cookiecutter-python-package

Release 1.8.3

Konstantinos Lampridis

Dec 15, 2023

CONTENTS:

CHAPTER

ONE

PYTHON PACKAGE GENERATOR

Python Package Generator supporting 3 different Project *types* to scaffold.
Emphasizing on CI/CD, Testing and Automation, implemented on top of Cookiecutter.

See the Documentation Site for a thorough read.

Source: https://github.com/boromir674/cookiecutter-python-package
Docs: https://python-package-generator.readthedocs.io/en/master/
PyPI: https://pypi.org/project/cookiecutter-python/
CI: https://github.com/boromir674/cookiecutter-python-package/actions/

CHAPTER
TWO

FEATURES

1. Scaffold a modern *ready-to-develop* Python Package (see *Quickstart*)
2. Automatically generate over 24 files, to setup *Test Suite*, *build* scripts & CI Pipeline
3. **Python Package Template** (source code at src/cookiecutter_python/) implemented as a *Cookiecutter*
4. Extensively **Tested** on various systems, factoring the below:
   a. System's platform: **"Linux"**, **"MacOS"** & **"Windows"**
   b. System's Python: **3.7**, **3.8**, **3.9** & **3.10**, **3.11**

   See the *Test Workflow* on the CI server.

2.1 Auto Generated Sample Package Biskotaki

Check the **Biskotaki** *Python Package Project*, for a taste of the project structure and capabilities this Template can generate!
It it entirely generated using this **Python Package Template:**

Source Code hosted on *Github* at https://github.com/boromir674/biskotaki
Python Package hosted on *pypi.org* at https://pypi.org/project/biskotaki/
CI Pipeline hosted on *Github Actions* at https://github.com/boromir674/biskotaki/actions

2.2 Generated Python Package Features

1. **Test Suite**, using pytest, located in *tests* dir
2. **Parallel Execution** of Unit Tests, on multiple cpu's
3. **Documentation Pages**, hosted on *readthedocs* server, located in *docs* dir
4. **Automation**, using tox, driven by single *tox.ini* file
   a. **Code Coverage** measuring
   b. **Build Command**, using the build python package
   c. **Pypi Deploy Command**, supporting upload to both pypi.org and test.pypi.org servers
   d. **Type Check Command**, using mypy
   e. **Lint** *Check* and *Apply* commands, using isort and black
5. **CI Pipeline**, running on Github Actions, defined in *.github/*
   a. **Job Matrix**, spanning different *platform*'s and *python version*'s
      1. Platforms: *ubuntu-latest*, *macos-latest*
      2. Python Interpreters: *3.6*, *3.7*, *3.8*, *3.9*, *3.10*
   b. **Parallel Job** execution, generated from the *matrix*, that runs the *Test Suite*

CHAPTER
THREE

QUICKSTART

3.1 Installation

```
pip install --user cookiecutter-python
```

3.2 Usage

Open a console/terminal and run:

```
generate-python
```

Now, you should have generated a new Project for a Python Package, based on the Template!
Just 'enter' (*cd* into) the newly created directory, ie *cd <my-great-python-package>*.

Develop your package's **Source Code** (*business logic*) inside *src/my_great_python_package* dir :)
Develop your package's **Test Suite** (ie *unit-tests*, *integration tests*) inside *tests* dir :-)

Try Running the Test Suite!

```
tox
```

Read the Documentation's Use Cases section for more on how to leverage your generated Python Package features.

CHAPTER
FOUR

LICENSE

- GNU Affero General Public License v3.0

4.1 Free/Libre and Open Source Software (FLOSS)

4.1.1 Introduction

This is **Cookiecutter Python Package**, a *Template Project* used to *generate* fresh new open source *Python Package*'s.
The Template is implemented as a *cookiecutter* and it is available both as source code and as a Python Package in itself.

Goal of this project is to automate the process of creating a new Python Package, by providing the user with a "bootstrap" method,
to quickly set up all the *support* files required to seemlessly build and publish the package on pypi.org (the official Python Pcakge Index public server).
Additionally, it instruments a basic **Test Suite**, multiple **Commands**, as well as a **CI** pipeline, with parallel execution of the *build matrix*, running on *Github Actions*.

This documentation aims to help people understand what are the features of the library and how they can use it. It presents some use cases and an overview of the library capabilities and overall design.

4.1.2 Why this Generator?

So, why would one opt for this Python Generator?
It is **easy to use**, allowing the generation of a completely fresh new *Python Package Project*, though a *cli*.
You can immediately have a *ci* infrastructure and multiple platform-agnostic *shell* commands working out-of-the-box, so you can focus on developing your *business logic* and your *test cases*.

- It allows scaffolding new projects with a **Test Suite** included, designed to run *Test Cases* in **parallel** (across multiple cpu's) for *speed*.
- New Projects come with a **CI pipeline**, that triggers every time code is pushed on the remote.
- Supports generating projects suited for developing a library (*module*), a cli (*module+cli*) or a pytest plugin.
- The pipeline hosts a **Test Workflow** on *Github Actions* CI, designed to *stress-test* your package.
- Generates a *job matrix* that spawns parallel CI jobs based on factors:: *python versions operating system* and *package installation methods*
- Extensively tested and built on established software, such as *cookiecutter* and *jinja2*.

4.1.3 Generate New Python Package Project

This *python generator* was designed to be installed via *pip* and then invoked through the cli.

Installing the Generator

Cookiecutter Python Package, available as source code on github, is also published on *pypi.org*.

Install as PyPi package

Installing *cookiecutter-python* with *pip* is the way to go, for getting the *generate-python* cli onto your machine. Here we demonstrate how to do that using a

In virtual environment (recommended)

As with any Python Package, it is recommended to install *cookiecutter-python* inside a python *virtual environment*. You can use any of *virtualenv*, *venv*, *pyenv* of the tool of your choice. Here we demonstrate, using *virtualenv*, by running the following commands in a console (aka terminal):

1. Create a virtual environment

```
virtualenv env --python=python3
```

Open a console (aka terminal) and run:

2. Activate environment

```
source env/bin/activate
```

3. Install *cookiecutter-python*

```
pip install cookiecutter-python
```

4. Create symbolic link for the (current) user

```
ln -s env/bin/generate-python ~/.local/bin/generate-python
```

Now the *generate-python* executable should be available (assuming ~/.local/bin is in your PATH)!

For user (option 2)

One could also opt for a *user* installation of *cookiecutter-python* package:

```
python3 -m pip install --user cookiecutter-python
```

For all users (option 3)

The least recommended way of installing *cookiecutter-python* package is to *directly* install in the *host* machine:

```
sudo python3 -m pip install cookiecutter-python
```

Note the need to invoke using *sudo*, hence not that much recommended.

Check installation

Now the *generate-python* cli should be available!
You can verify by running the following:

```
generate-python --version
```

Using the CLI

Using the cli is as simple as invoking *generate-python* from a console.

You can run the following to see all the available parameters you can control:

```
generate-python --help
```

The most common way to generate a new Python Package Project is to run:

```
generate-python
```

This will prompt you to input some values and create a fresh new Project in the current directory!
Now, simply *cd* into the generated Project's directory and enjoy some of the features the generator supplies new projects with!
More on use cases in the next section.
Ready to enjoy some of your newly generated Python Package Project **features** available out-of-the-box!?
For instance:

1. Leverage the supplied *tox environments* to automate various **Testing** and **DevOps** related activities.
   Assuming you have *tox* installed (example installation command: *python3 -m pip install –user tox*) and you have done a *cd* into the newly generated Project directory, you can do for example:
   a. Run the **Test Suite** against different combinations of *Python versions* (ie 3.7, 3.8) and different ways of installing (ie 'dev', 'sdist', 'wheel') the <*my_great_python_package*> package:

      ```
      tox -e "py{3.7, 3.8}-{dev, sdist, wheel}"
      ```

   b. Check the code for **compliance** with **best practises** of the *Python packaging ecosystem* (ie PyPI, pip), build *sdist* and *wheel* distributions and store them in the *dist* directory:

      ```
      tox -e check && tox -e build
      ```

   c. **Deploy** the package's distributions in a *pypi* (index) server:
      1. Deploy to **staging**, using the *test* pypi (index) server at test.pypi.org:

```
TWINE_USERNAME=username TWINE_PASSWORD=password PACKAGE_DIST_VERSION=1.0.0␣
↪tox -e deploy
```

2. Deploy to **production**, using the *production* pypi (index) server at pypi.org:

```
TWINE_USERNAME=username TWINE_PASSWORD=password PACKAGE_DIST_VERSION=1.0.0␣
↪PYPI_SERVER=pypi tox -e deploy
```

Note: Setting PYPI_SERVER=pypi indicates to deploy to *pypi.org* (instead of *test.pypi.org*).

Note: Please modify the TWINE_USERNAME, TWINE_PASSWORD and PACKAGE_DIST_VERSION environment variables, accordingly.
TWINE_USERNAME & TWINE_PASSWORD are used to authenticate (user credentials) with the targeted pypi server.
PACKAGE_DIST_VERSION is used to avoid accidentally uploading distributions of different versions than intended.

2. Leverage the **CI Pipeline** and its **build matrix** to run the **Test Suite** against a combination of different Platforms, different Python interpreter versions and different ways of installing the subject Python Package:
   Trigger the **Test Workflow** on the **CI server**, by *pushing* a git commit to a remote branch (ie *master* on github).
   Navigate to the *CI Pipeline web interface* (hosted on *Github Actions*) and inspect the **build** results!

Note: You might have already *pushed*, in case you answered *yes*, in the *initialize_git_repo* prompt, while generating the Python Package, and in that case, the **Test Workflow** should have already started running!
Out-of-the-box, *triggering* the **Test Workflow** happens only when pushing to the *master* or *dev* branch.

4.1.4 Developer's Corner

Here we offer **Guides** on how to leverage the **CI/CD** to do various Development Operations, in a **GitOps** way.

GitOps Guides

Streamline Documentation Updates

1. Branch of off *main* Branch, and checkout your *topical branch* (*tb*).
2. Create Docs-only changes and commit them to your *tb*.
3. Push git tag *quick-release*, to trigger the Docs Release Workflow, on the CI

A new PR, is expected to **open** from *tb* to a *dedicated docs* branch, and automatically **merge** if Docs Build passed on *rtd* CI.
Then, a new PR, is expected to **open** from *dedicated docs* branch to *main*, with extra commits with Sem Ver Bump, and Changelog updates.

4. Wait for second PR to open, go to github web IU to review it, and merge it.

A new **tag** is expected to be created (on the new main/master commit), and a *PyPI* distribution will be uploaded, a new Docker Image on Dockerhub, and a new Github Release will be created.

Workflows References

- quick-docs.yaml : Listens to *quick-release* git tag, and merges tb –> db, after opening PR.

4.1.5 API References

References to the API of the **cookiecutter_python** Python Distribution.

cookiecutter_python package

Subpackages

cookiecutter_python.hooks package

Submodules

cookiecutter_python.hooks.pre_gen_project module

Pre Cookie Hook: Templated File with jinja2 syntax

exception cookiecutter_python.hooks.pre_gen_project.**InputSanitizationError**

Bases: Exception

cookiecutter_python.hooks.pre_gen_project.**get_request**()

cookiecutter_python.hooks.pre_gen_project.**hook_main**(*request*)

cookiecutter_python.hooks.pre_gen_project.**input_sanitization**(*request*)

cookiecutter_python.hooks.pre_gen_project.**main**()

cookiecutter_python.hooks.post_gen_project module

Post Cookie Hook: Templated File with jinja2 syntax
Cookiecutter post generation hook script that handles operations after the template project is used to generate a target project.

cookiecutter_python.hooks.post_gen_project.**CLI_ONLY**(*x*)

cookiecutter_python.hooks.post_gen_project.**PYTEST_PLUGIN_ONLY**(*x*)

exception cookiecutter_python.hooks.post_gen_project.**PostFileRemovalError**

Bases: Exception

cookiecutter_python.hooks.post_gen_project.**exception**(*subprocess_exception: subprocess.CalledProcessError*)

cookiecutter_python.hooks.post_gen_project.**get_context**() → collections.OrderedDict

Get the Context, that was used by the Templating Engine at render time

cookiecutter_python.hooks.post_gen_project.**get_request**()

cookiecutter_python.hooks.post_gen_project.**git_commit**(*request*)

Commit the staged changes in the generated project.

cookiecutter_python.hooks.post_gen_project.**grant_basic_permissions**(*project_dir: str*)

cookiecutter_python.hooks.post_gen_project.**initialize_git_repo**(*project_dir: str*)

Initialize the Git repository in the generated project.

cookiecutter_python.hooks.post_gen_project.**is_git_repo_clean**(*project_directory: str*) → bool

Check to confirm if the Git repository is clean and has no uncommitted changes. If its clean return True otherwise False.

cookiecutter_python.hooks.post_gen_project.**iter_files**(*request*)

cookiecutter_python.hooks.post_gen_project.**main**()

Delete irrelevant to Project Type files and optionally do git commit.

cookiecutter_python.hooks.post_gen_project.**post_file_removal**(*request*)

Preserve only files relevant to Project Type requested to Generate.
Delete files that are not relevant to the project type requested to generate.
For example, if the user requested a 'module' project type, then delete the files that are only relevant to a 'module+cli' project.

Parameters **request** (*[type]*) – [description]

cookiecutter_python.hooks.post_gen_project.**post_hook**()

Delete irrelevant to Project Type files and optionally do git commit.

cookiecutter_python.hooks.post_gen_project.**run_process_python36_n_below**(**args*, ***kwargs*)

cookiecutter_python.hooks.post_gen_project.**run_process_python37_n_above**(**args*, ***kwargs*)

cookiecutter_python.hooks.post_gen_project.**subprocess_run**(**args*, ***kwargs*)

Module contents

Module contents

CHAPTER
FIVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

C

INDEX