STANDARD OPERATING PROCEDURE

Building a Second Brain with Obsidian + Claude Code

A living wiki that learns, links itself, and stops Claude from guessing.

Owner: Nuno Tavares • **Stack:** Obsidian + Claude Code + VS Code

The Big Idea

Most people give Claude context one of two ways: paste files in every session, or trust Claude's built-in memory. Both fail. Memory knows who you are. It does not know why a command breaks, what a working template looks like, or how two parts of your stack actually connect. So Claude guesses.

A second brain in Obsidian fixes this. You drop docs, transcripts, and screenshots into one folder. Claude reads them, builds a wiki, links every page to every related page, and gives you a graph view of your own knowledge. Every future agent query starts there instead of in the wild.

This SOP is for building a topic-specific expert (in this case, a Paperclip expert), but the same structure works for a personal second brain, a company brain, or one brain per business.

> **BEFORE YOU BUILD**
> Pick one topic per vault. Don't dump everything in one place. The smaller and tighter the scope, the faster Claude reasons over it. You'll create separate vaults per business, per tool, and per major project — then link them later.

Prerequisites

- Windows or Mac machine (Nuno-Strix or The Builder)
- Claude Code already installed and working
- VS Code installed (this is your control surface, not Obsidian's UI)
- Chrome browser (for the Obsidian Web Clipper extension)
- A working folder on disk where vaults live (e.g., D:\dev\)

The Workflow (Nuno's Order)

This is the modified order I run every time. The video creator goes Obsidian-first; I go folder-first because I'm a developer and I want my filesystem clean before any tool touches it. VS Code is the cockpit — Obsidian is the renderer.

1 Create the folder for the second brain

Decide what this brain is. Be specific. Not "general notes" — "Paperclip," "VCI Curriculum," "GoHighLevel SaaS Mode." One topic per vault.

Inside your dev directory, create a new folder named after the brain. Example:

```
D:\dev\Paperclip Second Brain\
```

That's it. Empty folder. No Obsidian yet. No files yet. Just the container.

2 Download and install Obsidian

Go to **obsidian.md** and download the installer for your OS. Mac and Windows both work. It's free.

Run the installer. Default options are fine. Finish the install but don't open it yet — or if it auto-opens, just leave it on the welcome screen.

While you're at it, install the **Obsidian Web Clipper** Chrome extension from **obsidian.md/clipper**. Pin it to your toolbar. You'll use it constantly to clip GitHub pages, articles, and discussions straight into your vault.

3 Point Obsidian at the folder you created

Open Obsidian. On the welcome screen, click "Create new vault."

1. Vault name: same as your folder (e.g., "Paperclip").
2. Location: click Browse → navigate to the folder you created in Step 1 → Select.
3. Click Create.

Obsidian now treats that folder as the vault. Confirm by checking the folder on disk — you'll see a hidden .obsidian subfolder appear. That's the marker.

QUICK SETTINGS
Open Settings (gear icon, bottom left). Switch to Light theme if that's your preference. Skip community plugins for now — we'll add them later if needed. The default install does 95% of what we need.

4 Open the vault folder in VS Code

This is where my workflow diverges from most tutorials. I don't run Claude from a standalone terminal — I run it from inside VS Code so I have the file tree, the editor, and the terminal all in one window.

4. Open VS Code.
5. File → Open Folder → select the same folder you created in Step 1.
6. Trust the folder when prompted.
7. Open the integrated terminal (Ctrl+` or Cmd+`).
8. Type **`claude`** and press Enter. Claude Code launches inside VS Code's terminal.

5 Hand Claude the LLM Wiki schema (Karpathy's prompt)

Karpathy's LLM Wiki is the prompt that turns Claude into a librarian. It tells Claude how to structure the vault — what folders to create, what the schema looks like, what rules to follow when adding new content.

Paste the entire LLM Wiki prompt into Claude Code. Hit Enter. Claude will plan the build:

- A raw/ folder for unprocessed inputs (transcripts, screenshots, clippings)
- A wiki/ folder where the actual living knowledge lives
- A CLAUDE.md schema file — the rulebook Claude reads at the start of every session
- An index file — the table of contents Claude uses to find pages
- A .gitignore (skip this if you're not pushing to GitHub — just tell Claude to ignore it)

Claude will ask you one clarifying question: what's the primary domain topic? Be specific. "A topic-specific expert on Paperclip AI setup, configuration, and troubleshooting" beats "general second brain."

CLAUDE.md DISCIPLINE

Keep CLAUDE.md under 200 lines. It's the static identity — what this brain is, who Claude is supposed to be when reasoning over it, and the rules for adding content. The actual knowledge lives in the wiki, not in CLAUDE.md.

6 Start feeding the brain

Now the structure exists. Empty wiki, empty raw folder. Time to fill it.

Method 1: Web Clipper

Browse to the source page (GitHub repo, discussions tab, blog post, docs page). Click the Obsidian Clipper extension → Add to Obsidian → Open Obsidian. The page lands in your vault as a markdown file with full text and links preserved.

Method 2: Drag in transcripts and how-tos

If you've got a folder of YouTube transcripts, manuals, PDFs, or how-to docs already on disk, copy them into the vault. Inside the **`raw/`** folder, create subfolders that match the type:

- raw/transcripts-and-how-tos/
- raw/screenshots/
- raw/clippings/

Don't worry about organizing perfectly — Claude will read everything and route it once you give the next prompt.

Method 3: Screenshots

Drop screenshots into raw/screenshots/ (or a topic-specific subfolder like raw/paperclip-screenshots/). Claude reads images and will reference them when generating SOPs and step-by-step guides.

7 Tell Claude to build the wiki and link the graph

Back in the VS Code terminal, give Claude the build instruction. Plain English works — something like:

PROMPT

> Read everything in raw/. For each meaningful concept, create a wiki page in wiki/. For each node, find other nodes that are systematically related and add wiki links between them. Start with the most-connected topics first. Then look at the screenshots and connect them to the right wiki pages.

Claude will read the raw files, write structured wiki pages, and add [[wiki-style links]] between them. Open Obsidian and switch to Graph View — you'll see nodes appear in real time as Claude writes pages and links them. That's the moment the brain comes alive.

8 Build the hot cache

Once the wiki has substance, ask Claude to create **`wiki-hot-cache.md`** — a synthesized summary of the entire wiki that any future agent reads first.

> **PROMPT**
> Create a file called wiki-hot-cache.md. It should be a synthesized summary of everything in the wiki — concepts, key commands, common errors, the org chart, and how the pieces connect. This is the first context any agent loads before answering questions about this domain.

This single file is what makes future queries fast. Without it, Claude reads the index, picks pages, and reads them one by one. With it, Claude grounds in the cache instantly and only opens specific pages when it needs depth.

9 Test it — generate an SOP

Time for the proof. Ask Claude to produce a real deliverable using only the wiki:

> **PROMPT**
> Using only the wiki, create a complete step-by-step SOP for installing Paperclip on a fresh Windows machine. Include screenshots from raw/screenshots where relevant. Output it as a Word doc.

If Claude produces a clean, accurate, screenshot-backed guide — the brain is working. If Claude hallucinates or skips steps, the wiki is missing data. Add the missing source files, re-run the link prompt, and try again.

Do the same with Canvas. Ask Claude to generate a Canvas view of the install flow. It'll create a visual map of nodes inside Obsidian's Canvas feature.

Adding New Content Over Time

The brain is only useful if it stays current. Here's the loop:

New article, video, or research

9. Find the source. Verify it's actually good (don't pollute the brain with junk).
10. Drop it into **`raw/`** — clip via the extension, paste a transcript, or save a markdown copy.
11. In Claude Code, say: "New file in raw/. Update the wiki — extract the concepts, create or update the relevant pages, and add wiki links."
12. Claude reads the new file, finds related wiki pages, updates them, and logs the change in CLAUDE.md.

Tool releases a new version

13. Drop the changelog or release notes into **`raw/`**.
14. Tell Claude: "New release notes in raw/. Find the affected wiki pages and update them."
15. Claude updates the affected pages and notes the version change.

Maintenance Triggers (5 Minutes Max)

Three triggers keep this thing alive forever. Set them as recurring habits.

TRIGGER	WHEN	WHAT TO DO
New release	When the tool ships a new version	Drop changelog into raw/. Tell Claude: "New file in raw/, update the wiki."
Wiki lint	Every two weeks	Tell Claude: "Run a wiki lint. Find contradictions or stale claims and report them."

TRIGGER	WHEN	WHAT TO DO
Something breaks	When a command or step fails in real use	Paste the error into Claude: "Does my troubleshooting wiki cover this? If not, add a page."

Where This Breaks (And When to Switch Tools)

Obsidian RAG is a graph + filesystem reasoning engine. It's incredible up to a point. Past that point, you switch to a vector database. Here's where it breaks:

1. Schema and index eat tokens at scale

Every session, Claude loads CLAUDE.md and the index. Keep CLAUDE.md tight (under 200 lines). Keep the index lean. Don't dump 184 transcripts in here — that hits about a million tokens by itself.

2. No semantic search

Obsidian finds pages by name and link, not meaning. So name your pages clearly. "token-usage-paperclip.md" beats "notes2.md" every single time.

3. Summaries go stale

If you update a source file outside the vault, the wiki won't auto-sync. Re-drop the updated file into raw/ and tell Claude to refresh the affected pages.

4. Even 1M context fills

At scale, the context window fills before Claude reads everything. The hot cache helps, but there's a ceiling.

5. ~100 sources is the sweet spot

Below 100 well-curated sources, Obsidian RAG is magic. Above ~10,000, it chokes. The fix is Pinecone or Supabase as a vector store, with Obsidian still doing the reasoning layer on top.

> **WHEN TO SWITCH TO PINECONE**
> Past ~100 sources, or when you're ingesting hundreds of transcripts (e.g., your full YouTube channel), spin up a Pinecone or Supabase vector index. Keep Obsidian for

> the graph and the reasoning. Hand off raw text retrieval to the vector store. Best of both worlds.

Obsidian RAG vs Pinecone RAG (The Mental Model)

OBSIDIAN RAG	PINECONE RAG
Claude loads schema (rulebook)	Docs are chunked into pieces
Reads the index (catalog)	Each chunk is embedded as a vector
Picks the most relevant pages	Vectors stored in Pinecone index
Loads pages fully (small handful)	Original text attached to each chunk
Pages already cross-reference each other	Query gets embedded into same vector space
Claude reasons across the graph	Pinecone runs nearest-neighbor search
Writes back, updates notes	Top chunks come back as context
Best for: <100 curated sources	Best for: massive archives, 1k+ docs

The marriage made in heaven: Pinecone for raw retrieval at scale + Obsidian for graph reasoning and cross-referencing. That's the next-level setup.

Vaults to Build (My Roadmap)

One brain per topic. Don't merge them. Link them later if needed.

- **Paperclip** — installation, configuration, agent setup, troubleshooting, GitHub releases
- **Claude Code** — commands, slash commands, MCP servers, hooks, subagents, common errors
- **Obsidian** — community plugins, Templater, Dataview, Canvas, advanced workflows
- **GoHighLevel** — workflows, custom fields, SaaS Mode, snapshots, troubleshooting

- **Per-business brains** — Automated Marketer, Arcytex, VCI, Rapid Active, nunomtavares.com (one each)
- **Personal master brain** — links to all of the above, holds the cross-business context (the orchestrator brain)

Quick Reference Card

Folder structure inside every vault

```
vault-name/
├── CLAUDE.md              ← schema, rules, identity (under 200 lines)
├── wiki-hot-cache.md      ← synthesized summary, first context for any agent
├── index.md               ← table of contents
├── raw/                   ← unprocessed source files
│   ├── transcripts-and-how-tos/
│   ├── clippings/
│   └── screenshots/
└── wiki/                  ← living, linked knowledge pages
```

Standard prompts to keep in your snippet manager

- **Build wiki:** "Read everything in raw/. Create wiki pages, link related nodes, start with the most-connected topics first."
- **Add new file:** "New file in raw/. Update the affected wiki pages and add wiki links."
- **Hot cache:** "Create wiki-hot-cache.md — synthesized summary of the entire wiki for fast agent context."
- **Lint:** "Run a wiki lint. Find contradictions, broken links, or stale claims. Output a report."
- **Generate SOP:** "Using only the wiki, create a step-by-step SOP for [task]. Include screenshots from raw/."

AI isn't hard. Just taught badly.

— Nuno Tavares