faz Documentation

Release 0.1.7

Hugo Martiniano

July 21, 2016

Contents

Contents:

CHAPTER 1

faz

Faz is a data workflow tool heavily inspired in .. _Drake: https://github.com/Factual/drake

The intended use is combining data treatment scripts in bash, python, ruby (or anything else, with a little coding) into a single text file.

The name “faz” is portuguese for “do” or “make”.

The various steps can be separated into tasks, with defined inputs and outputs. Dependencies between the tasks are determined from inputs and outputs of every task. The program executes all tasks in the appropriate order, checking for the existence of output and input files.

1.1 Why?

- Because I like Drake but can’t stand the startup time of java.
- Because I can (actually to see if I can, but it turns out I can).

1.2 Features

- simple but robust functionality
- easy to use and extend (the code, minus the tests, is around 300 lines of python)
- fast startup time (compared to Drake)
- Documentation: https://faz.readthedocs.org.

1.3 Installation

Using pypi

```
pip install faz
```

1.4 Usage

From the command line, just type

```
faz
```

without arguments, the program will read the tasks from a file called “fazfile”. If you want to use another filename, just give that as an argument to the program

```
faz <filename>
```

to get a list of command line arguments type

```
faz -h
```

1.5 Task file basics

The task file is a plain text file, with a syntax similar to Drake input files. The following is an example with two tasks

```
# file1 <-
touch file1

# file2 <- file1
cat file1 > file2
```

Lines starting with “#” and having the symbols “<-” signal a task. On the left of the “<-” is a (comma separated) list of the files produced by the task. On the right are the task dependencies, the files needed to run that task. In the above example the first task has no dependencies, and produces a file called “file1”. The second task has “file1” as a dependency, and has as output a file called “file2”.

The outputs and inputs and inputs of each task are used by the program to estabilish the order by which the tasks have to be run, and if they need to be run. In the example above, if a file called “file1” was already present in the directory the program was run, the first task would not be executed.

The code sections are all the lines in betweeen the two task lines. In these two tasks, they are just are just plain bash commands but could be, for example, python code

```
# file1 <-
touch file1

# file2 <- file1 :python
f1 = open("file1")
text = file1.read()
f2 = open("file2", "w")
f2.write(text)
```

note that, in the second task, there’s an extra option ”:python”, wich indicates to the program that the code from this task is python code. Options are a list of (comma separated) keywords following the ”:”, and must be placed after the inputs.

CHAPTER 2

Installation

From pipy:

```
$ pip install faz
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv faz
$ pip install faz
```

CHAPTER 3

Usage

From the command line, just type

```
faz
```

without arguments, the program will read the tasks from a file called “fazfile”. If you want to use another filename, just give that as an argument to the program

```
faz <filename>
```

to get a list of command line arguments type

```
faz -h
```

CHAPTER 4

Contributing

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

You can contribute in many ways:

4.1 Types of Contributions

4.1.1 Report Bugs

Report bugs at https://github.com/hmartiniano/faz/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

4.1.2 Fix Bugs

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

4.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "feature" is open to whoever wants to implement it.

4.1.4 Write Documentation

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

4.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/hmartiniano/faz/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 :)

4.2 Get Started!

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

1. Fork the *faz* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/faz.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv faz
$ cd faz/
$ python setup.py develop
```

4. Create a branch for local development:

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

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 faz tests
$ python setup.py test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

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.

4.3 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 2.6, 2.7, 3.3, and 3.4, and for PyPy. Check https://travis-ci.org/hmartiniano/faz/pull_requests and make sure that the tests pass for all supported Python versions.

4.4 Tips

To run a subset of tests:

```
$ python -m unittest tests.test_faz
```

CHAPTER 5

Credits

5.1 Development Lead

- Hugo Martiniano <hugomartiniano@gmail.com>

5.2 Contributors

None yet. Why not be the first?

CHAPTER 6

History

CHAPTER 7

0.1.0 (2014-01-11)

- First release.

CHAPTER 8

0.1.1 (2015-03-20)

- Bug fixes.

CHAPTER 9

0.1.2 (2015-10-17)

- Project name change.

CHAPTER 10

0.1.3 (2016-03-26)

- NetworkX dependency removed.

CHAPTER 11

0.1.4 (2016-05-19)

- Input and output names added to task environment.

CHAPTER 12

0.1.5 (2016-05-19)

- Bug Fixes in variable expansion code.

CHAPTER 13

0.1.6 (2016-07-18)

- Added a mechanism to include other task files.

CHAPTER 14

0.1.7 (2016-07-20)

- dependencies and outputs can now be on different directories.

CHAPTER 15

Indices and tables

- genindex
- modindex
- search