pytest-workflow Documentation

Release 1.3.0

Leiden University Medical Center

Apr 20, 2020

Contents

1 Introduction **3**

2 Installation **5**
- 2.1 In a virtual environment . . . 5
- 2.2 On Ubuntu or Debian . . . 5
- 2.3 Conda . . . 5

3 Writing tests with pytest-workflow **7**
- 3.1 Getting started . . . 7
- 3.2 Test options . . . 7
- 3.3 Writing custom tests . . . 8

4 Running pytest-workflow **11**
- 4.1 Usage . . . 11
- 4.2 Specific pytest options for pytest workflow . . . 11
- 4.3 Temporary directory cleanup and creation . . . 12
- 4.4 Running multiple workflows simultaneously . . . 12
- 4.5 Running specific workflows . . . 12

5 Examples **15**
- 5.1 Snakemake example . . . 15
- 5.2 WDL with Cromwell example . . . 15
- 5.3 WDL with miniwdl example . . . 16

6 Known issues **17**

7 Reporting bugs and feature requests **19**

8 Contributing **21**

9 Changelog **23**
- 9.1 version 1.3.0 . . . 23
- 9.2 version 1.2.3 . . . 24
- 9.3 version 1.2.2 . . . 24
- 9.4 version 1.2.1 . . . 24
- 9.5 version 1.2.0 . . . 24
- 9.6 version 1.1.2 . . . 24
- 9.7 version 1.1.1 . . . 25

9.8 version 1.1.0 25
9.9 Version 1.0.0 25
9.10 Version 0.4.0 25
9.11 Version 0.3.0 26
9.12 Version 0.2.0 26
9.13 Version 0.1.0 26

Table of contents

- *pytest-workflow*
- *Introduction*
- *Installation*
  - *In a virtual environment*
  - *On Ubuntu or Debian*
  - *Conda*
- *Writing tests with pytest-workflow*
  - *Getting started*
  - *Test options*
  - *Writing custom tests*
- *Running pytest-workflow*
  - *Usage*
  - *Specific pytest options for pytest workflow*
    - *Named Arguments*
  - *Temporary directory cleanup and creation*
  - *Running multiple workflows simultaneously*
  - *Running specific workflows*
- *Examples*
  - *Snakemake example*
  - *WDL with Cromwell example*
  - *WDL with miniwdl example*
- *Known issues*
- *Reporting bugs and feature requests*
- *Contributing*
- *Changelog*
  - *version 1.3.0*
  - *version 1.2.3*
  - *version 1.2.2*
  - *version 1.2.1*
  - *version 1.2.0*
  - *version 1.1.2*
  - *version 1.1.1*
  - *version 1.1.0*
  - *Version 1.0.0*

- *Version 0.4.0*
- *Version 0.3.0*
- *Version 0.2.0*
- *Version 0.1.0*

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, 3.7 and 3.8. Python 2 is not supported.

2.1 In a virtual environment

- Create a new python3 virtual environment.
- Make sure your virtual environment is activated.
- Install using pip `pip install pytest-workflow`.

2.2 On Ubuntu or Debian

- This requires the `python3` and `python3-pip` packages to be installed.
- Installing
  - system-wide: `sudo python3 -m pip install pytest-workflow`.
  - for your user only (no sudo needed): `python3 -m pip install --user pytest-workflow`
- `pytest` can now be run with `python3 -m pytest`.

Note: Running plain `pytest` on Ubuntu or Debian outside of a virtual environment will not work with `pytest-workflow` because this will start the python2 version of `pytest`. This is because python2 is the default python on any distribution released before January 1st 2020.

2.3 Conda

Pytest-workflow is also available as a conda package on conda-forge. To install with conda:

- Set up conda to use the conda-forge channel
  - If you want to use pytest-workflow together with bioconda you can follow the instructions here.
- `conda install pytest-workflow`.

CHAPTER 3

Writing tests with pytest-workflow

3.1 Getting started

In order to write tests that are discoverable by the plugin you need to complete the following steps.

- Create a `tests` directory in the root of your repository.
- Create your test yaml files in the `tests` directory. The files need to start with `test` and have a `.yml` or `.yaml` extension.

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

```yaml
- 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.

3.2 Test options

```yaml
- 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)
```

(continues on next page)

(continued from previous page)

```
        - "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)
    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
  tags:                              # A list of tags that can be used to select␣
↪which test
    - should fail                    # is run with pytest using the `--tag` flag.
  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.gz"       # Gzipped files can also be searched, provided␣
↪their extension is '.gz'
      contains:
        - "starring"
  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.

Note: Workflow names must be unique. Pytest workflow will crash when multiple workflows have the same name, even if they are in different files.

3.3 Writing custom tests

Pytest-workflow provides a way to run custom tests on files produced by a workflow.

```
import pathlib
import pytest
```

(continues on next page)

(continued from previous page)

```
@pytest.mark.workflow('files containing numbers')
def test_div_by_three(workflow_dir):
    number_file = pathlib.Path(workflow_dir, "123.txt")
    number_file_content = number_file.read_text()
    assert int(number_file_content) % 3 == 0
```

The `@pytest.mark.workflow('files containing numbers')` marks the test as belonging to a workflow named `files containing numbers`. This test will only run if the workflow 'files containing numbers' has run.

Multiple workflows can use the same custom test like this:

```
import pathlib
import pytest

@pytest.mark.workflow('my_workflow', 'another_workflow',
                      'yet_another_workflow')
def test_ensure_long_logs_are_written(workflow_dir):
    log = pathlib.Path(workflow_dir, "log.out")
    assert len(log.readtext()) > 10000
```

`workflow_dir` is a fixture. It does not work without a `pytest.mark.workflow('workflow_name')` mark. This is a pathlib.Path object that points to the folder where the named workflow was executed. This allows writing of advanced python tests for each file produced by the workflow.

Note: stdout and stderr are available as files in the root of the `workflow_dir` as `log.out` and `log.err` respectively.

CHAPTER 4

Running pytest-workflow

4.1 Usage

Run `pytest` from an environment with pytest-workflow installed or `python3 -m pytest` if using a system-wide or user-wide installation. 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. The `stdout` and `stderr` of the workflow command are also saved to this directory to `log.out` and `log.err` respectively.

To check the progress of a workflow while it is running you can use `tail -f` on the `stdout` or `stderr` file of the workflow. The locations of these files are reported in the log as soon as a workflow is started.

4.2 Specific pytest options for pytest workflow

```
usage: pytest [-h] [--kwd] [--kwdof] [--wt WORKFLOW_THREADS] [--symlink]
              [--tag WORKFLOW_TAGS]
```

4.2.1 Named Arguments

--kwd, --keep-workflow-wd Keep temporary directories where workflows are run for debugging purposes. This also triggers saving of stdout and stderr in the workflow directory.

Default: False

--kwdof, --keep-workflow-wd-on-fail Similar to –keep-workflow-wd, but only keeps the temporary directories if there are test failures. On success all directories are deleted.

Default: False

--wt, --workflow-threads The number of workflows to run simultaneously.

Default: 1

--symlink	Instead of copying the current working directory, create a similar directory structure where all files are replaced with symbolic links. This saves disk space, but should only be used for tests that do use these files read-only. Default: False
--tag	Run workflows with this name or tag. Default: []

4.3 Temporary directory cleanup and creation

The temporary 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` or `--kwd` flag to disable cleanup. This will also make sure the logs of the pipeline are not deleted. If you only want to keep directories when one or more tests fail you can use the `--keep-workflow-wd-on-fail` or `--kwdof` flag. `--keep-workflow-wd-on-fail` will keep all temporary directories, even from workflows that have succeeded.

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

Warning: When a directory is passed to `--basetemp` some of the directory contents will be deleted. For example: if your workflow is named `"my workflow"` then any file or directory named `my_workflow` will be deleted. This makes sure you start with a clean slate if you run pytest again with the same `basetemp` directory. DO NOT use `--basetemp` on directories where none of the contents should be deleted.

The temporary directories created are copies of pytest's root directory, the directory from which it runs the tests. If you have lots of tests, and if you have a large repository, this may take a lot of disk space. To alleviate this you can use the `--symlink` flag which will create the same directory layout but instead symlinks the files instead of copying them. This is *slower* for lots of small files, and it carries with it the risk that the tests may alter files from your work directory. If there are a lot of large files and files are used read-only in tests, then it will use a lot less disk space and be faster as well.

4.4 Running multiple workflows simultaneously

To run multiple workflows simultaneously you can use `--workflow-threads <int>` or `--wt <int>` flag. This defines the number of workflows that can be run simultaneously. This will speed up things if you have enough resources to process these workflows simultaneously.

4.5 Running specific workflows

To run a specific workflow use the `--tag` flag. Each workflow is tagged with its own name and additional tags in the `tags` key of the yaml.

```
- name: moo
  tags:
    - animal
  command: echo moo
- name: cock-a-doodle-doo
```

(continues on next page)

(continued from previous page)

```
  tags:
    - rooster sound
    - animal
  command: echo cock-a-doodle-doo
- name: vroom vroom
  tags:
    - car
  command: echo vroom vroom
```

With the command `pytest --tag moo` only the workflow named 'moo' will be run. With `pytest --tag 'rooster sound'` only the 'cock-a-doodle-doo' workflow will run. Multiple tags can be used like this: `pytest --tag 'rooster sound' --tag animal` This will run all workflows that have both 'rooster sound' and 'animal'.

Internally names and tags are handled the same so if the following tests:

```
- name: hello
  command: echo 'hello'
- name: hello2
  command: echo 'hello2'
  tags:
    - hello
```

are run with `pytest --tag hello` then both `hello` and `hello2` are run.

CHAPTER 5

Examples

5.1 Snakemake example

An example yaml file that could be used to test a snakemake pipeline is listed below.

```
- name: test-dry-run
  command: snakemake -n -r -p -s Snakefile
- name: test-full-run
  command: snakemake -r -p -s Snakefile
  files:
    - "my_output.txt"
  stderr:
    contains:
     - "(100%) done"
```

5.2 WDL with Cromwell example

Below an example yaml file is explained which can be used to test a WDL pipeline run through Cromwell.

By default Cromwell outputs its files in the execution folder in a deeply-nested folder structure. Cromwell can output to a separate workflow-outputs folder and since Cromwell version 40 it can also output the files in a structure that is not nested. For more information check the Cromwell documentation on global workflow options.

In order to run Cromwell for CI tests an options file should be present in the repository with the following contents:

```
{
"final_workflow_outputs_dir": "test-output",
"use_relative_output_paths": true,
"default_runtime_attributes": {
  "docker_user": "$EUID"
  }
}
```

`final_workflow_outputs_dir` will make sure all the files produced in the workflow will be copied to the `final_workflow_outputs_dir`. `use_relative_output_paths` will get rid of all the Cromwell specific folders such as `call-myTask` etc. The `default_runtime_attributes` are only necessary when using docker containers. It will make sure all the files are created by the same user that runs the test (docker containers run as root by default). This will ensure that files can be deleted by pytest-workflow afterwards.

The following yaml file tests a WDL pipeline run with Cromwell. In this case Cromwell is installed via conda. The conda installation adds a wrapper to Cromwell so it can be used as a command, instead of having to use the jar.

```
- name: My pipeline
  command: cromwell run -i inputs.json -o options.json moo.wdl
  files:
    - path: test-output/moo.txt.gz
      md5sum: 173fd8023240a8016033b33f42db14a2
  stdout:
    contains:
      - "WorkflowSucceededState"
```

5.3 WDL with miniwdl example

For miniwdl please consult the runner reference for more information on the localization of output files as well as options to modify the running of miniwdl from the environment.

Miniwdl will localize all the output files to an `output_links` directory inside the test output directory. If you have a workflow with the output:

Inside the `output_links` directory the directories `moo_file` and `stats` will be created. Inside these directories will be the produced files.

The following yaml file tests a WDL pipeline run with miniwdl.

```
- name: My pipeline
  command: miniwdl run -i inputs.json -d test-output/ moo.wdl
  files:
    - path: test-output/output_links/moo_file/moo.txt.gz
      md5sum: 173fd8023240a8016033b33f42db14a2
    - path: test-output/output_links/stats/number_of_moos_per_cow.tsv
      contains:
        - 42
    - path: test-output/output_links/stats/joy_invoking_moos.tsv
      must_not_contain:
        - 0
```

Please note that the trailing slash in `-d test-output/` is important. It will ensure the files end up in the `test-output` directory.

CHAPTER 6

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:

```
coverage run --source=<your_source_here> -m py.test <your_test_dir>
```

  This will work as expected.

CHAPTER 7

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 8

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 9

Changelog

9.1 version 1.3.0

Python 3.6 and pytest 5.4.0.0 are now minimum requirements for pytest-workflow. This was necessary for fixing the deprecation warning issue and the issue with the subdirectory evaluation. This also gave the opportunity to simplify the source code using new python 3.6 syntax.

- Using the `name` keyword argument in workflow marks will be deprecated from 1.4.0 onwards. A warning will be given if this is used. For example: `pytest.mark.workflow(name="my_workflow")`. Use the name as argument instead: `pytest.mark.workflow("my_workflow")`.
- Allow running custom tests on multiple workflows. You can now use `pytest.mark.workflow("worflow name 1", "workflow name 2", ...)`. (Issue #75)
- Add a miniwdl example to the documentation.
- Added a `--symlink` flag to the CLI that changes the copying behavior. Instead of copying, it creates a similar directory structure where all files are linked to with symbolic links. (Issue #96)
- Refactored the code base. Python 3.6's f-strings and type annotation were used consistently throughout the project. Some code was rewritten to be more concise and readable.
- Improved speed for searching string content in files. This was achieved by removing intermediate functions and simplifying the search function.
- Improved speed for calculating md5sums by increasing the read buffer size from 8k to 64k.
- Solve issue where pytest would display a lot of deprecation warnings when running pytest-workflow. (Issue #98)
- Fix issues with later versions of Cromwell and Snakemake in CI testing.
- Add correct subdirectory evaluation to fix issue where `/parent-dir/child` was evaluated as a subdirectory of `/parent` due to starting with the same string. (Issue #95)
- Fix error in cromwell example which did not allow it to remove folders correctly.

9.2 version 1.2.3

- Added missing `help` section for `--tag` on the CLI.
- Documentation: added usage chapter for pytest-workflow specific options.
- Documentation: updated Cromwell example.
- Removed redundant references to pylint in code comments and CI.
- Remove Codacy from the CI.

9.3 version 1.2.2

- Test against python3.8
- Do not test on python3.5 snakemake as it crashes. Added test for python3.7 snakemake.
- Fix a typo in the documentation.
- Add tags 'wdl', 'cromwell' and 'snakemake' to the package to increase discoverability.
- Remove pylint from the lint procedure as it was very strict and got stricter with every update, causing tests that previously succeeded to fail on a regular basis.
- Make sure pytest-workflow crashes when multiple workflows have the same name, even when they are in different files.
- Added setup.cfg to include license in source distributions on PyPI for future versions

9.4 version 1.2.1

- Since pytest 4.5.0 unknown markers give a warning. `@pytest.mark.workflow` markers are now added to the configuration. Information on usage shows up with `pytest --mark`.
- Updated documentation to reflect the move to conda-forge as requested on this github issue.
- Updated documentation on how to test Cromwell + WDL pipelines.

9.5 version 1.2.0

- Giving a `--basetemp` directory that is within pytest's current working directory will now raise an exception to prevent infinite recursive directory copying.
- The cleanup message is only displayed when pytest-workflow is used.
- Added a `--keep-workflow-wd-on-fail` or `--kwdof` flag. Setting this flag will make sure temporary directories are only deleted when all tests succeed.

9.6 version 1.1.2

- Fixed a bug where the program would hang indefinitely after a user input error.

9.7 version 1.1.1

- Added `--kwd` as alias for `--keep-workflow-wd`. Notify the user of deletion of temporary directories and logs.
- Released pytest-workflow as a conda package on bioconda.

9.8 version 1.1.0

- Enabled custom tests on workflow files.

9.9 Version 1.0.0

Lots of small fixes that improve the usability of pytest-workflow are included in version 1.0.0.

- Gzipped files can now also be checked for contents. Files with '.gz' as extension are automatically decompressed.
- `stdout` and `stderr` of workflows are now streamed to a file instead of being kept in memory. This means you can check the progress of a workflow by running `tail -f <stdout or stderr>`. The location of `stdout` and `stderr` is now reported at the start of each worflow. If the `--keep-workflow-wd` is not set the `stdout` and `stderr` files will be deleted with the rest of the workflow files.
- The log reports now when a workflow is starting, instead of when it is added to the queue. This makes it easier to see which workflows are currently running and if you forgot to use the `--workflow-threads` or `--wt` flag.
- Workflow exit code failures now mention the name of the workflow. Previously the generic name "Workflow" was used, which made it harder to figure out which workflows failed.
- When tests of file content fail because the file does not exist, a different error message is given compared to when the file exist, but the content is not there, which makes debugging easier. Also the accompanying "FileNotFound" error stacktrace is now suppressed, which keeps the test output more pleasant.
- When tests of stdout/stderr content or file content fail a more informative error message is given to allow for easier debugging.
- All workflows now get their own folder within the *same* temporary directory. This fixes a bug where if `basetemp` was not set, each workflow would get its own folder in a separate temp directory. For example running workflows 'workflow1' and 'workflow2' would create two temporary folders:

  '/tmp/pytest_workflow_**33mrz5a5**/workflow1' and '/tmp/pytest_workflow_**b8m1wzuf**/workflow2'

  This is now changed to have all workflows in one temporary directory per pytest run:

  '/tmp/pytest_workflow_**33mrz5a5**/workflow1' and '/tmp/pytest_workflow_**33mrz5a5**/workflow2'
- Disallow empty `command` and `name` keys. An empty `command` caused pytest-workflow to hang. Empty names are also disallowed.

9.10 Version 0.4.0

- Added more information to the manual on how to debug pipelines and use `pytest-workflow` outside a virtual environment.

- Reworked code to use `tempfile.mkdtemp` to create a truly unique temporary working directory if the `--basetemp` flag is not used. This replaces the old code which dependeded on pytest internal code which was flagged as deprecated. Also more information was added to the manual about the use of `--basetemp`.
- Added a test case for WDL pipelines run with Cromwell and wrote an example for using WDL+Cromwell in the manual.
- Added `--tag` flag to allow for easier selection of workflows during testing.
- Added a test case for snakemake pipelines and wrote an example for using pytest-workflow with snakemake in the manual.

9.11 Version 0.3.0

- Improved the log output to look nicer and make workflow log paths easier to find in the test output.
- Fixed an error that polluted the log message with a pytest stacktrace when running more than one workflow. Measures are taken in our test framework to detect such issues in the future.
- Added the possibility to run multiple workflows simultaneously with the `--workflow-threads` or `--wt` flag.
- Made code easier to maintain by using stdlib instead of pytest's `py` lib in all of the code.
- Added a schema check to ensure that tests have unique names when whitespace is removed.

9.12 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.

9.13 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.