PhpUnit

Testing Php one unit at a time!

Who am I?

I am Chris Ryan.

Lead Developer for the DWS Recreation group.
Over 15 years of software development experience.
Experience with Web, Desktop, Mobile platforms using different languages.

Fixed some really bad code.
Written some really bad code!

Thrown fits after someone broke something I had already fixed three times.

What is PhpUnit

PhpUnit is a tool for writing and running unit test for Php.

Written by Sebastian Bergman a co-founder of thePHP.cc and a pioneer in the field of quality assurance in PHP projects.

PhpUnit is a toolset and framework to help you write unit test for your Php Code. Unit tests should allow you to make fewer mistakes.

PhpUnit's goals are Easy to learn to write, Easy to write, Easy to read, Easy to execute, Quick to execute, Isolated.

http://manual.phpunit.de/

Why PhpUnit

Why do unit testing?

Unit testing is one part of test driven development. Even if you are not doing test driven development unit testing still helps increase the overall quality of code and reduce the overall error count by testing for expected behaviour.

Why use PhpUnit?

For Php it is a common tool that is well documented and understood by the industry. It also has a number of extensions available for using with other testing tools.

Installing PhpUnit

There are a number of different methods to get PhpUnit. Many of these are documented in PhpUnit's documentation. Some of the available methods are:

Download

```
wget https://phar.phpunit.de/phpunit.phar
php phpunit.phar
```

Pear

```
pear config-set auto_discover 1
pear install pear.phpunit.de/PHPUnit
```

Composer

```
{
    "require-dev": {
        "phpunit/phpunit": "3.7.*"
    }
}
```

Running PhpUnit

Basic:

```
phpunit MyTest SomeTests.php
```

or

```
phpunit TestDir
```

Better:

```
phpunit --bootstrap=TestDir/bootstrap.php --strict TestDir
```

Code Coverage:

```
phpunit --coverage-html ./report TestDir
```

Simplify your command by putting common options in phpunit.xml file. This allows you to be more consistent with the options you run each time.

Basic Example

tests/DoMathTest.php:

```
<?php
final class DoMathTest extends PHPUnit_Framework_TestCase {
    public function testAdd() {
        $obj = new DoMath();
        $this->assertEquals(3, $obj->add(1, 2));
    }
}
```

Basic Example

```
[cryan@devubuntu phpunit]$ vendor/bin/phpunit tests
PHPUnit 3.7.28 by Sebastian Bergmann.

Configuration read from /home/cryan/projects/phpunit/phpunit.xml

.

Time: 28 ms, Memory: 3.00Mb

OK (1 test, 1 assertion)
[cryan@devubuntu phpunit]$
```

~/projects/phpunit : bash

Be Descriptive

```
final class DoMathTest extends PHPUnit_Framework_TestCase {
    public function testAddIntegers() {
        $obj = new DoMath();
        $this->assertEquals(3, $obj->add(1, 2));
    }

    public function testAddFloats() {
        $obj = new DoMath();
        $this->assertEquals(1.3, $obj->add(0.5, 0.8));
    }
}
```

Be Descriptive for --testdox

```
[cryan@devubuntu phpunit]$ phpunit --testdox tests
PHPUnit 3.7.28 by Sebastian Bergmann.

Configuration read from /home/cryan/projects/phpunit/phpunit.xml

DoMath
 [x] Add integers
 [x] Add floats

[cryan@devubuntu phpunit]$
```

} Easier to read!

~/projects/phpunit : bash

Say what you are testing

```
/**
 * Test the math class
 *
 * @covers DoMath
 */
final class DoMathTest extends PHPUnit_Framework_TestCase {
    /**
     * @covers DoMath::add
     */
    public function testAddIntegers() {
    $obj = new DoMath();
    $this->assertEquals(3, $obj->add(1, 2));
    }

    .
    .
    .
```

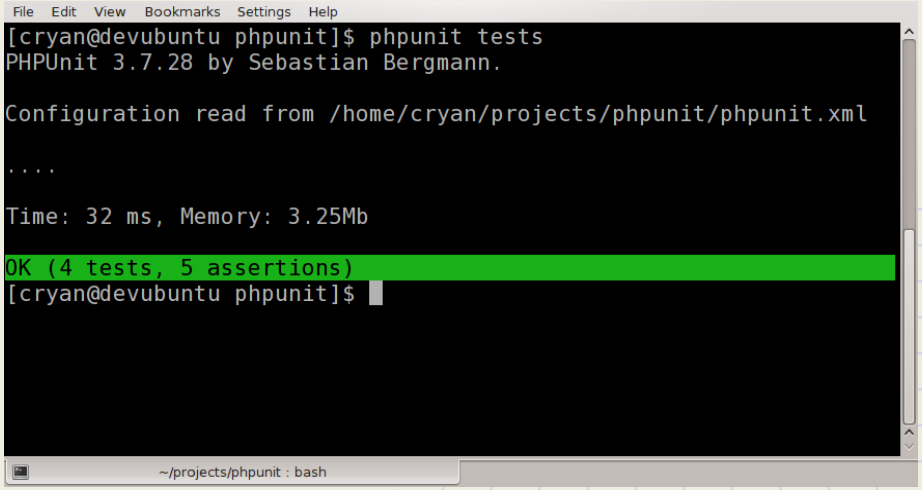
Use the most specific Assert

```
/**
 * @covers DoMath::add
 */
public function testAddHandlesNull() {
        $obj = new DoMath();
        // Really Poor assertion
        $this->assertEquals(false, is_null($obj->add(null, 1)));
        // Poor assertion
        $this->assertFalse(is_null($obj->add(null, 1)));
        // Good assertion
        $this->assertNotNull($obj->add(null, 1));
}
```

Actually use assertions

```
/**
 * @coversNothing
 */
public function testNoAssertion() {
    $obj = new DoMath();
    $obj->add(1, 2);
}
```

How do we know this really worked?



Use --strict

```
[cryan@devubuntu phpunit]$ phpunit --strict tests
PHPUnit 3.7.28 by Sebastian Bergmann.

Configuration read from /home/cryan/projects/phpunit/phpunit.xml

...I

Time: 38 ms, Memory: 3.25Mb

OK, but incomplete or skipped tests!
Tests: 4, Assertions: 5, Incomplete: 1.
[cryan@devubuntu phpunit]$
```

~/projects/phpunit : bash

Decouple test code & data

```
/**
 * @dataProvider provider
 * @covers DoMath::add
 */
public function testAddWithDataSet($a, $b, $c) {
    $obj = new DoMath();
    $this->assertEquals($c, $obj->add($a, $b));
}

public function provider() {
    return array(
        array(2, 3, 5),
        array(3, -2, 1),
        array(-2, -3, -5)
     );
}
```

Putting it into practice

Unit tests are great but you must:

- Write unit tests…. start with one
- Run the unit tests… automation helps
- Fix code that breaks unit tests… broken code/tests are problems
- Keep adding tests… test driven development

Things to avoid:

- Avoid changing the unit tests and the code at the same time
- Avoid testing dependencies
- Avoid writing tests that do not actually tests anything

Legacy code

- Legacy code can be harder to write for
- It is just as important to have unit tests for legacy code
- Start with one easy test
- Add more tests
- Do not change code and write the test at the same time

It is HARD but is worth the effort!

References

I heavily referenced the following materials to create this presentation.

http://thephp.cc/dates/2012/webexpoprague/phpunit-best-practices
http://phpunit.de/manual/current/en/index.html

All the code from this presentation is in Github.com

https://github.com/chrisryan/phpunit-example

Questions?

View this presentation at http://bit.ly/1eYCfCO