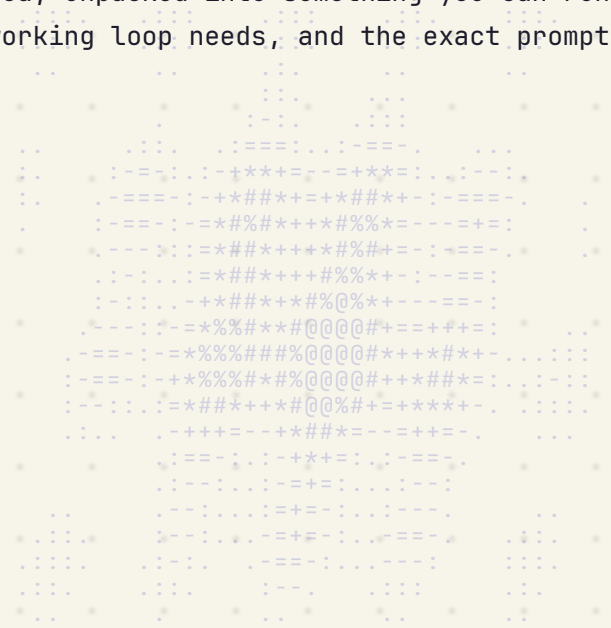
// SECTION 01 · PLAYBOOK

WRITE LOOPS NOT PROMPTS

Boris Cherny built Claude Code. Now he says his job is writing loops that prompt Claude for him. The idea, the anatomy of a working loop, and the prompt that sets up your first one tonight.

```
> boot opusjake_os
> resource: loops
> version: v1.0 · 2026
> status: ready_
```

Boris Cherny created Claude Code. Then he stopped prompting it. His words: "I don't prompt Claude anymore. I have loops running that prompt Claude and figuring out what to do. My job is to write loops." This is that idea, unpacked into something you can run tonight: what a loop is, the six parts every working loop needs, and the exact prompt that builds your first one.



THE IDEA

A loop is a small system that prompts Claude on your behalf. It kicks off the work, reads what came back, checks it against proof, and prompts again with fresh context until the job is done or a budget runs out.

The cycle is four beats:

1. **Act.** Claude does a pass at the task.
2. **Verify.** A command proves whether it worked. Tests, a build, a curl.
3. **Decide.** Done? Stop. Not done? Figure out what changed.
4. **Re-prompt.** Feed the result back in and go again.

Cherny calls the shift bigger than it sounds: "Going from agents to loops is as big a jump as going from code to agents." You stop being the operator who types every instruction and become the designer of the system that does the instructing.

The receipts, as he tells it: Claude Code wrote 100 percent of his code contributions over a recent 30 day stretch, 259 pull requests, and merges per engineer on his team went up 200 percent. Your mileage will vary. The mechanism will not.

THE ANATOMY

Every loop that survives contact with reality has six parts. Miss one and it either runs forever or quietly does nothing.

PART	WHAT IT ANSWERS
Trigger	What starts a pass. A timer, a Slack message, a failing check.
Scope	What it owns, and what it must never touch.
Action	The actual work each pass performs.
Verification	The command that proves it worked. Output, not vibes.
Budget	Max passes, max time, max spend. Always set one.
Stop + report	When to halt, and what to tell you when it does.

The lever that matters most is verification. Cherny has said self-verification is the key ingredient that lets a model run for long stretches and land close to what you wanted, and that a real feedback loop lifts result quality 2 to 3x. A loop without a proof command is just a prompt on repeat.

One more trick from his setup: the agent that does the work should not grade its own homework. A second agent, or just a dumb shell command, checks the result.

LOOPS HE ACTUALLY RUNS

These are the confirmed examples from Cherny's own talks and posts. Notice how boring they are. That is the point.

- **PR babysitting.** A loop watches his open pull requests, responds to review comments, fixes failing checks, and keeps them mergeable.
- **Slack triggers.** A message in a feedback channel kicks off a loop that triages the report and drafts the fix.
- **Post-merge sweeps.** After code lands, a loop cleans up: docs, changelogs, follow-up chores nobody wants.
- **PR pruning.** A scheduled pass closes stale branches and redundant pull requests.

None of these are moonshots. They are the 30 minutes of glue work you do every day, made into a system that does it without you.

RUN ONE IN A SESSION

Claude Code ships the primitives. You do not need to write a bash daemon.

// /LOOP

Runs a prompt on a repeating interval, right in your session.

```
/loop 10m check CI on this branch. If anything is red,
diagnose and fix it, then push. If green, do nothing.
```

Leave the interval off and Claude paces itself, deciding when the next pass should fire.

// /SCHEDULE

Same idea, but as a recurring cloud job. It runs on a cron schedule with your laptop closed. Use it for the daily and weekly loops: the Monday triage, the nightly dependency check.

// THE VERIFICATION HABIT

Whatever you loop, end the instruction with proof. "Run npm test and only stop when it exits clean" beats "fix the tests" every single time. Give the loop a command whose output cannot flatter you.

THE PROMPT

Paste this into Claude Code. Fill the two brackets. It designs the loop before it runs it, which is exactly the order Cherny works in.

```
I want a loop, not a one-off task.

The job: [WHAT YOU WANT KEPT TRUE, e.g. every open PR on
this repo stays green and review-ready].
Proof it worked: [A COMMAND, e.g. gh pr checks --watch
exits 0 on every open PR].

Before running anything, design the loop:

1. Trigger: how often a pass should fire, or what event
   kicks one off.
2. Scope: exactly what you may touch. List what is
   off-limits.
3. Action: the work each pass performs, as concrete steps.
4. Verification: run the proof command after every pass.
   Trust its output over your own summary.
5. Budget: hard cap of [N] passes. Stop early if two
   consecutive passes change nothing.
6. Report: when you stop, tell me what changed, what
   passed, and what still needs a human.

Show me the design in a short table first. Then start it
with /loop and run until the stop condition hits.
```

First run, make it read-only: have the loop report what it would do instead of doing it. Promote it to write access once you trust its judgment.

THE RULES

Loops amplify whatever you point them at, including your mistakes. Four rules before you leave one running:

1. **No loop without a budget.** A cap on passes or time, always.
2. **Proof is a command.** If success cannot be checked by something with an exit code, tighten the job until it can.
3. **Read the diffs.** Cherny still reviews what his loops produce. Comprehension debt compounds faster than tech debt.
4. **Start boring.** Your first loop should own one small recurring chore, not your roadmap.

Tonight's move: pick the one check you already do every day by hand, paste the prompt above, and let a loop own it by morning.