testimony Documentation

Release 1.0.1

sthirugn

July 12, 2016

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.

1. 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`.
2. 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 prefixed with `@` and suffixed with `:`. Otherwise, you may see incorrect results.

3.1 Sample Test Case

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

```python
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
  --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 *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_positive_login_1
---------------------

Assert:
 Login is successful

Setup:
 Setup Testsample1

Steps:
 1. Login to the application with valid credentials

Tags:
 t1, t2, t3

Test:
 Login with right credentials

Unexpected tokens:
  Bug: 123456
  Feture: Login - Positive
  Statues: Manual
  Types: Functional
```

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.

```
$ 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 helps ensure that all your tests have the minimal set of tokens defined. This command gives the required information which will help you identify the issues pertaining to each identified tests.

Note: To make easier integration with CI tools like `travis`, this command gives a non-zero return code when:

- a test case is missing the docstring.
- a test case is missing minimal set of tokens.
- a test case has an unexpected token.

```
$ testimony validate tests/

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

test_positive_login_1
---------------------
```

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

test_positive_login_2
---------------------

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

test_negative_login_5
---------------------

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

Total number of tests: 7
Total number of invalid docstrings: 3 (42.86%)
Test cases with no docstrings: 1 (14.29%)
Test cases missing minimal docstrings: 3 (42.86%)
Test cases with invalid tags: 1 (14.29%)
```

6.5 Misc Options

1. `--json` - a json output is provided when this option is specified.
2. `--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

Project Contribution

7.1 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

7.2 Author

This software is developed by Suresh Thirugn

7.3 Contributors

Og Maciel
Corey Welton
Elyézer Rezende

CHAPTER 8

Appendix

8.1 Python Test Modules

All files which match the following criteria:

- file names start with `test_`
- file extension matches `.py`

8.2 Python Test case functions

Python functions whose names start with `test_`