testimony Documentation

Release 1.0.1

sthirugn

Nov 25, 2020

Contents

Topics

CHAPTER 1

What is Testimony?

Testimony is an approach to document test cases in the Python source code using the function docstrings.

If your answer is `yes` to both the questions below, then `Testimony` is the right tool for you.

1. Are you using python to automate your test cases?
2. Are you tired of managing your test cases in a test case management tool?

Don't worry. Testimony can help you to use your Python automation framework as a test case repository tool.

CHAPTER 2

Advantages

Using Testimony brings lot of advantages to your project:

1. Avoid using a test case management tool to document test cases by leveraging function docstrings for the same.
2. Enforce standards for your test case docstrings.
3. Run with CI tools like Travis to validate your code after every check-in.
4. Save a lot of time from the conventional way of writing test cases using a test management tool.
5. Easily extract test case information using Testimony and port it to any test management tool.

CHAPTER 3

Test Case Docstring format

Testimony allows you to easily configure `Testimony tokens` which are the defined docstring items which will be used in test case parsing.

tokens Allowed values to be used as docstring items in your tests. Default tokens are `assert`, `bz`, `feature`, `setup`, `status`, `steps`, `tags`, `test` and `type`.

minimum-tokens minimum set of tokens that are needed for each of your tests. Default minimum tokens are `assert`, `feature` and `test`.

Note: To help test case parsing, make sure that each test case docstring has the tokens in the following format `:token:`. Also token matching is case insensitive.

3.1 Sample Test Case

A sample python test case with test case tokens is shown below:

```
def test_login_1(self):
    """Check if a user is able to login with valid userid and password

    More description for the test.

    :feature: Login
    :setup: Navigate to abc.com
    :steps:
        1. Launch the url
        2. Log in with valid user credentials
    :assert: Log in successful
    :bz: 1234567
    :automated: false
    """
```

In the above example, as you may guess - `feature`, `setup`, `steps`, `assert`, `bz`, `automated` are all tokens.

CHAPTER 4

How it works?

To understand how Testimony works, let's look at the `help` command:

```
$ testimony --help
  Usage: testimony [OPTIONS] REPORT [PATH]...

  Inspect and report on the Python test cases.

  Options:
  -j, --json              JSON output
  -n, --nocolor           Color output
  --tokens TEXT           Comma separated list of expected tokens
  --minimum-tokens TEXT   Comma separated list of minimum expected tokens
  -c, --config FILENAME   Configuration file (YAML)
  --help                  Show this message and exit.
```

Testimony does the following to parse the test case docstrings:

1. It captures all *Python Test modules* in the path(s) provided by the `PATH` argument.
    - As the definition implies, `PATH` accepts more than one value.
    - If `PATH` is a directory, then the directory and its subdirectories will be inspected for test modules as well.
2. Inside each identified test module, it looks for *Python Test case functions*
3. It then parses the function docstrings and extracts their tokens. Also, it creates namespaces for `module` and `class` level docstrings which will then be reused in the children tests. For example, if a module has a token called `feature`, then all tests in that module will inherit it by default. But the individual tests can choose to override this value by defining their own. The token lookup will happen in the following order and it will stop on the very first match:
    1. function level
    2. class level
    3. module level

CHAPTER 5

Installation

You can install Testimony from PyPI using pip:

```
pip install testimony
```

CHAPTER 6

Usage Examples

Note: For easy understanding of Testimony, this repository is already included with a sample python test module `tests/test_sample.py`. This module contains different test case format examples. The sample commands used below also use this data.

6.1 help command

See the *How it works?* section.

6.2 print command

Prints a nice summary of all captured tests with the parsed tokens for each test. Also it prints non-recognized tokens.

```
$ testimony print tests | head -n 27

tests/test_sample.py
====================

test_outside_class:8
--------------------

Assert:
 Testimony works with test functions

Feature:
 Test functions

Setup:
 Global setup
```

(continues on next page)

(continued from previous page)

```
Test:
 Test testimony works with test functions.


Testsample1::test_positive_login_1:27
-------------------------------------

Assert:
 Login is successful

Setup:
 Setup Testsample1
```

Note: The print command above uses the `head` command to show just one test case. Try without `head` command to see the entire output.

6.3 summary command

Gives a bird's-eye view of all the test cases in the given path. The report includes information such as:

- total number of test cases.
- number of test cases missing docstring.
- usage of different tokens across the given project.

For example:

```
$ testimony summary tests/

Total number of tests:          7
Test cases with no docstrings:  1 (14.29%)
Assert:                         5 (71.43%)
Bz:                             2 (28.57%)
Feature:                        4 (57.14%)
Setup:                          6 (85.71%)
Status:                         3 (42.86%)
Steps:                          6 (85.71%)
Tags:                           4 (57.14%)
Test:                           6 (85.71%)
Type:                           1 (14.29%)
```

6.4 validate command

Validates all the test cases in the given path. This command gives the required information which will help you identify the issues pertaining to each identified tests. Checks performed for each test are:

- docstring exists
- docstring can be parsed
- all required tokens are defined

- there are no tokens outside of expected tokens range
- all tokens have valid values (see *Tokens configuration*)

Note: To make easier integration with CI tools like `travis`, this command gives a non-zero return code if any of the checks above fails.

For example:

```
$ testimony validate tests/

tests/test_sample.py
====================

Testsample1::test_positive_login_1:27
-------------------------------------

* Docstring should have at least assert, feature, test token(s)
* Unexpected tokens:
  Bug: 123456
  Feture: Login - Positive
  Statues: Manual
  Types: Functional

Testsample1::test_positive_login_2:49
-------------------------------------

* Missing docstring.
* Docstring should have at least assert, feature, test token(s)

Testsample1::test_negative_login_5:87
-------------------------------------

* Docstring should have at least assert, feature, test token(s)

RSTFormattingTestCase::test_invalid_list_style:150
--------------------------------------------------

* Docstring has RST parsing issues. RST parser messages:

  * Enumerated list ends without a blank line; unexpected unindent.

      :Steps:
          1. Have a RST list on any of the tokens, like steps.
    >     2. Make sure one of the items on the list goes across multiple
          lines and the lines are not properly indented.


ConfigurationFileTestCase::test_multiple_invalid_keys:202
---------------------------------------------------------

* Unexpected tokens:
Caseimportance: Lowest

Total number of tests: 14
Total number of invalid docstrings: 5 (35.71%)
Test cases with no docstrings: 1 (7.14%)
```

(continues on next page)

(continued from previous page)

```
Test cases missing minimal docstrings: 3 (21.43%)
Test cases with unexpected tags: 2 (14.29%)
Test cases with unexpected token values in docstrings: 0 (0.00%)
Test cases with unparseable docstrings: 1 (7.14%)
```

6.5 Misc Options

--json A json output is provided when this option is specified.

--no-color a colored output is provided by default when the `termcolor` package is installed. This can be disabled by specifying this option.

CHAPTER 7

Tokens configuration

Tokens supported by Testimony can be configured with `--tokens`, `--minimum-tokens` and `--config` options.

`--tokens` takes comma-separated list of supported tokens. When testimony encounters token outside of this range, it will report it as error.

`--minimum-token` takes comma-separated list of required tokens. When testimony encounters test without all of tokens in this group, it will report it as error. Tokens specified here are automatically added to list of supported tokens (there is no need to specify single token in both `--minimum-tokens` and `--tokens`).

`--config` is path to YAML configuration file. YAML file should contain single map (equivalent of Python dict), where keys are names of tokens and values are maps consisting of `required`, `type` and other, type-dependant keys. Sample config files are provided in `tests` directory, as well as printed below:

```
---
Assert:
    required: True  # 'Assert' is required in each test
Feature:
    required: True
Test:
    required: True
# You can specify that token is not required explicitly, or leave it
# out - testimony will assume default value of 'False'
# Both tokens below are allowe, but not required
BZ:
    required: False
Setup: {}

# If 'type' is 'choice', 'choices' must be provided and must contain
# list of allowed values. 'casesensitive' declares if choices match
# should be done in case-sensitive way (default) or not
Status:
    required: False
    type: choice
    casesensitive: False
    choices:
```

(continues on next page)

(continued from previous page)

```
        - manual
        - automated
Steps: {}
Tags: {}
Type:
    required: False
```

CHAPTER 8

Project Contribution

8.1 How to Contribute?

1. Fork the repository on GitHub and make your changes
2. Test your changes
3. Send a pull request
4. Watch for the Travis update on the PR as it runs `flake8`
5. The PR will be merged after 2 ACKs

8.2 Author

This software is developed by Suresh Thirugn

8.3 Contributors

Og Maciel
Corey Welton
Elyézer Rezende

CHAPTER 9

Appendix

9.1 Python Test Modules

All files which match the patterns `test_*.py` and `*_test.py` are considered Python test modules.

9.2 Python Test case functions

Python functions whose names start with `test_`