


The Tester's Toolkit:

Start Testing Your Projects Today

Pete Krawczyk

Testing? How boring!

- Spend less time on bugs and regressions
- Solidify your application's behavior
- Refactor without worry and stress
- Regular exercise makes the project stronger
- Stronger code leads to better development

Standard module install

```
$ cd libwww-perl-5.803
$ perl Makefile.PL
...
$ make
...
$ make test
/usr/bin/perl t/TEST 0
base/common-req.......ok
base/cookies..........ok
base/date.............ok
base/headers-auth.....ok
base/headers-etag.....ok
base/headers-util.....ok
...
local/autoload........ok
local/get.............ok
local/http-get........ok
local/http............ok
local/protosub........ok
All tests successful.
Files=30, Tests=759, 25 wallclock secs ( 4.40 cusr +  1.27 csys =  5.67 CPU)
$ sudo make install
...
```

What is a test?

- A Perl program with extra modules
- Reports actual vs. expected results

List-Cycle-0.02/t/next.t

```
...
my $cycle = List::Cycle->new( {vals=> [2112, 5150, 90125]} );
isa_ok( $cycle, 'List::Cycle' );

is( $cycle->next,  2112, q{We are the priests} );
is( $cycle->next,  5150, q{Why can't this be love} );
is( $cycle->next, 90125, q{You can fool yourself} );
is( $cycle->next,  2112, q{What can this strange device be?} );
...
```

Running a test

- `make test` during module install
- `prove` a directory full of tests or a file
- `t/TEST` a directory full of tests or a file
- You can also run one by hand with Perl

Let's write some tests!

- `Acme::PETEK::Testkit`
- Example code to introduce testing
- Simple counter class and scripts

Acme::PETEK::Testkit

```
use Acme::PETEK::Testkit qw(add subtract);
my $c = Acme::PETEK::Testkit->new;

$c->incr;  $c->incr(3);
$c->decr;  $c->decr(3);
$c->reset; $c->reset(3);

my $v = $c->value;
my $s = $c->sign;

$c->incr(add(2,3));
$c->decr(add(2,3));

$c->incr(subtract(5,2));
$c->decr(subtract(5,2));
```

Common Aspects

- Make sure your most important code is tested
  - More is better, but don't jump through hoops
- Testing files should have a "plan"
- Don't print to STDOUT - use `diag()`
- Test for failure as well as success - don't assume
- Give tests a description, if applicable

Writing the first test

t/00_load.t

```
#!/usr/bin/perl -w

use strict;
use Test::More tests => 1;

BEGIN {
  use_ok('Acme::PETEK::Testkit')
}
```

Running the first test

Assumptions:

- Running from project root
- Project libraries in `./lib/`
- Tests in `./t/`

```
$ perl -Ilib t/00_load.t
1..1
ok 1 - use Acme::PETEK::Testkit;


$ prove -Ilib t/
t/00_load....ok
All tests successful.
Files=1, Tests=1,  0 wallclock secs ( 0.05 cusr +  0.02 csys =  0.07 CPU)
```

Test::More

- Rich testing methods for data and objects
  - Data: `is()` `cmp_ok()` `like()`
  - References: `is_deeply()` `eq_array()`
  - Modules: `isa_ok()` `can_ok()`
- Outputs diagnostics when tests fail


Test::More Example

t/basic.t

```
#!/usr/bin/perl -w

use strict;
use Test::More tests => 4;
BEGIN {
    use_ok('Acme::PETEK::Testkit');
}

my $c = Acme::PETEK::Testkit->new;
isa_ok($c, 'Acme::PETEK::Testkit');

$c->incr;
cmp_ok($c->value,'==',1,'first increment goes to 1');
is($c->sign,'positive','counter sign is positive');
```

Successful test output

```
$ prove -Ilib t/basic.t
t/basic....ok
All tests successful.
Files=1, Tests=4,  0 wallclock secs ( 0.04 cusr +  0.02 csys =  0.06 CPU)
```

```
$ prove -Ilib -v t/basic.t
t/basic....1..4
ok 1 - use Acme::PETEK::Testkit;
ok 2 - The object isa Acme::PETEK::Testkit
ok 3 - first increment goes to 1
ok 4 - counter sign is positive
ok
All tests successful.
Files=1, Tests=4,  0 wallclock secs ( 0.04 cusr +  0.02 csys =  0.06 CPU)
```

Failed test output

```
# added an extra $c->incr to the test, breaking the test

$ prove -Ilib t/basic.t
t/01_basic_simple....
#     Failed test (t/01_basic_simple.t at line 15)
# Looks like you failed 1 test of 4.
t/01_basic_simple....dubious
        Test returned status 1 (wstat 256, 0x100)
DIED. FAILED test 3
        Failed 1/4 tests, 75.00% okay
Failed Test          Stat Wstat Total Fail  Failed  List of Failed
-------------------------------------------------------------------------------
t/01_basic_simple.t    1   256     4    1  25.00%  3
Failed 1/1 test scripts, 0.00% okay. 1/4 subtests failed, 75.00% okay.

$ prove -Ilib -v t/basic.t
...
ok 1 - use Acme::PETEK::Testkit;
ok 2 - The object isa Acme::PETEK::Testkit
not ok 3 - first increment goes to 1
ok 4 - counter sign is positive
...
```

Test::More Functions and Failure Output

```
#!/usr/bin/perl -w
use Test::More tests => 14;
BEGIN { use_ok('FileHandle');
        use_ok('F1L3H4NDL3');
}
ok(1,'success');
ok(0,'failure');
diag('This is a comment.');
is('a','a','a eq a');
is('a','b','a eq b');
cmp_ok('1','<','2','one less than two');
cmp_ok('1','>','2','one greater than two');
like('abc',qr/b/,'b in abc');
like('abc',qr/d/,'d in abc');
is_deeply({a=>1},{a=>1},'refs have equal data');
is_deeply({a=>1},{b=>2},'refs are different');
isa_ok(FileHandle->new,'FileHandle');
isa_ok('FileHandle','FileHandle');
```

```
$ prove -v interface.t
interface....1..14
ok 1 - use FileHandle;
not ok 2 - use F1L3H4NDL3;
#     Failed test (interface.t at line 7)
#     Tried to use 'F1L3H4NDL3'.
#     Error:  Can't locate F1L3H4NDL3.pm in @INC (@INC contains...) at (eval 7) line 2.
# BEGIN failed--compilation aborted at interface.t line 7.
ok 3 - success
not ok 4 - failure
#     Failed test (interface.t at line 11)
# This is a comment.
ok 5 - a eq a
not ok 6 - a eq b
#     Failed test (interface.t at line 16)
#          got: 'a'
#     expected: 'b'
ok 7 - one less than two
not ok 8 - one greater than two
#     Failed test (interface.t at line 19)
#     '1'
#         >
#     '2'
ok 9 - b in abc
not ok 10 - d in abc
#     Failed test (interface.t at line 22)
#                   'abc'
#     doesn't match '(?-xism:d)'
ok 11 - refs have equal data
not ok 12 - refs are different
#     Failed test (interface.t at line 25)
#     Structures begin differing at:
#          $got->{b} = Does not exist
#     $expected->{b} = '2'
ok 13 - The object isa FileHandle
not ok 14 - The object isa FileHandle
#     Failed test (interface.t at line 28)
#     The object isn't a reference
# Looks like you failed 7 tests of 14.
dubious
        Test returned status 7 (wstat 1792, 0x700)
DIED. FAILED tests 2, 4, 6, 8, 10, 12, 14
        Failed 7/14 tests, 50.00% okay
Failed Test Stat Wstat Total Fail  Failed  List of Failed
-------------------------------------------------------------------------------
interface.t    7  1792    14    7  50.00%  2 4 6 8 10 12 14
Failed 1/1 test scripts, 0.00% okay. 7/14 subtests failed, 50.00% okay.
```

Skip and TODO

- Skip tests in certain cases
- Test with TODO, then implement

t/skip-todo.t

```
#!/usr/bin/perl -w
use Test::More tests => 3;

SKIP: {
        skip "Didn't find item", 2 unless $item;
        is($item->status,'Available',"We can ship it!");
        cmp_ok($item->cost,'==',1.95,'Everything is 1.95');
}

TODO: {
        local $TODO = 'Implement cost_cdn';
        cmp_ok(cost_cdn(1.95),'==',2.39,'Everything in Canada is 2.39');
}
sub cost_cdn {};
```

Test::Inline

- Put testing in your code as POD blocks
- Advantage: tests, code and docs together
- Disadvantage: Easier to change by mistake
- Uses `Test::More` function names
- Convert to .t files with `inline2test`

Test::Inline Building

lib/Acme/PETEK/Testkit.pm

```
...
=head1 SYNOPSIS

This Perl module is intended to be a collection of sample code for
the Tester's Toolkit presentation at YAPC::NA 2005 by the author.

=for example begin

  use Acme::PETEK::Testkit;
  my $c = Acme::PETEK::Testkit->new;
  $c->incr;

=for example end

=begin testing

  my $c = Acme::PETEK::Testkit->new;
  $c->incr;
  cmp_ok($c->value,'==',1,'incr sends value to 1');

=end testing
...
```

perldoc and inline2test

```
$ perldoc lib/Acme/PETEK/Testkit.pm
...
SYNOPSIS
        This Perl module is intended to be a workspace for the
        Tester's Toolkit presentation at YAPC::NA 2005 by the
        author.

            use Acme::PETEK::Testkit;
            my $c = Acme::PETEK::Testkit->new;
            $c->incr;

CONSTRUCTOR
...

$ inline2test --input=lib --output=t
   (creates t/acme_petek_testkit.t)
```

Test::Simple

- A very basic test module
- Only uses `ok()`

t/basic_simple.t

```
#!/usr/bin/perl -w
use strict;
use Test::Simple tests => 4;
BEGIN { eval { use Acme::PETEK::Testkit; };
        ok(!$@,'module loads OK'); }
my $c = Acme::PETEK::Testkit->new;
ok($c,'object returned');
$c->incr;
ok($c->value == 1,'first increment goes to 1');
ok($c->sign eq 'positive','counter sign is positive');
```

Test::Legacy

- Test::Legacy derives from Test.pm
- Use Test::Legacy to migrate Test.pm tests

t/basic_legacy.t

```
#!/usr/bin/perl -w
use strict;
use Test::Legacy;
BEGIN { plan tests => 4;
        eval {use Acme::PETEK::Testkit; }; ok !$@; }
my $c = Acme::PETEK::Testkit->new;
ok $c;
$c->incr;
ok $c->value, 1, 'first increment goes to 1';
ok $c->sign, 'positive', 'counter sign is positive';
```

Other Perl modules

- Other test modules add methods
- Simplify complex tasks like web browsing
- Most test modules can be easily combined

Apache::Test

- Creates an Apache environment
- Allows live web request testing
- Uses `Test::Legacy` syntax
- Requires Apache binary in test environment
- Also requires extra setup to use

Apache::Test Example

t/handler.t

```
#!/usr/bin/perl -w
use strict;
use Apache::Test qw(ok have_lwp plan);
use Apache::TestRequest qw(GET);

plan tests => 6;

my $r = GET '/count';
ok $r->is_success;
ok $r->content =~ /name="cur" value="(\d*)"/;
ok $1, 0, 'value starts at zero';
$r = GET '/count?incr1=%3E';
ok $r->is_success;
ok $r->content =~ /name="cur" value="(\d*)"/;
ok $1, 1, 'value increased to 1';
```

Test::WWW::Mechanize

- Simplifies scripted traversal of sites
- Handles cookies and form values
- Checks page content

Mechanize Example

t/browser.t

```
#!/usr/bin/perl -w

use strict;
use Test::More tests => 4;
use Test::WWW::Mechanize;
use Apache::TestRequest;
my $url = Apache::TestRequest::module2url('/count');

my $m = Test::WWW::Mechanize->new;
$m->get_ok($url,undef,'load counter page');
cmp_ok($m->value('cur'),'==',0,'form value starts at zero');
$m->click('incr1');
ok($m->success,'clicked incr1');
cmp_ok($m->value('cur'),'==',1,'form value increased to 1');
```

Test::DatabaseRow

- Quick database data tester
- Fetches data and checks validity
- Just assign a database handle to run against
- Can even generate the SQL for you

Test::DatabaseRow Example

t/dbrow.t

```
#!/usr/bin/perl -w
use strict;
use Test::More;
use Test::DatabaseRow;
use DBI;
eval "use DBD::SQLite";
plan skip_all => "DBD::SQLite required" if $@;
plan tests => 5;
my $dbh = DBI->connect("dbi:SQLite:dbname=db.sqlite","","");
isa_ok($dbh,'DBI::db');
local $Test::DatabaseRow::dbh = $dbh;

ok($dbh->do('CREATE TABLE foo ( id int, value varchar(10) )'),'table created');
ok($dbh->do('INSERT INTO foo (id,value) VALUES (?,?)',,1,"bar"),'row inserted');

row_ok( table => 'foo',
        where => [ id => 1 ],
        tests => [ value => "bar" ],
        label => "row 1 has value 'bar'");

ok(unlink('db.sqlite'),'db.sqlite removed');
```

Test::Expect

- Test interactive console apps
- Allows remote test execution via ssh/telnet
- Handles command input and output

Test::Expect Example

t/expect.t

```
#!/usr/bin/perl -w
use strict;
use Test::Expect;
use Test::More tests => 6;

expect_run(
   command => "perl -I../lib ../scripts/lc.pl",
   prompt  => "> ",
   quit    => ".",
);
expect_send("t","Sent pattern of 't'");
expect_send("t","Sent a 't'");
expect_send("u","Sent a 'u'");
expect_send("?","Asked for current matches");
expect_like(qr/Matches: 1/,"Expecting one match");
```

Test::Pod

- Checks project modules' POD syntax
- Standard test used on CPAN

t/pod.t

```
#!perl -T

use Test::More;
eval "use Test::Pod 1.14";
plan skip_all => "Test::Pod 1.14 required" if $@;
all_pod_files_ok();
```

Test::Pod::Coverage

- Checks project modules' POD coverage
- Standard test used on CPAN

t/pod_coverage.t

```
#!perl -T

use Test::More;
eval "use Test::Pod::Coverage 1.04";
plan skip_all => "Test::Pod::Coverage 1.04 required" if $@;
all_pod_files_ok();
```

Other Test Modules

- Template::Test
  - Helps test Template Toolkit v.2 templates
- Test::Differences
  - Puts test diffs in a table for viewing
- Test::LongString
  - Long string differences are abbreviated
- Test::Number::Delta
  - Checks numbers within a tolerance
- Test::SQL::Translator
  - Checks an expected schema against a real schema

Verifying your testing

- `Devel::Cover`, **available on CPAN**
- Transparently runs with your tests
- Compiles statistics on code use
- Creates reports to show test coverage


Running Devel::Cover

```
$ make test
...
Files=12, Tests=42,  6 wallclock secs ( 2.39 cusr +  0.78 csys =  3.17 CPU)
[warning] server localhost:8529 shutdown
$ HARNESS_PERL_SWITCHES='-MDevel::Cover' make test
...
Files=12, Tests=42, 70 wallclock secs (56.01 cusr +  4.18 csys = 60.19 CPU)
[warning] server localhost:8529 shutdown
$ cover
Reading database from .../Acme-PETEK-Testkit/cover_db
---------------------------- ------ ------ ------ ------ ------ ------ ------
File                           stmt branch   cond    sub    pod   time  total
---------------------------- ------ ------ ------ ------ ------ ------ ------
.../Apache/TestConfigData.pm  100.0    n/a    n/a  100.0    n/a   18.9  100.0
...lib/Acme/PETEK/Testkit.pm   60.0   25.0    n/a   60.0  100.0   23.3   60.7
...PETEK/Testkit/modperl1.pm   41.7    0.0    0.0   62.5  100.0   36.9   31.5
scripts/lc.pl                 100.0    n/a    n/a  100.0    n/a   20.9  100.0
Total                          57.7    9.1    0.0   68.2  100.0  100.0   50.3
---------------------------- ------ ------ ------ ------ ------ ------ ------
Writing HTML output to .../Acme-PETEK-Testkit/cover_db/coverage.html ...
done.
```

Coverage Summary

Coverage Summary

Database: /Users/petek/dev/testkit/trunk/Acme-PETEK-Testkit/cover_db

file	stmt	branch	cond	sub	pod	time	total
/Users/petek/.apache-test/Apache/TestConfigData.pm	100.0	n/a	n/a	100.0	n/a	18.9	100.0
blib/lib/Acme/PETEK/Testkit.pm	60.0	25.0	n/a	60.0	100.0	23.3	60.7
blib/lib/Acme/PETEK/Testkit/modperl1.pm	41.7	0.0	0.0	62.5	100.0	36.9	31.5
scripts/lc.pl	100.0	n/a	n/a	100.0	n/a	20.9	100.0
Total	57.7	9.1	0.0	68.2	100.0	100.0	50.3

File Coverage

File Coverage

File:	blib/lib/Acme/PETEK/Testkit.pm
Coverage:	60.7%

line	stmt	branch	cond	sub	pod	time	code
1							`package Acme::PETEK::Testkit;`
							...
78							`sub incr {`
79	3			3	1	27	`my ($self, $int) = @_;`
80	3	50				149	`$int = 1 unless defined($int);`
81	3					30	`$self->{'_counter'} += $int;`
82	3					32	`return $self->value;`
83							`}`
							...
92							`sub decr {`
93	0			0	1	0	`my ($self, $int) = @_;`
94	0	0				0	`$int = 1 unless defined($int);`
95	0					0	`$self->{'_counter'} -= $int;`

Branch Coverage

Branch Coverage

File:	blib/lib/Acme/PETEK/Testkit.pm
Coverage:	25.0%

line	%	coverage		branch
66	0	T	F	`unless defined $int`
80	50	T	F	`unless defined $int`
94	0	T	F	`unless defined $int`
118	50	T	F	`if $$self{'_counter'} < 0`

Test other languages

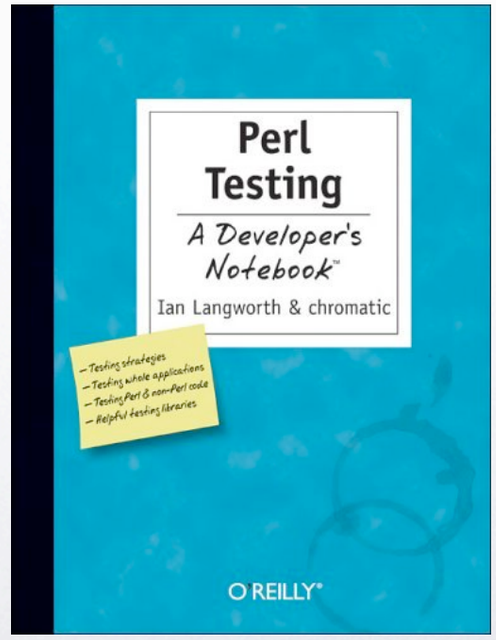
- PHP - `Apache::Test` or via CLI
- JavaScript - http://xrl.us/jsts, http://xrl.us/jstap
- C - http://xrl.us/libtap
- Roll your own with TAP
  - `perldoc Test::Harness::TAP`
  - http://xrl.us/tapf

Next Steps

- "Is the test written for that bug you're fixing?"
- Automate your automated tests
- Consider test-first development
- Help write test for modules you use
- Encourage others to test their code

Also see

- Perl Testing: A Developer's Notebook
  http://www.oreilly.com/catalog/perltestingadn/
- `Test::Tutorial`
- http://qa.perl.org/



Thanks for coming!

More testing at YAPC

- Solutions to Common Testing Problems - chromatic
  - Grand Ballroom East, 1:30 PM
- Writing Tests with Apache-Test - Geoffrey Young
  - Grand Ballroom Central, 1:30 PM
- Phalanx: From the Trenches - Marc Prewitt & Jim Keenan
  - Grand Ballroom East, 3:15 PM
- Testing C Projects with Perl - Stig Brautaset
  - Grand Ballroom East, 3:55 PM
- Lazy Test Development - Joe McMahon
  - Giovanni Room, 9:20 AM Wednesday

Bonus Section

- Why test?
- Perl's testing platform
- Test Anything Protocol (TAP)
- Test::Harness

Why test?

- Verify completeness
- Know what's broken and what's not
- Ensure changes are deliberate
- Improve confidence in code
- Refactor with impunity

Testing Platform

Test::Simple	Test::More
Test::Inline	Test::Legacy
Apache::Test	Test::WWW::Mechanize
Test::DatabaseRow	Test::Expect
Test::Pod	Test::Pod::Coverage
Test::Builder	
Test Anything Protocol (TAP)	
Test::Harness	
make test	prove	t/TEST

At the center: TAP

- "Test Anything Protocol"
- Simple text result format
- Allows custom test development without forcing Perl or writing binary formats
- `perldoc Test::Harness::TAP`
  - http://xrl.us/tapf

Test::Harness - test glue

- Responsible for the testing environment
- Runs the tests
- Summarizes results
- Includes the “prove” convenience wrapper
  - prove is in Perl as of 5.8.3