Testing PHP with Perl

Chris Shiflett
shiflett@php.net

Geoffrey Young
geoff@modperlcookbook.org
Why Perl?

- Testing has become very fashionable within the Perl community
- Perl testing tools are mature
- Some of these tools were designed for Apache
- PHP has strong Apache roots
- Ergo, Perl can help test PHP
  - unless you're using IIS, in which case you have bigger problems than testing
The Perl testing community has put lots of work into our tools to:
  - make automating tests easy
  - make writing tests intuitive

The Perl-centric Apache community has brought the goodness to Apache.

There is no reason why PHP can't take advantage of both.
Building Apache + PHP

$ tar -xvzf apache_1.3.31.tar.gz
$ tar -xvzf php-5.0.2.tar.gz
$ cd apache_1.3.31
$ ./configure
$ cd ../php-5.0.2
$ ./configure --prefix=/usr/local/php "
    --with-apache=../apache_1.3.31 --with-pear "
    --with-gd --with-mysql=/usr/local/mysql "
    --enable-sockets --with-zlib-dir=/usr/include
$ make
$ sudo make install

$ cd ../apache_1.3.31
$ ./configure --prefix=/usr/local/apache "
    --activate-module=src/modules/php5/libphp5.a "
    --enable-module=most --enable-shared=max
$ make
$ sudo make install
Building Apache + PHP

```
$ tar -xvzf apache_1.3.31.tar.gz
$ tar -xvzf php-5.0.2.tar.gz
$ cd apache_1.3.31
$ ./configure
$ cd ../php-5.0.2
$ ./configure --prefix=/usr/local/php --with-apache=../apache_1.3.31 --with-pear --with-gd --with-mysql=/usr/local/mysql --enable-sockets --with-zlib-dir=/usr/include
$ make
$ sudo make install

$ cd ../apache_1.3.31
$ ./configure --prefix=/usr/local/apache --activate-module=src/modules/php5/libphp5.a --enable-module=most --enable-shared=max
$ make
$ sudo make install
```
Getting Apache-Test

$ cvs -d :pserver:anoncvs@cvs.apache.org:/home/cvspublic
   login
$ cvs -d :pserver:anoncvs@cvs.apache.org:/home/cvspublic co httpd-test
$ cd httpd-test/perl-framework/Apache-Test

$ perl Makefile.PL
$ make
$ sudo make install
Test Automation

• Perl distributions typically start with a Makefile.PL
  
  \$ perl Makefile.PL

• This generates a Makefile with a bunch of useful make targets
  
  \$ make
  
  \$ sudo make install
  
  \$ make test
make test

• The test target is the basis for Perl testing
• make test will
  – search for *.t files under t/
  – execute them
  – collect results
  – write out a final report
So What?

• "We don't care about Perl. How does this help us?" you ask...

• Enter Apache-Test
Apache-Test

• Framework for testing Apache-based application components
• Gives you a self-contained, pristine Apache environment
• Provides HTTP-centric testing tools for client-side tests
• Provides PHP-centric testing tools for server-side tests
  – if you use the libraries from current CVS that I added for this talk
The test Target

• With Apache-Test, make test will
  – configure Apache
  – start Apache
  – execute the tests
  – issue the report
  – stop Apache

• All you need to do is write the tests
  – and get Apache-Test working
Integration Mechanics

1. Generate the test harness
2. Configure Apache
Step 1 - The Test Harness

- Generally starts from `Makefile.PL`
- There are other ways as well
use Apache::TestMM qw(test clean);
use Apache::TestRunPHP ();

# configure tests based on incoming arguments
Apache::TestMM::filter_args();

# generate the test harness (t/TEST)
Apache::TestRunPHP->generate_script();
Step 1 - The Test Harness

• Don't get bogged down with Makefile.PL details
Step 1 - The Test Harness

- Don't get bogged down with Makefile.PL details
- Lather, Rinse, Repeat
Integration Mechanics

1. Generate the test harness
2. Configure Apache
Step 2 - Configure Apache

• Apache needs a basic configuration to service requests
  – ServerRoot
  – DocumentRoot
  – ErrorLog
  – Listen

• Apache-Test "intuits" these

• But uses the exact same httpd binary
Apache-Test Intuition

- Apache-Test provides server defaults
  - ServerRoot t/
  - DocumentRoot t/htdocs
  - ErrorLog t/logs/error_log
  - Listen 8529

- Also provides an initial index.html
  http://localhost:8529/index.html

- You will need some PHP stuff
Adding to the Default Config

• Supplement default `httpd.conf` with custom configurations

• Create `t/conf/extra.conf.in`
extra.conf.in

• Same directives as `httpd.conf`
• Pulled into `httpd.conf` via `Include`
• Allow for some fancy variable substitutions
Create the Configuration

- We will be doing PHP specific stuff
- Let's add some standard PHP configuration directives
AddType application/x-httpd-php .php
DirectoryIndex index.php index.html

<IfModule @PHP_MODULE@>
    php_flag display_errors Off
    php_flag log_errors On
    php_value error_log @ServerRoot@/logs/php_errors
</IfModule>

<Files ~ "\.\.(inc|sqlite)" >
    Order allow,deny
    Deny from all
</Files>
Integration Mechanics

1. Generate the test harness
2. Configure Apache
3. Write the tests
4. Install the application into our tree
The \texttt{t/} Directory

- \texttt{t/} is the \texttt{ServerRoot}
  - \texttt{t/htdocs}
  - \texttt{t/cgi-bin}
  - \texttt{t/logs}

- Tests live in \texttt{t/}
Admin Application

- t/htdocs/admin/index.php
- t/htdocs/admin/add.php
- t/htdocs/admin/delete.php
Let's Test This Puppy

• The old way of testing an application was to fire up a browser

• Browser-based testing is so pre-bubble

• Apache-Test gives you a server just waiting to receive requests

• Perl provides lots of tools to automate the client-side

• Apache-Test provides magic for automated server-side PHP testing
Anatomy of a Test

• In the Perl testing world everyone does testing essentially the same way
  • create `t/foo.t`
  • `plan()` the number of tests
  • `call ok()` for each test you plan
    – where `ok()` is any one of a number of different functions
• All the rest is up to you
Perl versus PHP

• Apache-Test is a Perl tool
  – uses Perl to call test scripts in t/  
  – t/ scripts act as a browser

• PHP support is a bit different
  – still uses Perl scripts as a browser  
  – additional clients are autogenerated to call PHP server-side tests
Client versus Server

• Let's start with some client-side examples
• Show the cool server-side PHP stuff you really care about soon
• It's important to see the difference
use Apache::TestRequest;

use Test::More;

plan tests => 3;

my $uri = '/admin/';

{
    my $response = GET $uri;

    is ($response->code, 401, "no valid password entry");
}
Apache::TestRequest

• Provides a basic HTTP interface like PEAR::HTTP_Client
  – GET()
  – POST()
  – HEAD()
  – etc...

• Functions are self-aware
  – know which server and port to talk to
Test::More

• Interface into the Perl testing harness
• Provides simple functions so you don't need to print
  1..2
  \n  1 ok
  2 ok

  – ok()
  – is()
  – like()

• Takes care of bookkeeping
  – plan()
ok() 

• Used for simple comparisons
  
  ok($foo == $bar, '$foo equals $bar')

• Gives little diagnostic output on failure

  not ok 1 - $foo equal to $bar
  #    Failed test (test.pl at line 8)
is()

• Almost the same as ok()
  is($foo, $bar, '$foo equals $bar')

• Gives better diagnostic output on failure
  not ok 1 - $foo is $bar
  # Failed test (test.pl at line 8)
  # got: '1'
  # expected: '2'
like()

- Regular expression matching

  like($foo, qr/foo/, '$foo matches /foo/)

  not ok 1 - $foo matches /foo/
  #   Failed test (test.pl at line 7)
  #   'bar'
  #   doesn't match '(?-xism:foo)'
my $response = GET $uri, username => 'geoff',
    password => 'foo';

is ($response->code,
    401,
    "password mismatch");

my $response = GET $uri, username => 'admin',
    password => 'adminpass';

is ($response->code,
    200,
    "admin allowed to proceed");
```perl
{ my $response = GET $uri, username => 'geoff',
   password => 'foo';

   is ($response->code,
       401,
       "password mismatch");
}

{ my $response = GET $uri, username => 'admin',
   password => 'adminpass';

   is ($response->code,
       200,
       "admin allowed to proceed");
}
```
Drumroll...

• And now, what you really came here to see...
test_more.inc

• Apache-Test provides test_more.inc
• test_more.inc is PHP's Test:::More
  – ok()
  – is()
  – like()
  – plan()
  – etc
• include_path is adjusted
  <?php require 'test_more.inc'; ?>
PHP Server-Side Tests

• You can use `test_more.inc` functions to communicate with the Perl test harness

• How?
PHP Mechanics

• Create PHP scripts as
  
  t/response/TestFoo/bar.php

• Apache-Test will automagically create a client-side Perl script that calls bar.php
  
  t/foo/bar.t

• make test will
  
  – run bar.t
  
  – which will request bar.php
  
  – which will send data to the test harness
<?php
include '../functions.inc';

if (!check_admin($user, $password)) {
    echo '<p>Access Denied</p>';
    exit;
}
?>
check_admin()

function check_admin($user, $pass)
{
    if ($user == 'admin' && $pass == 'adminpass')
    {
        return true;
    }

    header('HTTP/1.0 401 Unauthorized');
    header('WWW-Authenticate: Basic realm="foo"');

    return false;
}
<?php
require 'test_more.inc';
require "${_SERVER['DOCUMENT_ROOT']}/functions.inc";

plan(2);

{
    $rc = check_admin('user', 'password');
    ok (!$rc, 'non-admin user/pass fails');
}

{
    $rc = check_admin('admin', 'adminpass');
    ok ($rc, 'admin user/pass found');
}
?>
require 'test_more.inc';
require "${_SERVER['DOCUMENT_ROOT']}/functions.inc";

plan(3);
{
    $password = 'funkyfunky';

    $newpass = encrypt_password($password);

    # the returned password should be different
    isnt ($newpass,
           $password,
           'password is at least different');
# and that it has basic md5 characteristics,
# such as being 32 characters long
is (strlen($newpass),
    32,
    'password is a proper 32 characters');

# and all 32 characters must be within hex range
like ($newpass,
    '/^[0-9a-fA-F]{32}$/','
    'password consists of only hex characters');
} ?>
# and that it has basic md5 characteristics,
# such as being 32 characters long
is (strlen($newpass),
    32,
    'password is a proper 32 characters');

# and all 32 characters must be within hex range
like ($newpass,
    '/^[0-9a-fA-F]{32}$/',
    'password consists of only hex characters');
}?>
Advantages

• PHP code tested in real environment
• Self-contained environment
• Simple tools to lower the testing barrier
• No tests in your application
Where is Apache-Test?

• mod_perl 2.0
• CPAN
• `httpd-test` project
  - [http://httpd.apache.org/test/](http://httpd.apache.org/test/)
  - test-dev@httpd.apache.org
More Information

- perl.com
- http://www.perl.com/pub/a/2003/05/22/testing.html

- Apache-Test tutorial
  - http://perl.apache.org/docs/general/testing/testing.html

- Apache-Test manpages
  - $ man Apache::TestRunPHP

- mod_perl Developer's Cookbook
  - http://www.modperlcookbook.org/
Slides

- These slides freely available at some long URL you will never remember...

http://www.modperlcookbook.org/~geoff/slides/nyphp