Give Claude Code a Memory

The L&D Practitioner's Setup Guide for Obsidian as Persistent Context

By Eian Newland

Claude Code + Obsidian: Persistent Memory & Living Documentation

Why I'm writing this for L&D folks. Well, L&D has a memory problem, and AI tools made it worse, not better.

We design programs, capture decisions in slide decks nobody re-opens, and start the next initiative from scratch. Then we add an AI assistant and run the same play one prompt at a time. The assistant forgets. We re-explain. The decisions, the context, the "we already tried that". All of it lives in our heads or in chat logs we'll never search.

This guide is the operating model fix. It's how I run my own work, and it's the same logic we already preach in L&D: capture, retrieve, reuse, and let the system remember so the practitioner can think.

You'll set up Obsidian (a free notes app) as Claude Code's long-term memory.

- Claude reads from it at the start of each session and writes to it as you work.
- You get a brain you can search.
- Claude gets context that survives.

The gap between "AI as one-shot prompt" and "AI as a system you actually build with" closes.

By the end, you'll have:

1. **Persistent memory:** Claude Code remembers what you've built, decided, and tried, across sessions and projects.
2. **Lower token cost:** Claude reads a small index first and loads only the notes it needs, rather than re-reading huge folders.
3. **Living documentation:** every session updates the vault, so the docs stay current without a separate writing step.

This works for any knowledge worker, but L&D folks will recognize the bones immediately. It's a curriculum for your own work, with prerequisites, anchor docs, and a feedback loop.

The vault is the operating system. Claude is the practitioner inside it.

Introduction: the 5-minute version

Read this before deciding whether to commit. If it sounds right, the full guide takes about an hour to set up and another week to feel.

The core idea

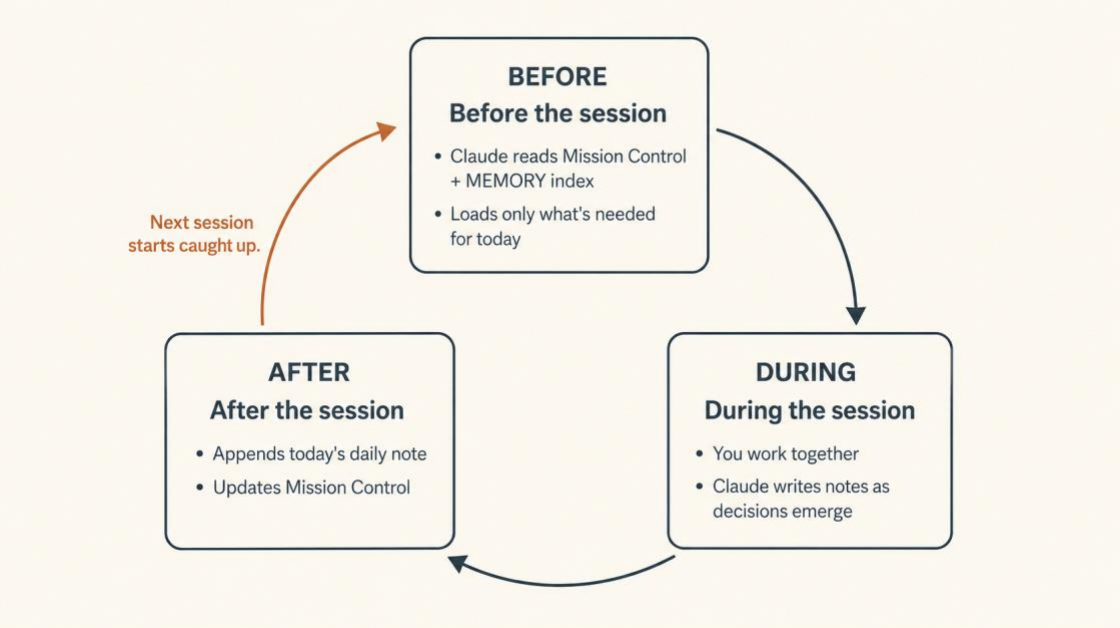
Claude Code is amazing inside one session and forgetful between sessions.

Every new conversation, you re-explain the project, your preferences, what you tried last time, what worked, and what didn't. That's expensive in time and tokens.

The fix is to put Claude's memory **outside Claude**, in a place that's also useful to you: an Obsidian vault.

- Claude reads from it at the start of each session and writes to it as you work.
- You get a brain you can search.
- Claude gets context that survives.

How the loop works



The Loop: Before → During → After → repeat

Before the session

1. Claude reads `Mission Control` and the `MEMORY` index. Small files, cheap to load.
2. Claude loads only the notes it needs for today's work, not the whole vault.

During the session

3. You work together.

4. Claude writes notes incrementally as decisions and facts emerge.

After the session

5. Claude appends to today's daily note.
6. Claude updates `Mission Control` on what has changed.
7. The next session starts, and I'm already caught up.

The discipline that makes it work **write incrementally, not in cleanup passes.** Notes get written as facts emerge, not in a dump at the end.

What you need:

Layer	Purpose	Time to set up
Obsidian vault + 5 plugins	Where notes live	15 min
`CLAUDE.md` instructions	Tells Claude how to use the vault	10 min
`MEMORY.md` index	The cheap-to-read map of what's remembered	5 min
Folder skeleton	Where things go	5 min
Bootstrap prompts	Seeds the system with your context	30 min

That's it. Total: about an hour, mostly Claude interviewing you to fill in memory files.

What it costs

- **Time:** ~1 hour setup, ~2 minutes at the end of each session for the wrap-up.
- **Money:** $0. Obsidian is free, the plugins are free, and this works on any Claude Code plan.
- **Discipline:** You have to run the end-of-session prompt. Skip it, and the system decays.

What it isn't

- It's not a replacement for Claude Code's built-in memory (the auto-memory in `~/.claude/projects/`). It complements it.
- It's not a knowledge base you have to maintain by hand. Claude does the writing.
- It's not all-or-nothing. The basic setup (Parts 1-6) gives you 80% of the value. Parts 7-8 are power-user additions.

How to use this guide

1. **First read:** skim Parts 1-2 to see what you're building. Don't install anything yet.
2. **Setup pass:** do Parts 1-2 in order. ~30 minutes.

3. **Bootstrap:** run the prompts in Part 3, in order, in a fresh Claude Code session. ~30 minutes.
4. **Live with it for a week.** Use the end-of-session prompt every time. Don't add anything else yet.
5. **Once it feels natural,** read Part 8 and pick ONE thing to add based on what's slow or annoying. Don't add all of them.

The most common failure mode is installing every plugin and MCP server on day one and never building the muscle of writing notes incrementally.

Resist that. The vault is only as good as the habit.

> 💡 **Platform note:** this guide uses `~/` as shorthand for your home directory (e.g. `~/.claude/` = your home folder's `.claude` subfolder).

- **Mac and Linux:** works as written.
- **Windows:** `~/` translates to `C:\Users\<your-username>\`. So `~/.claude/settings.json` becomes `C:\Users\<your-username>\.claude\settings.json`.
- **PowerShell:** understands `~` natively.
- **Command Prompt:** does not.

If you only remember three things

1. **The vault is the primary record.** Not Claude's chat history. Not your head. The vault.
2. **Read the index, not the contents.** That's how you save tokens. Claude loads `MEMORY.md` (small) and only opens individual notes when relevant.
3. **Write as you go, not at the end.** Cleanup passes always miss things.

Quick map: what to copy and where

Section	What it contains	Where to paste it
Part 1.3	`additionalDirectories` config	`~/.claude/settings.json`
Part 1.5	SessionStart + Stop hooks	`~/.claude/settings.json` (merge with above)
Part 1.5	git init commands	Terminal in your vault folder
Part 2.1	`CLAUDE.md` standing instructions	`~/.claude/CLAUDE.md` or vault root
Part 2.2	`MEMORY.md` index	`<vault>/09-Memory/MEMORY.md`
Part 3	Bootstrap prompts 1–7	Claude Code, in order
Part 5	Retrieval prompts	Claude Code, as needed
Part 8.1	MCP server config	`~/.claude/settings.json`

Section	What it contains	Where to paste it
Part 8.2	Custom skill template	`~/.claude/skills/<skill-name>/SKILL.md`
Part 8.4	PostToolUse hook	`~/.claude/settings.json`
Part 11	30-day review prompt	Claude Code, monthly

Ready? Start with Part 1.

Part 1: One-time setup

1.1 Install Obsidian and create the vault

1. Download Obsidian: https://obsidian.md
2. Create a new vault. Name it whatever you want (e.g., `Brain`, `Second-Brain`, `Workspace`).
3. Put the vault in a synced location so it follows you across machines:
    - **Pick ONE sync mechanism.** Either a cloud sync service (iCloud Drive, Proton Drive, Dropbox, Google Drive) **or** git with a remote like GitHub.
    - **Don't combine the two on the same vault.** That's what causes the conflict files everyone warns about.
    - **Cloud sync:** easier for non-technical folks.
    - **Git:** better if you want true version history and don't mind a manual sync step.
    - Either way works with this guide. Part 1.5 sets up local-only git for version history. If you also want it synced across machines via git, add a GitHub remote later.
4. Note the absolute path to the vault. You'll need it in step 1.3.
    - **Cloud sync path:** looks like `<your-sync-folder>/MyVault/`.
    - **Local or git-only path:** looks like `~/Documents/MyVault/`.
    - **The exact location varies by OS and sync service.** Check your sync app's docs if you're not sure where it stores files.
    - **Quick way to find it:** in Obsidian, **Settings → About → "Show debug info"** surfaces the absolute path of the open vault.

1.2 Install the core Obsidian plugins

Open Obsidian → Settings → Community plugins → Browse. Install and enable:

Plugin	Why
Dataview	Lets Claude query notes by tag/type/date instead of grepping every file. Huge token savings.

Plugin	Why
Templater	Lets Claude (and you) create notes with consistent frontmatter.
Periodic Notes	Auto-creates daily/weekly notes that become Claude's session log.
Tag Wrangler	Keeps tags clean over time so Dataview queries don't break.
Linter	Auto-fixes frontmatter and formatting on save. Keeps notes machine-readable.

Optional but recommended:

- **Omnisearch:** better full-text search than the built-in.
- **Git:** versions your vault if it isn't already in a cloud sync service.
- **Smart Connections:** local-first semantic search using embeddings stored in `.smart-env/`.
    - The single biggest token-saver in this stack. Let's Claude find the *meaning-relevant* notes instead of grepping every file.
    - ~786K downloads as of early 2026; actively maintained.
    - Add this once your vault has more than ~50 notes. It's overkill before then.
- **Bases:** built into Obsidian core, introduced in 1.9, expanded in 1.10+.
    - Obsidian's native database feature. Build filtered, projected views of your notes by frontmatter properties.
    - An AI agent can query a Base via the Local REST API and pull a pre-filtered slice instead of dumping a whole folder.
    - Treat it as the structured query layer alongside Dataview.
    - Base formula syntax changed in 1.9.2, so any examples written before late 2025 need re-checking against the current docs.

Dataview vs. Bases vs. Datacore — which one to use:

- **Dataview** is the workhorse for most existing vaults and what this guide defaults to.
- **Bases** is newer and better for property-driven views.
- **Datacore** (Dataview's successor by the same author, blacksmithgu) was added to Community Plugins in September 2025 and is safe to install.

Dataview still works fine. Migrate when you have time, not under a deadline.

Configure Periodic Notes after installing it:

Settings → Periodic Notes → enable Daily Notes → set: - Date format: `YYYY-MM-DD` - New file location: `08-Daily-Notes` - Template file location: leave blank for now (or point at a template later)

Without this, daily notes get dumped into the vault root, and Claude won't find them where they're expected.

Set up a default new-note template via Templater:

Settings → Templater → Folder Templates → add: `01-Projects` → template that inserts:

```
---
type: project-note
tags: []
created: <% tp.date.now("YYYY-MM-DD") %>
parent: ""
---
```

Now, every new note in `01-Projects/` gets front matter automatically.

Repeat for `09-Memory/` (`type: memory`) and `03-Resources/` (`type: resource`).

This means Claude doesn't have to remember to add frontmatter every time. The vault enforces it.

About auto-save. Obsidian saves every keystroke by default. You don't need Cmd-S.

If you're coming from Word or Google Docs, this is a real adjustment. Close the file, and your work is already saved.

1.3 Tell Claude Code where the vault is

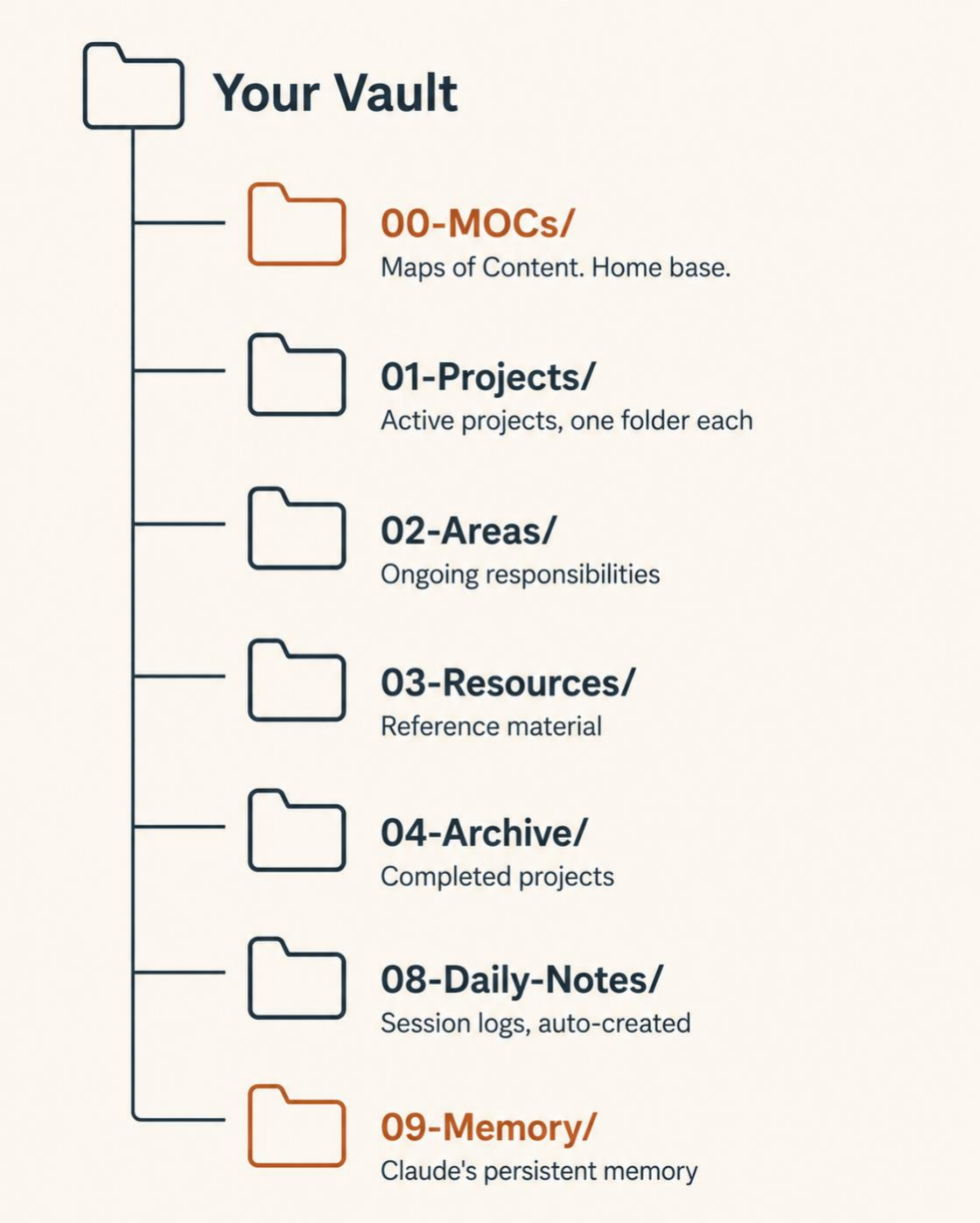
Claude Code can only read files inside its working directory or directories you've explicitly added. Edit `~/.claude/settings.json` (create it if it doesn't exist) and add the vault path:

```
{
  "additionalDirectories": [
    "/absolute/path/to/your/vault"
  ]
}
```

Replace the path with your actual vault location. Restart Claude Code after saving.

1.4 Create the vault skeleton

In Obsidian, create these top-level folders. The numbers are intentional. They keep the sidebar sorted.



Vault skeleton: 7 top-level folders, with Mission Control and Memory highlighted

```
00-MOCs/              ← Maps of Content (indexes)
01-Projects/          ← Active projects, one folder each
02-Areas/             ← Ongoing areas of responsibility (work, health, etc.)
03-Resources/         ← Reference material, research, links
04-Archive/           ← Completed projects, old notes
08-Daily-Notes/       ← Session logs (auto-created by Periodic Notes)
09-Memory/            ← Claude's persistent memory files
```

In `00-MOCs/`, create a file called `Mission Control.md`. This is the home base Claude reads at the start of every session.

1.5 The two pieces of automation worth doing on day one

The whole system depends on two things happening reliably:

1. The vault gets *saved* even if you forget the wrap-up prompt.
2. Claude *retrieves* current focus at the start of every session without you having to ask.

Two hooks in `~/.claude/settings.json` cover both.

Add this to `~/.claude/settings.json` (merge with whatever is already there):

```
{
  "hooks": {
    "SessionStart": [
      {
        "matcher": "startup",
        "hooks": [
          {
            "type": "command",
            "command": "echo '--- Current Focus from Mission Control ---' &&
sed -n '/## Current Focus/,/^## /p' '/absolute/path/to/vault/00-MOCs/Mission
Control.md' | head -50"
          }
        ]
      }
    ],
    "Stop": [
      {
        "hooks": [
          {
            "type": "command",
            "command": "cd '/absolute/path/to/vault' && git add -A && git com
mit -m \"claude session $(date +%Y-%m-%d-%H%M)\" 2>/dev/null || true"
          }
        ]
      }
    ]
  }
}
```

Replace `/absolute/path/to/vault` with your actual vault path in both places. Keep the single quotes. They let the path contain spaces.

What these do:

- **SessionStart hook:** every new Claude Code session begins by printing your Current Focus block from Mission Control. You see what you were doing before you type anything. Auto-retrieval, no prompt required.
- **Stop hook:** when a session ends, the vault auto-commits to git. Even if you forgot to wrap up properly, you have version history. Safety net for the auto-save discipline.

For the Stop hook to work, the vault needs to be a git repo. One-time setup:

```
cd /absolute/path/to/vault
git init
echo ".obsidian/workspace.json" > .gitignore
echo ".obsidian/workspace-mobile.json" >> .gitignore
echo ".trash/" >> .gitignore
git add -A && git commit -m "initial vault commit"
```

The `.gitignore` lines keep Obsidian's per-machine UI state out of the repo so syncing between machines doesn't fight with itself.

You don't need a remote (GitHub/GitLab). Local git is enough for version history. Add a remote later if you want offsite backup.

Test it. Restart Claude Code after editing `settings.json`, then:

1. Start a new session. You should see the Current Focus block print before you type.
2. End the session and run `git log` in the vault. You should see a fresh commit.

Part 2: The two files that make it all work

These two files are the difference between "Claude Code with a folder of notes" and "Claude Code with persistent memory."

2.1 `CLAUDE.md` (at the vault root or in your home directory)

This is Claude Code's standing instructions. It loads automatically every session. Put it at `~/.claude/CLAUDE.md` for global rules, or at the vault root for vault-specific rules.

Paste this in and customize the bracketed parts:

Convention used in code blocks below: anything in `<angle brackets>` is a placeholder you replace. Anything in `[square brackets]` in prose is a comment or example you should customize. Don't paste the brackets into the file.

```
# <Your Name>: Claude Code Configuration

## The vault is the primary record

My Obsidian vault at `/absolute/path/to/vault` is the single source of truth
for projects, decisions, research, and context. Read from it before answering
```

```
 questions about my work. Write to it as we work, not as cleanup at the end.

## Session start protocol

At the start of every session:
1. Read `00-MOCs/Mission Control.md` for current focus and open threads.
2. Read `09-Memory/MEMORY.md` for the memory index.
3. Only load other notes when they're directly relevant. Don't preload everyt
hing.

## Mid-session re-read protocol

If the topic shifts noticeably during a session (we move from project A to pr
oject B, or from research to implementation), re-read the relevant project's
`_index.md` before proceeding. Long sessions drift. Don't operate on stale co
ntext.

If we've been working for more than ~30 messages, proactively offer a checkpo
int: "Want me to save what we've decided so far to the vault before we keep g
oing?"

## Session end protocol

Before ending a session, update `00-MOCs/Mission Control.md` with:
- What we worked on
- Decisions made (and why)
- Open questions
- Next actions

If the session produced reusable knowledge (a pattern, a fix, a fact about my
 setup), write it as its own note in `09-Memory/` and add a one-line pointer
to `09-Memory/MEMORY.md`.

## Writing notes: the rules

- Every note has YAML frontmatter: `type`, `tags`, `created`, and `parent` if
 it belongs to a hub.
- Use `[[wikilinks]]` with basenames only. Obsidian resolves them globally.
- Write notes incrementally as we work, not in a big dump at the end.
- One concept per note. Long notes are hard to find and hard to update.
- Past tense for things that happened. Present tense for current state.

## What goes where

- **Projects** (`01-Projects/`). Anything with a goal and an end. One folder
per project, `_index.md` as the hub.
- **Areas** (`02-Areas/`). Ongoing responsibilities with no end (job, health,
 finances).
```

```
- **Resources** (`03-Resources/`). Reference material I'll reuse (prompts, te
mplates, research).
- **Memory** (`09-Memory/`). Facts about ME and my SETUP that you need across
 sessions (preferences, tools, accounts, standing decisions).
- **Daily Notes** (`08-Daily-Notes/`). What we did today. Append-only log.

## Token discipline

- Use Dataview queries to find notes by tag/type/date instead of grepping the
 whole vault.
- Read the index (`MEMORY.md`, `Mission Control.md`) first. Only load full no
tes when needed.
- If a folder has more than 20 files, ask before reading them all.

## What NOT to store in the vault

- API keys, passwords, tokens, or any secret. Use a password manager (1Passwo
rd, Bitwarden) and reference items by name only.
- Anything covered by an NDA or that names a current employer in a way you wo
uldn't want surfaced.
- Personal info about other people without their consent.

If I ask you to save something sensitive, push back and suggest a password ma
nager instead.

## Standing preferences

[Add your own. For example:]
- Direct, evidence-based responses. No filler.
- Cite sources when you use them.
- Show me the file path when you reference code.
- Ask before deleting files. Always create backups before destructive operati
ons.
```

2.2 `09-Memory/MEMORY.md` (the index)

Create this file in the `09-Memory/` folder. It's the index Claude reads to know what memories exist without loading them all.

```
# Memory Index

This is the index. One line per memory file. Load only what's relevant.

## About me
- [profile.md](profile.md). Who I am, role, goals
- [preferences.md](preferences.md). How I like to work, communication style

## My setup
- [tools.md](tools.md). Software I use, accounts, key paths
```

```
- [machines.md](machines.md). Computers I work from, OS versions, what's inst
alled where (only fill this in if you work across multiple machines)

## Standing decisions
- [decisions.md](decisions.md). Choices made once that shouldn't be re-litiga
ted

## Active projects
- See `01-Projects/` for project-level context. This index is for cross-proje
ct memory only.
```

Then create the files it points to as you go. Don't try to fill them all in on day one. Let Claude write them as facts come up.

Part 3: Bootstrap prompts

Run these in a fresh Claude Code session, in order. They get the system seeded.

> 💡 **How to use this section:** each prompt is meant to be copied verbatim into Claude Code. Don't paraphrase. The exact wording (especially file paths and section names) is what makes the system work.

Prompt 1: Verify the setup

Paste this into Claude Code:

```
Read CLAUDE.md (in my home directory or my vault root) and 09-Memory/MEMORY.m
d in my Obsidian vault. Confirm you can see both. Then list the top-level fol
ders in the vault and tell me if the structure matches what CLAUDE.md describ
es. If anything is missing, propose what to create. Don't create it yet.
```

Prompt 2: Seed the memory files

Paste this into Claude Code:

```
Interview me to seed my memory system. Ask me one question at a time about:
1. My role and what I'm trying to accomplish
2. The tools I use daily (with versions/accounts where relevant)
3. My communication preferences when working with you
4. Any standing decisions I want you to remember (e.g. "always use X over Y")

After each answer, write the relevant note in 09-Memory/ with proper frontmat
ter, and add a one-line pointer to MEMORY.md. Don't batch. Write each one as
we go.
```

Prompt 3: Create Mission Control

This prompt contains a nested code block (the Dataview query). Because Word/Markdown can't cleanly nest triple-backtick fences, the prompt is split into two parts. Paste them as **one continuous message** to Claude Code, with the Dataview block in the middle.

Part A — paste this first:

```
Create 00-MOCs/Mission Control.md with these sections (use this exact structu
re. The SessionStart hook in Part 1.5 looks for the "## Current Focus" headin
g):

# Mission Control

## Current Focus
(2-3 sentences: what I'm actively working on right now)

## Active Projects
(Use this Dataview query block verbatim, including the triple backticks:)
```

Part B — the Dataview query (paste this immediately after Part A, on the next line):

````
```dataview
TABLE status, file.mtime AS "Last edited"
FROM "01-Projects"
WHERE status = "active" AND file.name = "_index"
SORT file.mtime DESC
```
````

Part C — paste this last:

```
## Open Questions
(bullet list, oldest first)

## Session Handoff
- Last updated: <date>
- What we did last:
- What's next:

Show me the file when you're done so I can adjust.
```

Prompt 4: Set up your first project

Paste this into Claude Code (replace <name> with your actual project name):

```
I'm starting a new project called "<name>". Create 01-Projects/<name>/ with:
- _index.md as the hub note (frontmatter: type: moc, status: active, created:
 today)
- A "Decisions" section, "Open Questions" section, and "Next Actions" section
- Add a pointer to it in Mission Control under Active Projects
```

```
Then ask me 3-5 questions to flesh out the goal, constraints, and what "done"
 looks like, and write those into _index.md as we go.
```

Prompt 5: Mid-session checkpoint (use during long sessions)

Run this any time a session has gone past about 30 messages, or when you're about to switch topics.

Paste this into Claude Code:

```
Checkpoint. Before we keep going:
1. What have we decided in this session that should be written to the vault?
2. What new facts about my setup or preferences came up?
3. Write each one to the right file now (don't wait for end-of-session). Show
 me what you wrote.
```

Catches decisions before they slip away. Especially valuable if the session ends unexpectedly (laptop dies, you get pulled into a meeting).

Prompt 6: End of session

End your first real working session with this.

Paste this into Claude Code:

```
End-of-session update:
1. Append today's daily note in 08-Daily-Notes/ with what we did, decisions m
ade, and next actions.
2. Update Mission Control's Session Handoff section.
3. If we discovered anything reusable (a pattern, a preference, a fact about
my setup), write it as a memory note and add it to MEMORY.md.
4. Show me the diff of what you changed.
```

Prompt 7: Next session start

In your *next* session, start with this.

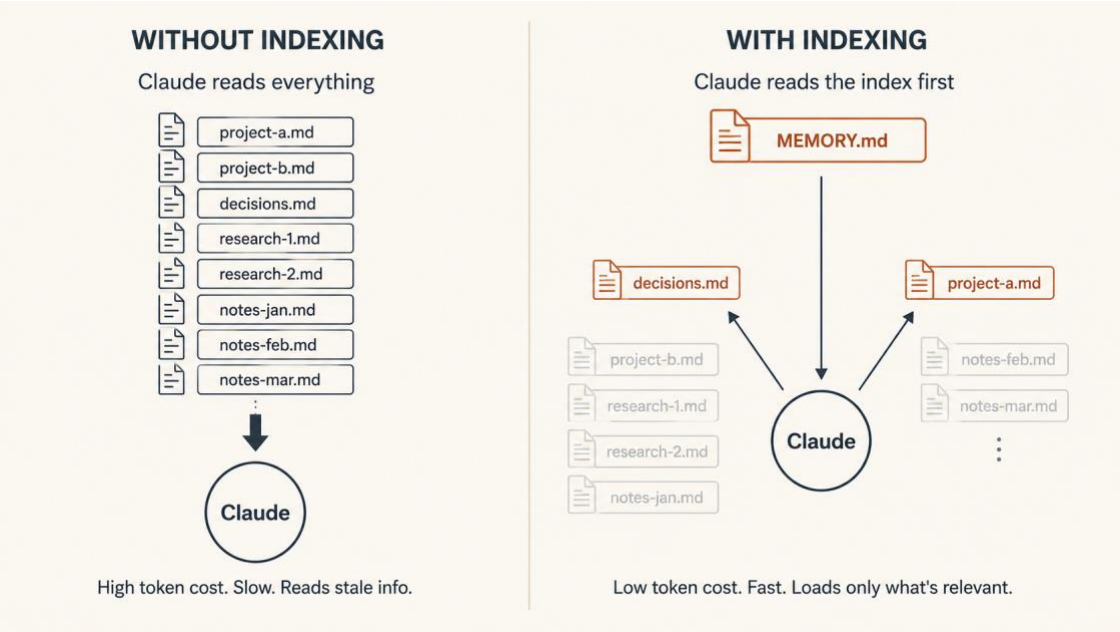
Paste this into Claude Code:

```
Catch me up. Read Mission Control and the most recent daily note. Tell me wha
t we were working on, what's open, and what you'd suggest as the next move.
```

If you set up the SessionStart hook in Part 1.5, you'll already see Current Focus printed before you type. But this prompt pulls in the daily note context too.

If Claude can answer that without you re-explaining anything, the system works.

Part 4: How this saves tokens



Read the index, not the contents: without indexing vs. with indexing

The trick is **read the index, not the contents**. Claude doesn't need every note in context. It needs to know what exists and load the right one.

The pattern: 1. **Small indexes load cheap.** `MEMORY.md` and `Mission Control.md` are under 200 lines. 2. **One-line pointers per entry.** Enough description that Claude knows whether to load the full note. 3. **Dataview runs on Obsidian's data model**, not by reading every file. 4. **One concept per note.** Loading one doesn't pull in unrelated content.

If `MEMORY.md` grows past ~200 lines, the entries are too long. Move detail into the topic files and shorten the index lines.

Part 5: Retrieval prompts you'll actually use

The whole point of the system is being able to pull things back out. These are copy-paste prompts for the most common retrieval needs. **Bookmark this section.**

> 💡 **Convention:** anything in `<angle brackets>` is a placeholder you replace before pasting. Don't paste the brackets themselves.

Catch me up (start of session)

```
Catch me up. Read Mission Control and the most recent 2 daily notes. Tell me what we were working on, what's open, and what you'd suggest as the next move.
```

Find a decision

```
Find every decision I've made about <topic>. Search across all project _inde
x.md files and 09-Memory/decisions.md. List them with date and reasoning. Don
't summarize. Give me the actual decisions.
```

What was I working on at a specific time

```
What was I working on the week of <date>? Read the daily notes from that week
 and summarize the top 3 things, with links to the projects they belong to.
```

Search across the vault

```
Search the vault for anything mentioning <topic or person or tool>. Use the f
ilesystem search, then for each hit tell me the file, the surrounding contex
t, and whether it's still current (check the file's date and status).
```

What are my open questions

```
List every open question across all my active projects. Read each _index.md i
n 01-Projects/ where status is active, find the "Open Questions" section, and
 consolidate. Group by project. Flag any that have been open for more than 2
weeks.
```

What have I tried that didn’t work

```
Find anti-patterns and failed approaches in my vault. Search for sections lab
eled "What didn't work", "Lessons", "Anti-patterns", or memory notes tagged w
ith #lesson. List them with the project they came from. Use this before sugge
sting an approach so we don't repeat past mistakes.
```

Cross-project lookup

```
Find every mention of <vendor/tool/person> across all projects. For each hit,
 tell me the project, the context, and what role they played. I want to see t
he full pattern of how this comes up across my work.
```

What’s stale

```
Find projects in 01-Projects/ where status is active but the _index.md hasn't
 been edited in more than 30 days. List them with last-edit date. These are c
andidates to either restart or archive.
```

Show me my preferences and standing decisions

```
Read 09-Memory/preferences.md and 09-Memory/decisions.md. Tell me everything
I've asked you to remember about how I work. I want to audit it for stuff tha
t's no longer true.
```

Pull a project briefing

```
Brief me on project <name>. Read 01-Projects/<name>/_index.md and any linked
notes. Give me: the goal, current status, decisions made so far, open questio
ns, and what "done" looks like. Treat this like onboarding a teammate.
```

Find related notes (when you don't know what you're looking for)

```
I'm thinking about <topic or problem>. Search the vault for anything that mig
ht be relevant. Direct mentions, adjacent topics, related projects, similar p
roblems I've solved before. Show me the top 5-10 hits ranked by relevance and
 tell me why each one matters.
```

What did I learn last month

```
Read all the daily notes from the past 30 days. Pull out: lessons learned, su
rprises, things I changed my mind about. Write the result to 03-Resources/mon
thly-retro-<date>.md so I can review it.
```

Upgrade my CLAUDE.md (run this monthly)

```
Read my current CLAUDE.md (in ~/.claude/ or vault root) and the last 30 days
of daily notes. Find:
1. Preferences I corrected you on more than once (those should be standing ru
les in CLAUDE.md, not memory notes).
2. Workflows I run repeatedly that aren't documented anywhere.
3. Things in CLAUDE.md you're routinely ignoring or that no longer match how
I actually work.

Propose specific edits to CLAUDE.md as a diff. Don't apply them. Show me firs
t.
```

This is the meta-loop. The system improves itself based on how you use it, not how you imagined you'd use it.

Tip: If you find yourself running the same retrieval prompt repeatedly, that's a candidate for a custom skill (Part 8.2). Wrap it up so you can just say "stale projects" or "what didn't work" and Claude runs the full prompt.

Part 6: Common mistakes to avoid

⚠ **The five most common ways this system breaks:**

1. **Don't put the vault inside a git repo Claude commits to.** Use iCloud, Proton Drive, or Dropbox for sync instead. Mixing the two causes conflicts.
    - The local-only git in Part 1.5 is fine. It's the *combination* of cloud sync and a Claude-committed remote that breaks.
2. **Don't write end-of-session memory dumps.** Write incrementally as facts emerge. Cleanup-at-the-end always misses things.
3. **Don't let `MEMORY.md` become the memory.** It's an index. The actual content lives in the files it points to.

4. **Don’t skip frontmatter.** Notes without `type` and `tags` won’t show up in Dataview queries and Claude can’t find them.
5. **Don’t make Claude re-discover your preferences every session.** If you correct Claude on the same thing twice, that’s a memory note.

Part 7: Maintenance

Once a quarter: - Review `09-Memory/`. Delete or update notes that are stale. - Check `MEMORY.md` line count. If it’s over 200, your index entries are too long. - Archive completed projects from `01-Projects/` to `04-Archive/`. - Run a Dataview query for notes with no tags or no type. Fix or delete them.

That’s the whole system. Start small, write incrementally, trust the index.

Part 8: Going further

Once the basic loop is working (you can end a session, come back the next day, and Claude catches itself up from Mission Control without you having to re-explain), these add real power. Pick based on what’s slow or annoying. Don’t add all of them on day one.

Quick map: what to add when

If this is your problem…	Add this
“Claude can read the vault but writing notes feels clunky”	**Obsidian Local REST API** (8.1)
“I keep re-typing the same instructions to Claude”	**Custom skills** (8.2)
“I want Claude to search the web, query my calendar, send messages”	**MCP servers** (8.3)
“I want Claude to behave automatically (auto-format on save, auto-commit)”	**Hooks** (8.4)
“I want a single place where ChatGPT, Gemini, and Claude all read the same facts about me”	**Context API pattern** (8.5)

8.1 Obsidian Local REST API: let Claude write to the vault directly

By default, Claude Code edits vault files through the filesystem, which is fine but skips Obsidian’s index.

That means new notes don't show up in Obsidian until you reopen the vault, and Dataview can be stale.

The Local REST API plugin fixes this. Claude writes through Obsidian itself, so the index updates immediately.

Install:

1. Obsidian → Settings → Community plugins → Browse → **Local REST API** → Install + Enable.
2. Open the plugin settings. Copy the API key it generates.
3. Note the port (default `27124` for HTTPS, `27123` for HTTP).

Wire it into Claude Code (via MCP. See 8.3 for the broader MCP setup):

Several community MCP servers wrap the Local REST API. The package landscape has shifted in 2026, so check current options before installing. As of May 2026:

- **`obsidian-mcp-server`** (cyanheads) — **the safest default.**
    - Current download leader (~9,800/week). Most actively maintained.
    - Latest v3.2.0 (May 17, 2026) ships both STDIO and Streamable HTTP transports.
- **`@bitbonsai/mcpvault`** — pin **v0.11.2** or later.
    - Renamed from unscoped `mcpvault` in March 2026 at Obsidian's trademark request. Not from the abandoned `mcp-obsidian` package, which is a different project.
    - v0.9.1 closed a symlink path-traversal bypass.
    - v0.10.0 added a `createServer()` factory.
    - v0.11.x fixed YAML frontmatter rewrites and added soft-delete.
    - Homepage: `mcpvault.org`.
- **`@oomkapwn/enquire-mcp`** (~5,000/week, May 18, 2026) — hybrid-retrieval / agentic-RAG focus. Worth a look if your vault is large and you want semantic-first search baked in.
- **`@huangyihe/obsidian-mcp`** — older (last release July 2025) and low-traffic. Skip unless you have a specific reason.

Note: the official `modelcontextprotocol/servers` repo does not yet ship an Obsidian server. All current options are community-maintained.

Sample config using `obsidian-mcp-server` (the recommended default):

Add this to `~/.claude/settings.json` (or wherever your Claude Code MCP config lives):

```
{
  "mcpServers": {
    "obsidian": {
      "command": "npx",
      "args": ["-y", "obsidian-mcp-server"],
      "env": {
        "OBSIDIAN_API_KEY": "paste-the-key-here",
        "OBSIDIAN_BASE_URL": "https://127.0.0.1:27124"
      }
    }
  }
}
```

Restart Claude Code. You'll get tools for reading, patching, appending, and searching vault files. Exact tool names vary by package. Check the README before configuring.

Self-signed cert note: the Local REST API uses HTTPS on port 27124 with a self-signed certificate by default. If the MCP server complains about cert verification:

- Switch to the HTTP port (27123) for local-only use: `OBSIDIAN_BASE_URL=http://127.0.0.1:27123`.
- Or follow the package README to trust the cert.

Local REST API now exposes its own MCP endpoint. As of recent versions, the Local REST API plugin also hosts an MCP server at `https://127.0.0.1:27124/mcp/`. You can point Claude Code at it directly instead of using a wrapper package.

Either path works. The wrapper packages add convenience tools; the built-in endpoint is one less moving piece.

Why it matters: appending to today's daily note becomes one tool call instead of "read file, modify string, write file". Fewer tokens, faster, and Obsidian sees the change instantly.

Token-saving tip — use PATCH on large notes. The Local REST API supports PATCH for editing specific headings, blocks, or frontmatter fields without round-tripping the whole file. When you ask Claude to update something specific ("update the Current Focus section"), tell it to use the PATCH endpoint rather than reading-modifying-writing. Big wins on large notes.

MCP `alwaysLoad` for vault servers. If your vault MCP server is one you'll use every session, add `"alwaysLoad": true` to its config block (added in Claude Code v2.1.121). It skips the deferred-tool loading flow and is ready immediately at session start.

8.2 Custom skills: package repeated workflows

A *skill* is a markdown file Claude Code loads on demand when its description matches what you're doing. Use them to package the workflows you do over and over.

Where they live: - Global: `~/.claude/skills/<skill-name>/SKILL.md` - Per-project: `<project>/.claude/skills/<skill-name>/SKILL.md`

Anatomy of a skill:

```
---
name: end-of-session
description: Use at the end of any working session to update Mission Control,
 append the daily note, and surface any new memory candidates. Trigger when t
he user says "wrap up", "end of session", "let's stop", or similar.
---

# End-of-Session Protocol

Run these steps in order:

1. **Append daily note:** open `08-Daily-Notes/YYYY-MM-DD.md` (create if miss
ing). Append a section with:
   - What we worked on
   - Decisions made and why
   - Next actions
   - Open questions

2. **Update Mission Control:** edit `00-MOCs/Mission Control.md`:
   - Update "Current Focus"
   - Update "Session Handoff" with date + 2-3 sentence summary
   - Move any completed items out of "Next Actions"

3. **Memory candidates:** scan today's conversation for:
   - Preferences I corrected you on (write to `09-Memory/preferences.md`)
   - Tools or accounts mentioned (update `09-Memory/tools.md`)
   - Standing decisions made (write to `09-Memory/decisions.md`)
   For each new memory, add a one-line pointer to `09-Memory/MEMORY.md`.

4. **Show the diff:** print what changed in each file before finishing.
```

Skills worth building first:

Skill	Trigger	What it does
`end-of-session`	"wrap up", "end of session"	The protocol above
`start-of-session`	"catch me up", session start	Reads Mission Control + latest daily note, summarizes state
`new-project`	"start a new project called X"	Creates `01-Projects/X/_index.md` with template, links from Mission Control

Skill	Trigger	What it does
`weekly-review`	"weekly review"	Dataview-summarizes the week, surfaces stale projects, flags untagged notes
`capture-decision`	"decision:" or "we decided to"	Writes the decision to the relevant project's `_index.md` with date and reasoning

The skill description is everything. Claude Code only loads a skill when its `description` matches the user's intent.

Be specific about *trigger phrases*. Vague descriptions mean the skill never fires.

8.3 MCP servers: give Claude new senses

MCP (Model Context Protocol) lets Claude Code talk to external tools. Web search, calendars, browsers, databases, your password manager, anything with an MCP server.

Worth installing for an Obsidian-centric workflow:

Server	What you get	Install
`obsidian-mcp-server`	Direct vault read/write via Local REST API	See 8.1
`filesystem`	Scoped filesystem access outside the vault	`npx -y @modelcontextprotocol/server-filesystem <path>`
`fetch`	Web fetching for research notes (Python, not npm)	`uvx mcp-server-fetch`
`tavily-mcp`	Web search (better than fetch alone)	Requires API key, free tier available. **Note:** the older `@modelcontextprotocol/server-brave-search` was archived from the official repo in February 2026. Use `tavily-mcp` instead.
`github`	Read issues/PRs into project notes	`npx -y @modelcontextprotocol/server-github` (or use GitHub's hosted remote MCP, which Anthropic and GitHub now both

Server	What you get	Install
		recommend over the local server)
`playwright` (Microsoft)	Headless browser for scraping pages into notes	`npx -y @playwright/mcp@latest`

Where to configure: `~/.claude/settings.json` under `mcpServers`. Each server is a `command` + `args` + optional `env`. Restart Claude Code after edits.

Discovery. The official list is at https://github.com/modelcontextprotocol/servers. Check there before writing your own. There's a server for almost everything (Slack, Linear, Notion, Google Drive, Postgres, SQLite, Stripe, etc.).

Cost discipline. Every MCP server adds tools to Claude's context window. Don't install ones you won't use. If a server isn't needed for a project, remove it from that project's config.

8.4 More hooks: beyond the day-one two

You already set up `SessionStart` (auto-print Current Focus) and `Stop` (auto-commit vault) in Part 1.5. These are additional hooks worth adding once you're comfortable.

Configured in `~/.claude/settings.json` under `hooks`. Example:

```
{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Edit|Write",
        "hooks": [
          {
            "type": "command",
            "command": "f=$(jq -r '.tool_input.file_path // empty'); if [[ \"
$f\" == */vault/* && \"$f\" == *.md ]]; then npx markdownlint-cli2 \"$f\" --f
ix; fi"
          }
        ]
      }
    ]
  }
}
```

This hook lints any markdown file Claude edits inside the vault, keeping frontmatter and formatting consistent without you thinking about it.

How hooks read tool data. PostToolUse hooks don't get an env var with the file path. They receive a JSON payload on stdin describing the tool call. The `jq -r '.tool_input.file_path'` snippet above extracts the path from that JSON.

You'll need `jq` installed:

- **Mac:** `brew install jq`
- **Debian/Ubuntu:** `apt install jq`
- **Windows:** `choco install jq` or `winget install jqlang.jq`

Other useful hooks: - `PreToolUse` on `Bash`. Block dangerous commands (rm -rf, force push) by exiting non-zero. - `UserPromptSubmit`. Log every prompt to a file for later retro. - `PostToolUse` on `Write` in `09-Memory/`. Auto-update `MEMORY.md` index whenever a new memory file is created.

Hooks > prompting. If you keep telling Claude "always do X after Y," that's a hook. The harness enforces it; Claude can't forget.

8.5 Context API pattern: one source of truth across all your AI tools

This one's overkill for most people, but the payoff is huge if you use multiple AI tools (Claude + ChatGPT + Gemini + Perplexity).

The problem: each tool has its own memory, and they drift. You tell ChatGPT your job title, then Claude doesn't know.

The pattern:

1. Put canonical facts about yourself in a folder of markdown files (e.g. `~/ai-context/facts/profile.md`, `tools.md`, `preferences.md`).
2. Stand up a tiny HTTP service that reads those files and returns them as JSON. (FastAPI in 50 lines, or even a static file server.)
3. Expose it on a domain via Cloudflare Tunnel or Tailscale.
4. Reference it from every tool's system prompt: *"Fetch fresh facts from `https://context.mydomain.com/facts/{name}`. Never paraphrase from memory."*

Now updating one file updates every AI tool that reads it. The vault is the human view; the API is the machine view.

Minimum viable version (no domain, no tunnel): just point Claude Code at the folder via `additionalDirectories` and tell it "always read `~/ai-context/facts/profile.md` for facts about me." You get most of the benefit without the infrastructure.

Build it for real when: you find yourself updating the same fact in three places, or you want ChatGPT and Claude to agree on what your job title is.

8.6 Session continuity and token-cost features (worth knowing about)

Claude Code has shipped several features in early 2026 that interact with a vault-based memory system. None require setup beyond what's already in this guide but knowing them helps you get more out of the system.

Session Recap (v2.1.108, April 14, 2026). When you return to a session after a gap, Claude offers to summarize what was happening before re-reading the full transcript. Configurable in `/config`; can also be triggered manually with `/recap`.

With a vault-based system, this is mostly redundant. Your session state is already in Mission Control and the daily note. But it's a useful safety net for sessions that ran long without a wrap-up.

Enhanced `/resume` (v2.1.116–117, late April 2026). Lists prior sessions interactively.

- v2.1.116 made it up to 67% faster on large (40MB+) sessions.
- v2.1.117 added a "summarize stale sessions before re-reading" option.

For vault-bound work, the better pattern is usually: end the session cleanly with the wrap-up prompt, then start fresh next time and let Claude read Mission Control. `/resume` is for when you didn't get to wrap up.

`/resume` vs. `/loop` vs. fresh sessions: - **Fresh session + Mission Control catch-up:** the default. Cleanest, cheapest, lowest token cost. - **`/resume`:** when a session ended unexpectedly and you need to pick up exactly where you were. - **`/loop`:** for unattended recurring tasks (e.g., a daily 6am skill that scans new vault notes and updates a cross-reference file). NOT for session continuity.

Prompt caching (on by default). Claude Code automatically caches your CLAUDE.md, settings, and recently read files. Cached reads cost roughly 10% of standard input tokens. This is what makes it cheap to read your vault repeatedly across a long session.

To extend cache duration to 1 hour (default is 5 minutes), set in your environment:

```
export ENABLE_PROMPT_CACHING_1H=1
```

Useful for sessions that drift across long pauses (lunch, meetings) where you'd otherwise lose the cache.

Opus 4.7 with 1M context (Max/Team/Enterprise plans, v2.1.111+). Available at standard pricing. No long-context premium.

Practically: you can fit roughly 5x more vault content in one session before hitting context limits. With this, Claude can hold your entire MEMORY index, all active project hubs, and the last month of daily notes in working memory at once. Combine with prompt caching and you can do all-day vault work without context anxiety.

Caveat: the new tokenizer can yield up to ~35% more tokens for the same input, so the effective context window is larger than 1M raw text characters might suggest.

Fast Mode now defaults to Opus 4.7 (v2.1.142, May 14, 2026). Relevant because Fast Mode is what handles compaction and recap by default. If those operations feel slower or more thorough than before, this is why.

Effort levels (`xhigh` is a new level for Opus 4.7). Sits between `high` and `max`. Adjust interactively via `/effort` if you want faster/cheaper responses for routine vault edits, or slower/more thorough for hard reasoning.

Hook output replacement (v2.1.121). Hooks can now rewrite tool output before Claude sees it, via `hookSpecificOutput.updatedToolOutput`.

Power-user use: a `PostToolUse` hook on `Read` for vault notes that strips frontmatter from what Claude sees (Claude already knows the schema; the actual frontmatter values rarely matter for reasoning). Saves tokens on every read.

Part 9: Where to learn more

- **Claude Code docs:** https://code.claude.com/docs
- **Claude Code changelog:** https://code.claude.com/docs/en/changelog
- **MCP server registry:** https://github.com/modelcontextprotocol/servers
- **Obsidian help:** https://obsidian.md/help/
- **Obsidian changelog:** https://obsidian.md/changelog
- **Dataview docs:** https://blacksmithgu.github.io/obsidian-dataview/
- **Datacore (Community Plugins):** search "Datacore" inside Obsidian
- **Skills examples:** look at `~/.claude/skills/` after you install Claude Code. The built-ins are good reference.

If something in this guide is wrong or out of date, the docs win. This was written against Claude Code and Obsidian as of May 2026.

A note about staying current

The features in this guide will change. Probably within weeks of you reading it. That's the nature of tools that ship this fast.

- New Claude Code versions land roughly every few days.
- Obsidian plugins evolve.
- MCP packages rename, fork, and consolidate.

Whatever I wrote here is a snapshot, not a contract.

Stay curious. Keep iterating. When something stops working the way the guide describes, that's not the guide failing — that's the tools moving. Check the changelog, read the new docs, ask Claude what changed, and update your own setup.

The discipline is the part that stays:

- Write incrementally, not in cleanup passes.
- Read the index, not the contents.

- Let the vault remember so you can think.

Everything else is implementation detail that will keep moving. The system is yours to evolve. Run the 30-day review in Part 11. Pay attention to what's slowing you down. And when you find a better way, write it back into your own CLAUDE.md so the next version of you inherits the upgrade.

That's the whole game.

Part 10: Troubleshooting

Symptom	Likely cause	Fix
Claude says "I can't access that path" or "no such file" when you reference the vault	Vault not in `additionalDirectories`, or path has a typo	Open `~/.claude/settings.json`, verify the path is absolute (starts with `/`) and exists. Restart Claude Code after editing.
New notes don't appear in Obsidian until you reopen the vault	Filesystem write but Obsidian's index is stale	Cmd-R inside Obsidian to reindex. For permanent fix, set up the Local REST API (Part 8.1).
Daily notes get created in the vault root instead of `08-Daily-Notes/`	Periodic Notes plugin not configured	Settings → Periodic Notes → Daily Notes → set "New file location" to `08-Daily-Notes`.
`SessionStart` hook prints nothing	Path in the hook is wrong, or Mission Control doesn't have a `## Current Focus` heading	Verify the path in `settings.json` matches the actual vault. Make sure Mission Control has exactly `## Current Focus` as a heading (Claude needs that to find the section).
`Stop` hook isn't committing	Vault isn't a git repo, or git isn't configured	Run `cd /vault/path && git status`. If it says "not a repository," do `git init`. If it complains about user.name/user.email, run `git config user.name "Your Name"` and `git config user.email "you@example.com"`.

Symptom	Likely cause	Fix
Dataview queries return nothing	Notes are missing frontmatter, or tags are inconsistent (`#L&D` vs `#ld`)	Check that target notes have `type:` and `tags:` in frontmatter. Use Tag Wrangler to merge inconsistent tags.
`MEMORY.md` is over 200 lines and Claude only loads part of it	Index entries are too long	Shorten each line to under ~150 chars. Move detail into the topic file the line points to.
Sync conflicts in iCloud/Dropbox/Proton (files showing up as `- conflicted-`)	Two machines edited the same file before sync caught up	Open both versions, manually merge, delete the conflicted copy. To prevent: close the vault on one machine before working on the other, or use Obsidian Sync ($4/mo) which handles this properly.
Claude keeps asking for the same fact every session	Fact isn't in `09-Memory/` yet, or `MEMORY.md` doesn't point to it	Tell Claude: "Save this as a memory note and add it to MEMORY.md." Verify both files actually got written.
Claude reads stale info from a memory file	Memory note hasn't been updated since the fact changed	Open the file, edit it, save. Tell Claude "I updated `09-Memory/X.md`. Re-read it."
Auto-commit creating noisy commits with workspace.json changes	Obsidian's per-machine UI state is being committed	Add to `.gitignore`: `.obsidian/workspace.json`, `.obsidian/workspace-mobile.json`, `.trash/`.
Claude can read the vault but Obsidian Local REST API tools don't show up	MCP server config typo, or Obsidian isn't running	Check `~/.claude/settings.json` for the `obsidian` server block. Make sure Obsidian is open (the API only runs when the app is open). Check the port matches what's in the plugin settings.
You can't tell if a hook is actually running	Hooks fail silently by design	Add `>> ~/claude-hook-debug.log 2>&1` to the end of your hook command.

Symptom	Likely cause	Fix
		Check that file after running Claude Code.

If you hit something not on this list, the fastest debug path is to ask Claude Code itself, with `--verbose` or by saying "show me what tool calls you're making and why." Claude can usually diagnose its own context problems if you ask it to.

Part 11: The 30-day system performance review (bonus prompt)

Run this once a month. Put a recurring reminder in your calendar. First Monday of the month works well. The whole point is to catch where the system is decaying or where new features could let you do more with less.

Paste this into Claude Code (replace <today> with the current date):

```
30-day system performance review for my Claude Code + Obsidian setup. Today's
 date is <today>.

Read the following:
1. CLAUDE.md (in ~/.claude/ or vault root)
2. 09-Memory/MEMORY.md and every file it points to
3. 00-MOCs/Mission Control.md
4. The last 30 daily notes in 08-Daily-Notes/
5. The last 30 days of git log on the vault (run: git log --since="30 days ag
o" --oneline)

Then evaluate the system across SIX dimensions and give me a written report:

A) RETRIEVAL HEALTH
- Did the SessionStart hook fire and show useful Current Focus content most s
essions?
- Were there sessions where I had to re-explain context that should have been
 in the vault?
- Is Mission Control's Session Handoff being updated reliably, or is it stal
e?

B) CAPTURE HEALTH
- Are decisions being written to the vault as they happen, or only at end-of-
session?
- Are daily notes getting populated, or are there gaps?
- Are there preferences I corrected you on more than once that should be in C
LAUDE.md or 09-Memory/preferences.md?
- Are there projects in 01-Projects/ with no _index.md or with stale status?

C) TOKEN EFFICIENCY
```

```
- Roughly how many tokens did we burn re-reading the same files this month?
- Are there notes over 500 lines that should be split?
- Is MEMORY.md still under 200 lines?
- Are there folders we routinely scan exhaustively that could use a Dataview
query, a Base, or Smart Connections instead?
- Should we enable ENABLE_PROMPT_CACHING_1H given my session patterns?

D) STRUCTURAL HEALTH
- Notes missing frontmatter (run a Dataview query for FROM "" WHERE !type)
- Tag inconsistencies (singular vs plural, capitalization drift)
- Projects in 01-Projects/ that haven't been touched in 30+ days (archive can
didates)
- Broken wikilinks
- Memory files that contradict each other

E) NEW CAPABILITIES TO ADOPT
- Have I been hand-doing anything that's now a custom skill candidate? (Look
for repeated prompt patterns in the daily notes.)
- Are there Claude Code features released since last review I'm not using? (C
heck the changelog at https://code.claude.com/docs/en/changelog.md.)
- Are there Obsidian or plugin updates I should install? (Check community plu
gins for updates.)
- Should any of the Part 8 "going further" items be promoted to my daily setu
p based on this month's patterns?

F) WHAT TO STOP DOING
- Memory notes that have never been retrieved
- Skills that fire too often or never fire
- Hooks that are noisy or broken
- Projects that should be killed, not finished

Format the report as:
- Top 3 wins (what's working well. Keep doing)
- Top 3 frictions (what's slowing me down)
- 1-3 specific changes to make THIS WEEK (with file paths, prompts, or comman
ds ready to run)
- 1-3 things to watch but not change yet

Save the report to 03-Resources/system-reviews/<date>-30-day-review.md so I c
an compare against next month's.
```

The change-this-week constraint is the important part. A monthly review that produces 20 recommendations gets ignored. A monthly review that produces 1-3 specific changes you can make in an hour compound.

After 6 months of these reviews, the system will look noticeably different from this guide. And that’s the point. The vault is yours, not mine.