Getting Started With New KernelCI CLI Tools

Automating Linux Kernel Testing and Validation

Arisu Tachibana

- KernelCI Infra WG member
- kci-dev creator / maintainer
- Gentoo Kernel leader
- CIP testing member
- Cybertrust Japan Co., Ltd.

What you'll learn in 35 minutes + QA

- How KernelCI fits into everyday kernel work
- How `kci-dev` keeps you in the terminal instead of dashboards
- A concrete workflow you can copy for your own trees
- Where we're going next with `kci-dev` + `kci-deploy`

Who this talk is for

- Kernel developers who *don’t* have time to babysit dashboards
- Maintainers juggling multiple trees and branches
- CI / lab folks who want developers to actually look at results

Motivation: Why Kernel QA Is Hard

- Many trees, branches and configurations
- Multiple architectures, boards and toolchains
- CI dashboards are powerful but not ergonomic
- Developers still click UIs or write ad-hoc scripts
- Command-line tools keep context in the terminal

KernelCI in One Slide

- Upstream, open testing for the Linux kernel
- Builds, boots and tests across distributed labs
- Huge amount of data, but interaction is:
  - Mostly via web UI
  - Raw REST APIs
  - In the last 7 days we did 7108 kernel builds
- Need something ergonomic in the terminal

Real-World Context #1: gentoo-sources & CI

- For `gentoo-sources` I already use:
  - `buildbot-try` to submit kernel builds/tests
  - `gkernelci` integration around Buildbot
- Nice properties of this workflow:
  - I stay in the terminal
  - CI is scriptable from my normal dev environment
- This experience is the main inspiration for **kci-dev**
  - “What if KernelCI felt like buildbot-try, but for any tree?”

From buildbot-try to kci-dev

- What I like about `buildbot-try`:
  - One command: “Here is my change, please test it”
  - Results come back to the console
- KernelCI already has:
  - Distributed labs and lots of coverage
  - Dashboards and APIs
- Missing piece:
  - A developer-first CLI that feels like `buildbot-try`
- **kci-dev** is that missing piece:
  - Unified CLI for querying KernelCI jobs and tests
  - Designed from real needs on `gentoo-sources`
  - But usable for *any* KernelCI-managed tree

Real-World Context #2: CIP & SLTS

- **Why CIP uses kci-dev**
  - SLTS kernels live for a decade: regressions and security issues must be caught early and tracked over years.
  - `kci-dev` lets CIP trigger KernelCI jobs on SLTS trees directly from the terminal and reuse the same KernelCI infrastructure that already publishes SLTS test results.

```
$ kci-dev checkout --giturl https://git.kernel.org/pub/scm/linux/kernel/git/cip/linux-cip.git --branch linux-6.1.y-cip --tipoftree
No job filter defined. All jobs will be triggered!
Retrieving latest commit on tree: https://git.kernel.org/pub/scm/linux/kernel/git/cip/linux-cip.git branch: linux-6.1.y-cip
Commit to checkout: 81cb6c23dfa8aae5f1aa5f7a1f9518946503fc15
OK
treeid: ad4b54a7c4671982ae805d014fb43e6ceb7bd27b4f94be5ced0e3dac5f419127
checkout_nodeid: 6937035a93bfb52307cd26b2
```

Real-World Context #3: ChromiumOS tree

- using kci-dev for code coverage statistics

```
$ kci-dev maestro coverage
- Tree/branch: chromiumos/chromeos-6.12
  Commit: 1d022843a344dd50b57600abc3e5b3f14f8a463f
  Build: https://staging.kernelci.org:9000/viewer?node_id=68b83f8a18776824c891f55c
  Function coverage: 19.3%
  Line coverage: 15.5%
  Coverage report: https://files-staging.kernelci.org/coverage-report-x86-68bac05fa6a30c5a1ec57f88/coverage-68b83f8a18776824c891f55c.html
  Coverage logs: https://files-staging.kernelci.org/coverage-report-x86-68bac05fa6a30c5a1ec57f88/log.txt
```

Other Real-World Context #4: kernelci-pipeline

- using kci-dev in a cronjob for finding builds/boots missing on dashboard
- send results by email

```
$ kci-dev maestro validate
```

code is here: https://github.com/kernelci/kernelci-pipeline/pull/1320

How kci-dev fits into KernelCI

kci-dev commands summary

kci-dev workflow example

KernelCI CLI: bringing CI into your terminal

- **kci-dev**
  - Developer-focused CLI to interact with KernelCI dashboards and Maestro
  - Installable from PyPI (`pip install kci-dev`) or development snapshots via poetry
  - Ships `results` (dashboard), Maestro, and validation commands in one binary
  - Completions bundled for bash, zsh, and fish
- **kci-deploy** *(work in progress)*
  - Tool for deploying local / internal KernelCI maestro stacks
- Goal: make KernelCI a first-class tool in your terminal

Install & Configure Fast

- `virtualenv .venv && source .venv/bin/activate`
- `pip install kci-dev`
- `kci-dev config` scaffolds `~/.config/kci-dev/kci-dev.toml` for Maestro auth
- Results-only workflows work **without** a config file
- Request API tokens via the KernelCI GitHub issue template when you need Maestro access

Shell Completions Included

- Bash: `source /path/to/kci-dev/completions/kci-dev-completion.bash`
- Zsh: add completions directory to `$fpath` and run `compinit`
- Fish: copy `completions/kci-dev.fish` to `~/.config/fish/completions/`
- Keep completions in sync with the installed kci-dev version

kci-dev: What You Can Do Today

Dashboard (`results`) commands

```
kci-dev results summary --giturl <git url> --branch <branch> --history
kci-dev results compare --giturl <git url> --branch <branch> <older> <newer>
kci-dev results hardware summary --name <vendor,board> --origin maestro --json
kci-dev results tests --giturl <git url> --branch <branch> --commit <sha> --filter filter.yaml
```

Maestro commands (trigger commands need config/token)

```
kci-dev checkout --branch <branch> --giturl <git url> --commit <sha> --job-filter <regex> --watch
kci-dev testretry --nodeid <node-id>
kci-dev watch --nodeid <node-id> --job-filter <regex>
kci-dev maestro validate builds --all-checkouts --days 7 --table-output
kci-dev maestro results --nodeid <node-id> --json
```

kci-dev: Everyday workflow (story)

1. **Start your day in the terminal**
   - `kci-dev results summary --history`
   - `kci-dev results hardware summary` for checking your boards
2. **Spot something suspicious**
   - `kci-dev results boots|tests --status fail --download-logs`
   - Jump straight into logs without touching the browser
3. **Decide: bug or infra?**
   - `kci-dev results compare` across commits to see real regressions
   - `kci-dev testretry` / `kci-dev checkout --watch` for flaky or infra cases
4. **Capture it for automation**
   - Turn the commands you just ran into a script or CI job

kci-dev: Quality-of-Life Details

- Rich output modes: table, JSON, quiet for scripts
- `--history` summaries with pass/fail/inconclusive color coding
- `compare` highlights regressions with dashboard/log links
- `results hardware` views boards + per-board summaries
- Filters: `--filter` YAML for hardware/tests, `--arch`, `--status`
- Profiles via `--instance` / `--settings` to switch endpoints
- Designed for piping into `jq`, `fzf`, notebooks

Under the hood: results vs Maestro

- `kci-dev` talks to two logical APIs:
  - **results (dashboard)**
    - Read-only view of builds, boots, and tests
    - Great for: “What is the status of my tree today?”
  - **Maestro**
    - Lower-level jobs and nodes
    - Powers `checkout`, `watch`, `testretry`, `maestro results`, `maestro validate`
- Same config file (`kci-dev.toml`) selects:
  - Base URLs for each instance
  - API tokens (only needed for Maestro / write-like actions)
- Typical journey:
  i. Start with **results-only** (no config, no token)
  ii. Add Maestro + tokens when you’re ready to act on CI

Adapting kci-dev to different roles

- **Maintainer of a busy tree**
  - Morning:
    - `kci-dev results summary --history`
    - `kci-dev results hardware summary` for your key boards
  - Before release:
    - `kci-dev results compare` across the last few commits
    - Scripted checks in your release pipeline
- **Feature / patch series developer**
  - While hacking:
    - `kci-dev results tests --commit <sha>` to confirm fixes
    - `kci-dev results boots --status fail --download-logs`
  - When CI is noisy:
    - `kci-dev testretry --nodeid <id>`
    - `kci-dev checkout --watch` on interesting jobs

Adapting kci-dev to different roles #2

- **Lab / CI person**
  - Keep infra honest:
    - `kci-dev maestro validate builds --all-checkouts --days 7 --table-output`
  - Handle flaky boards:
    - `kci-dev testretry` recipes instead of ad-hoc scripts
  - Reporting:
    - Cron jobs that send Matrix / mail summaries using `--json`

Dashboard Endpoint Reality Check

- `kci-dev results` currently pulls a multi-megabyte dashboard payload
- Initial download may take a few seconds; caching work is in progress
- Known upstream issue is tracked in the KernelCI dashboard repo

kci-deploy: For Lab Owners

- Simplifies standing up a Maestro stack
- Encodes best practices for networking and storage
- Shares the same UX patterns as kci-dev
- Early previews welcome: help shape the roadmap!

Automation Patterns

- **Pre-submit checks**: gate merges on KernelCI signal
- **Nightly reports**: email/Matrix summaries via cron
- **Release readiness**: track blockers for RCs
- **Local sanity tests**: run focused boards before shipping

```
    "name": "checkout",
    "owner": "staging.kernelci.org",
    "parent": null,
    "path": [
        "checkout"
```

```
    "inconclusive": 674
  }
}
arisu@gyarutoo ~/kcidev_demo $ kci-dev results hardware summary   --name mediat
ek,mt8195   --origin maestro   --json
```

Key takeaways

- KernelCI already has the data — kci-dev makes it feel local to your terminal
- You can start today with read-only results — no tokens needed
- Maestro-powered commands let you move from “observing” to “acting” on CI
- kci-deploy is the next step for making this flow the default in more trees

Roadmap & Collaboration #1

- Deeper git integration (auto-pick branch/commit)
- Better diffing between runs
- Inline links back to dashboards
- kci-deploy installer
- Looking for testers, lab partners, and feedback

Roadmap & Collaboration #2

- Caching for dashboard
- Document bisection command
- Triggering builds with local patches (uploaded somewhere)
- More distro packages
- Adding user agent information to kci-dev for prioritize kci-dev query

Documentation & Updates

- Docs: https://kci.dev
- Source: https://github.com/kernelci/kci-dev
- Token requests and issues: KernelCI GitHub templates

Getting Started After This Talk

- Try **kci-dev** with your tree this week
- Share your top 3 pain points in Kernel QA
- Join the KernelCI community calls / Discord
- Contribute docs, plugins, and issue reports

Thank You!

Slides: https://aliceinwire.github.io/presentations/
X: https://x.com/arisu_gyaru
Instagram: https://www.instagram.com/gyaru_arisu/
GitHub: https://github.com/aliceinwire

Questions welcome!
Will help shape the future of kci-dev

Asciinema links

kci-dev maestro

https://asciinema.org/a/760568

kci-dev KCIDB

https://asciinema.org/a/760572

Demo: from dashboard noise to a focused terminal workflow #1

```
# Configure once
virtualenv .venv && source .venv/bin/activate
pip install kci-dev
kci-dev config     # writes ~/.config/kci-dev/kci-dev.toml

# Morning health check
kci-dev results summary \
  --giturl https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git \
  --branch master \
  --history

kci-dev results hardware summary \
  --name mediatek,mt8195 \
  --origin maestro \
  --json
```

Demo: from dashboard noise to a focused terminal workflow #2

```
# Investigate a failure
kci-dev results boot --id maestro:<boot-node-id> --download-logs | less

# Retry if needed
kci-dev testretry --nodeid <node-id>

# Look at raw Maestro nodes when needed
kci-dev maestro results --nodeid <node-id> --json

# (Optional) validate that dashboard & Maestro agree for the last week
kci-dev maestro validate builds --all-checkouts --days 7 --table-output
```