generic-template

Release 0.0.0

Kyle Finley

Sep 02, 2022

DOCUMENTATION

Generic project template to go from zero to developing in seconds.

CHAPTER

ONE

EXAMPLE PAGE

Plain text with some `inline code`.

> quote block

```
"""Showcase custom pygments style."""
# this is just an example
import boto3

client = boto3.client('ec2')
print('client.describe_instances')
print('test\ntest')

def function(**kwarg):
    """placeholder."""

class Test:
    """placeholder."""

    def __init__(self):
        """placeholder."""
        pass
```

Header 1	Header 2	Header 3
body row 1	column 2	column 3
body row 2	Cells may span columns.	
body row 3	Cells may span rows.	• Cells • contain • blocks.
body row 4		

Note: This is a note.

Important: This is important.

Warning: This is a warning.

Danger: This is dangerous.

1. list item

- another list

CHAPTER

TWO

API DOCS

2.1 generic_template package

Blank file for Python transversal.

This can be deleted or renamed to store the source code of your project.

CHAPTER

THREE

BRANCHING STRATEGY

3.1 Master Branch

The **default branch** of this project is `master`.

This branch can also be thought of as the *latest* branch in that it contains all the latest changes. It can also be through of as the *development* branch in that it should be an exact match for what is currently deployed to a development and/or test environment where applicable. All branches should be based off of this branch and should be merged into this branch when a PR has been approved.

The contents of this branch should be continuously deployed to a development and/or test environment where applicable.

Releases are created from this branch by creating a tag. The tag should be in the format `v${MAJOR}.${MINOR}.${PATH}` following SemVer. The tagged release should be deployed to a production environment where applicable.

3.2 Feature Branches

A **feature branch** is a branch where one or more developers are actively developing a single *change*. This change can be a new feature/functionality, bug fix/patch, documentation update, or maintenance task.

A feature branch must be based off of the *Master Branch* and must be merged into the *Master Branch* when complete. Changes to the *Master Branch* should be integrated daily.

A feature branch can be deployed to a development environment where applicable.

3.2.1 Feature Branch Naming

When creating a feature branch, it must be named in a specific way for GitHub Actions to handle it correctly. The naming format is `${PREFIX}/${DESCRIPTION_OF_CHANGE}`.

Prefixes

bugfix | fix | hotfix
: The branch contains a fix for a big.

feature | feat
: The branch contains a new feature or enhancement to an existing feature.

docs | documentation
: The branch only contains updates to documentation.

chore | maintain | maint | maintenance
: The branch does not contain changes to the project itself to is aimed at maintaining the repo, CI/CD, or testing infrastructure. (e.g. README, GitHub action, integration test infrastructure)

release
: Reserved for maintainers to prepare for the release of a new version.

CHAPTER

FOUR

DOCUMENTATION STYLE GUIDE

Style guides to follow when writing documentation for this project.

4.1 Docstrings

In general, this project loosely follows the Google Python Style Guide for Comments and Docstrings.

We use the `napoleon` extension for Sphinx to parse docstrings. Napoleon provides an Example Google Style Python Docstring that can be referenced.

4.2 reStructuredText Formatting

In general, this project loosely follows the Python Style Guide for documentation.

4.2.1 reStructuredText Whitespace

All reST files use an indentation of 2 spaces; no tabs are allowed. There is no maximum line length but 80 characters is the recommended line length for normal text. It is preferred to only break a line at the end of a sentence.

Code example bodies should use normal indentation based on the convention of the language.

Make generous use of blank lines where applicable; they help group things together.

4.2.2 reStructuredText Sections

Section headers are created by underlining (and optionally overlining) the section title with a punctuation character, at least as long as the text.

1. `#` with overline, for **h1** (generally only one per file, at the top of the file)
2. `*` with overline, for **h2**
3. `=`, for **h3**
4. `-`, for **h4**
5. `^`, for **h5**
6. `"`, for **h6**

h1 and **h2** should have two or more blank lines separating them from sections with headings of the same level or higher.

A `rubric` directive can be used to create a non-indexed heading.

Example

```
########
Heading 1
########


*********
Heading 2
*********

Heading 3
=========

Heading 4
---------

Heading 5
^^^^^^^^^

Heading 6
"""""""""""

.. rubric:: Non-indexed Heading


*********
Heading 2
*********

Heading 3
=========
```

CHAPTER

FIVE

GETTING STARTED

Before getting started, fork this repo and clone your fork.

5.1 Local Development Environment

Setting up a workstation of development of this project.

5.1.1 Prerequisites

Some of the listed prerequisites are *recommended* but not required.

- Make (GNU recommended) for simplified actions
  - `brew install make` on macOS
  - `sudo apt install make` or `sudo apt install build-essential` or Ubuntu/Debian
  - `winget install -e --id GnuWin32.Make` on Windows
- npm for type checking & spell check (recommended to use nvm to install)
- poetry for Python virtual environments
- *(recommended)* direnv for setting environment variables (POSIX only)
- *(recommended)* nvm to install and manage different versions of node (POSIX only)
  - see nvm-windows for Windows support
- *(recommended)* pyenv to install and manage different versions of Python (POSIX Only)
  - see pyenv-win for Windows support
- *(recommended)* Visual Studio Code for standardized IDE settings

5.1.2 Setup

1. Clone the repo or your fork of this repo.
2. Change directory into the cloned directory.
3. Run `make setup`.
4. Start developing.

By running `make setup`, the following will happen:

1. poetry is used to setup a Python virtual environment.
2. pre-commit is configured from the virtual environment to run basic checks and formatting when a commit is made. These checks can be run manually using `make run-pre-commit`.
3. node dependencies are installed.

PYTHON MODULE INDEX

INDEX