

NAME

TAP::Harness - Run test scripts with statistics

VERSION

Version 3.17

DESCRIPTION

This is a simple test harness which allows tests to be run and results automatically aggregated and output to STDOUT.

SYNOPSIS

```
use TAP::Harness;
my $harness = TAP::Harness->new( \%args );
$harness->runtests(@tests);
```

METHODS

Class Methods

new

```
my %args = (
    verbosity => 1,
    lib       => [ 'lib', 'blib/lib', 'blib/arch' ],
)
my $harness = TAP::Harness->new( \%args );
```

The constructor returns a new `TAP::Harness` object. It accepts an optional hashref whose allowed keys are:

* verbosity

Set the verbosity level:

1	verbose	Print individual test results to STDOUT.
0	normal	
-1	quiet	Suppress some test output (mostly failures while tests are running).
-2	really quiet	Suppress everything but the tests summary.
-3	silent	Suppress everything.

* timer

Append run time for each test to output. Uses *Time::HiRes* if available.

* failures

Show test failures (this is a no-op if `verbose` is selected).

* comments

Show test comments (this is a no-op if `verbose` is selected).

* show_count

Update the running test count during testing.

* normalize

Set to a true value to normalize the TAP that is emitted in verbose modes.

* lib

Accepts a scalar value or array ref of scalar values indicating which paths to allowed libraries

should be included if Perl tests are executed. Naturally, this only makes sense in the context of tests written in Perl.

* switches

Accepts a scalar value or array ref of scalar values indicating which switches should be included if Perl tests are executed. Naturally, this only makes sense in the context of tests written in Perl.

* test_args

A reference to an @INC style array of arguments to be passed to each test program.

* color

Attempt to produce color output.

* exec

Typically, Perl tests are run through this. However, anything which spits out TAP is fine. You can use this argument to specify the name of the program (and optional switches) to run your tests with:

```
exec => ['/usr/bin/ruby', '-w']
```

You can also pass a subroutine reference in order to determine and return the proper program to run based on a given test script. The subroutine reference should expect the TAP::Harness object itself as the first argument, and the file name as the second argument. It should return an array reference containing the command to be run and including the test file name. It can also simply return undef, in which case TAP::Harness will fall back on executing the test script in Perl:

```
exec => sub {
    my ( $harness, $test_file ) = @_;

    # Let Perl tests run.
    return undef if $test_file =~ /\.t$/;
    return [ qw( /usr/bin/ruby -w ), $test_file ]
        if $test_file =~ /\.rb$/;
}
```

If the subroutine returns a scalar with a newline or a filehandle, it will be interpreted as raw TAP or as a TAP stream, respectively.

* merge

If merge is true the harness will create parsers that merge STDOUT and STDERR together for any processes they start.

* aggregator_class

The name of the class to use to aggregate test results. The default is *TAP::Parser::Aggregator*.

* formatter_class

The name of the class to use to format output. The default is *TAP::Formatter::Console*, or *TAP::Formatter::File* if the output isn't a TTY.

* multiplexer_class

The name of the class to use to multiplex tests during parallel testing. The default is *TAP::Parser::Multiplexer*.

* parser_class

The name of the class to use to parse TAP. The default is *TAP::Parser*.

* `scheduler_class`

The name of the class to use to schedule test execution. The default is *TAP::Parser::Scheduler*.

* `formatter`

If set `formatter` must be an object that is capable of formatting the TAP output. See *TAP::Formatter::Console* for an example.

* `errors`

If parse errors are found in the TAP output, a note of this will be made in the summary report. To see all of the parse errors, set this argument to true:

```
errors => 1
```

* `directives`

If set to a true value, only test results with directives will be displayed. This overrides other settings such as `verbose` or `failures`.

* `ignore_exit`

If set to a true value instruct *TAP::Parser* to ignore exit and wait status from test scripts.

* `jobs`

The maximum number of parallel tests to run at any time. Which tests can be run in parallel is controlled by `rules`. The default is to run only one test at a time.

* `rules`

A reference to a hash of rules that control which tests may be executed in parallel. This is an experimental feature and the interface may change.

```
$harness->rules(
    {   par => [
        { seq => '../ext/DB_File/t/*' },
        { seq => '../ext/IO_Compress_Zlib/t/*' },
        { seq => '../lib/CPANPLUS/*' },
        { seq => '../lib/ExtUtils/t/*' },
        '*'
    ]
};
```

* `stdout`

A filehandle for catching standard output.

Any keys for which the value is `undef` will be ignored.

Instance Methods

`runtests`

```
$harness->runtests(@tests);
```

Accepts an array of `@tests` to be run. This should generally be the names of test files, but this is not required. Each element in `@tests` will be passed to *TAP::Parser::new()* as a source. See *TAP::Parser* for more information.

It is possible to provide aliases that will be displayed in place of the test name by supplying the test as

a reference to an array containing [\$test, \$alias]:

```
$harness->runtests( [ 't/foo.t', 'Foo Once' ],  
                  [ 't/foo.t', 'Foo Twice' ] );
```

Normally it is an error to attempt to run the same test twice. Aliases allow you to overcome this limitation by giving each run of the test a unique name.

Tests will be run in the order found.

If the environment variable `PERL_TEST_HARNESS_DUMP_TAP` is defined it should name a directory into which a copy of the raw TAP for each test will be written. TAP is written to files named for each test. Subdirectories will be created as needed.

Returns a *TAP::Parser::Aggregator* containing the test results.

summary

Output the summary for a *TAP::Parser::Aggregator*.

aggregate_tests

```
$harness->aggregate_tests( $aggregate, @tests );
```

Run the named tests and display a summary of result. Tests will be run in the order found.

Test results will be added to the supplied *TAP::Parser::Aggregator*. `aggregate_tests` may be called multiple times to run several sets of tests. Multiple *Test::Harness* instances may be used to pass results to a single aggregator so that different parts of a complex test suite may be run using different *TAP::Harness* settings. This is useful, for example, in the case where some tests should run in parallel but others are unsuitable for parallel execution.

```
my $formatter = TAP::Formatter::Console->new;  
my $ser_harness = TAP::Harness->new( { formatter => $formatter } );  
my $par_harness = TAP::Harness->new(  
    {   formatter => $formatter,  
        jobs      => 9  
    }  
);  
my $aggregator = TAP::Parser::Aggregator->new;  
  
$aggregator->start();  
$ser_harness->aggregate_tests( $aggregator, @ser_tests );  
$par_harness->aggregate_tests( $aggregator, @par_tests );  
$aggregator->stop();  
$formatter->summary( $aggregator );
```

Note that for simpler testing requirements it will often be possible to replace the above code with a single call to `runtests`.

Each elements of the `@tests` array is either

- * the file name of a test script to run
- * a reference to a [file name, display name] array

When you supply a separate display name it becomes possible to run a test more than once; the display name is effectively the alias by which the test is known inside the harness. The harness doesn't care if it runs the same script more than once when each invocation uses a different name.

make_scheduler

Called by the harness when it needs to create a *TAP::Parser::Scheduler*. Override in a subclass to provide an alternative scheduler. `make_scheduler` is passed the list of tests that was passed to `aggregate_tests`.

jobs

Gets or sets the number of concurrent test runs the harness is handling. By default, this value is 1 -- for parallel testing, this should be set higher.

SUBCLASSING

TAP::Harness is designed to be (mostly) easy to subclass. If you don't like how a particular feature functions, just override the desired methods.

Methods

TODO: This is out of date

The following methods are ones you may wish to override if you want to subclass *TAP::Harness*.

summary

```
$harness->summary( \%args );
```

`summary` prints the summary report after all tests are run. The argument is a hashref with the following keys:

* start

This is created with `Benchmark->new` and it the time the tests started. You can print a useful summary time, if desired, with:

```
$self->output(
    timestr( timediff( Benchmark->new, $start_time ), 'nop' ) );
```

* tests

This is an array reference of all test names. To get the *TAP::Parser* object for individual tests:

```
my $aggregate = $args->{aggregate};
my $tests     = $args->{tests};

for my $name ( @$tests ) {
    my ($parser) = $aggregate->parsers($test);
    ... do something with $parser
}
```

This is a bit clunky and will be cleaned up in a later release.

make_parser

Make a new parser and display formatter session. Typically used and/or overridden in subclasses.

```
my ( $parser, $session ) = $harness->make_parser;
```

finish_parser

Terminate use of a parser. Typically used and/or overridden in subclasses. The parser isn't destroyed as a result of this.

REPLACING

If you like the `prove` utility and *TAP::Parser* but you want your own harness, all you need to do is write one and provide `new` and `runtests` methods. Then you can use the `prove` utility like so:

```
prove --harness My::Test::Harness
```

Note that while `prove` accepts a list of tests (or things to be tested), `new` has a fairly rich set of arguments. You'll probably want to read over this code carefully to see how all of them are being used.

SEE ALSO

Test::Harness