

Linux Plumbers Conference

Vienna, Austria | September 18-20, 2024



Adding Benchmark Results Support to KTAP/kselftest

Tim Bird, Sony Electronics
September 20 2024

Abstract

Benchmark test results are difficult to interpret in an automated fashion. They often require human interpretation to detect regressions because they depend on a number of variables, including configuration, cpu count, processor speed, storage speed, memory size, and other factors. Tim proposes a new system for managing benchmark data and interpretation in kselftest. It consists of 3 parts: 1) adding syntax to KTAP to support a consistent format for benchmark values in KTAP/kselftest test output, 2) the use of a set of criteria, external to the test itself, for interpreting benchmark result values, and 3) an automated tool to determine and set appropriate reference values to use in the test result criteria. A prototype system will be demonstrated, that supports converting benchmark values into KTAP "ok" and "not ok" results, consumable by humans and automated tools (such as CI systems). This system is intended to enable the detection of regressions in benchmark outputs, using appropriate threshold values that are customizable (in an automated fashion) by a tester for their own configuration and hardware.



Outline

- How are benchmarks used today?
- Problems with test automation for benchmarks
- Proposed solution
- Demonstration
- Issues and optional features (for discussion)



Benchmark use today

- A human knows what the benchmark value should be
 - And what variance is allowed
- The tester runs the benchmark
- The tester examines the results manually, and
 - Determines if the performance has regressed or not, on *their* system
- The tester reports a bug
- If something changes (e.g. software, hardware, configuration, settings, or workload), then
 - the tester relearns new benchmark reference values and variances



Rationale for supporting benchmark data

- Supporting benchmark data supports testing a whole range of code attributes, besides simple pass/fail
 - Can test performance (speed, duration, latency, power usage, CPU utilization, cache hits, etc.), in addition to functionality
- But...
 - Have to convert numeric data into pass/fail, for automatic detection of test result



Introducing Reference values

- Reference value = value used to determine the final result of a test (pass/fail)
 - Almost always represents a threshold for a value obtained from a test
 - May also include allowable variance (discussed later)
- Reference values may be kept separate from the test
 - But this raises several issues



Keeping reference values separate

- Should not encode the reference value into the test itself
- This makes the test specific to one situation
 - Can't share the test
 - Want to use different reference values for different machines and test environments
 - Want to report regressions on hardware that developers don't have
- As performance changes, you want to establish new baselines for what constitutes a regression in behavior
 - Tests themselves would have to be continually updated



Issues with reference values

- How are reference values stored?
 - Where are they stored? What format is used?
- How are reference values used?
 - How are reference values mapped to testcase outcomes?
 - How is the correct set of reference values used?
 - How are ref. values files correlated with different test environments?
- How are they generated and updated?



Proposal

New system for managing benchmark data and interpretation in kselftest

It consist of these parts:

1. Syntax for expressing benchmark values in KTAP/kselftest test output
2. A collection of reference values, that can be used with benchmark results
3. A set of criteria, for interpreting result values
4. A tool to convert numeric result values to pass/fail (ok/not ok) test results, based on the reference values and specified criteria
5. (optional) a tool to manage reference values (generate and update them)



Benchmark system objects

- Test result files in KTAP format, with:
 - test result numeric values (usually with units)
- Reference values files
 - reference values (with units)
 - (optional) meta-data describing applicable test parameters or environment
- Test criteria files
 - Has the mapping between (value names and reference values) to test result outcomes
 - Has the value id, direction of comparison, units, and allowed deltas (variances)
- Results evaluation tool
 - A tool or library to convert test result values to pass/fail (ok/not ok) test results



KTAP syntax extensions

- KTAP test value output
- KTAP undetermined testcase result



KTAP syntax extensions (cont.)

- KTAP test value output

- Syntax: "value {value_id} = {value} {units} [# {comment}]"
- Example:

```
value io_4k_seq_read_speed = 78 MB/s
```

- KTAP undetermined testcase result

- Used before outcome is determined by criteria processor
- Use the word "unknown" in place of "ok" or "not ok"
- Syntax: "unknown {testcase_num} {testcase description} [# SKIP or diagnostic data]"

Example:

```
unknown 12 io_4k_seq_read
```


Reference values declarations

- Reference values

- Can be in a standalone file, or embedded in criteria lines in the criteria file
- Syntax: same as the KTAP value line:
 - ie. "value {value_id} = {value} {units} [# {comment}]"
- Example: value io_4k_seq_read_speed = 90 MB/s

- Reference values files:

- Consists of a list (one or more) reference values
 - Each value declared with same syntax as a test results file
- Filename may reflect test name, machine identifier, or environment
- In Future: May also include meta-data about the environment for which the values are applicable:
 - e.g. kernel version, related configs, machine, hardware, etc.



Test criteria

- Test criteria = rule for evaluating test values to generate test outcomes
 - Each criteria references a value identifier, a reference value and testcase
 - Is similar to a conditional statement
 - Syntax (all on a single line):
 - "if {value_name} {operator} {ref_value} {units} [+ - {variance}] then {testcase_name}={ok|not ok}"
 - example: if io_4k_write_speed < 90 MB/s then io_4k_write=not ok
 - example with allowed variance:
 - if io_4k_write_speed < 90 MB/s +- 5% then io_4k_write=not ok



Results evaluation tool

- Is a tool and/or library to convert values to pass/fail (ok/not ok) test results, based on the result values, test criteria, and reference values
- Two modes of operation:
 - During-test mode
 - Post-test mode



Results eval tool – "During-test" mode

- Operation:
 - Test calls tool with an individual test value
 - Tool reads the reference values and test criteria and performs the evaluation
 - Test receives the ok/not ok outcome
 - Test records outcome in the test output during test execution
- Pros and cons:
 - Good: allows test to provide a value, and query the criteria and reference values, to get an outcome to put into the results during the test
 - Bad: requires the test to call the system (including having references to the reference values file and criteria files)
 - ie – test has to know more about its environment, file locations, etc.



Results Processing Flow (during test)

- With ref value (in dd-io-ref-bvalues-rpi4.txt) of:
 - io_4k_seq_read_speed = 90 MB/s
- and criteria rule (in dd-io-criteria.txt):
 - if io_4k_seq_read_speed < \$ref_value then io_4k_seq_read = not ok

```
$ process-results.py -e "io_4k_seq_read_speed = 78 MB/s" -n 12 \  
  -r dd-io-ref-values-rpi4.txt -c dd-io-criteria.txt  
not ok 12 io_4k_seq_read
```



Results Processing tool – "Post-test" mode

- Operation:
 - test: produces output with 'unknown' testcase outcomes
 - process-results.py tool:
 - Reads the test output (with 'value' test result lines)
 - Reads the reference values and test criteria
 - Writes new test output, with updated ok/not ok testcase lines
- Pros and cons:
 - Good: test only has to gather data, not evaluate outcome
 - Test doesn't have to know about reference values and criteria
 - Bad: output may have 'unknown' outcomes temporarily
 - Results are not known immediately (for example, test outcome summaries may be wrong until the test output is processed)



Evaluate criteria flow

- With ref value: `io_4k_seq_read_speed = 90 MB/s`
- and criteria rule:
 - if `io_4k_seq_read_speed < $ref_value` then `io_4k_seq_read = not ok`

```
...  
value io_4k_seq_read_speed = 78 MB/s  
unknown 12 io_4k_seq_read  
...
```

```
...  
value io_4k_seq_read_speed = 78 MB/s  
not ok 12 io_4k_seq_read  
...
```

Evaluate criteria flow

- With ref value: `io_4k_seq_read_speed = 90 MB/s`
- and criteria rule:
 - if `io_4k_seq_read_speed < $ref_value` then `io_4k_seq_read = not ok`

```
...  
value io_4k_seq_read_speed = 78 MB/s  
unknown 12 io_4k_seq_read  
...
```

```
...  
value io_4k_seq_read_speed = 78 MB/s  
not ok 12 io_4k_seq_read  
...
```

Demonstration

- This is a prototype system, that supports converting benchmark values into KTAP "ok" and "not ok" results
- Results are consumable by humans and automated tools (such as CI systems).
- System enables the detection of regressions in benchmark outputs
- Threshold values (reference values) are customizable by a tester for their own configuration and hardware



Demonstration

- dd-io test
 - initcall-duration test
 - spdx-headers test
 - boot-marker test
-
- demo video at: <https://youtu.be/4OoORXsZ0bs>



More criteria rules info

- Allowed variance
- Rule indirection
- Rule wildcarding



Allowed variance

- Allowed variance indicates to the criteria processor that the threshold is "fuzzy"
- Is an optional part of a criteria rule
- Expressed as "+- {percentage}" or "+- {value} {units}"
 - Examples:
 - if io_4k_write_speed < 90 MB/s +- 5% then io_4k_write=not ok
 - if arch_boot_duration > 100 ms +- 10 ms then arch_boot_time=not ok



Criteria rule wildcarding

- Can use wildcards to express multiple rules in a single line:
 - if `io_${size}_write_speed < $ref_value` then `io_${size}_write=not ok`
- `%{var}` is replaced on both sides (in the `value_id` and `test_id`) of the conditional expression, when it matches a reference value name
 - e.g. Can express rules for '1k', '4k', '16k' tests, etc. with a single line
 - This rule works for:
 - `io_1k_write_speed`
 - `io_4k_write_speed`
 - `io_16k_write_speed`



Criteria indirection

- Criteria can have in-place reference values
 - if `io_4k_write_speed < 90 MB/s` then `io_4k_write=not ok`
- Or use a variable that is replaced:
 - if `io_4k_write_speed < $ref_value` then `io_4k_write=not ok`
- During evaluation, `$ref_value` is replaced with the value from the ref. value file



Issues

- Need to resolve where reference value files live, and are maintained
 - Some reference files may be able to live upstream, but likely an out-of-tree repository of reference value files is needed
- Need to determine if criteria files can be upstream (probably).
 - In the case of in-test criteria processing, maybe the criteria can be safely embedded in the test itself (?)
 - Some tests may only need one rule (with wildcarding)
- May need more use cases, before pinning down the syntax and tools and making it into a standard
 - Need to write some more tests that use this, to determine how best to utilize these features



Issues (cont.)

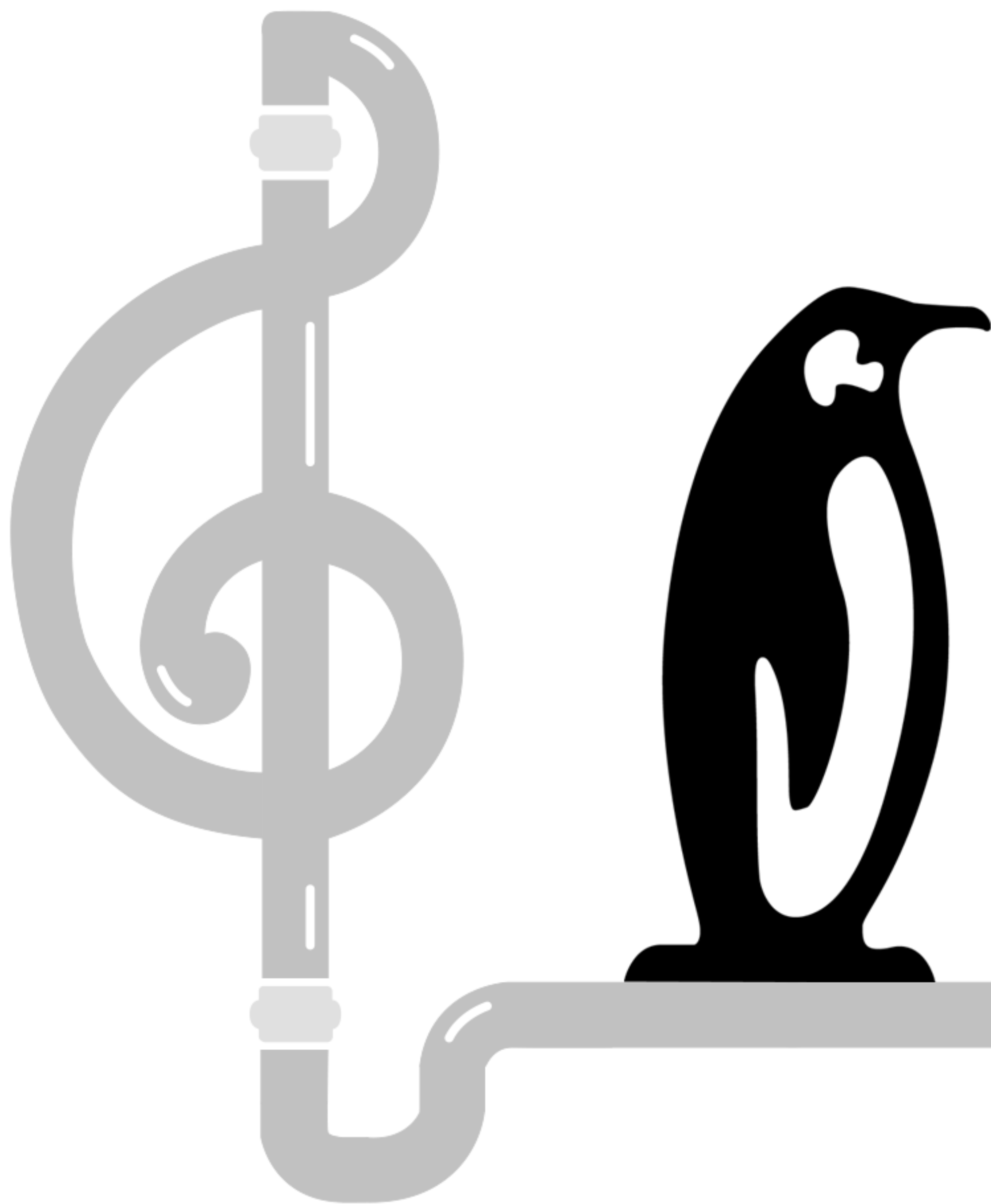
- How to automatically select the appropriate set of reference values?
 - What meta-data should be stored with reference values?
 - How to match reference values to current test?
 - Currently, I use a user-provided string (like a machine name)
 - e.g. rpi4, beaglebone, minnowboard
- Where are test results stored, to enable reference value updates?
 - Either automated updates, or manual with tool for assistance



Proposal Core Principles (Conclusions)

- Evaluating benchmark data supports testing a whole range of code attributes, besides simple pass/fail, for numeric values that differ widely on different systems
- Goal is to automate steps that previously required a human
 - Standardize format of numeric benchmark data
 - Support reference values separate from tests
 - Create ref value sets that are specific for different test environments
 - Support Criteria rules are separate from the reference values





Linux Plumbers Conference

Vienna, Austria | September 18-20, 2024

Additional Material

- Examples
- Miscellaneous notes
- process-results.py
- Outcome reason
- results-stats.py



Examples – reference values

```
$ cat dd-io-ref-values-timdesk.txt
```

```
value root_seq_read_1024MB_at_4096bs_speed=1.4 GB/s
```

```
value root_seq_read_1024MB_at_16384bs_speed=1.1 GB/s
```

```
value root_seq_read_1024MB_at_65536bs_speed=1.2 GB/s
```

```
value root_seq_read_1024MB_at_100000bs_speed=1.2 GB/s
```

```
value backup_seq_read_102MB_at_4096bs_speed=55.0 MB/s
```

```
value backup_seq_read_102MB_at_16384bs_speed=57.9 MB/s
```

```
value backup_seq_read_102MB_at_65536bs_speed=57.8 MB/s
```

```
value backup_seq_read_102MB_at_100000bs_speed=58.0 MB/s
```


Examples – criteria rules

```
$ cat dd-io-criteria.txt
```

```
if ${testcase}_speed < $ref_value +- 5% then ${testcase}=not ok
```

```
$ cat initcall-duration-criteria.txt
```

```
if ${func}_duration > $ref_value +- 10% then ${func}=not ok
```

```
$ cat spdx-stuff/spdx-missing-criteria.txt
```

```
# ipc should not have any files missing spdx headers!!
```

```
if ipc_spdx_missing_count > 0 files then ipc_spdx_status=not ok
```

```
if ${dir}_spdx_missing_count > $ref_value then ${dir}_spdx_status=not ok
```

Miscellaneous notes

- 'variance' is not being used here in the mathematical sense of the term (which is the square of the std. deviation)
- Because the ref values have the same format as values, you can use a results file as your reference value file
 - eg: symlink from spdx-test-ref-values.txt to spdx-test-results-2024-08-07.txt (a previous results file)
- You could also use this system to encode a set of acceptable testcase failures (unrelated to benchmarks)
 - By treating outcome counts as benchmark data, or by directly referencing non-numeric conditions to convert testcases to 'ok'
 - example: `if io_speed_not_ok_count < 4 then io_speed=ok`
 - example: `if driver_boot_duration = not ok then driver_boot=ok`




process-results.py

- input/output
 - -i <input_file> or -e <value expr>
 - -o <output_file>
 - -n <testcase_num>
 - file selection:
 - -r <ref_value_file>
 - -c <criteria file>
- or
- -d <artifact_dir>
 - -m <match_hint>



Outcome reason

- It's handy for humans to see the reason a test failed
 - With reference values separate from test, it may not be obvious from the results alone why the test failed
 - e.g. – what was the threshold for this value on this machine
 - May want to add 'outcome reason' string to test results output
- Example:



```
value io_4k_seq_read_speed = 78 MB/s
# reason: io_4k_seq_read_speed < ref. value of 90 MB/s
not ok 12 io_4k_seq_read
```



results-stats.py

- Is a tool to process past results
 - Purpose is to help determine reference values and criteria rules
- Can read KTAP results from many test runs, and output statistics
 - min, max, avg, standard deviation, zscore
- Can omit outliers (which otherwise skew results)
- Can output verbose data, or just new a reference value file
 - Selecting either min, max or avg values



Reference value update system - details

- Basic idea for updating reference values:
 - What do humans do?
 - Use knowledge of system and past results to establish threshold values for benchmark data
 - Use past results to establish the new baseline values
- Could be as simple as:
 - `$ grep ^value mytest_last_output.txt >mytest-ref-values.txt`
 - Ratchet test = test where any improvement establishes a new baseline for regressions (so just collect current values and call those reference values)
- Want something a bit more formal, that can handle more complexity

More details

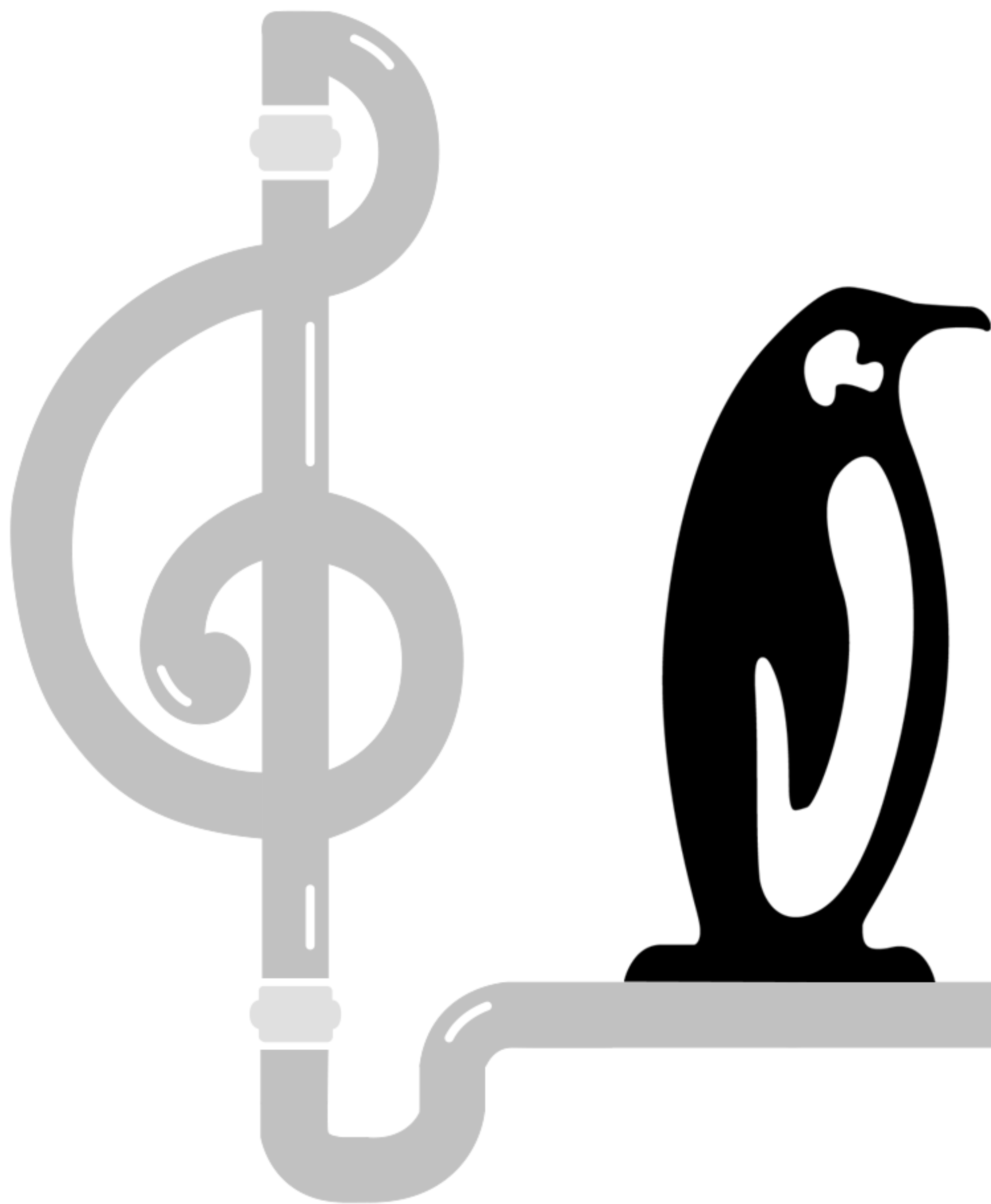
- Some tests don't output results in standardized units
 - Results units may not match the ref value units
 - `process-criteria.py` and `results-stats.py` can do unit conversions



A stylized illustration of plumbing pipes in shades of gray. A vertical pipe on the left has a circular loop. A horizontal pipe extends from the bottom left, with a small black penguin perched on top of it.

End of Additional Material

LINUX PLUMBERS CONFERENCE | Vienna, Austria
Sept. 18-20, 2024



Linux Plumbers Conference

Vienna, Austria | September 18-20, 2024