

Govern a fleet of coding agents by talking to one of them.

Type **asq** and you get one full-screen view over every project you have registered and every agent running in it. On the right sits a **manager** you task in prose. It turns your goal into contracts, spawns coders, testers and reviewers, reopens what fails, and posts READY with the PRs and the evidence — then stops, because the merge is a human's.

```
$ curl -fsSL https://raw.githubusercontent.com/AISquare-Studio/aisquare-cli/main/install.sh | sh
```

One line, on macOS, Linux and WSL2. It works out what the machine already has, installs only the gaps — uv, a private Python, tmux, git, gh, Node, Claude Code — registers the repo you ran it in, wires the hooks, and offers to open the UI. Run it again and it installs nothing.

PAYMENTS-API · RIGHT NOW

manager ▶

coder-idempotency ▶

coder-webhooks 🔔 NEEDS YOU

tester-1 ▶

reviewer-1 🗨

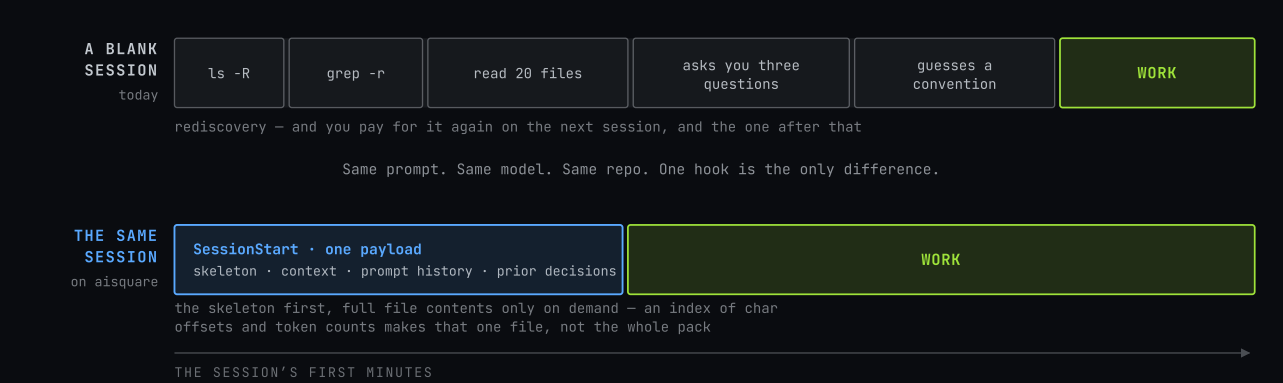
coder-retries 🔧 PR #412

Six real Claude Code sessions, one window. Click one and every key you type goes to the session itself — nothing is relayed or re-rendered as a chat. Close the UI and they all keep running.

Memory EVERYONE · ZERO COMMANDS	Every session begins already oriented: a packed snapshot of the codebase, the context entries in scope, the project's prompt history. This half needs nothing else on this page. If you only ever use it, you are using aisquare correctly.
Orchestration OPT-IN, PER REPO	Several agent sessions work one problem as a team against a shared board — contracts, atomic claims, adversarial verification, receipts. Skip it until you actually want parallel sessions; repos that never opt in see nothing.
Governance OPT-IN, ON TOP OF EITHER	Send sessions to an AISquare workspace and each becomes a Run you can read back — prompts, tool calls, tokens, cost, plus your prompts and board events. Off unless you ask, and a launch never waits on it.

A blank agent is an expensive agent.

Connecting Claude Code writes five lifecycle hooks into its settings — merged carefully, never clobbering yours. From then on every session opens on a directive pointing at a structure-only skeleton of the repo, your in-scope context, and what you have asked for here before.



The same prompt, in the same repo, on the same model. The skeleton comes first and full file contents only on demand; a per-file index of character offsets and token counts lets an agent open one file's slice of the pack instead of the whole thing.

THE FIVE COMMANDS THAT MATTER

<code>aisquare remember "prefer pytest over unittest"</code> Sticks everywhere, across every project.
<code>aisquare context add "run make check" --project</code> Sticks in this repo only.
<code>aisquare context search pytest</code> Full-text search over both pools.
<code>aisquare why</code> What the last session was actually shown, and why.
<code>aisquare doctor</code> Is everything wired — and the exact fix if not.

Nothing else is required reading. Context lives in two pools — user follows you everywhere, project is scoped to one repo — both searchable, exportable and injected consistently.

Set a repo's conventions up once and every worktree of it starts oriented.

— identity comes from the git common dir, so branches checked out side by side share one context pool, one snapshot and one board

Prompt history, per project

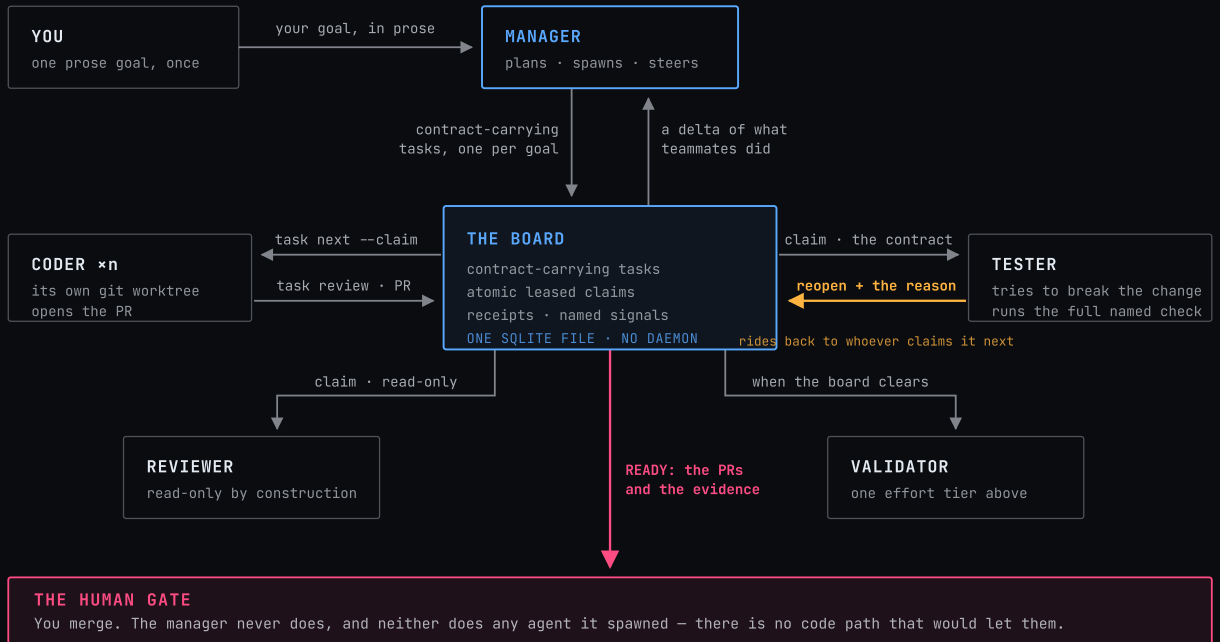
`aisquare log` is your own captured prompts — how you tend to ask, available to the next blank session.

Long-term recall

Decisions and outcomes distil into a per-project brain off the hot path. `recall "what did we decide about auth?"` searches it across weeks.

Intake, contracts, adversarial verification, one human gate.

The manager talks to its agents only through the board. That is the whole coordination model: no relaying, no forwarding, and no place for a decision to live where you cannot read it back.



The amber edge is the one that earns its keep. The tester runs the full check the contract named, tries to make the change fail, and then either closes the task with evidence or reopens it with the reason — and that reason rides back to whoever claims the task next. Nobody carries it there by hand.

Contracts, not tickets Every task carries an objective, a why, acceptance criteria and boundaries. A coder blocks instead of guessing when a task has no usable contract.	Claims are atomic and leased A single UPDATE, race-tested: exactly one winner. Leases renew from the session's own hooks, so a dead session hands its work straight back.
Dependencies are held task next only hands out work whose dependencies are done. A rollback rehearsal cannot start before the migration it rehearses.	Signals, not substring matching Named board states with structured payloads, so a watcher keys on a field and a note saying NOT READY can never fire it.

EVERY ROLE ON THE RIGHT MODEL, AND IT IS CHECKED

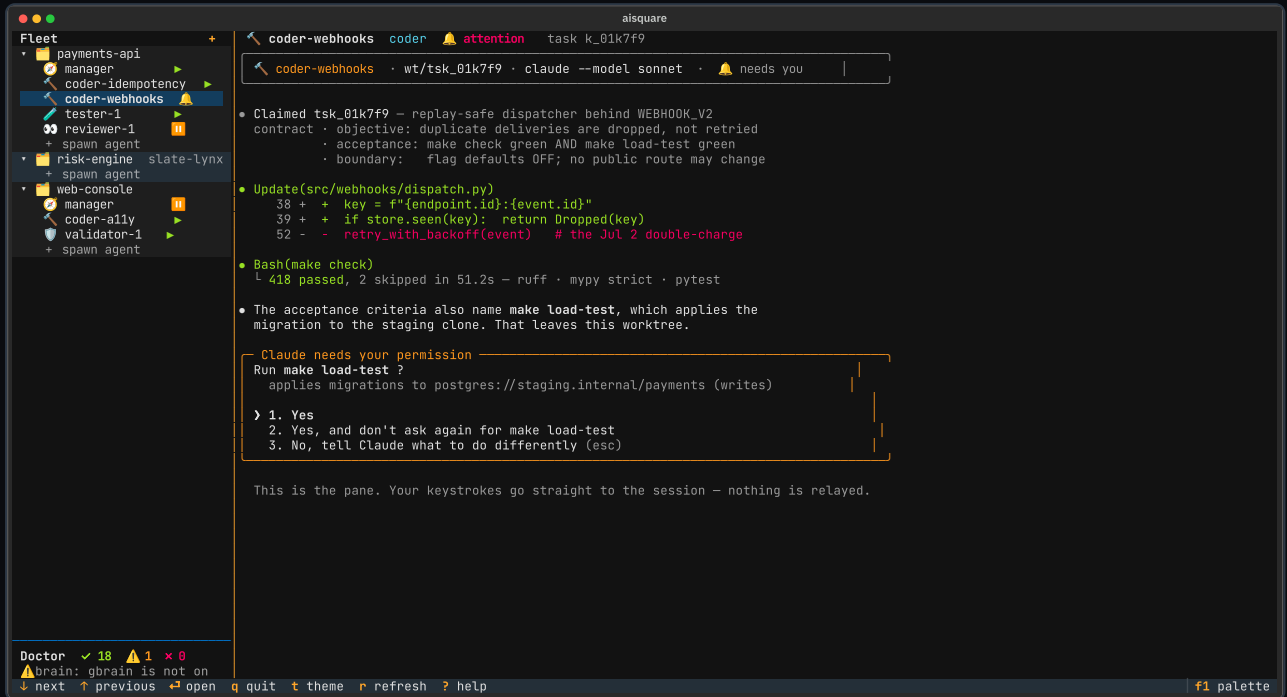
```

$ aisquare team harness
base effort: xhigh (inherited)
manager  fable→opus→sonnet  effort=xhigh  ( 0)
coder    sonnet→opus        effort=xhigh  ( 0)
tester    sonnet→opus        effort=xhigh  ( 0)
reviewer  sonnet→opus        effort=xhigh  ( 0)
validator fable→opus          effort=max    (+1)
  
```

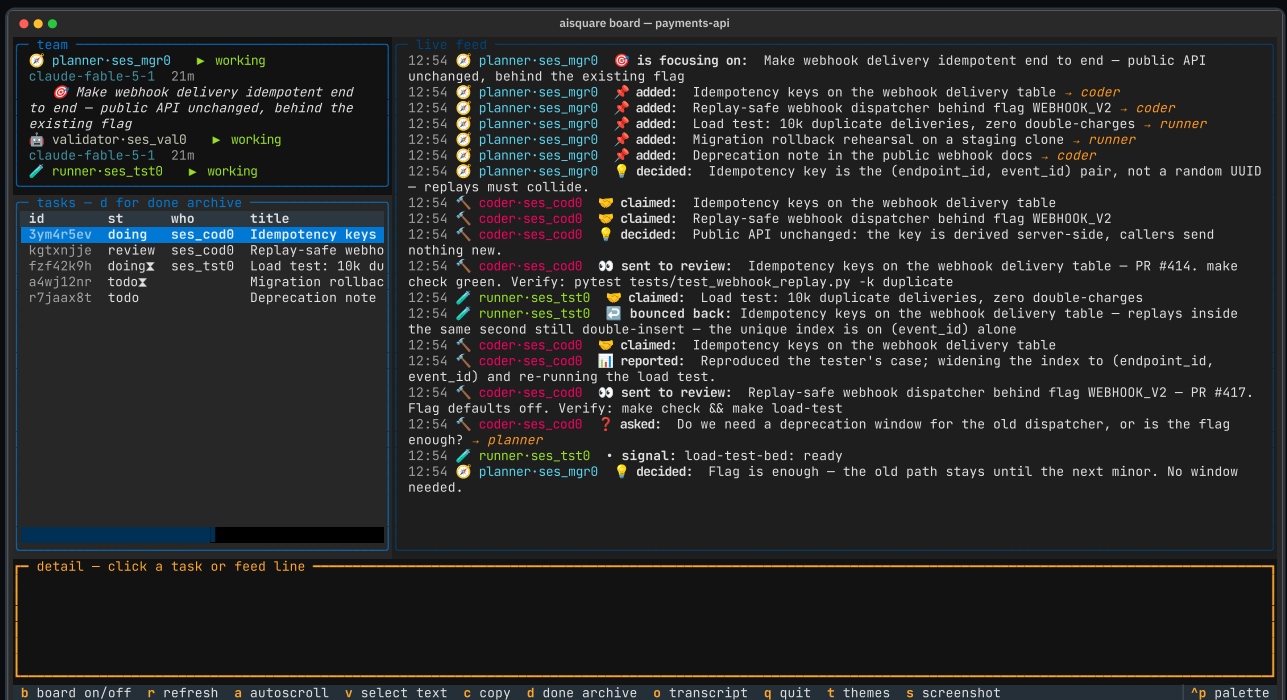
A ladder per role, strongest rung first. `claude --model` silently substitutes the default when a model is not on the account, so the harness verifies the reply before trusting a rung and demotes **only on proof** — an outage keeps your pick and labels it unverified rather than quietly downgrading your manager. The validator's +1 offset exists because a gate must outrank the work it checks.

Real sessions. Not a chat wrapper over them.

Each agent is a Claude Code process running as a window on a tmux server that is ours alone — your own tmux sessions, config and prefix key are never touched. The UI is a view over that server and the board, which is why closing it changes nothing.

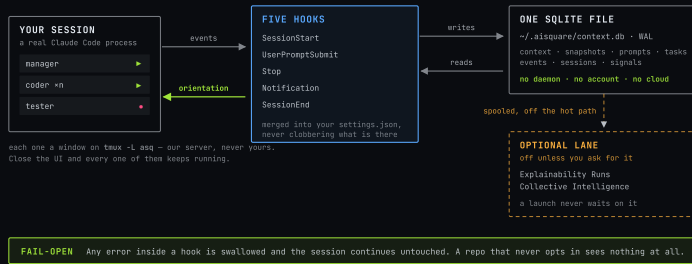


The coder hit a permission prompt, so its sidebar row went **NEEDS YOU** and the terminal rang. Click the row and you are in the session: the contract it claimed, the diff it wrote, make check green, and the one decision it will not take on its own. Your keystrokes go straight through. Worked example, not a customer's repo.



The same work, as the board sees it. Sessions with live state and the model each reports, the open tasks with their dependency holds, and a feed of every claim, decision, review, reopen and signal — each line clickable for its full detail, and o opens the author session's transcript at that exact moment. These rows are real: produced by driving the shipping CLI through the sequence above.

Built to be harmless in a repository you care about.



aisquare sits beside the agent, never in its path. The hooks carry events out and orientation back; state is one local SQLite file; the cloud lane is spooled, opt-in and off by default.

Everything is a command

Every action the UI takes is a plain CLI command, and every command takes `--json`. Piped or under `TERM=dumb`, no script ever meets a full-screen app.

Receipts you can re-prove

Writes print ✓ ... seq N. team verify 42 asks the board whether it is really there and exits 0 or 1 — and a receipt on another board is an honest not-found that names it.

Roadmap stays hidden

Unfinished commands are hidden from `--help` and exit 70 saying so, rather than half-working. What the help lists, works.

Contracts, not vibes

The Collective Intelligence server's JSON Schemas are vendored byte-for-byte, and every request this build can emit is validated against them in CI — never against our own reading of them.

● NOW • V0.6.0

- The fleet UI and the manager loop, end to end
- Memory: two pools, packed snapshots, prompt history
- Board: contracts, leased claims, receipts, signals
- Model ladder per role, probed, with effort offsets
- Several accounts driving one board
- Remote agents over MCP · Explainability Runs

● NEXT

- Sign in — one identity across your machines
 - sync: memory that follows you, and your team
 - On-limit handoff — a seat runs out, the work moves
 - Remote control: watch and steer off-terminal
- Not speculative: leases already release on session end, and a seat is already just a binary plus its environment.

○ LATER

- Connectors — meeting notes and tickets into the knowledge base
 - Organisation policy the agents cannot cross
 - A knowledge graph of findings, decisions and preferences
- Collective Intelligence runs against a live staging server today, off by default, and measures nothing yet. We will say when that changes.

A PILOT IS ONE REPOSITORY AND ONE AFTERNOON

```
$ curl -fsSL .../install.sh | sh # only the gaps
$ asq # point it at a repo
... then press Start manager and type the goal.
```

Requirements are what your developers already have: a Claude Code login, tmux 3.2+, git, gh. Nothing to deploy, no account to create, and no traffic leaves the machine unless you switch that on deliberately.

WHAT WE WOULD LIKE FROM YOU

One repository with real parallel work in it, and a team willing to talk to a manager instead of five terminals for an afternoon. We run the loop on a live goal and hand back the transcript, the board and the Runs.

pypi.org/project/aisquare-cli
github.com/AISquare-Studio/aisquare-cli
MIT licensed · screens captured from the shipping UI at v0.6.0.