nitpicker Documentation

Release 0.3.0

Kenneth Reitz

Sep 15, 2018

Contents

Contents:

CHAPTER 1

About Nitpicker

1.1 Nitpicker

Nitpicker is a CLI tool for QA written in Python

1.1.1 Motivation

The project has been started to fix some problems that many developers and testers might be familiar to:

1. QA tests are not under version control with the code. Why not? As developers, we would like to do some review of tests like code review. As a manger I would be calm knowing that all QA plans and cases are stored with the code on Git repository always available.

2. QA tests stay apart from the develop cycle. I can ban a merge request if it breaks my unit or integration tests because I see it at once by using CI tools. I believe it is possible for manual tests too. I want my CI tool to check if a tester do all the needed tests.

3. A QA tool should be interactive. When you see a whole test case with all the steps it is hard not to jump between them trying to do test as fast as possible. When a tester is in a dialogue with a tool and goes step-by-step, they can test more carefully. Especially, if the tool keep time tracking automatically.

1.1.2 How does it work?

All your tests and run reports are stored in YAML format with the code which they test.

```
project
|-src/
|-docs/
|-qa/
  |-feature_1/
  |-feature_2/
    |-plan_1/
```

(continues on next page)

(continued from previous page)

```
      |-test_case1.yml
      |-test_case2.yml
      |-test_case3.yml
      |-runs/
         |-20180820_232000_run.report
         |-20180820_232010_run.report
```

Nitpicker provides command to create a test case:

```
python -m nitpicker add test_case -p feature_1.plan_1
```

Then you should write the case by using your favourite text editor. It is a not bad idea to commit and push it, so your teammate can review the case before you run the plan which the case belongs to.

Now you can run the test plan:

```
python -m nitpicker run feature_1.plan_1
```

The program runs all the cases in the interactive mode leading the tester step by step. The results of the run will be written in directory *runs* in YAML format.

After all the test cases have been run you can push the reports into the git repo, so your CI server can check if all the test runs are passed

```
python -m nitpicker check --all-runs-passed
```

The project uses itself for testing. You can find *qa* directory in the repo. Also you can run some plans for demonstration.

1.1.3 Installation

```
pip install nitpicker
```

or

```
python -m install nitpicker
```

Currently Nitpicker supports Python 3.3 and newer

1.1.4 Documentation

See the last documentation here.

CHAPTER 2

Test Storing

Nitpicker provides storing test cases as files in YAML format:

```
project
|-qa/
  |-feature_1/
  |-feature_2/
    |-plan_1/
      |-test_case1.yml
      |-test_case2.yml
      |-test_case3.yml
      |-runs/
         |-20180820_232000_run.report
         |-20180820_232010_run.report
```

Directory *qa* is the root directory of all the QA tests (QA directory). It contains all tests plans and its test cases and test run reports as well. Test plans are represented by directories which include other test plans directory and test cases. So we have some hierarchy of the test plans.

Each plan contains all its run reports in directory *runs*.

Currently Nitpicker uses the name convention:

- A test case must have extension *.yml*
- A test run must end with *_run* and have extension *.report*

2.1 Test Case Format

A test case file is written in YAML format and has these following structure:

```
created: 2018-09-15 04:54:39
author: Aleksey Timin
email: atimin@gmail.com
```

(continues on next page)

(continued from previous page)

```
description: Checking if all the last runs are passed is success
tags: commands, fuzzy
setup:
  - Run command 'python -m nitpicker -d test_qa add test_test_case -p commands'
  - Save the case without changes and close the editor
  - Run command 'python -m nitpicker -d test_qa run commands' and passed all steps
steps:
  - Run command 'python -m nitpicker -d test_qa check --all-runs-passed'
    => It should be success

teardown:
  - Delete test_qa directory
```

All test case files should have the following mapping:

- *created* - The time when the test case was created in format %Y-%m-%d %H:%M:%S
- *author* - The author's name
- *email* - The author's email
- *description* - The short description of the case that should be displayed in all reports
- *tags* - The tags separated by comma (not implemented yet)
- *setup* - The actions that should be done before the test starts
- *steps* - The steps that contains the tester's actions and the expectations separated by symbol '=>'
- *teardown* - The actions that should be done after the test has been run

2.2 Test Run Format

A test run report file is generated by Nitpicker's command *run* in YAML format and has the following structure:

```
cases:
  add_new_case.yml:
    comment: ''
    description: Add a new case
    failed_action: 'Run command ''python -m nitpicker -r test_qa add test_test_case
      -p commands'' '
    failed_reaction: ' The new case should be opened in the editor'
    failed_step: 1
    finished: '2018-09-15 05:10:52'
    started: '2018-09-15 05:08:52'
    status: failed
  add_new_case_in_force.yml:
    description: Add a new case in force mode
    finished: '2018-09-15 05:10:56'
    started: '2018-09-15 05:10:53'
    status: passed
email: atimin@gmail.com
finished: '2018-09-15 05:10:56'
started: '2018-09-15 05:08:52'
tester: Aleksey Timin
```

CHAPTER 3

Nitpicker's Workflow

Nitpicker is created to testers and developer have common workflow and it's supposed that the QA tests are stored with the source code in CVS repository and new features are developed in the separated branches:

```
new_feature  master
    |          |
    *          *
    |          |
    *          |
    |          |
    +---------*
```

3.1 Step 1. Add new test cases

The tester starts their a new branch from *new_feature* for the new test cases and add a new case in plan *test_new_feature*

```
git checkout -b qa_new_feature
python -m nitpicker add some_new_case -p test_new_feature
```

The new case is opened in a text editor and the tester fills in it with some steps (see *Test Storing*). Then the new case can be committed and pushed to the repository

```
git add qa/test_new_feature/some_new_case.yml
git commit -m "Add some_new_case.yml"
git push origin qa_new_feature
```

If your team practices the code review (I hope it does), then the developer can have a look at the cases:

```
qa_new_feature
      |
      * - ("Add some_new_case.yml")
      |
```

(continues on next page)

(continued from previous page)

```
      | new_feature
      |      |
      |      *      master
      |      |          |
      ------*          *
            |          |
            *          |
            |          |
            +----------*
```

A test plan can contain test cases as many as you wants. And the tester can repeat command *python -m nitpicker add* for each test case or copy and rename the file of the first one.

3.2 Step 2. Run new test cases

In order to run all the created test cases in the test plan the tester must run command:

```
python -m nitpicker run test_new_feature
```

Nitpicker runs each test case in the interactive mode and the tester should answer if each step of the test case is passed or failed. After all the cases have been run, Nitpicker creates a new run report in directory *qa/test_new_feature/runs/20180820_232000_run.report*. (Note: The new report's name contains the time when the run was finished)

Then the tester commits the run report and push to the repository:

```
git add qa/test_new_feature/runs/20180820_232000_run.report
git commit -m "Run test plane 'test_new_feature'"
git push origin qa_new_feature
```

Wait a minute.. Why do we need to commit autogenerated data!? Because we have a CI server and Nitpicker provides some features for it too (see *Continuous Integration*).

3.3 Step 3. Merging

After step 2 the repository has the following graph:

```
qa_new_feature
      |
      * - ("Run test plane 'test_new_feature'")
      |
      * - ("Add some_new_case.yml")
      |
      | new_feature
      |      |
      |      *      master
      |      |          |
      ------*          *
            |          |
            *          |
            |          |
            +----------*
```

If all the tests are passed and the CI pipeline has no errors the maintainer can merge the branches in two steps:

```
git fetch origin

# Merge the QA branch
git checkout new_feature
git merge origin/qa_new_feature

# Merge the feature branch
git checkout master
git merge qa_new_feature

git push origin master
```

CHAPTER 4

Continuous Integration

Nitpicker has a special command to run on the side of the CI server:

```
python -m nitpicker check --all-runs-passed --has-new-runs
```

Flag *–all-runs-passed* provides a check if all the last run reports of the project have only passed tests. If the check failed the program exists with error 1.

Flag *–has-new-runs* provides a check if the current branch has some new runs comparing the main branch (*master* by default). If the check failed the program exists with error 1.

CHAPTER 5

Indices and tables

- genindex
- modindex
- search