AI EDGE

The Ultimate Guide to Looping

Stop prompting your AI. Start designing the loops that prompt it for you.

By Miles Deutscher

Every diagram, every command, and every loop from the video. Plus the full library.

READ THIS FIRST

Why this guide exists

In June 2026, the head of Claude Code at Anthropic said something that should stop you mid scroll: "I don't prompt Claude anymore. I have loops running that prompt Claude. My job is to write loops."

Within the same month, Anthropic and OpenAI both published official guidance saying the same thing. The skill that mattered for the last two years, writing clever prompts, is being replaced by a new one: designing loops.

Most explanations of this are written by developers, for developers. This one isn't. By the end of this guide you will understand exactly what a loop is, why the shift happened now, how to set up your first one today, and which loops are worth running in your business and your life.

What's inside

- The 7 diagrams from the video, with plain English explanations
- Every command shown on screen, ready to copy and paste
- How to write a "definition of done" that actually works
- The library: 60+ loop ideas across content, marketing, business, sales, finance and life
- The rules that keep loops useful, safe and cheap

> **One thing before you start.** Everything here runs in Claude Code, Anthropic's terminal agent. You need version 2.1.170 or later. Run `claude update` first, then `/model fable` to select the smartest model. That's the whole setup.

PART 1

How AI actually works

Picture the perfect employee. Brilliant at any task you hand them: writing, coding, analysis, strategy. There's one catch. They have total amnesia. Every time the conversation ends, they remember nothing.

Their entire memory is one whiteboard, called the context window. Everything gets written on it: your instructions, their attempts, the errors, your corrections. It fills up, it gets messy, and it never gets erased. When it's full, the beginning starts falling off.

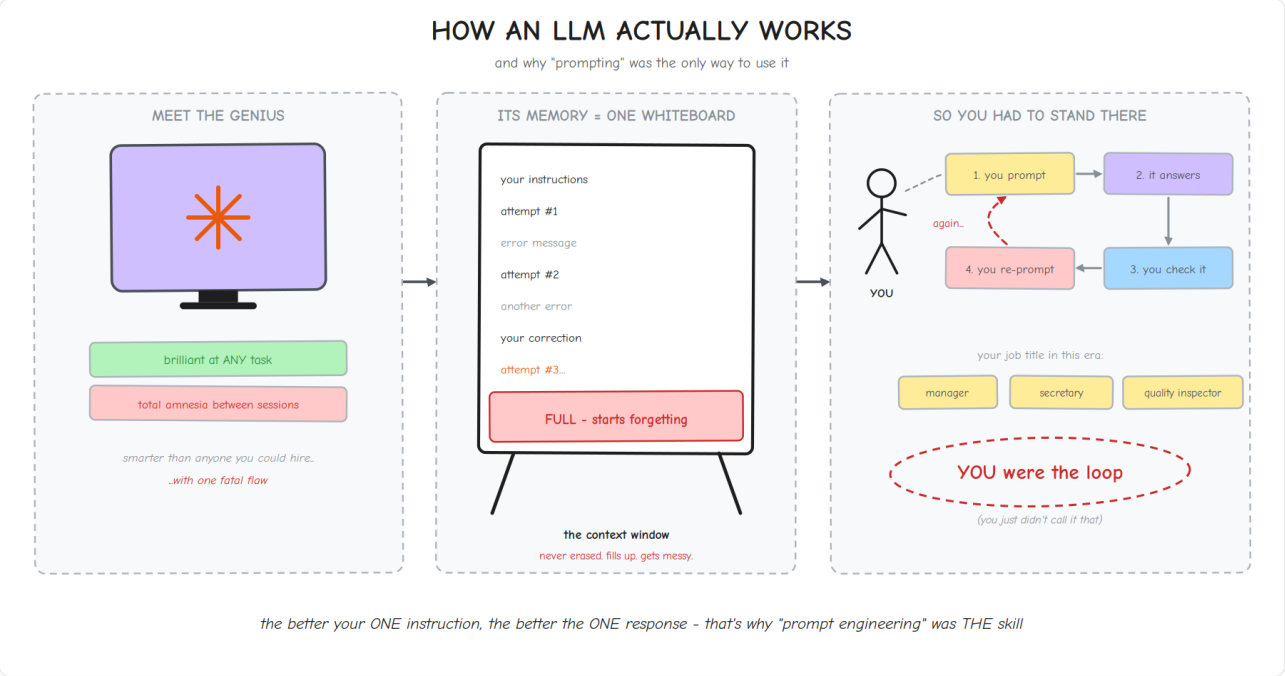

Diagram 1: a genius with amnesia and one whiteboard. That's every AI session.

Because of this, there was only ever one way to work with the genius: stand next to it. You prompt, it answers, you check, you re-prompt. Every single step travels through you. You were the manager, the secretary and the quality inspector, all at once, for every task.

Here's the reframe that changes everything: that cycle of prompt, check, re-prompt already has a name. It's a loop. You were the loop. You just never called it that.

PART 2

Then the models got smart

Remember the tricks? "Act as a world class expert." "Think step by step." "Take a deep breath." "I'll tip you $200." We all typed them, and for a while they genuinely helped. That era is over.

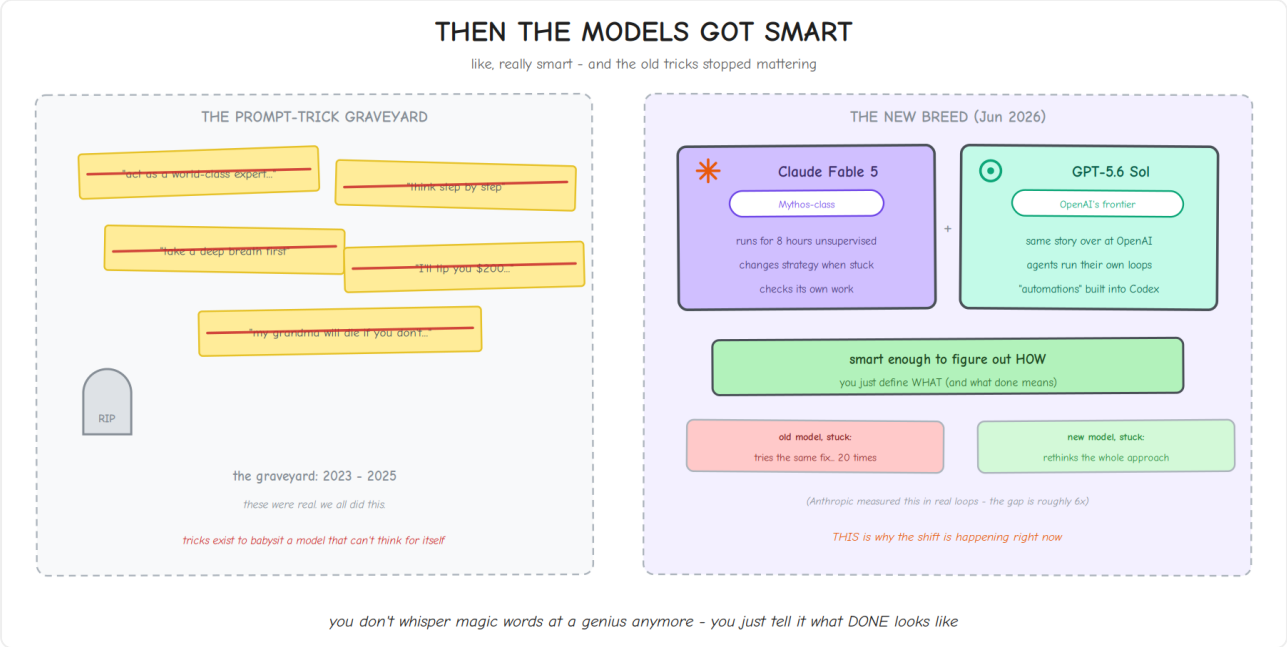

Diagram 2: the prompt trick graveyard, and what replaced it.

Tricks exist to babysit a model that can't think for itself. The new frontier models, Claude Fable 5 and GPT-5.6 Sol, don't need babysitting. They run for hours unsupervised, and when something fails they change their whole strategy instead of trying the same broken fix twenty times. Anthropic measured this in real loops: the gap between the new models and the last generation is roughly 6x.

The tricks didn't die because we got better at prompting. They died because the models stopped needing them. Once a model is smart enough to figure out HOW on its own, your only job left is to define WHAT, and what "done" means.

PART 3

The turning point

Two things changed at the same time, and together they made babysitting pointless.

First, agents got hands. Modern AI doesn't just write code, it runs it. It sees the error message. It reads the test results. The work talks back to the model directly, without you relaying anything.

Second, models got good enough to listen. When the work talks back, the new models actually use that feedback. Old models were blind: they only knew what you told them. The genius has its eyes open now.

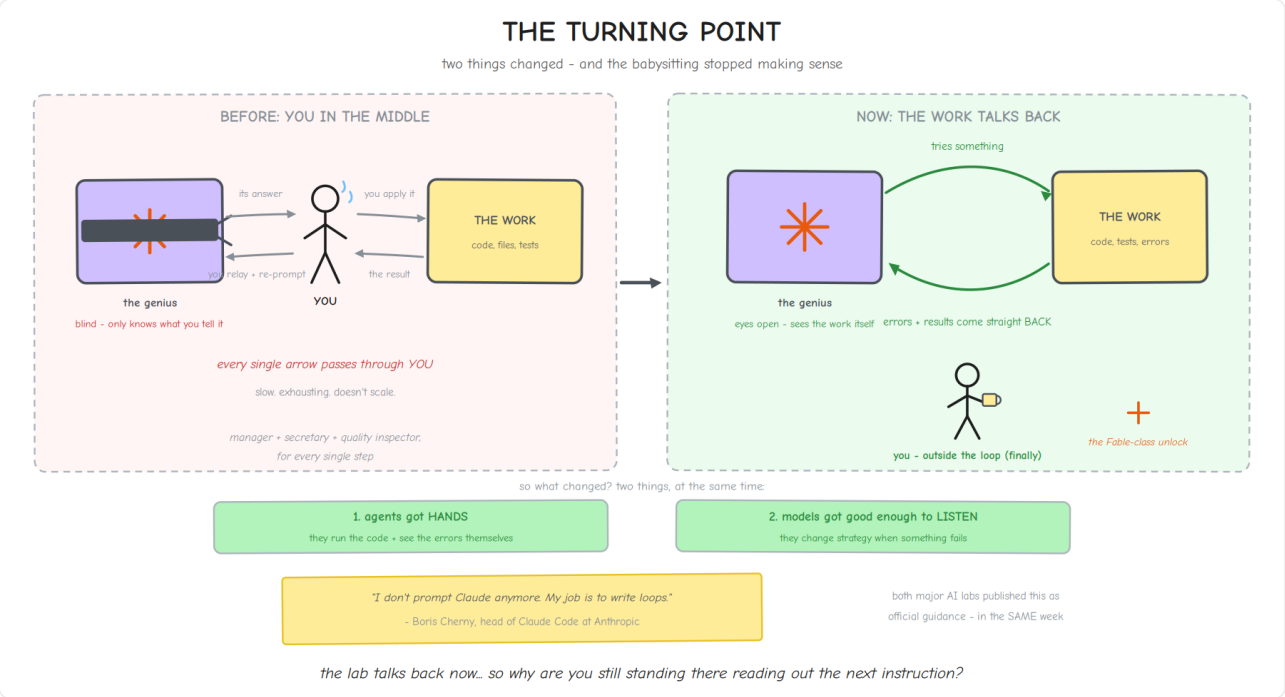

Diagram 3: before, every arrow passed through you. Now the work talks straight back to the model.

Look at the right side of that diagram. The arrows flow from the model to the work and back again, and you are standing outside the circle. That's the entire shift in one picture. The lab talks back now. So why are you still standing there reading out the next instruction?

PART 4

How to loop engineer

Stop micromanaging a laborer. Hire a contractor. You don't stand over a contractor all day. You give them four things and get out of the way.

- **The spec.** What you want built.
- **The checklist.** What "done" provably means. Not vibes. Checkable facts.
- **The inspector.** A separate model that grades the work. The builder never signs off its own work, because builders always think their work is great.
- **The budget.** A stop condition. "Or stop after 20 turns." This is your cost cap.

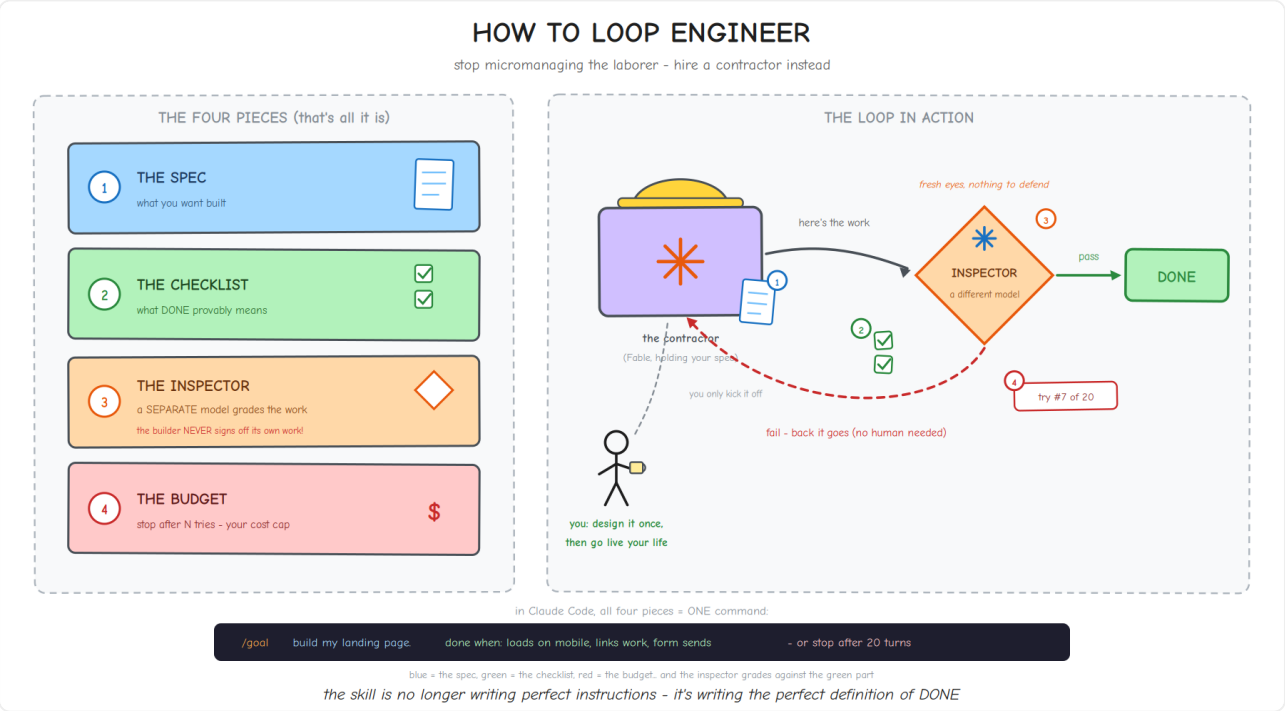

Diagram 4: the four pieces, and how they collapse into one command.

Those four pieces are the whole discipline, and in Claude Code they collapse into a single sentence. Look at the color coding at the bottom of the diagram: the blue part is the spec, the green part is the checklist, the red part is the budget, and the inspector grades against the green part.

> **The skill has changed.** It's no longer writing perfect instructions. It's writing the perfect definition of done.

PART 5

The two kinds of loop

Here's the confusion nobody clears up: everyone uses the word "loop" for two completely different machines.

The inner loop is a mission. Try, check, fix, repeat, until the finish line is provably reached. Then it stops itself. This is the `/goal` command. The grader decides when it's over, not you and not a clock.

The outer loop is a routine. It fires on a schedule: every minute, every hour, every morning. Each fire does the job, then waits for the next one. It stops only when you cancel it. This is the `/loop` command.

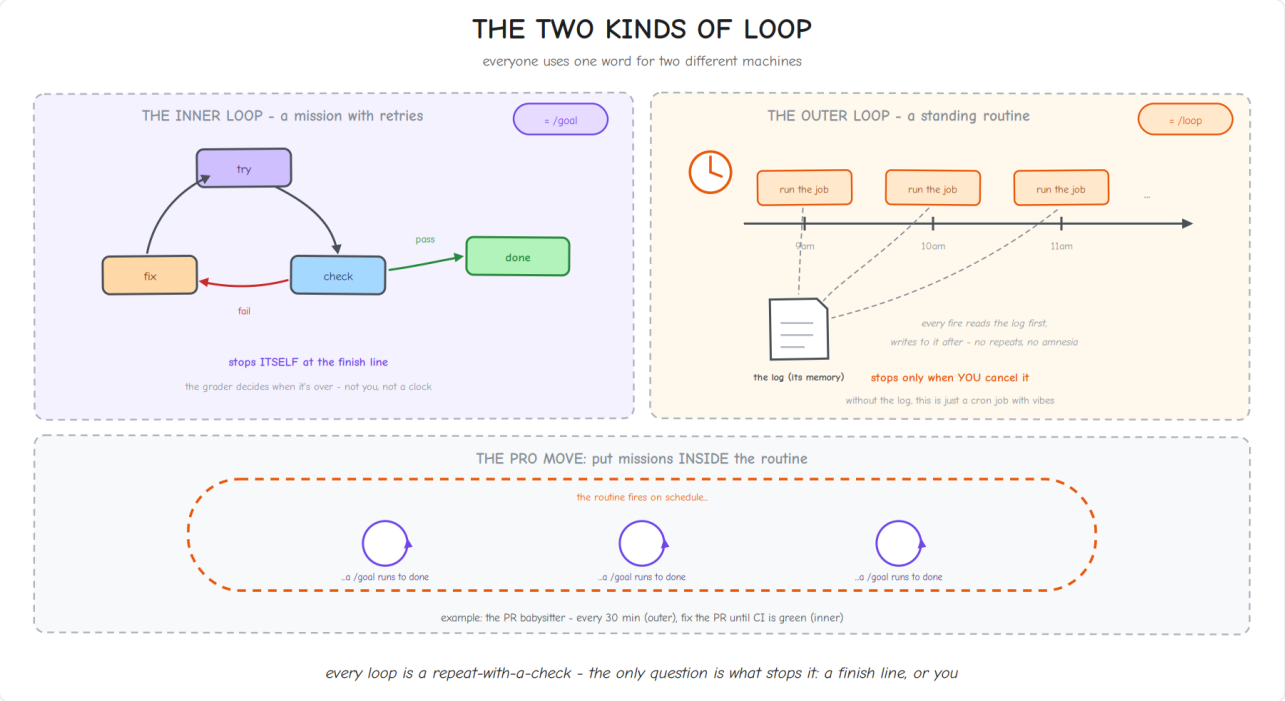

Diagram 5: a mission stops at the finish line. A routine stops when you say so.

One more thing, and it's the part most people miss. An outer loop without memory is just a cron job with vibes. Each fire starts blank, remember the amnesia. The fix is a log file: every fire reads the log first, does its work, then writes what it did. No repeats, no amnesia, and the loop genuinely gets more useful over time because its log gets richer.

The pro move is combining them. The routine fires on schedule, and each fire runs a mission until it's provably done. A routine made of missions. The famous example is the PR babysitter: every 30 minutes (outer), fix the pull request until the tests are green (inner).

> **If you remember one line:** every loop is a repeat with a check. The only question is what stops it. A finish line, or you.

PART 6

Inside a running /goal

Here's what literally happens when you press enter on a `/goal`.

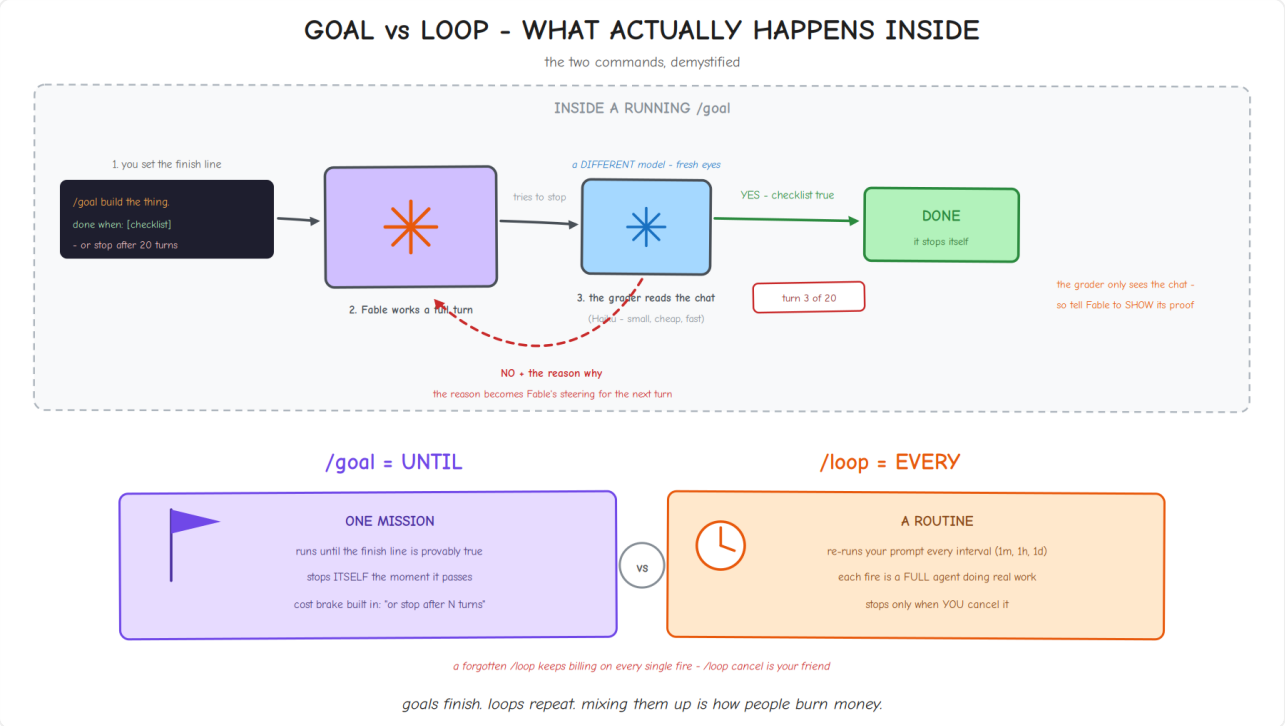

Diagram 6: the machinery. A worker, a grader, and a finish line.

- You set the finish line: the task, the "done when" checklist, and a turn cap.
- Fable works a full turn, then tries to stop.
- A grader reads the conversation and checks your checklist. The grader is a different, cheaper model (Haiku by default). Fresh eyes, nothing to defend.
- If the answer is no, the goal doesn't clear. Fable goes back to work, and the grader's reason becomes its steering for the next turn.
- If the answer is yes, the goal clears and it stops itself.

This is the real difference from a normal prompt. A prompt ends when the model thinks it's done. A goal ends when it's proven done. You've fired yourself as the quality inspector.

> **The tip almost nobody knows:** the grader can only see what's in the chat. It can't open your files. So always end your checklist with "verify this yourself and show the results". Fable then has to put the proof on screen where the grader can read it.

PART 7

The trust ladder

You wouldn't hand a day one employee the company card. The same rules apply to loops. Anthropic's own guidance defines four rungs, and you climb them one at a time.

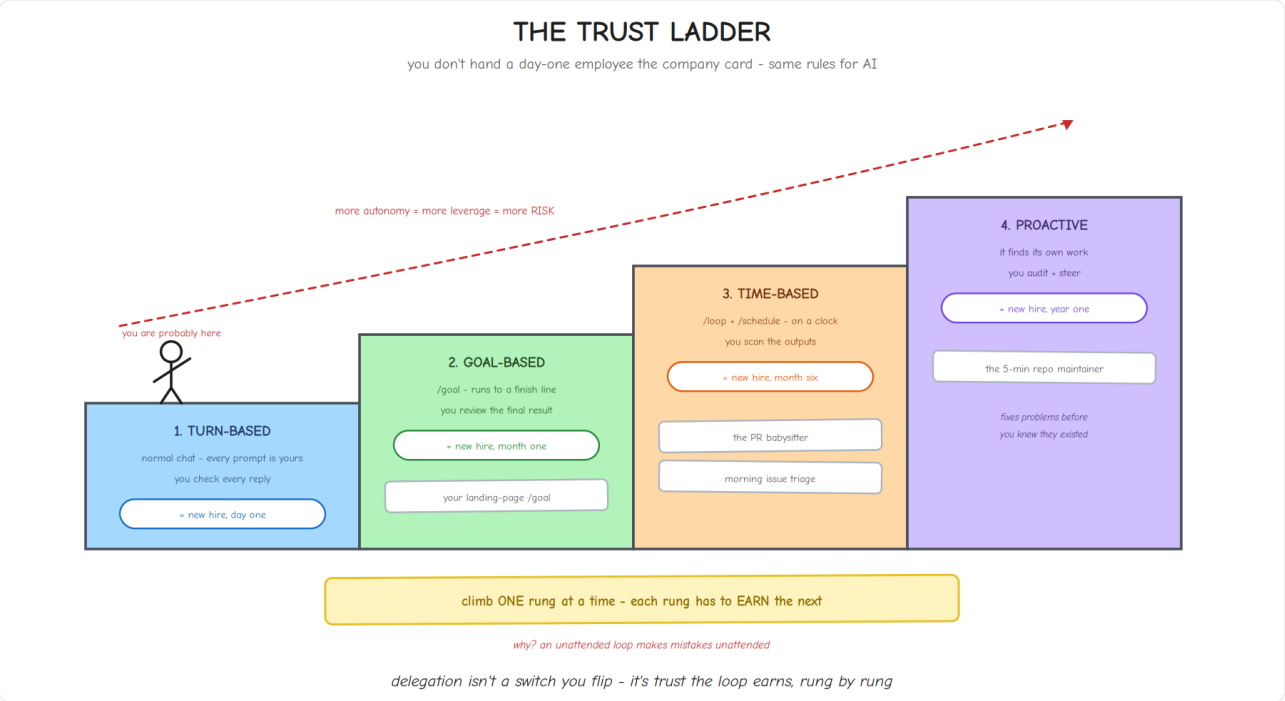

Diagram 7: four rungs of trust. Each one has to earn the next.

- **Rung 1, turn based.** Normal chat. You check every reply. It's a new hire on day one.
- **Rung 2, goal based.** `/goal` runs a mission to a finish line. You review the result. A hire in month one.
- **Rung 3, time based.** `/loop` and `/schedule` run routines on a clock. You scan the outputs. Month six.
- **Rung 4, proactive.** The loop finds its own work. You audit and steer. A hire at year one.

Why climb slowly? Because an unattended loop makes mistakes unattended. More autonomy means more leverage and more risk, at the same time. Delegation isn't a switch you flip. It's trust the loop earns, rung by rung.

PART 8

The commands, ready to paste

Everything from the video. Swap anything in [brackets] for your own details.

1. Your first mission: the website roast

```
/goal Fetch my site [YOUR URL] and audit it like a conversion consultant I'm paying $5k. Write the findings to roast.md. Done when: it lists at least 10 specific issues, every issue has a severity, the exact location on the page, and a fix I could ship today, and the top 3 are ranked by revenue impact. Every item must reference something actually on the page, no generic advice. Or stop after 12 turns.
```

2. The real automation: the competitor watcher

```
/loop 1m /
goal Fetch the latest uploads from these YouTube channels via their public RSS feeds: [read from competitor-feeds.md]. Check each against watch-log.md. For every video not yet logged, record it, and for any that overlaps topics in my-niche.md, write a 3-line brief in competitor-brief.md: their angle, their title formula, whether we should respond. Done when: every new video is logged and every overlap has a brief. Or stop after 8 turns.
```

Set the interval to 1m while testing, then 1d for real use.

3. The one word upgrade: run it while you sleep

`/loop` runs on your machine, so it stops when your laptop closes. To move the same routine to the cloud, switch to `/schedule`:

```
/
schedule every day at 6am: fetch the latest uploads from these YouTube channels via RSS, check each against watch-log.md, log anything new, and write a 3-line brief in competitor-brief.md for anything overlapping my-niche.md
```

At your desk: /loop. Asleep: /schedule. Same sentence, different address.

4. The life one: the Downloads janitor

```
/loop 1d /goal Scan my Downloads folder against janitor-log.md. For every new file: rename it to something a human can read, move it into the right subfolder (docs, images, installers, junk), flag likely duplicates, and log every action. Done when:
```

```
nothing unprocessed remains in the root of Downloads and every move is logged. Or
stop after 10 turns.
```

5. The kill switch

```
/loop cancel
```

Goals end themselves. Loops never do. A forgotten loop keeps billing on every single fire, so cancel anything you're not actively using.

Bonus: getting a channel's RSS feed

Open the channel page, view the page source, search for `channel_id`, then drop the ID into `youtube.com/feeds/videos.xml?channel_id=UC...`

Or make Claude do it:

```
Find the YouTube RSS feed URLs for these channels: [channel names]. Fetch each
channel page, extract the channel_id, build the feeds/videos.xml URL, verify each
one returns XML, and save the working list to competitor-feeds.md.
```

PART 9

Writing a definition of done that works

The whole game lives in the "done when" line. Four rules:

1. Guide with adjectives, verify with numbers. Words like "strong" and "clean" steer the worker, but the grader can't check them. The checkable rules do the enforcing: exact counts, character limits, "every test passes", "coverage above 80%". Use both layers on purpose.

2. Make it show its proof. The grader only reads the chat. End every checklist with "verify this yourself and show the results".

3. Always cap it. "Or stop after N turns" goes in every goal. It costs nothing and it's the difference between a bounded experiment and a runaway bill.

4. Ban the generic. Rules like "every item must reference something actually on the page" force specific output. It's the single best quality lever you have.

A quick test of whether your idea is a goal or a loop: if you can write "done when" as a checklist, it's a `/goal`. If the trigger is time, it's a `/loop`. If it's a schedule that runs checklists, you've found the pro move.

PART 10

The library: missions

Use these as starting points. Change the file names and details to fit your world, keep the structure.

Area	Mission ideas (/goal)
Content	20 title options, all under 60 characters, 5 with a number • tighten a script under a word count without losing any callouts • turn one transcript into 10 tweets, 3 posts and a newsletter section, each with a hook • 15 thumbnail text options, max 4 words, no word shared with the title • rewrite an old post in your current voice, matched against 3 examples
Marketing	build a landing page that renders clean on mobile, all links working, form validating • a 5 email welcome sequence, each under 200 words with one CTA • 12 ad variants grouped by angle, each under the character cap • an outline for every keyword missing from your content map • a competitor teardown where every claim cites something on their actual site
Business	turn meeting notes into actions, every one with an owner, deadline and success criterion • write an SOP where every step names the tool and a new hire could follow it cold • clean a messy client list: no duplicates, consistent formatting, gaps flagged • draft the weekly team digest: wins, blockers, decisions needed, every section non empty
Sales	enrich every lead in a CSV with size, niche and a personalization hook, no row left empty • write a first line for every lead, under 25 words, no two openings alike • score the pipeline and rank the 10 deals most likely to close, with a reason each
Finance	categorize every transaction, flag anything ambiguous, totals must match the source file • produce a burn and runway snapshot with the top 5 growing cost lines • reconcile invoices against payments and list every mismatch
Coding	refactor a module: every test passes, no function over 40 lines, zero type errors • raise test coverage above 80% • build a working game or tool and verify every feature yourself before finishing • hunt one specific bug until the failing test passes
Personal	a weekly review from your daily notes: shipped, slipped, lessons, next week's top 3 • distill every article on your reading list into 5 bullets and a "so what" • plan a trip where every day has a booked-checkable item and a backup

PART 11

The library: routines

Routines earn their keep when the input changes on its own clock: the web, your folders, other people. Every routine here keeps a log file. That log is the memory, and it's the difference between an automation and a cron job with vibes.

Area	Routine ideas (/loop or /schedule)
Watching the world	competitor uploads via RSS, briefed daily • price watch on anything with a public API, flag moves over a threshold • news scan on your niche keywords each morning • your own channel stats, logged daily so trends show up
Inbound streams	a drops folder: every new file summarized and logged • new transcripts turned into a content queue in your voice • new bug reports labeled by severity with a repro attempt on anything critical • form submissions triaged into act now, later, ignore
Housekeeping	the Downloads janitor: rename, sort, flag duplicates • a project folder audit: anything unnamed, misfiled or stale gets flagged weekly • log rotation: archive anything older than 30 days
Standing briefs	a 6am morning brief from your calendar, tasks and notes: top 3 priorities and one thing to say no to • a Friday week-in-review digest • a Sunday content pipeline status: what's scripted, filmed, edited, stuck
Dev routines	the PR babysitter: every 30 minutes, fix CI and address comments until green • nightly test suite fixer • daily dependency and security scan with a one line verdict

> **The combined pattern, one more time.** Routine on the outside, mission on the inside: `/loop 1d /goal [job]. Done when: [checklist]. Or stop after N turns.` The clock, the mission, the finish line and the memory, in one sentence. That's loop engineering.

PART 12

The rules to remember

1. You were always the loop. Now you design it instead of being it.
2. A prompt ends when the model thinks it's done. A goal ends when it's proven done.
3. The builder never grades its own work. Fresh eyes or it doesn't count.
4. Goals are UNTIL. Loops are EVERY. Mixing them up is how people burn money.
5. A loop without a log is a cron job with vibes. The magic is the file it keeps.
6. Every goal gets a turn cap. Every loop gets reviewed or cancelled.
7. Guide with adjectives, verify with numbers.
8. /loop runs at your desk. /schedule runs while you sleep.
9. Climb one rung at a time. An unattended loop makes mistakes unattended.
10. Delegate the mission, never the judgment. What ships is still on you.

Set it once. It runs forever.

Loops make the doing nearly free, which means judgment becomes the scarce resource. The people who win with this aren't the ones who run the most loops. They're the ones who write the sharpest definitions of done, and who still check what ships.

Start with one. The website roast takes five minutes and will probably pay for itself today. Then pick one job you did manually this week and turn it into a routine. That's the whole path.
