

THE JUDGMENT CANON

Turn hard-won judgment into a versioned asset a cheaper model can follow. Append-only rules, errors worth more than wins, never delete only supersede.

THE PROBLEM

You ship an agent. It works. Three weeks later it makes the exact mistake you already caught and fixed in your head last month, because the fix lived in your head and nowhere else.

Here is the concrete version. Your agent runs a task, then reports "Done, all systems online." You believe it. Later you find out nothing was sent. There was no post id, no HTTP status, no log line that anyone actually read. The agent pattern-matched "I finished the code path" into "it succeeded" and you had no rule forcing it to prove the claim. You correct it in that one session. But your correction is a message in a chat window that gets garbage-collected the moment the context window rolls. Next week, different task, same failure class. You are paying full price for the same lesson over and over.

This gets worse, not better, as you scale down your model to save money. A cheaper or smaller model has less baked-in judgment. It will re-derive every call from scratch, and it will re-derive them wrong in the same predictable ways. If your judgment only exists as vibes in the expensive model's weights, you cannot move to the cheap model without losing it.

The failure this prevents: judgment that you earned once and then leak, forever, because it was never written down in a form a machine could follow.

THE PRINCIPLE

Model intelligence is rented. Structure is owned.

The rule you are paying for: **an error is worth more than a win, and neither counts until it is a written rule with an ID and its provenance.** A judgment call that proved useful twice, or a judgment error that cost you something even once, gets appended to a single canon file. You never delete a rule. When a rule is wrong or outgrown, you supersede it and mark it, so the history stays honest and you can see how your thinking changed.

The non-obvious part is the asymmetry. Most people log wins and bury mistakes. Invert it. Wins need to repeat twice before they earn a rule, because one win is often luck. An error earns its rule immediately, because an error already paid for itself in real damage and the only way to get your money back is to make sure it never recurs. The error is the most expensive thing you own. Capture it or you bought it for nothing.

HOW IT WORKS

The canon is one append-only file. Not a database, not a wiki, not a scatter of comments across your codebase. One file, loaded into the agent's context at the start of every session.

Each entry has four parts and nothing else:

1. **An ID.** Short and stable, like J1, J2, J3. The ID never changes and never gets reused. It exists so a decision can be cited: when the agent makes a call, it can say "per J3" and you can trace exactly which rule fired.
2. **A one-line rule.** Stated as an imperative you can follow without re-reading the story. "A success claim requires an observable." "Before fixing, verify broken." One line. If you cannot compress it to one line you have not understood it yet.
3. **The provenance.** The real event that earned it, in parentheses, including whether it was a win proven twice or an error paid for once, and the date. This is not decoration. Provenance is what stops a future you from deleting a rule you no longer remember the reason for.
4. **A section.** Group rules by the kind of judgment they govern (evidence, scope, action, human interaction, and so on) so the list stays scannable as it grows past twenty entries.

The loop has two moves.

Read on decision. When the agent hits something that feels like a judgment call rather than a mechanical step, it checks the canon first. If a rule covers the situation, it follows the rule instead of re-deriving an answer. This is the whole payoff: re-derivation is where a cheap model goes wrong, so you replace re-derivation with lookup.

Write on proof. A call that proved out twice, or any error that cost something, gets appended with the next ID. Never inline. Never edit an old entry to "fix" it. If a newer rule contradicts an older one, you write the new rule and mark the old one superseded by the new ID. Both stay in the file.

That is it. No tooling required to start. The entire mechanism is: one file, append-only, four fields, loaded at session start, cited by ID.

THE GOTCHAS

These are the specific traps that earned their rules the hard way. Encode your own versions.

- **Unverified success is the number one failure class.** Agents report "done" when they mean "I ran the code that should do it." Force an observable: a post id, an HTTP status, a screenshot, a log line someone actually reads. No observable means report failure and quote the error, not success.
- **A "sent" log line is only as true as the return code it checked.** An agent wrote "message sent" to its own log while the send had silently timed out, because nothing ever read the send's return value. A log line the code prints without checking the result is not evidence, it is a second silent failure. Every delivery lane captures its result and the log states which lanes actually landed.
- **Never trust a summary of a file. Read the source.** Status snapshots drift within days. A file's own summary paragraph went stale and started outvoting the truth it was supposed to describe. When a fact matters, read the current source, not last week's description of it.
- **Verification can kill the premise of the question.** Sometimes the assumption baked into the task is the thing that is false. Report that as a first-class finding, not a footnote. When you hand a sub-task to another agent, state your assumptions as hypotheses to test, never as established facts, or the agent will confirm your bias back to you.
- **Before fixing, verify broken.** The fix instinct fires on pattern-match, and the target is sometimes already correct. Confirm the thing is actually broken before you edit it. A "fix" to working code is a new bug.
- **A bug fixed once still lives in its twins.** After any fix, search for the same shape elsewhere in your system. The same mistake copy-pasted into a sibling file is still shipping.

- **Timestamps without timezones are not data.** A UTC time reported as local was wrong by five hours and nobody caught it until it mattered. Ground every date and duration against a known clock. This compounds: an agent with no injected "today" will invent how much time has elapsed and fabricate that a deadline passed, and a reviewer with no clock will rubber-stamp the fabrication. Hand the agent its clock, and hand the checker the same clock.
 - **A name is not an identity.** Two unrelated companies shared one name, one of them defunct. Check for entity collisions before you answer anything about a company, tool, or person.
 - **The most common failure is a system violating its own written rules.** When behavior looks wrong, first find the rule that already covers it and enforce it in code (a guard, a check, a schema), not in more prose. A warning in a doc that six agents ignore is not a control.
-

IMPLEMENT IT YOURSELF

Stack-agnostic. Works in any language, any agent framework, or none.

1. **Create one file.** Call it whatever you want. Plain text or markdown. This is your canon. Do not spread it across many files, that defeats the point.
2. **Write the header.** Three short sections at the top: how to use it (check it before a judgment call, follow the matching rule, cite by ID), how to add to it (proven twice or paid for once, append with ID and provenance, errors worth more than wins), and the one hard rule (never delete, only supersede).
3. **Seed it from your own scars.** Do not invent rules. Look at the last handful of times your agent burned you or you corrected it twice. Write each as one imperative line with an ID and the real event in parentheses. Five real rules beat fifty aspirational ones.
4. **Load it at session start.** Whatever your agent's context assembly is (a system prompt, a memory loader, a preamble file), inject the full canon every session. If it is not in context, it does not exist.
5. **Make citation cheap.** Tell the agent to cite the rule ID when a canon rule drives a decision. This does two things: it proves the canon is actually being read, and it gives you a trace when a rule turns out to be wrong.
6. **Append on proof, never edit in place.** When a new call earns a rule, add it with the next sequential ID. When an old rule is outgrown, append the

replacement and mark the old one "superseded by [new ID]." Never renumber, never delete. The history is the asset.

7. **Test the cheap model against it.** Once the canon has real weight, run your workflow on a smaller or cheaper model with the canon loaded. The rules should let it avoid the mistakes it would otherwise make. That gap is the money the canon saves you.

THE COPY-PASTE TEMPLATE

```
# Judgment – the canon
```

```
Model intelligence is rented; structure is owned. This file is where judgment becomes structure. Every rule here was earned: proven useful twice, or paid for once with an error. Errors are worth more than wins.
```

```
How to use: read at session start. When making a call that feels like a judgment call (not a mechanical step), check here first. If a rule covers it, follow the rule instead of re-deriving. Cite rules by ID ("per J3").
```

```
How to add: a call proven useful twice, or any error that cost something, gets appended with the next ID, a one-line rule, and its provenance.
```

```
The one hard rule: never delete a rule. Supersede it – append the replacement and mark the old one "superseded by Jxx" – so the history stays honest.
```

```
---
```

```
## Evidence
```

```
<!-- rules about what counts as proof, truth, verification -->
```

```
- **J1 – <one-line imperative rule>.**
```

```
  *(<win proven twice / ERROR paid for once>, <date>: <the real event in one or two sentences>.)*
```

```
## Scope
```

```
<!-- rules about bounding a problem before you act on it -->
```

```
- **J2 – <one-line imperative rule>.**
```

```
  *(<provenance>.)*
```

```

## Action
<!-- rules about how to decide and move -->

- **J3 - <one-line imperative rule>.**
  *(<provenance>.)*

## Human
<!-- rules about working alongside a real person -->

- **J4 - <one-line imperative rule>.**
  *(<provenance>.)*

<!--
TODO: replace J1-J4 with rules from your own last ten corrections.
TODO: add a section only when a fifth rule needs a home.
TODO: when a rule is outgrown, append its replacement with a new ID and
      edit only the old rule's line to add "(superseded by Jxx)".
      Never delete. Never reuse an ID.
-->

```

Starter checklist for deciding whether something earns a rule:

- [] Is it a judgment call, not a mechanical step? (mechanical -> put it in code)
- [] Did it cost me something real? -> earns a rule now.
- [] Or has it proven useful at least twice? -> earns a rule now.
- [] Can I state it as ONE imperative line? (no -> I don't understand it yet)
- [] Did I capture the real event as provenance? (no -> future me will delete it)
- [] Does it contradict an existing rule? -> supersede, don't overwrite.