KTAP General Tooling

"KTAP Swiss-Army Knife"

Rae Moar <rmoar@google.com>

Background on KTAP

What is KTAP?

- Test Result Format for Linux Kernel Tests
  - Find specification in kernel docs
- Been upstream since 5.17 in 2021
- Based on simple, text-based TAP
- Components of KTAP specification
  - Version Line - “KTAP version (#)”
  - Test Plan - “1..(#)”
  - Results Lines - “ok/not ok (#) - …”
  - Diagnostic Lines - “# hello_world”
  - Directives - “# SKIP”
  - Nested Tests

```
KTAP version 1
1..1
    KTAP version 1
    1..2
    ok 1 test_1
    # test_2: this test is flaky and super slow
    not ok 2 test_2
not ok 1 test_suite
```

Example of Simple KTAP

Where is KTAP now?

- **Current Status:** KTAP is now widely used with small differences in frameworks
- Major efforts have been made over the past few years to follow specification

KUnit Results

```
KTAP version 1
1..1
    KTAP version 1
    # Subtest: string-stream-test
    # module: string_stream_test
    1..12
    ok 1 string_stream_managed_init_test
    ...
    ok 12 string_stream_performance_test
# string-stream-test: pass:12 fail:0 skip:0 total:12
# Totals: pass:12 fail:0 skip:0 total:12
ok 1 string-stream-test
```

kselftest Results

```
TAP version 13
1..1
# timeout set to 45
# selftests: core: close_range_test
# TAP version 13
# 1..7
# # Starting 7 tests from 1 test cases.
# #  RUN           global.core_close_range ...
# #            OK  global.core_close_range
# ok 1 global.core_close_range
...
# #  RUN           global.close_range_cloexec_unshare_syzbot ...
# #            OK  global.close_range_cloexec_unshare_syzbot
# ok 7 global.close_range_cloexec_unshare_syzbot
# # PASSED: 7 / 7 tests passed.
# # Totals: pass:7 fail:0 xfail:0 xpass:0 skip:0 error:0
ok 1 selftests: core: close_range_test
```

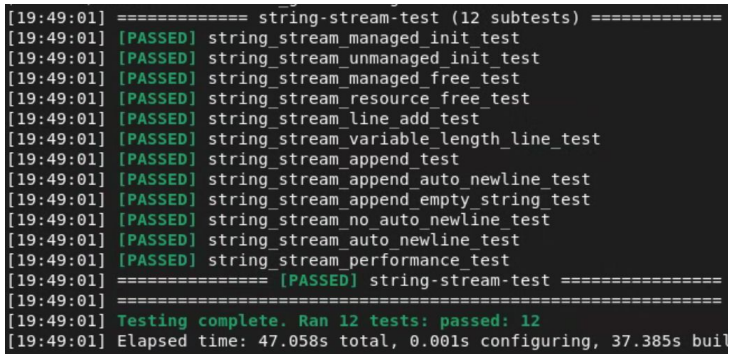
What is the Current Status of KTAP v2?

- KTAP version 2 is ready to go
  - Current list of compiled patches (link)
- KTAP Metadata has been approved for KTAP version 2
  - As discussed at last LPC
  - Framework for outputting supplemental test information (test speed, module name, etc.)
- Method of accepting patches unclear
  - Potential plan to bring accepted patches in via KUnit

Current KTAP Tooling

Current KTAP Tooling

- Parsers
  - Plain KTAP docs can be dense and hard to find in kernel output
  - Simple parsers that check for "ok"/"not ok"
  - KUnit Parser outputs a pretty-print with a human-readable summary
  - Other framework-specific parsers
- Hidden features within current parsers
  - Summary lines
  - Isolate KTAP documents
  - Compile lists of attributes and metadata
- Overall: Framework-specific, not modular



Example of KUnit Parser Results

Discussion on KTAP Tooling

What is our current system doing well?

- Tooling has greatly improved the experience working with KTAP (especially parsers)
- Fulfills framework-specific needs

What can be improved?

- Decentralized so there is redundancy in code
  - Not being shared between frameworks
- Not visible
  - Are people aware of the current available tooling?
- We can do more!
  - Other tools may be useful (converters, splitters), where should these be located?

KTAP General Tooling

KTAP General Tooling

- **Proposal**: Create a new library with common KTAP tooling to be used by multiple frameworks
  - Common parser as well as additional tools
- **Objective**: "Swiss-Army Knife" All Your Favorite Tools in One Place
  - Reduce redundancy
  - Increase visibility of KTAP tooling
  - Reinforce specification
- **Where?**
  - Located at new directory `tools/testing/ktap`
  - Written in python with command-line interface or ability to directly use methods

What can it do?

- Parser
  - To isolate and output KTAP results in a pretty-print human-readable format (based on KUnit current parser)
  - Compliant with current KUnit and kselftest outputs
- Isolator
  - To filter out non-KTAP in output from Kernel log
- Splitter/Combiner
  - To split multiple KTAP documents into individuals or to combine KTAP documents into one
- Summarizer
  - To produce a summary line of results from KTAP documents
- Converters
  - To convert KTAP to another format
  - Currently considering JSON and JUnit XML
  - This could be used to upload results to a CI system

Example of Combiner

```
KTAP version 1
1..1
    KTAP version 1
    1..3
    ok 1 test_1
    ok 2 test_2
    not ok 3 test_3
not ok 1 test_suite_a
[Kernel log]
KTAP version 1
1..1
    KTAP version 1
    1..1
    ok 1 test_1
ok 1 test_suite_b
```

```
KTAP version 1
1..2
    KTAP version 1
    1..3
    ok 1 test_1
    ok 2 test_2
    not ok 3 test_3
not ok 1 test_suite_a
    KTAP version 1
    1..1
    ok 1 test_1
ok 1 test_suite_b
```

Parser

Output of kselftest results passed into current KUnit parser

*with small to tweak to allow for "# " indentation

kselftest Results

```
TAP version 13
1..1
# timeout set to 45
# selftests: core: close_range_test
# TAP version 13
# 1..7
# # Starting 7 tests from 1 test cases.
# #  RUN           global.core_close_range ...
# #            OK  global.core_close_range
# ok 1 global.core_close_range
...
# #  RUN           global.close_range_cloexec_unshare_syzbot ...
# #            OK  global.close_range_cloexec_unshare_syzbot
# ok 7 global.close_range_cloexec_unshare_syzbot
# # PASSED: 7 / 7 tests passed.
# # Totals: pass:7 fail:0 xfail:0 xpass:0 skip:0 error:0
ok 1 selftests: core: close_range_test
```

Parser Output

```
[23:20:57] ============================================================
[23:20:57] ======================== (7 subtests) ========================
[23:20:57] [PASSED] global.core_close_range
[23:20:57] [PASSED] global.close_range_unshare
[23:20:57] [PASSED] global.close_range_unshare_capped
[23:20:57] [PASSED] global.close_range_cloexec
[23:20:57] [PASSED] global.close_range_cloexec_unshare
[23:20:57] [PASSED] global.close_range_cloexec_syzbot
[23:20:57] [PASSED] global.close_range_cloexec_unshare_syzbot
[23:20:57] ======== [PASSED] selftests: core: close_range_test ========
[23:20:57] ============================================================
[23:20:57] Testing complete. Ran 7 tests: passed: 7
```

Discussions on KTAP General Tooling

Pros

- Frameworks can share resources and removes redundancy in tooling
- One library will reinforce one common understanding of the KTAP specification
- Makes KTAP tooling more accessible and visible to kernel developers
- Incentivises new KTAP tooling features by creating a space for it

Cons

- Disincentives framework-specific parsing and other tooling
- Creates a new tooling library to be maintained
- Potentially more disagreement on development decisions (more players to get approval)

Open Questions

Open Questions

- Would this new library be useful? Are people interested in it?
- Specifically, would kselftest be interested in using any of the potential features including the parser?
- Which features are people interested in from those listed and any ideas for additional features?
- Are there changes to the KTAP specification that could improve the transition to using this general tooling?