THE LINUX FOUNDATION
OPEN SOURCE SUMMIT
KOREA

서울

How to Make AI-Assisted OSS Contributions Review-Ready

Jaewoo Choi

About the Speaker

By Day

DevOps Engineer at Hyundai AutoEver

HYUNDAI AutoEver

After Work / Weekends

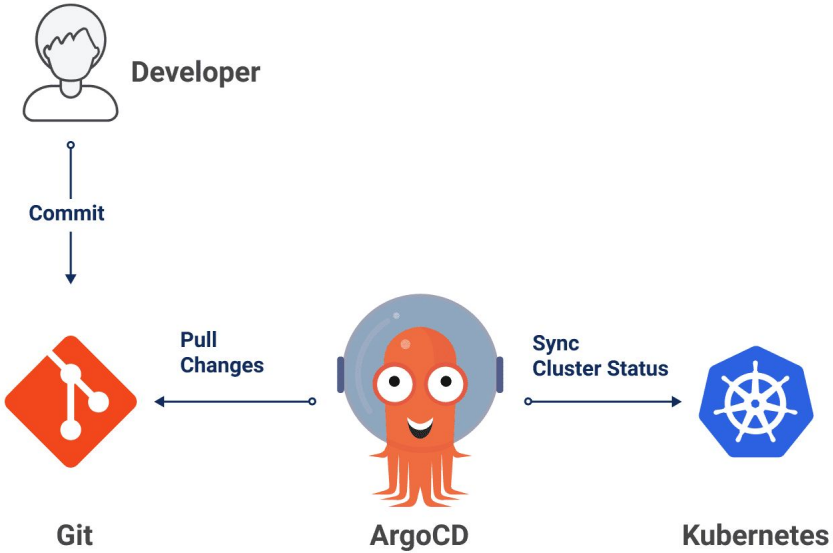
Argo CD as a maintainer

both a user and a maintainer

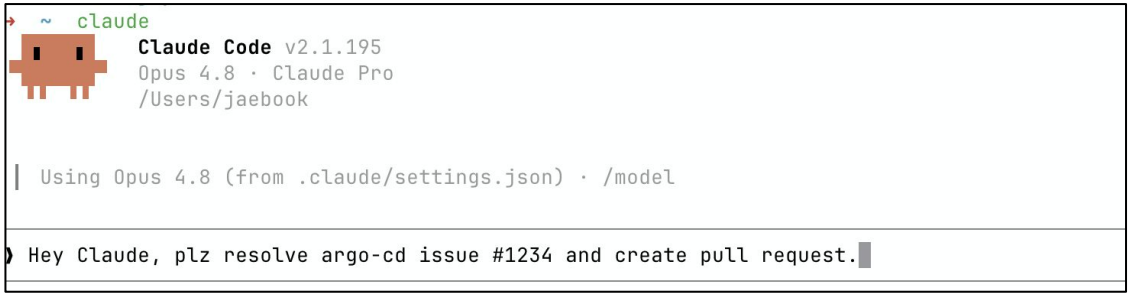
of open source

What Is Argo CD?

- CNCF Graduated Project
- declarative, GitOps
- CD tool for Kubernetes
- most widely used
- battle-tested for 7+ years



Will This PR Actually Land?



- **Issue number**
- **Auto mode on**
- **please.**

10 minutes later…

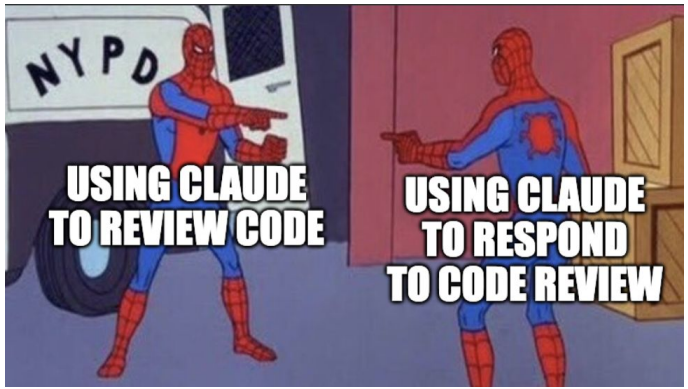
- Failing CI
- No DCO Sign-Off

Test looks fine?

but the fix isn't
really grounded in the project

The Question Isn't Whether You Used AI — It's How

- Maintainers use LLMs too
- Contributors also use LLMs
- First-time Contributors also use LLMs too
- Everyone knows each other uses LLMs



The Difference is

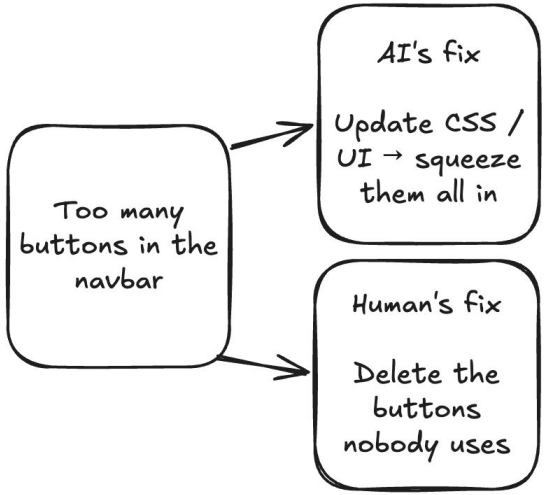
How did you use it

The Question Isn't Whether You Used AI — It's How

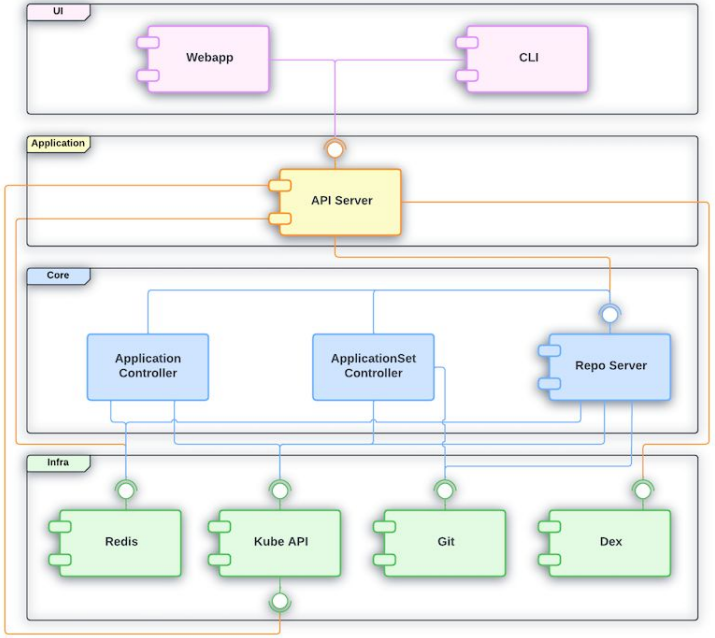
- Long-lived OSS projects
  still keep a human in the loop.
- Maintainers verify changes
  in their own environment
  - They don't just say "CI passed, let's merge it."
- Even AI-native projects usually
  have some
  kind of validation pipeline in place

Code Carries Context That AI Doesn't See

- AI keeps getting better,
  but Mature codebases carry history.
- Every project also has a direction and
  design philosophy that changes must respect.
- The goal isn't the best code
  — it's code that fits this project.



A Fix Doesn't Always Stay in a Single Codebase



- Not always one codebase
- Server · CLI · UI · Controller · Docs…
- Some parts outside the main repo
  -> upstream first
- Shared across sub-projects
  -> can ripple outward

Case 1: The Happy Path

- Not every PR is complex — many follow the happy path
- Issue → reproduce → fix → PR → review → merge
- But even a typo/docs fix can affect other pages
- Still verify: nothing broken, no hidden text
- Simple ≠ skip verification

Case 2: The Sad Path

- Now the complex one — the vendor-managed libraries
- Fix upstream first → then bump the version back
- One change can affect other projects too
- Can't just merge → coordinate + check downstream
- AI misses the full dependency graph → verify with agent + reviewers

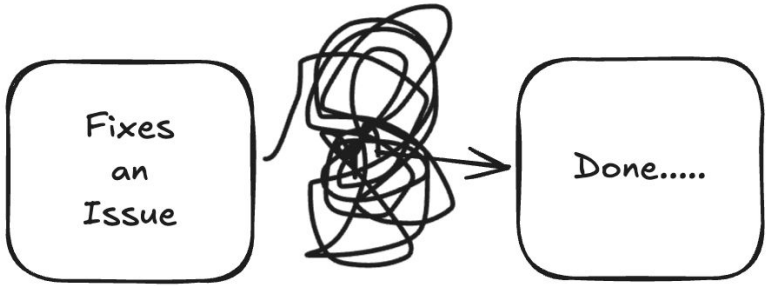
1. Upgrade argo-ui to React 18 (Upgrade Guide here)
2. Upgrade argo-cd to React 18
3. Test Extensively to ensure no regressions
4. Upgrade Argo-ui to React 19 (guide here)
5. Upgrade Argo-cd to React 19

A few weeks later…

argo-ui: argoproj/argo-ui#614
argo-cd: #27091

Case 3: The Painful Path

- Bug fixed in the vendor library on main, but older releases still break
- Backport ≠ bump latest commit, code has diverged
- Need the Release process + branch structure (not in the README)
  - Cherry-pick in the vendor repo
  - Create a branch or tag
  - Bump into the older release branch
- Blocked without permissions → ask maintainers with a clear proposal

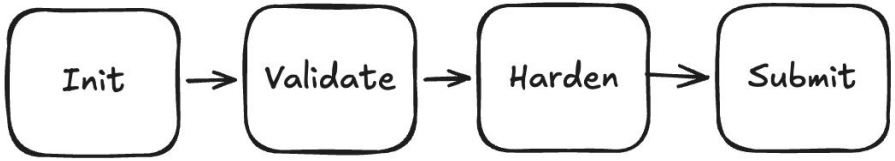


The Framework: 4 Stages

- Here's the framework. 4 stages workflow.
- AI gets better every month, so this isn't the perfect workflow forever.
  - February: Sonnet 4.6
  - April: Opus 4.7
  - May: Opus 4.8
  - June: Fable 5
  - July: Sonnet 5
- Adapt it to your own project and your own style.

The Framework: 4 Stages

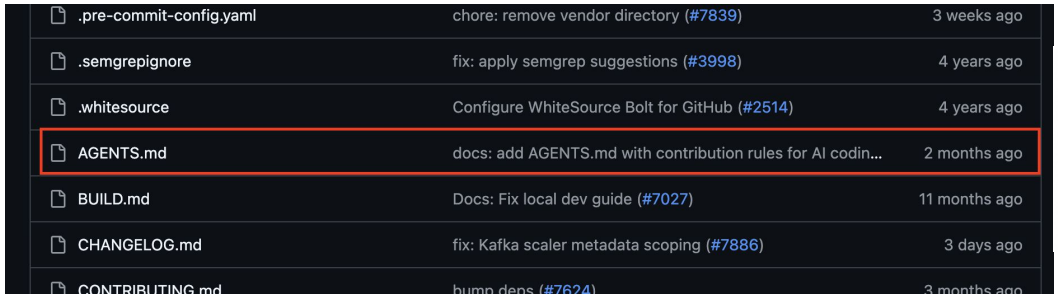
- The 4-stage workflow



- But one rule holds across all four stages
  - At every step, a human checks that it actually worked.
- If something looks wrong → stop there. Debug it.
- Don't just keep going and hope it'll be fine later.

The Framework: 4 Stages

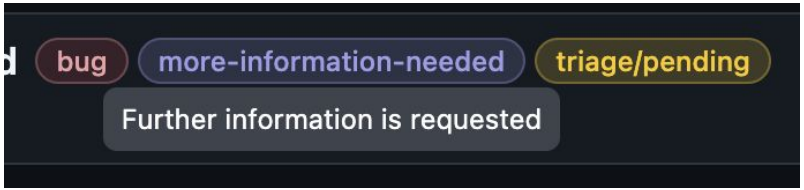
- Before we dive into step 1 — a quick note.
- LLM-assisted contributions are so common now that many projects already ship an AGENTS.md and CONTRIBUTING.md.
- So you can point your LLM at those docs and let it help with the basics.



Stage 1: Init

- Stage 1 is simple.
- Before you write even one line of code
  you need a working local environment.
- Build it, run it, confirm it works — then start.
- No environment, no contribution.
- e.g. Argo CD
  - two ways to set up a local test environment
  - docs for testing the docs site and CLI too

Stage 2: Validate

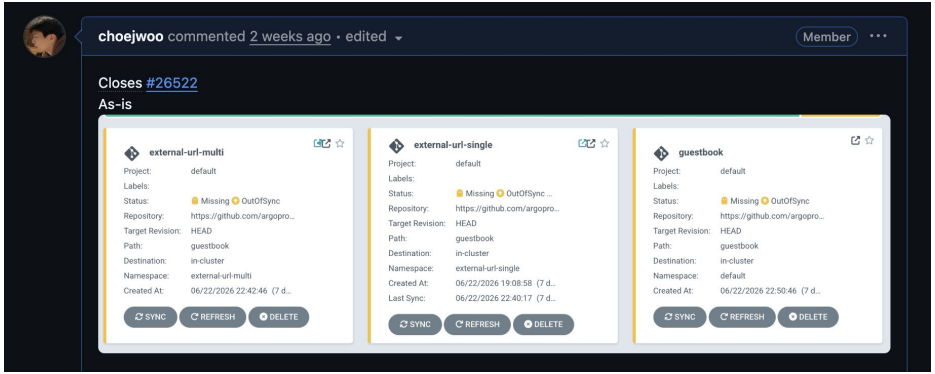


- Next up: validating the problem.
- Very often the issue report is incomplete — missing details, or an edge case.
- Don't just throw it at AI → talk to the reporter first.
- Confirm
  - reproduce it locally
  - the expected behavior
- maybe it's an EOL version, or already fixed in a newer release
- Without a reproduction, you can't move to the next step.

Stage 3: Harden

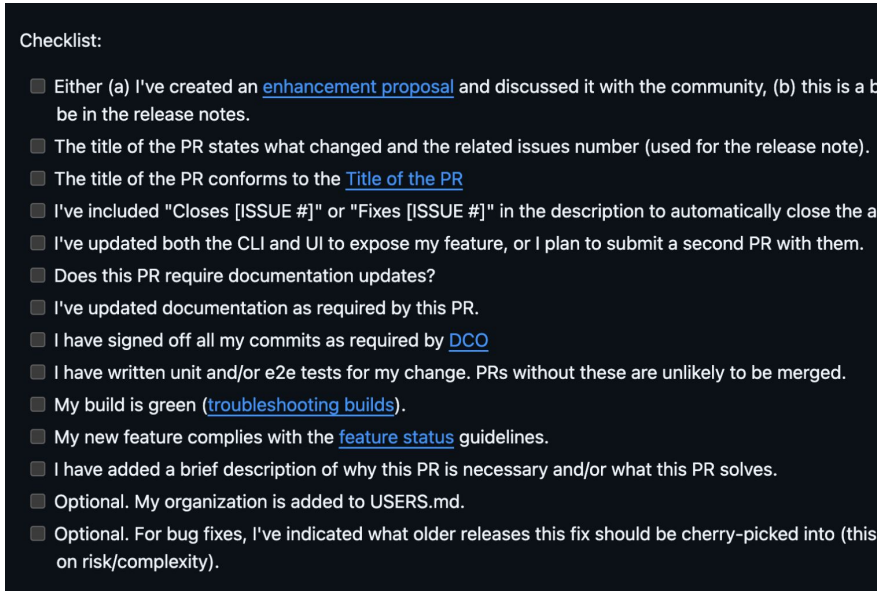
- Now you have the env, a repro, and you understand the problem.
- Fix it in code and keep track of your changes.
- Yes — add tests. But honestly:
  - mock-data unit tests aren't enough on their own
  - AI-generated tests often miss the edge cases real users hit
- tests pass, but nobody checked the fix in a real environment

Stage 3: Harden



- So attach the evidence:
  - screenshots, logs, recordings
  - manual test results, test code
- Verify in the real environment, e.g.:
  - changed SSE / EventSource handling? → test real reconnection in a browser
  - changed the UI? → check different screen sizes and states
- Reviewers know these things don't show up in unit tests.
- Vendor-related? You may need to open a PR upstream first and wait for it to merge.
  - That feels slow — it's normal.

Stage 4: Submit

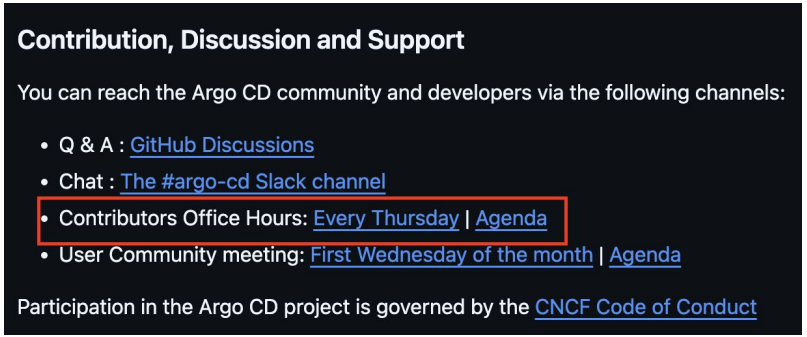


Checklist:

- [ ] Either (a) I've created an enhancement proposal and discussed it with the community, (b) this is a b be in the release notes.
- [ ] The title of the PR states what changed and the related issues number (used for the release note).
- [ ] The title of the PR conforms to the Title of the PR
- [ ] I've included "Closes [ISSUE #]" or "Fixes [ISSUE #]" in the description to automatically close the a
- [ ] I've updated both the CLI and UI to expose my feature, or I plan to submit a second PR with them.
- [ ] Does this PR require documentation updates?
- [ ] I've updated documentation as required by this PR.
- [ ] I have signed off all my commits as required by DCO
- [ ] I have written unit and/or e2e tests for my change. PRs without these are unlikely to be merged.
- [ ] My build is green (troubleshooting builds).
- [ ] My new feature complies with the feature status guidelines.
- [ ] I have added a brief description of why this PR is necessary and/or what this PR solves.
- [ ] Optional. My organization is added to USERS.md.
- [ ] Optional. For bug fixes, I've indicated what older releases this fix should be cherry-picked into (this on risk/complexity).

- Finally, open the PR.
  - You already have real evidence from the previous steps, use it.
  - Create a branch with a clear name and a clear commit message.
- Put the important things in the PR description:
  - how to reproduce
  - what environment you used
  - what changed — before & after
- And fill in the PR template + checklists. Don't skip them.

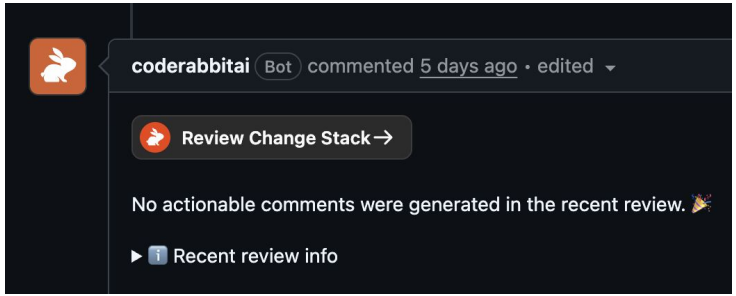
Stage 4: Submit

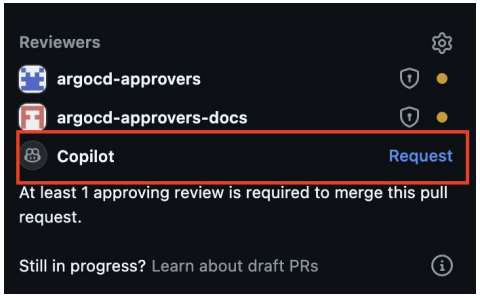


- Think about other possible solutions.
- Most problems don't have only one answer.
  - if you considered another approach, write it down.
  - This helps other contributors understand your reasoning and shows you weighed the trade-offs.
- most OSS projects have office hours.
  - bring your problem there.

Self-Review with AI

- These days there are plenty of AI review tools
  - e.g., Copilot review or CodeRabbit, depending on the project.
- Run one right after you submit the PR.
- It catches the obvious stuff early, before other contributors even look.
- It shortens the review cycle.
- shows maintainers you're managing your own PR.





How I Actually Use the 4-Stage Workflow

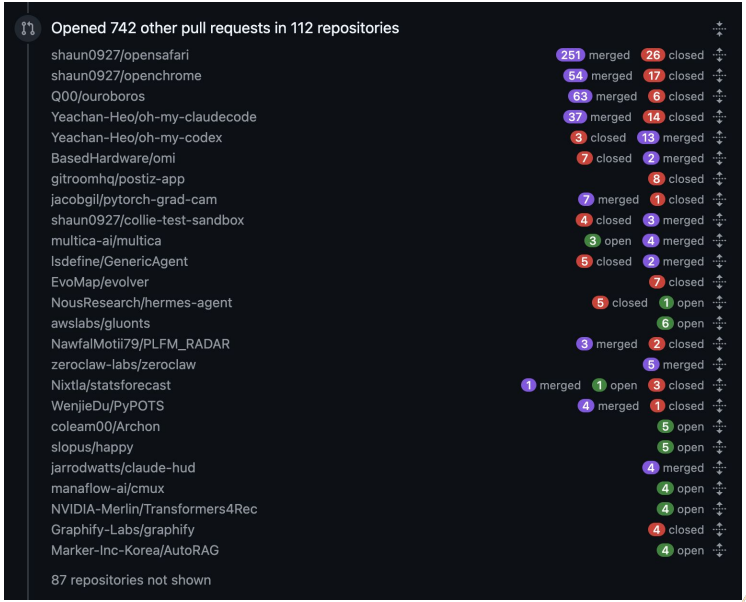
```
❯ /oss-

/oss-init                    Stage 1 — Init: Bootstrap Local Dev Environment (project)
/oss-submit                  Stage 4 — Submit: Prepare and Hand Off the PR (project)
/oss-harden                  Stage 3 — Harden (project)
/oss-validate                Stage 2 — Validate (project)
```

- Turned each stage into a LLM skill/commands
- Every skill stops at a human checkpoint
- Confirm before it moves on

Can Fully LLM-Generated PRs Actually Get Merged?

- This workflow isn't the final answer
  - AI moves too fast.
- But one story is worth telling.
- Junghwan(@shaun0927), who maintains many LLM-based OSS projects.
- "Can fully LLM-generated PRs actually get merged?"



500+ Commits in 72 hours

- He built a strict contribution pipeline.
  - Generated 500+ Commits in 72 hours — and some really got merged.
- The key
  - no raw AI dumps.
  - His pipeline acts like human-in-the-loop
  - the AI checks the fix.
  - if something fails → the workflow stops and drops it
- It shows the direction isn't wrong.
- He also became the first dev banned by GitHub, then reinstated.

"Abuse is definitely subjective."
"The rules and automation around it change on an hourly basis."
"Activity on repos you don't own is viewed differently from activity on your own."
"We might flag this as a potential account takeover."

그리고 마무리는 이 문장이었습니다.

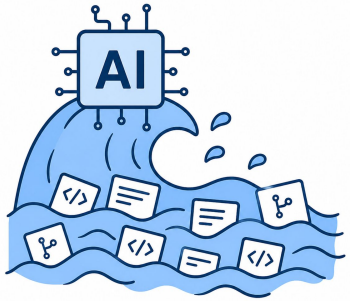
▌ "The open source community is a community, not just the transfer of bytes of code."

From "Who Wrote It?" to "Who's Responsible?"

- So what about the future of OSS contribution?
- The question is shifting
  - "Did a human write this code?"
  - "Who takes responsibility for this change?"
- A human?
- A pipeline with real verification gates?
- The human who built it?

Verification Gates vs AI Flood

- Projects that solve this well → move faster
- Projects that let raw AI output flood the tracker → burn out maintainers
- And honestly — the 4-stage workflow today is basically a manual version of what those stronger pipelines automate.



Open Source Runs on People

- At the end of the day, this whole framework is about one thing.
  - reducing the load on maintainers.
- Open source isn't a fully-staffed company.
  - We're users, contributors, and maintainers — often all at once.
  - And we're just people — with jobs, families, and other responsibilities.
- Time is the real bottleneck in OSS.
- So let's make it easier for each other.

Wrap-Up

- Stage 1 — Init: no env, no verification
- Stage 2 — Validate: understand the real problem before touching code
- Stage 3 — Harden: a passing test isn't proof — verify it for real
- Stage 4 — Submit: submit with proof, make the review easier
- Agents can generate code — they can't sign off. That's still on us.

Q&A

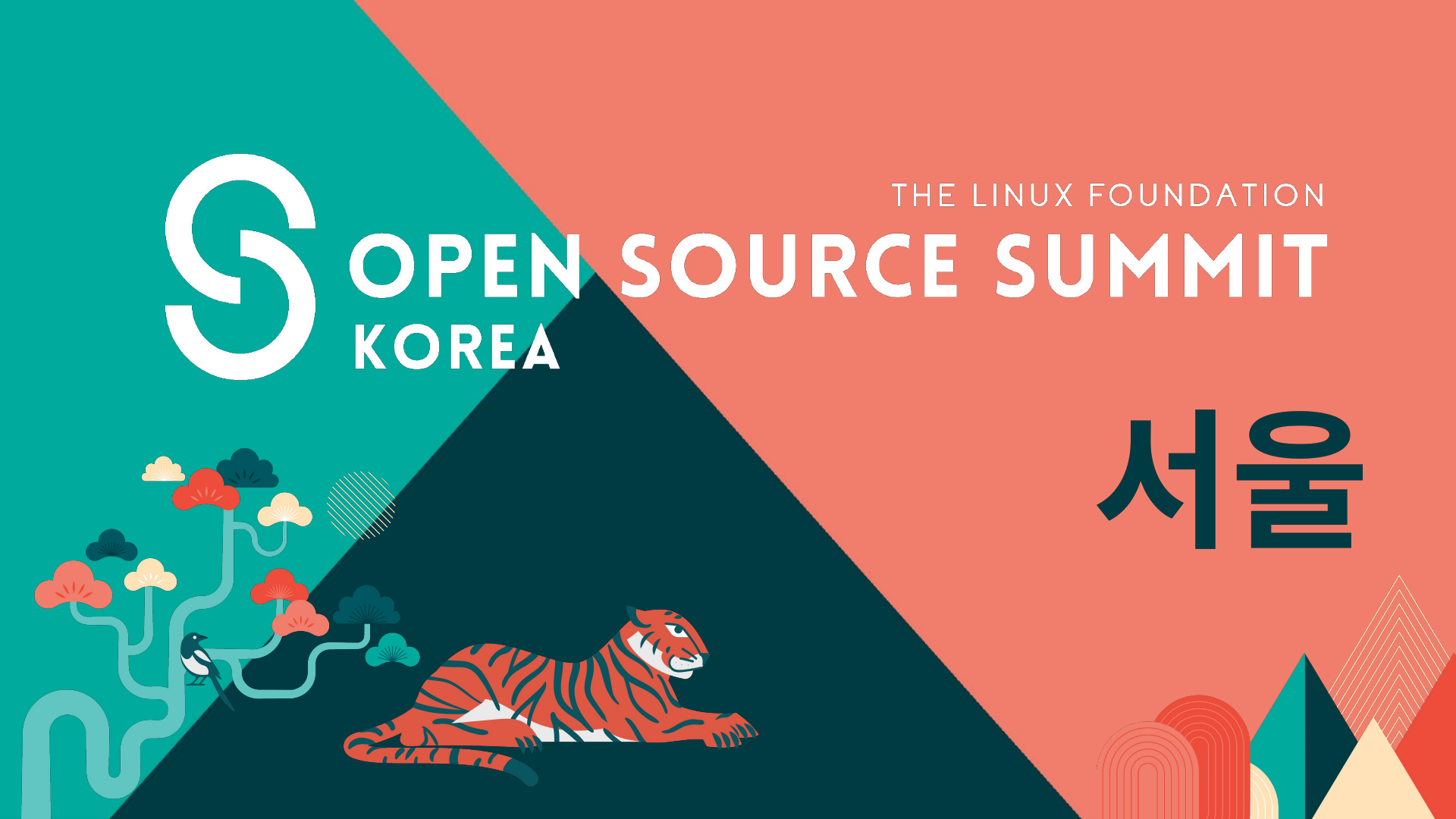
Feedback

How to Make AI-Assisted OSS Contributions Review-Ready

서울

Thank you for listening

THE LINUX FOUNDATION
OPEN SOURCE SUMMIT
KOREA


THE LINUX FOUNDATION
OPEN SOURCE SUMMIT
KOREA
서울