package$_helper_2Documentation$

Release 0.1.0

François Durand

Jan 20, 2022

CONTENTS:

CHAPTER

ONE

PACKAGE HELPER 2

Package Helper 2 explains how to create, develop and maintain a Python package.

The most prominent feature of Package Helper 2 is a **tutorial** that gives a checklist of how to:

- Create your package in a few minutes with Cookiecutter,
- Develop and maintain your package with PyCharm.

Optionally, you can also use GitHub, PyPI and Codecov. For more readability, the tools that you do not use can be hidden in the tutorial.

Package Helper 2 also provides a **template** of Python package. It is a fork of https://github.com/audreyr/cookiecutter-pypackage/. Here are the main differences with the original template:

- Personalize default options.
- Include example files for classes, with examples of documentation and testing.
- Documentation:
  - Use a GitHub action and GitHub Pages (instead of ReadTheDocs) to publish the documentation.
  - Use `sphinx.ext.napoleon` to benefit from NumPy style of documentation.
  - Use ReadTheDocs theme.
  - Add a "reference" section in the documentation of the package.
- Use a GitHub action (instead of Travis CI) to perform unit tests.
- Use a GitHub action (instead of Travis CI) to deploy the package on PyPI.
- Generate a local html page displaying the test coverage.
- Use Codecov.
- Minor tweaks in `setup.py`.
- Remove version numbers from `requirements_dev.txt`.

Documentation: https://package-helper-2.readthedocs.io/.

CHAPTER

TWO

TUTORIAL

Package Helper 2 helps you create, develop and maintain a package. If you use all the tools presented in this tutorial, things will work this way:

- You create the file structure of your package in less than a minute with Cookiecutter.
- You use the IDE PyCharm. It is configured to:
  - Generate the documentation of your package locally,
  - Run the unit tests.
  - Generate a local html page displaying what parts of the package are covered by the unit tests.
- Your project is on GitHub. When you push modifications:
  - GitHub automatically generates the documentation in order to check if it works correctly,
  - If the branch is your default branch ("main" or "master"), Github automatically publishes the documentation online,
  - GitHub automatically runs the unit tests on several versions of Python.
  - Codecov displays what parts of the package are covered by the unit tests.
- When you make a *release* on GitHub, GitHub automatically publishes your package on PyPI. This way, any Python user can install it with `pip install`.

Tick the tools that you want to use:

2.1 Preliminaries: Once and for All

2.2 Create Your package

2.3 Develop and Maintain Your Package

CHAPTER

THREE

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:

3.1 Types of Contributions

3.1.1 Report Bugs

Report bugs at https://github.com/francois-durand/package_helper_2/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.

3.1.2 Fix Bugs

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

3.1.3 Implement Features

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

3.1.4 Write Documentation

Package Helper 2 could always use more documentation, whether as part of the official Package Helper 2 docs, in docstrings, or even on the web in blog posts, articles, and such.

3.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/francois-durand/package_helper_2/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 :)

3.2 Get Started!

Ready to contribute? Here's how to set up *package_helper_2* for local development.

1. Fork the *package_helper_2* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/package_helper_2.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 package_helper_2
$ cd package_helper_2/
$ 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 package_helper_2 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.

3.3 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 3.6, 3.7 and 3.8,. Check https://github.com/francois-durand/package_helper_2/actions and make sure that the tests pass for all supported Python versions.

3.4 Tips

To run a subset of tests:

```
$ py.test tests.test_package_helper_2
```

3.5 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:

```
$ bumpversion patch # possible: major / minor / patch
$ git push
$ git push --tags
```

GitHub will then deploy to PyPI if tests pass.

CHAPTER

FOUR

CREDITS

This package is a fork of https://github.com/audreyr/cookiecutter-pypackage/ by Audrey Roy Greenfeld.

4.1 Development Lead

- François Durand <fradurand@gmail.com>

4.2 Contributors

- Fabien Mathieu (local coverage page).
- Maxime Mouchet (run configurations for PyCharm).

CHAPTER

FIVE

HISTORY

5.1 0.1.5 (2022-01-20): Run Configurations

- Cookiecutter generates automatically the run configurations for PyCharm: “All tests” and “Generate docs”.
- It is now possible to choose “main” or “master” as the default git branch.

5.2 0.1.4 (2022-01-18): Local Coverage

- Generate a local html page displaying the test coverage.
- The “build” folder for the documentation is created when generating the file structure of the package.
- Change the default branch from “master” to “main”.
- Update the tutorial to fit with the current versions of PyCharm, GitHub and Codecov.
- Several minor updates in the tutorial.

5.3 0.1.3 (2020-11-25): Fix deployment on PyPI

- Make the title levels consistent between files `README.rst` and `HISTORY.rst`, to avoid a deployment bug on PyPI.

5.4 0.1.2 (2020-11-25): Add release titles

- In the file `HISTORY.rst` of the template, add release titles.

5.5 0.1.1 (2020-11-25): Fix bug in “docs” action

- Fix a bug in the “docs” action.
- Add support for notebooks in documentation by default.

5.6 0.1.0 (2020-11-25)

- First release.