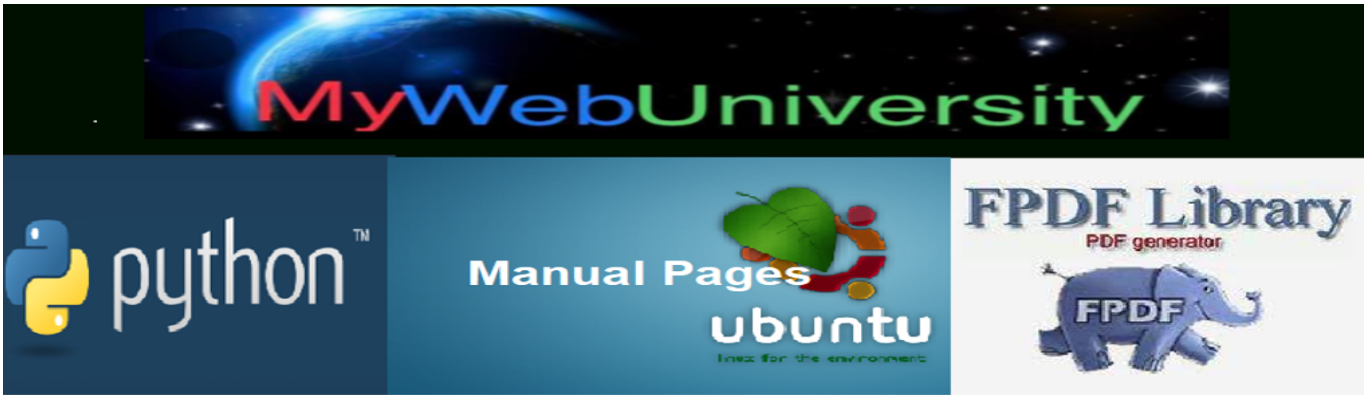

Full credit is given to the above companies including the Operating System (OS) that this PDF file was generated!

Rocky Enterprise Linux 9.2 Manual Pages on command 'Apache::TestRunPHP.3pm'

$ man Apache::TestRunPHP.3pm

Apache::TestRunPHP(3pm) User Contributed Perl Documentation Apache::TestRunPHP(3pm)

NAME

Apache::TestRunPHP - configure and run a PHP-based test suite

SYNOPSIS

```
use Apache::TestRunPHP;
Apache::TestRunPHP->new->run(@ARGV);
```

DESCRIPTION

The "Apache::TestRunPHP" package controls the configuration and running of the test suite for PHP-based tests. It's a subclass of "Apache::TestRun" and similar in function to "Apache::TestRunPerl".

Refer to the "Apache::TestRun" manpage for information on the available API.

EXAMPLE

"TestRunPHP" works almost identially to "TestRunPerl", but in case you are new to "Apache-Test" here is a quick getting started guide. be sure to see the links at the end of this document for places to find additional details.

because "Apache-Test" is a Perl-based testing framework we start from a "Makefile.PL", which should have the following lines (in addition to the standard "Makefile.PL" parts):

```
use Apache::TestMM qw(test clean);
use Apache::TestRunPHP ();
Apache::TestMM::filter_args();
Apache::TestRunPHP->generate_script();
```

"generate_script()" will create a script named "t/TEST", the gateway to the Perl testing harness and what is invoked when you call "make test". "filter_args()" accepts some

"Apache::Test"-specific arguments and passes them along. for example, to point to a specific "httpd" installation you would invoke "Makefile.PL" as follows

```
  $ perl Makefile.PL -httpd /my/local/apache/bin/httpd
```

and "/my/local/apache/bin/httpd" will be propagated throughout the rest of the process.

note that PHP needs to be active within Apache prior to configuring the test framework as shown above, either by virtue of PHP being compiled into the "httpd" binary statically or through an active "LoadModule" statement within the configuration located in "/my/local/apache/conf/httpd.conf". Other required modules are the (very common) mod_alias and mod_env.

now, like with "Apache::TestRun" and "Apache::TestRunPerl", you can place client-side Perl test scripts under "t/", such as "t/01basic.t", and "Apache-Test" will run these scripts when you call "make test". however, what makes "Apache::TestRunPHP" unique is some added magic specifically tailored to a PHP environment. here are the mechanics.

"Apache::TestRunPHP" will look for PHP test scripts in that match the following pattern

```
  t/response/TestFoo/bar.php
```

where "Foo" and "bar" can be anything you like, and "t/response/Test*" is case sensitive.

when this format is adhered to, "Apache::TestRunPHP" will create an associated Perl test script called "t/foo/bar.t", which will be executed when you call "make test". all "bar.t" does is issue a simple GET to "bar.php", leaving the actual testing to "bar.php".

in essence, you can forget that "bar.t" even exists.

what does "bar.php" look like? here is an example:

```
  <?php
    print "1..1\n";
    print "ok 1\n"
  ?>
```

if it looks odd, that's ok because it is. I could explain to you exactly what this means, but it isn't important to understand the gory details. instead, it is sufficient to understand that when "Apache::Test" calls "bar.php" it feeds the results directly to "Test::Harness", a module that comes with every Perl installation, and "Test::Harness" expects what it receives to be formatted in a very specific way. by itself, all of this is pretty useless, so "Apache::Test" provides PHP testers with something much better.

here is a much better example:

```
  <?php
```

```
    # import the Test::More emulation layer
    # see
    #   http://search.cpan.org/dist/Test-Simple/lib/Test/More.pm
    # for Perl's documentation - these functions should behave
    # in the same way
    require 'test-more.php';
    # plan() the number of tests
    plan(6);
    # call ok() for each test you plan
    ok ('foo' == 'foo', 'foo is equal to foo');
    ok ('foo' != 'foo', 'foo is not equal to foo');
    # ok() can be other things as well
    is ('bar', 'bar', 'bar is bar');
    is ('baz', 'bar', 'baz is baz');
    isnt ('bar', 'beer', 'bar is not beer');
    like ('bar', '/ar$/', 'bar matches ar$');
    diag("printing some debugging information");
    # whoops! one too many tests.  I wonder what will happen...
    is ('biff', 'biff', 'baz is a baz');
  ?>
```

the include library "test-more.php" is automatically generated by "Apache::TestConfigPHP" and configurations tweaked in such a a way that your PHP scripts can find it without issue. the functions provided by "test-more.php" are equivalent in name and function to those in "Test::More", a standard Perl testing library, so you can see that manpage for details on the syntax and functionality of each.

at this point, we have enough in place to run some tests from PHP-land - a "Makefile.PL" to configure Apache for us, and a PHP script in "t/response/TestFoo/bar.php" to send some results out to the testing engine. issuing "make test" would start Apache, issue the request to "bar.php", generate a report, and shut down Apache. the report would look like something like this after running the tests in verbose mode (eg "make test TEST_VERBOSE=1"):

```
  t/php/bar....1..6
  ok 1 - foo is equal to foo
```

```
not ok 2 - foo is not equal to foo
#     Failed test (/src/devel/perl-php-test/t/response/TestFoo/bar.php at line 13)
ok 3 - bar is bar
not ok 4 - baz is baz
#     Failed test (/src/devel/perl-php-test/t/response/TestFoo/bar.php at line 17)
#          got: 'baz'
#     expected: 'bar'
ok 5 - bar is not beer
ok 6 - bar matches ar$
# printing some debugging information
ok 7 - baz is a baz
FAILED tests 2, 4, 7
        Failed 3/6 tests, 50.00% okay
Failed Test Stat Wstat Total Fail  Failed  List of Failed
-------------------------------------------------------------------------------
t/php/bar.t                 6    3  50.00%  2 4 7
Failed 1/1 test scripts, 0.00% okay. 1/6 subtests failed, 83.33% okay.
```

note that the actual test file that was run was "t/php/bar.t". this file is autogenerated based on the "t/response/TestFoo/bar.php" pattern of your PHP script. "t/php/bar.t" happens to be written in Perl, but you really don't need to worry about it too much.

as an interesting aside, if you are using perl-5.8.3 or later you can actually create your own "t/foo.php" client-side scripts and they will be run via php (using our "php.ini"). but more on that later...

SEE ALSO

the best source of information about using Apache-Test with PHP (at this time) is probably the talk given at ApacheCon 2004 (<http://xrl.us/phpperl>), as well as the code from the talk (<http://xrl.us/phpperlcode>). there is also the online tutorial <http://perl.apache.org/docs/general/testing/testing.html> which has all of the mod_perl-specific syntax and features have been ported to PHP with this class.

AUTHOR

"Apache-Test" is a community effort, maintained by a group of dedicated volunteers. Questions can be asked at the test-dev <at> httpd.apache.org list For more information see: http://httpd.apache.org/test/.