NAME

Test::Harness - Run Perl standard test scripts with statistics

VERSION

Version 2.64

SYNOPSIS

```
  use Test::Harness;

  runtests(@test_files);
```

DESCRIPTION

STOP! If all you want to do is write a test script, consider using Test::Simple. Test::Harness is the module that reads the output from Test::Simple, Test::More and other modules based on Test::Builder. You don't need to know about Test::Harness to use those modules.

Test::Harness runs tests and expects output from the test in a certain format. That format is called TAP, the Test Anything Protocol. It is defined in *Test::Harness::TAP*.

`Test::Harness::runtests(@tests)` runs all the testscripts named as arguments and checks standard output for the expected strings in TAP format.

The *prove* utility is a thin wrapper around Test::Harness.

Taint mode

Test::Harness will honor the `-T` or `-t` in the #! line on your test files. So if you begin a test with:

```
    #!perl -T
```

the test will be run with taint mode on.

Configuration variables.

These variables can be used to configure the behavior of Test::Harness. They are exported on request.

`$Test::Harness::Verbose`

The package variable `$Test::Harness::Verbose` is exportable and can be used to let `runtests()` display the standard output of the script without altering the behavior otherwise. The *prove* utility's `-v` flag will set this.

`$Test::Harness::switches`

The package variable `$Test::Harness::switches` is exportable and can be used to set perl command line options used for running the test script(s). The default value is `-w`. It overrides `HARNESS_PERL_SWITCHES`.

`$Test::Harness::Timer`

If set to true, and `Time::HiRes` is available, print elapsed seconds after each test file.

Failure

When tests fail, analyze the summary report:

```
  t/base..............ok
  t/nonumbers.........ok
  t/ok................ok
  t/test-harness......ok
  t/waterloo..........dubious
```

```
        Test returned status 3 (wstat 768, 0x300)
DIED. FAILED tests 1, 3, 5, 7, 9, 11, 13, 15, 17, 19
        Failed 10/20 tests, 50.00% okay
Failed Test  Stat Wstat Total Fail  List of Failed
---------------------------------------------------------------
t/waterloo.t    3   768    20   10  1 3 5 7 9 11 13 15 17 19
Failed 1/5 test scripts, 80.00% okay. 10/44 subtests failed, 77.27% okay.
```

Everything passed but *t/waterloo.t*. It failed 10 of 20 tests and exited with non-zero status indicating something dubious happened.

The columns in the summary report mean:

Failed Test

The test file which failed.

Stat

If the test exited with non-zero, this is its exit status.

Wstat

The wait status of the test.

Total

Total number of tests expected to run.

Fail

Number which failed, either from "not ok" or because they never ran.

List of Failed

A list of the tests which failed. Successive failures may be abbreviated (ie. 15-20 to indicate that tests 15, 16, 17, 18, 19 and 20 failed).

FUNCTIONS

The following functions are available.

runtests(@test_files)

This runs all the given *@test_files* and divines whether they passed or failed based on their output to STDOUT (details above). It prints out each individual test which failed along with a summary report and a how long it all took.

It returns true if everything was ok. Otherwise it will `die()` with one of the messages in the DIAGNOSTICS section.

execute_tests(tests => \@test_files, out => *FH)

Runs all the given `@test_files` (just like `runtests()`) but doesn't generate the final report. During testing, progress information will be written to the currently selected output filehandle (usually `STDOUT`), or to the filehandle given by the `out` parameter. The *out* is optional.

Returns a list of two values, `$total` and `$failed`, describing the results. `$total` is a hash ref summary of all the tests run. Its keys and values are this:

```
bonus           Number of individual todo tests unexpectedly passed
max             Number of individual tests ran
ok              Number of individual tests passed
sub_skipped     Number of individual tests skipped
todo            Number of individual todo tests
```

```
    files           Number of test files ran
    good            Number of test files passed
    bad             Number of test files failed
    tests           Number of test files originally given
    skipped         Number of test files skipped
```

If `$total->{bad} == 0` and `$total->{max} > 0`, you've got a successful test.

`$failed` is a hash ref of all the test scripts that failed. Each key is the name of a test script, each value is another hash representing how that script failed. Its keys are these:

```
    name        Name of the test which failed
    estat       Script's exit value
    wstat       Script's wait status
    max         Number of individual tests
    failed      Number which failed
    canon       List of tests which failed (as string).
```

`$failed` should be empty if everything passed.

```
    my($leader, $ml) = _mk_leader($test_file, $width);
```

Generates the 't/foo........' leader for the given `$test_file` as well as a similar version which will overwrite the current line (by use of \r and such). `$ml` may be empty if Test::Harness doesn't think you're on TTY.

The `$width` is the width of the "yada/blah.." string.

```
  my($width) = _leader_width(@test_files);
```

Calculates how wide the leader should be based on the length of the longest test name.

EXPORT

`&runtests` is exported by Test::Harness by default.

`&execute_tests`, `$verbose`, `$switches` and `$debug` are exported upon request.

DIAGNOSTICS

`All tests successful.\nFiles=%d,  Tests=%d, %s`

If all tests are successful some statistics about the performance are printed.

`FAILED tests %s\n\tFailed %d/%d tests, %.2f%% okay.`

For any single script that has failing subtests statistics like the above are printed.

`Test returned status %d (wstat %d)`

Scripts that return a non-zero exit status, both $? >> 8 and $? are printed in a message similar to the above.

`Failed 1 test, %.2f%% okay. %s`

`Failed %d/%d tests, %.2f%% okay. %s`

If not all tests were successful, the script dies with one of the above messages.

`FAILED--Further testing stopped: %s`

If a single subtest decides that further testing will not make sense, the script dies with this message.

ENVIRONMENT VARIABLES THAT TEST::HARNESS SETS

Test::Harness sets these before executing the individual tests.

`HARNESS_ACTIVE`

This is set to a true value. It allows the tests to determine if they are being executed through the harness or by any other means.

`HARNESS_VERSION`

This is the version of Test::Harness.

ENVIRONMENT VARIABLES THAT AFFECT TEST::HARNESS

`HARNESS_COLUMNS`

This value will be used for the width of the terminal. If it is not set then it will default to `COLUMNS`. If this is not set, it will default to 80. Note that users of Bourne-sh based shells will need to `export COLUMNS` for this module to use that variable.

`HARNESS_COMPILE_TEST`

When true it will make harness attempt to compile the test using `perlcc` before running it.

NOTE This currently only works when sitting in the perl source directory!

`HARNESS_DEBUG`

If true, Test::Harness will print debugging information about itself as it runs the tests. This is different from `HARNESS_VERBOSE`, which prints the output from the test being run. Setting `$Test::Harness::Debug` will override this, or you can use the `-d` switch in the *prove* utility.

`HARNESS_FILELEAK_IN_DIR`

When set to the name of a directory, harness will check after each test whether new files appeared in that directory, and report them as

```
LEAKED FILES: scr.tmp 0 my.db
```

If relative, directory name is with respect to the current directory at the moment runtests() was called. Putting absolute path into `HARNESS_FILELEAK_IN_DIR` may give more predictable results.

`HARNESS_NOTTY`

When set to a true value, forces it to behave as though STDOUT were not a console. You may need to set this if you don't want harness to output more frequent progress messages using carriage returns. Some consoles may not handle carriage returns properly (which results in a somewhat messy output).

`HARNESS_PERL`

Usually your tests will be run by `$^X`, the currently-executing Perl. However, you may want to have it run by a different executable, such as a threading perl, or a different version.

If you're using the *prove* utility, you can use the `--perl` switch.

`HARNESS_PERL_SWITCHES`

Its value will be prepended to the switches used to invoke perl on each test. For example, setting `HARNESS_PERL_SWITCHES` to `-W` will run all tests with all warnings enabled.

`HARNESS_TIMER`

Setting this to true will make the harness display the number of milliseconds each test took. You can also use *prove*'s `--timer` switch.

`HARNESS_VERBOSE`

If true, Test::Harness will output the verbose results of running its tests. Setting `$Test::Harness::verbose` will override this, or you can use the `-v` switch in the *prove* utility.

If true, Test::Harness will output the verbose results of running its tests. Setting `$Test::Harness::verbose` will override this, or you can use the `-v` switch in the *prove* utility.

`HARNESS_STRAP_CLASS`

Defines the Test::Harness::Straps subclass to use. The value may either be a filename or a class name.

If HARNESS_STRAP_CLASS is a class name, the class must be in `@INC` like any other class.

If HARNESS_STRAP_CLASS is a filename, the .pm file must return the name of the class, instead of the canonical "1".

EXAMPLE

Here's how Test::Harness tests itself

```
$ cd ~/src/devel/Test-Harness
$ perl -Mblib -e 'use Test::Harness qw(&runtests $verbose);
  $verbose=0; runtests @ARGV;' t/*.t
Using /home/schwern/src/devel/Test-Harness/blib
t/base..............ok
t/nonumbers.........ok
t/ok................ok
t/test-harness......ok
All tests successful.
Files=4, Tests=24, 2 wallclock secs ( 0.61 cusr + 0.41 csys = 1.02 CPU)
```

SEE ALSO

The included *prove* utility for running test scripts from the command line, *Test* and *Test::Simple* for writing test scripts, *Benchmark* for the underlying timing routines, and *Devel::Cover* for test coverage analysis.

TODO

Provide a way of running tests quietly (ie. no printing) for automated validation of tests. This will probably take the form of a version of runtests() which rather than printing its output returns raw data on the state of the tests. (Partially done in Test::Harness::Straps)

Document the format.

Fix HARNESS_COMPILE_TEST without breaking its core usage.

Figure a way to report test names in the failure summary.

Rework the test summary so long test names are not truncated as badly. (Partially done with new skip test styles)

Add option for coverage analysis.

Trap STDERR.

Implement Straps total_results()

Remember exit code

Completely redo the print summary code.

Straps->analyze_file() not taint clean, don't know if it can be

Fix that damned VMS nit.

Add a test for verbose.

Change internal list of test results to a hash.

Fix stats display when there's an overrun.

Fix so perls with spaces in the filename work.

Keeping whittling away at _run_all_tests()

Clean up how the summary is printed. Get rid of those damned formats.

BUGS

Please report any bugs or feature requests to `bug-test-harness at rt.cpan.org`, or through the web interface at *http://rt.cpan.org/NoAuth/ReportBug.html?Queue=Test-Harness*. I will be notified, and then you'll automatically be notified of progress on your bug as I make changes.

SUPPORT

You can find documentation for this module with the *perldoc* command.

```
    perldoc Test::Harness
```

You can get docs for *prove* with

```
    prove --man
```

You can also look for information at:

* AnnoCPAN: Annotated CPAN documentation

  http://annocpan.org/dist/Test-Harness

* CPAN Ratings

  http://cpanratings.perl.org/d/Test-Harness

* RT: CPAN's request tracker

  http://rt.cpan.org/NoAuth/Bugs.html?Dist=Test-Harness

* Search CPAN

  http://search.cpan.org/dist/Test-Harness

SOURCE CODE

The source code repository for Test::Harness is at *http://svn.perl.org/modules/Test-Harness*.

AUTHORS

Either Tim Bunce or Andreas Koenig, we don't know. What we know for sure is, that it was inspired by Larry Wall's *TEST* script that came with perl distributions for ages. Numerous anonymous contributors exist. Andreas Koenig held the torch for many years, and then Michael G Schwern.

Current maintainer is Andy Lester <`andy at petdance.com`>.

COPYRIGHT

Copyright 2002-2006 by Michael G Schwern <`schwern at pobox.com`>, Andy Lester <`andy at petdance.com`>.

This program is free software; you can redistribute it and/or modify it under the same terms as Perl itself.