Perl Testing Reference Card

Version 1.0, 04 Jul 2004

http://langworth.com/PerlTestCard

All examples start with Test::More and use the module in thet heading unless specified otherwise. These are mostly refactored synopses. Enjoy!

Test::More

```
use Test::More 'no_plan';
use Test::More tests => 25;
use Test::More skip_all => 'not finished';

BEGIN {
  use_ok  ( 'Some::Module' );
}
require_ok( 'Some::Module' );

diag     ( 'here is a comment, no test'      );
ok       ( $num==30, 'num is thirty'         );
is       ( $grass, 'green', 'its green'      );
isnt     ( $sky, 'blue', 'the sky isnt blue' );
like     ( $msg, qr/hello/, 'is a greeting'  );
unlike   ( $msg, qr/\d/, 'no digits'         );
cmp_ok   ( $a, '!=', $b, 'not equals'        );
can_ok   ( $obj, @methods                    );
isa_ok   ( $obj, $class                      );
isa_ok   ( $hashref, 'HASH'                  );
is_deeply( \%hash1, \%hash2, 'equal hashes'  );

SKIP: {
  skip $why, $num unless $feature;
  # do tests
}
TODO: {
  local $TODO = $why;
  # do tests
}
```

Test::Differences

```
# takes an optional last argument: name of test
# shows diff in diagnostics if tests fail
eq_or_diff $got_para, $expected_ para;
eq_or_diff \@got_array, \@expected_array;
```

Test::LongString

```
# show only a portion of the string when not ok
use Test::LongString; # 50 char of string is default
use Test::LongString max => 100; # or do this
is_string( $string1, $string2 );
```

Test::Env

```
env_ok( TERM, 'VT220' );
```

Test::Deep

```
cmp_deeply( $hashref1, $hashref2, "hashes match" );

my %person = (
  name   => 'John Q. Taxpayer',
  phone  => '6175551212',
  colors => [qw( red green )]
);
cmp_deeply(
  \%person,
  {
    name   => re( '/^John/' ),
    phone  => re( '/\d{10}/' ),
    colors => array_each( re( '/^(red|green|blue)$/' ),
  },
  "person hash is okay',
);
```

Test::XML

```
is_xml   ( '<stuff/>', '<stuff></stuff>' );
isnt_xml ( '<stuff/>', '<thing/>'        );
```

Test::Cmd

```
$test = Test::Cmd->new(
  prog    => 'my_program',
  workdir => 'data',
  verbose => 1 );

$test->run( args => '-a -b -c28'    );
ok( $?==0, 'program was successful' );
```

Test::File

```
# all tests take an optional extra
# argument that is the test message

file_exists_ok        ( 'file.txt' );
file_empty_ok         ( 'file.txt' );
file_size_ok          ( 'file.txt', 1024 );
file_max_size_ok      ( 'file.txt', 1024 );
file_min_size_ok      ( 'file.txt', 1024 );
file_readable_ok      ( 'file.txt' );
file_writeable_ok     ( 'file.txt' );
file_executable_ok    ( 'file.txt' );

file_not_exists_ok    ( 'file.txt' );
file_not_empty_ok     ( 'file.txt' );
file_not_readable_ok  ( 'file.txt' );
file_not_writeable_ok ( 'file.txt' );
file_not_executable_ok( 'file.txt' );
```

Test::Warn

```
# all tests take optional last argument of
# name of test

# check for a single warning
warning_is    { do_stuff() }
  "No arguments";
warning_is    { do_stuff() }
  {carped => "No arguments"};
warning_like  { do_stuff() }'
  qr/Args/i;

# check for more than one warning
warnings_are  { do_stuff() }
  ["No arguments", "Its Tuesday"];
warnings_like { do_stuff() }
  [qr/Args/i, qr/Undefined/i];

# check for warnings by name
warnings_like { do_stuff() }
  [ 'recursion', 'bareword, 'void' ];

# check for no warning
warnings_are  { do_stuff() } [];
```

Test::NoWarnings

```
use Test::NoWarnings;
# automatically adds one extra test to ensure
# that the test script emitted no warnings
```

Test::Exception

```
dies_ok   { do_stuff() }
  'do_stuff should die';
lives_ok  { do_stuff() }
  'do_stuff should NOT die';
throws_ok { do_stuff() } qr/division by zero/,
  'divide by zero was caught';
throws_ok { do_stuff() } 'Error::Simple',
  'simple error was thrown';
lives_and { ok($x==1)  }
  'x was 1, no exception thrown';
```

Test::DatabaseRow

```
# takes optional 'label' key which is test name
# simple, with SQL
row_ok(
  sql   => 'SELECT * FROM people WHERE pid = 24',
  tests => [ name => 'Bob' ],
);
# simple, with shortcuts
row_ok(
  table => 'people',
  where => [ pid  => 24 ],
  tests => [ name => 'Bob' ],
);
# complex
row_ok(
  table => 'people',
  where => { '='    => { name => 'Bob'       },
             'like' => { url  => '%some.com' },
           },
  tests => { '=='   => { pid  => 24          },
             'eq'   => { city => 'Concord'   },
             '=~'   => { type => qr/^(a|b)$/ },
);
```

Test::Inline

```
# on command line: pod2test lib/MyModule.pm t/mymodule.t
# see Test::Inline::Tutorial

=item * C<do_stuff> - Do stuff does things.

=begin testing

my $var = do_stuff();
ok( $var == 1 );

=end testing

=cut

sub do_stuff { ... }
```

Test::Distribution

```
# auto-creates tests for all modules and their POD,
# syntax, $VERSION and more.
# instead of 'use Test::Distribution;', do:
eval "require Test::Distribution";
plan skip_all => "no Test::Distribution" if $@;
import Test::Distribution;
```

Test::Pod

```
# test a single POD file
pod_file_ok( 'stuff.pod', 'stuff docs are valid POD' );
# all POD files in distribution
all_pod_files_ok();
```

Test::Prereq

```
eval "use Test::Prereq";
plan skip_all => "no Test::Prereq" if $@;
prereq_ok();

# perl version, test name, or module
# names to skip
prereq_ok( $version, $name, \@skip );
```

Test::Signature

```
# verify that the generated signature
# made with Module::Signature is correct
signature_ok();
```

Acme::Test::Weather

```
is_rainy();
isnt_rainy();
is_cloudy();
# etc..
```