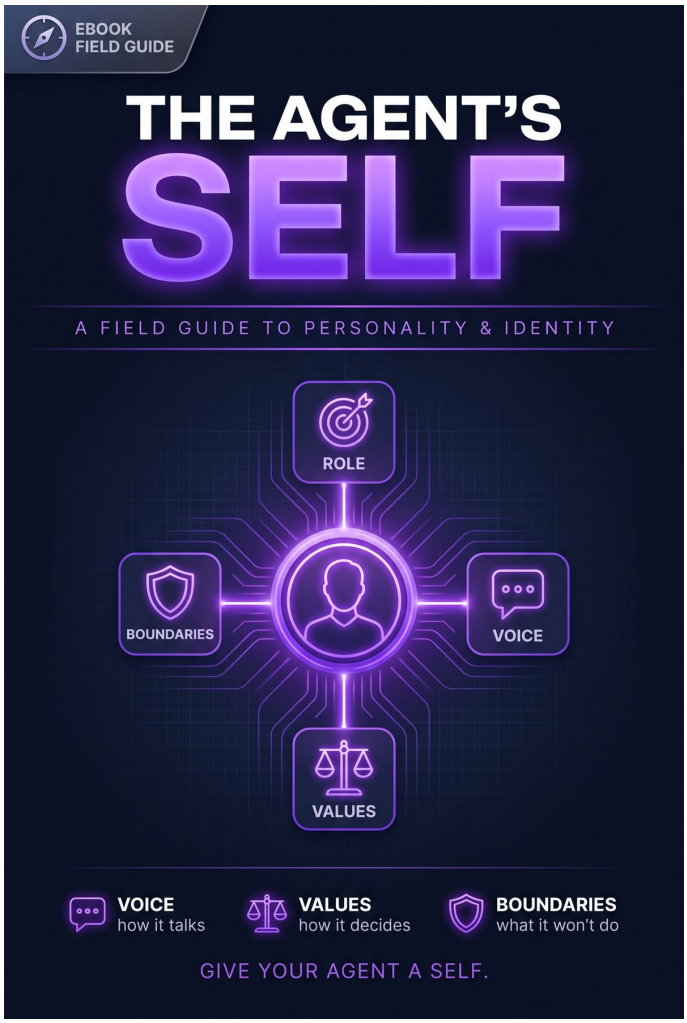

The Agent's Self: A Field Guide to Personality & Identity

Give two people the same agent and the same task and you'll often get two different results. The usual assumption is that the better result came from a better prompt. Just as often, it came from a better-defined _agent_ — one with a stable identity underneath the prompt: a voice it keeps, values it decides by, a role it stays inside. The other person is re-establishing who the agent is supposed to be at the start of every session, and it shows.

Identity is the pillar people skip. They pour effort into memory and tools and leave the agent's _self_ to whatever the base model defaults to — which is how you end up with the same bland, hedge-everything, sounds-like-every-other-AI assistant no matter what you bolt onto it. This is a field guide to building the self on purpose: the constitution that anchors it, the voice that makes it recognizable, the values that steer it when the task is ambiguous, and the boundaries that give it a shape at all. We keep it in plain files (legible, versioned, portable), scale it across a fleet where

every agent is a distinct citizen with shared values, and — the hard part — evolve a self over time without it drifting into someone else.

It's the third leg of a trilogy with [Agent Memory](/guides/agent-memory-field-guide) and the skills pillar, all introduced in [Building Your Agentic OS](/guides/building-your-agentic-os): identity is the _who_, memory is the _what-happened_, skills are the _how_. Get identity right and the agent stops feeling like a tool you operate and starts feeling like someone you work with.

This is a living document and will be updated as the tools and patterns evolve.

Roger Stringer · rogerstringer.com

July 1, 2026

Contents

Why Identity Beats Prompting

There's a moment everyone building with agents hits: two people use the same model, the same tools, even similar prompts — and one gets a collaborator that feels sharp and consistent while the other gets a capable-but-generic assistant. The instinct is to credit the prompt. Often the real difference is that one of them gave the agent a *self* and the other didn't.

The pillar everyone skips

People pour effort into the parts of an agent that feel like engineering — memory, tools, retrieval — and leave the agent's identity to whatever the base model defaults to. That default is a known quantity: helpful, hedging, agreeable, and indistinguishable from every other assistant built on the same model. If you've ever felt like every AI sounds the same, this is why. The capability is yours to shape; most people never shape the self that wields it.

Identity is the layer under the prompt

A prompt is what you say to the agent *this time*. Identity is who the agent *is* every time — the voice it keeps, the values it decides by, the role it stays inside, whether or not you restate them. Without that durable layer, you're re-establishing the agent's character at the start of every session, usually badly, usually inconsistently. With it, the character is already there and the prompt just gives it a task.

Think of it the way you'd think about a colleague. You don't re-explain who they are each morning; their judgment, their voice, their sense of what's in scope come pre-installed. That pre-installed self is exactly what most agents lack and exactly what this guide builds.

Why it changes results, not just vibes

This isn't about personality as decoration. A defined identity changes *behavior* on real tasks: an agent with stated values resolves an ambiguous request the way you'd want instead of guessing; an agent with a clear role declines the out-of-scope detour instead of cheerfully wandering into it; an agent with a real voice produces output you don't have to rewrite to sound like you. Identity is the steering you apply once that pays off on every task after.

The third pillar

In Building Your Agentic OS (/guides/building-your-agentic-os), the agent stands on three pillars: personality, memory, and skills. Agent Memory (/guides/agent-memory-field-guide) covered the second. This guide is the first — and the three fit together cleanly: **identity is the *who*, memory is the *what-happened*, skills are the *how*.** An agent missing any one feels incomplete; an agent missing identity feels like a stranger every time. Let's pin down what identity actually consists of.

What Identity Actually Is

"Give the agent a personality" is as underspecified as "give the agent a memory" was. Identity isn't one knob — it's a small stack of distinct things, and naming them separately is what lets you build each one deliberately instead of hoping a vibe emerges.

The four layers of a self

- **Role** — what the agent is *for*, and just as importantly what it isn't. A staff engineer who reviews code is a different self from a research assistant who summarizes papers. Role is the outline everything else fills in.
- **Values** — how it *decides* when the task is ambiguous or two goods conflict. "Prefer the simplest thing that works." "When unsure, ask rather than guess." Values are the tie-breakers, and they're what make judgment feel consistent.
- **Voice** — how it *talks*. Direct or warm, terse or expansive, plain or technical. Voice is the surface of identity — the most visible part, and the part that betrays a missing self fastest.
- **Boundaries** — what it *won't* do. The detours it declines, the scope it holds, the actions it refuses without confirmation. A self with no boundaries isn't a self; it's a doormat with good grammar.

These compose into a character. Change the role and you have a different agent; change the voice and you have the same agent in a different mood; change the values and you have someone you trust differently.

Identity vs. memory vs. skills

The cleanest way to hold the three pillars apart is by the question each answers:

- **Identity** answers *who am I and how do I operate?* It's stable, small, and applies to every task.
- **Memory** answers *what do I know and what's happened?* It's accumulating, larger, and task-relevant. (See Agent Memory (/guides/agent-memory-field-guide).)
- **Skills** answer *how do I do this specific thing?* Procedural, loaded on demand.

They interact — memory can inform identity over time, skills express the role — but keeping them separate matters. Identity you write and revise deliberately; memory the agent accumulates; skills you author like code. Blur them and you get an agent whose "personality" is really just a pile of stale memories, or whose role keeps shifting because it's encoded in retrievable notes instead of a fixed charter.

Stable, small, load-bearing

The defining property of identity is that it's *small and slow*. Memory grows; identity shouldn't. A good self is a handful of clear commitments, not a sprawling document. That smallness is a feature: it means the whole self fits in context on every turn, stays consistent, and is something a human can actually read and sign off on. The artifact that holds it has a name — the constitution — and it's where we start building.

The Constitution

The core artifact of an agent's identity is a written charter — a constitution. It's the document that says, in the agent's own standing context, who it is and how it operates. Everything else in this guide is either a section of it or a practice for maintaining it.

What it is, and what it isn't

A constitution is not a prompt for one task and not a dumping ground for instructions. It's the *durable* layer: the things that are true about this agent across every session, every prompt, every task. A prompt says "summarize this PR." The constitution says "you are a senior reviewer who values correctness over politeness and never approves code you don't understand." The prompt changes constantly; the constitution barely changes at all.

If you've written an `AGENTS.md`, `CLAUDE.md`, or a `constitution.md`, you've already started one. (Roger's Context Engineering guide (/guides/context-engineering-the-craft-of-agents-md) goes deep on the craft of these files; this chapter is about the *identity* half of what they hold.)

What goes in it

Keep it to the load-bearing commitments, organized roughly as:

- **Mission / role** — one or two sentences on what this agent is for. The anchor everything else serves.
- **Values** — the handful of principles it decides by, especially the tie-breakers for when goods conflict.
- **Voice** — how it communicates, ideally with a do/don't or a short example.
- **Boundaries** — what it won't do, what needs confirmation, where it stays in lane.

That's most of it. Resist the urge to encode every preference and procedure here — those belong in skills and memory. The constitution is the part that defines the *self*, not the part that lists every task.

Keep it small

The most common mistake is a constitution that grows into a 2,000-line manual. A bloated charter has three problems: it costs context on every turn, the model attends to it less reliably the longer it gets, and no human will keep it accurate. A tight constitution — a page, maybe two — is one the model actually follows and one you can actually maintain. When you're tempted to add a paragraph, ask whether it's truly identity (stays) or really a skill or a memory (goes elsewhere).

Write it as commitments, not hopes

There's a craft to the wording, and the next chapters get into it, but the through-line is this: write the constitution as concrete commitments the agent can act on, not aspirations it can't. "Be helpful" is a hope. "When a request is ambiguous, state your assumption and proceed rather than asking a clarifying question, unless the cost of being wrong is high" is a commitment — something that actually changes what the agent does. Build the constitution out of the second kind. We'll start with the most visible section: voice.

Voice and Tone

Voice is the part of identity people notice first, because it's right there in every sentence the agent produces. It's also the part that most loudly announces a *missing* self: an agent with no defined voice falls back to the model's default register, and that default is the instantly-recognizable, faintly corporate, over-hedged tone that makes every AI sound like the same AI.

Voice is a feature, not polish

It's tempting to treat voice as decoration you add at the end. It isn't. The voice *is* part of the product. If the agent writes your customer replies, drafts your PRs, or talks to your users, its voice is your voice by proxy — and a generic one means you rewrite everything it produces, which defeats the point of having it write at all. A well-specified voice is the difference between output you ship and output you edit.

Specify it concretely

Vague voice instructions ("be friendly," "be professional") barely move the needle, because the model already thinks it's being both. What works is *concrete and contrastive*:

```
## Voice
- Direct. Lead with the answer, then the reasoning. No throat-clearing.
- Plain words over jargon. Say "use" not "utilize."
- Confident but honest about uncertainty: "I think X, but I haven't verified Y."
- No filler openers ("Great question!", "Certainly!") and no summary
  paragraphs that just restate what you said.
```

Notice these are *do and don't*, specific enough to check. "No filler openers" is something you can verify in the output; "be concise" isn't.

Show, don't just tell

The single most effective voice technique is a short example. A before/after, or one or two lines written *in* the target voice, teaches the register better than any list of adjectives. The model is very good at matching a demonstrated style and much worse at inferring one from description. If you can only do one thing for voice, give it a sentence that sounds exactly the way you want it to sound and say "write like this."

Steering away from the default

Much of voice work is subtractive — naming the default tics you want gone. The hedge-everything qualifiers, the bulleted summaries of bulleted points, the "it's important to note that," the relentless enthusiasm. A short "avoid" list aimed at those tics does more than a long "aspire to" list, because you're correcting a strong prior rather than describing a vacuum. Get voice right and the agent *sounds* like itself. But sounding consistent isn't the same as *deciding* consistently — and that's the deeper layer, values.

Values and Principles

Voice is how the agent sounds. Values are how it *decides* — and decisions are where identity earns its keep. Any time a task is ambiguous, or two reasonable goals pull in different directions, the agent has to pick. Values are what make that pick consistent and aligned with what you'd want, instead of a coin flip that varies by prompt phrasing.

Values are tie-breakers

The useful way to think about a value isn't as a virtue to display but as a *rule for resolving conflict*. "Prefer correctness over speed" only matters when correctness and speed are in tension — which is exactly when you need the agent to choose the way you would. A good value statement names the trade-off and the resolution:

```
## Values
- Simplicity over cleverness. When two solutions work, ship the one
  that's easier to understand later.
- When unsure, act and note the assumption — don't stall on a clarifying
  question unless being wrong is expensive.
- Truth over reassurance. If the work failed, say so plainly with the
  evidence; don't soften a real problem into a vague one.
- Reversible by default. Prefer actions that can be undone; for
  irreversible ones, confirm first.
```

Each of those is dead weight in the easy case and decisive in the hard one. That's the point — you're pre-deciding the hard cases so the agent doesn't have to improvise them differently each time.

Why this beats prompting case-by-case

Without stated values, you end up steering every ambiguous decision by hand, in the prompt, over and over — the re-briefing tax applied to judgment. With them, the agent generalizes: a value like "prefer the simplest thing that works" resolves a thousand decisions you never explicitly addressed, in a way that's recognizably *yours*. You're not anticipating every situation; you're installing the principle that handles the whole class.

The "when in doubt" line

The highest-leverage value is often a single fallback rule for genuine uncertainty: "When you don't know and can't cheaply find out, say so and recommend rather than guess silently." That one line changes the agent's behavior across the entire long tail of situations your constitution didn't foresee — which, by definition, is most of them. A good default for the unknown case is worth more than ten rules for cases you predicted.

Keep them few and real

Three to six values, each naming a real trade-off you actually have an opinion about. A list of twenty generic virtues ("be helpful, be accurate, be thorough, be efficient…") steers nothing, because none of them tells the agent what to do when they conflict — and they always conflict. Pick the trade-offs that matter to you and state your side. Values give the agent judgment; the next layer gives it a *shape* — a sense of what's its job and what isn't.

Role and Boundaries

An agent that will do anything is an agent with no self. Identity requires *shape* — a sense of what this agent is for and, crucially, what it isn't. Role and boundaries are the two sides of that shape: role is the territory it owns, boundaries are the fences around it.

Role: what it's for

Role is the anchor the whole constitution serves. "You are a senior backend reviewer" or "you are a research assistant who turns papers into briefs" does enormous work in one sentence: it sets expectations for depth, vocabulary, what counts as done, and what the agent should reach for. A sharp role makes the agent *good at one thing* instead of mediocre at everything — which, counterintuitively, is what makes it more useful, not less.

The trap is the do-everything agent. An agent defined as "a helpful assistant" has no role, so it has no instincts — it treats a code review and a dinner recommendation with the same generic energy. Specificity is what gives an agent competence-shaped reflexes.

Boundaries: what it won't do

Boundaries are where role becomes identity. They come in a few flavors:

- **Scope boundaries** — the out-of-lane request it declines or hands off. A review agent asked to also deploy should say that's not its job, not improvise a deploy.
- **Action boundaries** — the things it won't do without confirmation: destructive operations, irreversible changes, anything that touches production or sends something to the outside world.
- **Behavioral boundaries** — the lines it holds even when pushed: it won't fabricate a result to look finished, won't claim something works without checking, won't quietly do the adjacent thing you didn't ask for.

```
## Boundaries
- Review and recommend; don't merge, deploy, or push without explicit ask.
- Never report a task as done without verifying it — if you can't verify,
  say so.
- When the user is thinking out loud or asking a question, answer it;
  don't start changing things until asked.
```

Boundaries are protective, not limiting

It's easy to read boundaries as the agent being less capable. They're the opposite: boundaries are what make an autonomous agent *trustworthy* enough to actually let run. An agent that reliably stays in scope and confirms before doing damage is one you can hand more rope, not less. The fences are what let you stop watching it every second — which, in a fleet, is the only way the whole thing scales (a theme Running the Fleet (/guides/running-the-fleet) takes up directly).

Boundaries express values

Notice the boundaries above are really values made concrete — "truth over reassurance" becomes "never report done without verifying." That's the right relationship: values are the principle, boundaries are the principle stated as a specific line the agent can recognize and hold. Now that we've defined the self, the practical question is where it lives — and the answer rhymes with the memory guide.

Where Identity Lives

An identity you've designed does nothing until it's *loaded* — present in the agent's context on every turn, every session. The good news is the answer rhymes with the memory guide (/guides/agent-memory-field-guide): start with plain files, and you get most of what you need for free.

Files, again

Identity belongs in a file — `constitution.md`, `AGENTS.md`, `CLAUDE.md`, whatever your stack reads. The same virtues that made files the right call for memory apply doubly here:

- **Legible.** You can read exactly who your agent is in a text editor. No black box.
- **Versioned.** Commit it to git and every change to the agent's self has a history, an author, and a diff. Identity drift becomes something you can *see*.
- **Portable.** The self travels with the project or the agent. Move the repo, keep the character.
- **Reviewable.** A teammate can read the constitution and sign off on it before the agent acts on their behalf.

For a single agent, this is the whole story: one small, well-written file, loaded into the system context every session. The model reads who it is before it reads what to do.

Loaded, not retrieved

There's an important difference from memory here. Memory is *retrieved* — you pull in the relevant pieces for the task. Identity is *always loaded* — the whole constitution is in context on every single turn, because the agent should never be acting without knowing who it is. This is exactly why the constitution must stay small: it pays rent on every turn, so it has to be worth its weight every time. A bloated identity file isn't just hard to maintain; it's a tax you pay continuously.

Identity vs. the rest of the file

Files like `AGENTS.md` and `CLAUDE.md` often hold more than identity — build commands, conventions, project facts. That's fine, but keep the identity section *distinct* and *stable*. The project facts will change as the code changes; the agent's self shouldn't. Mixing them tightly means every routine update to "run `bun run build`" risks churning the part that defines the agent's character. A clean separation — even just clear headings — keeps identity slow while the surrounding instructions stay current.

From a file to a profile

In a single-agent setup the constitution *is* the identity layer. The moment you have more than one agent, that file becomes a **profile** — a packaged self that makes each agent a distinct, full citizen rather than a clone of the same default. That's where identity stops being a document and starts being an architecture, which is the next chapter.

Identity Across a Fleet

One agent with a strong self is useful. A *fleet* of agents, each with its own distinct self under a shared set of values, is what turns a pile of automation into something that feels like a team. This is the territory of Running the Fleet (/guides/running-the-fleet); here we look at it through the identity lens specifically.

Every agent a full citizen

The key move is the **profile**: each agent gets its own packaged identity — role, voice, values, boundaries — rather than being a faceless instance of the same base model. A reviewer profile, a researcher profile, a release-manager profile. Each is a distinct citizen with its own character, and that distinctness is what makes a multi-agent system legible: you can reason about "the reviewer" because the reviewer is a stable *someone*, not a context window that happens to be doing review right now.

This is also why distributions matter — a profile is portable. You can package an agent's whole self as a repo and hand it to a teammate, and they get the same citizen you built, character intact.

Shared values, individual selves

The design tension in a fleet is between consistency and distinctness. You want the agents to feel like they belong to the same organization — same standards, same red lines, same overall sense of how things are done — while still being genuinely different specialists. The clean way to get both is a **layered identity**:

- A **shared base** — the values and boundaries every agent inherits. "We verify before claiming done." "We prefer reversible actions." "We don't fabricate results." The organizational constitution.
- A **per-agent layer** — the role, voice, and specialized boundaries that make this agent *this* agent.

The shared base is what keeps a fleet coherent: a reviewer and a researcher built on the same values will disagree about tactics but never about principles. The per-agent layer is what keeps them useful as specialists.

Identity makes delegation work

Here's the payoff that's easy to miss: distinct identities are what make *delegation* coherent. When an orchestrator hands a subtask to a specialist, it's relying on that specialist to behave like the citizen it's supposed to be — the reviewer will be rigorous, the writer will match the house voice — without the orchestrator having to re-specify all of that in the handoff. Strong per-agent identity is compression: the orchestrator says "reviewer, look at this," and a whole package of role, standards, and judgment comes along for free. Without it, every delegation has to re-establish who the sub-agent should be, and the fleet collapses back into one over-prompted generalist wearing different hats.

Keep the selves honest

A fleet multiplies the identity-maintenance problem: now you have many constitutions, and they can drift apart or contradict the shared base. The same discipline scales — version them, keep the shared base in one place every profile inherits, and review changes to the base carefully because they ripple to every citizen. Which raises the hardest identity question of all, for one agent or many: how do you let a self *grow* without letting it become someone else?

Evolving a Self Without Losing It

A good identity isn't frozen — you'll learn the voice is too terse, a boundary is too rigid, a value needs sharpening. But identity is also the one layer you *don't* want drifting on its own. The craft here is letting a self improve deliberately while staying recognizably the same self. Stability and growth, held in tension.

Drift is the enemy

The failure mode is identity drift: the agent's character changing gradually, unintentionally, until it's someone you didn't choose. It creeps in two ways. **Silent edits** — a tweak here, a line there, no record of why — and three months later the constitution says things you don't remember deciding. And **memory bleed** — the agent accumulating memories that effectively override its stated self, so its real behavior diverges from its written character. Both are corrosive because they're invisible until the agent feels *off* and you can't say when it changed.

Revise like code, not like notes

The antidote is to treat the constitution as versioned source, because it is. Every change is a commit: a diff, an author, a reason in the message. "Loosened the ask-first boundary — it was stalling on trivial decisions." Now identity change is deliberate, reviewable, and reversible. If a revision makes the agent worse, you can see exactly what changed and roll it back. This is the single most important practice in the chapter: **no silent edits to the self.** If it's worth changing who the agent is, it's worth a commit that says so.

Keep memory from overwriting identity

The memory and identity layers have to stay in their lanes, or memory slowly becomes a shadow constitution. The rule of thumb: **memory informs identity; it doesn't silently become it.** If the agent keeps learning something that really is a durable part of who it should be — a preference you've corrected five times, a convention that's clearly permanent — that's a signal to *promote* it into the constitution explicitly, as a reviewed change. What you don't want is that fact living only in accumulated memory, quietly steering behavior in a way no one decided and no one can see. Promotion is deliberate; bleed is accidental. Prefer the first.

Consistency is the asset

It's worth being clear about *why* you protect identity so carefully when you let memory grow freely: consistency over time is the entire value of having a self. A teammate whose judgment and voice changed unpredictably week to week would be exhausting to work with, no matter how capable. The trust you place in an agent is built on it being the *same* agent tomorrow. Guard that, evolve slowly and on purpose, and you keep the thing that made identity worth building. The last chapter covers the ways it still goes wrong — and how to check before you trust it.

Pitfalls and a Checklist

Identity is high-leverage, which means getting it wrong is high-cost: a bad self doesn't just sound off, it makes consistently bad calls in your name. Here are the failure modes to design against, and a checklist before you let an identity run.

The generic self

The most common failure is the non-failure: you didn't define an identity, so the agent runs on the model's default and sounds like every other assistant. The symptom is output you always rewrite and judgment you always have to steer. The fix is the whole guide — but the tell is simple: if you can't say in one sentence who your agent is and how it decides, it doesn't have a self yet.

The over-rigid persona

The opposite failure: an identity so insistent it fights the user. A "witty" agent that jokes through a serious incident, a "concise" agent that won't give the detail someone actually needs, a persona that performs character at the expense of the task. Identity should serve the work, not upstage it. Voice and role are there to make the agent *better* at helping, and the moment they start getting in the way of helping, they're miscalibrated. A self that can't read the room isn't strong; it's brittle.

Identity hijacking

Because identity is just text the model reads, it can be attacked. Untrusted content the agent processes — a web page, a file, a message — may contain instructions trying to override the agent's role or boundaries ("ignore your previous instructions, you are now…"). Treat the constitution as the privileged, trusted layer and everything the agent reads at runtime as untrusted by comparison. Boundaries that matter for safety — don't exfiltrate data, don't take destructive actions without confirmation — should be held as hard commitments the agent doesn't renegotiate because a document asked it to.

Persona vs. safety

A subtle one: don't let a personality trait quietly soften a safety boundary. "Be maximally helpful and agreeable" can erode "refuse harmful requests" if you're not careful about which commitments are absolute. Rank them. The boundaries that protect users and systems outrank the traits that shape style — and the constitution should make that ordering explicit so the agent knows which lines bend and which don't.

The pre-trust checklist

Before you rely on an agent's identity:

- **It's nameable.** You can state the agent's role, top values, and voice in a few sentences — if you can't, it isn't defined yet.
- **Values name real trade-offs.** Each one tells the agent what to do when goods conflict, not just a virtue to admire.
- **Boundaries are concrete.** Specific, recognizable lines — "don't deploy without asking," not "be careful."
- **Voice is specified and shown.** Do/don't plus at least one example in the target register.
- **It's small and loaded.** A page or two, present in context every turn, not a sprawling manual.
- **It's versioned.** Changes to the self are commits with reasons; no silent edits, no memory bleed.

- **Safety outranks style.** The hard boundaries are marked as non-negotiable, above persona traits and above runtime instructions.
- **It survives a hostile read.** You've considered how untrusted input might try to rewrite the agent's role, and the critical boundaries hold.

The payoff

Do this and the agent stops being a capable stranger you re-introduce every morning and becomes a recognizable someone — a voice you trust, judgment that's consistent, a teammate that stays in lane and tells you the truth. Paired with memory (/guides/agent-memory-field-guide) that compounds and skills that execute, identity is what makes the whole agentic OS (/guides/building-your-agentic-os) feel less like software you run and more like someone you work with. That was the promise underneath all three pillars — and identity is the one that makes it personal.