pytest-workflow Documentation

Release 0.2.0

Leiden University Medical Center

Mar 04, 2019

Contents

Table of contents

CHAPTER 1

Introduction

Writing workflows is hard. Testing if they are correct is even harder. Testing with bash scripts or other code has some flaws. Is this bug in the pipeline or in my test-framework? Pytest-workflow aims to make testing as simple as possible so you can focus on debugging your pipeline.

CHAPTER 2

Installation

Pytest-workflow is tested on python 3.5, 3.6 and 3.7. Python 2 is not supported.

- Make sure your virtual environment is activated.
- Install using pip `pip install pytest-workflow`
- Create a `tests` directory in the root of your repository.
- Create your test yaml files in the `tests` directory.

CHAPTER 3

Running pytest-workflow

Run `pytest` from an environment with pytest-workflow installed. Pytest will automatically gather files in the `tests` directory starting with `test` and ending in `.yaml` or `.yml`.

The workflows are run automatically. Each workflow gets its own temporary directory to run. These directories are cleaned up after the tests are completed. If you wish to inspect the output of a failing workflow you can use the `--keep-workflow-wd` flag to disable cleanup.

If you wish to change the temporary directory in which the workflows are run use `--basetemp <dir>` to change pytest's base temp directory.

CHAPTER 4

Writing tests with pytest-workflow

Below is an example of a YAML file that defines a test:

```
- name: Touch a file
  command: touch test.file
  files:
    - path: test.file
```

This will run `touch test.file` and check afterwards if a file with path: `test.file` is present. It will also check if the `command` has exited with exit code `0`, which is the only default test that is run. Testing workflows that exit with another exit code is also possible.

A more advanced example:

```
- name: moo file                        # The name of the workflow (required)
  command: bash moo_workflow.sh         # The command to execute the workflow (required)
  files:                                # A list of files to check (optional)
    - path: "moo.txt"                   # File path. (Required for each file)
      contains:                         # A list of strings that should be in the file␣
↪(optional)
        - "moo"
      must_not_contain:                 # A list of strings that should NOT be in the␣
↪file (optional)
        - "Cock a doodle doo"
      md5sum: e583af1f8b00b53cda87ae9ead880224   # Md5sum of the file (optional)

- name: simple echo                     # A second workflow. Notice the starting `-`␣
↪which means
  command: "echo moo"                   # that workflow items are in a list. You can add␣
↪as much workflows as you want
  files:
    - path: "moo.txt"
      should_exist: false               # Whether a file should be there or not.␣
↪(optional, if not given defaults to true)
  stdout:                               # Options for testing stdout (optional)
```

(continues on next page)

(continued from previous page)

```
    contains:                          # List of strings which should be in stdout␣
↪(optional)
      - "moo"
    must_not_contain:                  # List of strings that should NOT be in stout␣
↪(optional)
      - "Cock a doodle doo"

- name: mission impossible             # Also failing workflows can be tested
  command: bash impossible.sh
  exit_code: 2                         # What the exit code should be (optional, if not␣
↪given defaults to 0)
  files:
    - path: "fail.log"                 # Multiple files can be tested for each workflow
    - path: "TomCruise.txt"
  stderr:                              # Options for testing stderr (optional)
    contains:                          # A list of strings which should be in stderr␣
↪(optional)
      - "BSOD error, please contact the IT crowd"
    must_not_contain:                  # A list of strings which should NOT be in␣
↪stderr (optional)
      - "Mission accomplished!"
```

The above YAML file contains all the possible options for a workflow test.

CHAPTER 5

Known issues

- `pytest-workflow` does not work well together with `pytest-cov`. This is due to the temporary directory creating nature of `pytest-workflow`. This can be solved by using:

  This will work as expected.

CHAPTER 6

Reporting bugs and feature requests

Bugs can be reported and features can be requested on our Github issue tracker.

The aim of this project is to be as user-friendly as possible for writing workflow tests, so all suggestions and bug reports are welcome!

CHAPTER 7

Contributing

If you feel like this project is missing a certain something, feel free to make a pull request. You can find our Github page here.

CHAPTER 8

Changelog

8.1 Version 0.2.0

- Cleanup the readme and move advanced usage documentation to our readthedocs page.
- Start using sphinx and readthedocs.org for creating project documentation.
- The temporary directories in which workflows are run are automatically cleaned up at the end of each workflow test. You can disable this behaviour by using the *–keep-workflow-wd* flag, which allows you to inspect the working directory after the workflow tests have run. This is useful for debugging workflows.
- The temporary directories in which workflows are run can now be changed by using the *–basetemp* flag. This is because pytest-workflow now uses the built-in tmpdir capabilities of pytest.
- Save stdout and stderr of each workflow to a file and report their locations to stdout when running `pytest`.
- Comprehensible failure messages were added to make debugging workflows easier.

8.2 Version 0.1.0

- A full set of examples is now provided in the README.
- Our code base is now checked by pylint and bandit as part of our test procedure to ensure that our code adheres to python and security best practices.
- Add functionality to test whether certain strings exist in files, stdout and stderr.
- Enable easy to understand output when using pytest verbose mode (`pytest -v`). The required code refactoring has simplified the code base and made it easier to maintain.
- Enable the checking of non-existing files
- Enable the checking of file md5sums
- Use a schema structure that is easy to use and understand.

- Pytest-workflow now has continuous integration and coverage reporting, so we can detect regressions quickly and only publish well-tested versions.
- Fully parametrized tests enabled by changing code structure.
- Initialized pytest-workflow with option to test if files exist.