cc-pypackage Documentation

Release 0.4.2

Roberto Preste

Aug 31, 2023

Contents

cc-pypackage is my custom Cookiecutter template for a Python package.

CHAPTER 1

Getting Started

1.1 cc-pypackage

My custom Cookiecutter template for a Python package.

- GitHub repo: https://github.com/robertopreste/cc-pypackage/
- Documentation: https://cc-pypackage.readthedocs.io/
- Free software: BSD license

1.1.1 Features

- Testing setup with `unittest` and `python setup.py test` or `pytest` (used by default)
- Travis-CI: Ready for Travis Continuous Integration testing
- Tox testing: Setup to easily test for Python 3.4, 3.5, 3.6
- 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)
- Command line interface using Click (optional)

1.1.2 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/robertopreste/cc-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`)
- Register your project with PyPI.
- Run the Travis CLI command *travis encrypt –add deploy.password* 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 info see the pip docs for requirements files.
- Activate your project on pyup.io.

For more details, see the cc-pypackage tutorial.

1.1.3 Credits

This Cookiecutter was adapted for my personal needs from the original Cookiecutter-PyPackage by audreyr.

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 cc-pypackage repo:

```
cookiecutter https://github.com/robertopreste/cc-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.

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/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)

```
bump2version 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.

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.

CHAPTER 2

Basics

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 for testing.

use_pypi_deployment_with_travis Whether to use PyPI deployment with Travis.

add_pyup_badge Whether to add the PyUp badge in the README file.

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 an AUTHORS.rst file.

open_source_license What license will be used for the package. Options: ["MIT license", "BSD license", "ISC license", "Apache Software License 2.0", "GNU General Public License v3", "Not open source"]

CHAPTER 3

Advanced Features

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:

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 Your Release Process

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

```
bump2version 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