



Infrastructure for One

APRIL 2026

10 min read • 2,221 words

Building software for yourself used to be a rich man's hobby.

Not rich in money — rich in the rarer currencies of expertise, momentum, and uninterrupted weekends. The activation energy for personal infrastructure was brutal. A plugin that shaved thirty seconds off a daily workflow was not, by any honest accounting, worth the fourteen hours it would take you to write it, debug it, maintain it, forget how it worked, and rewrite it again in two years. So you tolerated the imperfect fit of tools designed for a statistical user who wasn't quite you. You learned Notion's opinions about databases. You bent your note-taking to Evernote's tag model. You accepted that the shape of your thinking would be quietly pressed into whatever mold the product team at some company in San Francisco had decided was most representative of their addressable market.

This was rational. The math was the math. Building your own always cost more than living with someone else's assumptions, and at the aggregate scale that economics pays attention to, the aggregate scale won.

AI collaboration has broken that math. Not softened it, not adjusted the coefficients — broken the equation. When you can ship a working Obsidian plugin in an hour instead of a weekend, the question stops being is this worth building? and becomes what would I build if it were free? For more and more of us, the honest answer is: everything. Infrastructure for one is quietly becoming

the default rather than the exception, and the tools that result have a qualitatively different character than products do. They can be exactly right, because they don't have to be approximately right for anyone else.

The threshold that disappeared

Five weeks ago I wrote [Obsidian Vaults & Claude Code A Second Brain That Thinks Back](#), an essay about this vault. That vault had 467 markdown files. Its top level was numbered folders — `000 Meta`, `200 Projects`, `300 Areas`, `400 Knowledge`, and so on, the PARA method with Dewey Decimal cosplay. I described the numbered hierarchy as scaffolding that enforced spatial order in the file browser. I was telling the truth. I was also telling the truth a structure imposes, which is not always the same truth.

That vault is gone.

The one I'm writing in now has named sections — `Knowledge/`, `Writing/`, `System 777/`, `Life/`, `Library/`, `Projects/`, `Daily/` — organized by meaning rather than by number. It holds 786 markdown files and around 685,000 words, which is a decade of output or roughly seven to ten novels' worth of prose, depending on how generous you're being with the word "novel." The structure didn't change because I learned a better productivity system. It changed because my relationship to the vault deepened to the point where the numeric scaffolding started feeling like training wheels welded onto a bicycle I'd already learned to ride. The named version is a home. The numbered version was an address.

I bring this up because the old essay couldn't have anticipated the particular shape of this one. Between then and now, in a single afternoon, I built three things I would have carried on a someday-maybe list indefinitely before AI collaboration got good enough to make "someday" collapse into "now."

The first was [sync-repo.py](#), a bidirectional sync engine between this vault and the git repo that backs my website. The old essay didn't mention sync because there wasn't any — just a one-way import script that pulled essays into the vault and could never push back. The new script walks both sides, pairs files across incompatible filename conventions (the repo uses `YYYY-MM-slug_with_underscores.md`, the vault uses `Human Readable Title.md`, bridged by

date and slug frontmatter fields), applies four bidirectional transforms, and uses content hashing to see past iCloud's habit of touching mtimes during its own internal housekeeping.

The four transforms: frontmatter strip when pushing vault → repo (the repo doesn't use frontmatter), Tufte `<label><input type="checkbox"/>[^sn-orphan-1]` HTML rewritten into Obsidian `[^sn-name]` footnotes with the sidenote ID preserved under an `sn-` prefix so a round trip is byte-exact, slugged repo filename ↔ human-readable vault filename bridged via date and slug frontmatter fields, and `index.md` ↔ `FolderName.md` to match the vault's folder-index convention. Each transform took maybe fifteen minutes to get right in collaboration with Claude. Each one would have taken me an hour alone, minimum, and I'd have cut at least one of them from scope to save time.

Content-equivalent files stop at the hash check; genuinely different files let the newer side win. The script can commit and push the repo when it's done. None of this was hard. It required knowing what I wanted and being able to describe it precisely, which is a different skill from being able to write the regexes myself.

The second was `kr-vault`, a custom Obsidian plugin of maybe two hundred lines of JavaScript whose only job is to surface the sync scripts as command-palette entries. Modal dialog, streaming stdout and stderr, status dot that pulses while running and settles green when done, stop button. `cmd-p`, `sync`, return. The terminal had always been four keystrokes away and I had made that trip for twenty years, but four keystrokes turns out to be exactly enough friction to stop you from doing something at the thousand-times-a-year scale that matters for habits. Not enough to stop you once. Enough to stop you on the tenth time. Later in the same afternoon I added a publish command that fuzzy-picks a draft, rewrites its frontmatter (injects the date, injects the slug, strips `status: draft`), moves it from `Writing/Drafts/` to `Writing/Essays/`, and chains into sync-and-push. End-to-end from draft to live website in a single command.

The third was a `--normalize` flag bolted onto the sync script. The regular content-hash sync had been correctly ignoring 271 old essays that still carried raw Tufte sidenote HTML from the old one-way import — hash matched the repo, no action needed — but I wanted them normalized into Obsidian footnote form on the vault side regardless. One flag, four seconds of execution, 271 essays rewritten. The little clickable checkboxes that had been scattered through years of my own prose, and that had unconsciously trained me to skip past the

margins as interactive chrome, were gone. I read three old essays afterward and noticed sidenotes I had apparently been blind to for months. The tool changed what I could see.

Also, somewhere in there, a mobile safe-area fix on my custom Obsidian theme — the iPhone's Dynamic Island had been clipping the title on iOS. Three CSS rules, two minutes.

What this is evidence of

None of those artifacts is the subject of this essay. They're the evidence.

The evidence is: every one of those builds, ten years ago, would have been on a list. The filename bijection would have required reasoning through three edge cases before I started typing. The regex for the Tufte sidenote rewrite would have meant loading the shape of the DOM into my working memory and keeping it there across a weekend of interrupted attention. The Obsidian plugin would have required reading the Obsidian API docs, internalizing their conventions, figuring out how to spawn a child process from inside a plugin sandbox, handling stream multiplexing, learning their modal primitives. Each one was a project. In aggregate, an impossible pile of projects, which is why they would have stayed on the list, which is why I would have kept running `python3 scripts/sync-repo.py` from the terminal like an animal.

In an AI-collaborative workflow, they were an afternoon.

Two of my custom Obsidian themes, Reitz Mono and tufte-reitz, predate all of this work. I built them by hand, slowly, over months, because I wanted them enough to pay the old price. Evidence that the impulse was going to find expression regardless — AI just dropped the threshold so that more of the impulses actually get built instead of staying as itches.

This is the phase change I think people still haven't quite named. It's not that AI makes personal infrastructure economically worth it in some new way. It's that AI makes the threshold evaluation disappear. The project doesn't need to be worth a weekend. It needs to be worth an afternoon. Sometimes an hour. At that price, a lot of things that were never going to get built get built, and the cumulative effect is not a small productivity improvement. It's a different relationship between a person and their tools.

The recursive shape

The tools you build for yourself shape you. This sounds obvious when you say it out loud, and it is, but the implication is less obvious: at $n=1$, the shaping is radically personal in a way that mass-market software can never quite be.

The vault I described in March had trained me, over two years, to think in numbered categories. The vault I'm writing in now is training me to think in named relationships. Neither is better in the abstract. Both are environments, and environments shape their inhabitants. What changed isn't my productivity. It's what thoughts I find myself having when I open the app in the morning — which are, in a not-small way, a function of what the first screen looks like and what labels sit on the folders. I built a home, and now I live in that home, and the home lives in me back.

The plugin I built this afternoon changed what I publish. Not abstractly. Specifically. Before the publish command existed, the friction between "done writing" and "live on the site" was high enough that I'd finish an essay and let it sit in Drafts/ for days while I summoned the energy for the deploy ritual. Now the friction is three words in a command palette. The number of essays that reach the public side of my writing is going to be higher this year than last, and the reason isn't discipline. The reason is infrastructure. Infrastructure is never neutral. It is always encoding a set of defaults, and defaults compound.

This is [For Humans](#) operating at $n=1$. The philosophy I spent a decade applying to Python libraries — design from the mental model of the user outward, let the interface fit the hand — was always, secretly, a philosophy about a single specific human on the other end of the code. "Most developers" is a story you tell yourself to justify building for the one developer you actually know, which is yourself and your friends. Scale the n down far enough — all the way to one — and the principle doesn't break. It sharpens. An interface designed for exactly one reader can afford to fit exactly, because it isn't trying to fit anyone else.

This is, I think, the thing to watch. AI is being talked about as a writer, as a coder, as a tutor. What it also is, and what I suspect will turn out to matter more, is an accessibility device for the impulse to build for yourself. The impulse was always there. Most of the impulses died on to-do lists. Now most of them don't.

Caveats, honestly

A few things I want to name before I close, because the cleanest version of this argument is also the least true one.

The over-engineering impulse was always there for me. I would have built these things eventually, because the kind of person who writes two Obsidian themes for their own vault is the kind of person who is going to keep finding things to build whether or not the activation energy is low. AI didn't make me want to do this. AI compressed "eventually" into "this afternoon." That's a real change, but it's not the creation of a new impulse. It's the enablement of an old one.

Not all friction is bad. Some of it is load-bearing. The terminal-first workflow I spent twenty years in did have the property that it forced a small moment of deliberation before I ran something, and I will occasionally miss that on the day I run sync five times in fifteen minutes because I can. The goal was never zero friction. The goal was to choose which frictions — to retain the ones that served me and eliminate the ones that didn't — rather than inheriting whatever set of frictions had been bequeathed to me by whoever designed Notion or Evernote or the macOS Finder.

And there is a trap at the end of all this that I want to flag clearly. You can spend your life building infrastructure for your work instead of doing your work. Personal tooling can become a very sophisticated form of procrastination, and AI collaboration makes the procrastination more ambitious without necessarily making it less procrastinatory. The question to keep asking is whether the tool you're building will let you do more of the thing you actually care about, or whether the tool has become the thing you care about. Both answers are sometimes legitimate. Neither is automatic.

But with those caveats held honestly, the direction seems clear. The default is shifting. Infrastructure for one is becoming ordinary. The tools that result will be stranger, more idiosyncratic, and — at their best — more exactly right than anything a product team could ever build for a population of users that includes but does not center you.

You are, it turns out, allowed to live in software that fits.

