Power PHP Testing

Chris Shiflett

chris@omniti.com

Geoffrey Young

geoff@modperlcookbook.org

I Don't Have Time To Test!

- Common misconception: testing takes too much time
- It giveth more than it taketh away
  - trim down your technical debt
- You don't have time *not* to test
- You are probably "testing" anyway
- Let's formalize what you're doing

Formalization

- Testing Theory in a Nutshell™
- Actual tests
- Along the way…
  - good practices
  - live demos
  - candy

Test Types

- There are lots of different kinds of tests
  - Integration
  - Function
  - Unit
  - Acceptance
  - Regression
- Called by different names, still same ideas

Integration Tests

- End-to-End testing
- Exercises the application as a whole
- Makes sure that all the parts work together
- Typically performed by QA
- "Does the application work?"

Function Tests

- "Does the developer interface work?"
- Exercise official API
- Standard data, observed bugs
- Most commonly found

Unit Tests

- Guts testing
- "Does the code work?"
- Exercise implementation
  - private subroutines
- Bugs, edge cases, branches, conditions
- "Twiddle my bits"

Acceptance Tests

- Requirements testing
- "Does it do what the client wants?"

Regression Tests

- Back compat tests
- "Does it unfix previous fixes?"

Test Types

- There are lots of different kinds of tests
  - Integration
  - Function
  - Unit
  - Acceptance
  - Regression
- All are important
- Doing one does not excuse you from doing the others

Function v Unit v Integration

- Unit tests
  - exercise function logic
  - that logic might be wrong, so
- Function tests
  - exercise the API
  - APIs are always part of a system
- Integration tests
  - exercises the entire system

PHP Testing

- Show some PHP code
- Try to test it using a few different frameworks
  - `phpt`
  - `Simple-Test`
  - `PHPUnit`
  - `Apache-Test`
- hint: `Apache-Test` rocks

functions.inc

```php
<?php

function create_user($username, $password) {
  ...
}

function delete_user($username) {
  ...
}

function hash_password($password) {
  ...
}

function glean_credentials() {
  ...
}

function authenticate_user($username, $password) {
  ...
}
?>
```

create_user()

```
function create_user($user, $pass)
{
  $clean = array();
  $sqlite = array();

  ... data validation ...

  $sqlite['user'] = sqlite_escape_string($clean['user']);
  $sqlite['pass'] = sqlite_escape_string($clean['pass']);

  $db = sqlite_open('/tmp/db.sqlite');

  $sql = "INSERT
          INTO   users
          VALUES ('{$sqlite['user']}', '{$sqlite['pass']}')";

  if (sqlite_query($db, $sql))
  {
      return TRUE;
  }

  return FALSE;
}
```

What to Test?

- This is actually the hardest part
- Hopefully we can help :)

Testing is a Skill

- Part of our Craft
- Nobody possess it at first
- Developed
  - if you have the dedication and patience
- Honed over time
- Lost if not exercised

Kata

- A prearranged series of movements
- Designed to teach new skills
- Instructs on many different levels

Kata: The Student

- Learn the motions
- Focus on the mechanics
- Understanding is not required

Kata: The Master

- The motions are fluid and second nature
- Understanding begins
  - individual movements
  - kata as a whole

Kata: The Artist

- Personal expression
- Application to new situations
- Continued learning

Kata: Power PHP Testing

- Common testing methodologies
- PHP testing frameworks

What to Test?

- This is actually the hardest part
- Hopefully we can help :)
- `create_user()` adds a user to *something*
- What aspects of that process do you care about?
- If you were following XP you would figure this out *before* you wrote the function

Unit Test Kata

- Data Validation
  - no null users or passwords
  - bad characters, etc
- Normal Condition
  - users can be added
- Edge Cases
  - duplicate users
  - sql injection, etc

create_user() Tests

```
<?php

require 'test-more.php';
require dirname(__FILE__) . '/../inc/functions.inc';

plan(9);

{
    # no user or password
    $return = create_user('', '');
    ok (!$return, 'no user/pass fails');
}

{
    # no user
    $return = create_user('', 'password');
    ok (!$return, 'password but no user fails');
}

{
    # no password
    $return = create_user('user', '');
    ok (!$return, 'user but no password fails');
}
```

Testing Basics

- All testing frameworks apply the same basic principles:
  - understand your input
  - compare expected output to actual output
- The differences are mostly in how that simple task is accomplished

```
$db_file = '/tmp/db.sqlite';

{
    $db = sqlite_open($db_file);
    ok ($db, 'created database successfully');

    $sql = "CREATE TABLE users
                (
                    username    varchar(50),
                    password    varchar(32),
                    PRIMARY KEY (username)
                )";

    $return = sqlite_query($db, $sql);
    ok ($return, 'added table successfully');
}

{
    # some generic user/password
    $return = create_user('user', 'password');
    ok ($return, 'generic user/pass successfully added');

    # cleanup
    delete_user('user');
}
```

Be Thou Self-Contained

- Failures are Bad™
- Inconsistent failures are Very Bad™
- To save you from inconsistent failures every test *must*
  - create its own environment
  - clean up after itself
- That way, every test can be run again and again and again and again and again and again and again and again...

```
{
    # test key uniqueness
    $return = create_user('user', 'password');
    ok ($return, 'unique user/pass successfully added');

    # sqlite throws duplicate user warnings - turn those off
    # but only here.  don't be sloppy :)
    $return = @create_user('user', 'password');
    ok (!$return, 'duplicate user/pass could not be added');

    # cleanup
    delete_user('user');
}

# database cleanup
# always leave your testing environment the way you
# found it so that the test is completely rerunnable

{
    $return = unlink($db_file);
    ok ($return, 'db.sqlite successfully removed');
}

?>
```

So Far...

- We have shown a few basic test scenarios
  - what to test
  - be self-contained
- We glossed over the framework-specific foo
- Let's do that now...

Apache-Test

```php
<?php

require 'test-more.php';
require dirname(__FILE__) . '/../inc/functions.inc';

plan(9);

{
    $return = create_user('', '');
    ok (!$return, 'no user/pass fails');
}

{
    $return = create_user('', 'password');
    ok (!$return, 'password but no user fails');
}

{
    $return = create_user('user', '');
    ok (!$return, 'user but no password fails');
}
```

Apache-Test

- Part of the mod_perl ASF project
- Provides full testing integration with Apache and Apache-based modules
  - like PHP
- Written in Perl
  - Geoff likes this
  - Chris, not so much
- `Apache-Test` rocks

test-more.php

- Automagically generated
- Interface into Apache-Test
- Provides simple, intuitive functions
  - ok()
  - is()
  - like()
- Takes care of bookkeeping
  - plan()
- Known to include_path

The `test-more` Paradigm

- Adopted from the time-tested Perl mythology (sic)
- `plan()` the number of tests
- call `ok()` for each test you plan
  - or `is()`, or `like()`, or `unlike()`, etc…

More on Apache-Test

- `Makefile` driven

  ```
  $ make test
  ```

- Fully integrated with Apache
  - configures `httpd`
  - starts `httpd`
  - stops `httpd`
  - tests can run in real `httpd` environment
- Other goodies
  - issues final report
  - verbose mode

phpt

- Uses the `pear` binary
  - in other words, included with PHP
- Dirt simple
  - says Chris

phpt

```
--TEST--
create_user() function
--FILE--
<?php

require dirname(__FILE__) . '/../inc/functions.inc';

{
    $return = create_user('', '');
    var_dump($return);
}

{
    $return = create_user('', 'password');
    var_dump($return);
}

{
    $return = create_user('user', '');
    var_dump($return);
}

?>
--EXPECT--
bool(false)
bool(false)
bool(false)
```

More on phpt

- As simple as it gets
- Lacks features
  - almost like not having a tool at all
- Comparing output in bulk will not scale
  - which of 237 tests failed?
  - and why?
- Cruft
  - we'll get to that later

Simple-Test

- Written by Marcus Baker
- Heavily Object Oriented
  - for tests? you *must* be kidding.
- Popular

Simple-Test

```php
<?php

require_once('../simpletest-1.0.0/unit_tester.php');
require_once('../simpletest-1.0.0/tap-reporter.php');
require dirname(__FILE__) . '/../inc/functions.inc';

class CreateUserTest extends UnitTestCase
{
    public function testBlankCredentials()
    {
        $return = create_user('', '');
        $this->assertFalse($return);
    }

    public function testBlankUser()
    {
        $return = create_user('', 'password');
        $this->assertFalse($return);
    }

    public function testBlankPassword()
    {
        $return = create_user('user', '');
        $this->assertFalse($return);
    }
}

$test = &new CreateUserTest();
$test->run(new TapReporter());

?>
```

unit_tester.php

- `Simple-Test`'s main library
- Holds comparison functions
- Names are not exactly intuitive

unit_tester.php

test-more.php	Simple-Test
• ok()	• assertTrue()
• is()	• assertEqual()
• isnt()	• assertNotEqual()
• like()	• assertWantedPattern()
• unlike()	• assertNoUnwantedPattern()

And Don't Forget...

- You *must* call these from within a method in a class in your test file
  - with `Simple-Test`, that is

More on Simple-Test

- HTML-based report
- Objects smobjects
  - but if you insist, it has mock objects
- Other tools
  - like the ones you get with Perl
- Popular

PHPUnit

```
<?php

require_once 'PHPUnit2/Framework/TestCase.php';
require dirname(__FILE__) . '/../inc/functions.inc';

class CreateUserTest extends PHPUnit2_Framework_TestCase
{
    public function testBlankCredentials()
    {
        $return = create_user('', '');
        $this->assertEquals(FALSE, $return);
    }

    public function testBlankUser()
    {
        $return = create_user('', 'password');
        $this->assertEquals(FALSE, $return);
    }

    public function testBlankPassword()
    {
        $return = create_user('user', '');
        $this->assertEquals(FALSE, $return);
    }
}

?>
```

TestCase.php

- PHPUnit's main library
- Not quite as bad as Simple-Test
- Still pretty bad

TestCase.php

test-more.php	PHPUnit
• ok()	• assertTrue()
• is()	• assertEquals()
• isnt()	• assertNotEquals()
• like()	• assertRegExp()
• unlike()	• assertNotRegExp()

And Again…

- You *must* call these from within a method in a class in your test file

More on PHPUnit

- Truckload of dependencies
  - Truckload wasn't the word Chris used
  - More on that later
- Popular
  - Zend framework

Running the Tests

- Thus far, we've covered what *you* write
- Tests are where you *should* spend most of your time
- Getting ready to run the tests comes in varying levels of difficulty
  - *should* be a one time cost
  - boy, can it be expensive...

`make` rules

- Before you were born, there was `make`
- We created a `Makefile` so

```
$ make test
```

ran the tests for each framework

- Here's what we did...

Makefile for phpt

```
test:
	pear run-tests t/*.phpt
```

When Tests Fail

- Ordinarily you should have no ongoing test failures
- "oh, that test always fails"
  - BAD, BAD, BAD!
  - decreases the integrity of your suite
- But when failures happen, they should be easy to debug

Hopefully, you saw...

- `make test` output looks no different on failure
- Instead `phpt` pukes all over the filesystem
- We found this incredibly annoying

```
$ make assertNoUnwantedPuke
$ make clean
```

Makefile for PHPUnit

- This was an iterative process
- First, we tried

```
$ phpunit t/*.php
Warning: require(PHPUnit2/...):
  failed to open stream: No such
  file or directory
```

Makefile for PHPUnit

- Then, we altered `include_path`:

```
$path = dirname(__FILE__);
$path = realpath($path);
ini_set('include_path', "$path/PEAR");
```

```
$ ./phpunit t/*.php
Warning: require(PEAR/...):
 failed to open stream: No such
 file or directory
```

Makefile for PHPUnit

- Then, we altered `include_path` again:

```
$path = dirname(__FILE__);
$path = realpath($path);
ini_set('include_path', "$path:$path/PEAR");
```

```
$ ./phpunit t/*.php
Warning: require(CreateUserTest.php):
 failed to open stream: No such file
 or directory
```

Makefile for PHPUnit

- We altered `include_path` yet again:

```
$path = dirname(__FILE__);
$path = realpath($path);
ini_set('include_path',
  "$path:$path/PEAR:$path/PEAR/PHPUnit2");
```

```
$ ./phpunit t/*.php
Warning: require(../Something):
  failed to open stream: No such file
  or directory
```

Makefile for PHPUnit

- We altered `include_path` yet again:

```
$path = dirname(__FILE__);
$path = realpath($path);
ini_set('include_path',
  "$path:$path/PEAR:$path/PEAR/PHPUnit2:.");
```

```
$ ./phpunit *.php
  Class AuthenticateUserTest could not
  be found in CreateUserTest.php.
```

Makefile for PHPUnit

- Hey, let's try the expansion ourselves

```
$ ./phpunit AuthenticateUserTest.php
  CreateUserTest.php


Class AuthenticateUserTest could not be
  found in CreateUserTest.php.
```

Makefile for PHPUnit

- hmph

```
$ ./phpunit AuthenticateUserTest.php
$ ./phpunit CreateUserTest.php
$ ./phpunit DeleteUserTest.php
$ ./phpunit HashPasswordTest.php
```

- This doesn't scale, so...

Makefile for PHPUnit

```
test:
    cd t && for i in *Test.php; do ./phpunit $$i; done
```

- You're Welcome :)

Makefile for Simple-Test

```
test:
	cd t && for i in *Test.php; do php $$i; done
```

- pretty much the same as PHPUnit
  - without the pain

Apache-Test Makefile.PL

- `Apache-Test` is written in Perl
- It follows standard Perl module foo

```
$ perl Makefile.PL
$ make
$ make test
```

- Don't be scared
  - besides, I know you've all done it before

Hopefully, you saw...

- `make test`
- `t/TEST -v`
- `t/TEST t/create_user.php`
- `t/TEST -start`
- browser

glean_credentials()

```
function glean_credentials()
{
    $credentials = array();
    $credentials[] = '';
    $credentials[] = '';

    if (isset($_GET['username']) &&
        isset($_GET['password']))
    {
        $credentials[] = $_GET['username'];
        $credentials[] = $_GET['password'];
    }

    return $credentials;
}
```

Options?

- With all of these frameworks you can stick the test file under `/htdocs` and bang on it with a browser
- That sucks
- Or, you can stick the test file under `/htdocs` and bang on it with a custom client that aggregates results
- That also sucks

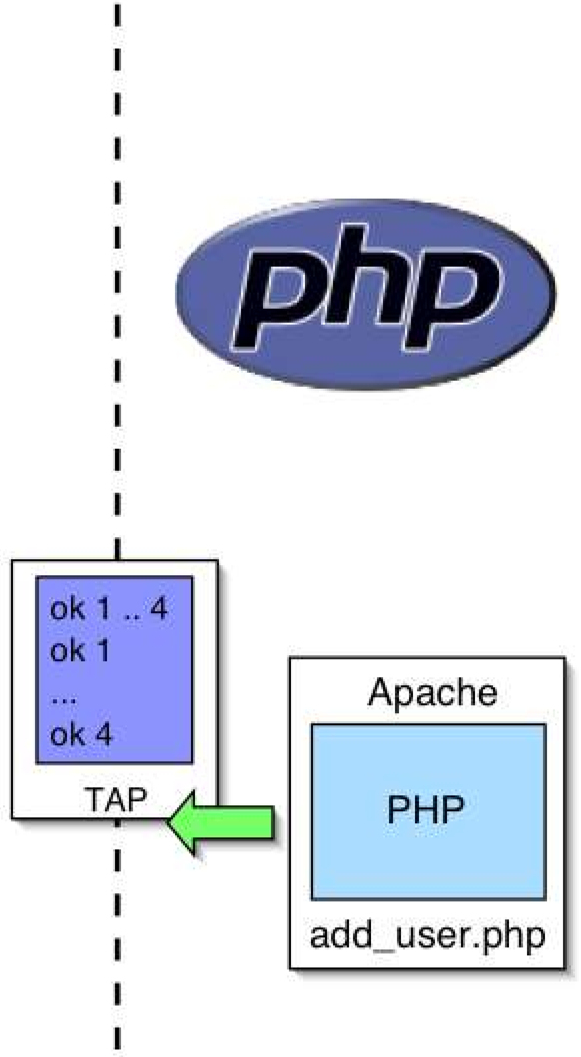
Behold the Power of Perl

- `Apache-Test` rocks
- Let `Apache-Test` do the heavy lifting
- It will
  - configure `httpd`
  - start the server
  - run the tests
  - stop the server
  - issue a report

Apache Foo

- Apache needs a basic configuration to service requests
  - `ServerRoot       t/`
  - `DocumentRoot     t/htdocs`
  - `ErrorLog         t/logs/error_log`
  - `Listen           8529`
  - `LoadModule`
- `Apache-Test` "intuits" these and creates its own `httpd.conf`
- Configures all that is required to `GET`

```
http://localhost:8529/index.html
```


ok 1 .. 4
ok 1
...
ok 4
TAP
Apache
PHP
add_user.php

A Brief Digression…

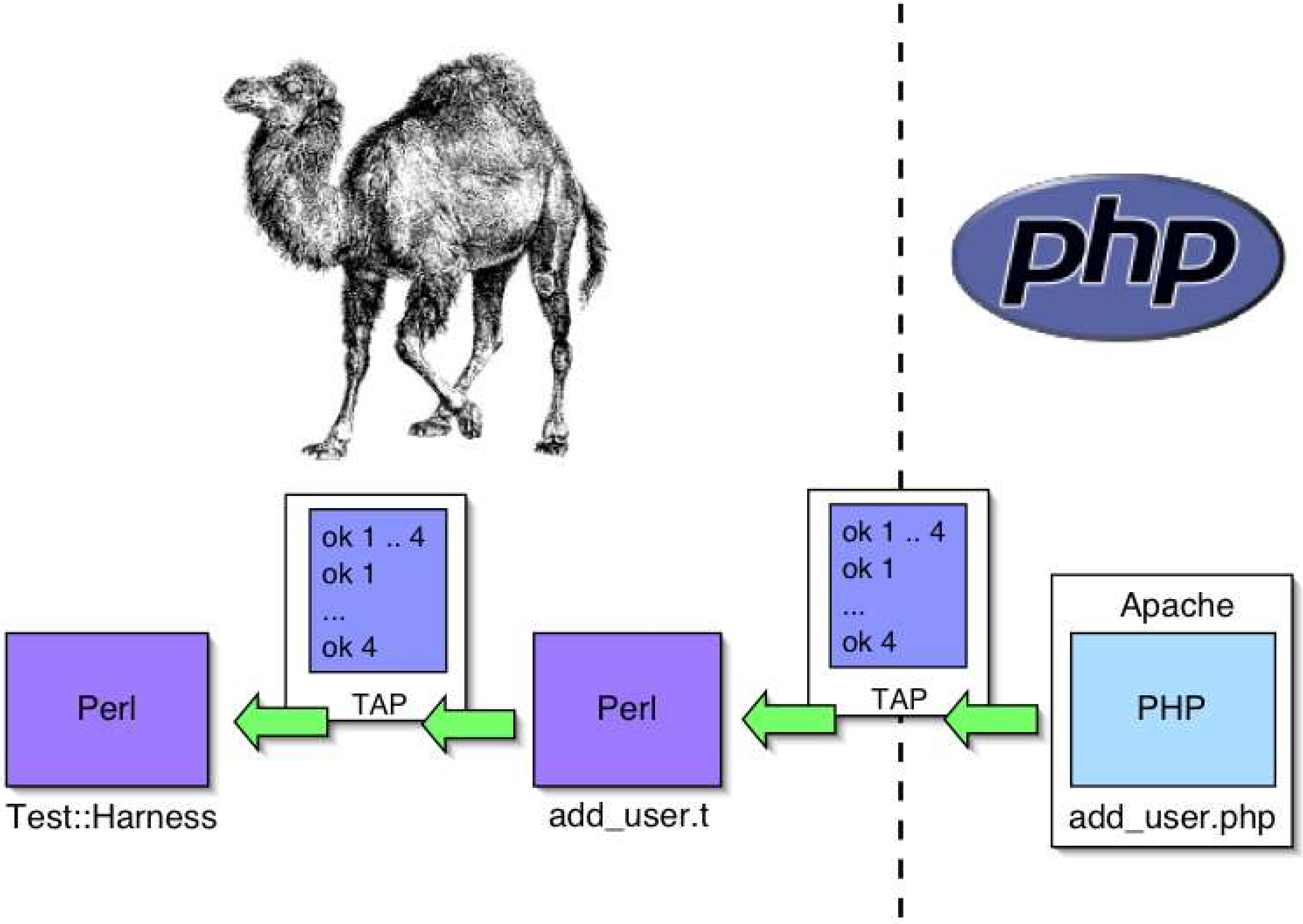
- TAP – the <u>T</u>est <u>A</u>nything <u>P</u>rotocol
  - aka

```
1..2
ok 1
# this is a comment
not ok 2
```

- Documented in `Test::Harness::TAP`
- Why the name?

Marketing++

- Once TAP was properly branded things started happening
- There are now TAP implementations in
  - PHP (`test-more.php`)
  - C (`libtap`)
  - JavaScript (`TestSimple.js`)
- Once you can generate TAP all you need to do is feed it to `Test::Harness`


Perl
Test::Harness
ok 1 .. 4
ok 1
...
ok 4
TAP
Perl
add_user.t
ok 1 .. 4
ok 1
...
ok 4
TAP
Apache
PHP
add_user.php

Writing the Client

- Magical things happen if you follow a specific filesystem pattern
- In our case

  `t/response/TestFoo/glean_creds.php`

automagically generates

  `t/foo/glean_creds.t`

- This is a Perl client
- Simply requests the test file
  - no special foo

glean_credentials.t

```
# WARNING: this file is generated, do not edit
# generated on Sat Dec 10 23:57:36 2005
# 01: /Apache/TestConfig.pm:942
# 02: /Apache/TestConfig.pm:960
# 03: /Apache/TestConfigPerl.pm:136
# 04: /Apache/TestConfigPerl.pm:569
# 05: /Apache/TestConfig.pm:624
# 06: /Apache/TestConfig.pm:639
# 07: /Apache/TestConfig.pm:1593
# 08: /Apache/TestRun.pm:507
# 09: /Apache/TestRunPerl.pm:90
# 10: /Apache/TestRun.pm:726
# 11: /Apache/TestRun.pm:726
# 12: t/TEST:28

use Apache::TestRequest 'GET_BODY_ASSERT';
print GET_BODY_ASSERT "/TestFunctions/glean_credentials.php";
```

Writing the Client

- You can write your own client
- In Perl or PHP

glean_credentials.php

```php
<?php

$path = dirname(__FILE__) . '/../..';
$path = realpath($path);
ini_set('include_path', ".:$path:$path/PEAR");

require 'HTTP/Request.php';

$host = 'http://127.0.0.1:8529';
$path = '/TestFunctions/glean_credentials.php';

$req = new HTTP_Request("$host$path");
$req->setMethod(HTTP_REQUEST_METHOD_POST);
$req->addPostData('username', 'testuser');
$req->addPostData('password', 'testpass');

if (!PEAR::isError($req->sendRequest()))
{
    echo $req->getResponseBody();
}

?>
```

Brought To You By...

http://shiflett.org/

http://modperlcookbook.org/~geoff/