cookiecutter-cliffapp-template Documentation

Release 2021.3.0

Dave Dittrich

Jan 22, 2023

CONTENTS:

This document (version 2021.3.0) describes the `cookiecutter-cliffapp-template`.

CHAPTER

ONE

INTRODUCTION

This chapter introduces `cookiecutter-cliffapp-template` for creating a feature-filled Git repository for a Python command line interface (CLI) that you can use to develop, test, and publish as a package on the Python Package Index.

1.1 Features

- Uses the `cookiecutter-cliffapp-template` to produce a bare-bones functional Python CLI app built on the OpenStack cliff – Command Line Interface Formulation Framework.

  `cliff` provides many useful features like modularizing subcommands into groups, built-in help for internally documenting commands, and producing output in clean tabular form or in one of several data formats you can feed into other tools or automation platforms.

- Immediately after `cookiecutter` generates the app directory, it sets up the Git repository for your new app with `main` and `develop` branches and an initial version tag, ready to push to GitHub.

  The app directory is also pre-configured with these features ready to go:

  - Sphinx documentation for generation with ReadTheDocs including `cliff` autoprogram Sphinx integration for documenting commands from the same `--help` output you get at the command line.
  - Unit testing with pytest, Python security vulnerability scanning with bandit, integration and system testing with BATS (bats-core), and Python library dependency security scanning with GitHub's dependabot.
  - Testing against Python versions 3.6, 3.7, 3.8, and 3.9 using Tox.
  - Version number bumping with a single command using bump2version and version number generation for packages and `--version` output using setuptools_scm.
  - Workflow processing for automatic testing using GitHub Actions.
  - The GitHub Actions workflow will also auto-release packages to PyPI or Test PyPI when you push a new version tag on the `main` branch or a special `rc` tag on the `develop` branch.

CHAPTER

TWO

USAGE

These instructions assume you will be doing all of the following:

- Maintaining your source code repository on GitHub and using GitHub Actions workflows for driving testing and package publication;
- Publishing full releases of your project as Python packages to PyPI and candidate or test releases to Test PyPI; and
- Publishing documentation for your project on ReadTheDocs.

Note: Future releases of this template *may* support opting out of one or more of the features currently set up by the template. Feel free to submit a Pull Request for consideration!

2.1 Quickstart

Attention:

Sticks and stones may break your bones, but namespace clashes will really mess you up!

You won't have an issue with namespace clashes between projects on GitHub where projects start with your account name (e.g., `daveditrich/python_secrets`), but PyPI, Test PyPI, and ReadTheDocs have flat namespaces and the name you want to use for your Python app may already be in use by someone else.

Before creating your new project, check to make sure *the same name is available* on all of these services.

2.1.1 Preparing your accounts

If you haven't do so yet, you will need to create an account on each of the services mentioned above. The service accounts will be linked in order for commits pushed to GitHub to in turn push out new documentation and/or packages.

TL;DR

- Create projects on ReadTheDocs, PyPI and Test PyPI
- Create tokens on PyPI and Test PyPI
- Store tokens from PyPI and Test PyPI as encrypted secrets on GitHub.

Integration of the services involved is accomplished by enabling the integration in one or both of the services involved, or by creating an access token on service `A` and storing that token on service `B` to authenticate connections at a later time from `B` back to service `A`.

2.1.2 PyPI and Test PyPI

TL;DR

- Set up your accounts (if necessary)
- Create a project
- Create a token

- Python Packaging User Guide
- Packaging Python Projects
- How to Publish an Open-Source Python Package to PyPI, by Geir Arne Hjelle, RealPython
- How to upload your python package to PyPi, by joelbarmettlerUZH, Medium, May 7, 2018

2.1.3 ReadTheDocs

TL;DR

- Set up your account (if necessary)
- Create a project
- Generate an integration to use as a webhook from GitHub

https://readthedocs.org/dashboard/import/manual/?

The `.readthedocs.yml` file includes configuration settings that control the documentation build process.

2.1.4 GitHub

TL;DR

- Create a new project
- Enable Dependabot checking on the Security tab for your project
- Set a webhook for linking to ReadTheDocs

- Setting up a PyPI project where you will release your project as a Python package, and a parallel Test PyPI project for releasing test packages.
- Setting up a GitHub repository for your project.
- Setting up a ReadTheDocs documentation project that is connected to the GitHub repository.

- Using GitHub Actions workflows for continuous integration testing and publishing your PyPI/test PyPI packages.

See also:

- Packaging Python Projects, PyPA
- How to Publish an Open-Source Python Package to PyPI, RealPython

2.2 Baking the project

Now that all the accounts are set up, you are ready to bake the project!

When you first create your Python package directory with `cookiecutter` using this template, it will prompt you to enter values for the variables below before using them to generate your initial project directory.

`author`
Your full name

`author_email`
Your email address

`github_username`
Your GitHub username

`project_name`
The name of your new Python package project. This is used to to create the namespace, the package name, and the command name you will type at the console. It should be short and only contain lowercase characters `a-z` and the dash (`-`) character.

`project_slug`
The name of your Python package on PyPi. This name will have dashes converted to underscore characters (_) for use by `import` and variable names.

`project_short_description`
A 1-sentence description of what your Python package is and does.

`release_date`
The date of the first release (*YYYY-MM-DD* format).

`project_version`
The starting version number of the package (defaults to *YYYY.MM.0* format).

`copyright_name`
Name of copyright holder (defaults to `author`).

`copyright_year`
The year of the initial package copyright in the license file (defaults to the current year).

`pypi_username`
Your Python Package Index account username for both PyPI and Test PyPI.

`license`
The chosen license.

Note: If any of these are not exactly what you need, just chose something (or accept the default) and change it after `cookiecutter` has rendered files from the template.

---

1. Miniconda provides a consistent Python virtual environment experience across Mac OSX, Windows 10, and Linux (either native, or on Windows 10 using Windows Subsystem for Linux (WSL2)).

Create a Conda environment with the version of Python you prefer for developing your new app. It's easier to remember which environment to activate if the environment name matches that of your app:

```
$ conda create -n cliffapp python=3.9
```

2. Install the latest Cookiecutter if you haven't installed it yet (Cookiecutter 1.4.0 or higher is required) into your chosen Python virtual environment:

```
$ conda activate cliffapp
$ python3 -m pip install -U cookiecutter gitpython
```

3. In the directory where you keep your Git repos, use `cookiecutter` to generate a new Git repo directory for your Python package project (changing the answers to suit your project, of course):

```
$ cd ~/git
$ python3 -m cookiecutter https://github.com/davedittrich/cookiecutter-cliffapp-
↪template.git
author [Dave Dittrich]:
author_email [dave.dittrich@gmail.com]:
github_username [davedittrich]:
project_name [cookiecutter-cliffapp]:
project_slug [cookiecutter_cliffapp]:
project_short_description [The cookiecutter-cliffapp project command line interface.
↪]:
release_date [2021-03-29]:
project_version [2021.3.0]:
copyright_name [Dave Dittrich]:
copyright_year [2021]:
pypi_username [davedittrich]:
Select license:
1 - MIT license
2 - BSD license
3 - ISC license
4 - Apache Software License 2.0
5 - GNU General Public License v3
6 - Other/Proprietary License
Choose from 1, 2, 3, 4, 5, 6 [1]: 4
$ cd cookiecutter-cliffapp
```

As you can see, the repo is all set up with `develop` and `main` branches, with the `main` branch checked out and ready to push to GitHub!:

```
$ git remote -v
origin  git@github.com:davedittrich/cookiecutter-cliffapp.git (fetch)
origin  git@github.com:davedittrich/cookiecutter-cliffapp.git (push)
$ git log
commit 42fd5b4405d54b87fc62255da47ff1cfa0449b81 (HEAD -> main, tag: v2021.3.0rc1,␣
↪develop)
Author: Dave Dittrich <dave.dittrich@gmail.com>
Date:   Mon Mar 29 19:18:40 2021 -0700

    Initial commit
```

4. Push the repo branches:

- Using Git hubflow:

```
$ git hf init
Using default branch names.

Which branch should be used for tracking production releases?
   - develop
   - main
Branch name for production releases: [main]

Which branch should be used for integration of the "next release"?
   - develop
Branch name for "next release" development: [develop]

How to name your supporting branch prefixes?
Feature branches? [feature/]
Release branches? [release/]
Hotfix branches? [hotfix/]
Support branches? [support/]
Version tag prefix? []
. . .
To github.com:davedittrich/cookiecutter-cliffapp.git
 * [new branch]      main -> main
```

- Using Git commands directly:

```
$ git push -u origin master
. . .
$ git checkout develop
$ git push -u origin develop
. . .
```

The pushes should trigger GitHub Actions workflows, which should pass all tests (but not trigger any release publication at this point.)

5. Manually release your first test package to Test PyPI from the new repo directory. This initializes the project (which needs to be done *before* you can create API tokens). You can use this command:

```
$ make release-test
. . .
twine upload dist/* -r testpypi
Uploading distributions to https://test.pypi.org/legacy/
Uploading cookiecutter_cliffapp-2021.3.0rc1-py2.py3-none-any.whl
100%| extblock extblock extblock extblock extblock extblock extblock extblock
extblock extblock extblock extblock extblock extblock extblock extblock extblock
extblock extblock extblock extblock extblock extblock extblock extblock extblock
extblock extblock extblock extblock extblock extblock extblock extblock extblock
extblock extblock extblock extblock extblock extblock extblock extblock extblock
extblock extblock extblock extblock extblock extblock extblock extblock extblock
extblock extblock extblock extblock extblock extblock extblock extblock extblock
extblock|
19.6k/19.6k [00:01<00:00, 14.1kB/s]
Uploading cookiecutter_cliffapp-2021.3.0rc1.tar.gz
100%| extblock extblock extblock extblock extblock extblock extblock extblock
```

(continues on next page)

(continued from previous page)

```
extblock extblock extblock extblock extblock extblock extblock extblock extblock
extblock extblock extblock extblock extblock extblock extblock extblock extblock
extblock extblock extblock extblock extblock extblock extblock extblock extblock
extblock extblock extblock extblock extblock extblock extblock extblock extblock
extblock extblock extblock extblock extblock extblock extblock extblock extblock
extblock extblock extblock extblock extblock extblock extblock extblock extblock
extblock|
32.8k/32.8k [00:01<00:00, 27.0kB/s]

View at:
https://test.pypi.org/project/cookiecutter-cliffapp/2021.3.0rc1/
```

6. Once the new project has been created on Test PyPI, log in, select the project, choose **Settings**, then choose **Create a token for <yourprojectname>**. Token names must be unique to your account, so use part of your project name (e.g., `CLIFFAPP_TEST_PYPI_PASSWORD`) to differentiate this token from those for your other projects. Limit the scope of the token to just this project and then **Add token**. Note that you will only be able to see the token value once.

   Copy the token and paste it into the **Value** field for a new token (using the same name) in an encrypted secret in your GitHub project window to be used in the GitHub Actions workflow.

   Test the workflow by creating a new tag with `rc` in the name (e.g., `v2021.3.0rc2`) on the `develop` branch and doing `git push --tags`, which will then automatically trigger a workflow. The result should be that the publish portion pushes a new package to your Test PyPI project.

   When you are comfortable that tagging and publishing release candidates to Test PyPI is working smoothly, repeat the token creation and storage steps for PyPI (this time using `CLIFFAPP_PYPI_PASSWORD` as the token name). Check out the `master` branch and use `make release` to push the initial package and create the project. From then on, when you create a new release version tag (e.g., `v2021.3.1`) on the `master` branch and push to GitHub, the GitHub Actions workflow will publish the package on PyPI (after the tests succeed, of course).

Now that things are up and running, see *Development Lifecycle Tasks*.

CHAPTER

THREE

DEVELOPMENT LIFECYCLE TASKS

This section covers the tasks at various stages in the lifecycle of a Python project created using the `cookiecutter-cliffapp-template`.

Note: If you have not set things up yet, see *Usage* for getting started and then come back here.

3.1 Development Testing

You will want to regularly test your code in one or more ways. The simplest and most frequent checks you will want to do are `pep8` and `bandit` tests to ensure your code is clean and free of security bugs.

There are several levels of testing you may want to perform.

- You can perform limited code quality tests of your source code and documentation using `tox` directly like this:

```
$ tox -e pep8,bandit,docs
. . .
pep8: commands succeeded
bandit: commands succeeded
docs: commands succeeded
congratulations :)
```

- You can do unit testing and runtime integration testing of your code against several versions of Python using `tox` directly like this:

```
$ tox -e py36,py37,py38,py39
```

- You can do runtime integration testing of your code using `bats` via `tox` like this:

```
$ tox -e bats
```

- You can testing of the full Python package using `twine` via `tox` like this:

```
$ tox -e pypi
```

- You can also use `make` to invoke some or all of the above by specifying one or more targets from the `Makefile`. Use `make help` to see a list of available targets, or read the file with an editor to see the rules. The target for running all of the tests, first code quality, then more expensive unit, integration, and package tests, with this command:

```
$ make test
```

When all of the latter tests pass, you can push your commits to GitHub. If you forget to do this, the GitHub Action workflow will do it for you and any failures will result in email messages informing you of the failures.

3.2 Releasing on PyPI

3.2.1 For Every Release

In addition to making sure that your code passes all of the tests covered in the last section, you will also want to update documentation, bump version numbers, merge branches, etc.

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:

```
$ make test
```

6. Push the commit:

```
$ git push
```

7. Push the tags, creating the new release on both GitHub and PyPI:

```
$ git push --tags
```

CHAPTER

FOUR

CREDITS

4.1 Development Lead

- Dave Dittrich <dave.dittrich@gmail.com>

4.2 Contributors

None yet. Why not be the first?

CHAPTER

FIVE

LICENSE

```
Copyright (c) 2021, Dave Dittrich. All rights reserved.
Author: <dave.dittrich@gmail.com>

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

   http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
```

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search