cookiecutter-pyopensci Documentation

Release 0.9.0

pyOpenSci

Nov 24, 2020

Contents

Cookiecutter template for a pyOpenSci Python packge.

- GitHub repo: https://github.com/pyOpenSci/cookiecutter-pyopensci
- Documentation: https://cookiecutter-pyopensci.readthedocs.io/
- pyOpenSci Guidebook: https://pyopensci.github.io/dev_guide
- Free software: BSD license

CHAPTER 1

Features

- **Tox** and **pytest** testing: Setup to easily test for Python 3.6, 3.7, 3.8
- **Travis-CI**: Ready for Travis Continuous Integration testing
- **codecov**: Code coverage report and badge using codecov and Travis
- **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)

CHAPTER 2

Quickstart

Install the latest Cookiecutter if you haven't installed it yet (this requires Cookiecutter 1.4.0 or higher)

```
pip install -U cookiecutter
```

Generate a Python package project:

```
cookiecutter https://github.com/pyOpenSci/cookiecutter-pyopensci.git
```

For more details, see the `cookiecutter-pyopensci tutorial <https://cookiecutter-pyopensci.readthedocs.io/en/latest/tutorial.html>`__

CHAPTER 3

Getting Started

3.1 Template Contents

After you run cookiecutter to create your package from the template, you'll have a directory full of files and subdirectories. What are they all for? This page explains. For starting out, the most important files and directories are marked with *.

First, here is the full contents of the template project that cookiecutter creates:

```
example_project/
├── .editorconfig
├── .gitignore
├── .travis.yml
├── CONTRIBUTING.rst
├── CONTRIBUTORS.rst
├── docs\*
│   ├── conf.py
│   ├── contributing.rst
│   ├── contributors.rst
│   ├── history.rst
│   ├── index.rst
│   ├── installation.rst
│   ├── make.bat
│   ├── Makefile
│   ├── readme.rst
│   └── usage.rst\*
├── HISTORY.rst
├── LICENSE
├── Makefile
├── MANIFEST.in
├── example_project\*
│   ├── __init__.py
│   └── example_project.py
├── README.rst\*
```

(continues on next page)

(continued from previous page)

```
├── requirements_dev.txt
├── setup.cfg
├── setup.py
├── tests\*
│   └── test_example_project.py
└── tox.ini
```

3.1.1 Top-level .rst files: CONTRIBUTING, CONTRIBUTORS, HISTORY, README*

These documents store important info about your project.

- **README.rst**:* This is usually the first document users will look at. It will show up in the main page of your GitHub repository. Be sure to include the essential info about your package, including how to install and use it (or at least direct them to other documentation with this info).
- **CONTRIBUTING.rst**: Contains guidelines for others who want to contribute code or documentation to your project. The template has some basic, common sense guidelines to start out. Feel free to modify or add to them, but be welcoming to new contributors! Their work can help improve your project.
- **CONTRIBUTORS.rst**: List of contributors to your project. Starts with just you. As others contribute, you should credit them here.
- **HISTORY.rst**: Use this to track changes to your package as you release new versions.

3.1.2 docs/*

Documentation files that will be used to build your ReadTheDocs website.

- **usage.rst***: Instructions for how to use your package. Starts with the barebones: how to import your package. You should build on it as you add functionality to your package. Be sure to include examples for all the major functions in your package.
- **conf.py**, **make.bat**, and **Makefile** are there to help ReadTheDocs build your documentation. They don't need modification, at least not at first.
- Some of the .rst files share names with the top-level .rst files we just discussed (e.g. docs/readme.rst and README.rst). These files just point to the top-level .rst files. The rest have their own content
- **index.rst**: The index file for your documentation. Only change this if you add new files to your docs/ directory.
- **installation.rst**: Installation instructions. Starts with a basic outline for installing from github or PyPI (once your project is released there).

3.1.3 LICENSE

A text file containing the software license you selected in the cookiecutter prompts. This describes the legal terms of use, distribution, and modification of your package. You shouldn't have to touch this file, but if you would like to use a different open source license than the options in the set up you can replace the LICENSE file with the text of your chosen license.

3.1.4 Makefile and MANIFEST.in

You shouldn't have to modify either of these files.

- **Makefile**: Includes instructions for make to perform various setup/testing tasks. For example, running "make test" with run through your package's testing suite.
- **MANIFEST.in**: Defines the list of files to include in the release distribution of your package.

3.1.5 example_project/*

The actual code for your Python package. Comes with a single main module template file. If your package is contained within a single file, copy and paste it into this template. Otherwise, you can add more files as needed.

- **example_project.py**: Template main module file for your package. Contains two very basic example functions: sum_numbers and max_number. These are used for example tests (more on that in a minute).
- **__init__.py**: This file tells Python to treat the directory as containing a package. It also has some basic metadata (author, email, version #). You don't have to modify anything here.

3.1.6 tests/

Contains your testing files. These files provide functions that test that your module's functions are producing the expected results. pytest automatically discovers files starting with "test_" and runs the functions prefixed by "test_". An example file is included:

- **test_example_project.py**: Includes two example tests: test_sum_numbers and test_max_number. These check that the functions are returning the expected results. The "generate_numbers" pytest fixture generates a list of random numbers for the tests. As you add your code to the project, be sure to replace these examples with tests for all the major functions and methods in your package.

3.1.7 requirements_dev.txt, setup.py, and setup.cfg

Files to facilitate installing your package.

- **requirements_dev.txt**: List of Python packages required for working on developing your package.Includes the version numbers. If you add more dependencies to your package, be sure to add them here as well. All of these packages can be installed at once:
- **setup.py**: Allows users to install your package from the commandline: "python setup.py install". You should not have to modify this at all.
- **setup.cfg**: Contains extra configuration options for setup.py.

3.1.8 tox.ini

Config file for tox. tox runs your tests on several versions of Python. The default is 3.6, 3.7 and 3.8. Notice that tox calls pytest for each version of Python.

3.1.9 .travis.yml

Configuration file for Travis CI. Travis builds and tests your package each time you commit to GitHub. This makes it a lot easier to guard against changes that break your package. This .yml file tells Travis how to install and test your package. It even includes the option for Travis to automatically deploy your package to PyPI when you commit a new version of your package. To start with, you can ignore that part. When you are ready to publish your package on PyPI, you can add your encrypted PyPI password to this config file and Travis will automatically release new versions to PyPI. We have instructions for setting up Travis and PyPI here.

3.1.10 .gitignore and .editorconfig

These simple files make working on your package easier. You shouldn't need to modify them.

- **.gitignore**: Tells git which files and patterns to exclude from tracking. If you do "git add .", git will add all files and directories EXCEPT those that match patterns in .gitignore.
- **.editorconfig**: Specifies text editor options like tabs vs. spaces, trailing whitespace handling, etc. editorconfig allows you to specify these options at the project level and distribute them to contributors. You probably don't have to use it: many code editors will already adopt the typical standards for Python. Not all editors come with support for .editorconfig files built in, but most have a plugin you can install. Read more: http://editorconfig.org

3.2 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.

3.2.1 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
```

3.2.2 Step 2: Generate Your Package

Now it's time to generate your Python package.

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

```
cookiecutter https://github.com/pyOpenSci/cookiecutter-pyopensci.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. Here's a list of the options, and a brief description:

- **full_name**: Enter package creator's full name.
- **email**: Enter email address.
- **github_username**: Package creator's github username.
- **project_name**: Choose a short and descriptive name for your package.
- **project_slug**: A shorthand name for your project, using "_" instead of spaces and avoiding "-".
- **project_short_description**: Short (one sentence) description of your package and what it does.
- **pypi_username**: Username for deploying to PyPI. If you don't have one, just leave as default for now. This is not important for now.
- **version**: Version number for your package. If unsure, leave as default.
- **use_pypi_deployment_with_travis**: Travis CI can automatically deploy your project to PyPI when you update to a new version number. If you are unsure, leave as "n". This can be changed later.
- **add_pyup_badge**: PyUp.io is a service that helps keep your package compatitible with updates to its dependencies.
- **open_source_license**: Select an open source software license. For more guidance, see the license section of the pyOpenSci guidebook. If you would like to use a license not on this list, just choose one and replace the LICENSE file with the license of your choosing.

The result will be a newly-created repository in a new folder called `[project_slug]`. Next, we'll upload this repository to GitHub.

3.2.3 Step 3: Create a GitHub Repo

Next we need to create an online repository on GitHub where you can push your newly-created local repository.

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.

Note: 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
# Create an empty repository
git init .
# Add and commit all the files in this repository
git add .
git commit -m "Initial skeleton."
# Add our GitHub.com repository as a "remote" and push to it
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 an ssh key to push the repo to GitHub. You can Generate a key or Add an existing key that allows your to do so.

3.2.4 Step 4: Install Development Requirements

Next you'll need to install the packages needed to develop your repository. Any repository should come packaged with a list of the packages needed to work with it (both as a developer and as a user). When you ran the `cookiecutter` command, a sample requirements file was created, called `requirements_dev.txt`.

You should still be in the root folder of your repository. It should contain a `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
```

This will install all of the packages specified in `requirements_dev.txt` into your environment.

3.2.5 Step 5: Set Up Travis CI

Travis CI org*[0] is a continuous integration tool used to prevent integration problems. Every commit to the master branch will trigger automated builds of the application, and it is common to run a series of **tests** along with each build to ensure the code still behaves the way we'd expect.

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.

If you chose to turn on automatic PyPI deployment via Travis CI, see *Travis/PyPI Setup* for more information.

3.2.6 Step 6: Set Up ReadTheDocs

ReadTheDocs hosts documentation for the open source community. Think of it as Continuous Documentation. When you make changes to a branch, ReadTheDocs will automatically re-build your documentation so they reflect the latest changes.

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.

3.2.7 Step 7 (optional): 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.

[0] For private projects go to Travis CI com

3.2.8 Step 8 (optional): 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.

Note: If you are submitting your package for pyOpenSci peer-review, we ask that you wait to release your package on PyPI. This makes it easier to implement changes during the review process. If you are already on PyPI, that's no problem of course!

When you are ready, see PyPI Help for more information about submitting a package.

Here's a release checklist you can use: *PyPI Release Checklist*:

If you turned on automatic deployment to PyPI via Travis, see *Travis/PyPI Setup* for more info.

3.2.9 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.

3.3 PyPI Release Checklist

3.3.1 Before Your First Release

1. Register the package on PyPI:

```
python setup.py register
```

2. Visit PyPI to make sure it registered.

3.3.2 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:
    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/audreyr/cookiecutter/releases). Paste the release notes into the release's release page, and come up with a title for the release.

3.3.3 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.

CHAPTER 4

Basics

CHAPTER 5

Advanced Features

5.1 Travis/PyPI Setup

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

5.1.1 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 installationinstructions for your operating system:

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

5.1.2 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.

5.1.3 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*.

5.1.4 More Details

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

5.2 Troubleshooting

Note: Can you help improve this file? Edit this file and submit a pull request with your improvements!

5.2.1 Windows Issues

- Some people have reported issues using git bash; try using the

Command Terminal instead.

- If you're on Windows 10, you may also want to try Windows Subsystem for Linux, which allows you to run a linux command line from within Windows. It is still somewhat experimental, but is useful for developing and testing code written in linux.
- 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.

CHAPTER 6

Indices and tables

- genindex
- modindex
- search