NAME

Test::Simple - Basic utilities for writing tests.

SYNOPSIS

```
use Test::Simple tests => 1;

ok( $foo eq $bar, 'foo is bar' );
```

DESCRIPTION

** If you are unfamiliar with testing **read Test::Tutorial** first! **

This is an extremely simple, extremely basic module for writing tests suitable for CPAN modules and other pursuits. If you wish to do more complicated testing, use the Test::More module (a drop-in replacement for this one).

The basic unit of Perl testing is the ok. For each thing you want to test your program will print out an "ok" or "not ok" to indicate pass or fail. You do this with the ok() function (see below).

The only other constraint is you must pre-declare how many tests you plan to run. This is in case something goes horribly wrong during the test and your test program aborts, or skips a test or whatever. You do this like so:

```
use Test::Simple tests => 23;
```

You must have a plan.

ok

```
ok( $foo eq $bar, $name );
ok( $foo eq $bar );
```

ok() is given an expression (in this case `$foo eq $bar`). If it's true, the test passed. If it's false, it didn't. That's about it.

ok() prints out either "ok" or "not ok" along with a test number (it keeps track of that for you).

```
# This produces "ok 1 - Hell not yet frozen over" (or not ok)
ok( get_temperature($hell) > 0, 'Hell not yet frozen over' );
```

If you provide a $name, that will be printed along with the "ok/not ok" to make it easier to find your test when if fails (just search for the name). It also makes it easier for the next guy to understand what your test is for. It's highly recommended you use test names.

All tests are run in scalar context. So this:

```
ok( @stuff, 'I have some stuff' );
```

will do what you mean (fail if stuff is empty)

Test::Simple will start by printing number of tests run in the form "1..M" (so "1..5" means you're going to run 5 tests). This strange format lets Test::Harness know how many tests you plan on running in case something goes horribly wrong.

If all your tests passed, Test::Simple will exit with zero (which is normal). If anything failed it will exit with how many failed. If you run less (or more) tests than you planned, the missing (or extras) will be considered failures. If no tests were ever run Test::Simple will throw a warning and exit with 255. If the test died, even after having successfully completed all its tests, it will still be considered a failure and will exit with 255.

So the exit codes are...

```
    0                   all tests successful
    255                 test died
    any other number    how many failed (including missing or extras)
```

If you fail more than 254 tests, it will be reported as 254.

This module is by no means trying to be a complete testing system. It's just to get you started. Once you're off the ground its recommended you look at *Test::More*.

EXAMPLE

Here's an example of a simple .t file for the fictional Film module.

```
    use Test::Simple tests => 5;

    use Film;  # What you're testing.

    my $btaste = Film->new({ Title    => 'Bad Taste',
                             Director => 'Peter Jackson',
                             Rating   => 'R',
                             NumExplodingSheep => 1
                           });
    ok( defined($btaste) and ref $btaste eq 'Film',     'new() works' );

    ok( $btaste->Title      eq 'Bad Taste',     'Title() get'    );
    ok( $btaste->Director   eq 'Peter Jackson', 'Director() get' );
    ok( $btaste->Rating     eq 'R',             'Rating() get'   );
    ok( $btaste->NumExplodingSheep == 1,        'NumExplodingSheep() get'
);
```

It will produce output like this:

```
    1..5
    ok 1 - new() works
    ok 2 - Title() get
    ok 3 - Director() get
    not ok 4 - Rating() get
    #    Failed test (t/film.t at line 14)
    ok 5 - NumExplodingSheep() get
    # Looks like you failed 1 tests of 5
```

Indicating the Film::Rating() method is broken.

CAVEATS

Test::Simple will only report a maximum of 254 failures in its exit code. If this is a problem, you probably have a huge test script. Split it into multiple files. (Otherwise blame the Unix folks for using an unsigned short integer as the exit status).

Because VMS's exit codes are much, much different than the rest of the universe, and perl does horrible mangling to them that gets in my way, it works like this on VMS.

```
    0     SS$_NORMAL        all tests successful
    4     SS$_ABORT         something went wrong
```

Unfortunately, I can't differentiate any further.

NOTES

Test::Simple is **explicitly** tested all the way back to perl 5.004.

Test::Simple is thread-safe in perl 5.8.0 and up.

HISTORY

This module was conceived while talking with Tony Bowden in his kitchen one night about the problems I was having writing some really complicated feature into the new Testing module. He observed that the main problem is not dealing with these edge cases but that people hate to write tests **at all**. What was needed was a dead simple module that took all the hard work out of testing and was really, really easy to learn. Paul Johnson simultaneously had this idea (unfortunately, he wasn't in Tony's kitchen). This is it.

SEE ALSO

Test::More

More testing functions! Once you outgrow Test::Simple, look at Test::More. Test::Simple is 100% forward compatible with Test::More (i.e. you can just use Test::More instead of Test::Simple in your programs and things will still work).

Test

The original Perl testing module.

Test::Unit

Elaborate unit testing.

Test::Inline, *SelfTest*

Embed tests in your code!

Test::Harness

Interprets the output of your test program.

AUTHORS

Idea by Tony Bowden and Paul Johnson, code by Michael G Schwern <schwern@pobox.com>, wardrobe by Calvin Klein.

COPYRIGHT