APRIL 2026 · FIRST EDITION

Obsidian + Claude Code
Rebuild Your Second Brain with AI

Build an AI-powered personal knowledge management system from scratch

Obsidian + Claude Code — Rebuild Your Second Brain with AI

Sources: Obsidian official docs • kepano/obsidian-skills • Karpathy LLM Wiki • hands-on practice

Version: v1.0.0

Published: April 2026

Huashu

Huashu

> Orange Book Series • Free and Open Source

This guide was written with the assistance of Claude Code, based on Obsidian official documentation, kepano's obsidian-skills project, Andrej Karpathy's LLM Wiki practice, and the author's personal knowledge management experience. AI tools evolve rapidly — please verify against official docs.

This guide is continuously updated. Follow @AlchainHust on X for the latest version.

Table of Contents

Table of Contents

Preface My Notes Were Never Organized by Me

Preface

Before writing this book, I hesitated for a moment. Because I'd never actually used Obsidian myself.

That sounds absurd. But once you finish this preface, you'll understand why someone who's never used Obsidian might actually be the most suitable author.

Let me start with a number: my "writing" folder currently has over 2,000 Markdown files. WeChat articles, video scripts, research notes, Orange Book manuscripts, training materials, reading excerpts, Prompt templates -- all in one Git repo. Every day, a dozen or so files get created or modified.

I almost never organize these files by hand.

Not because I'm lazy (okay, maybe a little), but because I have a full-time "knowledge manager": Claude Code. It categorizes, indexes, tags, and archives old projects for me. I just throw things in; it makes sure those things can be found.

This system wasn't designed. It grew.

It Started with a CLAUDE.md

In early 2025, I started using Claude Code full-time for content creation. Back then, my writing folder had just a few scattered Markdown files with no real structure. Every time Claude Code launched, it was like a new colleague parachuting in, needing a full briefing from scratch.

So I wrote the first version of CLAUDE.md -- just a few lines: who I am, what I'm doing, what's in the folder.

The effect was immediate. Claude Code stopped asking stupid questions like "where do you want to save the article?" It knew articles go in 01-WeChat-Articles/, video scripts in 03-Video-Creation/, research notes in 07-Research-Analysis/.

Then I got hooked. Every time Claude Code made a mistake -- saving a draft to the wrong place, using a dash I hate, or starting to write without searching for the latest information -- I'd add a rule to CLAUDE.md. Over six months, CLAUDE.md grew from a few lines into a routing system: the root-level CLAUDE.md dispatches tasks to different subdirectories based on keywords, and each subdirectory has its own CLAUDE.md defining specific rules.

As of now, I have 9 CLAUDE.md files spread across different workspaces. Claude Code enters any directory and knows exactly how to work there.

55 Skills

CLAUDE.md solved the "rules" problem. Skills solved the "capabilities" problem.

Skills are Claude Code's capability extension mechanism -- you can package common workflows into reusable skills. I now have 55 Skills, roughly in three categories:

The first category is perspective skills. I distilled the thinking patterns and expression styles of over twenty thinkers who've deeply influenced me (Feynman, Munger, Naval, Taleb, Karpathy, Paul Graham) into Skills. When I'm stuck writing, I invoke a perspective skill to run my argument through that person's lens, which often reveals a new angle. This project is called "Nuwa" and has over 6,000 stars on GitHub.

The second category is content creation. From topic selection, research, first drafts, triple-round proofreading, illustrations, layout, to publishing on WeChat, repurposing for X, and generating Xiaohongshu posts -- every stage has a corresponding skill. The full workflow for writing a single WeChat article calls about seven or eight skills.

The third category is utilities. PDF generation, video material analysis, danmaku generation, Feishu document operations, image hosting uploads. These are the infrastructure of daily operations.

This is my entire content operation's "operating system." Running a one-person media business, all powered by this system.

The Ceiling of a Stitched-Together Solution

But I have to admit: this system has a problem. It's stitched together.

My daily work environment is Cursor's terminal. File tree on the left to locate where I am in the knowledge base; terminal on the right where Claude Code runs. Two tools cobbled together, barely functional.

But this stitched-together approach has several obvious shortcomings.

First, no bidirectional links. My research notes mention Karpathy, and another article of mine also cites Karpathy's views, but there's zero connection between these two notes. Unless I manually have Claude Code grep for "Karpathy," these scattered fragments will never automatically connect.

Second, no knowledge graph. The relationships among 2,000+ files in my knowledge base exist only in my head and in the text descriptions in CLAUDE.md. If one day I forget what I've previously written on a topic, I can only search and hope for the best.

Third, low navigation efficiency. Finding a file in Cursor's file tree means constantly expanding folders, scrolling, clicking. No quick jump, no fuzzy search, no favorites.

These problems are exactly what Obsidian solves.

[[Bidirectional links]] make knowledge connect automatically. The graph view lets me see relationships among 2,000 files. Cmd+O fuzzy search -- type a few characters and jump to any note.

So this book is really a record of my own migration. From a stitched-together setup of "Cursor file tree + Claude Code + bare Markdown folders" to the native solution of "Obsidian + Claude Code."

You Don't Need to Be a Note-Taking Guru

While researching the Obsidian community, I noticed something interesting: many power users spend enormous time designing beautiful templates, complex tagging systems, and deeply nested folder structures. Their vaults look like works of art.

But their pain is identical to people who don't use Obsidian: maintaining the system itself becomes a heavy burden. Tag taxonomies get built then collapse, templates get revised again and again, links between notes are never complete.

AI changes the game.

You don't need to become a disciplined note-taking guru. You don't need to maintain a perfect tagging system. You don't even need to manually create bidirectional links. You just need to do one thing: **store your knowledge in a format that AI can read, and let AI handle the rest.**

Obsidian provides the best container: a local Markdown folder. Claude Code provides the best manager: an AI Agent that can read, write, search, and connect. The combination of these two is the optimal solution for personal knowledge management in 2026.

This book won't teach you how to become an advanced Obsidian user. There are already plenty of Obsidian tutorials out there. What this book teaches you is: **how to build a knowledge system that AI can understand and operate, and then let AI help you grow it over time.**

If you feel like you're not the kind of person who "organizes notes every day," then perfect -- this book was written for you.

Huashu
April 2026
Shenzhen

§01 The Information Graveyard

The Information Graveyard

Your note library is like a library whose librarian has gone missing. The books are all there, but nobody knows where anything is. AI is that librarian.

How Many Zombies Live in Your Note App

Think back to the last time you opened your note app. Not to jot something down -- everyone does that -- but to find something, to reuse something.

Chances are, you couldn't find it.

You probably have an Evernote account with articles bookmarked in 2018 that you've forgotten why you saved. You probably have a Notion workspace with dozens of "to be organized" databases that never got organized. You probably have a Notes app stuffed with random thoughts, but none of those thoughts are connected to each other.

These notes aren't knowledge. They're an information graveyard.

It's not that you didn't try. Every note was valuable the moment it was captured: a good idea, a useful piece of information, an experience worth revisiting. But between "writing it down" and "actually using it," there's a massive chasm: organizing, connecting, retrieving, reusing.

Humans are inherently bad at crossing that chasm.

Three Things You Can't Do

The human brain is an excellent association machine but a terrible indexing system.

You can't keep organizing consistently. Tag systems sound wonderful, but you won't stick with them for three months. Tagging requires discipline, and discipline is a consumable resource. The first week, you carefully tag every note with a well-designed taxonomy. By week three, you start cutting corners, dumping things into a folder called "To Be Organized." By month three, "To Be Organized" has 200 notes and you never want to open it again.

You can't make global connections. A book you read last month and a problem you encountered today might be deeply related, but you can't see it. Because human working memory can only handle about 7 chunks of information at once. When your knowledge base has hundreds or thousands of notes, there's no way you can remember what each one says.

You can't retrieve efficiently. You remember you "saw some analysis about something somewhere," but you can't recall which folder, which note, or what keywords you used. Search engines can find things on the internet for you, but in your own note library, search quality depends entirely on the words you happened to use when you wrote the note.

These three impossibilities compound: **the larger your knowledge base grows, the lower its effective utilization rate.**

The most ironic scenario: you know you wrote down something useful, but since you can't find it, you go search the internet for the same thing all over again. Your note app has become a black hole -- things go in but never come out.

What the Tools Have Tried

Note-taking tools have been trying to solve this problem for years. Each generation is slightly better than the last, but none address the fundamental contradiction.

Folders were the earliest approach. Sort notes into different folders by topic. The problem: a single note often belongs to multiple topics. An article about "AI applications in healthcare" -- does it go in the "AI" folder or the "Healthcare" folder? Wherever you put it, you won't find it from the other folder.

Tags were the fix for folders. A note can have multiple tags, free from folder constraints. But the maintenance cost of tag systems is extremely high. You need to remember which tags you've used, what each tag means, and which tags to apply to new notes. Inconsistent tagging ("AI" vs "Artificial Intelligence" vs "Machine Learning") is worse than having no tags at all.

Bidirectional links were popularized by Roam Research in 2020. If you mention note B in note A, note B automatically knows "A referenced me." Better than folders and tags, because connections form naturally as you write.

But bidirectional links have a prerequisite: you have to remember to link. You write in your journal "read a Munger speech today," but if you don't actively type [[Munger]], that journal entry has no relationship to your earlier Munger reading notes. Link quality depends entirely on your diligence.

Folders, tags, bidirectional links -- these three approaches share a common problem: **they all place the responsibility of organizing on the human.**

What AI Changes

Humans are bad at indexing and connecting. AI is good at it. Humans are good at generating ideas and making judgments. AI isn't. So let each do what they're best at.

Humans handle: recording, thinking, deciding.

AI handles: organizing, connecting, retrieving, maintaining.

This is already happening.

Stefan Imhoff is a German developer who keeps a journal in Obsidian. His daily entries mention people, places, and book titles. He used to spend 10-15 minutes every day manually adding bidirectional links to those names: searching the vault for corresponding notes, creating new ones if they don't exist, then inserting [[links]] back into the journal.

Then he started using Claude Code.

Claude Code reads his journal, automatically identifies all the names of people, places, and books. It searches the vault for existing entity notes, creates new ones if needed. Then inserts [[wikilinks]] back into the original text. The whole process takes seconds. 10-15 minutes of manual work, reduced to a single command.

Another case. A writer at MakeUseOf had accumulated 5 years of notes scattered across various folders -- no tags, no links, a complete mess. He'd tried to manually organize them several times, but every time he opened that folder he felt despair, then closed it.

He used Claude Code + Obsidian and sorted it all out in 90 minutes.

Claude Code read all the notes, automatically categorized them by topic, added frontmatter to each note (title, tags, summary), established bidirectional links between notes, and generated index files. Five years of accumulated knowledge, from chaos to order, in under two hours.

There's an even more interesting case. A product manager shared four daily workflows using Obsidian + Claude Code: having AI auto-summarize notes when researching a new domain, having AI search the vault for historical discussions before making product decisions, having AI generate first drafts based on existing knowledge when writing requirement docs, and having AI extract insights from journal entries during weekly reviews.

Notice what these four workflows have in common: **humans make the decisions, AI does the organizing and retrieving.** Each doing what they're best at.

But AI Isn't Magic

So can't you just dump your notes into ChatGPT and call it a day?

Not that simple. AI can help you organize notes, but only if it can actually access them.

If your notes are in Apple Notes, AI can't touch them. It's a closed system with no external file interface.

If your notes are in Notion, AI can only access them indirectly through an API -- limited permissions, limited speed, limited operations. You can't let AI freely read, write, and search.

If your notes are in Evernote, it's even worse. Proprietary format, difficult to export, and AI has to do format conversion before it can even understand the content.

If your notes are in a folder, all in Markdown format, then AI can directly read, write, search, and reorganize. Zero friction.

This is why note format matters. **The format you choose to store your knowledge determines how much AI can help you.**

Choose Markdown, and AI can be your full-time knowledge manager. Choose a proprietary format, and AI can only look at your notes through a pane of glass.

The Librarian Is Back

Imagine your note library as a real library. Books keep coming in, but the librarian disappeared years ago. No index cards, no classification shelves -- books piled on the floor, the pile growing taller. Occasionally you need to find a book, dig around for ages, give up, and go buy a new copy online.

AI is that librarian coming back. And it's a librarian that never sleeps, never slacks off, and can process every book simultaneously.

But for the librarian to do its job, there's one prerequisite: it needs to be able to touch the books. If your books are locked in glass cases (cloud-based proprietary formats), the librarian can only look through the glass. If the books are on open shelves (local Markdown files), the librarian picks them up and immediately starts categorizing, labeling, and indexing.

This leads to two questions: what kind of container is most AI-friendly? And what kind of AI is best suited to manage knowledge?

§02 Why Obsidian

Why Obsidian

Three completely independent billion-dollar projects independently chose the exact same approach for storing AI memory. That's not a coincidence -- it's inevitable. And Obsidian happens to be standing right at the center of that inevitability.

A Folder Is a Vault

Let's start with the simplest thing: what is an Obsidian vault?

A folder.

Not a database, not a cloud service, not a proprietary format. Just a folder on your computer, filled with .md files. Open it in Finder and you see a bunch of Markdown files and subfolders. Open it in VS Code and you can edit them normally. Drag it into a terminal and Claude Code can read and write directly.

What Obsidian does is add a UI layer on top of that folder: bidirectional links, graph view, search, plugins. But the underlying files are standard Markdown from start to finish. You can stop using Obsidian at any time, switch to any tool that reads Markdown, and your notes won't lose a single character.

This design decision seems unremarkable, but in the age of AI, it becomes Obsidian's greatest competitive advantage.

Markdown Is the Native Language of LLMs

This isn't a metaphor -- it's a technical fact. Markdown is one of the primary formats in LLM training data. Billions of README.md files on GitHub, technical documentation, blog posts, tutorials -- LLMs have read them all. Their understanding of ## headings, **bold**, and - lists is as natural as your understanding of punctuation.

Specifically, three properties:

High token efficiency. The same information represented in Markdown uses 30-50% fewer tokens than JSON or XML. This means within the same context window, an LLM can "see" more of your notes at once. With Claude's 1M token context window, if your notes are in Markdown, it can fit roughly 2-3 million words -- enough to cover most people's entire knowledge base.

Naturally clear structure. ## is a heading, > is a quote, ``` is a code block. These markers are natural structural delimiters. You don't need to tell AI "this is a heading, this is body text" -- the Markdown syntax itself says it.

AI's default output format. Ask ChatGPT or Claude a question and the response comes in Markdown. AI reads Markdown most naturally and writes Markdown most naturally. When your knowledge base is in Markdown, AI feels right at home.

Local Files = AI Can Read and Write Directly

Obsidian's other key decision: data stored entirely locally.

This used to be seen as a disadvantage: how do you sync across devices? What if you lose data without cloud backup? But in the age of AI Agents, "data lives locally" becomes a decisive advantage.

Here's how Claude Code works: it runs in a terminal and directly operates on the local file system. It can read files, write files, search files, move files, rename files. Every operation is direct access to the local file system -- no API needed, no authentication, no network requests.

Point Claude Code at an Obsidian vault and it can work immediately:

推荐

Obsidian (local Markdown):
Claude Code cd into vault directory -> directly read/write all notes -> zero config, zero latency, full permissions

不推荐

Notion (cloud proprietary format):
Requires API setup -> limited by permissions and rate limits -> can't freely search or batch-operate -> lossy format conversion

This difference isn't "slightly better vs slightly worse experience." It's "can do vs can't do." Inside an Obsidian vault, Claude Code is a manager with full access. Inside Notion, Claude Code is an outsider who can only pass messages through an API window.

Convergent Evolution

If this were just theoretical analysis, you might think "okay, makes sense, but whatever." But when you see three completely independent billion-dollar projects make the exact same choice, you have to take it seriously.

Manus. An AI Agent company, acquired by Meta for $2 billion in 2026. What do their Agents use to store memory during long tasks? task_plan.md and notes.md. Markdown files.

OpenClaw. An open-source AI Agent framework with 355,000+ stars on GitHub. Where is Agent knowledge stored? MEMORY.md. Where is Agent personality defined? SOUL.md. All Markdown files.

Claude Code. Anthropic's AI coding tool. Where is project context stored? CLAUDE.md. Where are user preferences and long-term memory stored? Markdown files in the memory/ directory.

Three different teams, solving different problems, serving different users. Without referencing each other, they made the same architectural decision: **use Markdown files as the memory layer for AI Agents.**

Not a vector database. Not SQL. Not JSON. Not any fancy technical solution. Just .md files.

Why?

Dimension	Markdown Files	Vector Database
Latency	Near-zero (local file read)	Requires network call + embedding computation
Cost	$0 (local storage)	$50-200/GB/month
Readability	Any text editor can view it	Black box, requires specialized tools
Version Control	Native Git support	Requires additional solutions
Human Intervention	Open and edit directly	Requires specialized tools
Portability	Just copy the files	Vendor lock-in

Vector databases have an edge for semantic search at the million-record scale. But for personal knowledge management -- a few thousand to tens of thousands of notes -- Markdown files win on every dimension. They're faster, cheaper, more transparent, more controllable, more durable.

And an Obsidian vault is exactly a bunch of Markdown files. Every note you accumulate in Obsidian, every link, every tag is structured data that AI can directly read and operate on.

Official Endorsement

Obsidian's CEO is Steph Ango, known online as kepano. He's very active in the Obsidian community and has strong personal influence on the product's direction.

In January 2026, kepano published a repository on GitHub: obsidian-skills.

This isn't a community enthusiast's side project. This is Obsidian's CEO personally building official Skills that teach Claude Code how to properly handle Obsidian-specific formats. As of this writing, the repo has over 20,000 stars.

obsidian-skills includes several Skill files:

- **obsidian-markdown**: teaches Claude Code to properly handle .md files, including [[wikilinks]], callouts, frontmatter, and other Obsidian-specific syntax
- **obsidian-bases**: teaches Claude Code to handle .base files (Obsidian's database feature)
- **json-canvas**: teaches Claude Code to handle .canvas files (Obsidian's spatial note maps)
- **defuddle**: a web content cleaning tool that strips ads, navigation bars, and page decorations before saving web content to the vault, saving tokens

kepano said on X:

> I'm starting a set of Claude Skills for Obsidian... so far they're centered around helping Claude Code edit .md, .base, and .canvas files.

The signal is clear: Obsidian officially considers AI Agents operating on vaults a core use case, and is personally providing support for it.

For comparison, Notion's AI is a built-in closed system -- you can only use Notion's own AI, you can't choose models, and you can't let external Agents directly operate on your data. LogSeq is also local-first Markdown, but its AI plugin ecosystem and community scale are far behind Obsidian's.

Obsidian vs Notion vs LogSeq

Let's go straight to the comparison.

Dimension	Obsidian	Notion	LogSeq
Data Storage	Local Markdown files	Cloud proprietary format	Local Markdown/EDN
AI Agent Accessibility	Very high (direct file system read/write)	Low (requires API)	High (local files)
AI Integration	Open: choose any model or tool	Closed: built-in Notion AI	Community plugins
Data Ownership	Fully yours	On Notion's servers	Fully yours
Team Collaboration	Weak (requires paid Sync)	Strong (native real-time collaboration)	Weak
Learning Curve	Moderate	Gentle	Steep
Community Size	Largest (Reddit 100K+)	Large	Small
Plugin Ecosystem	1,000+ community plugins	Integration marketplace	200+ plugins
Price	Free	From $10/month	Free & open source
Open Source	No (free but closed source)	No	Yes

Bottom line: **choose Notion for team collaboration; choose Obsidian for personal knowledge management + AI.**

This isn't to say Notion is bad. Notion 3.0's AI Agent capabilities are impressive -- it can autonomously edit documents, operate databases, and build forms. If your core need is team collaboration, project management, and knowledge sharing, Notion is still the best choice.

But if your core need is personal knowledge management and you want AI deeply involved in the process, Obsidian's architectural advantage is decisive:

First, Agent accessibility. Claude Code entering an Obsidian vault is like walking into its own home -- it can look at anything, change anything. Entering Notion is like visiting someone else's office -- swipe a badge, sign in, wait for approval, and you can only work in designated areas.

Second, open ecosystem. Today you use Claude Code; tomorrow something better might come along. Obsidian doesn't lock in your AI choice -- the vault is just a folder, any AI can operate on it. Notion's AI is built in; you can only use what they give you. In an era of rapid AI iteration, an open architecture means you can always use the latest and best tools.

Third, data durability. If Obsidian shuts down tomorrow (unlikely, but hypothetically), you won't lose a single character -- your notes are just files on your hard drive. If Notion stops its service, you need to export your data, and export processes inevitably lose format and structure.

LogSeq is the closest competitor to Obsidian -- also local-first, Markdown storage, and fully open source. If you care about open source, LogSeq is worth considering. But its AI plugin ecosystem and community activity are an order of magnitude behind Obsidian's. The value of a knowledge management tool comes largely from its community: templates, plugins, tutorials, best practices. Obsidian's advantage here is very clear.

What Obsidian Gives You on Top

If all you need is a folder for Markdown, you could just use Finder. Why install Obsidian?

Because Obsidian does several critical things on top of the folder.

[[Bidirectional links]]. In any note, type [[and Obsidian pops up a list of all your notes -- select one and the link is established. The linked note automatically shows "who linked to me." This means connections between knowledge are bidirectional and automatically tracked. You don't need to maintain an index; the links themselves are the index.

Graph view. Open Graph View and you see your vault's entire knowledge network. Each note is a node, links are lines between nodes. Isolated notes drift at the edges; densely linked notes cluster at the center. This view isn't just pretty -- it helps you discover gaps and clusters in your knowledge structure.

Quick search and navigation. Cmd+O opens the quick switcher -- type a few characters and jump to any note. Cmd+Shift+F for full-text search, blazing fast. Compared to browsing through file trees, it's an order of magnitude more efficient.

Community plugins. Obsidian has over 1,000 community plugins covering everything from calendars, kanban boards, and dataview to AI integrations. Chapter 8 will cover which ones are worth installing.

These features share a common trait: they enhance the connections and discoverability between pieces of knowledge. A folder lets you store things; Obsidian lets you build a network on top of what you store. And that network is extremely valuable to AI too. Claude Code can traverse related knowledge along bidirectional links, rather than processing individual files in isolation.

Huashu's Perspective: Why I Decided to Migrate

Before writing this book, my knowledge management tool was Cursor's file tree + Claude Code.

As I mentioned in the preface, I had a system of CLAUDE.md files + Skills, with separate workspaces and rules for WeChat articles, videos, and Orange Books. When writing an article, I'd invoke seven or eight skills to handle everything from research to illustrations to publishing. It worked.

But as I studied Obsidian + Claude Code, I realized: many things I'd been manually implementing through my stitched-together approach are native features in Obsidian.

My Stitched-Together Approach	Obsidian Native Approach
CLAUDE.md router (dispatch tasks by keywords)	MOC (Map of Content), with [[link]] one-click navigation
Each subdirectory's CLAUDE.md	Each folder's index.md, same function
_knowledge_base/INDEX.md index	Native search + graph view + semantic search plugins
Cursor file tree navigation	Cmd+O fuzzy search + favorites + recent files
Claude Code grep search	Native full-text search + Smart Connections semantic search
None	[[Bidirectional links]] + backlinks
None	Graph view
None	1,000+ community plugins

The first five rows show my methodology was correct. Without knowing Obsidian community best practices, I independently evolved nearly identical organizational patterns. The last three rows show there's significant room for improvement at the tooling level.

Bidirectional links excited me the most. My research notes frequently mention the same people and concepts, but those notes have no automatic connections between them. In Obsidian, the moment I type [[Karpathy]] anywhere, all notes mentioning Karpathy automatically form a network. Claude Code can traverse that network instead of relying on grep to get lucky.

Graph view was the second. The relationships among my 2,000+ files existed only in my head and in CLAUDE.md's text descriptions. In Obsidian, those relationships are visualized -- at a glance you can see which topics are tightly connected and which areas are islands.

So migration was inevitable.

> **For those on the fence:** if you're like me and already manage knowledge with a Markdown folder, migrating to Obsidian is nearly zero cost. Open Obsidian, select your folder as a vault, done. Every file you have stays intact -- Obsidian just adds a UI layer on top. If you don't have any knowledge management system yet, even better -- just start with Obsidian.

I think five years from now, looking back, 2025-2026 will be seen as a watershed for personal knowledge management. Not because of some new concept, but because AI became genuinely capable of managing your knowledge for the first time. And Obsidian happens to be the container that lets AI reach in.

Next chapter: 30 minutes, build a usable vault from scratch.

§03 Get Started with Obsidian in 30 Minutes

Get Started in 30 Minutes

The first two chapters covered the why. This chapter is where we get our hands dirty. The goal is clear: in 30 minutes, you'll have a working vault, your first note written, and an understanding of the three most important features.

Download and Install

Go to obsidian.md and download the installer. It's free on all platforms. No registration, no account, no trial period.

After you open it, Obsidian asks you just one thing: pick a folder to be your vault.

If you already have a folder full of Markdown files, just point it there. Obsidian won't touch any of your files — it simply adds a UI layer on top of that folder. Check in Finder: every file is exactly the same, with just one new .obsidian/ config directory added.

Starting from scratch? Just create an empty folder.

Write Your First Note

Once you open the vault, there's a "New Note" button in the top left (or press Cmd+N / Ctrl+N). Click it, and a blank note appears.

What should you write?

Anything. A thought you had today, something you're working on, a passage you just read.

I know what you're thinking: shouldn't I design a categorization system first? Shouldn't I create some folders?

No.

推荐

Good approach:
Write 100 notes first, then decide how to organize based on actual content

不推荐

Bad approach:
Spend 3 hours designing a tagging system and folder structure before writing a single note

The most common trap for beginners is "system first." You spend an afternoon researching methodologies, carefully designing a perfect categorization system. A week later you realize your actual

notes don't fit your preconceived structure, and you start over from scratch.

A note system should grow out of your notes, not out of a blueprint.

Chapter 5 of this book covers vault architecture design in detail, but that comes after you have 100 notes. For now, just write.

Three Features That Actually Matter

Obsidian has a lot of features, but three are what truly set it apart from an ordinary Markdown editor.

Feature One: Bidirectional Links [[]]

In any note, type two left brackets [[. Obsidian pops up a search box listing every note in your vault. Pick one, hit Enter, and the link is created.

Say you write in today's journal: "Read [[Poor Charlie's Almanack]] today." That [[Poor Charlie's Almanack]] becomes a clickable link. Click it, and you jump straight to the reading notes you wrote earlier.

What if there's no "Poor Charlie's Almanack" note in your vault yet? No problem — Obsidian creates a blank note waiting for you. Create the link now, fill in the content later.

That's the first half. The second half is even more interesting.

Open your "Poor Charlie's Almanack" note and scroll down. There's a "Backlinks" panel. It tells you which notes in your vault link to this one. Your journal links to it, your "Investment Principles" note links to it, your "Munger Quotes" note links to it too.

You don't need to maintain any index. Link relationships are tracked automatically. Just add a [[]] as you write, and Obsidian keeps track of what's connected to what.

> **A comparison:** In my previous setup, connections between knowledge were entirely dependent on Claude Code using grep to search — basically guessing. With [[bidirectional links]], connections are explicit and traceable. Claude Code traversing knowledge along links is an order of magnitude more efficient than grep.

Feature Two: Tags #tag

Type # anywhere in a note, followed by a word, and you've created a tag. For example: #reading-notes #investing #munger.

A note can only live in one folder, but it can have as many tags as you want. That piece on "AI Applications in Healthcare" — does it go in the "AI" folder or the "Healthcare" folder? With tags, no dilemma: tag it #AI

and #healthcare at the same time, and you can find it from both dimensions.

Tags don't need to be pre-defined. Just type whatever comes to mind, and Obsidian automatically collects every tag that's ever been used.

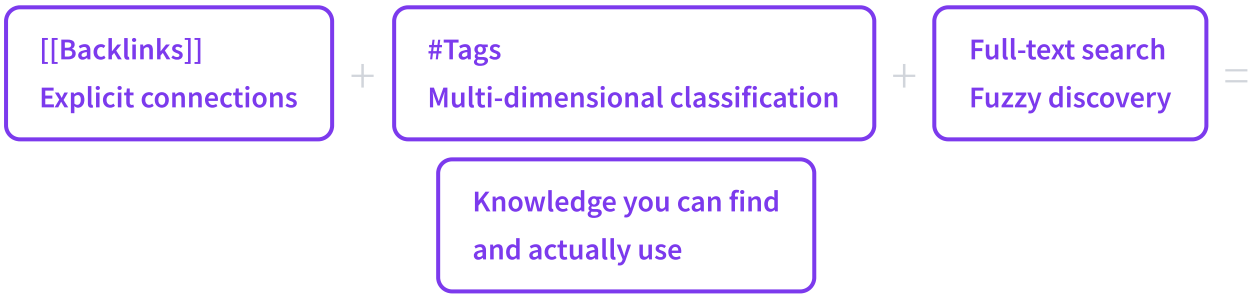
Feature Three: Full-Text Search

Cmd+Shift+F (Ctrl+Shift+F on Windows), type a keyword, and search across everything in your vault.

Thousands of notes, instant results. Supports regex, filtering by tags, folders, and time ranges.

You remember "I read something about XX somewhere," but can't recall which note. Full-text search means you don't need exact memory — a fuzzy search gets you there.

Together, these three features form a complete knowledge discovery system:



Graph View: See Your Knowledge Network

Press Cmd+P, type "graph," and open it. Each note is a dot, and each [[bidirectional link]] is a line connecting two dots.

At first, there are just a few scattered dots. Don't worry. After a few dozen notes, the graph starts getting interesting:

Some notes get linked to by many others — they become hub nodes in your network. These are usually your most important concepts.

Some notes float alone at the edges with no links at all — these orphans are a reminder that they should connect to something, you just haven't built the link yet.

Some notes form clusters you didn't expect — helping you discover hidden connections between ideas.

Graph view isn't essential. But as your vault grows, it's a great "knowledge health check tool" that helps you see what your knowledge structure actually looks like.

Starter Plugins: Install Only Two

Obsidian has over 1,000 community plugins. Ignore them for now.

Install just two core plugins. Go to Settings -> Community Plugins -> Browse, and search to install.

Daily Notes

Once enabled, a note named after today's date is automatically created each day. Open Obsidian, and your daily note is already there waiting.

Daily notes are the lowest-friction entry point in your entire note system. No need to think of a title, pick a folder, or decide what category a note belongs to. Whatever comes to mind today, just write it in your daily note. Done. Organizing comes later (Chapter 7 covers how to let AI extract insights from your daily notes).

Templates

Create note templates and apply them with one click when creating new notes. For example, you could make a "Book Notes Template":

Field	Content
Book Title	(fill in)
Author	(fill in)
Core Ideas	(fill in)
My Thoughts	(fill in)
Related Notes	[[]]

The benefit of templates is keeping your notes structurally consistent. Consistently structured notes are very AI-friendly — Claude Code knows exactly where to find each piece of information. This advantage becomes apparent in Chapter 4.

> **A principle about plugins:** The biggest mistake beginners make is installing twenty plugins right off the bat. Every plugin has a learning curve, and too many means you won't use any of them well. Use the basics for two months first. When you clearly know what capability you're missing, then go find the right plugin. Chapter 8 has a curated recommendation list for you.

A Note for Cursor/VSCode Users

If you use Cursor or VS Code regularly, Obsidian's interface will feel very familiar.

Feature	VS Code / Cursor	Obsidian
File Browser	Left sidebar file tree	Left sidebar file tree
Multi-file Editing	Tabs	Tabs
Quick Jump	Cmd+P	Cmd+O (nearly identical)
Command Palette	Cmd+Shift+P	Cmd+P
Split Editing	Supported	Supported
Custom Shortcuts	Supported	Supported
Plugin Ecosystem	VS Code Extensions	Community Plugins

The workflow is nearly identical — zero learning curve to switch over.

The fundamental difference is in the design goal. An IDE is built for code; Obsidian is built for knowledge. They look alike, but solve completely different problems.

One fun approach is using both together. Run Claude Code in Cursor to operate on vault files, and use Obsidian to browse the knowledge graph and backlinks. Code tools do the heavy lifting; knowledge tools handle navigation.

Four Common Beginner Mistakes

Let's wrap up with what not to do.

Mistake One: Spending too much time designing a system. You haven't even written 50 notes, but you're already researching every methodology out there. You try each one for three days before switching. Result: barely any notes, but five or six abandoned systems. Write first, organize later — that's always the right order.

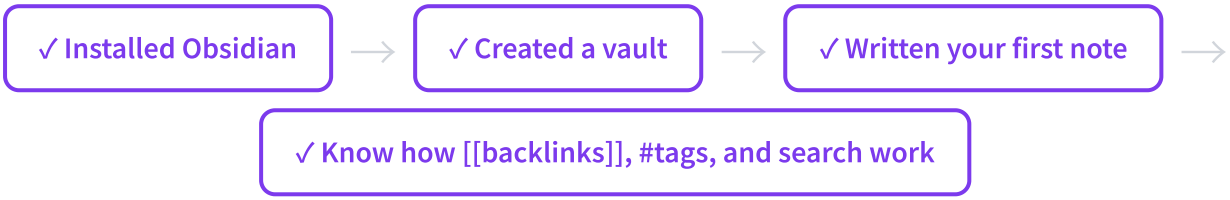
Mistake Two: Installing too many plugins. The community plugin list is like a candy store — everything looks tempting. You install twenty plugins, your sidebar fills with unrecognizable icons, shortcuts conflict, startup slows down. Two weeks later you disable them all, back to square one. At the start, all you need is Daily Notes and Templates. Really.

Mistake Three: Trying to categorize perfectly. Every note needs the right folder, the right tags, the right links. This perfectionism turns note-taking into a stressful chore, and eventually you stop taking notes altogether. Remember: links matter more than folders, and organizing later is more important than being perfect now. Plus, after Chapter 4, you can hand the organizing work to AI.

Mistake Four: Starting from someone else's complex vault. There are plenty of people online sharing vault templates they've used for three years — dozens of folders, hundreds of templates. Looks impressive, but that structure evolved from thousands of notes over time. Copying it directly is a recipe for frustration. Start with an empty vault and let the system grow naturally.

30-Minute Checklist

If you've been following along, by now you should have:



That's enough.

There's a lot more to Obsidian you haven't seen yet. Canvas, Bases, Publish. No rush — explore them over time. The most important thing right now is to start writing notes and building up content.

Next chapter, the real star of the show enters the stage.

§04 Enter Claude Code

Enter Claude Code

Last chapter, you set up your vault in 30 minutes. Now, let's bring AI into the picture. The whole process takes less than 10 seconds: open your terminal, cd to your vault directory, type claude. That's it.

4.1 The Simplest Connection: Just cd In

When most people hear "AI + Obsidian," their first reaction is: I need to install a plugin, configure an API, set up MCP.

None of that. Open your terminal, two commands:

```
cd ~/Documents/my-vault
claude
```

Once Claude Code launches, it automatically has read and write access to the entire vault directory. Reading notes, creating files, searching content, renaming, moving, deleting — all of it. No configuration needed.

Why? Because a vault is just a folder full of Markdown files. Claude Code operates on the local file system. The two are naturally compatible — no translation layer required.

You can also install the Obsidian-Shell plugin to open a terminal right inside Obsidian. The default working directory is the vault root, so you don't even need to cd.

推荐

Claude Code + Obsidian vault:
cd into the directory → directly read/write all .md files → zero config, zero latency

不推荐

Other AI + note tools:
Configure API key → install integration plugin → limited permissions → can only operate indirectly through APIs

The first time you launch Claude Code, it scans the current directory. If your vault already has dozens or hundreds of notes, try asking: "What's in my vault? Give me an overview." See how it responds. This is a great smoke test to confirm the connection works.

4.2 CLAUDE.md: The Employee Handbook for Your AI

Connected, but not enough.

Imagine you've just hired a new assistant. On their first day, you point at a room full of filing cabinets and say, "Everything's in there, get to work." They'll either be paralyzed or start rummaging around randomly. What's missing is an employee handbook.

CLAUDE.md is that handbook.

Create a file called CLAUDE.md in your vault's root directory. Claude Code automatically reads it every time it starts up. Whatever you write in there, the AI follows accordingly.

A CLAUDE.md for knowledge management should include at least four sections:

Section One: Who you are. Your identity, areas of interest, preferences. Three to five lines is plenty.

Section Two: How the vault is organized. Folder structure, naming conventions, what each area is for. Once the AI reads this, it knows where things belong and won't misplace anything.

Section Three: Behavioral boundaries. What it can and can't touch. For example: "Feel free to add tags and links, but don't delete any existing content."

Section Four: Note templates. What a new note should look like, which frontmatter fields to include. With templates in place, notes created by AI will be consistent with the ones you write by hand.

Huashu's CLAUDE.md Router

My own CLAUDE.md is considerably more complex than what I described above.

The "Writing/" directory has 9 workspaces:

```
Writing/
├── CLAUDE.md              ← Router: dispatches tasks to subdirectories based on keywords
├── 01-WeChat-Articles/
│   ├── CLAUDE.md          ← Dedicated rules for WeChat article writing
│   ├── projects/          ← Articles in progress
│   ├── _knowledge_base/   ← Knowledge base with INDEX.md
│   └── _archive/          ← Published articles
├── 03-Video-Production/
│   ├── CLAUDE.md          ← Dedicated rules for video production
│   └── ...
├── 10-Orange-Books/
│   ├── CLAUDE.md          ← Dedicated rules for Orange Book production
│   └── ...
├── .claude/skills/        ← 55 Skills
└── tools/                 ← Utility scripts
```

The root CLAUDE.md is a router. It contains a routing table:

```
| Keyword              | Workspace         | File to Read                       |
|----------------------|-------------------|------------------------------------|
| Write article, WeChat| WeChat Articles   | /01-WeChat-Articles/CLAUDE.md      |
| Video script         | Video Production  | /03-Video-Production/CLAUDE.md     |
| Orange book, epub    | Orange Books      | /10-Orange-Books/CLAUDE.md         |
| Research, competitor | Research & Analysis| /07-Research-Analysis/             |
```

When I say "help me write a WeChat article," Claude Code reads the routing table in the root CLAUDE.md and automatically jumps to read /01-WeChat-Articles/CLAUDE.md. That file defines the complete ruleset for WeChat writing: style requirements, proofreading standards, image guidelines, publishing workflow.

When I say "make an Orange Book," it goes to read /10-Orange-Books/CLAUDE.md, which contains the dedicated workflow: how to write chapter fragments, how to build the EPUB, how to publish on WeRead.

One person managing this many workspaces isn't about being hyper-efficient — it's about Claude Code reading the CLAUDE.md and knowing how to handle everything, so I don't have to explain from scratch every time.

You don't need to build something this complex from the start. My system grew organically over more than a year. Every time Claude Code made a mistake, I'd add a rule to CLAUDE.md. It started as just a few lines and gradually evolved into a routing system.

> **CLAUDE.md grows naturally.** Don't try to write the perfect employee handbook on day one. Start with the basics: who you are, how your folders are organized. Then every time the AI does something wrong, add a rule. Three months later, you'll look back and find it's become a remarkably thorough guide.

A CLAUDE.md Template for Knowledge Management

If your vault is primarily for personal knowledge management, you can use this template directly:

````
# My Knowledge Base

## About Me
- [Your identity/profession]
- Areas of interest: [list 3-5]
- Preferences: write in [your language], don't over-format

## Vault Structure
- daily/: Daily entries, named YYYY-MM-DD.md
- notes/: Permanent notes, named "topic-title.md"
- projects/: Active projects, one folder per project
- raw/: Unsorted raw materials
- archive/: Completed or inactive content

## Note Template
Use this frontmatter when creating new notes:
```yaml
---
title: Note title
tags: []
created: YYYY-MM-DD
type: permanent
summary: One-line summary
---
```

## Behavior Rules
- Allowed: adding tags, creating [[bidirectional links]], generating summaries, organizing ar
- Not allowed: deleting existing note content, modifying my original records
- New notes must follow the frontmatter template above
- Update the relevant folder's index.md after each organizing session

## Tag System
- By domain: #tech #business #reading #life
- By status: #todo #in-progress #done
- By type: #idea #reference #project
````

Copy this template, customize it for your situation, and drop it in your vault root. That's your starting point.

4.3 index.md: The Navigation File That Does 80% of the Work

CLAUDE.md gives AI the global rules for the entire vault. But with ten folders and dozens of files in each, how does the AI know where to look for things?

Michael Crist discovered a ridiculously simple trick in practice: **put an index.md in each major folder.**

An index.md doesn't need to be complicated. Three to five lines will do:

```
# Reading Notes

This folder stores all reading notes, named "Author-Title.md".

## Key Files
- [[Kahneman-Thinking-Fast-and-Slow]] - The classic on cognitive biases
- [[Munger-Poor-Charlies-Almanack]] - Cross-disciplinary mental models
- [[Taleb-Antifragile]] - Benefiting from uncertainty
```

That little bit of content makes a huge difference. Without index.md, Claude Code enters a folder, sees 50 files, and has to scan through them one by one. With an index.md, it reads that file first and knows the folder's purpose and where the important files are within 3 seconds.

Interestingly, I independently came up with the exact same approach without knowing about his work. That tells you this pattern is a natural evolution, not something anyone invented.

> **核心建议**
>
> You don't need to manually write an index.md for every folder. Once Claude Code is connected to your vault, just ask it to generate them. It'll scan the folder contents and automatically write descriptions and key file lists. This task itself can serve as your "first collaboration" exercise.

4.4 Installing kepano/obsidian-skills

Chapter 2 mentioned that kepano, Obsidian's CEO, published obsidian-skills on GitHub. Let's install it now.

The process is simple. Create a .claude/skills/ folder in your vault directory (if it doesn't exist yet), then copy the Skill files from obsidian-skills into it:

```
# Run in your vault directory
mkdir -p .claude/skills
# Copy the obsidian-skills files here
```

Claude Code will automatically load these Skill files the next time it starts.

Why do you need this? Obsidian's Markdown has some unique syntax that Claude Code might not know about:

Obsidian Syntax	Standard Markdown	Difference
`[[Note Name]]`	`[text](url)`	Wikilinks — Obsidian's unique bidirectional link syntax
`> [!note] Title`	`> Quote text`	Callout blocks — typed and collapsible quote blocks
`.canvas files`	No equivalent	JSON Canvas — spatial note maps
`![[image.png]]`	`![alt](path)`	Embed syntax — can embed notes, images, PDFs

Once installed, Claude Code knows to use [[]] instead of [text](url) in your vault, understands callout syntax, and knows the .canvas file format. A one-time setup that takes 30 seconds.

4.5 First Collaboration: Let AI Organize Your Notes

Connection is set up, CLAUDE.md is written, Skills are installed. Time to do something real.

Grab a bunch of scattered notes and toss them into your vault. These could be things you jotted down in your phone's notes app, article excerpts you saved in WeChat, or random passages you wrote while reading. Format doesn't matter — just throw them in.

Then tell Claude Code in the terminal:

```
Organize the scattered notes in my vault.
Read all files without frontmatter, categorize them by topic,
add frontmatter to each note, create bidirectional links,
then generate index.md files.
```

What you'll see next is Claude Code getting to work:

Read → Scans all files in the vault, understands each note's content

Categorize → Moves notes to corresponding folders by topic

Annotate → Adds frontmatter to each note (title, tags, summary)

Link → Discovers relationships between notes, inserts [[bidirectional links]]

Index → Generates index.md for each folder

Five minutes. Maybe faster.

After the organizing is done, open Obsidian's graph view. You'll see dots that were previously scattered start forming a network. Lines begin appearing between notes. A note about "attention" connects to your Kahneman reading notes, and also to a reflection on social media. These connections used to exist only as vague impressions in your head — now they're visible, clickable links.

All you did was toss things in. The AI turned it into a structured knowledge base. **You handle the recording and thinking; AI handles the organizing and connecting.**

> **A small suggestion:** After AI organizes things the first time, spend a few minutes reviewing the results. Check if the categorization makes sense, if the tags are accurate, if the links are meaningful. If anything's off, add a rule to CLAUDE.md. For example: "Tags for tech notes must include the specific technology name — don't just use a generic #tech." This is how CLAUDE.md grows naturally.

In under 20 minutes, your vault went from a passive folder to a knowledge system with an AI librarian on duty.

But the architecture is still fairly basic. Next chapter covers how to design your vault structure so that collaboration between you and AI becomes even more efficient.

§05 Design Your Vault Architecture

Design Your Vault

In the last chapter, you connected Claude Code to your vault in under 20 minutes. It works now, but there's a gap between "works" and "works well"—and that gap is architecture. This chapter is about how to organize your vault so AI can understand and operate on it efficiently.

Six Design Principles

These aren't theory. They come from repeatedly stumbling while collaborating with Claude Code. Violate a few of them, and you'll notice a clear drop in AI performance.

Principle 1: Markdown Is the Only Format

Don't store PDFs, Docx files, or rich text in your vault.

Any information you want AI to process should be converted to Markdown before it goes into the vault. Read a book? Notes go in .md. Received a PDF report? Extract the key content into .md. Recorded a meeting? AI transcription saved as .md.

You can keep the original files, but store them outside the vault. Only Markdown lives inside the vault. That's the bottom line.

推荐

Inside the vault:
reading-notes/poor-charlies-almanack.md (your excerpts and reflections)
meeting/2026-04-11-product-review.md (meeting notes)

不推荐

Don't put in the vault:
poor-charlies-almanack.pdf (original scanned PDF)
meeting-recording-0411.m4a (original audio)

Principle 2: Keep Terminology Consistent

In one note you write "RAG," in another "Retrieval-Augmented Generation," in a third "retrieval augmented generation." To you, they're the same thing. But when Claude Code searches for "RAG," it only hits the first one.

The fix: standardize terminology across your vault. Pick the form you use most often and stick with it.

Standardize on	Don't mix with
RAG	Retrieval-Augmented Generation / retrieval augmented generation
Claude Code	CC / claude-code / Anthropic's coding tool
LLM	large model / large language model / foundation model
frontmatter	metadata / YAML header / file header info

You can place a glossary.md (terminology list) in the vault root, listing the standard terms you've chosen. Claude Code will reference it when processing your notes.

Principle 3: Flat First

Don't nest folders more than three levels deep.

"Learning/AI/LLM/RAG/Papers/"—five levels of nesting, precisely categorized, looks tidy. But for AI, each level of nesting consumes path tokens, and deeply nested folders easily get overlooked during traversal.

A better approach: use tags and links for categorization, and let folders handle lifecycle only (active / completed / archived).

推荐

Flat structure (recommended):
notes/rag-retrieval-strategies.md (tags: #AI #RAG)
notes/rag-chunking-methods.md (tags: #AI #RAG)
notes/transformer-attention.md (tags: #AI #LLM)

不推荐

Deep nesting (not recommended):
learning/AI/LLM/RAG/retrieval-strategies/notes.md
learning/AI/LLM/RAG/chunking-methods/notes.md
learning/AI/LLM/fundamentals/attention-mechanism/notes.md

Rule of thumb: if your file path has more than 3 slashes (root/level1/level2/file.md), it's time to flatten.

Principle 4: Every Note Needs a Summary

Add a summary field in the frontmatter—one sentence that says what this note is about. When Claude Code scans the vault, it doesn't need to read the full text; the summary alone tells it whether this note is relevant to the current task.

```
---
summary: Compared three RAG retrieval strategies (vector/keyword/hybrid); conclusion is hybri
---
```

Tip for writing summaries: imagine a friend asks you "What's this note about?" and you answer in one sentence. That's your summary.

Principle 5: Five Frontmatter Fields

Five recommended fields:

```
---
title: RAG Retrieval Strategy Comparison
tags: [AI, RAG, tech-notes]
created: 2026-04-11
type: permanent
summary: Compared three retrieval strategies; hybrid retrieval works best in most scenarios
---
```

Field	Purpose	Why it matters to AI
title	Note title	Key signal for semantic search
tags	Topic categories	Batch-filter notes within a specific domain
created	Creation date	Assess information freshness (in AI, info from 3 months ago may already be outdated)
type	Note type	Different processing strategies (permanent = read carefully, fleeting = okay to skip)
summary	One-sentence abstract	Quick relevance check without reading the full text

The three values for the type field are borrowed from the Zettelkasten method:

- **fleeting** (fleeting notes): quick thoughts jotted down, material to be organized later. AI can skip these or help you tidy them up
- **literature** (literature notes): notes taken after reading a book or article. AI can help extract key points and build connections
- **permanent** (permanent notes): your own considered opinions and summaries, refined through thinking. AI prioritizes these when generating content

You don't need to add complete frontmatter to every note from day one. Build the habit with new notes first. Old notes can be batch-updated by Claude Code—Chapter 7 covers exactly how.

Principle 6: Separate Human Input from AI Output

This one is easy to overlook, but it's important.

Your original notes and AI-generated content (summaries, connection analyses, auto-generated indexes) should be stored separately. The simplest approach is to use folders: your notes go in notes/, AI-generated content goes in ai-output/ or gets an _ai- prefix in the filename.

Why separate them? Two reasons.

Different levels of trust. You know the source and reliability of your own notes. AI-generated content might hallucinate, might be outdated, might misunderstand your intent. Mix them together, and over time you can't tell which are your original records and which are AI artifacts.

Different processing strategies. When handling your original notes, AI is more cautious (this is source data—don't modify freely). When handling its own output, AI can be bolder (this is derived data—it can always be regenerated). Separating them lets Claude Code know which strategy to use.

> **核心建议**
>
> A practical approach: state clearly in CLAUDE.md which folders contain human output and which contain AI output. For example: "Files under notes/ are my original notes—don't modify content without confirmation. Files under ai-output/ are AI-generated and can be freely updated."

Three Architecture Templates

Principles covered. Now let's look at how to actually organize folders. Here are three templates, covering everything from minimal to complex. Don't overthink which one to pick—start with Template A, and upgrade to B or C when you outgrow it.

Template A: Minimalist

Best for Obsidian beginners, or vaults with fewer than 200 notes.

```
my-vault/
├── daily/ # Daily notes
├── notes/ # All notes (flat storage)
├── projects/ # Active projects
├── archive/ # Completed projects and outdated notes
└── CLAUDE.md # AI's operating manual
```

Four folders plus a CLAUDE.md. That's it.

daily/ holds daily notes (Obsidian has a core Daily Notes plugin that generates today's journal with one click). notes/ holds all your knowledge notes—no subfolders, use tags and links for categorization. projects/ holds files for active projects. archive/ holds finished stuff.

CLAUDE.md is the key. It tells Claude Code: what the vault structure is, what goes in each folder, and what your preferences are. Without it, Claude Code walks in like entering an unlabeled warehouse—it can only search blindly.

Template A's advantage is zero decision overhead. New note? Toss it into notes/. No need to think about which folder it belongs in. The downside is that notes/ gets large over time, but tags and search handle

that just fine.

Template B: PARA

Best for people managing multiple projects across multiple domains. PARA is a personal knowledge management framework by Tiago Forte that divides all information into four dimensions: Projects, Areas, Resources, and Archive.

```
my-vault/
├── 1-Projects/ # Projects with clear deadlines
├── 2-Areas/ # Ongoing areas of responsibility (health, finance, skills...)
├── 3-Resources/ # Reference material on topics of interest
├── 4-Archive/ # Completed or inactive content
├── daily/ # Daily notes
└── CLAUDE.md # AI's operating manual
```

The core idea of PARA is ranking by "actionability": Projects need the most action (they have deadlines), Areas need ongoing attention, Resources are backup references, and Archive is storage.

My own writing system is essentially a PARA variant, even though I didn't intentionally design it that way. Independently evolving into nearly the same structure suggests PARA maps to a very natural way of organizing information.

Template B's advantage is structural clarity—every note has a clear home. The downside is you need to "move" things regularly: when a project is done, it moves from 1-Projects to 4-Archive. But that's an operation Claude Code can automate.

Template C: Karpathy Wiki

Best for researchers, knowledge-intensive workers, or anyone who wants to run their vault like a personal wiki.

```
my-vault/
├── raw/ # Raw input (article excerpts, meeting notes, ideas)
├── wiki/ # Curated knowledge entries (one concept per file)
├── output/ # Content produced from the knowledge base (articles, reports, scripts)
├── SCHEMA.md # Format specification for knowledge entries
└── CLAUDE.md # AI's operating manual
```

This template is inspired by Andrej Karpathy. In an interview, he mentioned maintaining a personal wiki where each concept gets its own page, organized like Wikipedia.

Template C's core is the raw → wiki → output information flow. raw/ is the input layer—throw anything in, no organization needed. wiki/ is the knowledge layer—one file per concept, refined and structured. output/ is the output layer—content produced based on wiki knowledge.

SCHEMA.md defines the standard format for wiki entries. For example:

```
## Wiki Entry Specification

Every wiki entry must include:
- frontmatter: title, tags, created, summary
- Definition: explain the concept in one or two sentences
- Key points: 3-5 core takeaways
- Connections: [[links]] to related concepts
- Sources: where the original information came from
```

With SCHEMA.md in place, Claude Code can generate new wiki entries in a consistent format. You dump a paper's PDF excerpt into raw/, Claude Code reads it and generates a properly formatted knowledge entry in wiki/.

Template C's advantage is the highest knowledge reuse rate—the wiki layer is curated and can be directly referenced. The downside is higher maintenance cost, but that cost is mostly borne by AI. The next chapter covers in detail how Claude Code auto-generates wiki entries from raw materials.

How to choose? Fewer than 200 notes, or just starting out? Template A. Managing multiple projects in parallel? Template B. Research-oriented work, need a highly structured knowledge base? Template C. Not sure? Start with A, then decide after two weeks.

Frontmatter in Practice

A complete frontmatter block looks like this:

```
---
title: Note Title
tags: [AI, knowledge-management]
created: 2026-04-11
type: permanent
summary: One-sentence abstract
---

Body text starts here.
```

A few things to note:

Use English for tags. While Obsidian supports tags in any language, English tags perform more reliably with Claude Code's grep searches. Use #AI rather than localized equivalents, #reading-notes rather than translated alternatives.

Use ISO format for dates. 2026-04-11, not "April 11th" or "11/04/2026." ISO format (YYYY-MM-DD) is the only unambiguous date format—both AI and humans can parse it at a glance.

Stick to three values for type. fleeting, literature, permanent. Don't invent more. The finer the categories, the higher the maintenance cost, and the more likely you are to abandon the system. Three values are enough.

Keep summaries to one sentence. Between 30 and 80 characters. If you can't summarize a note in one sentence, the note itself probably needs splitting.

> **核心建议**
>
> Obsidian has a Templates feature (core plugin) that lets you preset frontmatter templates. One click inserts the template when creating a new note—no manual typing each time. Path: Settings → Core plugins → Templates → Set template folder.

Wikilinks vs. Tags vs. Folders

Obsidian gives you three tools for organizing notes: [[wikilinks]], #tags, and folders. Many people agonize over which to use. The answer is: use all three, but each for a different job.

Folders Handle Lifecycle

Folders answer the question: "What stage is this at?"

Active projects go in projects/. Done? Move to archive/. Daily logs go in daily/. Knowledge entries go in notes/ or wiki/.

Folders should not be used for topic-based categorization. An "AI" folder, an "Investing" folder, a "Health" folder—don't do this. A note might span AI and investing simultaneously (e.g., "Using AI for Quantitative Trading"), and it doesn't belong neatly in either folder.

#Tags Handle Topics

Tags answer the question: "What domain does this belong to?"

A note can have multiple tags. #AI #investing #quant—three tags coexisting, perfectly solving the "which folder" problem. When Claude Code searches for notes on a topic, grepping tags is far more efficient than traversing folders.

The key to a tag system is restraint. Don't exceed 20 top-level tags. Nested tags (#AI/RAG) are fine, but don't go deeper than two levels.

[[Wikilinks]] Handle Relationships

Wikilinks answer the question: "Which concepts are directly related?"

When you mention a concept in one note, and your vault happens to have a dedicated note on that concept, [[link]] them. These links are semantic—more precise than tags. A tag says "these two notes are both about AI." A wikilink says "this note references a specific point from that note."

Claude Code can traverse wikilinks to follow connected knowledge. You ask it to write an article about RAG; it sees your RAG note links to [[vector database]] and [[embedding]], so it automatically reads those two notes for fuller context.

All Three Working Together

Say you just read a paper on RAG optimization:

```
File location: notes/rag-optimization-2026.md    ← Folder handles lifecycle (notes = knowledg
Tags: #AI #RAG #paper                            ← Tags handle topics
Body links: [[vector database]] [[embedding]] [[chunking]]  ← Wikilinks handle relationships
```

Three tools, each doing its own job, no conflicts.

> **注意**
>
> The most common mistake is using only one tool. Folders only → a note can't belong to multiple topics. Tags only → no direct links between notes. Wikilinks only → no global categorization view. Use all three together to unlock Obsidian's full power.

What Your CLAUDE.md Should Contain

Putting all the above principles together, here's a CLAUDE.md template for a vault:

```
# Vault Operating Manual

## Structure
- daily/: Daily notes, named YYYY-MM-DD
- notes/: Knowledge notes, flat storage, categorized by tags and links
- projects/: Active projects, one subfolder per project
- archive/: Completed projects and outdated notes

## Frontmatter Specification
Every note must include: title, tags, created, type, summary
type values: fleeting / literature / permanent

## Glossary
- Use "RAG," not "Retrieval-Augmented Generation"
- Use "LLM," not "large language model"
- Use "vault," not "knowledge base"

## Operating Rules
- Files under notes/ are my original notes—don't modify content without confirmation
- You may modify frontmatter (fill in missing tags, summary, etc.)
- New notes must include complete frontmatter
- Move completed projects to archive/
```

Short, but dense with information. Start with a basic version, and whenever AI does something wrong, add a rule.

Good vault architecture lets AI understand the most information with the fewest tokens. With the design done, the next chapter is where the real work begins.

§06 Let AI Maintain Your Knowledge Base

Let AI Maintain Your Knowledge Base

Instead of building RAG to let AI retrieve your notes, let AI maintain your knowledge base directly. That's not me talking—that's Karpathy. And 16 million people saw it.

Compiler, Not Retriever

In April 2026, Andrej Karpathy posted a tweet.

It wasn't long. The gist: he was using an LLM to maintain a personal wiki. Not having AI search his notes and answer questions, but having AI read raw material and write structured wiki articles directly—articles that would be kept, iterated on, and cross-referenced.

The tweet got 16 million+ views.

Why did it blow up? Because it touched on something many people sensed but couldn't articulate: **RAG might be headed down the wrong path.**

RAG is currently the most mainstream way to "let AI use your data." Dump your notes into a vector database; when AI gets a question, it first searches for relevant snippets, stuffs them into the prompt, and generates a response.

Sounds reasonable. But use it and you'll see the problem.

Every time you ask AI a question, it starts from scratch. Search, match, stitch, generate. Ask a related question next time, and it runs the same pipeline all over again. AI has no memory, no accumulation. Today it helps you sort out "the difference between RAG and fine-tuning"; tomorrow you ask again, and it has to search and think through everything from the beginning.

Like an intern who wakes up with amnesia every morning. Does a decent job researching and writing reports, but the next day forgets everything from the day before and starts the same topic from scratch.

Karpathy's insight is simple: **let the intern save the reports.**

Let AI write its understanding into wiki articles. These articles stay in the knowledge base, can be referenced by subsequent questions, updated with new information, and cross-linked with other articles.

AI isn't retrieving—it's compiling.

A retriever does redundant work every time. A compiler does the work once and produces reusable output. Once your raw notes are compiled by AI into wiki articles, the next time you need that knowledge, you just read the wiki—no need to reprocess the raw material.

RAG approach: each question → search raw material → stitch together a response → response is discarded after use

LLM Wiki approach: AI reads raw material → writes wiki articles → articles are kept and iterated → subsequent questions answered from the wiki

RAG's output is use-and-discard. LLM Wiki's output grows richer over time. It's a positive feedback loop.

Three-Layer Architecture

Karpathy's approach, implemented in an Obsidian vault, is a three-layer structure.

Layer 1: raw/ (Source Material)
Everything you throw in as raw input. Articles, papers, meeting notes, reading notes, web clippings, chat screenshots, jotted-down ideas.

Rule: append only, never modify. This is the source of truth. No matter how AI processes it later, the raw material always stays intact. Like cooking—don't touch the groceries you brought home; the processing happens in the kitchen.

Layer 2: wiki/ (Structured Knowledge)
Structured articles written by AI after reading material from raw/. Divided into three subdirectories by type:
- concepts/: Concept pages. e.g., "RAG," "Vector Database," "Agent Memory"
- entities/: Entity pages. e.g., "Karpathy," "Obsidian," "Claude Code"
- topics/: Topic pages. e.g., "AI Knowledge Management Tools Comparison," "2026 LLM Trends"

Rule: primarily written and maintained by AI; humans can make corrections. This layer is where the system's core value lies.

Layer 3: output/ (Query Products)
Reports, analyses, and answers generated from the wiki. For example, you ask AI to "compare Obsidian and Notion's competitiveness in the AI era," and it generates an analytical report based on existing concept and entity pages in the wiki, stored in output/.

Good output can feed back into the wiki. If an analytical report produces new insights, AI can update them back into the corresponding wiki articles.

The complete vault structure looks like this:

```
vault/
├── raw/          # Layer 1: Source material (immutable)
│                 # Everything you throw in
│                 # Rule: append only, never modify
│
├── wiki/         # Layer 2: Structured knowledge (AI-maintained)
│   ├── INDEX.md  # Global index
│   ├── concepts/ # Concept pages
│   ├── entities/ # Entity pages
│   └── topics/   # Topic pages
│
├── output/       # Layer 3: Query products
│                 # Reports, analyses, answers
│
├── SCHEMA.md     # The wiki's "constitution"
└── CLAUDE.md     # AI's employee handbook
```

The data flow between layers is crystal clear: raw → wiki → output. Raw material is compiled into structured knowledge; structured knowledge is used to generate output. Information flows in only one direction, and each layer has a well-defined responsibility.

You may have noticed two special files: SCHEMA.md and CLAUDE.md. We covered CLAUDE.md in Chapter 4—it's the employee handbook for AI. SCHEMA.md is new, serving specifically the wiki layer.

SCHEMA.md: Turning a General AI into a Professional Wiki Curator

Claude Code is a general-purpose AI. Ask it to "help me organize my notes," and it will, but the approach might differ every time. Tags might be #ai-tools today and #AI-Tools tomorrow.

SCHEMA.md solves this. It defines the wiki's naming conventions, tag taxonomy, wikilink rules, and article templates. With it, a general AI becomes a wiki curator that strictly follows standards.

Here's a complete SCHEMA.md example:

````
# Wiki Schema

## Naming Conventions
- Concept pages: `concepts/concept-name.md` (lowercase English, hyphen-separated)
  - Example: `concepts/retrieval-augmented-generation.md`
- Entity pages: `entities/entity-name.md` (full English name for people)
  - Example: `entities/andrej-karpathy.md`
- Topic pages: `topics/topic-description.md`
  - Example: `topics/ai-knowledge-management-tools.md`

## Frontmatter Template
Every wiki article must include:
```yaml
---
title: Article Title
type: concept | entity | topic
tags: [tag1, tag2]
sources: [source file paths in raw/]
created: YYYY-MM-DD
updated: YYYY-MM-DD
summary: One-sentence abstract
---
```

## Tag Taxonomy
- Domain tags: #ai, #programming, #product, #business
- Status tags: #stub (needs expansion), #mature (complete)
- Do not create new top-level tags; update this file first if needed

## Wikilink Rules
- When first mentioning a concept that has an existing wiki page, use [[link]]
- Within the same article, only link the same concept the first time
- If a mentioned concept has no wiki page, create a stub

## Article Structure
Concept page: Definition → Key Points → Relationships to Other Concepts → Sources
Entity page: Introduction → Key Contributions → Related Concepts/Entities → Sources
Topic page: Overview → Multi-angle Analysis → Conclusion → Sources

## Global Index
After every new or modified wiki article, update wiki/INDEX.md
INDEX.md lists all wiki pages by category with a one-sentence summary
````

Under 50 lines, but it unifies four things: naming, format, tags, and linking. AI no longer wrestles with "should this be called RAG.md or retrieval-augmented-generation.md," and you won't get #ai and #AI coexisting in chaos.

Customize it to your needs. Doing investment research? Add a status field (bullish/bearish/neutral). In academia? Add a citation format. SCHEMA.md's core value isn't about what specifically you write—it's

about having a clear specification that AI strictly follows.

Hands-On: Building a Knowledge Wiki from Scratch

Enough concepts. Let's build one.

Step 1: Prepare your raw/ material.

Find 5-10 articles, notes, or documents you've recently read, and drop them into your vault's raw/ directory. No cleanup needed, no renaming—just dump them in as-is. These are your source materials.

For example, if you've been following the AI knowledge management topic, you might have:

- Karpathy's LLM Wiki tweet screenshots and discussion threads
- A technical blog post about how RAG works
- A comparison article between Obsidian and Notion
- Your own earlier notes on vector databases
- Notes from a podcast about knowledge management tools

Step 2: Write SCHEMA.md.

Copy the template above into your vault root and tweak it for your needs. Don't change too much at first—just run with the defaults and iterate later.

Step 3: Have Claude Code read the material and write wiki articles.

In the terminal, cd to your vault directory, launch Claude Code, and tell it:

```
Read all files in raw/, then create wiki articles for the key concepts,
people, and topics found in them, following the SCHEMA.md specification.
Every article should include [[cross-references]] to other articles.
When done, update wiki/INDEX.md.
```

Then watch it work.

Claude Code will read through your material, extract key concepts (like RAG, vector database, LLM Wiki), key people (like Karpathy), and key topics (like AI knowledge management tool comparison). Then, following SCHEMA.md's format, it creates wiki articles one by one.

While writing the RAG concept page, it automatically links to [[vector database]]. While writing Karpathy's entity page, it links to [[LLM Wiki]] and [[RAG]]. While writing topic pages, it references multiple concept and entity pages. The knowledge network grows itself.

Step 4: Open Obsidian's Graph View.

Click on Graph View, and you'll see a small knowledge network appear in what was an empty vault moments ago. Nodes are wiki articles; edges are [[wikilinks]]. Conceptually related articles cluster

together.

The network is small now. But as you keep dropping new material into raw/ and having AI compile them into wiki articles, the network grows denser and the connections between knowledge points grow richer.

Step 5: Ask AI a question.

Now test it. Ask Claude Code:

```
RAG vs. LLM Wiki—what scenarios is each best suited for?
```

It doesn't need to search raw material again. The wiki already has a concept page for RAG and one for LLM Wiki, and they're cross-referenced. AI reads through this pre-compiled knowledge and gives you a high-quality comparative analysis on the spot.

That's the power of compiling: invest time once to organize the knowledge well, and every subsequent use is instant.

Scale and Boundaries

How big can this approach handle?

Based on my own testing and community feedback, **for knowledge bases under 400,000 characters (~200K words), you don't need a vector database at all.** Claude Code reading Markdown files directly is sufficient. A few dozen structured articles in wiki/, plus INDEX.md for navigation, and AI finds relevant knowledge quickly and accurately.

Why is 400K characters enough? Because Claude's context window is large enough. The standard 200K-token window holds roughly 500K-800K characters of Markdown. If you use the 1M-token extended window, that's 2-3 million characters. For the vast majority of personal knowledge bases, this capacity is more than sufficient.

AI doesn't need to read all your wiki articles at once. It reads INDEX.md first to know what's available, then selectively reads the relevant articles based on the current question. Same as how you use an encyclopedia—check the table of contents first, then flip to the right page.

But beyond the million-character scale, plain-text search efficiency drops. That's when you might want to add a semantic search layer—the Smart Connections plugin recommended in Chapter 8 does exactly this. It uses AI embeddings to build a semantic index for every note in your vault, matching not just keywords but semantic similarity.

Knowledge base scale	Recommended approach	Vector database needed?
Under 100K characters	wiki/ + INDEX.md + Claude Code direct reading	No
100K-400K characters	Same as above, wiki directory split by topic	No
400K-1M characters	Add Smart Connections semantic search assist	Optional (built into plugin)
Over 1M characters	Must pair with RAG for semantic retrieval	Yes

This approach depends on large context windows. In 2026, 200K tokens is standard for mainstream models, and 1M isn't rare. As windows continue to grow, the scale that a plain-text wiki approach can support will grow with it. Karpathy's claim that "you don't need a vector database" already holds true in most scenarios.

Huashu's _knowledge_base/ Was Already a Wiki Prototype

When I first saw Karpathy's tweet, my reaction was: wait—isn't this exactly what I've been doing?

My _knowledge_base/ directory has over fifty files, organized by topic, with an INDEX.md. Structurally, it's almost identical to wiki/.

Dimension	Huashu's _knowledge_base/	Karpathy's wiki/
Index	INDEX.md (table format)	INDEX.md (table format)
Categorization	By topic folders (tech/people/products)	By type folders (concepts/entities/topics)
Format	Markdown, with summaries	Markdown, with frontmatter
Cross-references	Few manual links	Systematic [[wikilinks]]
Maintenance	Manual curation + AI assistance	AI handles everything
Specification	Implicit in CLAUDE.md	Dedicated SCHEMA.md

The core philosophy is the same. The differences lie in two places: density of cross-references, and degree of automation.

Files in my knowledge base rarely link to each other; in Karpathy's approach, every wiki article is tightly connected through [[wikilinks]]. AI can traverse links to follow related knowledge instead of looking at isolated files.

My knowledge base relies mostly on manual curation; Karpathy's approach fully automates the process: SCHEMA.md defines the specification, AI creates articles, updates the index, and builds cross-references

according to spec—no human involvement required.

From a manually maintained knowledge warehouse to a self-growing knowledge network. Not much extra work, but the result is a qualitative leap.

> **For those with an existing knowledge base:** You don't need to start from zero. Treat your existing notes directory as raw/, and have Claude Code compile them into wiki articles following SCHEMA.md. Everything you've already built is still valuable—it's just reorganized in a more efficient way.

AI isn't your search engine—it's your knowledge compiler.

Next chapter: 7 real-world workflows, all ready to copy and use.

§07 7 Real Workflows

7 Real Workflows

Enough theory. This chapter is all hands-on stuff you can copy directly. For each workflow, I'll give you concrete steps and prompts. Read through them and you can start running these in your own vault right away.

7.1 Daily Notes + AI Weekly Review

This is the lowest-barrier workflow. If you're only going to try one thing, start here.

Obsidian has a core plugin called Daily Notes. Turn it on, and every day you hit a shortcut (Cmd+D or click the calendar icon in the sidebar), Obsidian automatically creates a file named after the date, like 2026-04-11.md. You don't need to create the file yourself, think of a filename, or decide where to store it. One shortcut, and you're writing.

A few lines a day is plenty. Don't try to write long journal entries; you won't keep it up for two weeks. Three to five lines will do: what you did today, what you thought about, anything interesting you came across. Any format works, no structure needed. You're writing for your future self and for AI, not for anyone else.

Something like this:

```
## 2026-04-11

- Spent the morning writing Chapter 7 of the orange book, the Obsidian workflows chapter
- Saw an article saying Cursor is building an Obsidian plugin, saved it to raw/
- Afternoon meeting with the publisher about print layout, decided to drop the print book ide
- Tried Gemini 3 Pro's image generation in the evening, quality way better than last month
- Thought: AI tools are updating so fast that "writing a book" as a format can barely keep up
```

That's it. Five minutes a day.

After a week of this, things get interesting. You ask Claude Code to read this week's daily notes and generate a weekly summary.

1. **Set up the Daily Notes plugin**
   Settings → Core plugins → Enable Daily notes → Set your template and storage path (I recommend a daily/ folder)

2. **Write a few lines every day**
   Cmd+D to open today's note, jot things down as they happen. No format required, no need for complete sentences. The key is keeping the psychological barrier to recording as low as possible.

3. **Let Claude Code generate a weekly review on the weekend**
   Open your terminal in the vault directory and have Claude Code read through this week's notes and produce a summary. Here's a prompt you can use:

```
Read all daily notes from this week (April 7 to April 11) in the daily/ directory.
Generate a weekly summary that includes:
1. What I mainly worked on this week (grouped by project)
2. Ideas and insights worth remembering
3. Things to follow up on next week
Save it to weekly/2026-W15.md
```

Claude Code scans your notes and organizes the scattered fragments into a structured weekly review.

The year-end Wrapped is even more fun. At the end of the year, have Claude Code read all your weekly reviews (or go straight through 365 days of daily notes) and generate an annual review: how many projects you worked on, how many books you read, what topics you thought about most frequently, which ideas kept coming back.

Spotify Wrapped shows you what songs you listened to all year. Obsidian + Claude Code shows you what you thought about all year.

> **核心建议**
> The greatest value of daily notes isn't in reviewing them, but in accumulation. A thought you jot down today might get surfaced by Claude Code three months later when you're writing an article, becoming a brilliant piece of material. But the prerequisite is that you actually write it down first.

7.2 Reading Notes + AI Distillation

What's the biggest waste when reading a book? Finishing it and retaining nothing.

You highlighted passages, wrote annotations, thought "this part is brilliant" at the time. Three months later you want to reference a certain idea from the book, only to find you can only remember "something about something something," with zero recollection of the specifics.

The fix is simple: dump your highlights and annotations into the raw/ folder in your vault and let Claude Code generate structured literature notes for you.

1. **Highlight freely while reading**
   Read normally on Kindle, WeChat Read, or a physical book. Highlight and annotate without worrying about format.

2. **Save your raw annotations into the vault after finishing**
   Copy all your highlights and annotations into a single file and save it as raw/antifragile-raw-annotations.md. No need to organize; just paste them as-is.

3. **Let Claude Code generate literature notes**
   Here's a prompt you can use:

```
Read raw/antifragile-raw-annotations.md — these are my highlights and annotations from readir
Generate a structured reading note that includes:
1. The book's core arguments (3-5)
2. Key evidence and examples for each argument
3. My reactions and thoughts from the annotations (keep the original flavor, don't rewrite th
4. Which existing notes in the vault relate to this book (search the vault)
Save to books/antifragile.md
Also, use [[bidirectional links]] to connect to related concepts that already exist in the va
```

That last step is the key. Claude Code isn't just organizing notes for one book; it searches your vault, finds existing related notes, and builds connections. Say you previously wrote a note about [[Taleb]], or mentioned [[Black Swan]] in a journal entry — Claude Code will automatically establish those links.

Each book is no longer an island. It gets woven into your entire knowledge network.

> **核心建议**
> Advanced option: If you use Kindle, you can install the Readwise plugin (a third-party service) that automatically syncs Kindle highlights to your Obsidian vault. Once synced, let Claude Code process them, and you can skip the "copy and paste highlights" step entirely.

7.3 Research + Knowledge Accumulation

This is the workflow I use the most. Before writing every article for my public account, I do research: reading official docs, tech media coverage, discussions on X, trying out products. These research findings used to be scattered across browser tabs and temp files, gone once the article was done.

Now I store all research findings in my vault.

1. **Save raw information to raw/ while researching**
   When you find useful articles, product docs, or user reviews, save the key content to the raw/ folder. No need to organize — preserve raw information first, format doesn't matter.

2 **Let Claude Code compile it into a wiki article**
Once you've done enough research, have Claude Code synthesize the information in raw/ into a structured wiki article:

```
Read all research files about Obsidian in the raw/ directory.
Synthesize them into a wiki article and save to knowledge/Obsidian.md.
Requirements:
1. Organize by topic (what it is, core features, competitor comparison, community ecosystem,
2. Cite the source for each piece of information
3. Use [[bidirectional links]] to connect to existing related notes in the vault
4. List questions discovered during research that are worth exploring further
```

3 **Next time you research the same topic, AI already has context**
Three months later you need to write another Obsidian-related article. Claude Code reads knowledge/Obsidian.md and instantly knows what you've already researched, allowing it to do incremental updates on top of existing knowledge instead of starting from scratch.

Each research session is no longer a one-time consumable — it's an incremental update to your knowledge base. The more you research, the thicker the context in your vault grows. And the higher AI's starting point becomes.

By the time I was writing my seventh Claude Code article, I could clearly feel the difference. Claude Code already knew what I'd said before, what I'd tested, what my opinions were. The first draft no longer needed to explain everything from zero.

The more you write, the thicker it gets. The thicker it gets, the easier writing becomes. A positive feedback loop.

7.4 Writing Workflow (From Notes to Articles)

The first three workflows (daily notes, reading notes, research) are all about "putting things into" the vault. This workflow is about "using" them.

The traditional way to write an article is to open a blank document and start from scratch. Ideas are vague, and a first draft feels impossibly far away.

With a vault, you never face a blank page.

1 **Tell Claude Code what you want to write**
No outline needed, no structure needed. Just describe your topic in one sentence. For example: "I want to write an article about why AI tools update so fast that writing a book has become really difficult."

2 **Claude Code searches your vault first**
It searches through your daily notes, reading notes, research files, and previous articles for everything related to this topic. Then it gives you an "existing materials list," telling you what's already available in your vault.

3 **Generate a first draft based on existing knowledge**
Claude Code doesn't fabricate an article from thin air — it organizes a first draft based on your existing notes, ideas, and materials. An insight you once wrote in a journal entry, a reaction you expressed in a book annotation, a data point you discovered during research — all of these can be woven into the draft.

The first draft is built on top of everything you've ever thought and written.

An AI-generated first draft isn't a final draft — you still need to edit. But you're editing a draft that already contains your own ideas and materials, not a blank page.

不推荐

Writing without a vault:
Blank page → Try to recall materials from memory → Search the internet again → Piece together a draft → Endless revisions

推荐

Writing with a vault:
One-sentence topic → AI searches vault for existing materials → First draft generated from your own knowledge → Focused editing

The more notes in your vault, the more powerful this workflow becomes. That's why the first three workflows (daily notes, reading notes, research) matter so much: they're the ammunition depot for this one.

7.5 Project Management

Obsidian isn't a professional project management tool — don't expect it to replace Jira or Linear. But for personal projects and small teams, Obsidian's project management capabilities are more than enough, and it has one unique advantage: project knowledge and project management live in the same place.

The method is simple: one folder per project, with an index.md inside each folder.

```
projects/
├── orange-book-obsidian/
│   ├── index.md          # Project status, to-dos, key decisions
│   ├── research/         # Research notes
│   ├── drafts/           # Chapter drafts
│   └── materials/        # Reference materials
├── blog-claude-source-analysis/
│   ├── index.md
│   ├── research.md
│   └── draft.md
└── archive/               # Completed projects
```

What goes in index.md? Project status, to-do items, key decision records. That's it.

Obsidian's advantage here is that knowledge across projects can reference each other through [[bidirectional links]].

For example, while writing the orange book you researched Claude Code's CLAUDE.md mechanism — and that knowledge is also relevant to your "blog article" project. You type [[CLAUDE.md]] in your research note, and that concept now appears in the context of both projects. You don't need to maintain a cross-project knowledge index; the links themselves are the index.

When a project is done, move the entire folder to archive/. Don't delete, just archive. Notes in archive/ can still be searched and linked to. A completed project's knowledge assets don't disappear — they simply shift from "active" to "consultable."

> **核心建议**
> If you want a kanban view (like Trello), install the Kanban plugin. It turns Markdown files into draggable kanban cards, and all data remains in Markdown, so Claude Code can read and write it directly.

7.6 Auto-Organizing Old Notes

You have a pile of messy old notes. You know there's good stuff in there, but every time you open them you want to close them right back.

Someone on MakeUseOf shared their experience: 5 years of notes, no tags, no links. They tried organizing manually three times and gave up every time. Finally pointed the whole folder at Claude Code — 90 minutes and it was done.

1. **Put old notes into a folder in your vault**
   Something like inbox/ or unsorted/. Whatever format they're in, just throw them in. If they're in txt or doc format, batch-convert them to Markdown first (Claude Code can help with that too).

2 Let Claude Code scan and organize

Give Claude Code a prompt like this:

```
Scan all Markdown files in the unsorted/ directory.
For each file, do the following:
1. Add frontmatter (title, date, tags)
2. Based on the content's topic, move it to the appropriate folder (books/, ideas/, projects/
3. Search other notes in the vault and add relevant [[bidirectional links]]
4. Generate an organization report and save to unsorted/organization-report.md

Notes:
- Don't delete any content, only add structure
- If unsure about classification, keep it in inbox/ and flag it in the report
- Auto-generate frontmatter tags based on content, no more than 5 per file
```

3 Review the organization report

Claude Code generates a report listing what it did: how many files it moved, how many links it added, which classifications it was uncertain about. Give it a quick scan and adjust anything that's off.

After organizing, your old notes go from "information graveyard" to structured, linked, searchable knowledge assets. More importantly, they're now part of the same network as your new notes. A journal entry you write today could end up linked to a three-year-old note through bidirectional links.

> **Practical advice:** Don't dump too many files on Claude Code at once. If you have thousands of old files, process them in batches (100-200 at a time), review quality after each batch, then move on to the next. This way you can correct Claude Code's classification logic along the way, and later batches will get progressively more accurate.

7.7 Automatic Backlinks (Agentic Note-Taking)

All the workflows above require you to trigger them manually. This one can run semi-automatically.

Bidirectional links are great, but you have to remember to create them. You write in your journal "had lunch with Lao Wang today" — if you don't manually type [[Lao Wang]], that entry has no connection to your previous notes about Lao Wang.

German developer Stefan Imhoff used to spend 10 to 15 minutes daily adding [[wikilinks]] by hand. Until he tried Claude Code.

His workflow looks like this:

1. **Write your daily notes normally**
   No links needed — just write as if you're using a plain notepad. Mention names, book titles, and concepts in plain text.

2. **Run Claude Code after you're done writing**
   Have Claude Code scan today's journal entry and automatically identify all linkable entities. A prompt like this:

```
Read today's journal entry at daily/2026-04-11.md.
Find all person names, place names, book titles, and concept nouns.
For each entity:
1. Search the vault for an existing note that matches
2. If found, replace the plain text in the journal with a [[wikilink]]
3. If not found, create a new entity note (with basic information), then link it
Save modifications directly to the original file.
```

3. **Your journal automatically becomes a linked network**
   "Had lunch with Lao Wang today, chatted about Taleb's new book" automatically becomes "Had lunch with [[Lao Wang]] today, chatted about [[Taleb]]'s new book." If there's no note for "Lao Wang" in the vault, Claude Code will create one for you.

Doing this manually takes 10 to 15 minutes. Claude Code does it in seconds.

And Claude Code is more thorough. You might miss a name; it won't. You might forget there's a note about some concept in your vault; it will find it.

This is what Stefan Imhoff calls "Agentic Note-Taking." The organizing and linking work is handed off to an AI Agent, and you just write. Every journal entry, every note you make, gets automatically woven into your knowledge network.

不推荐

Manual linking:
Finish writing → Go back and check for linkable entities → Search the vault one by one → Add [[links]] one by one → 10-15 minutes daily

推荐

AI auto-linking:
Finish writing → One command → Claude Code auto-identifies, searches, and links → Done in seconds

核心建议

Advanced move: Turn this auto-linking into an Obsidian template command or an automated rule in CLAUDE.md. Every time you save a journal entry, Claude Code automatically triggers the linking process. From "manually triggered" to "linked on save."

Summary Table

Seven workflows, from simple to complex, summarized below:

Workflow	What you do	What AI does	Daily time cost
Daily Notes + AI Weekly Review	Write a few lines each day	Generate weekly summary	5 min
Reading Notes	Highlight and annotate while reading	Generate structured notes + build links	Reading time (no extra)
Research Accumulation	Save raw info to raw/	Compile into wiki + incremental updates	Research time (no extra)
Writing Workflow	State your topic in one sentence	Search vault + generate first draft	0 (just trigger it)
Project Management	Create folder + write index.md	Cross-project knowledge linking	As needed
Organize Old Notes	Dump old notes into vault	Classify, tag, build links	One-time (review report)
Auto Backlinks	Write notes normally	Auto-identify entities, build links	0 (runs automatically)

Look at that rightmost column. **These workflows don't add extra burden.** You do the same things you've always done. AI works behind the scenes to turn information into structured, connected knowledge.

You don't need to adopt all seven. Start with daily notes, keep it up for two weeks, and you'll naturally want to try the rest.

§08 Plugins and Tools Ecosystem

Plugins and Tools

Obsidian's plugin ecosystem has over 1,000 options, but you probably only need 4. This chapter helps you skip the trial and error.

8.1 Must-Have: obsidian-skills

I'm only recommending one "must-have" because it truly is the only one you'd regret not installing.

obsidian-skills was built by Obsidian CEO kepano himself. It's not a traditional plugin — it's a set of Claude Code Skills that teach the AI how to correctly understand Obsidian's proprietary syntax.

Obsidian's Markdown has its own extended syntax: [[wikilinks]] for bidirectional links, callout blocks, frontmatter properties, .canvas spatial notes, and .base database files. These aren't standard Markdown, and Claude Code doesn't recognize them by default — it might mess up the formatting when working with them.

What obsidian-skills does is tell Claude Code through CLAUDE.md: don't treat [[]] as standard Markdown links, don't break apart callout structures, and preserve YAML formatting in frontmatter.

Installation takes just 30 seconds:

1. **Open your terminal and navigate to the vault**

   ```
   cd /path/to/your/vault
   ```

2. **Install Skills**

   ```
   claude install kepano/obsidian-skills
   ```

After installation, you don't need to configure anything. Claude Code will automatically read these Skills when working with your vault and know how to handle Obsidian files correctly.

8.2 Three Strongly Recommended Ones

The following three aren't mandatory, but each one noticeably improves the Obsidian + AI experience. Pick based on your usage habits.

Claudian: Use Claude Code Directly Inside Obsidian

You're reading notes in Obsidian and want Claude Code to handle something — so you switch to the terminal, run a command, check the result, switch back. Constant window-switching.

Claudian embeds Claude Code's chat interface into Obsidian's sidebar. Type instructions on the right, watch your notes change in real time in the main editor. Reading files, writing files, searching, analyzing images — exactly the same as in the terminal. Just one less Alt+Tab.

> **核心建议**
> Claudian requires Claude Code to be installed locally. It's not a standalone AI — it's a bridge between Obsidian and Claude Code.

Smart Connections: Semantic Search

Obsidian's built-in search is keyword matching. Search for "investing" and you'll only find notes that contain the exact word "investing." "Asset allocation" or "long-term returns"? Not found.

Smart Connections uses AI embeddings to build a semantic index of your vault. Search for "investing" and all notes with related concepts surface.

Even more useful: open any note and the sidebar automatically shows the most semantically similar notes. You'll often discover connections you weren't aware of.

It supports running embeddings with local models, so your notes don't need to be uploaded to the cloud.

Copilot for Obsidian: Knowledge Base Q&A

You want to know what you've previously written about pricing strategy. Or what thinking patterns Munger and Feynman have in common.

Copilot builds a RAG system at the vault level. Ask a question directly, and it searches relevant notes in your vault, generates an answer based on your note content, and cites the sources.

The difference from asking ChatGPT: the answers come from your own notes, not generic internet information. You're querying your own knowledge base.

推荐

Copilot for Obsidian:
Answers based on your vault content → Every point has a source in your notes → A reorganization of your own knowledge

不推荐

Asking ChatGPT directly:
Answers based on generic internet knowledge → No personal context → May not match your actual situation

8.3 Install As Needed

The following plugins address more specific scenarios — not everyone will need them.

Plugin	What It Does	Who It's For
Smart Composer	A Cursor-like AI writing assistant with inline completions and rewrites in the editor	People who write a lot every day
Text Generator	Template-based text generation — define templates and batch-produce content	People who need to generate similar content in bulk
Agent Client	A unified interface to connect Claude Code, Codex, Gemini CLI, and other AI tools	People who use multiple AI tools simultaneously
Templater	An advanced template engine with JavaScript scripting and dynamic content support	Heavy template users
Dataview	Query your vault like a database using SQL-like syntax to filter and display notes	People who want to turn their vault into a structured knowledge system

Don't install too many at once. Use the basics for a week or two, and when something hurts, install the plugin that solves it. Too many plugins shift your attention from notes to configuration.

8.4 MCP: Most People Don't Need It

MCP (Model Context Protocol) is a protocol from Anthropic that enables two-way communication between AI tools and applications. You see this term a lot in the AI community.

But in the Obsidian context, most people don't need it. Claude Code can already read and write local files directly — no need for MCP as a middle layer.

When is MCP useful? Two scenarios:

- You want Claude Desktop (not Claude Code) to also operate on your vault. Claude Desktop can't access the local filesystem directly, so it needs MCP as a bridge.
- You need to invoke advanced features of Obsidian plugins. For example, having the AI execute a Dataview query and then analyze the results. Dataview's query engine runs inside Obsidian — Claude Code can't replicate that by reading files alone, so you need MCP to expose Dataview's capabilities to the AI.

Consider MCP when you hit a need that "just reading files" can't solve. Until then, ignore it.

8.5 Skills Marketplace

Claude Code has its own Skills ecosystem, complementary to Obsidian plugins. The community already has 700,000+ skills, and the ones related to knowledge management are worth checking out:

- **obsidian-second-brain**: A Skills collection designed specifically for the "second brain" use case, including automated workflows for note organization, connection discovery, and periodic review
- **claude-obsidian**: Community-maintained Obsidian operation Skills that cover more scenarios than kepano's official Skills

Installation is the same as obsidian-skills:

```
claude install [skill-name]
```

Plugins extend Obsidian's UI and functionality. Skills teach Claude Code new operational capabilities. One modifies the tool, the other modifies the AI.

> **Selection principle:** Install obsidian-skills first, then pick 1-2 plugins based on your most pressing need, and finally explore the Skills Marketplace as needed. Don't be greedy — every layer you add is cognitive overhead.

§09 Advanced Patterns

Advanced Patterns

The foundation is set. Now it's time to make the system your own. Version control, custom Skills, local AI, multi-vault strategies, and real-world lessons from migrating a patchwork setup to Obsidian.

9.1 Managing Your Vault with Git

An Obsidian vault is just a folder. Git is the most mature tool for tracking changes in folders. Managing your vault with Git gives you three things.

Complete change history. Every time Claude Code modifies a note, it leaves a record. What changed, what it looked like before — all traceable. Worst case, just git revert.

Transparency for AI operations. After having AI reorganize a batch of notes, `git diff` shows you exactly what it touched. Way better than opening dozens of files to check one by one.

Confidence to let go. "Reorganize the folder structure for these 50 notes" — without Git, you'd hesitate. With Git, you know you can always roll back.

Initialization is dead simple:

1. **Navigate to your vault directory**

   `cd /path/to/your/vault`

2. **Create .gitignore**

   Add `.obsidian/workspace.json` and `.obsidian/workspace-mobile.json`. These files track window layout and change every time you open Obsidian — no need to track them.

3. **Initialize the Git repository**

   `git init && git add -A && git commit -m "Initialize vault"`

Day-to-day usage after that:

```
# See what the AI changed
git diff

# Happy with it? Commit
git add -A && git commit -m "Claude organized reading notes"

# Not happy? Revert
git checkout -- .
```

> **核心建议**
> If you don't want to run Git manually, Obsidian has an "Obsidian Git" plugin that can auto-commit on a schedule. For example, automatically commit every 30 minutes.

9.2 Custom Skills

obsidian-skills teaches Claude Code how to handle Obsidian files. Custom Skills go further: they teach it to execute your personal workflows.

A Skill is just a natural language instruction in your CLAUDE.md. Whatever rules you write, it follows.

A few practical examples:

Auto-Generate Backlinks for Journal Entries

Your journal mentions a book, a person, a concept. Manually adding [[links]] is tedious. Write a rule in CLAUDE.md:

> When I ask you to "process today's journal," read the latest daily note, identify all mentions of people, books, and concepts. Check if corresponding notes already exist in the vault — if they do, replace the mentions with [[wikilinks]]. If they don't, create empty placeholder notes in the wiki/ directory, then link to them.

This single rule automates Stefan Imhoff's entire workflow from Chapter 1.

Weekly Compilation from raw/ to wiki/

If you set up the dual-zone structure from earlier chapters with raw/ (source material) and wiki/ (refined knowledge), you can add a rule for Claude Code to periodically distill raw/ notes into wiki/ entries:

> When I ask you to "compile this week's notes," read all files in the raw/ directory modified in the last 7 days. For each file, extract the core concepts and check if related entries already exist in wiki/. If they do, append the new information. If they don't, create new entries. When done, give me a summary of changes.

Auto-Create Wiki Pages for New Concepts

Add a general rule in CLAUDE.md:

> While working with the vault, if you encounter an important concept in a note that doesn't have a corresponding wiki page, proactively create a page with a basic definition in the wiki/ directory and add a [[link]] in the original note.

This way, Claude Code continuously enriches your wiki during routine operations, without you having to trigger it manually.

Quick Guide to Skill Development

Writing custom Skills doesn't require programming. They're just natural language instructions in CLAUDE.md. A few key points:

- **Make trigger conditions explicit.** "When I say XX" is better than "when you think it's appropriate." The former is predictable; the latter depends on the AI's mood.
- **Scope the operations.** "Read files in the raw/ directory" is better than "read all files." Limiting scope prevents the AI from accidentally modifying notes you don't want touched.
- **Specify the output format.** "Created wiki pages should have an H1 title + one paragraph definition + related links" is better than "create a page." The AI needs a template to stay consistent.
- **Test on a small scale first.** After writing a Skill, try it on a few files. Once you're happy, expand it to the entire vault.

9.3 Local AI Options

All the AI in this book runs in the cloud. Your data gets sent to servers for processing. Most people don't mind, but when your vault contains company secrets, when you're on a plane with no internet, or when you simply don't want data leaving your machine — you need a local solution.

Ollama lets you run open-source large language models (Llama 3, Mistral, Qwen) locally. Download a model, run it on your computer, and nothing gets sent to any external server.

Ways to combine Obsidian with local AI:

- **Smart Connections + Ollama**: Use a local model for embeddings and semantic search, completely offline
- **Copilot for Obsidian + Ollama**: Use a local model for vault-level Q&A, data never leaves your machine
- **Text Generator + Ollama**: Use a local model for text generation and rewriting

推荐

Advantages of local AI:
Complete data privacy / Works without internet / No API costs / No rate limits

不推荐

Trade-offs of local AI:
Model capabilities lag behind cloud (GPT-4/Claude-level models can't run locally) / Requires decent hardware (16GB+ RAM recommended) / Initial model download requires internet

Practical advice: use cloud AI for daily work (stronger, faster), switch to local AI for sensitive content. It's not either/or — the two can coexist.

9.4 Multi-Vault Strategy

All notes in one vault, or split across multiple?

A single vault is simple: all knowledge in one place, search and links stay intact. But when content volume grows, or when you need to isolate different types of content, multiple vaults start making sense.

Two common approaches:

Work + Personal. Work vault for project docs, meeting notes, client materials. Personal vault for reading notes, journals, life records. The advantage is clean boundaries — when you change jobs, the work vault stays (or goes with you), while your personal vault stays with you forever. It also avoids the awkwardness of company secrets and personal diaries living side by side.

Public + Private. If you blog or maintain a Digital Garden, you can put notes intended for publication in one vault and private notes in another. The public vault can be deployed directly to a website; the private vault gets a stricter backup strategy.

Things to keep in mind with multiple vaults:

- [[Bidirectional links]] don't work across vaults. Notes in different vaults can't reference each other.
- Each vault has its own .obsidian configuration — plugins and themes need to be set up separately.
- Claude Code can only operate on one vault at a time (one working directory). Switching vaults requires cd-ing to another directory.
- Each vault can have its own CLAUDE.md, defining different AI behavior rules. The AI in your work vault might be more formal; the AI in your personal vault might be more casual.

> **核心建议**
>
> If you're just getting started with Obsidian, stick with a single vault. Wait until your notes accumulate to the point where you feel "these two types of content really shouldn't be in the same place" before splitting. Splitting too early adds management overhead.

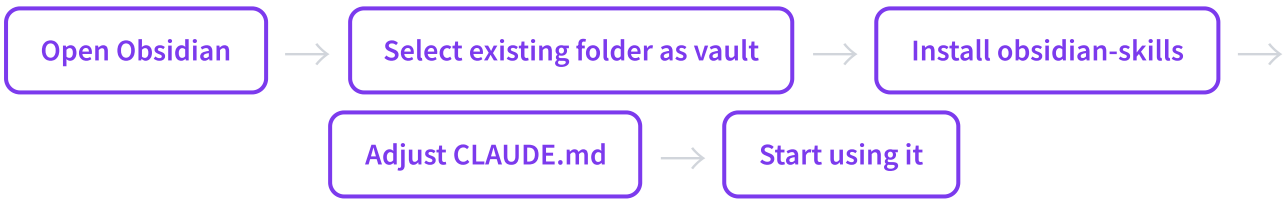
9.5 Migrating from a "Patchwork Setup" to Obsidian

This section is about my own migration experience. What pitfalls I hit, what I kept, what I changed.

Before migrating, I used a folder, Cursor's file tree, and Claude Code for all reading, writing, and searching. Each subdirectory had its own CLAUDE.md defining rules, with a root-level CLAUDE.md as the router. 55 custom Skills covered everything from research to publishing.

It worked. Used it for over half a year and shipped quite a bit. But at its core, it was a "patchwork setup" — using the filesystem plus CLAUDE.md to manually simulate knowledge management, reimplementing every feature that a proper tool should provide.

Migration Steps (Minimal Version)



Yes, that's really it. Because your files are already Markdown, Obsidian opens them without any conversion. All your existing files, directory structures, and CLAUDE.md files stay exactly as they are. Obsidian just adds a UI layer on top.

Pitfalls I Hit

The CLAUDE.md router needs adjusting. My original CLAUDE.md had extensive routing rules like "when you receive XX task, first read the CLAUDE.md in XX directory." These still work in Obsidian (Claude Code reads CLAUDE.md the same way), but some routing logic can be replaced by Obsidian's native features. For example, "search the knowledge base" no longer needs hand-written grep rules — Smart Connections' semantic search does it better. Redundant routing rules should be cleaned up gradually.

_knowledge_base/INDEX.md can be retired. I used to manually maintain a knowledge base index file listing all entries and their locations. In Obsidian, Cmd+O quick search and the graph view are a hundred times more useful than a manual index. Keep INDEX.md as a reference if you want, but it no longer needs maintenance.

The .obsidian/ directory appears in the file tree. Obsidian creates a .obsidian/ directory at the vault root to store configuration. If you manage your vault with Git, remember to add `.obsidian/workspace*.json`

to .gitignore (layout files change frequently). Other config files (plugins, themes) can be tracked in Git for syncing across machines.

What's Worth Keeping

The entire CLAUDE.md system carries over. CLAUDE.md isn't a Cursor concept — it's a Claude Code concept. Whether you use Cursor, Obsidian, or a plain terminal, Claude Code reads it. The time you spent defining rules — not a second wasted.

Directory structure carries over completely. Obsidian doesn't require a specific directory structure. Nine workspaces, three-tier organization — all running as before after migration.

All Skills carry over completely. All 55 Skills are editor-independent. The UI changed; the execution logic didn't.

What Must Change

Build the habit of using [[bidirectional links]]. This is the biggest new capability Obsidian gives you. When writing notes and encountering important concepts, names, or project titles, just type [[and link to them. It feels like an extra step for the first few days, then becomes muscle memory within a week.

Check the graph view regularly. Spend 5 minutes each week opening Graph View for a scan. Which topics have formed clusters? Which notes are islands? Which concepts bridge different domains? This bird's-eye perspective is something a plain filesystem can never give you.

> **What migration really is:** Not switching tools — adding a better UI layer on top of what you already have. Files, structure, rules, Skills all carry over. What Obsidian adds is bidirectional links, graph view, and a plugin ecosystem — things CLAUDE.md alone can't deliver. The cost is close to zero.

A Final Word

Everything in this book — Obsidian, Claude Code, bidirectional links, automated organization — each one is just a tool or technique on its own. But together, they point to a bigger shift.

For the past few decades, every knowledge tool we've used — from notebooks to Word to Notion — shared a common assumption: organizing is your job. You're responsible for categorizing, tagging, indexing, and maintaining structure. The tool provides the container; you provide the order.

AI changes that assumption. You can just throw things in, and let the AI handle the organizing. You write; it weaves the web.

This sounds like a small change, but its consequences may be bigger than we think. When "organizing" is no longer the bottleneck, your note-taking behavior changes. You record more, because you're no longer worried about "writing it down but never finding it again." You associate more freely, because AI helps

you discover connections. Your note system transforms from a garden that requires maintenance into something that grows on its own.

I'm not sure what this ultimately grows into. But I know one thing: the people who start accumulating now, as AI capabilities continue to improve, will possess something others don't. Not tools, not techniques. Data — everything you've thought about and written down over the years.

Tools will change. Today it's Obsidian; tomorrow it might be something else. But those Markdown files in your vault — they're yours.

Obsidian + Claude Code

Rebuild Your Second Brain with AI

Huashu

AI Native Coder • Indie Developer • Tech Content Creator

Creator of "Kitten Ring Light" (App Store Paid App #1)

GitHub Repository →

Bilibili • X/Twitter • YouTube • Website

Created by Huashu • v1.0.0 • April 2026