Olympus Engine

How We Use the Retro

A working breakdown of the session retrospective practice, as it runs today. Prepared for adoption and adaptation.



About this document. This is reference material describing a retrospective practice and how to enforce it with tooling. Read it, adapt it to your own environment, and decide for yourself what to adopt. Nothing here depends on anyone else's systems.

The practice in one sentence: **after each working session, append one honest entry about how the session went, and promote anything that repeats into something that binds.**

01 Where the retro sits in a session

The retro is not a separate chore. It is one of the inputs to the single command that wraps a session.

1. **Start: check status.** List the open session entries: yours, ones another live session owns, and orphans nobody closed.
2. **Before the first write: open an entry.** The session gets a log entry and an ID.
3. **Work.** Notice friction and corrections as they happen, while they are still sharp.
4. **Wrap: write the retro.** The wrap step names a retro file and the tool files it.
5. **Close and commit.** The close refuses without a retro or an owned skip, then commits only the files the session named.

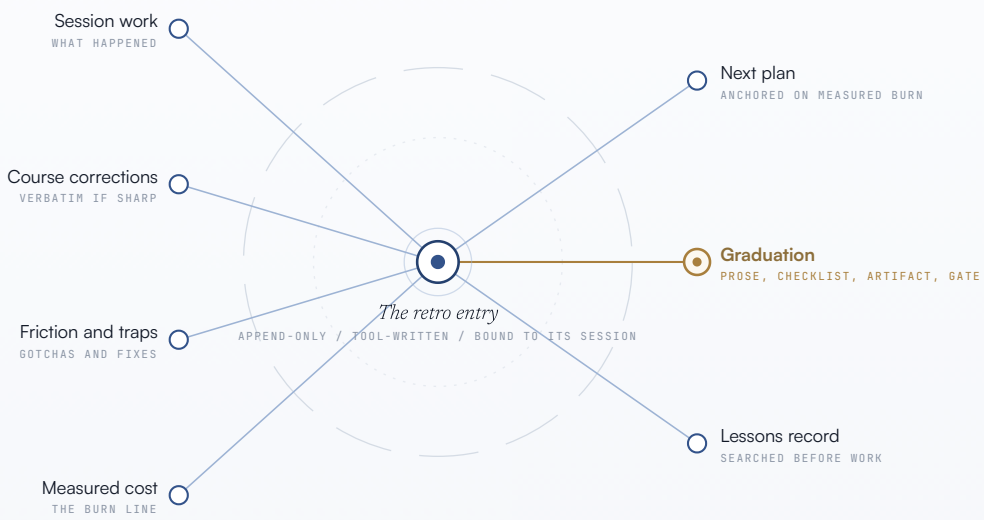
The log records what was done. The retro records how it went.

© 2026 Atlas Saint Enterprises · Powered by Olympus

OLYMPUS ENGINE :: HOW THE RETRO WORKS

One entry per session, *promoted when it repeats*

Four kinds of session signal feed one append-only entry. Two outputs read it back. A lesson that keeps repeating moves down the ladder into something that binds.



BLUE CARRIES THE SIGNAL. GOLD MARKS WHERE A LESSON BECOMES BINDING.

01

Bound to a session

A session ID ties each entry to the work that produced it.

02

One per entry

A second retro for the same entry is refused.

03

Refused if malformed

Wrong header or leftover placeholders never get filed.

04

A skip is owned

No retro means a written reason, kept in the log.

© 2026 Atlas Saint Enterprises · Powered by Olympus

02 What a retro is for

A retro records **how the session went**: process, judgment, friction. It does not record what was built. Keep a separate log for that.

- Be specific and honest. The most valuable line is usually a course correction someone made. Capture it verbatim when it was sharp.
- Never skip something because it is unflattering. A retro that only records wins has no signal in it, and the whole practice is worth exactly what its worst entries admit.
- If the session was pure trivial execution, one line (`nothing material learned`) is acceptable and better than filler.

The retro is a feeder, not the distillate. A pattern that repeats across sessions graduates out of it into a durable form. A retro nobody graduates from is a diary.

03 Entry template

```
## [YYYY-MM-DD] retro | <session title>
session: <session id>
**Focus:** <area>   **Outcome:** <shipped | partial | blocked>

**What worked** - <moves, decisions, tools that paid off>
**What didn't** - <dead ends, wasted effort, wrong turns, rework>
**Course corrections** - <where you were redirected, and the lesson; verbatim if sharp>
**Gotchas + fixes** - <tooling or process traps hit, and how each was resolved>
**Do differently next time** - <one concrete, actionable process change>
**Burn** - <agents used and rough cost | main loop size | type of work>
**Graduate?** - <none | lesson | decision | rule>
```

Two notes on using it:

- **Prose is fine.** In practice the section labels can shrink to short paragraphs ("What worked: ... What did not: ... Adjust next time: ..."). The content matters more than the headers. Keep "Do differently next time" concrete enough that a reader could act on it.
- **The Burn line is what makes planning honest.** See the Burn section below.

© 2026 Atlas Saint Enterprises · Powered by Olympus

04 Rules that make it work

1. **One file, append-only.** All entries live in one retro file. Past entries are never edited. A committed line that was changed should fail a check.
2. **One retro per session entry.** Bind each retro to the session that produced it by a `session: <id>` line, so a closing step can find it mechanically.
3. **Work after a close starts a new entry.** If a session continues after it was closed, it opens a continuation entry and owes its own retro.
4. **The header format is fixed.** First line is `## [YYYY-MM-DD] retro | <title>`. Inside the entry body, use `###` or deeper for any heading, because `##` is reserved for entry headers.
5. **No leftover template placeholders.** An entry that still contains `<...>` prompts is refused.
6. **A skip must be owned.** If a session genuinely has no retro, closing it requires an explicit skip reason that is written into the log, for example `retro: skipped (orphan wrap by <id>)`. A silent skip is not allowed.
7. **Only the tool writes the file.** Entries are appended through one command, not hand-edited, so the rules above can be enforced in one place.

05 How a tool enforces it

Because only one command writes the file, every rule above can be mechanical rather than a matter of discipline.

RULE	WHAT HAPPENS IF IT IS BROKEN
First line is `## [date] retro | title`	The append is refused.
A `session:` line matches the session being closed	Refused. The ID is how the close finds the retro.
One retro per session entry	A second retro for the same entry is refused. Work after a close opens a continuation entry that owes its own retro.
No leftover template placeholders	Refused until the prompts are replaced with content.
No `##` headings inside the body	Refused. That prefix is reserved for entry headers. Use `###`.
Closing needs a retro or an owned skip	The close fails unless a retro exists or a written reason is given. The reason is written into the log.
Past entries are append-only	The status check fails if a committed line has been edited.

© 2026 Atlas Saint Enterprises · Powered by Olympus

06 The Burn line: why every entry records cost

Plans should not guess. Every plan carries a cost estimate, and that estimate is anchored to **measured** numbers from past retros. So each retro ends with a Burn line. An illustrative example, not a real session:

```
**Burn.** Two agents (one grader per round), about 250k tokens in
total. Main loop moderate. Type: document build with a grade loop.
```

Over time the file becomes a cost table by type of work (build, audit, reconciliation, grade loop). A new plan quotes cost by summing comparable past rows instead of improvising. A line that holds only an estimate says so ("est"), and measured figures are preferred.

07 Graduation: where repeated lessons go

KIND OF THING	GRADUATES TO
A direction-changing realization about how work should be done	A doctrine record
A single non-obvious call with real reasoning behind it	A decision record
A correction to how the agent or operator behaves, which must fire every session	The standing operating instructions
A lesson reusable across many tasks	A lessons table that is checked before substantive work

The rule that makes graduation real: a fix that stays in the same medium as the failure it fixes is not a promotion. Prose that was ignored does not become binding by acquiring bold text or a fresh incident citation. Use this ladder, from least to most binding:

1. Prose
2. A checklist line
3. A forced artifact (the work cannot proceed without producing it)
4. A tool-boundary gate (a hook or check that blocks the action)
5. Structural impossibility (the wrong move cannot be expressed)

A repeat failure moves down at least one rung. If a lesson was already written as prose and the same mistake recurs, the next fix is a checklist line or stronger, never more prose.

An illustration. Suppose a rule says every search result must be opened before a report is written. If that rule keeps being missed as prose, the next fix is not louder prose. It is a hook that denies writing the report until

© 2026 Atlas Saint Enterprises · Powered by Olympus

every result was either read or named.

Record graduation on the entry itself in the `Graduate?` line, so the next reader can see whether a lesson was promoted or only noted.

08 Reading the retro before work

A retro that is never read is a diary too. Before substantive work, search the retro and the lessons table for the nouns in the task, load what matches, and say what was loaded. Treat any numbers found there as pointers to verify, not as truth, because they were correct on the day they were written. Anything that can change without an edit (revenue, counts, live status) is re-checked against the live source.

09 Minimal implementation

If you want to adopt this with little tooling:

1. Create `retro.md` with the template at the top.
2. Create a small script with three checks: the header matches the fixed format, the `session:` line matches the session being closed, and no placeholder text remains. Have it append the entry and refuse a second retro for the same session.
3. Make your session-closing step call that script, and require either a retro or a written skip reason.
4. Add a short periodic pass that scans recent entries for repeated "Do differently" lines and graduates them using the ladder above.

The script in the second step is where the practice stops depending on discipline. Without it, the retro is skipped on exactly the busy sessions that most need one.

10 Short version to remember

1. One honest entry per session, bound to its session ID.
2. The tool writes it. The close refuses without it.
3. Record what the session cost, so the next plan is anchored on numbers.
4. Repeat lessons move down the ladder, never sideways into more prose.
5. Read the retro before work, and verify any number in it before using it.

© 2026 Atlas Saint Enterprises · Powered by Olympus

Behind every lasting thing
is an unseen order.



A standalone, adoptable version of this protocol is in the companion file retro-protocol.md, and Retro-Protocol-Blueprint.pdf shows how it is wired on one sheet. Adapt it to your own environment and keep what fits.

olympusengine.io

© 2026 Atlas Saint Enterprises · Powered by Olympus