Pipenv Pipes Documentation

Release 0.7.1

Gui Talarico

Dec 07, 2018

Contents:

CHAPTER 1

Pipes

Pipenv Environment Switcher

1.1 Overview

Pipes is a Pipenv companion CLI tool that provides a quick way to jump between your pipenv powered projects.

1.2 Documentation

Documentation is hosted on pipenv-pipes.readthedocs.io

1.3 License

MIT License

1.4 Credits

Package created with Cookiecutter + cookiecutter-pypackage

1.5 Author

Send me a message on twitter

CHAPTER 2

Installation

2.1 Compatibility

- Python 3.4+ (PRs for 2.7 welcome!)
- Unix + Windows Support

2.2 Stable Release

To install Pipenv Pipes, run this command in your terminal:

MacOs + Ubuntu:

```
$ pip3 install pipenv-pipes --user
```

Windows:

```
$ pip3 install pipenv-pipes
$ pip3 install curses --find-links=https://github.com/gtalarico/curses-win/releases
```

Note: Pipes requires the curses module, which is part of the python standard library. Unfortunately, curses it's currently not supported on Windows, so Windows users need to install the unofficial curses windows binaries, kindly created Christoph Gohlke. You can see the bug tracker on the issue here. To make this step easier, I have added a copy of the windows curses binaries to a repo, but you can also download and install it yourself from the author's page.

If you haven't heard of Christoph, he has received a PSF Community Service Award for his work in maintaining an impressive collection of Windows binaries.

This is the preferred method to install Pipenv Pipes, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.3 Known Issues

`PIPENV_VENV_IN_PROJECT` is not currently supported. If have ideas on how Pipes could support local venvs please start an issue and include your proposed implementation.

2.4 Curses on Windows

The interactive selector uses Curses, which is not natively supported on windows. To enable curses on Windows we must install additional Windows binaries, which were obtained from here

The installer should automatically install the windows curses binaries if you include the `find-link` as shown above. Should you need to install them manually, just download the appropriate wheel from the link above and use pip to install it:

```
$ pip install ..packetPath..\curses-2.2-cpXX-none-win_XXX.whl
```

You can read more about windows support for curses here: https://bugs.python.org/issue2889

2.4.1 Terminal

While Pipes should work on the standard Windows console (cmd.exe) a terminal like Cmder is highly recommended:

CHAPTER 3

Usage

3.1 Interactive Switcher

The easiest way to use Pipes is to use the interactive switcher.

```
$ pipes
```

Note: Before you can use Pipes to activate a given project, the selected environment must have a project directory associated with it. To understand how Pipes links Project Directories with corresponding virtualenvs read the section on how to *Set Project Directory*.

3.1.1 Keyboard Shortcuts

The Interactive environment switcher accepts the following commands:

- `UP` + `DOWN`: Scrolls through the list
- `PAGE UP` + `PAGE DOWN`: Scrolls through the list in larger increments
- `ENTER`: Selects and activates the environemnt
- `ESC`: Exit Pipes
- `LEFT` + `RIGHT`: Cycles through the available information on each virtual environment
- `QUERY`: Use any alphanumeric characters to filter the list
- `BACKSPACE`: Delete last character from filter term
- `DEL`: Clear filter

3.2 From the Command Line

You can activate an environment directly from the command line by calling `pipes` followed by a query term which is used to select the desired environment:

```
$ pipes project1
```

This would *cd* into directory `/path/to/project1` and activate the corresponding Pipenv Shell.

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.

3.3 List Environments

Use Pipes to see all detected Pipenv Environments.

```
$ pipes --list
```

Output:

```
project1-LwEMcb8W
project2-R1v7_ynT
```

The `--list` flag can also be used with the `--verbose` option, which shows additional information about the environments such as the environment path, project directory (if available) and the python version of the virtual environment.

```
$ pipes --list --verbose
```

Vebose Output:

```
PIPENV_HOME: /Users/user/.local/share/virtualenvs

project1-LwEMcb8W *
    Environment:       $PIPENV_HOME/project1-LwEMcb8W
    Binary:            Python 3.5.5
    Project Dir:       ~/dev/project

project2-R1v7_ynT *
    Environment:       $PIPENV_HOME/project2-R1v7_ynT
    Binary:            Python 3.4.8
    Project Dir:       ~/dev/project2
```

The presence of an asterisk (`*`) on the environment list indicates if the virtual environment already has a project directory associated.

The *lack* of a `*` indicates the Environment has not yet been associated with a project directory. If you try switching into an environment without the `*`, Pipes will tell you need to *link* the environment with a project directory first.

3.4 Delete Environment

Use Pipes to delete a Pipenv Environment.

```
$ pipes project1 --delete
```

Output:

```
Environment 'project1-LwEMcb8W' deleted
```

3.5 Set Project Directory

To link a project directory with its environment use the `--link` flag:

```
$ pipes --link /path/to/project1
```

Pipes will find the associated Pipenv Environmnet by running `pipenv --venv` from from the target directory.

If the target directory finds a valid environment, Pipes will create a new `.project` with the project path and save it inside the virtual environment. This file is used by Pipes to detect the project directory.

Note: If you are using the latest release of Pipenv (v2018.05.18 or later), it should automatically create a `.project` file to store your project path and the use of `--link` is unnecessary.

Credit: Charlie Denton for sending this Pull Request.

3.6 Unlink a Project

To unlink `project1` directory from its Pipenv Environment run:

```
$ pipes project1 --unlink
```

This will delete the `.project` file created by the *Set Project Directory* command.

3.7 Usage Help

You can see the list of available commands directly from the command line:

```
$ pipes --help
```

CHAPTER 4

Shell Completion

Pipes includes a helper `--_completion` flag which can be used by terminals to provide autocomplete options. Below are instructions for setting up autocompletion for Bash, Zsh, and Fish.

If you have ideas for improving these please share them with use over at the Pipes repo

Warning: Pipes cannot activate an pipenv shell when one is already active, thefore the autocompletion helper does not run either. Make sure you are not inside a Pipenv Shell when trying to use the autocomplete helper.

4.1 Fish

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

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

4.2 Bash + Zsh

Original code kindly provided by jonatasbaldin here

Add the code below to your *.bashrc*:

```
# .bashrc
export BASE_SHELL=$(basename $SHELL)

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

_pipenv-pipes_completions() {
COMPREPLY=($(compgen -W "$(pipes --_completion)" -- "${COMP_WORDS[1]}"))
}

complete -F _pipenv-pipes_completions pipes
```

CHAPTER 5

Contributing

Contributions are welcome and greatly appreciated! Every little bit helps, and credit will always be given.

5.1 How to Contribute

5.1.1 Report Bugs

Report bugs at https://github.com/gtalarico/pipenv-pipes/issues.

5.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

5.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

5.1.4 Write Documentation

Pipenv Pipes could always use more documentation, whether as part of the official Pipenv Pipes docs, in docstrings, or even on the web in blog posts, articles, and such.

5.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/gtalarico/pipenv-pipes/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome.

5.2 Setup Pipes Development Environment

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

1. Fork the the Pipes repository from GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:YOUR_GITHUB_USERNAME/pipenv_pipes.git
```

3. Create a virtualenvironment - we will use Pipenv:

```
$ cd pipenv_pipes
$ pipenv install --dev
$ python setup.py develop
```

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

```
$ git checkout -b name-of-your-bugfix-or-feature
```

5. When you're done making changes, check that your changes pass all tests. See the *Testing* section below for more details on testing.
6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

5.3 Testing

5.3.1 Run test suite

Tests use Pytest. To run the test suite run:

```
$ pytest
or
$ python setup.py test or
```

5.3.2 Linter

Make sure the code follows Flake 8 standard by using a linter within your code editor or use the command below:

```
$ flake8 pipenv_pipes tests
```

5.4 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.7, 3.4, 3.5 and 3.6, and for PyPy. Check https://travis-ci.org/gtalarico/pipenv_pipes/pull_requests and make sure that the tests pass for all supported Python versions.

5.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bump2version  major / minor / patch / release
$ git push
$ git push --tags
```

Note: Travis should run all tests but integration with PyPI is currently disabled. To deploy to Pypi use `make release`.

CHAPTER 6

History

6.1 0.7.0 (2018-11-07)

- Add *–delete* feature

6.2 0.6.3 (2018-05-27)

- Add Color to paths on activation
- Fix: Autocompletion with no envs - Issues #18
- Separated curses dependency for Windows

6.3 0.6.2 (2018-05-19)

- Fix Windows Python Version getter call

6.4 0.6.1 (2018-05-15)

- Fix Python binary version detect call
- Improved testing speed and fixtures”

6.5 0.6.0 (2018-05-15)

- Added completion helper flag and completion docs

6.6 0.5.1 (2018-05-13)

- Bug fix: Setup.py missing picker package

6.7 0.5.0 (2018-05-12)

- Curses Interface
- Show Python Version

6.8 0.4.2 (2018-05-06)

- Fixed Envname character bug
- Added –version
- AddedFull Test Suite
- Adde Windows Default Path support
- Cleanned up dependencies
- Cleanned up env validation

6.9 0.4.1 (2018-05-02)

- Allow New Line on Read (PR #3)
- Typo PR #2

6.10 0.4.0 (2018-05-02)

- Renamed Set/Unset to Link/Unlink
- Use pipenv –venv to verify project directory
- Improved Set/Unset CLI
- Improved validation for Project Directories
- New CLI API (single command for simplicity)
- Indicates if has project directory set on project list
- Updated Verbose Project List
- Disallow Pipes when VENV is active.
- Added Colorama

6.11 0.1.0

- First release on PyPI.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search