n'co studio

R&D lab for markpreston.co

GOVERNANCE INFORCEMENT · INTERNAL DOCUMENT

How Repo OS Enforces Governance

How the rules are made unbreakable — the daemon, the queue, the phase sequence, and the gate chain every automated build must pass to merge.

AUTHOR
Mark Preston

DATE
July 2026

PERIOD
massimo (archived)

STATUS
Reference

01 · THE MENTAL MODEL

One brief, from intake to merge

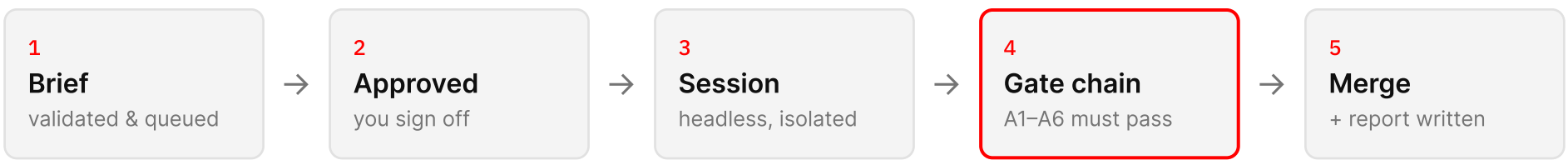
The unit of work is a brief: one task, one branch, one merge. Everything else is machinery for getting that brief built, checked, and merged without a person babysitting it.

A brief is validated and de-duplicated at intake, then waits in the queue for your approval. A background daemon picks up approved briefs and runs each one as a headless, worktree-isolated Claude session that walks a fixed sequence of phases — the "rooms." When the session tries to commit, a chain of gates decides whether the work is allowed through. Every run leaves a report behind, and anything that goes wrong leaves an escalation.

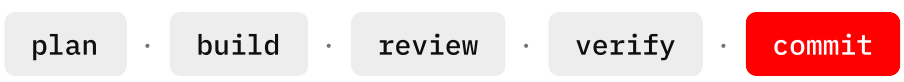
KEY TERMS

Brief	One task, one branch, one merge — a markdown file carrying scope, approach, and verification criteria.
Daemon	The runner.sh process. Polls the queue every 60 seconds and writes a .health heartbeat.
Session	A headless claude -p run, executed in its own isolated git worktree.
Rooms	The phase sequence a session walks: plan, build, review, verify, commit.
Gates	Blocking checks that run at commit. Fail one and the commit is refused.

FIGURE 1 — ONE BRIEF'S PATH THROUGH THE SYSTEM



INSIDE STAGE 3 · THE ROOMS



The gate chain fires at commit — the last room.

02 · HOW THE RULES HOLD

Enforced, then proven

Governance here is not a document people are asked to follow. It is enforced by code on every run — and then attacked on purpose, to prove the enforcement cannot be dodged.

ON EVERY RUN · MECHANICAL ENFORCEMENT

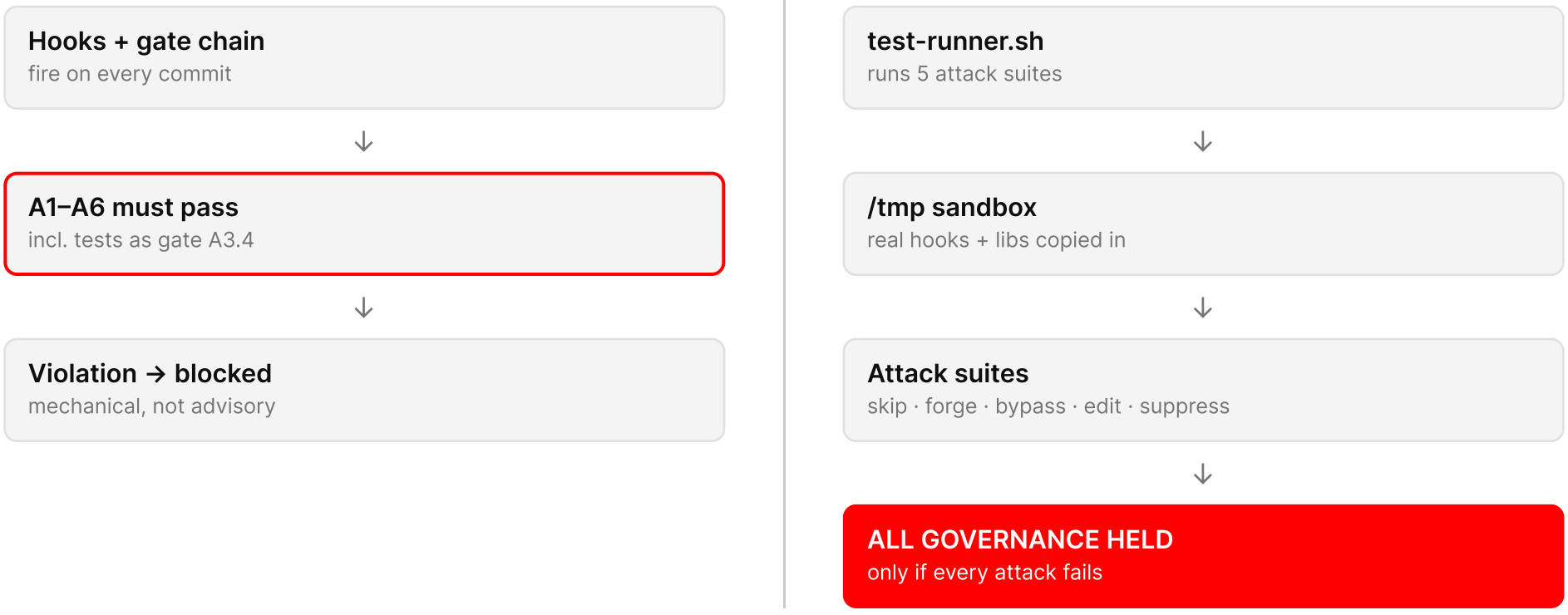
Hooks and the gate chain fire on every commit and block anything that breaks the rules — an undeclared decision, a secret, a failing build, an unreviewed diff. It is mechanical, not advisory. The project's own unit and integration tests plug in here as one gate among many (A3.4), checked the same way as everything else.

BY ATTACK · ADVERSARIAL PROOF

Enforcement is only as good as its weakest bypass, so tests/test-runner.sh stands up a disposable /tmp sandbox, copies the real hooks and libraries into it, and runs five attack suites that actively try to cheat the system: skipping phases, forging state files, bypassing gates, editing their own queue entry, and suppressing escalations. It prints "ALL GOVERNANCE HELD" only when every attack fails.

> *The rules are enforced by code — and the enforcement is attacked on purpose*

FIGURE 2 — ENFORCEMENT, AND THE PROOF IT HOLDS



03 · THE GATE CHAIN

What has to be true to commit

When a session reaches the commit phase, a chain of blocking checks runs. Fail any one and the commit is refused — most gates carry a named override for emergencies, and every override is logged.

The gates run in order. A1 and A2 cover decisions and process; A3 covers quality, including the build check (A3.3) and the test run (A3.4); A4 covers safety; A5 is the Critic's two-pass diff review; A6 covers task-group contracts. Pushing a branch adds B1–B3; merging a release adds C1–C3.

FIGURE 3 — THE COMMIT GATE CHAIN (RUN IN ORDER)

A1	**Decision compliance**	triggered decision records must be declared
A2	**Process**	a plan exists; scope over 100 lines needs approval
A3	**Quality**	formatter · lint · then:
A3.3	**Build**	tsc --noEmit
A3.4	**Tests**	npm test --if-present
A4	**Safety**	secret scan · PII · high-risk paths need sign-off
A5	**Cohesion**	Critic reviews the diff twice — both must be APPROVED
A6	**Task group**	contract staged · integration artifacts exist

ALL PASS → COMMIT ALLOWED

ANY FAIL → REFUSED (OVERRIDE LOGGED)

PUSH ADDS B1–B3 · RELEASE ADDS C1–C3

04 · THE QUEUE AND THE DAEMON

How work moves, and who moves it

The queue is just folders. A brief is a file that moves from one folder to the next as its state changes — and the daemon is the process that does the moving.

Approved briefs are picked up, run, and land in completed, review, failed, or blocked. The daemon retries a failed brief up to three times, defers a brief whose dependencies aren't ready (giving up after three cycles, which sends it to blocked), and on startup recovers any work orphaned by a crash.

> **WHY THE RUNNER DOES SO MUCH**
>
> The interactive Stop hook doesn't fire in headless mode (Decision 0017), so the runner has to replicate every post-session task itself — merging, sealing the branch, updating state, escalating on a dirty tree. RUNNER-PARITY.md tracks all fifty-plus of those responsibilities.

FIGURE 4 — A BRIEF'S LIFE IN THE QUEUE

pending-review
intake: validate + dedupe

→

approved
you sign off · retries return here

→

running
daemon runs the session

RUNNING ENDS IN ONE OF:

`completed` | `review` | `failed` | `blocked`

↺ failed → approved — the daemon retries, up to 3 attempts

→ retries exhausted, or deferral cap (3 cycles) reached → blocked

THE DAEMON

`polls every 60s` `.health heartbeat` `HALT / FORCE-HALT` `retry ×3` `worktree-isolated` `crash recovery`

05 · WHAT "WORKING" LOOKS LIKE

How you know it is healthy

There is no live dashboard. The system's state lives in a handful of files the daemon writes as it runs — reading them is how you know it is working.

WHERE TO LOOK

`.health`	the heartbeat — status, pid, uptime, and sessions completed vs failed
`reports/`	one YAML per session outcome — completion, blocker, or crash, with exit code, phase at exit, and git status
`escalations/`	fail-closed signals to resolve — dirty tree, stale branch, gate failure, governance block
`test-runner.sh`	run the attack suites — "ALL GOVERNANCE HELD" means the rules still hold

LAST RECORDED RUN · FROM .health

8

sessions completed

0

sessions failed

ALL GOVERNANCE HELD

no bypasses found

A healthy run is one the daemon built, checked, and merged under the rules — with the reports and escalations there to prove it.