poetry-poems

Release 0.2.2

Jakub Oles

Oct 08, 2022

CONTENTS:

CHAPTER

ONE

POEMS

Poetry Environment Switcher

1.1 Overview

Poetry-poems is a tool that speeds up switching between Python Poetry-based projects by:

- navigating to a specific project in the terminal
- activating Poetry shell at the same time.

Poetry-poems was inspired and is based on amazing project pipenv-pipes - Pipenv Environment Switcher.

1.2 How does it work?

The list of available projects has to be populated before usage!

1.2.1 Populating poems list

1.3 Documentation

Documentation is available on poetry-poems.readthedocs.io

1.4 License

- lgpl-3.0
- license note

1.5 Credits

Project based on Pipes, Pipenv Environment Switcher. A modified version of Pick for curses based interactive selection list in the terminal is also used.

1.6 Author

harper25

CHAPTER

TWO

INSTALLATION

2.1 Compatibility

- Python 3.6+ (for Python 3.6 it may be necessary to run `export LC_ALL=en_US.utf-8`)
- Ubuntu
- MacOS
- Windows10
  - Working in Command Prompt, PowerShell, Git Bash, Cmder
  - For Git Bash invoking the tool may require prefixing the commands with `winpty`. More reading: git-for-windows, winpty.

```
$ winpty poems
```

2.2 Stable Release

To install most recent Poetry Poems version, run the following command in your terminal (MacOS + Ubuntu + Windows):

```
$ pip3 install poetry-poems
```

Note: Poems requires the curses module, which is a part of the Python standard library and ready to use on Unix systems. However, curses module is not available on Windows. Therefore, Poems automatically installs windows-curses for Windows.

CHAPTER

THREE

USAGE

Terminology

poem

Python project developed with Poetry

Poems Registry

`.poetry-poems` file that stores paths to all poems available to poetry-poems. It is created in your `$HOME` directory.

3.1 Interactive Switcher

Choose your poem from a prepopulated list with an interactive switcher. Adding poems to the Poem Registry is required, see: *Add a new poem*.

```
$ poems
```

Choosing a poem will *cd* into the project directory and activate the corresponding Poetry Shell.

3.2 Add a new poem

Before using Poems to activate a poem, its path has to be added to the Poems Registry:

```
$ poems --add <project_path>
$ poems --add $PWD
$ poems --add .
```

Note: Virtual environments created with Poetry (*poetry shell*) by default are kept in *poetry config virtualenvs.path*. They do not store paths to projects (as you may use the same virtual environment in multiple projects). However, it is possible to get a link from the project to the corresponding Poetry virtual environment. Therefore, Poems takes advantage of a hidden file *.poetry-poems* that is created to store paths to Poetry projects.

3.2.1 .venv in project

In case you would like to activate a poem with a virtual environment located in the project directory (created by Poetry or with *virtualenv*), please make sure that Poetry is configured correctly in the project directory:

```
$ cd <project_path>
$ poetry config --local virtualenvs.in-project true
```

Currently, virtual environments called `.venv` are supported.

Note: This command is helpful to see the virtual environment associated with Poetry project:

```
$ poetry env list --full-path
```

3.2.2 Keyboard Shortcuts

The Interactive environment switcher accepts the following commands:

- `UP` + `DOWN`: Scroll through the list
- `ENTER`: Select and activate the poem
- `ESC`: Exit Poems
- `LEFT` + `RIGHT`: See the detailed information about each poem
- `QUERY`: Start writing a poem name to filter the list
- `BACKSPACE`: Delete last character from filter term
- `DEL`: Clear filter

3.3 Activate a poem from the Command Line

Activate a project directly by writing poems followed by a poem name:

```
$ poems project1
```

If a query term (eg. `proj`) matches more than one project, the *Interactive Switcher* will launch with the list filtered by the entered query term.

3.4 Other commands

3.4.1 List Environments

This command will list all poems saved in Poems Registry:

```
$ poems --list
```

Output:

```
BBreaker *
MemoryMuppets *
venv_by_virtualenv *
venv_by_poetry *
not_poetry_project *
not_existent_project *
```

See more details about each poem:

```
$ poems --list --verbose
```

Output:

```
POETRY_HOME: /.cache/pypoetry/virtualenvs

BBreaker *
    Environment:         ~.cache~pypoetry~virtualenvs~bbreaker-z2QUBx6S-py3.9
    Binary:              Python 3.9.1
    Project Dir:         ~apps~BBreaker

MemoryMuppets *
    Environment:         ~.cache~pypoetry~virtualenvs~memorymuppets-HTGmVbtZ-py3.8
    Binary:              Python 3.8.7
    Project Dir:         ~apps~MemoryMuppets

venv_by_virtualenv *
    Environment:         ~apps~venv_by_virtualenv~.venv
    Binary:              Python 3.8.7
    Project Dir:         ~apps~venv_by_virtualenv

venv_by_poetry *
    Environment:         ~apps~venv_by_poetry~.venv
    Binary:              Python 3.7.9
    Project Dir:         ~apps~venv_by_poetry

not_poetry_project *
    Environment:         -- Not configured --
    Binary:              -- Not configured --
    Project Dir:         ~apps~not_poetry_project

not_existent_project *
    Environment:         -- Not configured --
    Binary:              -- Not configured --
    Project Dir:         ~apps~not_existent_project
```

3.4.2 Delete poem

This deletes only a path to the poem from the Poems Registry. The project and virtual environment remain untouched.

```
$ poems --delete not_poetry_project
```

Output:

```
Are you sure you want to delete: '/apps/not_poetry_project' from poems file? [y/N]: y
Poem 'not_poetry_project' deleted from poems file
```

3.4.3 Custom Poems Registry

It is possible to use custom Poems Registry file (in case you work on microservices belonging to one particular project):

```
$ poems --poems_file <custom_poems_file>
```

3.4.4 Usage Help

The list of available commands together with short descriptions can be accessed right in the command line:

```
$ poems --help
```

CHAPTER

FOUR

SHELL COMPLETION

Autocomplete option is available because of a special `--_completion` flag, provided by the poems cli. Below are instructions for setting up autocompletion for Bash, Zsh, Fish, and pdksh.

Note: After setting the completions, please, remember to restart your session or open a new terminal ;)

Warning: Autocompletin does not work when a virtualenv shell is already active.Make sure that you are not inside one before usage!

4.1 Bash + zsh

Add the code below to your `.bashrc/.zshrc`:

```
export BASE_SHELL=$(basename $SHELL)

if [[ "$BASE_SHELL" == "zsh" ]] ; then
autoload bashcompinit && bashcompinit
fi

_poetry_poems_completions() {
COMPREPLY=($(compgen -W "$(poems --_completion)" -- "${COMP_WORDS[1]}"))
}
complete -F _poetry_poems_completions poems
```

4.2 Fish

Add a new file `poems.fish` to your Fish config folder (eg. `~/.config/fish/completions/poems.fish`):

```
complete --command poems --arguments '(poems --_completion (commandline -cp))' --no-files
```

4.3 pdksh

To have a shell completion, write into your personal `~/.profile`, after the call of exported environments variables for your Python, as WORKON_HOME:

```
set -A complete_poems -- $(poems --_completion)
```

4.4 Credits

pipenv-pipes completions

CHAPTER

FIVE

CONTRIBUTING & DEVELOPMENT

Contributions are welcome and greatly appreciated!

5.1 How to Contribute

5.1.1 Report Bugs

Report bugs at https://github.com/harper25/poetry-poems/issues

5.1.2 Fix Bugs

Scan through the issues on GitHub. Please, feel free to implement a fix to any issue tagged with "bug" and "help wanted".

5.1.3 Implement Features

Scan through the issues on GitHub. Please, feel free to implement any feature tagged with "enhancement" and "help wanted".

5.1.4 Write Documentation

Please, update the documentation after the bugfixes, new feature implementations and in any case when it is unclear, not detailed enough or outdated.

5.1.5 Submit Feedback

The best way to send feedback is to:

- Create a new issue at https://github.com/harper25/poetry-poems/issues
- Star the project :)

When you are proposing a new feature:

- Explain carefully your idea
- Keep the scope as narrow as possible, to make it easier to implement
- Consider involving youself in providing a solution :)

5.2 Setup Development Environment

Ready to contribute? Here's how to set up Poems for local development.

1. Fork poetry-poems project on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:YOUR_GITHUB_USERNAME/poetry-poems.git
```

3. Create a virtual environment with Poetry:

```
$ cd poetry_poems
$ poetry install
$ poetry shell
```

4. Create a branch for local development so you can make your changes locally:

```
$ git checkout -b issue-<issue-no>/<name-of-your-bugfix-or-feature>
```

5. Make sure that the code passes all tests after implementing changes. See the *Testing* section below for more details.
6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Short and meaningful commit message" -m "Detailed␣
↪description of your changes."
$ git push origin issue-<issue-no>/<name-of-your-bugfix-or-feature>
```

7. Submit a pull request through the GitHub website.

5.3 Testing

5.3.1 Run unit tests

Unit tests are written in pytest.

```
$ pytest
```

5.3.2 Tox

It is possible to run the tests on all configured Python versions locally (it is also done in the CI pipeline). However, when using `pyenv` it is necessary to make them available to tox. It is possible by running:

```
$ pyenv local 3.6.9 3.7.5 3.8.3 3.9.0
$ tox
```

Note: Python versions have to be installed with Pyenv first:

```
$ pyenv versions
$ pyenv install --list
$ pyenv install 3.6.9
```

5.3.3 Linter

The code is formatted with isort and black. Flake8 is used as a static linter.

```
$ isort .
$ black .
$ flake8 .
```

5.4 Pull Request Guidelines

Before submitting your pull request, please check if it meets these guidelines:

1. The pull request should contain tests that cover the new code (or decent amount).
2. The new functionalities should be described in the updated documentation.
3. The pull request should work for Python 3.6+. Please, check if the CI pipeline is passing: https://travis-ci.org/github/harper25/poetry-poems/pull_requests.

5.5 Deployment

Reminder on how to release a new version:

- Bump a version in the project
- Push a tag to GitHub
- Release manually to PyPI

CHAPTER

SIX

HISTORY

6.1 0.2.2 (2022-10-08)

- Fix Dockerfiles for testing poetry-poems (#12)
- Fix slow CLI responsiveness when checking details (partially #3)
- Fix error message when activating non-existent project (#4)
- Update project dependencies

6.2 0.2.1 (2021-06-06)

- Fix poems for Windows - crashing on subproccess calls
- Fix displaying poems details for Windows in colored terminals

6.3 0.2.0 (2021-02-11)

- First release on PyPi
- ReadTheDocs Documentation
- Show hint when adding a project with virtual environment inside project directory

6.4 0.1.0 (2021-01-24)

- First release on GitHub

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search