# poetry-up

**Claudio Jolowicz**

**Jan 28, 2021**

# CONTENTS


1 Reference 1

2 Contributor Guide 5

3 Contributor Covenant Code of Conduct 7

4 License 11

5 Installation 13

6 Usage 15

Python Module Index 17

Index 19

CHAPTER

ONE

# REFERENCE


- *poetry_up.console*
- *poetry_up.git*
- *poetry_up.github*
- *poetry_up.poetry*
- *poetry_up.update*


## 1.1 poetry_up.console

Command-line interface.

## 1.2 poetry_up.git

Git wrapper.

**class** `poetry_up.git.`**`MergeRequest`**(*title: str*, *description: str*)

Merge request with a title and a description.

`poetry_up.git.`**`add`**(*paths*)

Add the specified paths to the index.

**Return type** `None`

`poetry_up.git.`**`branch_exists`**(*branch*)

Return True if the branch exists.

**Return type** `bool`

`poetry_up.git.`**`commit`**(*message*)

Create a commit using the given message.

**Return type** `None`

`poetry_up.git.`**`current_branch`**()

Return the checked out branch.

**Return type** `str`

`poetry_up.git.`**`git`**(**args*, *check=True*)
Invoke git.

**Return type** `CompletedProcess`

`poetry_up.git.`**`is_clean`**(*paths=()*)
Return True if the working tree, or the given files, are clean.

**Return type** `bool`

`poetry_up.git.`**`push`**(*remote*, *branch*, *merge_request=None*)
Push the branch to the remote.

**Parameters**

- **`remote`** (`str`) – The remote to push to.
- **`branch`** (`str`) – The branch to be pushed.
- **`merge_request`** (`Optional[MergeRequest]`) – The merge request to create for the branch (optional).

**Return type** `None`

`poetry_up.git.`**`remove_branch`**(*branch*)
Remove the specified branch.

**Return type** `None`

`poetry_up.git.`**`resolve_branch`**(*branch*)
Return the SHA1 hash for the given branch.

**Return type** `str`

`poetry_up.git.`**`switch`**(*branch*, *create=False*, *location=None*)
Switch to the specified branch.

**Parameters**

- **`branch`** (`str`) – The branch to be switched to.
- **`create`** (`bool`) – Create the branch.
- **`location`** (`Optional[str]`) – The location at which the branch should be created.

**Return type** `None`

## 1.3 poetry_up.github

GitHub wrapper.

`poetry_up.github.`**`create_pull_request`**(*title*, *body*)
Create a pull request for the checked out branch.

**Return type** `None`

`poetry_up.github.`**`pull_request_exists`**(*branch*)
Return True if a pull request exists for the given branch.

**Return type** `bool`

## 1.4 poetry_up.poetry

Poetry wrapper.

**class** poetry_up.poetry.**Package**(*name: str*, *old_version: str*, *new_version: str*)
Package with current and available versions.

poetry_up.poetry.**show_outdated**()
Yield outdated packages.

**Return type** Iterator[*Package*]

poetry_up.poetry.**update**(*package*, *lock=False*, *latest=False*)
Update the given package.

**Parameters**

- **package** (*Package*) – The package to be updated.
- **lock** (bool) – If True, do not install the package into the environment.
- **latest** (bool) – If True, update the version constraint when required.

**Return type** None

## 1.5 poetry_up.update

Update module.

**class** poetry_up.update.**Action**(*updater*)
Base class for actions.

**property required**
Return True if the action needs to run.

**Return type** bool

**class** poetry_up.update.**Actions**(*switch: poetry_up.update.Switch*, *update: poetry_up.update.Update*, *commit: poetry_up.update.Commit*, *rollback: poetry_up.update.Rollback*, *push: poetry_up.update.Push*, *pull_request: poetry_up.update.PullRequest*)
Actions for a package update.

**classmethod create**(*updater*)
Create the package update actions.

**Return type** *Actions*

**class** poetry_up.update.**Commit**(*updater*)
Create a Git commit for the update.

**property required**
Return True if the action needs to run.

**Return type** bool

**class** poetry_up.update.**Options**(*latest: bool, install: bool, commit: bool, push: bool, merge_request: bool, pull_request: bool, upstream: str, remote: str, dry_run: bool, packages: Tuple[str, ...]*)
Options for the update operation.

**class** poetry_up.update.**PackageUpdater**(*package*, *options*, *original_branch*)
Update a package.

**property required**
Return True if the package needs to be updated.

**Return type** bool

**run**()
Run the package update.

**Return type** None

**show**()
Print information about the package update.

**Return type** None

**class** poetry_up.update.**PullRequest**(*updater*)
Open a pull request for the update branch.

**property required**
Return True if the action needs to run.

**Return type** bool

**class** poetry_up.update.**Push**(*updater*)
Push the update branch to the remote repository.

**property required**
Return True if the action needs to run.

**Return type** bool

**class** poetry_up.update.**Rollback**(*updater*)
Rollback an attempted package update.

**property required**
Return True if the action needs to run.

**Return type** bool

**class** poetry_up.update.**Switch**(*updater*)
Switch to the update branch.

**property required**
Return True if the action needs to run.

**Return type** bool

**class** poetry_up.update.**Update**(*updater*)
Update the package using Poetry.

**class** poetry_up.update.**Updater**(*options*)
Update packages.

**run**()
Run the package updates.

**Return type** None

CHAPTER

TWO

# CONTRIBUTOR GUIDE

Thank you for your interest in improving this project. This project is open-source under the MIT License 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*

## 2.1 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.

## 2.2 How to request a feature

Request features on the Issue Tracker.

## 2.3 How to set up your development environment

You need Python 3.6+ 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 poetry-up
```

## 2.4 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.

## 2.5 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.

You can ensure that your changes adhere to the code style by reformatting with Black:

```
$ nox --session=black
```

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.

CHAPTER

THREE

# CONTRIBUTOR COVENANT CODE OF CONDUCT

## 3.1 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.

## 3.2 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

## 3.3 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.

## 3.4 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.

## 3.5 Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported to the community leaders responsible for enforcement at mail@claudiojolowicz.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.

## 3.6 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:

### 3.6.1 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.

### 3.6.2 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.

### 3.6.3 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.

### 3.6.4 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.

## 3.7 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.html.

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.

CHAPTER

FOUR

# LICENSE

MIT License

Copyright (c) 2020 Claudio Jolowicz

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Command-line tool for upgrading Python dependencies using Poetry.

By default, this tool determines outdated dependencies using `poetry show --outdated`, and performs the following actions for every reported package:

1. Switch to a new branch `poetry-up/<package>-<version>`.
2. Update the dependency with `poetry update`. For incompatible updates, also update `pyproject.toml`.
3. Commit the changes to pyproject.toml and poetry.lock.
4. Push to origin (optional).
5. Open a pull request or merge request (optional).

If no packages are specified on the command-line, all outdated dependencies are upgraded.

CHAPTER

FIVE

# INSTALLATION

To install poetry-up, run this command in your terminal:

```
$ pip install poetry-up
```

CHAPTER

SIX

# USAGE

poetry-up's usage looks like:

```
$ poetry-up [<options>] [<packages>]
```

**--install**
Install dependency into virtual environment. This is the default behavior.

**--no-install**
Do not install dependency into virtual environment.

**--commit**
Commit the changes to Git. This is the default behavior.

**--no-commit**
Do not commit the changes to Git.

**--push**
Push the changes to the remote repository.

**--no-push**
Do not push the changes to the remote repository. This is the default behavior.

**--merge-request**
Open a merge request.

**--no-merge-request**
Do not open a merge request. This is the default behavior.

**--pull-request**
Open a pull request.

**--no-pull-request**
Do not open a pull request. This is the default behavior.

**--upstream** `<branch>`, **-u** `<branch>`
Specify the upstream branch. By default, branches are created off the master branch.

**--remote** `<remote>`, **-r** `<remote>`
Specify the remote to push to. By default, branches are pushed to origin.

**-C** `<directory>`, **--cwd** `<directory>`
Change to this directory before performing any actions.

**-n, --dry-run**
Just show what would be done.

**--version**
Display the version and exit.

**`--help`**
Display a short usage message and exit.

# PYTHON MODULE INDEX

## p

poetry_up.console, 1
poetry_up.git, 1
poetry_up.github, 2
poetry_up.poetry, 3
poetry_up.update, 3

# INDEX

## Symbols


-C <directory>
    command line option, 15
--commit
    command line option, 15
--cwd <directory>
    command line option, 15
--dry-run
    command line option, 15
--help
    command line option, 15
--install
    command line option, 15
--merge-request
    command line option, 15
--no-commit
    command line option, 15
--no-install
    command line option, 15
--no-merge-request
    command line option, 15
--no-pull-request
    command line option, 15
--no-push
    command line option, 15
--pull-request
    command line option, 15
--push
    command line option, 15
--remote <remote>
    command line option, 15
--upstream <branch>
    command line option, 15
--version
    command line option, 15
-n
    command line option, 15
-r <remote>
    command line option, 15
-u <branch>
    command line option, 15


## A


Action (*class in poetry_up.update*), 3
Actions (*class in poetry_up.update*), 3
add() (*in module poetry_up.git*), 1


## B


branch_exists() (*in module poetry_up.git*), 1


## C


command line option
    -C <directory>, 15
    --commit, 15
    --cwd <directory>, 15
    --dry-run, 15
    --help, 15
    --install, 15
    --merge-request, 15
    --no-commit, 15
    --no-install, 15
    --no-merge-request, 15
    --no-pull-request, 15
    --no-push, 15
    --pull-request, 15
    --push, 15
    --remote <remote>, 15
    --upstream <branch>, 15
    --version, 15
    -n, 15
    -r <remote>, 15
    -u <branch>, 15
Commit (*class in poetry_up.update*), 3
commit() (*in module poetry_up.git*), 1
create() (*poetry_up.update.Actions class method*), 3
create_pull_request() (*in module poetry_up.github*), 2
current_branch() (*in module poetry_up.git*), 1


## G


git() (*in module poetry_up.git*), 1


## I


is_clean() (*in module poetry_up.git*), 2

## M

`MergeRequest` (*class in poetry_up.git*), 1

## O

`Options` (*class in poetry_up.update*), 3

## P

`Package` (*class in poetry_up.poetry*), 3
`PackageUpdater` (*class in poetry_up.update*), 3
`poetry_up.console` (*module*), 1
`poetry_up.git` (*module*), 1
`poetry_up.github` (*module*), 2
`poetry_up.poetry` (*module*), 3
`poetry_up.update` (*module*), 3
`pull_request_exists()` (*in module poetry_up.github*), 2
`PullRequest` (*class in poetry_up.update*), 4
`Push` (*class in poetry_up.update*), 4
`push()` (*in module poetry_up.git*), 2

## R

`remove_branch()` (*in module poetry_up.git*), 2
`required()` (*poetry_up.update.Action property*), 3
`required()` (*poetry_up.update.Commit property*), 3
`required()` (*poetry_up.update.PackageUpdater property*), 4
`required()` (*poetry_up.update.PullRequest property*), 4
`required()` (*poetry_up.update.Push property*), 4
`required()` (*poetry_up.update.Rollback property*), 4
`required()` (*poetry_up.update.Switch property*), 4
`resolve_branch()` (*in module poetry_up.git*), 2
`Rollback` (*class in poetry_up.update*), 4
`run()` (*poetry_up.update.PackageUpdater method*), 4
`run()` (*poetry_up.update.Updater method*), 4

## S

`show()` (*poetry_up.update.PackageUpdater method*), 4
`show_outdated()` (*in module poetry_up.poetry*), 3
`Switch` (*class in poetry_up.update*), 4
`switch()` (*in module poetry_up.git*), 2

## U

`Update` (*class in poetry_up.update*), 4
`update()` (*in module poetry_up.poetry*), 3
`Updater` (*class in poetry_up.update*), 4