cookiecutter-modern-pypackage

Release 0.1.2

Federico Jaureguialzo

Jun 14, 2020

CONTENTS

CHAPTER

ONE

GETTING STARTED

1.1 Cookiecutter Modern PyPackage

tests Read the Docs Updates

Cookiecutter template for a modern Python package.

- GitHub repo: https://github.com/fedejaure/cookiecutter-modern-pypackage.git
- Documentation: https://cookiecutter-modern-pypackage.readthedocs.io
- Free software: MIT license

1.1.1 Features

- Dependency tracking using Poetry
- Testing setup with Pytest
- Github Actions ready for Continuous Integration testing
- Linting provided by Flake8 with Flakehell
- Docstring linting provided by Darglint using the Google Python Style Guide
- Static type checking by Mypy
- Formatting provided by Black and Isort
- Checks dependencies for known security vulnerabilities with Safety
- Git hooks managed by pre-commit.
- All development tasks (lint, format, test, etc) wrapped up in a python CLI by invoke
- Multiple Python environments testing provided by Nox
- Documentation provided by Sphinx ready for generation with, for example, Read the Docs
- 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/fedejaure/cookiecutter-modern-pypackage.git
```

Then:

- Create a repo and put it there.
- Install the dev requirements into a virtualenv. (`poetry install`)
- Install pre-commit hooks. (`poetry run inv install_hooks`)
- Add the repo to your Read the Docs account + turn on the Read the Docs service hook.
- Release your package by pushing a new tag to master.
- Activate your project on pyup.io.

For more details, see the tutorial.

1.1.3 Credits

This cookiecutter was built for learning purpose and inspired by:

- audreyr/cookiecutter-pypackage: Cookiecutter template for a Python package.
- briggySmalls/cookiecutter-pypackage: A fork from audreyr/cookiecutter-pypackage using Poetry for package management, with linting, formatting and more.
- hypermodern-python: Hypermodern Python article series.

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

Install cookiecutter:

```
$ pip install cookiecutter
```

We'll also need poetry so [install that too](https://python-poetry.org/docs/#installation):

1.2.2 Step 2: Generate Your Package

Now it's time to generate your Python package.

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

```
$ cookiecutter https://github.com/fedejaure/cookiecutter-modern-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.

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
mypackage $ git init .
mypackage $ git add .
mypackage $ git commit -m "Initial skeleton."
mypackage $ git remote add origin git@github.com:myusername/mypackage.git
mypackage $ 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 `pyproject.toml` file.

Install the new project's local development requirements inside a virtual environment using poetry:

```
$ poetry install
$ poetry run inv install_hooks
```

1.2.5 Step 5: Set Up Read the Docs

Read the Docs hosts documentation for the open source community. Think of it as Continuous Documentation.

Log into your account at Read the Docs . 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.

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

1.2.6 Step 6: 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.

1.2.7 Step 7: 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.

1.2.8 Having problems?

Visit our Issues page and create a new Issue. Be sure to give as much information as possible.

1.3 Changelog

All notable changes to this project will be documented in this file.

The format is based on Keep a Changelog, and this project adheres to Semantic Versioning.

1.3.1 Unreleased

1.3.2 0.1.2 - 2020-06-14

Fixed

- Read the docs build config.

Removed

- Pytype from the dev requirements.

1.3.3 0.1.1 - 2020-06-14

Added

- New option `serve` to the invoke docs task.

Changed

- Improve docs tutorial section.
- Improve docs index section.

Fixed

- README spelling.
- Ivoke pytype task typo.

1.3.4 0.1.0 - 2020-06-11

Added

- First release.

CHAPTER

TWO

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.

version The starting version number of the package.

2.1.2 Options

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

open_source_license Whether to add a license file. Options: ["MIT", "BSD", "ISC", "Apache Software License 2.0", "GNU General Public License v3", "Not open source"s]

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"]

2.2 Invoke

The generated project is ready to run some useful tasks like formatting, linting, testing.

To do this we use pyinvoke to wrap up the required commands.

Execute *inv[oke] –list* to see the list of available commands.

```
$ poerty shell
$ inv[oke] --list
Available tasks:

  clean           Run all clean sub-tasks.
  clean-build     Clean up files from package building.
  clean-docs      Clean up files from documentation builds.
  clean-python    Clean up python file artifacts.
  clean-tests     Clean up files from testing.
  coverage        Create coverage report.
  docs            Build documentation.
  flake8          Run flake8.
  format          Format code.
  hooks           Run pre-commit hooks.
  install-hooks   Install pre-commit hooks.
  lint            Run all linting.
  mypy            Run mypy.
  safety          Run safety.
  tests           Run tests.
  version         Bump version.
```

CHAPTER

THREE

ADVANCED FEATURES

3.1 Console Script Setup

Optionally, your package can include a console script using Click (Python 3.6+).

3.1.1 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 pyproject.toml that points to the main function in cli.py.

3.1.2 Usage

To use the console script in development:

```
pip install -e projectdir
```

'projectdir' should be the top level project directory with the pyproject.toml file

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

--help show help menu and exit

3.1.3 More Details

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