point will match the project\_slug. Options: ['Click', "No command-line interface"]

**create\_author\_file** Whether to create a AUTHORS.rst file.

**open\_source\_license** Choose a open source license.

### 3.1 Travis/PyPI Setup

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

#### 3.1.1 Install the Travis CLI tool

This is OS-specific.

### Mac OS X

We recommend the Homebrew travis package:

```
brew install travis
```

### Windows and Linux

Follow the official Travis CLI installation instructions for your operating system:

<https://github.com/travis-ci/travis.rb#installation>

#### 3.1.2 How It Works

Once you have the *travis* command - line tool installed, from the root of your project do:

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

This will encrypt your locally-stored PyPI password and save that to your *.travis.yml* file. Commit that change to git.

#### 3.1.3 More Details

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

## 3.2 Console Script Setup

Optionally, your package can include a console script

### 3.2.1 How It Works

If the ‘command\_line\_interface’ option is set to [‘click’] during setup, cookiecutter will add a file ‘cli.py’ in the project\_slug subdirectory. An entry point is added to setup.py that points to the main function in cli.py.

### 3.2.2 Usage

To use the console script in development:

```
pip install -e projectdir
```

‘projectdir’ should be the top level project directory with the setup.py file

The script will be generated with output for no arguments and –help.

**--help**                      show help menu and exit

### 3.2.3 Known Issues

Installing the project in a development environment using:

```
python setup.py develop
```

will not set up the entry point correctly. This is a known issue with Click. The following will work as expected:

```
python setup.py install  
pip install mypackage
```

With ‘mypackage’ adjusted to the specific project.

### 3.2.4 More Details

You can read more about Click at: <http://click.pocoo.org/>



## CHAPTER 4

---

### Indices and tables

---

- `genindex`
- `modindex`
- `search`