The Kernel Test Anything Protocol (KTAP), version 1

TAP, or the Test Anything Protocol is a format for specifying test results used by a number of projects. It's website and specification are found at this link. The Linux Kernel largely uses TAP output for test results. However, Kernel testing frameworks have special needs for test results which don't align with the original TAP specification. Thus, a "Kernel TAP" (KTAP) format is specified to extend and alter TAP to support these use-cases. This specification describes the generally accepted format of KTAP as it is currently used in the kernel.

KTAP test results describe a series of tests (which may be nested: i.e., test can have subtests), each of which can contain both diagnostic data -- e.g., log lines -- and a final result. The test structure and results are machine-readable, whereas the diagnostic data is unstructured and is there to aid human debugging.

KTAP output is built from four different types of lines: - Version lines - Plan lines - Test case result lines - Diagnostic lines

In general, valid KTAP output should also form valid TAP output, but some information, in particular nested test results, may be lost. Also note that there is a stagnant draft specification for TAP14, KTAP diverges from this in a couple of places (notably the "Subtest" header), which are described where relevant later in this document.

Version lines

All KTAP-formatted results begin with a "version line" which specifies which version of the (K)TAP standard the result is compliant with.

For example: - "KTAP version 1" - "TAP version 13" - "TAP version 14"

Note that, in KTAP, subtests also begin with a version line, which denotes the start of the nested test results. This differs from TAP14, which uses a separate "Subtest" line.

While, going forward, "KTAP version 1" should be used by compliant tests, it is expected that most parsers and other tooling will accept the other versions listed here for compatibility with existing tests and frameworks.

Plan lines

A test plan provides the number of tests (or subtests) in the KTAP output.

Plan lines must follow the format of "1..N" where N is the number of tests or subtests. Plan lines follow version lines to indicate the number of nested tests.

While there are cases where the number of tests is not known in advance -- in which case the test plan may be omitted -- it is strongly recommended one is present where possible.

Test case result lines

Test case result lines indicate the final status of a test. They are required and must have the format:

System Message: WARNING/2 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\dev-tools\(linux-master)(Documentation)(dev-tools)ktap.rst, line 71)

Cannot analyze code. No Pygments lexer found for "none".

```
.. code-block:: none

        <result> <number> [<description>][ # [<directive>] [<diagnostic data>]]
```

The result can be either "ok", which indicates the test case passed, or "not ok", which indicates that the test case failed.

<number> represents the number of the test being performed. The first test must have the number 1 and the number then must increase by 1 for each additional subtest within the same test at the same nesting level.

The description is a description of the test, generally the name of the test, and can be any string of words (can't include #). The description is optional, but recommended.

The directive and any diagnostic data is optional. If either are present, they must follow a hash sign, "#".

A directive is a keyword that indicates a different outcome for a test other than passed and failed. The directive is optional, and consists of a single keyword preceding the diagnostic data. In the event that a parser encounters a directive it doesn't support, it should fall back to the "ok" / "not ok" result.

Currently accepted directives are:

- "SKIP", which indicates a test was skipped (note the result of the test case result line can be either "ok" or "not ok" if the SKIP directive is used)

- "TODO", which indicates that a test is not expected to pass at the moment, e.g. because the feature it is testing is known to be broken. While this directive is inherited from TAP, its use in the kernel is discouraged.
- "XFAIL", which indicates that a test is expected to fail. This is similar to "TODO", above, and is used by some kselftest tests.
- “TIMEOUT”, which indicates a test has timed out (note the result of the test case result line should be “not ok” if the TIMEOUT directive is used)
- “ERROR”, which indicates that the execution of a test has failed due to a specific error that is included in the diagnostic data. (note the result of the test case result line should be “not ok” if the ERROR directive is used)

The diagnostic data is a plain-text field which contains any additional details about why this result was produced. This is typically an error message for ERROR or failed tests, or a description of missing dependencies for a SKIP result.

The diagnostic data field is optional, and results which have neither a directive nor any diagnostic data do not need to include the "#" field separator.

Example result lines include:

System Message: WARNING/2 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\dev-tools\(linux-master)(Documentation)(dev-tools)ktap.rst, line 120)

Cannot analyze code. No Pygments lexer found for "none".

```
.. code-block:: none

        ok 1 test_case_name
```

The test "test_case_name" passed.

System Message: WARNING/2 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\dev-tools\(linux-master)(Documentation)(dev-tools)ktap.rst, line 126)

Cannot analyze code. No Pygments lexer found for "none".

```
.. code-block:: none

        not ok 1 test_case_name
```

The test "test_case_name" failed.

System Message: WARNING/2 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\dev-tools\(linux-master)(Documentation)(dev-tools)ktap.rst, line 132)

Cannot analyze code. No Pygments lexer found for "none".

```
.. code-block:: none

        ok 1 test # SKIP necessary dependency unavailable
```

The test "test" was SKIPPED with the diagnostic message "necessary dependency unavailable".

System Message: WARNING/2 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\dev-tools\(linux-master)(Documentation)(dev-tools)ktap.rst, line 139)

Cannot analyze code. No Pygments lexer found for "none".

```
.. code-block:: none

        not ok 1 test # TIMEOUT 30 seconds
```

The test "test" timed out, with diagnostic data "30 seconds".

System Message: WARNING/2 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\dev-tools\(linux-master)(Documentation)(dev-tools)ktap.rst, line 145)

Cannot analyze code. No Pygments lexer found for "none".

```
.. code-block:: none

        ok 5 check return code # rcode=0
```

The test "check return code" passed, with additional diagnostic data “rcode=0”

Diagnostic lines

If tests wish to output any further information, they should do so using "diagnostic lines". Diagnostic lines are optional, freeform text, and are often used to describe what is being tested and any intermediate results in more detail than the final result and diagnostic data line provides.

Diagnostic lines are formatted as "# <diagnostic_description>", where the description can be any string. Diagnostic lines can be anywhere in the test output. As a rule, diagnostic lines regarding a test are directly before the test result line for that test.

Note that most tools will treat unknown lines (see below) as diagnostic lines, even if they do not start with a "#": this is to capture any other useful kernel output which may help debug the test. It is nevertheless recommended that tests always prefix any diagnostic output they have with a "#" character.

Unknown lines

There may be lines within KTAP output that do not follow the format of one of the four formats for lines described above. This is allowed, however, they will not influence the status of the tests.

This is an important difference from TAP. Kernel tests may print messages to the system console or a log file. Both of these destinations may contain messages either from unrelated kernel or userspace activity, or kernel messages from non-test code that is invoked by the test. The kernel code invoked by the test likely is not aware that a test is in progress and thus can not print the message as a diagnostic message.

Nested tests

In KTAP, tests can be nested. This is done by having a test include within its output an entire set of KTAP-formatted results. This can be used to categorize and group related tests, or to split out different results from the same test.

The "parent" test's result should consist of all of its subtests' results, starting with another KTAP version line and test plan, and end with the overall result. If one of the subtests fail, for example, the parent test should also fail.

Additionally, all lines in a subtest should be indented. One level of indentation is two spaces: " ". The indentation should begin at the version line and should end before the parent test's result line.

"Unknown lines" are not considered to be lines in a subtest and thus are allowed to be either indented or not indented.

An example of a test with two nested subtests:

System Message: WARNING/2 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\dev-tools\(linux-master)(Documentation)(dev-tools)ktap.rst, line 205)

Cannot analyze code. No Pygments lexer found for "none".

```
.. code-block:: none

        KTAP version 1
        1..1
          KTAP version 1
          1..2
          ok 1 test_1
          not ok 2 test_2
        # example failed
        not ok 1 example
```

An example format with multiple levels of nested testing:

System Message: WARNING/2 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\dev-tools\(linux-master)(Documentation)(dev-tools)ktap.rst, line 218)

Cannot analyze code. No Pygments lexer found for "none".

```
.. code-block:: none

        KTAP version 1
        1..2
          KTAP version 1
          1..2
            KTAP version 1
            1..2
            not ok 1 test_1
            ok 2 test_2
          not ok 1 test_3
```

```
  ok 2 test_4 # SKIP
not ok 1 example_test_1
ok 2 example_test_2
```

Major differences between TAP and KTAP

Feature	TAP	KTAP
yaml and json in diagnosic message	ok	not recommended
TODO directive	ok	not recognized
allows an arbitrary number of tests to be nested	no	yes
"Unknown lines" are in category of "Anything else"	yes	no
"Unknown lines" are	incorrect	allowed

The TAP14 specification does permit nested tests, but instead of using another nested version line, uses a line of the form "Subtest: <name>" where <name> is the name of the parent test.

Example KTAP output

System Message: WARNING/2 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\dev-tools\(linux-master)(Documentation)(dev-tools)ktap.rst, line 253)

Cannot analyze code. No Pygments lexer found for "none".

```
.. code-block:: none

        KTAP version 1
        1..1
          KTAP version 1
          1..3
            KTAP version 1
            1..1
            # test_1: initializing test_1
            ok 1 test_1
          ok 1 example_test_1
            KTAP version 1
            1..2
            ok 1 test_1 # SKIP test_1 skipped
            ok 2 test_2
          ok 2 example_test_2
            KTAP version 1
            1..3
            ok 1 test_1
            # test_2: FAIL
            not ok 2 test_2
            ok 3 test_3 # SKIP test_3 skipped
          not ok 3 example_test_3
        not ok 1 main_test
```

This output defines the following hierarchy:

A single test called "main_test", which fails, and has three subtests: - "example_test_1", which passes, and has one subtest:

- "test_1", which passes, and outputs the diagnostic message "test_1: initializing test_1"

- "example_test_2", which passes, and has two subtests:
  - "test_1", which is skipped, with the explanation "test_1 skipped"
  - "test_2", which passes

- "example_test_3", which fails, and has three subtests
  - "test_1", which passes
  - "test_2", which outputs the diagnostic line "test_2: FAIL", and fails.
  - "test_3", which is skipped with the explanation "test_3 skipped"

Note that the individual subtests with the same names do not conflict, as they are found in different parent tests. This output also exhibits some sensible rules for "bubbling up" test results: a test fails if any of its subtests fail. Skipped tests do not affect the result of the parent test (though it often makes sense for a test to be marked skipped if _all_ of its subtests have been skipped).

See also:

- The TAP specification: https://testanything.org/tap-version-13-specification.html
- The (stagnant) TAP version 14 specification: https://github.com/TestAnything/Specification/blob/tap-14-specification/specification.md
- The kselftest documentation: Documentation/dev-tools/kselftest.rst
- The KUnit documentation: Documentation/dev-tools/kunit/index.rst