

---

# **cookiecutter-pylib Documentation**

***Release 0.9.0***

**Audrey Roy Greenfeld**

**Nov 16, 2018**



---

## Contents

---

<b>1</b>	<b>Getting Started</b>	<b>1</b>
1.1	Cookiecutter PyLib . . . . .	1
1.2	Tutorial . . . . .	3
1.3	PyPI Release Checklist . . . . .	5
<b>2</b>	<b>Basics</b>	<b>7</b>
2.1	Prompts . . . . .	7
<b>3</b>	<b>Advanced Features</b>	<b>9</b>
3.1	Travis/PyPI Setup . . . . .	9
3.2	Console Script Setup . . . . .	10
<b>4</b>	<b>Indices and tables</b>	<b>11</b>



### 1.1 Cookiecutter PyLib

Cookiecutter template for a Python PyPI ready package library, extends upon [audreyr/cookiecutter-pypackage](#)

Instead of hacking boilerplate setup you can jump to coding immediately. All the check are already in place and a your package can be nicely released to pypi.

- GitHub repo: <https://github.com/wooyek/cookiecutter-pylib/>
- Documentation: <https://cookiecutter-pylib.readthedocs.io/>
- Free software: BSD license

#### 1.1.1 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 multiple Python versions
- [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)
- Split file [requirements](#) management with inheritance and support for private locally installed packages
- Command line interface using [Click](#) (optional)
- [Initialization of Git](#) repo, github origin remote and git-flow
- Python [virtual environment bootstrapping](#)
- Git master/develop local/origin sync with a single command

- [Release](#) (sync, test, bump, publish) with a single command

### 1.1.2 Quickstart

Before we start we need some tooling:

```
curl https://raw.githubusercontent.com/mitsuhiko/pipsi/master/get-pipsi.py | python
pipsi install cookiecutter
pipsi install detox
pipsi install pew
pipsi install vex
pipsi install pipenv
pip install "backports.shutil.which ; python_version < '3.4'"
```

Now we can generate and initialize a Django app package project:

```
cookiecutter https://github.com/wooyek/cookiecutter-pylib.git
```

You'll be prompted for some information, based on your choices cookiecutter create a directory that is your new package.

This directory should contain fully initialized project. Empty but passing tests and ready to be pushed and released.

### 1.1.3 Post generation provisioning

- Activate virtual environment and run tests with `tox`
- Create a repo on Github (or Bitbucket)
- Add the repo to your [Travis-CI](#) account.
- Add the repo to your [ReadTheDocs](#) account + turn on the ReadTheDocs service hook.
- Update `requirements/base.txt` file that specifies the packages you will need for your project and their versions.  
For more info see the [pip docs for requirements files](#).

For more details, see the [cookiecutter-pylib tutorial](#).

## Running Tests

Code has been written, but does it actually work? Let's find out!

```
source .pyvenv/bin/activate
(pyvenv) $ pip install -r requirements/development.txt
(pyvenv) $ pytest
(pyvenv) $ python setup.py test
(pyvenv) $ tox
```

## Register releasing on PyPI

First make sure you have newest setuptools installed:

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

Once you've got at least a prototype working and tests running, it's time to register the app on PyPI:

```
python setup.py register
```

Time to release a new version and upload it to PyPi? Bam!

```
$ make publish
```

It will sync your local and origin repo, test, increment version number, setup and release package then push to origin master.

## 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:

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

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

Activate your environment:

```
source 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 cookiecutter-pylib repo:

```
cookiecutter https://github.com/wooyek/cookiecutter-pylib.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
```

If you have issues building the wheel for cryptography, make sure that the required dependencies are installed. Follow the [Cryptography Instruction](#) for your OS

### 1.2.5 Step 5: Set Up Travis CI

[Travis CI org](#)<sup>0</sup> 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 script to do your Travis CI setup:

```
python travis_pypi_setup.py
```

This will:

- Encrypt your PyPI password in your Travis config.

---

<sup>0</sup> For private projects go to [Travis CI com](#)



- 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/audreyr/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)

```
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.

### 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.

## 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.

**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.

### 2.1.2 Options

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

**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"]

### 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 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.

#### 3.1.2 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.

#### 3.1.3 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*.

### 3.1.4 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`