NERO PROJECT

Command Center - Every Project on One Live Screen

If it is on the board, it is current.

Paul Schindler | Built with Claude | November 2026

Executive summary

My home AI operations system carries 55 projects, and their written records kept going out of date without anyone noticing. I built Command Center, one live dashboard that shows every project with its status, its next step and the date a person last confirmed that step. A plain script draws it from one structured file as a web page on my home network and as a status image sent to my phone on request, so viewing costs no AI usage. A nightly run, a check at the start of every AI working session and a 10-minute Monday review flag what has gone stale, in code rather than by memory or AI judgment. Next steps left unconfirmed for 30+ days fell from 10 of 22 live entries in October 2026 to 3 of 40 in November 2026.

55	30 days	0	Live
projects on one board	before an unconfirmed next step greys out	AI tokens spent to view a report	since June 2026

What the board answers at a glance

- Where each project stands, in one of five plain states.
- The single next step that moves it.
- When a person last re-read that step and confirmed it.

What I decided it must never do

- Make an old entry look fresh because a file changed nearby.
- Spend AI usage every time someone looks at it.
- Let a new project exist on the machine without showing up.

What is in this report

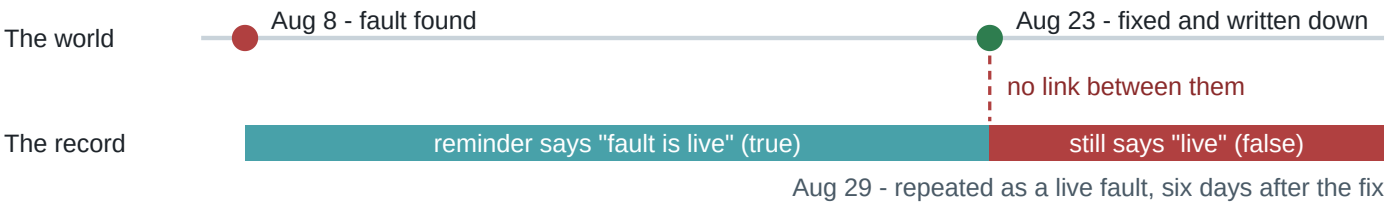
Page	Contents
2	The problem - records that stay confident after the world has moved on
3	Design choices, what I ruled out and the "two clocks" idea at the center of the design
4	How it works - the step flow, the system diagram and what each project card shows
5	What the board looks like, the five project states and ten rules the system enforces
6	Results, honest status, what is next and what the project shows about how I work

1. The problem - records that outlive reality

Every project I run leaves a written record behind it, with a status, a next step and a note on what is blocking it. My failure was never missing records. It was records that stayed confident after the world had changed.

One evening in August 2026, four separate records fired at once, each stating something that had stopped being true. One reminder kept reporting a fault as live six days after the fix was written down, in the very document that reminder cited. Nothing connected the record to the change. The same day, 9 of 10 open reminders showed as overdue. A screen where everything is red teaches its reader to ignore red, so I treated that as a failure of the screen and not of the reader.

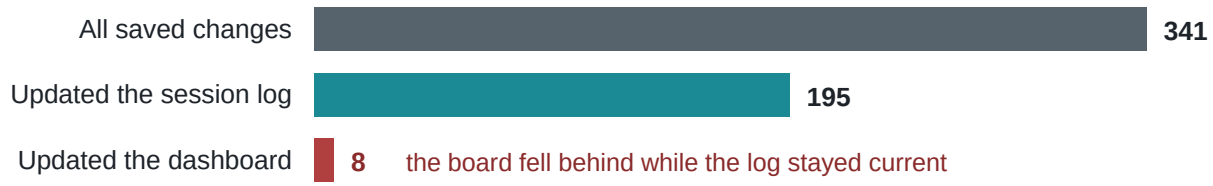
How one record went stale (August 2026)



The dashboard had the same disease

I started in March 2026 with a single list of every project, kept by hand in my notes. The board that replaced it went live in June 2026 but still depended on whoever finished the work to update its entry by hand. I measured the result over the 30 days to November 6, 2026.

Changes saved to the system in 30 days, and which record each one kept current



Writing the session log is part of finishing a session, so it got done. Updating the dashboard was a separate chore, so it slid. Counts come from version history for the 30 days to November 6, 2026.

By early November 2026 I had stopped opening the board, because it read as out of date. I ruled that the data had to be fixed before any further visual work on it. A status screen nobody trusts is worse than no screen, because it still gets quoted.

Slow is not stale

A task can be overdue for months and still be described accurately. Aging records out automatically would have deleted true information. A record is stale when it says something about the world that is no longer true, or when no person has vouched for it inside a set window.

Why this matters beyond one desk

Any program office with many workstreams has the same failure. A weekly status deck can show every line as current while many of its next steps have quietly expired. The fix has to live in the process, not in good intentions.

One class this system cannot catch, by design

Some evidence lives outside the machine. A deliverable mailed and answered in my inbox leaves no trace in the project folder, so a file-date check would call it abandoned. Before calling any deliverable unreceived or any project dead, I check the channel it went out on, not just its folder.

2. Design choices and what I ruled out

I weighed three places the board could get its data, then tested several ways to keep that data honest. Two of the options below were built and then failed in real use.

Option	Why I ruled it out
An AI model reads every project note on each view	It re-reads everything every time anyone looks, which made it the most expensive option by far. It also hands the freshness judgment to a model, and models are weak at exactly that judgment (see the box below).
Status fields scattered across every project's notes	The same fact ends up hand-written in several places that drift apart. I adopted one home per fact instead.
Move the "updated" date forward whenever files change	Built, then removed in October 2026. An outside audit of the whole system found it made stale entries look fresh, because activity near a project is not a review of what its record says.
A weekly AI pass that hunts for contradictions	The obvious move and the measured worst one. It would leave records stale for days between passes while raising confidence in its output.
Age old records out automatically	Slow is not stale. It would throw away accurate records of long, legitimate waits.
Add optional freshness fields and trust people to fill them	Built in August 2026, and the fields stayed empty. Of 32 open reminders, 31 had none set because nothing asked. Adding a field is not the fix; making something demand it is.

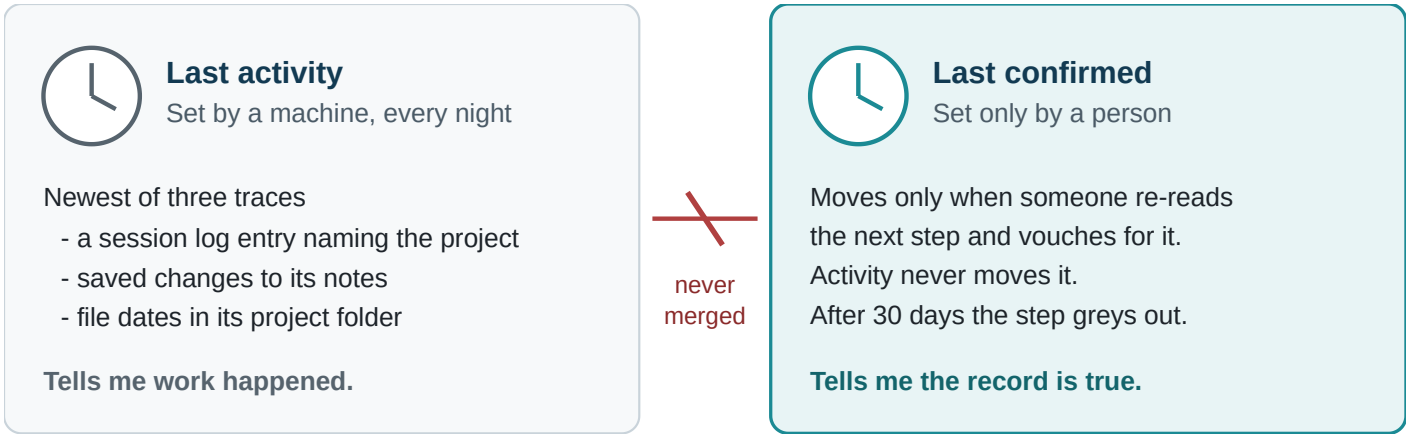
What I chose

One structured file is the single source of truth for every project. A plain script draws the board from it. Freshness is a date comparison done in code, and the code only says "re-read this one." A person does the judging.

Why freshness lives in code

Asking a language model to judge which record is newest gives answers that drift from run to run. A date comparison in code gives the same answer every time. So the 30-day rule is ordinary code, not an instruction to an AI.

The two clocks



Activity is not review. The board shows both dates on every card, and only the right-hand one counts as confirmation.

3. How it works

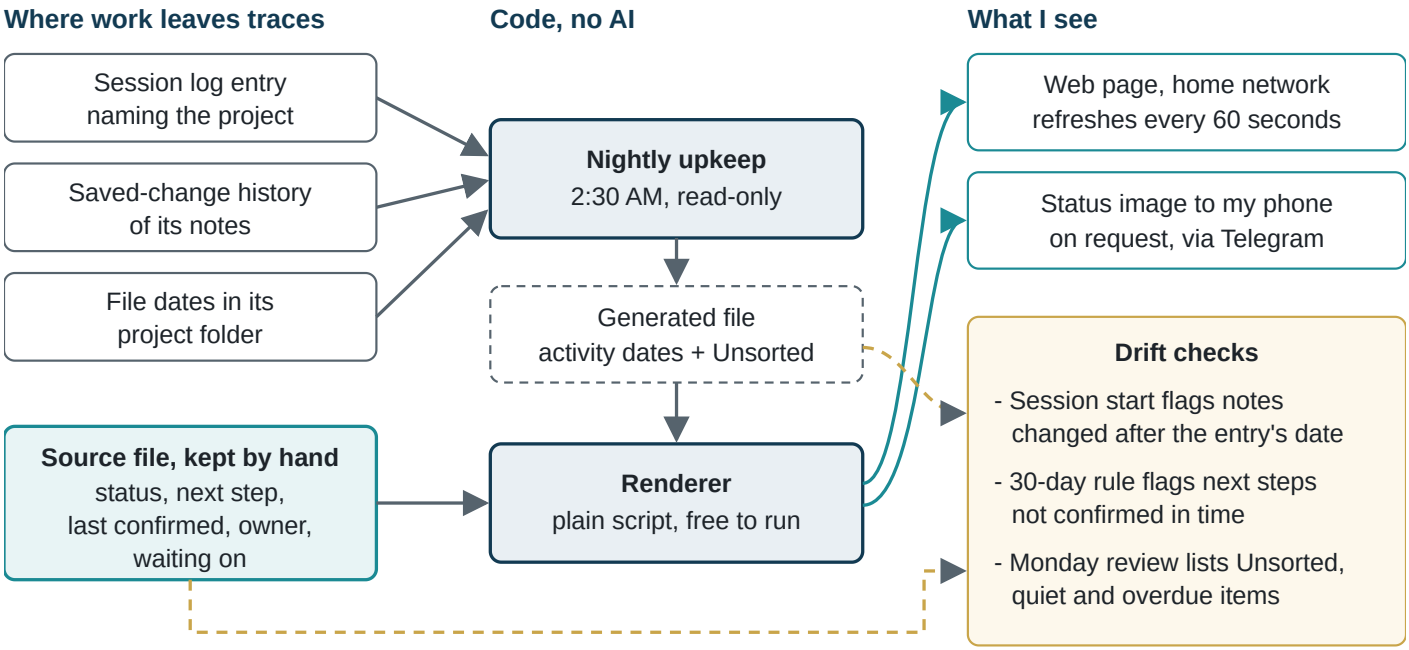
1. Work happens
An AI working session finishes and names its project on one line of the session log.

2. Nightly upkeep
At 2:30 AM a script records each project's last activity and lists anything it cannot place.

3. Plain script draws
A renderer with no AI in it draws the web page and the status image from the source file.

4. I look
The page refreshes every 60 seconds. A message to my Telegram bot returns the status image.

5. Checks flag drift
The session-start check, the 30-day rule and the Monday review name what needs a re-read.



The nightly script writes only its own generated file and never touches the hand-kept source file. Viewing runs no AI model at any step.

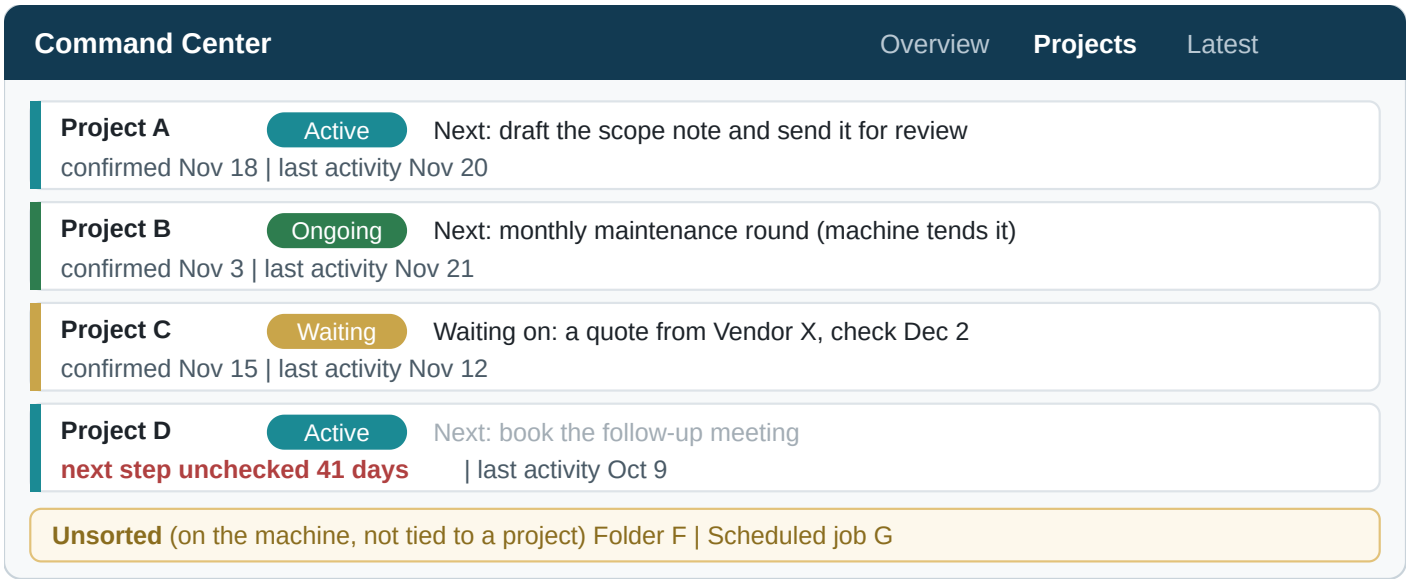
What each project card shows, and who sets it

Field	Set by	What it tells me
Status	A person	Where the project stands, in one of five plain states (page 5).
Next step	A person	The single action that moves the project forward.
Last confirmed	A person only	When someone last re-read the next step and vouched for it. Past 30 days the step greys out.
Last activity	Nightly script	When work last touched the project, from the three traces above.
Waiting on	A person	For a Waiting project, what it waits for, from whom and the date to check.
Recent	A person	One-line notes, newest first, that feed the board's "Latest" panel.

One change I made to my own plan

The plan had the nightly script write its dates into the hand-kept source file. I changed that when it was built. A nightly write would have left a fresh, unexplained edit in the version history every morning and blurred who had changed the record. The script now writes to its own generated file, and the board reads both.

4. What the board looks like



Structure of the Projects view, drawn with invented placeholder rows. Project D shows the 30-day rule at work. The real board holds private projects and is not shown.

Five project states

State	Meaning
Active	Has a finish line, and I am working on it now.
Ongoing	No finish line; tagged "you" if I tend it or "system" if a machine does.
Waiting	Stuck on a specific outside event from a named party, with a check date. My own next step is never Waiting.
Someday	Not now. Stays visible, and nothing nags.
Closed	Finished or dropped.

Adopted November 2026. The split follows two established methods, PARA (projects versus ongoing areas) and Getting Things Done (its "Waiting For" and "Someday/Maybe" lists).

Ten rules the system enforces

1. **One source of truth.** Every view comes from one file. No second list.
2. **Viewing is free.** No AI model sits in the path that draws the board.
3. **Confirmation is human.** Activity never sets "last confirmed."
4. **30-day rule.** An unconfirmed next step greys out and gets named.
5. **Session-start check.** Each AI working session opens by listing entries whose notes changed after the entry's date.
6. **Hands off the hand-kept file.** The nightly run writes only its own file.
7. **Nothing hides.** An unmapped folder, job or project shows as Unsorted.
8. **Waiting names something.** What, from whom and a check date.
9. **Exceptions only on Monday.** Unsorted items, Active projects quiet 14 days, overdue Waiting checks.
10. **Checks fail quiet.** A scan bug never breaks the session briefing, and scans under-report before they cry wolf.

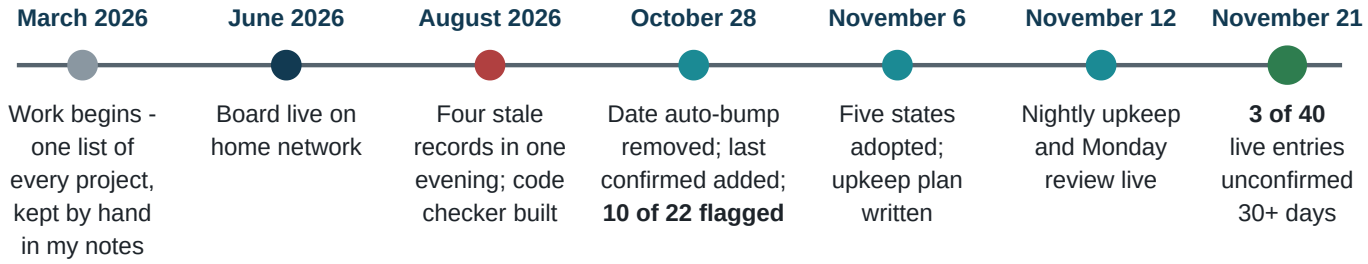
5. Results, status and what is next

10 of 22
live next steps unconfirmed 30+ days, October 2026

3 of 40
live next steps unconfirmed 30+ days, November 2026

9
unmapped project folders caught on the first nightly run

10 min
weekly review, exceptions only



All dates 2026. "Flagged" and "unconfirmed" both mean a live next step nobody had confirmed in 30+ days.

The board grew from 22 live entries to 40 while unconfirmed next steps fell from 10 to 3.

Part	Status	Detail
Web page and phone image	Live since June 2026	Both drawn by the same plain script from the same file.
30-day confirmation rule	Live since October 2026	Shown on the board and in the session-start briefing.
Nightly upkeep and Monday review	Live since November 2026	Its first run found 9 unmapped project folders. The Unsorted list is not yet empty; sorting it is my open task.
Catching a working session that left no log entry	Designed, not built	Deferred until the Unsorted list runs clean.
Per-owner views and a scheduled sync from project notes	Not built	The owner field already exists. Any sync would use a small, cheap AI model per project and stay out of the view path.

6. What it shows about how I work

Running a program is mostly keeping a shared picture true. I treated a stale status as a process failure, not a people failure. I put the update where the work already happens (the session log), split what a machine can know (activity) from what only a person can vouch for (confirmation) and made the exceptions come to me once a week. When a feature I had built made the data look better than it was, I removed it.

The same discipline carries to a partner pipeline, a grant calendar or a multi-agency rollout. Each line names its owner, its next step and the last date someone confirmed it, and the system says which lines have gone quiet. I designed Command Center, set its rules and verified it; AI agents wrote the code.

Sources for every figure in this report are internal design notes, change briefs, the session log and version history for the system, dated June to November 2026. The March 2026 start comes from my own earlier notes, which predate those records. The November 21, 2026 count is my own tally of the live board.