

KCOV: code coverage

- KCOV keeps track of code run during execution
- CONFIG_ARCH_HAS_KCOV=y
- CONFIG_KCOV
- My tool to do a quick check of individual test coverage
 - `kcov_collect [-c command] | addr2line -afi -e vmlinux`

GCOV: code coverage

- GCOV keeps track of code run during execution
- CONFIG_GCOV_KERNEL
- CONFIG_ARCH_HAS_GCOV_PROFILE_ALL=y
- Generates reports
- Show what code ran, and what code did not

GCOV content from Brendan Higgins, KUnit maintainer

Code Coverage **IS**

- A great way to quickly find what code **IS** tested and what code **IS NOT** tested.
- Allows you to quickly identify problem areas, and drill down into a report.
- Identify missed branches.
- Identify unused code.

Code Coverage **IS NOT**

- Code coverage is a tool, not a panacea
- Code coverage helps quickly identify and prioritize problem areas
- Code coverage summaries **do not tell you whether your testing is good or bad**
 - What is the right amount of line coverage?
 - 50%?
 - 70%?
 - 90%?
 - 100%?

What's the right coverage?

- How do we measure coverage?
 - % of lines?
 - % of functions?
 - % of branches?
- What about absolute vs incremental?

What's the right coverage?

- Absolute coverage:
 - What you expect.
 - Everything in the entire codebase at some point in time.
- Incremental coverage:
 - The test coverage of the Δ in a change

Absolute vs. Incremental Coverage

- Incremental Coverage is usually more interesting
 - It's much easier to achieve high incremental coverage immediately
 - Helps prioritize code more likely to be buggy
 - More actionable by developers
 - Code that has not changed in a long time is *more* likely to be fine

Absolute vs. Incremental Coverage

- Absolute Coverage is still important, just less important
 - Old code may be less likely to contain bugs...
 - ...but it's often worse when it does
- Often easier for comparing coverage health of subsystems
- Easier to compute

Discussion – Q&A

Thank you for your time