# Git Portfolio

staticdev

Dec 18, 2022

# CONTENTS

1 Features 3

2 Requirements 5

3 Installation 7

4 Basic usage 9

5 Contributing 11

6 License 13

7 Issues 15

8 Credits 17

Python Module Index 31

Index 33

# CHAPTER ONE

# FEATURES

- Configure multiple working repositories.
- Batch git commands with subcommands: `add`, `branch`, `checkout`, `commit`, `diff`, `fetch`, `init`, `merge`, `mv`, `pull`, `push`, `rebase`, `reset`, `rm`, `show`, `switch`, `status` and `tag`.
- Batch API calls on GitHub: create/close/reopen `issues`, create/close/reopen/merge `pull requests` and delete `branches` by name.
- Batch Poetry commands such as: `add`, `version patch`, `install` or `update`.

# CHAPTER TWO

# REQUIREMENTS

- Create an auth token for GitHub, with the `repo` privileges enabled by clicking on Generate new token. You will be asked to select scopes for the token. Which scopes you choose will determine what information and actions you will be able to perform against the API. You should be careful with the ones prefixed with write:, delete: and admin: as these might be quite destructive. You can find description of each scope in docs here.

Important: safeguard your token (once created you won't be able to see it again).

- Install git (optional) - this is needed for all git commands. For colored outputs please use the configuration:

```
$ git config --global color.ui always
```

CHAPTER

THREE

# INSTALLATION

You can install *Git Portfolio* via pip from PyPI:

```
$ pip install git-portfolio
```

CHAPTER

# FOUR

# BASIC USAGE

1. Create initial configuration with:

```
$ gitp config init
```

2. Execute all the commands you want. Eg.:

```
$ gitp issues create  # create same issue for all projects
$ gitp checkout -b new-branch  # checks out new branch new-branch in all projects
$ gitp poetry version minor  # bumps minor version of all projects that have pyproject.
↪toml version
```

Note: by convention GitHub commands are always the resource name and action: eg. `branches delete`, `issues create` and `prs merge` (for pull requests). This avoid conflicts with batch git commands, as in `gitp branch` (executes git command) and `gitp branches delete` (execute operations using GitHub API).

Complete instructions can be found at git-portfolio.readthedocs.io.

# CHAPTER FIVE

# CONTRIBUTING

Contributions are very welcome. To learn more, see the Contributor Guide.

# CHAPTER SIX

# LICENSE

Distributed under the terms of the MIT license, *Git Portfolio* is free and open source software.

# CHAPTER SEVEN

# ISSUES

If you encounter any problems, please file an issue along with a detailed description.

# CHAPTER EIGHT

# CREDITS

This project was generated from @cjolowicz's Hypermodern Python Cookiecutter template.

## 8.1 Usage

### 8.1.1 Basic usage

1. Create initial configuration with:

```
$ gitp config init
```

2. Execute all the commands you want. Eg.:

```
$ gitp issues create  # create same issue for all projects
$ gitp checkout -b new-branch  # checks out new branch new-branch in all projects
$ gitp poetry version minor  # bumps minor version of all projects that have pyproject.
↪toml version
```

Note: by convention GitHub commands are always the resource name and action: eg. `branches delete`, `issues create` and `prs merge` (for pull requests). This avoid conflicts with batch git commands, as in `gitp branch` (executes git command) and `gitp branches delete` (execute operations using GitHub API).

### 8.1.2 Complete usage

#### gitp

Git Portfolio.

```
gitp [OPTIONS] COMMAND [ARGS]...
```

#### Options

**--version**
Show the version and exit.

### add

Batch *git add* command.

```
gitp add [OPTIONS] [ARGS]...
```

#### Arguments

**ARGS**
Optional argument(s)

### branch

Batch *git branch* command.

```
gitp branch [OPTIONS] [ARGS]...
```

#### Arguments

**ARGS**
Optional argument(s)

### branches

Branches command group.

```
gitp branches [OPTIONS] COMMAND [ARGS]...
```

#### delete

Batch deletion of branches on GitHub.

```
gitp branches delete [OPTIONS]
```

### checkout

Batch *git checkout* command.

```
gitp checkout [OPTIONS] [ARGS]...
```

#### Arguments

**ARGS**

Optional argument(s)

### clone

Batch *git clone* command on current folder. Does not accept aditional args.

```
gitp clone [OPTIONS]
```

### commit

Batch *git commit* command.

```
gitp commit [OPTIONS] [ARGS]...
```

#### Arguments

**ARGS**

Optional argument(s)

### config

Config command group.

```
gitp config [OPTIONS] COMMAND [ARGS]...
```

#### init

Initialize *gitp* config.

```
gitp config init [OPTIONS]
```

#### repos

Configure current working *gitp* repositories.

```
gitp config repos [OPTIONS]
```

### diff

Batch *git diff* command.

```
gitp diff [OPTIONS] [ARGS]...
```

#### Arguments

**ARGS**
: Optional argument(s)

### fetch

Batch *git fetch* command.

```
gitp fetch [OPTIONS] [ARGS]...
```

#### Arguments

**ARGS**
: Optional argument(s)

### init

Batch *git init* command.

```
gitp init [OPTIONS] [ARGS]...
```

#### Arguments

**ARGS**
: Optional argument(s)

### issues

Issues command group.

```
gitp issues [OPTIONS] COMMAND [ARGS]...
```

#### close

Batch close issues on GitHub.

```
gitp issues close [OPTIONS]
```

#### create

Batch creation of issues on GitHub.

```
gitp issues create [OPTIONS]
```

#### reopen

Batch reopen issues on GitHub.

```
gitp issues reopen [OPTIONS]
```

### merge

Batch *git merge* command.

```
gitp merge [OPTIONS] [ARGS]...
```

#### Arguments

**ARGS**

Optional argument(s)

### mv

Batch *git mv* command.

```
gitp mv [OPTIONS] [ARGS]...
```

#### Arguments

**ARGS**

Optional argument(s)

### poetry

Batch *poetry* command.

```
gitp poetry [OPTIONS] [ARGS]...
```

#### Arguments

**ARGS**

Optional argument(s)

### prs

Pull requests command group.

```
gitp prs [OPTIONS] COMMAND [ARGS]...
```

#### close

Batch close pull requests on GitHub.

```
gitp prs close [OPTIONS]
```

#### create

Batch creation of pull requests on GitHub.

```
gitp prs create [OPTIONS]
```

#### merge

Batch merge of pull requests on GitHub.

```
gitp prs merge [OPTIONS]
```

#### reopen

Batch reopen pull requests on GitHub.

```
gitp prs reopen [OPTIONS]
```

### pull

Batch *git pull* command.

```
gitp pull [OPTIONS] [ARGS]...
```

#### Arguments

**ARGS**
Optional argument(s)

### push

Batch *git push* command.

```
gitp push [OPTIONS] [ARGS]...
```

#### Arguments

**ARGS**
Optional argument(s)

### rebase

Batch *git rebase* command.

```
gitp rebase [OPTIONS] [ARGS]...
```

#### Arguments

**ARGS**
Optional argument(s)

### reset

Batch *git reset* command.

```
gitp reset [OPTIONS] [ARGS]...
```

#### Arguments

**ARGS**

Optional argument(s)

### rm

Batch *git rm* command.

```
gitp rm [OPTIONS] [ARGS]...
```

#### Arguments

**ARGS**

Optional argument(s)

### show

Batch *git show* command.

```
gitp show [OPTIONS] [ARGS]...
```

#### Arguments

**ARGS**

Optional argument(s)

### status

Batch *git status* command.

```
gitp status [OPTIONS] [ARGS]...
```

#### Arguments

**ARGS**

Optional argument(s)

### switch

Batch *git switch* command.

```
gitp switch [OPTIONS] [ARGS]...
```

#### Arguments

**ARGS**

Optional argument(s)

### tag

Batch *git tag* command.

```
gitp tag [OPTIONS] [ARGS]...
```

#### Arguments

**ARGS**

Optional argument(s)

## 8.2 Reference

### 8.2.1 git_portfolio

Git Portfolio.

## 8.3 Contributor Guide

Thank you for your interest in improving this project. This project is open-source and welcomes contributions in the form of bug reports, feature requests, and pull requests.

Here is a list of important resources for contributors:

- Source Code
- Documentation
- Issue Tracker
- Code of Conduct

### 8.3.1 How to report a bug

Report bugs on the Issue Tracker.

When filing an issue, make sure to answer these questions:

- Which operating system and Python version are you using?
- Which version of this project are you using?
- What did you do?
- What did you expect to see?
- What did you see instead?

The best way to get your bug fixed is to provide a test case, and/or steps to reproduce the issue.

### 8.3.2 How to request a feature

Request features on the Issue Tracker.

### 8.3.3 How to set up your development environment

You need Python 3.9+ and the following tools:

- Poetry
- Nox

Install the package with development requirements:

```
$ poetry install
```

You can now run an interactive Python session, or the command-line interface:

```
$ poetry run python
$ poetry run git-portfolio
```

### 8.3.4 How to test the project

Run the full test suite:

```
$ nox
```

List the available Nox sessions:

```
$ nox --list-sessions
```

You can also run a specific Nox session. For example, invoke the unit test suite like this:

```
$ nox --session=tests
```

Unit tests are located in the `tests` directory, and are written using the pytest testing framework.

### 8.3.5 How to submit changes

Open a pull request to submit changes to this project.

Your pull request needs to meet the following guidelines for acceptance:

- The Nox test suite must pass without errors and warnings.
- Include unit tests. This project maintains 100% code coverage.
- If your changes add functionality, update the documentation accordingly.

Feel free to submit early, though—we can always iterate on this.

To run linting and code formatting checks before commiting your change, you can install pre-commit as a Git hook by running the following command:

```
$ nox --session=pre-commit -- install
```

It is recommended to open an issue before starting work on anything. This will allow a chance to talk it over with the owners and validate your approach.

## 8.4 Contributor Covenant Code of Conduct

### 8.4.1 Our Pledge

We as members, contributors, and leaders pledge to make participation in our community a harassment-free experience for everyone, regardless of age, body size, visible or invisible disability, ethnicity, sex characteristics, gender identity and expression, level of experience, education, socio-economic status, nationality, personal appearance, race, religion, or sexual identity and orientation.

We pledge to act and interact in ways that contribute to an open, welcoming, diverse, inclusive, and healthy community.

### 8.4.2 Our Standards

Examples of behavior that contributes to a positive environment for our community include:

- Demonstrating empathy and kindness toward other people
- Being respectful of differing opinions, viewpoints, and experiences
- Giving and gracefully accepting constructive feedback
- Accepting responsibility and apologizing to those affected by our mistakes, and learning from the experience
- Focusing on what is best not just for us as individuals, but for the overall community

Examples of unacceptable behavior include:

- The use of sexualized language or imagery, and sexual attention or advances of any kind
- Trolling, insulting or derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or email address, without their explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

### 8.4.3 Enforcement Responsibilities

Community leaders are responsible for clarifying and enforcing our standards of acceptable behavior and will take appropriate and fair corrective action in response to any behavior that they deem inappropriate, threatening, offensive, or harmful.

Community leaders have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, and will communicate reasons for moderation decisions when appropriate.

### 8.4.4 Scope

This Code of Conduct applies within all community spaces, and also applies when an individual is officially representing the community in public spaces. Examples of representing our community include using an official e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event.

### 8.4.5 Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported to the community leaders responsible for enforcement at staticdev-support@protonmail.com. All complaints will be reviewed and investigated promptly and fairly.

All community leaders are obligated to respect the privacy and security of the reporter of any incident.

### 8.4.6 Enforcement Guidelines

Community leaders will follow these Community Impact Guidelines in determining the consequences for any action they deem in violation of this Code of Conduct:

#### 1. Correction

**Community Impact**: Use of inappropriate language or other behavior deemed unprofessional or unwelcome in the community.

**Consequence**: A private, written warning from community leaders, providing clarity around the nature of the violation and an explanation of why the behavior was inappropriate. A public apology may be requested.

#### 2. Warning

**Community Impact**: A violation through a single incident or series of actions.

**Consequence**: A warning with consequences for continued behavior. No interaction with the people involved, including unsolicited interaction with those enforcing the Code of Conduct, for a specified period of time. This includes avoiding interactions in community spaces as well as external channels like social media. Violating these terms may lead to a temporary or permanent ban.

#### 3. Temporary Ban

**Community Impact**: A serious violation of community standards, including sustained inappropriate behavior.

**Consequence**: A temporary ban from any sort of interaction or public communication with the community for a specified period of time. No public or private interaction with the people involved, including unsolicited interaction with those enforcing the Code of Conduct, is allowed during this period. Violating these terms may lead to a permanent ban.

#### 4. Permanent Ban

**Community Impact**: Demonstrating a pattern of violation of community standards, including sustained inappropriate behavior, harassment of an individual, or aggression toward or disparagement of classes of individuals.

**Consequence**: A permanent ban from any sort of public interaction within the community.

### 8.4.7 Attribution

This Code of Conduct is adapted from the Contributor Covenant, version 2.0, available at https://www.contributor-covenant.org/version/2/0/code_of_conduct/.

Community Impact Guidelines were inspired by Mozilla's code of conduct enforcement ladder.

For answers to common questions about this code of conduct, see the FAQ at https://www.contributor-covenant.org/faq. Translations are available at https://www.contributor-covenant.org/translations.

# PYTHON MODULE INDEX

g

git_portfolio, 25

# INDEX

## Symbols

--version
    gitp command line option, 18

## A

ARGS
    gitp-add command line option, 18
    gitp-branch command line option, 18
    gitp-checkout command line option, 19
    gitp-commit command line option, 19
    gitp-diff command line option, 20
    gitp-fetch command line option, 20
    gitp-init command line option, 20
    gitp-merge command line option, 21
    gitp-mv command line option, 22
    gitp-poetry command line option, 22
    gitp-pull command line option, 23
    gitp-push command line option, 23
    gitp-rebase command line option, 23
    gitp-reset command line option, 24
    gitp-rm command line option, 24
    gitp-show command line option, 24
    gitp-status command line option, 25
    gitp-switch command line option, 25
    gitp-tag command line option, 25

## G

git_portfolio
    module, 25
gitp command line option
    --version, 18
gitp-add command line option
    ARGS, 18
gitp-branch command line option
    ARGS, 18
gitp-checkout command line option
    ARGS, 19
gitp-commit command line option
    ARGS, 19
gitp-diff command line option
    ARGS, 20
gitp-fetch command line option
    ARGS, 20
gitp-init command line option
    ARGS, 20
gitp-merge command line option
    ARGS, 21
gitp-mv command line option
    ARGS, 22
gitp-poetry command line option
    ARGS, 22
gitp-pull command line option
    ARGS, 23
gitp-push command line option
    ARGS, 23
gitp-rebase command line option
    ARGS, 23
gitp-reset command line option
    ARGS, 24
gitp-rm command line option
    ARGS, 24
gitp-show command line option
    ARGS, 24
gitp-status command line option
    ARGS, 25
gitp-switch command line option
    ARGS, 25
gitp-tag command line option
    ARGS, 25

## M

module
    git_portfolio, 25