Why I Use py.test and You Probably Should Too

Andy Todd – @andy47

PyCon Australia 2013

How to *Start* Unit Testing

Andy Todd – @andy47

PyCon Australia 2013

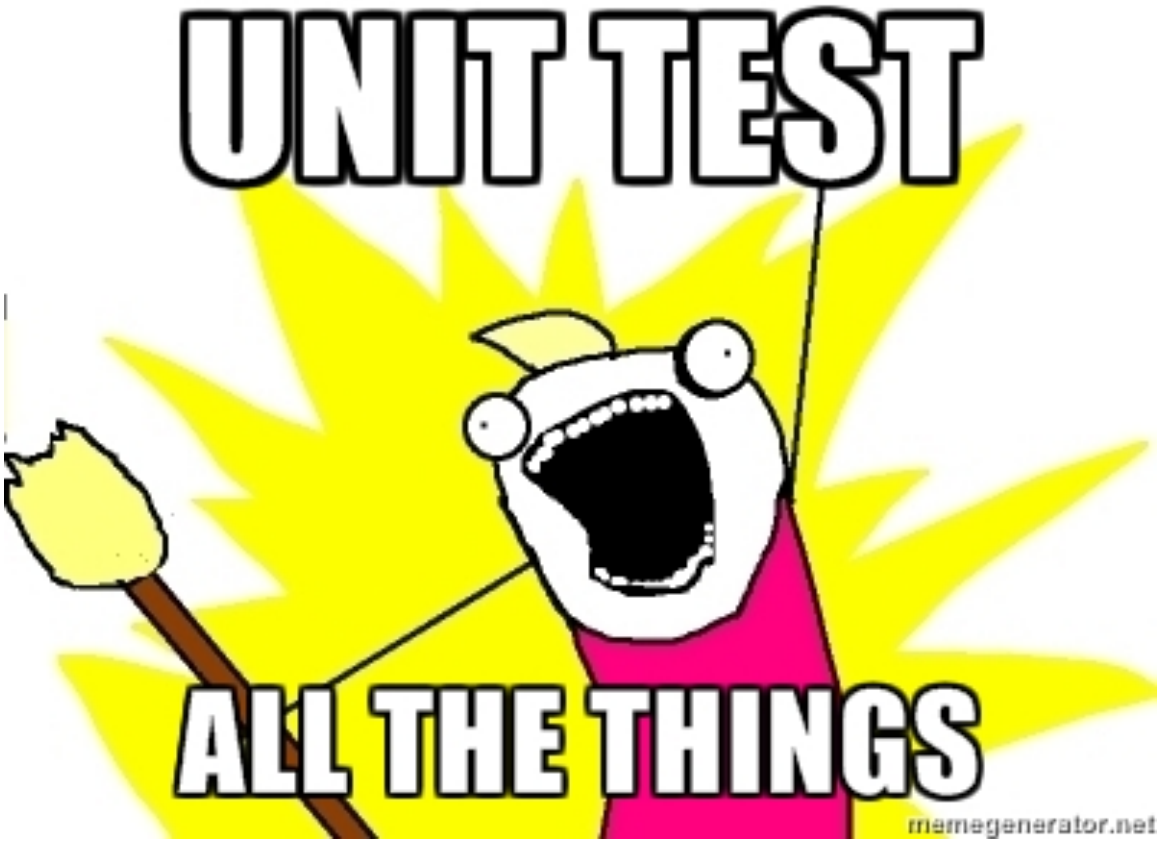
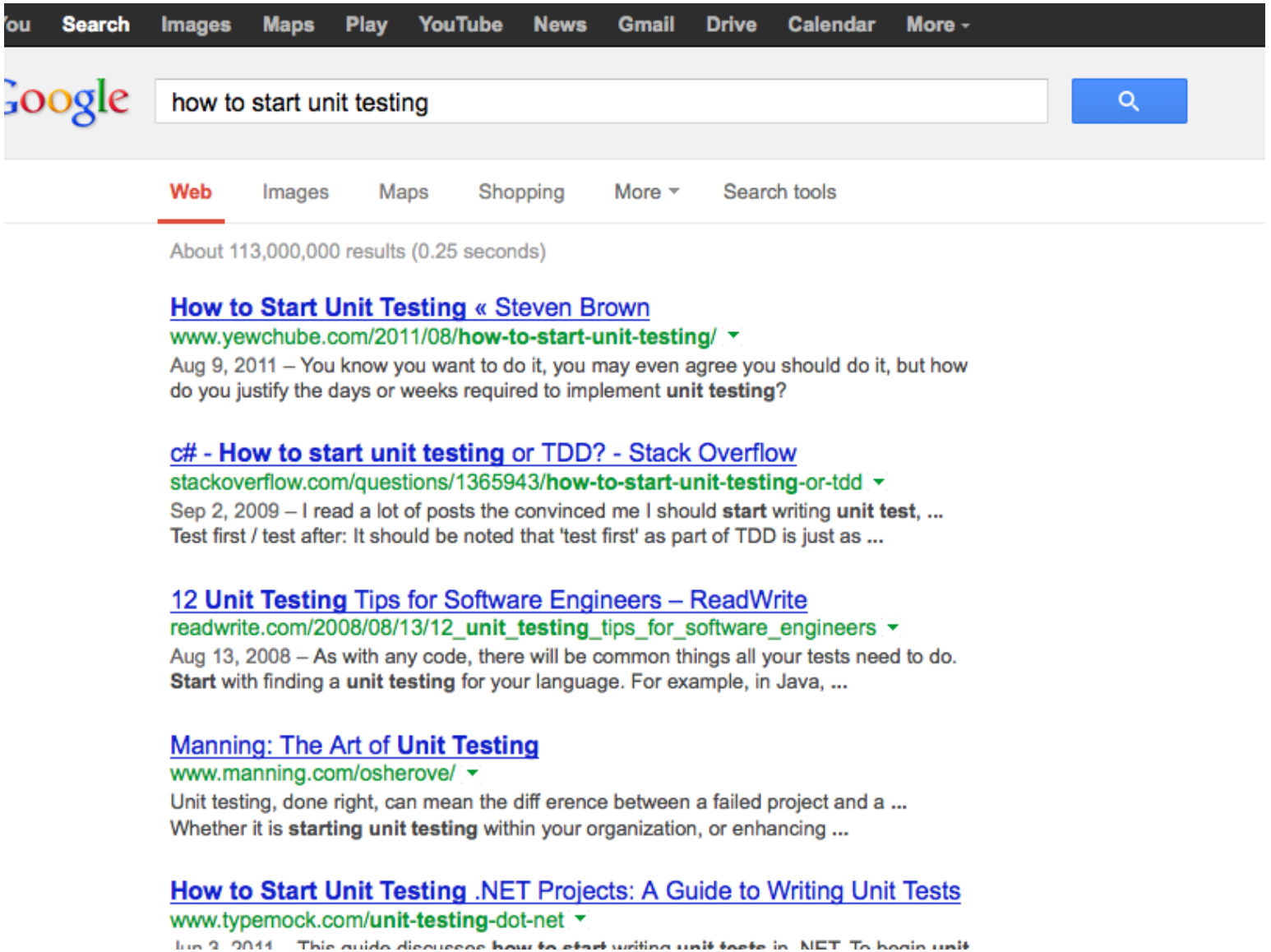


Image from http://techblog.appirio.com/2013/03/deliver-value-with-unit-testing-in-cloud.html

Yes, another talk on how great unit testing is


You Search Images Maps Play YouTube News Gmail Drive Calendar More
Google
how to start unit testing
Web Images Maps Shopping More Search tools
About 113,000,000 results (0.25 seconds)
How to Start Unit Testing « Steven Brown
www.yewchube.com/2011/08/how-to-start-unit-testing/
Aug 9, 2011 – You know you want to do it, you may even agree you should do it, but how do you justify the days or weeks required to implement unit testing?
c# - How to start unit testing or TDD? - Stack Overflow
stackoverflow.com/questions/1365943/how-to-start-unit-testing-or-tdd
Sep 2, 2009 – I read a lot of posts the convinced me I should start writing unit test, ... Test first / test after: It should be noted that 'test first' as part of TDD is just as ...
12 Unit Testing Tips for Software Engineers – ReadWrite
readwrite.com/2008/08/13/12_unit_testing_tips_for_software_engineers
Aug 13, 2008 – As with any code, there will be common things all your tests need to do. Start with finding a unit testing for your language. For example, in Java, ...
Manning: The Art of Unit Testing
www.manning.com/osherove/
Unit testing, done right, can mean the diff erence between a failed project and a ... Whether it is starting unit testing within your organization, or enhancing ...
How to Start Unit Testing .NET Projects: A Guide to Writing Unit Tests
www.typemock.com/unit-testing-dot-net

What is Unit Testing?

Unit Testing

Activity

1. any repeatable activity that checks that the individual units of code within a module or application work as expected.

Isolated

http://www.flickr.com/photos/mabufeu/952012124

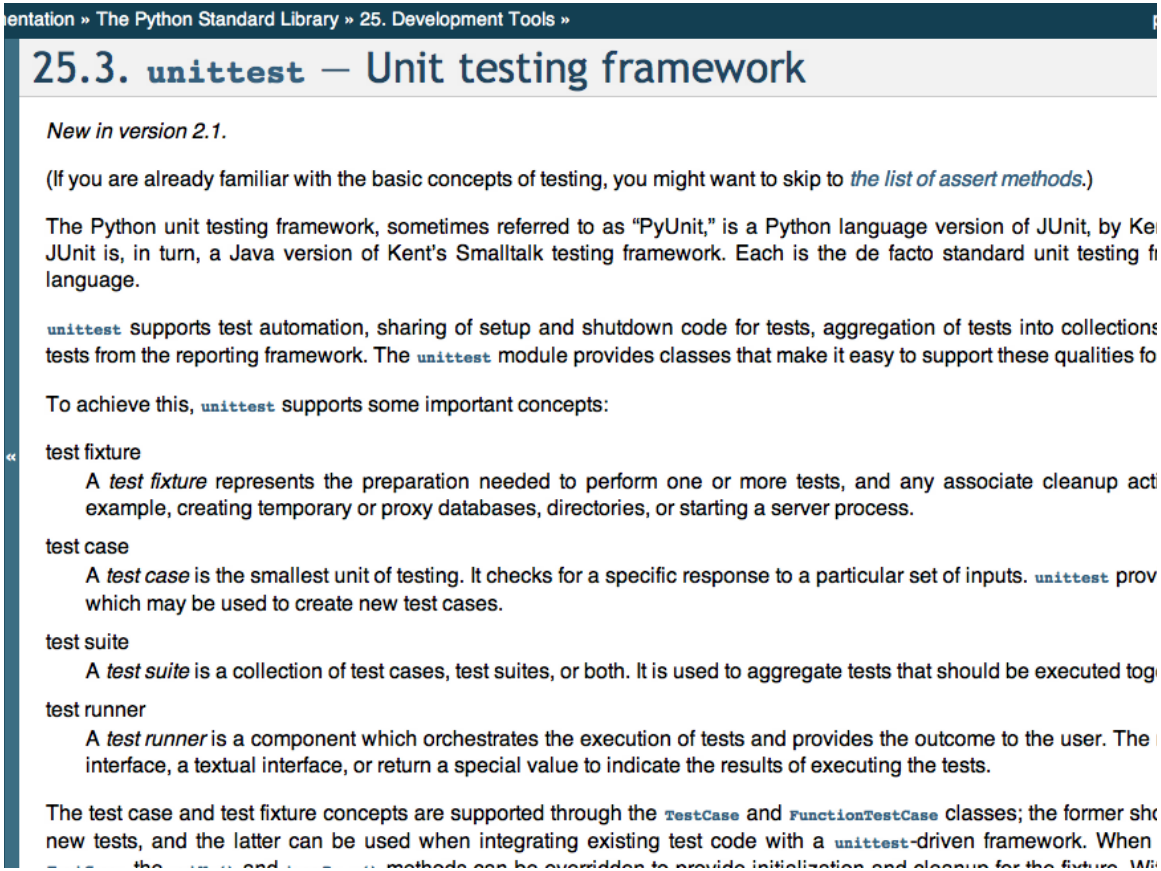
Automated

http://www.flickr.com/photos/s__i/8543101413

Grouped into Batches

http://www.flickr.com/photos/24736216@N07/9068587656

Unit Testing in Python



entation » The Python Standard Library » 25. Development Tools »

25.3. `unittest` — Unit testing framework

New in version 2.1.

(If you are already familiar with the basic concepts of testing, you might want to skip to *the list of assert methods*.)

The Python unit testing framework, sometimes referred to as "PyUnit," is a Python language version of JUnit, by Ke
JUnit is, in turn, a Java version of Kent's Smalltalk testing framework. Each is the de facto standard unit testing f
language.

`unittest` supports test automation, sharing of setup and shutdown code for tests, aggregation of tests into collections
tests from the reporting framework. The `unittest` module provides classes that make it easy to support these qualities fo

To achieve this, `unittest` supports some important concepts:

test fixture
: A *test fixture* represents the preparation needed to perform one or more tests, and any associate cleanup act
example, creating temporary or proxy databases, directories, or starting a server process.

test case
: A *test case* is the smallest unit of testing. It checks for a specific response to a particular set of inputs. `unittest` prov
which may be used to create new test cases.

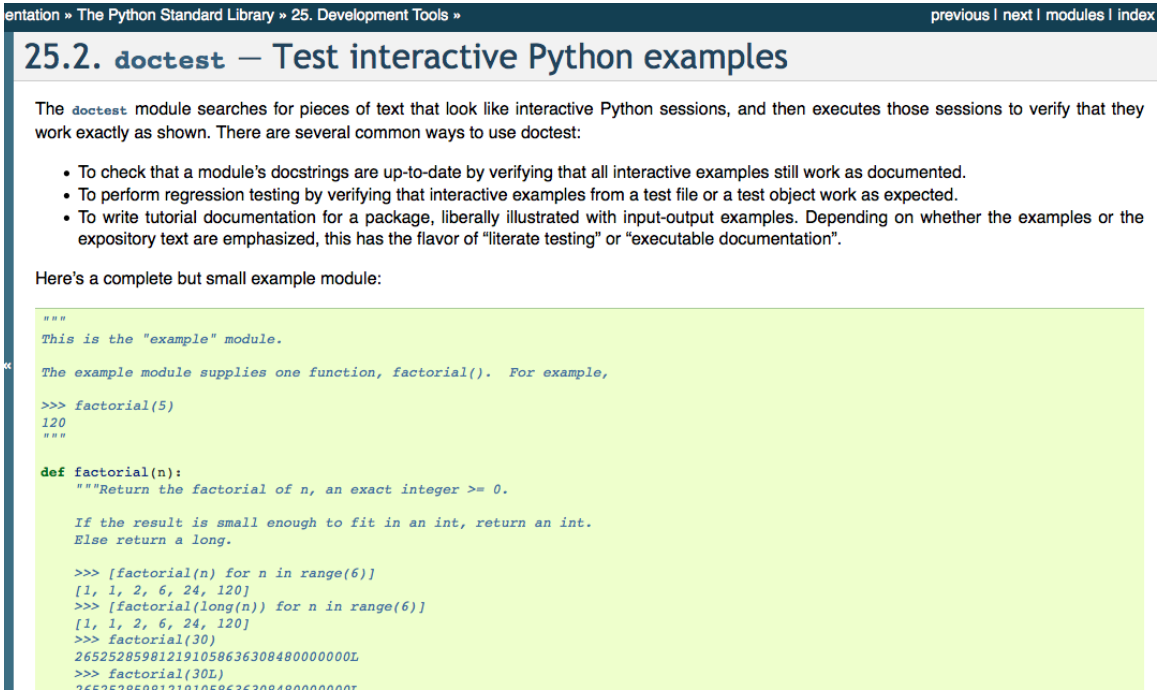
test suite
: A *test suite* is a collection of test cases, test suites, or both. It is used to aggregate tests that should be executed tog

test runner
: A *test runner* is a component which orchestrates the execution of tests and provides the outcome to the user. The
interface, a textual interface, or return a special value to indicate the results of executing the tests.

The test case and test fixture concepts are supported through the `TestCase` and `FunctionTestCase` classes; the former sh
new tests, and the latter can be used when integrating existing test code with a `unittest`-driven framework. When

Unittest

http://docs.python.org/2/library/unittest.html



entation » The Python Standard Library » 25. Development Tools » previous | next | modules | index

25.2. doctest — Test interactive Python examples

The doctest module searches for pieces of text that look like interactive Python sessions, and then executes those sessions to verify that they work exactly as shown. There are several common ways to use doctest:

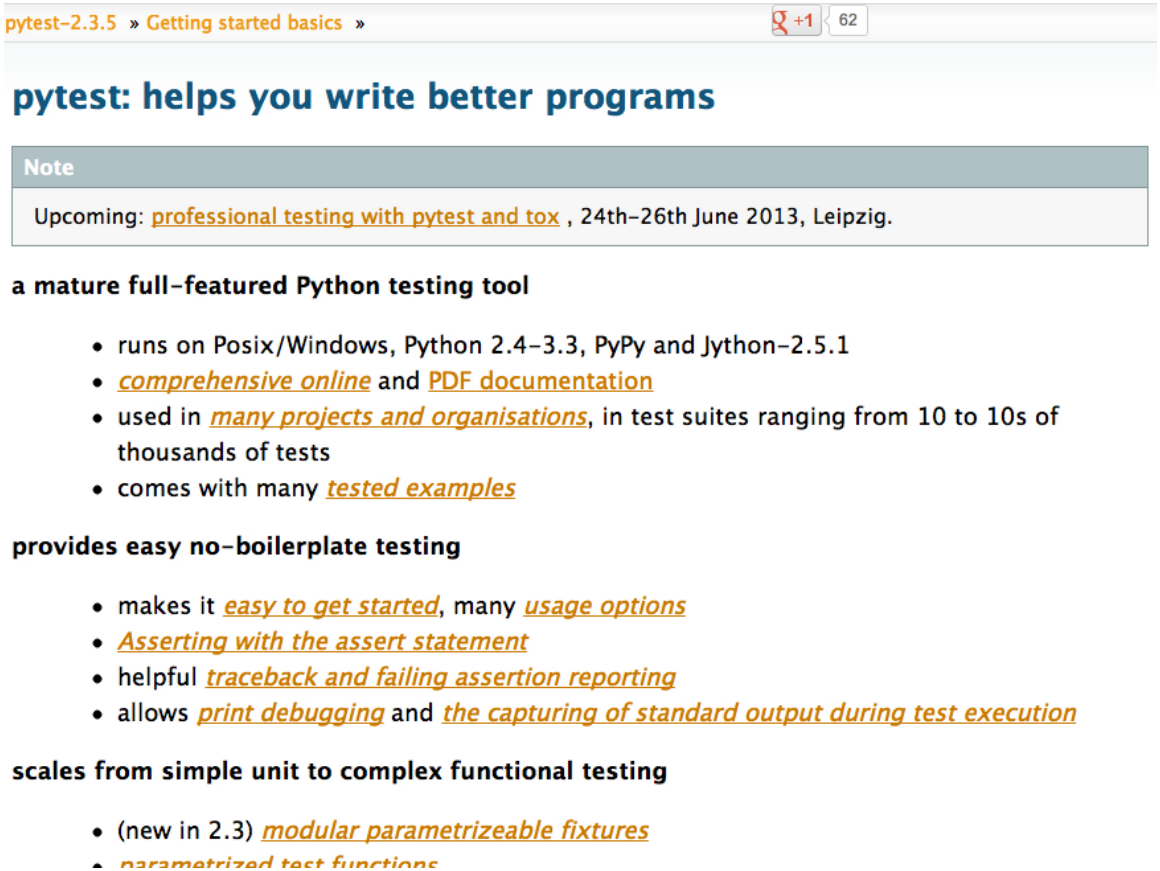
- To check that a module's docstrings are up-to-date by verifying that all interactive examples still work as documented.
- To perform regression testing by verifying that interactive examples from a test file or a test object work as expected.
- To write tutorial documentation for a package, liberally illustrated with input-output examples. Depending on whether the examples or the expository text are emphasized, this has the flavor of "literate testing" or "executable documentation".

Here's a complete but small example module:

```
"""
This is the "example" module.

The example module supplies one function, factorial().  For example,

>>> factorial(5)
120
"""

def factorial(n):
    """Return the factorial of n, an exact integer >= 0.

    If the result is small enough to fit in an int, return an int.
    Else return a long.

    >>> [factorial(n) for n in range(6)]
    [1, 1, 2, 6, 24, 120]
    >>> [factorial(long(n)) for n in range(6)]
    [1, 1, 2, 6, 24, 120]
    >>> factorial(30)
    265252859812191058636308480000000L
    >>> factorial(30L)
```

doctest

http://docs.python.org/2/library/doctest.html



py.test

http://pytest.org/latest/

Why py.test?

- Simple unit test discovery
- Set up at module, class and test level
- Less boilerplate
- Full feature set when needed

```
def parse_connection(connection_string):
    """Work out the user name and password in connection_string

    Connection string should be of the form 'database://username@password'
    """
    if '@' in connection_string:
        # Username is characters between '://' and '@'
        slash_position = connection_string.find('://')
        at_position = connection_string.find('@')
        user_name = connection_string[slash_position+3:at_position]
        password = connection_string[at_position+1:]
        return user_name, password

if __name__ == "__main__":
    print parse_connection('mysql://user@pass')
    print parse_connection('a random string')
```

Example code

https://gist.github.com/andy47/5893154

```python
from parse_conn import parse_connection
import unittest

class InvalidInputs(unittest.TestCase):
    def testNoAt(self):
        """parse_connection should raise an exception
           for an invalid connection string"""
        self.assertRaises(ValueError, parse_connection, 'invalid uri')

if __name__ == "__main__":
    unittest.main()
```

unittest test case

https://gist.github.com/andy47/5893166

```
from parse_conn import parse_connection
import py.test

def test_not_at():
    py.test.raises(ValueError, parse_connection, 'invalid uri')
```

py.test test cases

https://gist.github.com/andy47/5893180

How Do I Start?

Just Write Some Tests

But Which Ones?

Bug Triage Unit Testing

- Write a test for the correct behaviour
- Change your code until it passes
- Repeat

New Feature Unit Testing

- Write test(s) for the new feature
- Change your code until it passes
- Repeat

```
class Portfolio(object):
    ...
    def get_portfolio(self, portfolio_code):
        ...
        stmt = """SELECT id FROM portfolios WHERE code=?"""
        cursor.execute(stmt, (portfolio_code,))
        portfolio_id = cursor.fetchone()[0]
        ...
    def add_value(self, portfolio_code, ...):
        ...
        stmt = """SELECT id FROM portfolios WHERE code=?"""
        cursor.execute(stmt, (portfolio_code,))
        portfolio_id = cursor.fetchone()[0]
        ...
```

Refactoring

https://gist.github.com/andy47/5893615

Avoid Premature Optimisation

- Write tests only for the code that you are changing
- Don’t need to test every possible scenario

Tested Code Looks Different

- Smaller methods/functions
- More verbose

Makes Code Smaller

“Testing code that does lots of things is difficult.

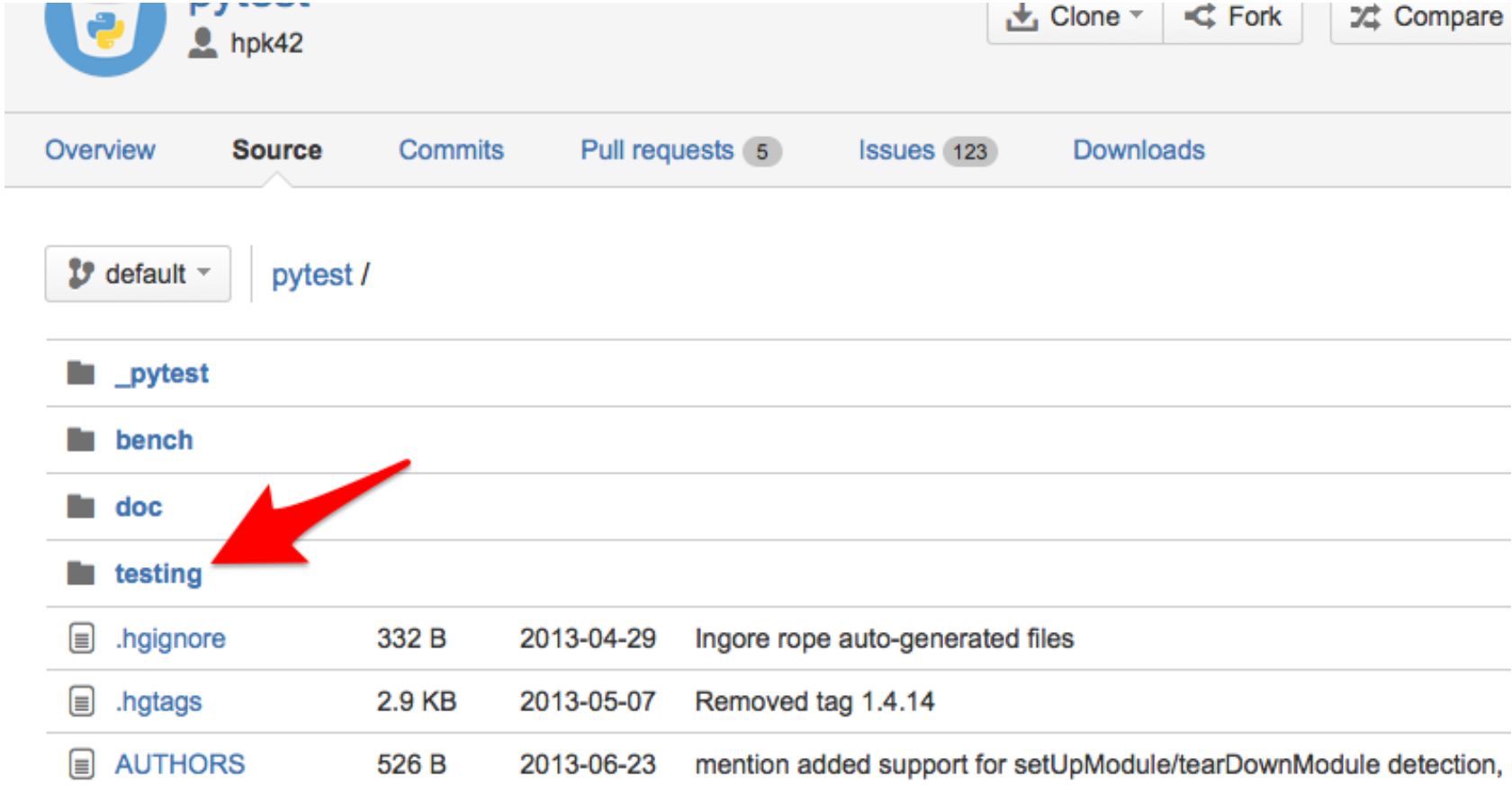
Debugging code that does lots of things is difficult.

The solution to both of these problems is to write code that doesn't do lots of things.

Write each function so that it does one thing and only one thing. This makes them easy to test with a unit test.”

http://programmers.stackexchange.com/a/195992/3497

Where Do Tests Go?



Where do tests go?

https://bitbucket.org/hpk42/pytest/src/

You Can Retrofit Tests

http://www.infoworld.com/d/open-source-software/what-you-can-learn-the-monster-libreoffice-project-212908

Wait, There's More

Further Reading

- http://obeythetestinggoat.com/
- http://pythontesting.net/

- http://wiki.python.org/moin/PythonTestingToolsTaxonomy

- http://pytest.org
- https://nose.readthedocs.org/en/latest
- https://pypi.python.org/pypi/zope.testrunner
- http://www.simplistix.co.uk/software/python/testfixtures
- https://pypi.python.org/pypi/testtools
- http://nedbatchelder.com/code/coverage/
- http://testrun.org/tox/latest/
- https://pypi.python.org/pypi/behave

Further Reading

http://www.halfcooked.com/presentations/