Getting Started with PHPUnit

Matt Frost
http://joind.in/13536
@shrtwhitebldguy

Who am I?

- Director of Engineering at Budget Dumpster
- Marathoner (4 marathoner)
- Dad of 2 awesome little people
- Open Source Project Creator/Contributor
- Loosely Coupled Podcast Co-Host
- Testing advocate
- Agile skeptic

What Can I Expect?

- Coverage of essential PHPUnit concepts
- Practical examples/code along sessions
- Just enough theory
- Assertions, Mocking, Setup, Tear Down, Test suite structure, Dependency Injection, Code Coverage; to name a few things.
- https://github.com/mfrost503/phpunit-tutorial

Finally...

- My aim is give you something you can take back to work with you,
- ...provide assistance working through real world examples,
- ...and provide you a reference resource to help you get started testing.

Let’s get going!

Take a few minutes to get your environment setup if you haven’t already. Raise your hand if you need some help getting started!

Why write unit tests?

Predictability

- Helps you KNOW that things work
- Helps teammates know things work
- Makes deployment automation easier
- Allows you to go home on time
- Useful as a debugging tool

Mostly though...

It's because you care and caring is good!

A little history

- Created by Sebastian Bergmann
- Version 1.0.0 was released April 8th, 2001
- PHP gets a bad wrap, QA/QC have helped make it a sustainable language.
- PHP runs almost 3/4 of all web apps
- People like to know their stuff is broken before someone else points it out to them.

Why PHPUnit?

- Most PHP projects use it
- It's actively maintained and adapted
- There's really not a better option for Unit Tests
- It does more than Unit Tests (there are better options)
- Most testing resources in PHP assume PHPUnit as the testing framework

Show me the goods!

ok...

Testable Code

- Dependency Injection
- Separation of Concerns
- Proper Use of Access Modifiers (public, private, protected, final)
- Building with composition and useful inheritance
- **All these things are going to make testing your code easier, more reliable and more fun!**

Dependency Injection

- Utilize existing functionality by passing it in!
- Eliminate the “new” keyword in method bodies
- Each component can be individually tested
- Creates flexibility for the different types of functionality (interfaces, Liskov, etc)

Example Time!

```php
use Snaggle\OAuth1\Client\Credentials\AccessCredentials;
use Snaggle\OAuth1\Client\Credentials\ConsumerCredentials;
use Snaggle\OAuth1\Client\Signatures\HmacSha1;
use Snaggle\OAuth1\Client\Signatures\Plaintext;
use Snaggle\OAuth1\Client\Header\Header;

// first we need to represent our tokens, these should be stored securely
$consumer = new ConsumerCredentials('CONSUMER_KEY', 'CONSUMER_SECRET');

$access = new AccessCredentials('ACCESS_TOKEN', 'ACCESS_SECRET');

$signature = new HmacSha1($consumer, $access)
    ->setResourceURL('https://api.example.com/v1/users')
    ->setHttpMethod('get');
```

Example from Snaggle

Assertions

- Define what we expect our code to do
- If they fail, the test fails
- Different types of assertions handle different scenarios
- There's more than true/false/equals!

Examples

assertTrue($condition,string $message);

```
$this->assertTrue(is_numeric(1));
```

assertFalse($condition, string $message);

```
$this->assertFalse(is_string(1234));
```

assertStringStartsWith($prefix, $string, $message);

```
$this->assertStringStartsWith(“pic”, “picture”);
```

assertSame($expected, $actual, $message);

```
$this->assertSame(‘1234’, ‘1234’);
```

assertObjectHasAttribute($attribute, $object, $message);

```
$this->assertObjectHasAttribute(‘name’, new User);
```

assertInstanceOf($expected, $actual, $message);

```
$this->assertInstanceOf(‘\PDO’, $pdo);
```

assertEquals($expected, $actual, $message);

```
$this->assertEquals(10, $sum);
```

Finally some code!

Take a look at the following files:

src/Math/Addition.php

tests/Math/AdditionTest.php

Write a test!

To start, we want to write a test that verifies that our code is adding the numbers in the array correctly.

Take a few minutes to do that, I'll be walking around if you have any questions

Exceptions

@expectedException annotation in the docblock

$this->setExpectedException($exception);

Data Providers

@dataProvider <methodName>

Here's an example!

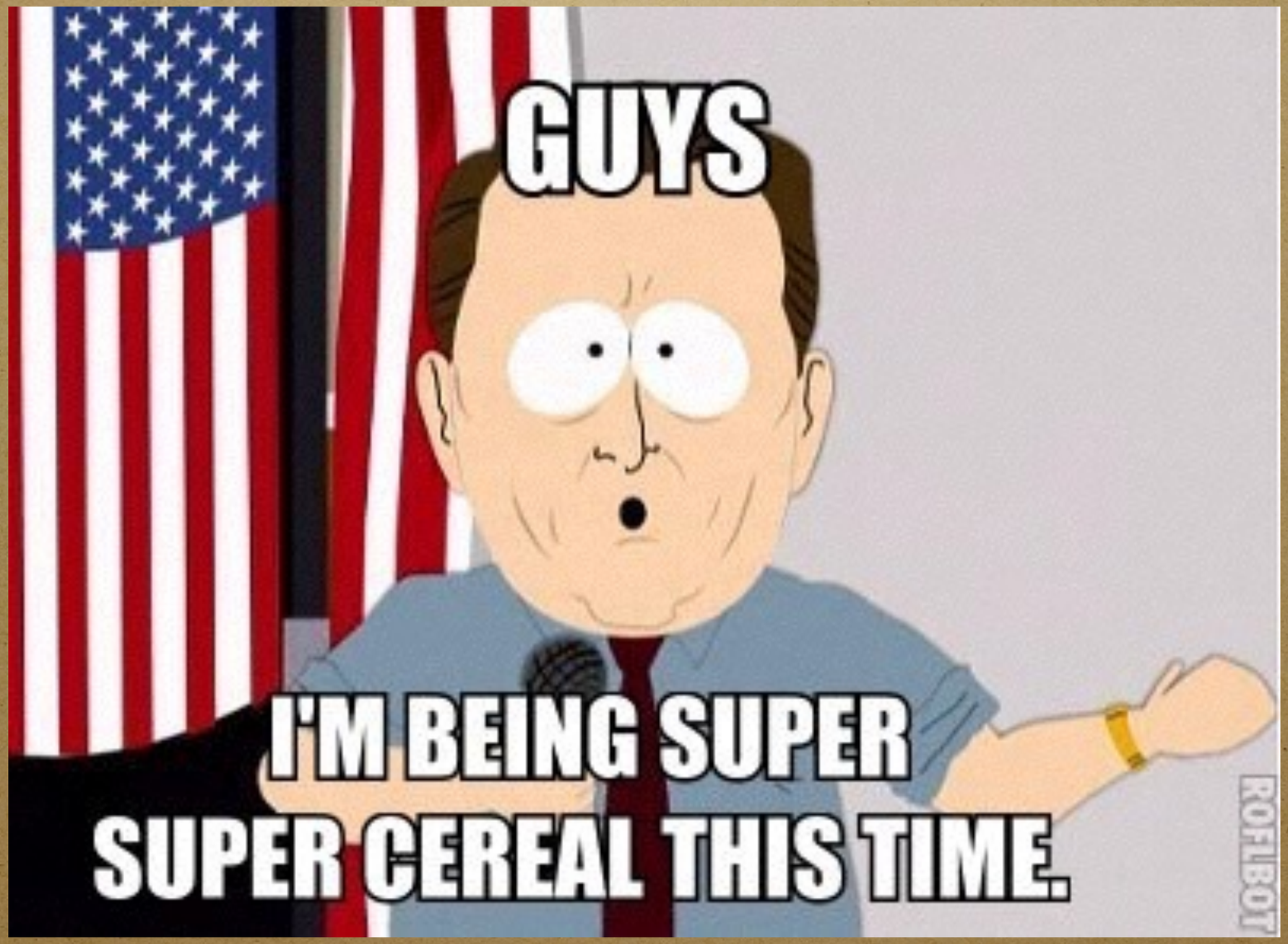
```
public function numberProvider()
{
    return [
        [1, 2, 3],
        [2, 3, 4],
        [4, 5, 6]
    ];
}


 /**
  * @dataProvider numberProvider
public function testNumbers($num1, $num2, $num3)
{
```

Add tests

Take a look at the code. Other than returning the correct answer, what are some other things that can happen?

Write some tests to verify the other behaviors

Serializers and Interfaces



Data Serialization

The transformation of data from a certain format to another format

Common types

- JSON
- PHP Array
- PHP Object
- serialize()
- json_encode()
- json_decode()

A Word on Interfaces

Serializers are a good example for interfaces, since the need to transform data to a from different formats is a common need.

Contracts

By adhering to a contract, the serializer types can be interchanged as needed and type hinted on the interface

Serializer Tests

If you look at src/Tutorial/Serializers - you will see a number of different types. Write some tests for them

Cue the Jeopardy Theme

src/Tutorial/Serializers

tests/Tutorial/Serializers

Communicating with data stores

- Don’t connect to them
- Don’t talk to them
- Don’t expect them to be available
- Don’t expect data to be there

In the context of your tests, dependencies are strangers and shouldn't trust them.

You see...

- Using actual data sources means your tests expect a certain state
- If source is down, tests fail...tells us nothing about our code
- These are unit tests, we want them to run quickly
- Ever seen an API with a 10 sec response time? I have, they exist...

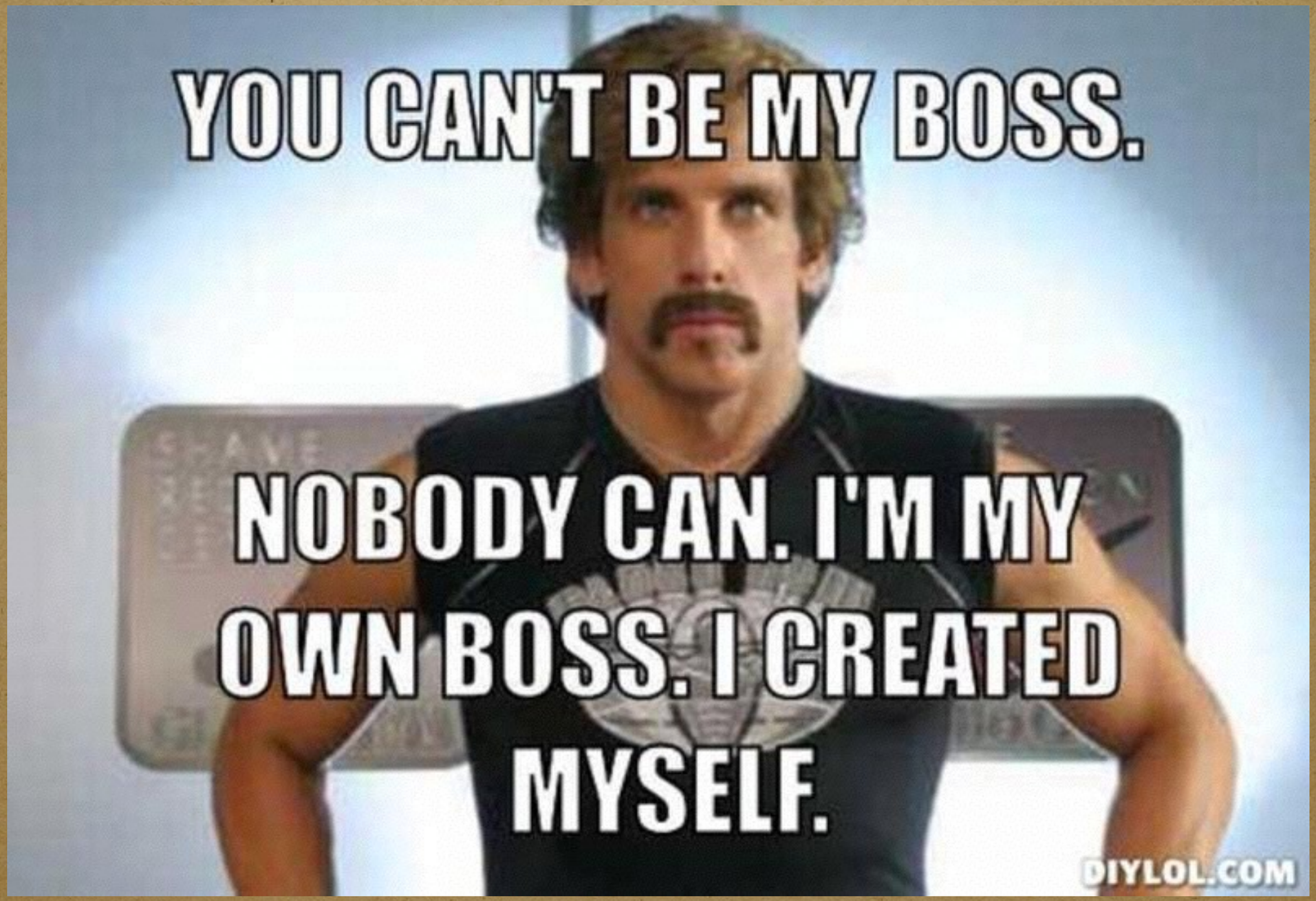
It’s not fanatical dogma

You will hate testing

and that's not what we want

Test Doubles

- The act of replicating dependencies
- We model them for scenarios
- We use them to avoid strangers
- We know them because we make them


YOU CAN'T BE MY BOSS.
NOBODY CAN. I'M MY OWN BOSS. I CREATED MYSELF.
DIYLOL.COM

What can we mock?

- API calls
- Database calls
- Internal functions*

* we can use runkit to change how internal functions like mail interact in the test

What about the data?

How?

- Data Fixtures
- Predefined responses
- Data providers
- Data fixtures are realistic data that is read into a test to simulate an actual response

It only works if it's realistic

- Model realistic responses
- Save an API/DB/Cache call to a file
- Read it in, have the mock return it
- It's that easy!

Sometimes...

There is no response and we just need to fill a parameter requirement or we just want to make sure a method in a dependency is called

Components

- Method - what method are we mocking
- Expectation - how often do we expect it to be called
- Parameters - what does it need to be called with?
- Return - Is it going to return data?

Example

```
$pdoMock = $this->getMock('PDO');
$statement = $this->getMock('PDOStatement');


$pdoMock->expects($this->once())
    ->method('prepare')
    ->with("SELECT * from table")
    ->will($this->returnValue($statement));
```

That's all there is too it

So when we run this code, we are mocking a prepared statement in PDO

We can mock PDO Statement to handle the retrieval of data

Let's give it a go!

Have a look at:
src/Tutorial/Mocking/PDO

Try to write some tests for these classes

Did you notice?

We could very easily add a second parameter and serialize the return data into our User value object. In which case, we could choose to mock or not mock the serializer.

API Mocking

- Guzzle
- Slack API
- Practice Mocking Guzzle
- GuzzleHttp\Client
- GuzzleHttp\Message\Response

Exceptions

There are some example tests, finish the tests - looking for exceptions!

Code Coverage Report

- Different output formats
- Analyze how completely you've tested
- Can be used to find testing gaps
- 100% shouldn't be your primary goal - write tests that make sense, not just to up coverage

Generating Reports

- vendor/bin/phpunit —coverage-html <location>
- Must have xdebug installed to generate reports
- Make sure you generate them to a directory in your gitignore.