

Extracted from:

# Python Testing with pytest

Simple, Rapid, Effective, and Scalable

This PDF file contains pages extracted from *Python Testing with pytest*, published by the Pragmatic Bookshelf. For more information or to purchase a paperback or PDF copy, please visit <http://www.pragprog.com>.

Note: This extract contains some colored text (particularly in code listing). This is available only in online versions of the books. The printed versions are black and white. Pagination might vary between the online and printed versions; the content is otherwise identical.

Copyright © 2017 The Pragmatic Programmers, LLC.

All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the publisher.

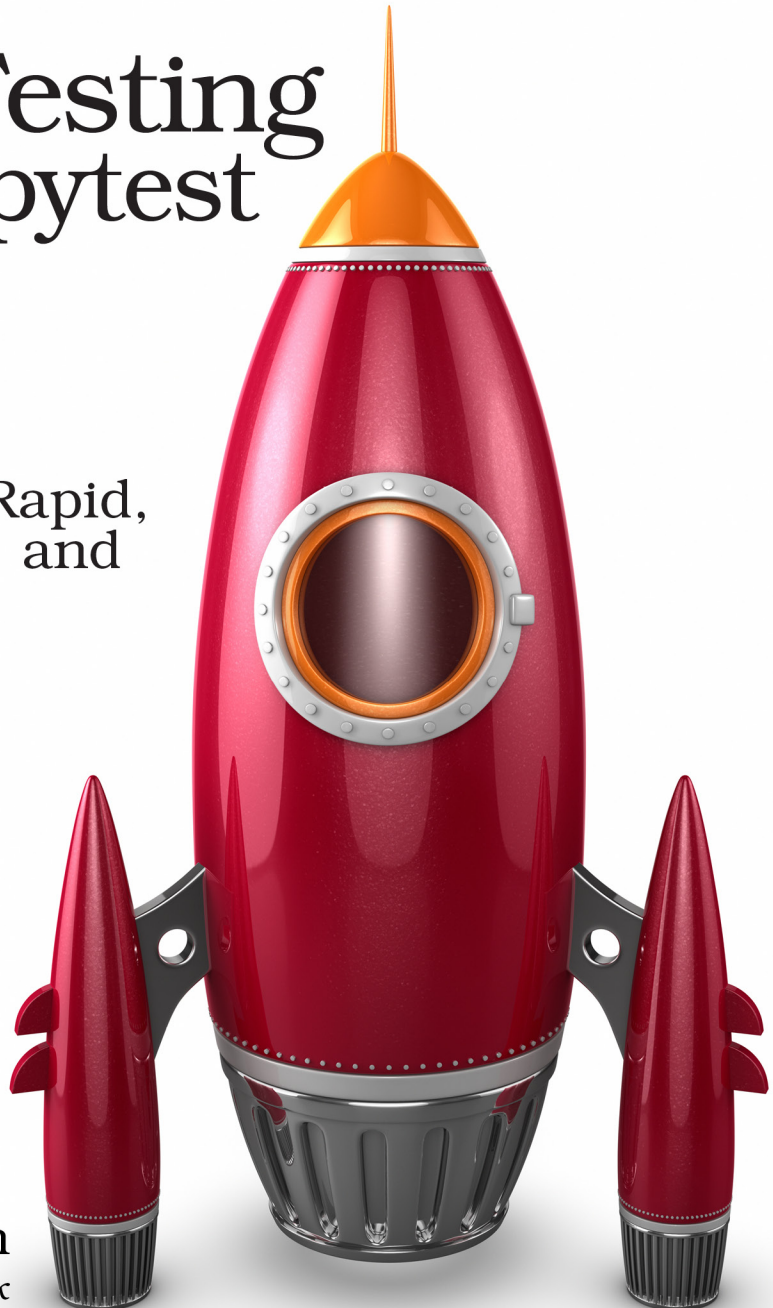
The Pragmatic Bookshelf

Raleigh, North Carolina

The  
Pragmatic  
Programmers

# Python Testing with pytest

Simple, Rapid,  
Effective, and  
Scalable



**Brian Okken**

*edited by Katharine Dvorak*

# Python Testing with pytest

Simple, Rapid, Effective, and Scalable

Brian Okken

The Pragmatic Bookshelf

Raleigh, North Carolina



Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and The Pragmatic Programmers, LLC was aware of a trademark claim, the designations have been printed in initial capital letters or in all capitals. The Pragmatic Starter Kit, The Pragmatic Programmer, Pragmatic Programming, Pragmatic Bookshelf, PragProg and the linking *g* device are trademarks of The Pragmatic Programmers, LLC.

Every precaution was taken in the preparation of this book. However, the publisher assumes no responsibility for errors or omissions, or for damages that may result from the use of information (including program listings) contained herein.

Our Pragmatic books, screencasts, and audio books can help you and your team create better software and have more fun. Visit us at <https://pragprog.com>.

For sales, volume licensing, and support, please contact [support@pragprog.com](mailto:support@pragprog.com).

For international rights, please contact [rights@pragprog.com](mailto:rights@pragprog.com).

Copyright © 2017 The Pragmatic Programmers, LLC.

All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the publisher.

Printed in the United States of America.

ISBN-13: 978-1-68050-240-4

Encoded using the finest acid-free high-entropy binary digits.

Book version: B1.0—May 17, 2017

## Testing Plugins

Plugins are code that needs to be tested just like any other code. However, testing a change to a testing tool is a little tricky. When we developed the plugin code in [Writing Your Own Plugins, on page ?](#), we tested it manually by using a sample test file, running pytest against it, and looking at the output to make sure it was right. We can do the same thing in an automated way using a plugin called `pytester` that ships with `pytest` but is disabled by default.

Our test directory for `pytest-nice` has two files: `conftest.py` and `test_nice.py`. To use `pytester` we need to add just one line to `conftest.py`:

```
ch5/pytest-nice/tests/conftest.py
pytest_plugins = 'pytester'
```

This turns on the `pytester` plugin. We will be using a fixture called `testdir` that becomes available when `pytester` is enabled.

Often, tests for plugins take on the form we've described in manual steps:

1. Make an example test file.
2. Run `pytest` with or without some options in the directory that contains our example file.
3. Examine the output.
4. Possibly check the result code—0 for all passing, 1 for some failing.

Let's look at one example:

```
ch5/pytest-nice/tests/test_nice.py
def test_pass_fail(testdir):

    # create a temporary pytest test module
    testdir.makepyfile("""
        def test_pass():
            assert 1 == 1

        def test_fail():
            assert 1 == 2
    """)

    # run pytest
    result = testdir.runpytest()

    # fnmatch_lines does an assertion internally
    result.stdout.fnmatch_lines([
        '*.F', # . for Pass, F for Fail
    ])

    # make sure that that we get a '1' exit code for the testsuite
    assert result.ret == 1
```

The `testdir` fixture automatically creates a temporary directory for us to put test files. It has a method called `makepyfile()` that allows us to put in the contents of a test file. In this case, we are creating two tests: one that passes and one that fails.

We run `pytest` against the new test file with `testdir.runpytest()`. We can pass in options if we want. The return value can then be examined further, and is of type `RunResult`.<sup>5</sup>

Usually, I look at `stdout` and `ret`. For checking the output like we did manually, use `fnmatch_lines`, passing in a list of regular expression strings that we want to see in the output, and then making sure that `ret` is 0 for passing sessions and 1 for failing sessions.

We can use our example file for more tests. So instead of duplicating that code, let's make a fixture:

```
ch5/pytest-nice/tests/test_nice.py
@pytest.fixture()
def sample_test(testdir):
    testdir.makepyfile("""
        def test_pass():
            assert 1 == 1

        def test_fail():
            assert 1 == 2
    """)
    return testdir
```

Now, for the rest of the tests we can use `sample_test` as a directory that already contains our sample test file. Here are the tests for the other option variants:

```
ch5/pytest-nice/tests/test_nice.py
def test_with_nice(sample_test):
    result = sample_test.runpytest('--nice')
    result.stdout.fnmatch_lines(['*.0', ]) # . for Pass, 0 for Fail
    assert result.ret == 1

def test_with_nice_verbose(sample_test):
    result = sample_test.runpytest('-v', '--nice')
    result.stdout.fnmatch_lines([
        '*:test_fail OPPORTUNITY for improvement',
    ])
    assert result.ret == 1

def test_not_nice_verbose(sample_test):
    result = sample_test.runpytest('-v')
```

5. [https://docs.pytest.org/en/latest/writing\\_plugins.html#\\_pytest.pytester.RunResult](https://docs.pytest.org/en/latest/writing_plugins.html#_pytest.pytester.RunResult)

```

result.stdout.fnmatch_lines([
    '*::test_fail FAILED',
])
assert result.ret == 1

```

Just a couple more tests to write. Let's make sure our thank-you message is in the header:

```

ch5/pytest-nice/tests/test_nice.py
def test_header(sample_test):

    result = sample_test.runpytest()
    result.stdout.fnmatch_lines([
        'Thanks for running the tests.'
    ])
    assert result.ret == 1

```

This could have been part of the other tests also, but I like to have it in a separate test so that one test checks one thing.

Finally, let's check the help text:

```

ch5/pytest-nice/tests/test_nice.py
def test_help_message(testdir):
    result = testdir.runpytest('--help')

    # fnmatch_lines does an assertion internally
    result.stdout.fnmatch_lines([
        'nice:',
        '*--nice*nice: turn FAILED into OPPORTUNITY for improvement',
    ])

```

I think that's a pretty good check to make sure our plugin works.

To run the tests, let's start in our `pytest-nice` directory and make sure our plugin is installed. We do this either by installing the `.zip.gz` file or installing the current directory in editable mode:

```

$ cd pytest-nice
$ ls
README.txt  pytest_nice.py  setup.py  tests/
$ pip install -e .
Obtaining file:///path/to/code/ch5/pytest-nice
Installing collected packages: pytest-nice
  Found existing installation: pytest-nice 0.1.0
  Uninstalling pytest-nice-0.1.0:
    Successfully uninstalled pytest-nice-0.1.0
  Running setup.py develop for pytest-nice
Successfully installed pytest-nice
$ pytest -v
===== test session starts =====
cachedir: .cache

```

```
Thanks for running the tests.
plugins: nice-0.1.0
collected 6 items
```

```
tests/test_nice.py::test_pass_fail PASSED
tests/test_nice.py::test_with_nice PASSED
tests/test_nice.py::test_with_nice_verbose PASSED
tests/test_nice.py::test_not_nice_verbose PASSED
tests/test_nice.py::test_header PASSED
tests/test_nice.py::test_help_message PASSED
```

```
===== 6 passed in 0.37 seconds =====
```

```
$ pip uninstall pytest-nice
```

```
Uninstalling pytest-nice-0.1.0:
```

```
  /path/to/venv/lib/python3.6/site-packages/pytest-nice.egg-link
```

```
Proceed (y/n)? y
```

```
  Successfully uninstalled pytest-nice-0.1.0
```

Yay! All the tests pass. A great way to learn more about plugin testing is to look at the tests contained in other pytest plugins available through PyPI.

## Creating a Distribution

Believe it or not, we are almost done with our plugin. From the command line, we can use this setup.py file to create a distribution:

```
$ cd pytest-nice/
```

```
$ python setup.py sdist
```

```
running sdist
```

```
...
```

```
creating dist
```

```
Creating tar archive
```

```
...
```

```
$ ls dist
```

```
pytest-nice-0.1.0.tar.gz
```

(Note that sdist stands for “source distribution.”)

Within pytest-nice, a dist directory contains a new file called pytest-nice-0.1.0.tar.gz. This file can now be used anywhere to install our plugin, even in place:

```
$ pip install dist/pytest-nice-0.1.0.tar.gz
```

```
Processing ./dist/pytest-nice-0.1.0.tar.gz
```

```
Installing collected packages: pytest-nice
```

```
  Running setup.py install for pytest-nice ... done
```

```
Successfully installed pytest-nice-0.1.0
```

However, you can put your “.tar.gz” files anywhere you’ll be able to get at them to use and share.



## Distributing Plugins Through a Shared Directory

pip already supports installing packages from shared directories, so all we have to do to distribute our plugin through a shared directory is pick a location we can remember and put the `.tar.gz` files for our plugins there. Let's say we put `pytest-nice-0.1.0.tar.gz` into a directory called `myplugins`.

To install `pytest-nice` from `myplugins`:

```
$ pip install --no-index --find-links myplugins pytest-nice
```

The `--no-index` tells pip to not go out to PyPI to look for what we want to install. The `--find-links myplugins` tells PyPI to look in `myplugins` for packages to install. And of course, `pytest-nice` is what we want to install.

If we've done some bug fixes and there are newer versions in `myplugins`, we can upgrade by adding `--upgrade`:

```
$ pip install --upgrade --no-index --find-links myplugins pytest-nice
```

This is just like any other use of pip, but with the `--no-index --find-links myplugins` added.

## Distributing Plugins Through PyPI

If you want to share your plugin with the world, there are a few more steps we need to do. Actually, there are quite a few more steps. However, because this book isn't focused on contributing to open source, I recommend checking out the thorough instruction found in the Python Packaging User Guide.<sup>6</sup>

When you are contributing a pytest plugin, another great place to start is by using the `cookiecutter-pytest-plugin`<sup>7</sup>:

```
$ pip install cookiecutter
$ cookiecutter https://github.com/pytest-dev/cookiecutter-pytest-plugin
```

This project first asks you some questions about your plugin. Then it creates a good directory for you to explore and fill in with your code. Walking through this is beyond the scope of this book; however, please keep this project in mind. It is supported by core pytest folks, and they will make sure this project stays up-to-date.

6. <https://packaging.python.org/distributing>

7. <https://github.com/pytest-dev/cookiecutter-pytest-plugin>

## Exercises

In `ch4/cache/test_slower.py`, there is an autouse fixture called `check_duration()`. We used it in the Chapter 4 exercises as well. Now, let's make a plugin out of it.

1. Create a directory for “slower” similar to the directory described in [Creating an Installable Plugin, on page ?](#).
2. Fill out all the files of the directory to make “slower” an installable plugin.
3. Write test code for the plugin. What kinds of test cases are needed?
4. Take a look at the Python Package Index<sup>8</sup> and search for “pytest-.” Find a pytest plugin that looks interesting to you.
5. Install the plugin you chose and try it out on Tasks tests. What plugin did you pick?

## What's Next

We've used `conftest.py` a lot so far in this book. There are also configuration files that affect how pytest runs, such as `pytest.ini`. In the next chapter, we'll run through the different configuration files and learn what you can do there to make your testing life easier.

---

8. <https://pypi.python.org/pypi>