NAME

Test::Harness::TAP - Documentation for the TAP format

SYNOPSIS

TAP, the Test Anything Protocol, is Perl's simple text-based interface between testing modules such as Test::More and the test harness Test::Harness.

TODO

Exit code of the process.

THE TAP FORMAT

TAP's general format is:

```
1..N
ok 1 Description # Directive
# Diagnostic
....
ok 47 Description
ok 48 Description
more tests....
```

For example, a test file's output might look like:

```
1..4
ok 1 - Input file opened
not ok 2 - First line of the input valid
ok 3 - Read the rest of the file
not ok 4 - Summarized correctly # TODO Not written yet
```

HARNESS BEHAVIOR

In this document, the "harness" is any program analyzing TAP output. Typically this will be Perl's *prove* program, or the underlying `Test::Harness::runtests` subroutine.

A harness must only read TAP output from standard output and not from standard error. Lines written to standard output matching `/^(not )?ok\b/` must be interpreted as test lines. All other lines must not be considered test output.

TESTS LINES AND THE PLAN

The plan

The plan tells how many tests will be run, or how many tests have run. It's a check that the test file hasn't stopped prematurely. It must appear once, whether at the beginning or end of the output.

The plan is usually the first line of TAP output and it specifies how many test points are to follow. For example,

```
1..10
```

means you plan on running 10 tests. This is a safeguard in case your test file dies silently in the middle of its run. The plan is optional but if there is a plan before the test points it must be the first non-diagnostic line output by the test file.

In certain instances a test file may not know how many test points it will ultimately be running. In this case the plan can be the last non-diagnostic line in the output.

The plan cannot appear in the middle of the output, nor can it appear more than once.

The test line

The core of TAP is the test line. A test file prints one test line test point executed. There must be at least one test line in TAP output. Each test line comprises the following elements:

* `ok` or `not ok`

    This tells whether the test point passed or failed. It must be at the beginning of the line. `/^not ok/` indicates a failed test point. `/^ok/` is a successful test point. This is the only mandatory part of the line.

    Note that unlike the Directives below, `ok` and `not ok` are case-sensitive.

* Test number

    TAP expects the `ok` or `not ok` to be followed by a test point number. If there is no number the harness must maintain its own counter until the script supplies test numbers again. So the following test output

    ```
    1..6
    not ok
    ok
    not ok
    ok
    ok
    ```

    has five tests. The sixth is missing. Test::Harness will generate

    ```
    FAILED tests 1, 3, 6
    Failed 3/6 tests, 50.00% okay
    ```

* Description

    Any text after the test number but before a # is the description of the test point.

    ```
    ok 42 this is the description of the test
    ```

    Descriptions should not begin with a digit so that they are not confused with the test point number.

    The harness may do whatever it wants with the description.

* Directive

    The test point may include a directive, following a hash on the test line. There are currently two directives allowed: `TODO` and `SKIP`. These are discussed below.

To summarize:

* ok/not ok (required)

* Test number (recommended)

* Description (recommended)

* Directive (only when necessary)

DIRECTIVES

Directives are special notes that follow a # on the test line. Only two are currently defined: `TODO` and `SKIP`. Note that these two keywords are not case-sensitive.

TODO tests

If the directive starts with # `TODO`, the test is counted as a todo test, and the text after `TODO` is the explanation.

```
not ok 13 # TODO bend space and time
```

Note that if the TODO has an explanation it must be separated from TODO by a space.

These tests represent a feature to be implemented or a bug to be fixed and act as something of an executable "things to do" list. They are **not** expected to succeed. Should a todo test point begin succeeding, the harness should report it as a bonus. This indicates that whatever you were supposed to do has been done and you should promote this to a normal test point.

Skipping tests

If the directive starts with # SKIP, the test is counted as having been skipped. If the whole test file succeeds, the count of skipped tests is included in the generated output. The harness should report the text after # SKIP\S*\s+ as a reason for skipping.

```
ok 23 # skip Insufficient flogiston pressure.
```

Similarly, one can include an explanation in a plan line, emitted if the test file is skipped completely:

```
1..0 # Skipped: WWW::Mechanize not installed
```

OTHER LINES

Bail out!

As an emergency measure a test script can decide that further tests are useless (e.g. missing dependencies) and testing should stop immediately. In that case the test script prints the magic words

```
Bail out!
```

to standard output. Any message after these words must be displayed by the interpreter as the reason why testing must be stopped, as in

```
Bail out! MySQL is not running.
```

Diagnostics

Additional information may be put into the testing output on separate lines. Diagnostic lines should begin with a #, which the harness must ignore, at least as far as analyzing the test results. The harness is free, however, to display the diagnostics. Typically diagnostics are used to provide information about the environment in which test file is running, or to delineate a group of tests.

```
...
ok 18 - Closed database connection
# End of database section.
# This starts the network part of the test.
# Daemon started on port 2112
ok 19 - Opened socket
...
ok 47 - Closed socket
# End of network tests
```

Anything else

Any output line that is not a plan, a test line or a diagnostic is incorrect. How a harness handles the incorrect line is undefined. Test::Harness silently ignores incorrect lines, but will become more stringent in the future.

EXAMPLES

All names, places, and events depicted in any example are wholly fictitious and bear no resemblance to, connection with, or relation to any real entity. Any such similarity is purely coincidental, unintentional, and unintended.

Common with explanation

The following TAP listing declares that six tests follow as well as provides handy feedback as to what the test is about to do. All six tests pass.

```
1..6
#
# Create a new Board and Tile, then place
# the Tile onto the board.
#
ok 1 - The object isa Board
ok 2 - Board size is zero
ok 3 - The object isa Tile
ok 4 - Get possible places to put the Tile
ok 5 - Placing the tile produces no error
ok 6 - Board size is 1
```

Unknown amount and failures

This hypothetical test program ensures that a handful of servers are online and network-accessible. Because it retrieves the hypothetical servers from a database, it doesn't know exactly how many servers it will need to ping. Thus, the test count is declared at the bottom after all the test points have run. Also, two of the tests fail.

```
ok 1 - retrieving servers from the database
# need to ping 6 servers
ok 2 - pinged diamond
ok 3 - pinged ruby
not ok 4 - pinged saphire
ok 5 - pinged onyx
not ok 6 - pinged quartz
ok 7 - pinged gold
1..7
```

Giving up

This listing reports that a pile of tests are going to be run. However, the first test fails, reportedly because a connection to the database could not be established. The program decided that continuing was pointless and exited.

```
1..573
not ok 1 - database handle
Bail out! Couldn't connect to database.
```

Skipping a few

The following listing plans on running 5 tests. However, our program decided to not run tests 2 thru 5 at all. To properly report this, the tests are marked as being skipped.

```
1..5
ok 1 - approved operating system
# $^O is solaris
ok 2 - # SKIP no /sys directory
ok 3 - # SKIP no /sys directory
ok 4 - # SKIP no /sys directory
ok 5 - # SKIP no /sys directory
```

Skipping everything

This listing shows that the entire listing is a skip. No tests were run.

```
1..0 # skip because English-to-French translator isn't installed
```

Got spare tuits?

The following example reports that four tests are run and the last two tests failed. However, because the failing tests are marked as things to do later, they are considered successes. Thus, a harness should report this entire listing as a success.

```
1..4
ok 1 - Creating test program
ok 2 - Test program runs, no error
not ok 3 - infinite loop # TODO halting problem unsolved
not ok 4 - infinite loop 2 # TODO halting problem unsolved
```

Creative liberties

This listing shows an alternate output where the test numbers aren't provided. The test also reports the state of a ficticious board game in diagnostic form. Finally, the test count is reported at the end.

```
ok - created Board
ok
ok
ok
ok
ok
ok
# +------+------+------+------+
# |      |16G   |      |05C   |
# |      |G N C |      |C C G |
# |      |  G   |      |  C  +|
# +------+------+------+------+
# |10C   |01G   |      |03C   |
# |R N G |G A G |      |C C C |
# |  R   |  G   |      |  C  +|
# +------+------+------+------+
# |      |01G   |17C   |00C   |
# |      |G A G |G N R |R N R |
# |      |  G   |  R   |  G   |
# +------+------+------+------+
ok - board has 7 tiles + starter tile
1..9
```

AUTHORS

Andy Lester, based on the original Test::Harness documentation by Michael Schwern.

ACKNOWLEDGEMENTS

Thanks to Pete Krawczyk, Paul Johnson, Ian Langworth and Nik Clayton for help and contributions on this document.

The basis for the TAP format was created by Larry Wall in the original test script for Perl 1. Tim Bunce and Andreas Koenig developed it further with their modifications to Test::Harness.

COPYRIGHT

Copyright 2003-2005 by Michael G Schwern `<schwern@pobox.com>`, Andy Lester `<andy@petdance.com>`.

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

See *http://www.perl.com/perl/misc/Artistic.html*.