The Business That Remembers Itself

Give your AI a permanent memory — in files you own

MICHAEL PURVIS

THE CONTEXT ARCHITECTURE SYSTEM

The Business That Remembers Itself

Give your AI a permanent memory — in files you own

Michael Purvis

First edition. Every interface screenshot is a real capture from a complete build run, August 2026; the If-this-looks-different boxes exist because software moves.

contextarchitecturesystem.com · contextarcana.com

Contents

How to Read This Book

This is a short book with one job: by the end of it, your business will have a permanent, portable memory that any AI can read — and you will own every file of it.

What you need. A computer (Windows or Mac) — or an iPad, which has its own honest path in the appendix — plus the AI chat you already use, and one free afternoon for Chapter 5. This book plays no favorites among AI tools: it teaches one universal method that works in all of them, with specific routes for Claude, ChatGPT, and Cursor in the appendix. No programming experience is assumed anywhere in these pages. If you can use email and a word processor, you are qualified.

How it's built. Chapters 1 through 4 are the *why* — read them anywhere, no computer needed. Chapter 5 is the *build*: five stations, every click written out, done in one sitting with this book beside the keyboard. Chapters 6 and 7 wake the system up and give it its first week. Chapter 8 is where it all lands.

Windows, Mac, and iPad. Where platforms differ, the step says so by name, right at the step — this book never hands you instructions for an operating system you don't have, and never assumes software its own pages didn't install. Instructions come in two layers: first *what the step accomplishes* (true everywhere, in any year), then the platform's own clicks, each stamped with the date they were verified.

The boxes. Throughout the build chapters you'll meet two recurring boxes. **Safe to try** tells you exactly what a step can and

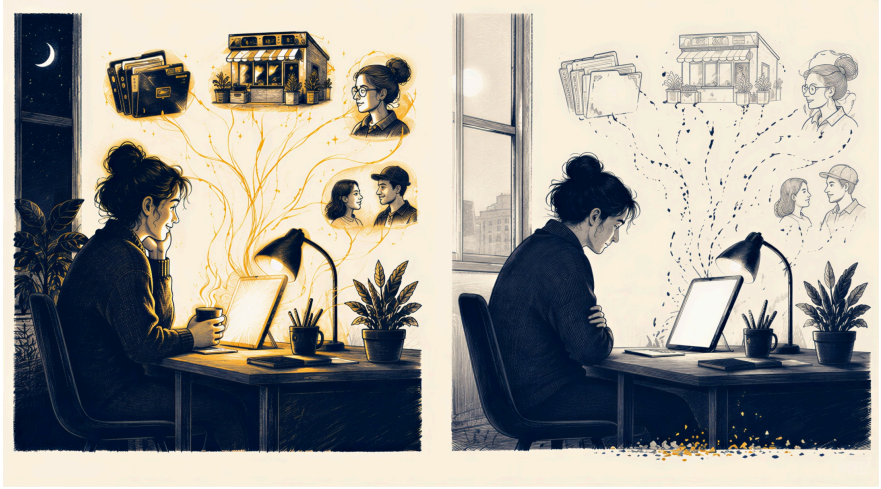
cannot affect — it exists so you never have to wonder whether you're about to break something. **If this looks different** describes what each button is *for*, so that when an app redesigns itself — they always do — you can still find your way. Every screenshot is stamped with the date it was taken.

The links. Every web address in this book is printed in full, like `contextarchitecturesystem.com`, so you can type it from the printed page. In the electronic editions, the same addresses are clickable. All of them are collected once more on the Resources page at the back.

One suggestion: don't read Chapter 5 in bed. It will make you want to stand up and build the thing — better to arrive at it with the afternoon already cleared.

CHAPTER 1

The Amnesiac Genius



It's eleven at night and you're still at your desk, because for once the conversation is going somewhere.

You've been typing back and forth with an AI for two hours. Somewhere around the forty-minute mark, something shifted. It stopped giving you the generic answers — the ones that could apply to any business, anywhere — and started giving you *yours*. It knows, by now, that you teach on Thursdays. It knows your best client found you through a referral and what that referral said. It knows the thing you sell isn't really the thing you sell — that people come to you for the deposit on trust, and the invoice just formalizes it. You explained all of that, piece by piece, and the AI held it, and the last twenty minutes of drafts have been

good. Actually good. You copy the best one into an email, send it, and go to bed feeling like you finally have a colleague.

The next morning you open a new chat and say, "Let's keep going on the campaign."

And the colleague is gone.

In its place is a polite stranger with the same name and the same face, asking what kind of business you have.

If you've felt that moment, you already know it isn't a small thing. And the forgetting is only half of it. An operator I work with — a man who runs the daily machinery of a retreat business and has no interest in technology for its own sake — once asked an AI to help him connect two tools he actually uses. It answered confidently, prescribed a fix for a setup he didn't have, and left the real problem exactly where it found it: inconclusive. That's the other face of the same disease. **A context-less AI doesn't just forget you; it guesses at you** — it substitutes a generic user, on a generic computer, running a generic business, and serves that person instead. And when I ask operators what they'd want an AI to hold permanently, the answers are never exotic: *what I do. Who my clients are. What I'm in the middle of right now.* From the schedule-driven ones, a fourth: *my priorities — first thing in the morning, last thing at night.* That's not a wish list for a smarter machine. That's a description of a coworker — the minimum you'd expect from anyone who shares your office, and precisely what the chat window doesn't give you.

Here's the part that matters: **this is not a malfunction.** Nothing is broken. The forgetting is how these tools are built. A chat is a sealed room — whatever you say inside it stays inside it, and when the conversation ends, the room is demolished. The AI

companies are adding memory features, and some of them help, but every one of them shares the same flaw: the memory lives *inside their product*. It's theirs. You can't read it as a plain list, you can't carry it to a competitor's tool, and if you cancel your subscription — or they redesign the feature, or the company pivots — it's gone. The genius isn't just amnesiac. The genius's notebook belongs to someone else.

So you pay what I've come to call the re-explaining tax. You pay it in minutes: the ramp-up at the start of every session, catching the stranger up before any real work can happen. You pay it in quality: the tenth explanation of your business is shorter and lazier than the second, so the AI works from a thinner version of you each time. And you pay it in ambition — this is the expensive one. There's a whole category of work you never even attempt, because it would only be possible with a collaborator who *already knows everything*. Not the one-off tasks. The compounding ones. The marketing plan that builds on last month's. The client file that deepens with every call. The pricing decision that remembers why you priced it that way the first time.

Add up those three costs across a year, and the forgetting stops being an annoyance. It's the ceiling on the whole relationship.

Now, the fix — and I want to be precise about what I'm claiming, because you've heard big promises from this industry before.

The fix is not a new app. It's not a subscription, and it's not a smarter model. The fix is almost embarrassingly old-fashioned: **you write down what your business knows, in plain files, in folders, in a structure any AI can read — and you keep those files somewhere you own.** Then every conversation starts the same way: the AI reads your files, and the colleague from eleven

o'clock is simply *back*. Not rebuilt from your tired summary. Back — with your clients, your calendar, your reasons, your work-in-progress, exactly as you recorded them.

That structure has a name — the Context Architecture System, an open standard you'll meet properly in Chapter 3 — and building your own starter version of it takes about one afternoon. I've watched people do it who had never heard the word "repository" that morning. The rest of this book is that afternoon, one step at a time, plus the first week that turns the files into a habit.

There's one more thing worth saying before we start, because it's the quiet heart of everything that follows. The files will do more than brief your AI. A business that writes down what it knows — what it offers, who it serves, what it decided and why — becomes clearer to its *owner*, too. The AI is the reason you'll build this. It won't be the biggest thing you get out of it.

Do this now: open whatever notes app is closest and write down the three things you most wish your AI already knew about your business. Don't polish the list. In Chapter 7 you'll watch those exact three items find their homes — and you'll never type them from scratch again.

CHAPTER 2

Files You Own

Ask yourself a strange question: *where does your business live?*

Not the legal entity. The knowledge. The thing that would walk out the door if you did. For most one- and two-person businesses, the honest inventory looks like this: most of it lives in your head. A layer lives in chat scrolls — text threads with your partner, old AI conversations you scroll back through looking for that one good draft. Some lives in email, findable only by remembering who said it. Some lives in eleven browser tabs you're afraid to close, and a folder called `New Folder (2)`, and a notes app you half-trust, and — if you're like one operations partner I work with — in the daily juggling act itself: which client account needs access set up, which payment is stuck between two countries, which task was mid-flight when the week caught fire. Nowhere, in all of that, is there a single place where the business *states what it knows*.

You may have tried to fix this before. Most of us have a note-app graveyard: the Evernote era, the Notion workspace built over one ambitious weekend, the task manager that lasted a month, the binder. Each system died the same death — not because you lacked discipline, but because the system asked you to live inside *its* rooms, and your business didn't fit the floor plan. And here's the detail worth noticing: whatever you poured into those tools is, for practical purposes, gone. Still hosted somewhere, maybe. Exportable in some format you'll never open, maybe.

But gone from your working life. When the app died, the knowledge died with it.

Now flip it around. Think about what *has* survived every app you've ever abandoned.

Plain files. The folder of documents you've carried across four computers. The letter your dad typed in 1994 that still opens today. Text files are the cockroaches of computing — I mean that as the highest compliment. Formats with companies behind them die when the companies do; plain text has outlived every one of them, and will outlive every AI company operating today. A folder of plainly written files needs no app, no account, no subscription, and no permission to keep existing.

CAS Principle 2: the context lives in portable files.

So here is the move this whole book turns on. Take that scattered inventory — the head, the scrolls, the tabs — and give it a home: **a set of plain-text files, organized in folders, owned by you.** What you do, in one file. Who your clients are, one file each. How you run the things you run every week. What you decided, and why, dated, so future-you stops relitigating it. None of this is exotic. It's the kind of writing you already do in stray notes — except now it accumulates in one place, in a structure built for it.

Ownership here isn't a mood; it's a checklist, and it's worth being literal about. You own the files: they sit on your computer, and you can copy them to a thumb drive, a backup disk, or a new machine with a drag of the mouse. No export button, because there's nothing to export *from*. You own the format: anything that opens text can open them — today's cheapest laptop, and with near-certainty, whatever we're all using in 2046.

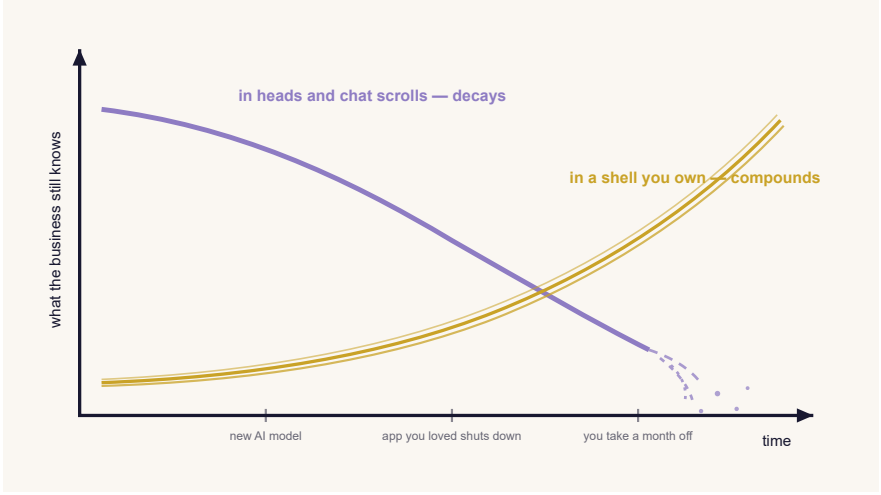
And you own the arrangement: the folders are yours to reshape. No vendor's floor plan. No rooms you can't move.

CAS Principle 1: the user owns the context.

"But what does this have to do with the amnesiac genius?" Everything — because of one convenient fact about the current moment: **every AI on the market can read files.** Claude, ChatGPT, Gemini, the open-source model running on some enthusiast's desktop — all of them, handed a folder of plain text, can absorb it in seconds and behave as if they'd known it all along. Your files become the briefing packet. The stranger reads it and stops being a stranger — in the first minute, every time.

Notice what that quietly does to your relationship with the AI industry. The memory no longer lives inside anyone's product. It lives in your folder, and the AI tools become interchangeable readers of it. If a better model ships next spring — switch; your new hire reads the same briefing on day one. If your favorite tool triples its price — leave; the notebook was never theirs. The tools keep improving and keep churning, and none of the churn touches what you've built, because what you've built is upstream of all of them.

CAS Principle 4: the context layer outlives any model, vendor, or tool.



I won't pretend the files write themselves. The afternoon of setup is real, and so is the habit that follows it — this is a practice, not a purchase, and Chapter 7 is honest about the maintenance. But the writing turns out to be lighter than the graveyard systems ever were, for a reason you'll feel by the end of this book: every file you add makes your AI visibly smarter *that same day*. The feedback is immediate. Systems with immediate feedback are the ones that survive.

One question remains before you build. If the files can be *anything*, you'll freeze at the blank folder — the graveyard was built on blank pages. What you need is a floor plan: which files, which folders, what goes where — one that thousands of hours of real client work have already shaped, and that any AI can navigate without instructions. That floor plan exists, it's free, and it has a name.

Do this now: spend five minutes on your own knowledge inventory. Down the left of a page, list where your business currently keeps what it knows — head, chats, tabs, apps, binders. Next to each, write one thing living there that would

hurt to lose. Keep the page; in Chapter 5 you'll watch each line find a safer home.

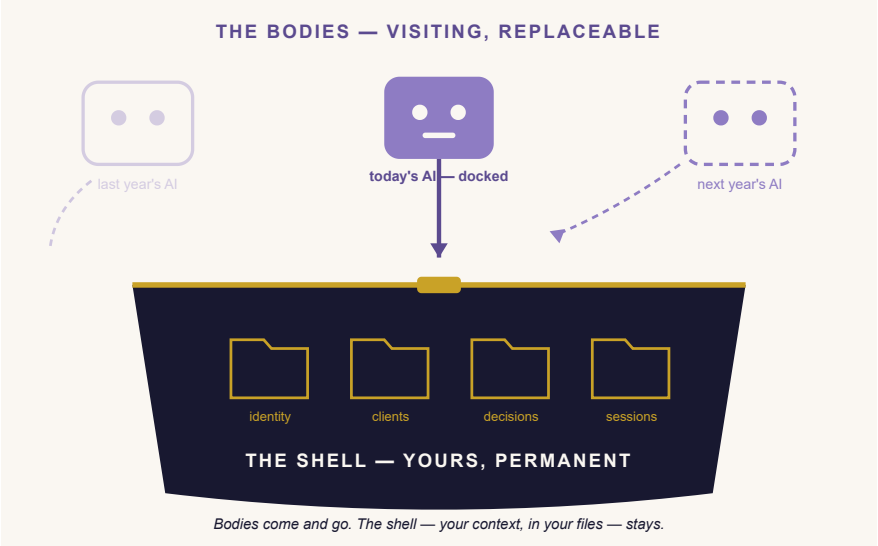
CHAPTER 3

The Shell and the Body

There's an old idea from robotics that earns its keep here: separate the mind's *knowledge* from the machine that carries it, and you can move the knowledge between machines.

The Context Architecture System — CAS from here on — is that idea, applied to your business. It's an open standard: a published, free-to-use blueprint for the folder of files you met in Chapter 2. Open means nobody can take it away from you — the blueprint is licensed so that it stays public, permanently, the way a recipe in a published cookbook can't be un-published. And *standard* means the arrangement is agreed and documented, which is what lets any AI find its way around your folder without you drawing it a map.

CAS gives the two halves of the arrangement names worth learning, because you'll think in them from now on.



Bodies come and go. The shell — your context, in your files — stays.

The shell is your folder of files — everything your business knows, written down, owned by you. It's called a shell because it's shaped like your business: built up slowly, hardened over time, and yours no matter what's living inside it at the moment.

The body is whatever AI you plug into the shell this year. Claude today, maybe something else next spring. Bodies are temporary by design. They visit, they read the shell, they do excellent work *because* of what they read — and when a better body comes along, you swap it, and the shell doesn't notice.

Hold onto that picture, because it inverts the way most people frame AI. The common question is "which AI should I commit to?" — as if you were choosing a spouse. The shell turns it into "which AI is doing the best work *this quarter*?" — hiring a contractor, with the company knowledge staying in the company. You stop betting your workflows on any vendor's roadmap. The commitment, the accumulation, the *investment*, all live on your side of the line.

That's the theory. Theory is cheap. Let me show you a shell with the drawers open.

A tour of Riverstone Coffee

The CAS standard ships with a complete worked example: Riverstone Coffee Roasters, a fictional small-batch roastery operating out of a converted mill in a town called Hollingford. Two people, a five-kilo roaster, a booth at the Saturday market. Their whole business memory is a public example shell you can browse file by file, linked from the standard's home page at `contextarchitecturesystem.com` — and reading it teaches more than any diagram, because you can feel what it would be like if these drawers held *your* pages. Four stops.

Stop one: identity. Riverstone's `brand-voice.md` doesn't say "we value authenticity." It legislates. Four rules ("Sensory, not academic. Honest about scale."), a banned-words list — *premium*, *curated*, *artisanal* — and a table of real rewrites: not "hand-crafted with passion" but "Roasted Tuesdays. Bagged Wednesdays. Shipped Thursdays." Then one test to settle every argument: if a sentence would sound out of place spoken aloud at the Saturday market booth, rewrite it. Hand an AI that file and ask for an product announcement, and it *cannot* produce the marketing sludge you've been deleting from every draft — the file forbids it. That's what it means for the shell to hold your voice: taste, made enforceable.

Stop two: offers. The wholesale program file reads like the owner's actual knowledge, because that's what it is: three pricing tiers with volume breaks and payment terms, what's included, and — the part a brochure would never show — what's *not*, with reasons. No equipment loans: "we're not big enough."

There's even the pricing philosophy underneath: they'd rather have ten steady mid-size accounts than thirty sporadic small ones. Now the AI drafting a quote for a new café isn't guessing at your prices or, worse, inventing generous terms you never offered. It's reading the sheet.

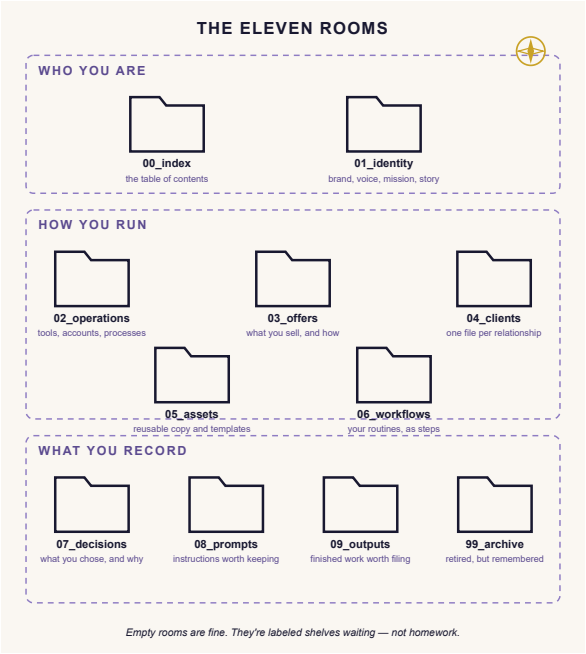
Stop three: decisions. In September 2025, Riverstone decided to shift two of their four coffee origins to direct-trade sourcing. The decision record holds the whole reasoning in one page: what prompted it (two wholesale accounts kept asking about farm relationships), what they decided, why, and what it committed them to. Every business makes calls like this. Almost no small business writes them down — so a year later, the *why* has evaporated and the argument happens again from scratch. A dated one-pager ends the relitigating, and gives your AI something no chat history ever holds: not just what you do, but what you've *ruled out, and for what reason*.

Stop four: workflows. The roast-cycle file is Tuesday morning, written down: warm the roaster at 6:30, run a warm-up batch of cheap Brazilian, then per-batch steps with times and temperatures, and a troubleshooting list of what goes wrong and what each failure means. Whatever your Tuesday is — client onboarding, monthly invoicing, the setup checklist for a session — writing it once buys the same two dividends: anyone (human or AI) can now execute or sanity-check the routine, and the routine stops living in your head, where it costs attention every time you run it.

The eleven rooms

Those four stops are drawers in a larger cabinet. A CAS shell has eleven context folders, numbered so they always sort in order.

Here's the whole map — one plain sentence each, no memorizing required:



- **00_index** — the table of contents: what's in the shell and where.
- **01_identity** — who you are: brand, voice, mission, founding story.
- **02_operations** — how the machine runs: tools, accounts, processes.
- **03_offers** — what you sell, at what price, positioned how.
- **04_clients** — one file per relationship: history, preferences, state.
- **05_assets** — reusable material: copy that worked, templates, media notes.
- **06_workflows** — your repeatable routines, written as steps.

- **07_decisions** — dated records of what you chose and why.
- **08_prompts** — instructions to your AI that worked well enough to keep.
- **09_outputs** — finished AI work worth keeping on file.
- **99_archive** — retired content, kept because history is context too.

Empty rooms are fine — Riverstone's shell has gaps, and yours will too. The standard's own rule is that the structure holds even when folders sit empty; they're labeled shelves waiting, not homework. You'll fill perhaps four rooms in your first week, and the rest as the business walks material through the door.

One honest caveat before the toolbox comes out. A shell is a working file cabinet, not a diary — the standard assumes its files may someday be read by an assistant, a collaborator, or a buyer of your business, and Chapter 5 will show you the guardrail that keeps passwords and secrets *out* of it entirely. What goes in is the knowledge; what stays out is the keys.

So: the floor plan exists, the example is browsable, and every room has a name. What stands between you and a shell of your own is the mechanical part — creating your copy, putting it somewhere safe, and learning the two-click habit that saves its history forever. That's the next two chapters, and it's where the two fears you brought to this book — *what if I break something, what if I lose my work* — get permanently retired.

Do this now: browse the Riverstone example shell — start at `contextarchitecturesystem.com` and follow the example link (it's also on the resources page at the back, next to the template you'll use in Chapter 5). Open any three files that pull at you.

You're not studying — you're noticing, for three files, the same thought: *I know this about my business too. It's written nowhere.*

CHAPTER 4

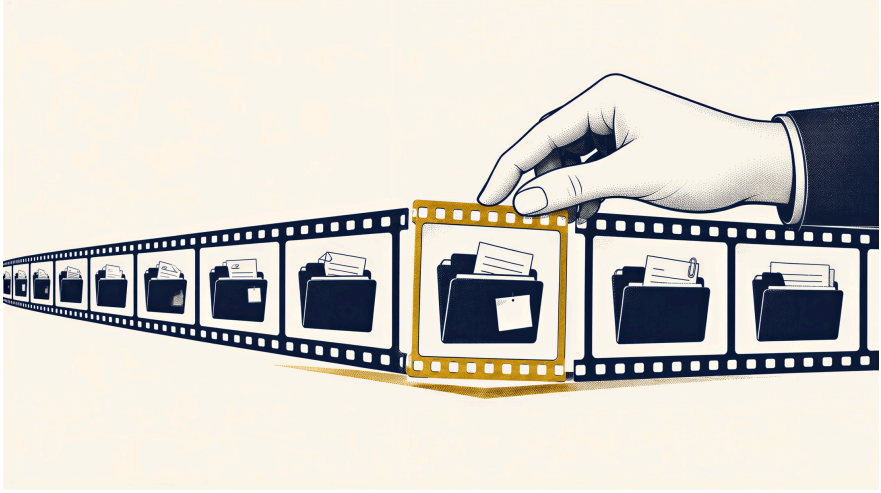
The Undo Machine

Before you build anything, we need to retire two fears. When I ask operators what stops them from trying a new system, the same two answers come back almost word for word: *breaking something*, and *losing my work*. They're good fears. They come from experience. And this chapter exists because the tool that dissolves both of them has the worst public image in software.

That tool is called **Git**, and I'd like to reintroduce it, because you've only ever seen it from across the room at a party full of programmers.

Git was built in 2005 for people who could not afford to lose work — thousands of programmers editing the same enormous project, where one overwritten file could cost months. The solution they built is not complicated in spirit. It's this: *keep everything, forever.*

Here's the whole idea. Git watches a folder. Whenever you tell it "this is a moment worth keeping," it takes a *snapshot* — called a **commit** — of every file in that folder, exactly as it is right now, along with a one-line note from you about what changed. The snapshot is permanent. Take another tomorrow, and Git keeps both. A hundred snapshots later, you have a hundred moments of your folder's life, every one of them still there — and you can open any of them, any time, and pull any file back from any point in its history.



Think of it as an undo machine for a whole folder — an undo that doesn't expire when you close the program, doesn't run out after thirty steps, and never gets confused about which version was which, because you labeled each one on the way in.

Now walk your two fears through that machine and watch what happens to them.

What if I break something? Say it's three weeks from now and your shell has two dozen files. You rewrite your offers file and, in the process, delete the paragraph explaining your pricing — the one with reasoning you'd already half-forgotten. In the old world, that paragraph is simply gone; you notice in a month, and reconstruct it badly. In the new world, the version from before your edit sits in a snapshot, labeled and dated. Getting it back is a two-minute, click-based errand — and in Chapter 7 you'll run exactly that errand on purpose, so you've *felt* it work before you ever need it for real.

What if I lose my work? This one has a second layer, because a snapshot on your computer can't protect you from the computer

itself — the stolen laptop, the spilled coffee, the drive that won't spin up. So Git has a companion: **GitHub**, a website that holds a linked copy of your folder and its entire snapshot history. Each time you take a snapshot, you also **push** it — send it up to the safe copy — with one click. From then on, your business memory exists in two places that would have to fail on the same day. New laptop? Sign in, download your working copy, and every file and every snapshot is there, back to the first afternoon. (You met this pattern in Chapter 2 as Principle 3: version everything, where possible, with Git.)

A fair question at this point: *why do I need any of this machinery for a folder of notes?* You could keep your shell as a plain folder, and it would still beat the chat scrolls. But the machinery is what upgrades the files into a *record*. The history answers questions the files alone can't: what did our prices look like in March? When did we change the guarantee, and what did it say before? And the safe copy means the record can't be lost to one machine's bad day. For a business memory meant to last a decade, "current version only, one copy" isn't keeping records. It's keeping luck.

Two honest cautions before the toolbox opens, so nothing in the next chapter surprises you.

First: Git is a professional tool with a large vocabulary, and most of it is built for those thousands of simultaneous programmers — merging, branching, coordinating parallel work. **You are one person with one folder. You will use four moves, ever:** copy the folder down, edit files, take a snapshot, push it to the safe copy. Everything else in Git's vocabulary you will never touch, and this book will never ask you to. When some future tutorial tries to teach you "branching," you have my permission to close the tab.

Second: you'll hear that Git means "using the terminal" — typing commands into a black window. It doesn't, not anymore. Nearly everything you'll ever do happens in **GitHub Desktop**, a free point-and-click program made by GitHub for exactly this. In this entire book, the terminal appears once, in the next chapter, for one two-line moment that installs your shell's safety guard — and you'll see the exact text it prints back before you ever run it.

So here's where you now stand. A snapshot system that keeps every version of every file, forever. A safe copy that survives any single machine. A point-and-click tool that drives it all. Breaking something stopped being possible the moment the first snapshot existed — from then on you can only add versions, never destroy them. What's left is to set it all up, once.

That's the next chapter: one afternoon, five stations, every click written out. **Do this now:** nothing — except, if you like, one small ceremony. Find the folder called `New Folder (2)`, or whatever yours is named, and look at it one last time. You're about to replace the era it stands for.

CHAPTER 5

Your First Shell in an Afternoon

At the end of this chapter you will have: your own shell, on your own computer, with a safe copy on the internet, a working undo machine, and a guard that keeps secrets out of it. Plan for sixty to ninety minutes, most of it the comfortable kind — filling in forms and watching progress bars.

Five stations. Do them in order; each one assumes the last. Every station tells you what you'll see when it worked. If a screen ever doesn't match these pages, look for the **If this looks different** box nearby — the purpose of each button is described alongside its location, so you can find it even after the next redesign.

Version stamp: every screenshot in this chapter is a real capture from the live services, photographed in August 2026 during a complete run of these five stations.

> **Working from an iPad (or no computer at all)?** This chapter's path uses installed desktop apps. There is an honest iPad route — same shell, same history, done in the browser — in the appendix **Working From an iPad**. Read Stations 1 and 2 here (they're identical), then switch.

Station 1 — Your GitHub account

GitHub is the website that will hold the safe copy of your shell (Chapter 4). Accounts are free, and the free tier includes everything this book uses — including keeping your shell private.

Step 1. In your web browser, go to `github.com/signup`.

Step 2. You'll see a form titled **Create your free account**, asking for four things: an email address, a password, a username, and your country. (You can instead use the *Continue with Google* or *Continue with Apple* buttons if you prefer — either is fine. If you do, skip to Step 4.)

Step 3. Two of the fields have rules worth knowing before you type:

- *Password* — at least 15 characters, **or** at least 8 characters if they include a number and a lowercase letter. Let your password manager generate it if you use one.
- *Username* — letters, numbers, and single hyphens only. Choose it a little carefully: it becomes part of your shell's web address, the way `context-arcana-llc` appears in the addresses you'll see below. Your name or business name, lowercased with hyphens, is the classic choice.

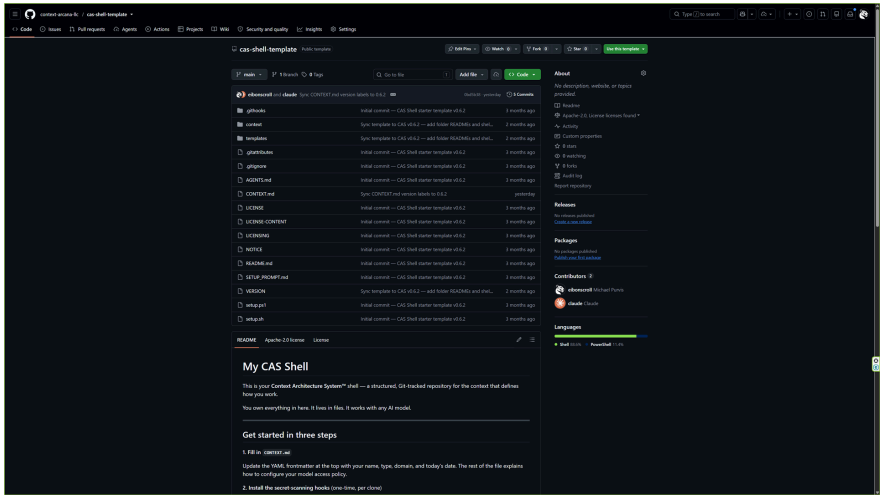
Step 4. Click **Create account** and complete the verification GitHub asks for (a code sent to your email, and sometimes a short puzzle). When you can see a page with your username in the top-right corner, you have an account. You'll see an empty home page — GitHub assumes you're a programmer and offers programmer things. Ignore all of it.

> **Safe to try:** nothing at this station touches your files or your computer. The worst possible outcome is an unused free account.

Station 2 — Copy the template

The starter shell — the eleven folders from Chapter 3, the manifest file, the guard scripts, all of it — exists as a public **template** you copy with one button. Your copy will be entirely yours: separate from the original, invisible to me, editable without asking anyone.

Step 1. While signed in, go to `github.com/context-arcana-llc/cas-shell-template`.



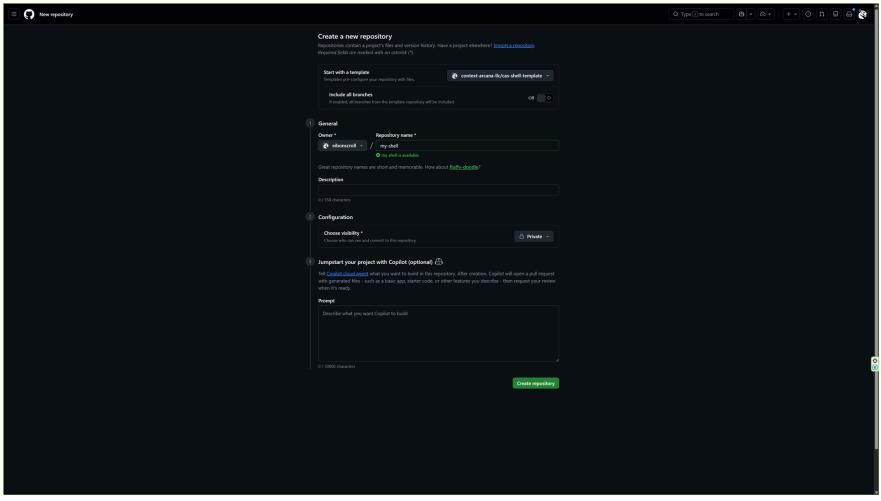
Step 2. Near the top right of the page, find the green button labeled **Use this template** and click it, then choose **Create a new repository**. (*Repository* — a folder with a memory; this is the only

new word this station needs, and it's just what GitHub calls the thing you're creating.)

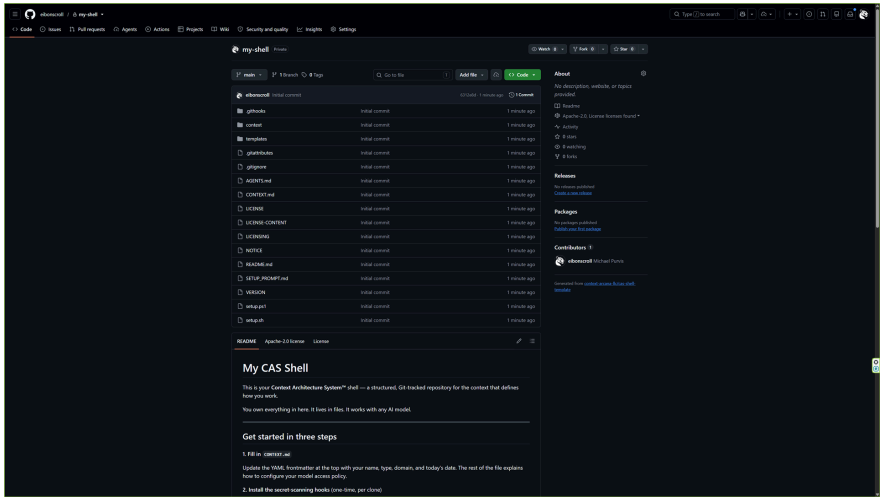
> **If this looks different:** if there is no green button, you are not signed in — this button is only shown to signed-in visitors. Sign in and reload the page. If the button has moved in a redesign, you're looking for the control that copies a template into your own new repository; it has carried the words "template" since the feature existed.

Step 3. GitHub asks you to name your copy and choose who can see it:

- *Repository name* — something plain and permanent: `my-shell`, or your business name with `-shell` on the end, like `riverstone-shell`.
- *Public or Private* — choose **Private**. Your shell will hold client names and business detail; Private means only you, and people you explicitly invite, can ever see it. You can change this later; start closed.



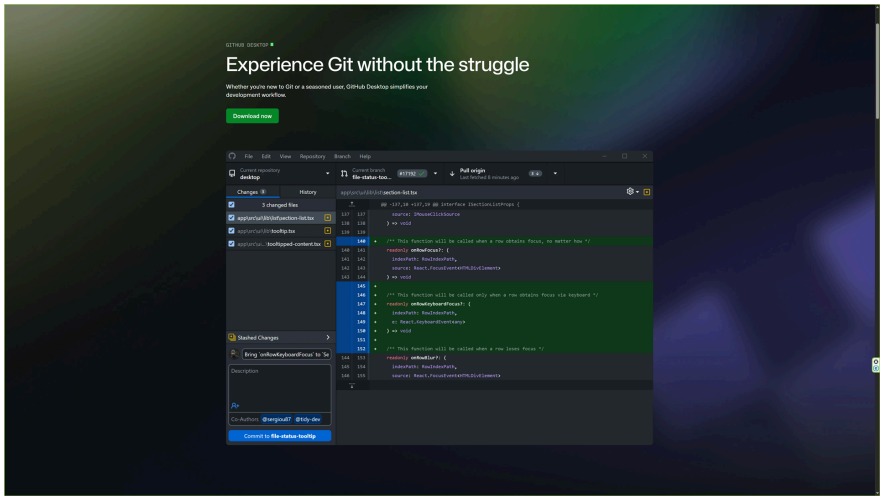
Step 4. Click **Create repository**. After a few seconds you'll see a page that looks like the template — same folders, same files — but with *your* username in the address and a **Private** label at the top. That page *is* your shell. It just doesn't live on your computer yet.



Station 3 — GitHub Desktop

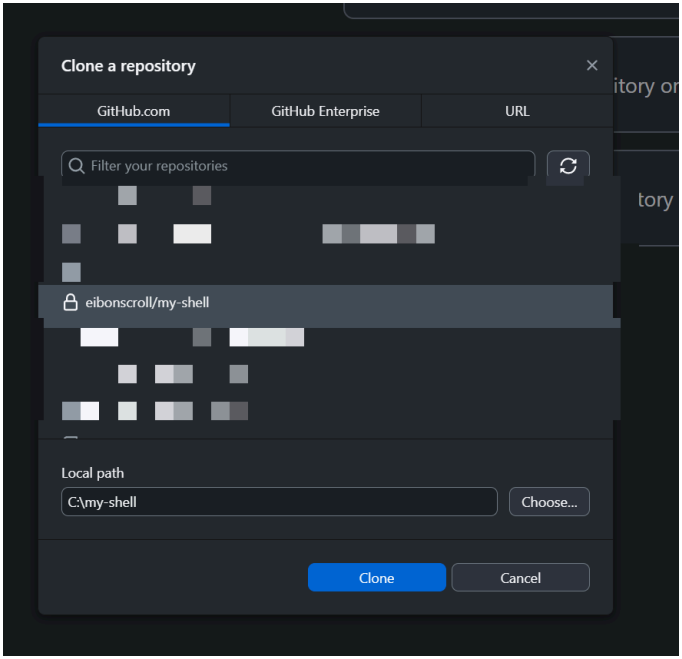
GitHub Desktop is the free point-and-click program from Chapter 4 — the reason this book almost never needs a terminal.

Step 1. Go to `desktop.github.com`, click the download button, and install it the way you'd install anything.



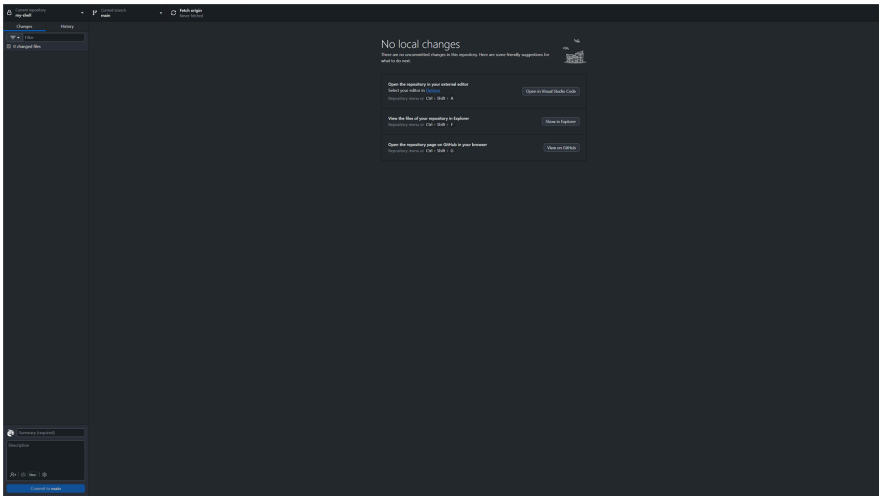
Step 2. Open GitHub Desktop. It will ask you to sign in to GitHub — do, using the account from Station 1. (It hands you to your web browser to approve, then returns.)

Step 3. From the menu, choose **File** → **Clone repository...** (*clone* — download your own working copy). You'll see a list of your repositories on GitHub — a short list, since it contains exactly one thing: the shell you created at Station 2. Click it.



Step 4. Desktop asks where on your computer to put the folder — the suggested location is fine, or choose something short and easy to find (in the figure above we used `C:\my-shell`). Note the path, then click **Clone**. A progress bar runs for a few seconds.

Step 5. You'll see Desktop's main window with your shell's name at the top left and the message **No local changes** in the middle. That message is Desktop's resting state; it means *your computer and the safe copy currently match*. You'll see it a hundred times from now on, and it will always be good news.



Open the folder itself (menu: **Repository** → **Show in Explorer** — on a Mac, **Show in Finder**) and look around. Eleven numbered folders. A `CONTEXT.md`. The shell from Chapter 3, on your machine, in plain files.

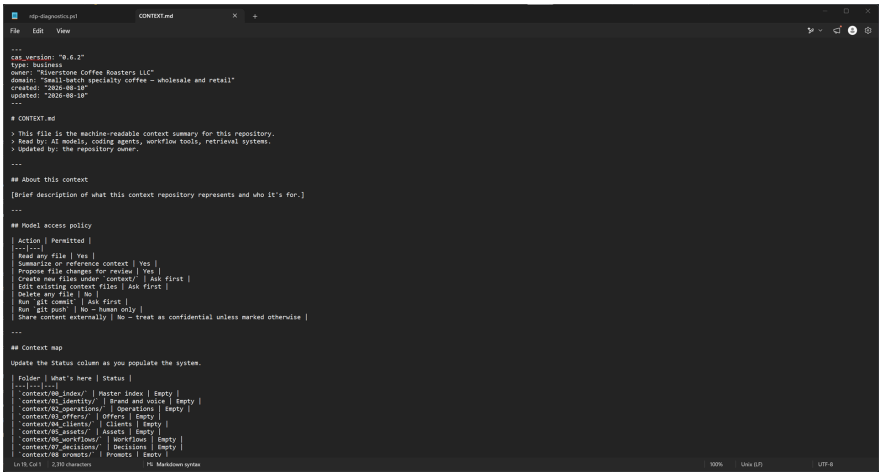
Station 4 — Claim it, and take your first snapshot

The shell doesn't know it's yours yet. Its manifest file — `CONTEXT.md`, the first thing any AI will read — still carries the template's placeholders.

Step 1. In the shell folder, open `CONTEXT.md` by double-clicking it. If Windows asks what to open it with, choose **Notepad** — any plain text editor is fine, and the one already on your computer is plenty.

Step 2. At the very top, between two `---` lines, is the label block — seven short lines the standard calls the manifest. Edit four of them:

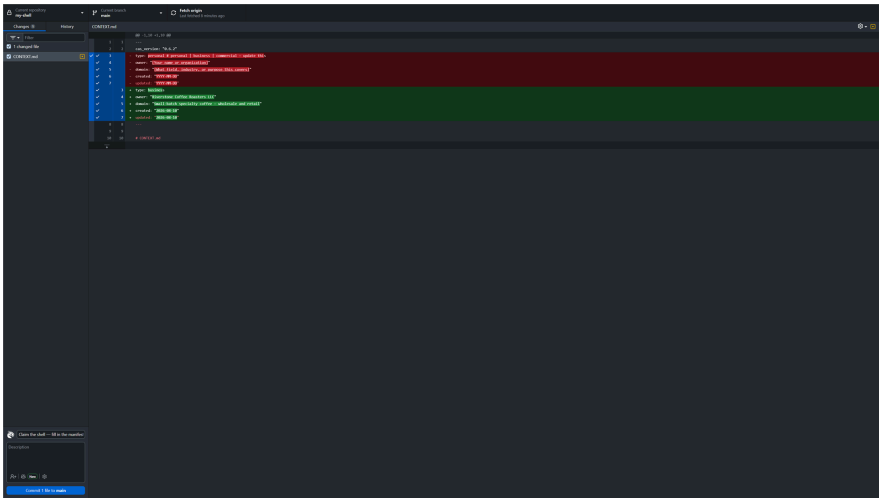
- `owner:` — replace the bracketed placeholder with your name or business name, keeping the quotation marks: `owner: "Riverstone Coffee Roasters LLC"`
- `domain:` — one plain phrase for what this shell covers: `domain: "Small-batch specialty coffee — wholesale and retail"`
- `created:` and `updated:` — today's date, in year-month-day form: `created: "2026-08-08"`
- One more, if it applies: `type:` is set to `personal` — if this shell is for a business, change it to `business`. Leave `cas_version` exactly as it is.



Step 3. Read the rest of the file — it's one page. It contains the model access policy: the house rules stating what an AI may do in your shell (read anything, propose changes) and what it may not (delete, or push work to the safe copy — that button stays human). You met this idea as Principle 5. Here it is as an actual document you own and can amend.

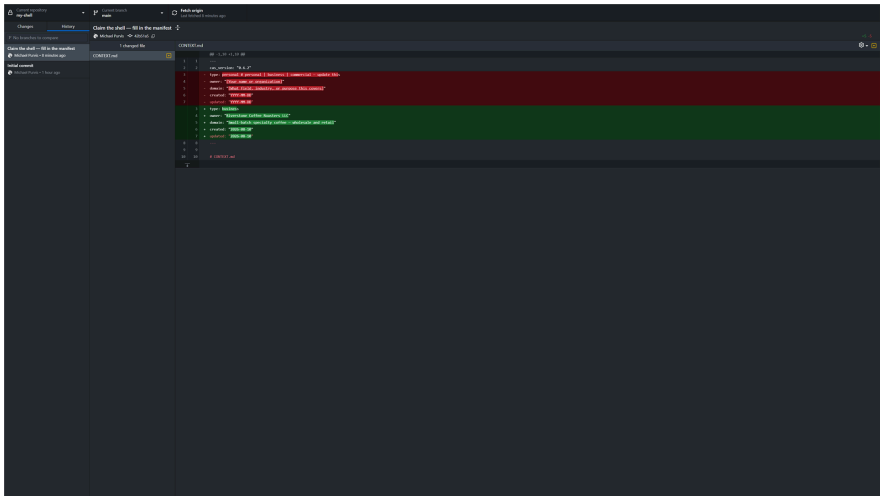
Step 4. Save the file and close Notepad. Now look at GitHub Desktop: you'll see it noticed. The left panel lists `CONTEXT.md` as changed, and the main panel shows exactly which lines are different — old in red, new in green. Nothing is saved to history yet; Desktop is showing you what *would* be in the snapshot.

Step 5. In the lower-left corner, find the small text box labeled **Summary (required)**. Type a short note to your future self: `Claim the shell — fill in the manifest`. Then click the blue button beneath it, **Commit to main** — *commit*, your first snapshot, per Chapter 4. (*Main* is just the name of the shell's timeline.)



Step 6. One click remains: at the top of the window, click **Push origin** — push, send the snapshot to the safe copy. The arrow spins briefly, then the window returns to **No local changes**.

Step 7. Proof. Click the **History** tab on the left. There it is: your snapshot, with your note, under today's date — permanent now, on your machine and on GitHub both.



That two-click rhythm — *Commit to main*, then *Push origin* — is the entire operating skill of the undo machine, and you've just performed it.

> **Safe to try:** from this moment on, nothing you do inside this folder can be permanently lost. Edit boldly. Any file can be brought back to this instant — and to every snapshot you take from now on — and in Chapter 7 you'll practice exactly that restore.

Station 5 — The guard (and the one terminal moment)

Chapter 3 promised that keys stay out of the file cabinet: no passwords, no account numbers, nothing you'd call a secret. Your shell ships with an automatic guard that enforces this — a script that inspects every snapshot *before* it's taken and refuses any that appears to contain a secret or personal identifier.

Installing it is the one moment this book asks you to type into a terminal, and one supporting install makes it work.

Step 1. The guard runs on the full Git toolkit, which GitHub Desktop does not install on its own. Go to `git-scm.com/downloads`, download **Git for Windows** (or the Mac version), and install it accepting every default option. Nothing on your screen will look different afterward — this adds machinery, not a program you'll open.

Step 2. Open a terminal *in your shell's folder*: in GitHub Desktop's menu, choose **Repository → Open in Command Prompt** (that's the wording on a fresh install; if yours says PowerShell or Terminal, use it — every variant works).

Step 3. Type this line exactly, then press Enter:

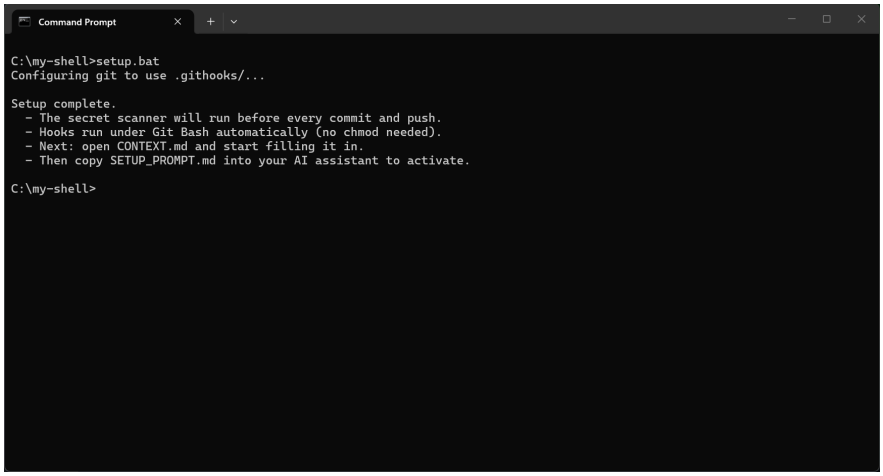
```
setup.bat
```

(If your menu opened PowerShell instead, the line is `.\setup.ps1`. On a Mac, open Terminal from the Repository menu and run `bash setup.sh`.)

Step 4. You'll see the script report what it's doing, ending with:

```
Setup complete.
  - The secret scanner will run before every commit and
push.
  - Hooks run under Git Bash automatically (no chmod
needed).
  - Next: open CONTEXT.md and start filling it in.
  - Then copy SETUP_PROMPT.md into your AI assistant to
activate.
```

Close the terminal. You will not need it again.



> **If this looks different:** if the script instead prints **"Error: Git is not installed,"** Step 1 didn't complete — install Git for Windows and try again. If it prints **"This doesn't look like a Git repository,"** the terminal opened in the wrong folder — use the Repository menu route in Step 2, which always opens in the right place.

> **Safe to try:** the script does one thing — it tells Git to run the guard scripts that already sit inside your shell's `.githooks` folder. It reads; it deletes nothing; it changes no file you've written.

Step 5. Never trust a guard you haven't seen catch something. Open Notepad, and in a new file type one deliberately fake secret:

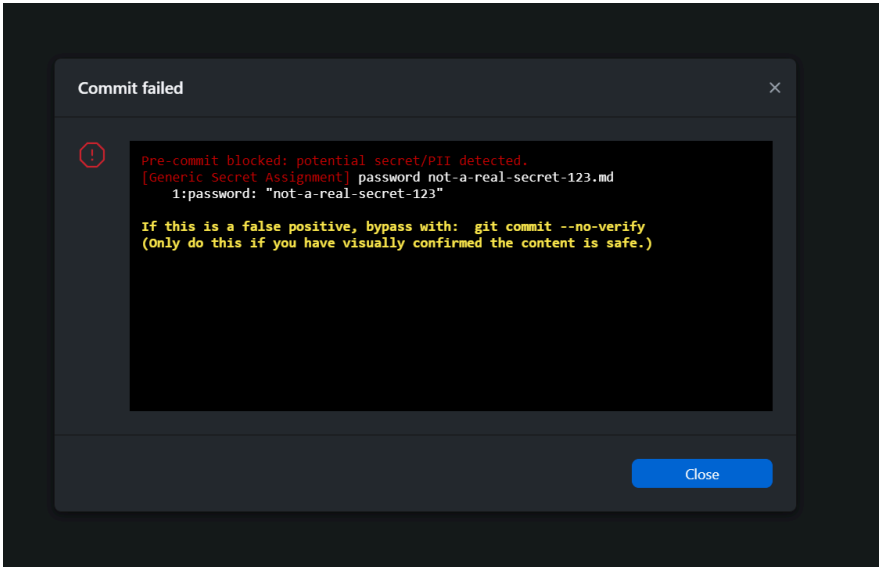
```
password: "not-a-real-secret-123"
```

Save it in your shell folder as `guard-test.md`. In GitHub Desktop, type a summary (`Testing the guard`) and click

Commit to main — and watch the commit be *refused*. Desktop surfaces the guard's report, which begins:

```
Pre-commit blocked: potential secret/PII detected.
```

naming your file and the offending line.



The snapshot was never taken. Now delete `guard-test.md` — and because the file never made it into a snapshot, Desktop's changes panel simply empties back to **No local changes**, as if the whole experiment never happened. The guard has now proven it will do for a *real* mistake — a pasted client email with a bank detail, a password dropped in a note — exactly what it did here: stop the snapshot cold, before it enters permanent history.

Stand up and look at what exists now that didn't this morning. Your business has a file cabinet with eleven labeled rooms; a

manifest that names you its owner and states the rules; an undo machine holding its first snapshot; a safe copy on the internet that matches your desk; and a guard on the door. What it doesn't have yet is the one thing this was all for — a mind plugged into it.

That takes one paste, and it's the next chapter. **Do this now:** nothing. If you've reached this line with the five stations done, the afternoon delivered what it promised — close the laptop; Chapter 6 is a fresh sitting.

CHAPTER 6

Wake It Up

Everything so far has been carpentry. This is the chapter where the lights come on.

Your shell contains a file you haven't opened yet: `SETUP_PROMPT.md`. It's short — one page — and it exists for a single purpose: it is the *briefing you never have to write again*. The top of the file says exactly what to do with it: paste the block inside into a new conversation with any AI assistant. Claude, ChatGPT, Gemini, whatever arrives next year — the file itself commits to that neutrality, and by the end of this chapter you'll have tested it.

Read the block once before you paste it, because it's worth seeing what you're about to hand over. It tells the AI three things. *What this is:* "my owned context layer — a structured collection of files that defines how I operate, what I've decided, and what I've built." *Where to start:* read `CONTEXT.md` first — the manifest you claimed in Chapter 5 — then the index, then confirm it has oriented itself before doing anything else. *And how to behave:* read freely; propose changes rather than making them; never commit or push on its own; cite which file it's drawing from; flag anything that looks like a secret. You'll recognize every line — it's the model access policy from your manifest, the house rules of Principle 5, restated as marching orders. The last line of the block is the whole book in one sentence: *"This context is mine. It outlives any model, tool, or vendor. Treat it accordingly."*

The one mechanical fact

Before the magic moment, one honest mechanical fact — skipping it is how readers end up stuck, and this book doesn't do that to you.

A plain chat window cannot see your computer. If you paste the setup block into a fresh web chat and nothing else, the AI will understand its orders perfectly — and then tell you, correctly, that it has no files to read. The briefing packet has to *reach* it. There are two ways, and both are easy; which one you use depends on the AI app in front of you.

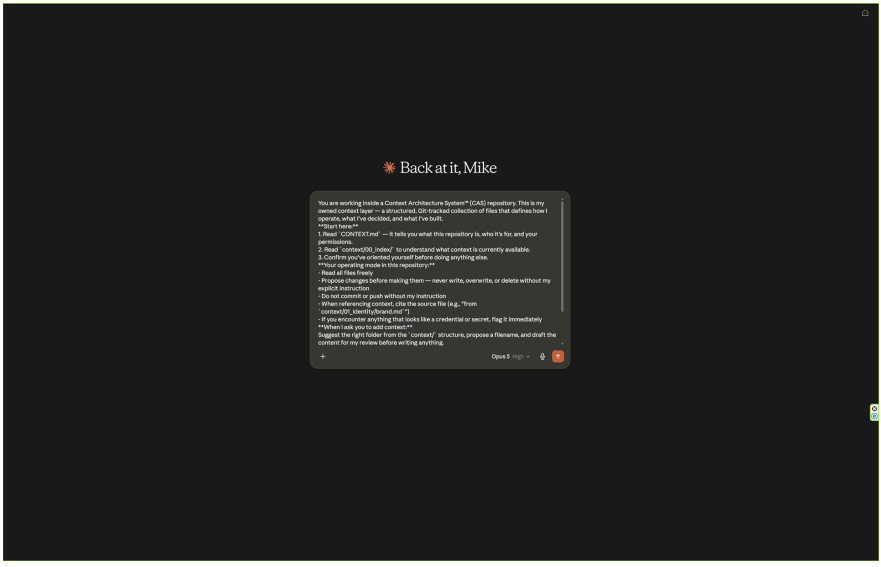
The universal way — bring the files to the chat. Every major AI chat can accept files you attach, the same way you attach things to email. Paste the setup block, then attach `CONTEXT.md` and whichever files today's work needs — working on an announcement? attach your voice file; drafting a client email? attach that client's file. For small tasks you can even skip the attachment and paste a file's contents directly into the chat. This works everywhere, today, with no setup.

The better way — let the AI see the folder. The AI apps are racing to close this gap, and most now offer some version of a persistent workspace — a project, a space, a knowledge area — where you add your files *once* and every new conversation starts already holding them. Some desktop AI apps can be pointed at the shell folder itself. The names and buttons for these features change too often for a book to promise you specifics — so here's the durable version: **look for your AI's feature that keeps files attached across conversations, and put your shell files in it.** Ten minutes of one-time setup, and the re-explaining tax drops to zero clicks.

Either way, notice what your shell just did to a problem that looked unsolvable in Chapter 1. The AI companies' memory features are partial, proprietary, and theirs. Your memory is complete, portable, and yours — and *their* newest features have become mere delivery trucks for it.

The moment

Now do it. Open a new conversation, paste the setup block, and get the files to it by whichever route fits your tool.



Then ask for something real — the file suggests good first moves, and the best one is this:

> "Audit what's missing from my context and suggest what to add first."

Watch what comes back. Not the generic stranger from Chapter 1 — an assistant that answers *from your files, citing them by name*:

that your offers folder mentions a workshop your identity file never describes, that two clients are named in a session note but have no files of their own. It has read your manifest and your rules. It knows what it may do and what it must ask before doing. The colleague from eleven o'clock is back — and this time the briefing didn't come from your tired typing. It came off the shelf.

Now prove the part that matters most. Open a *different* AI — a brand you don't normally use; the free tier is fine. Same paste, same files, same question. Thirty seconds later, the same oriented colleague is looking back at you, wearing a different face. That's Principle 4 — the context layer outlives any model or vendor — not as a claim on a page, but as something you have now *watched happen*. No app promised you that portability. No app could. The folder did it.

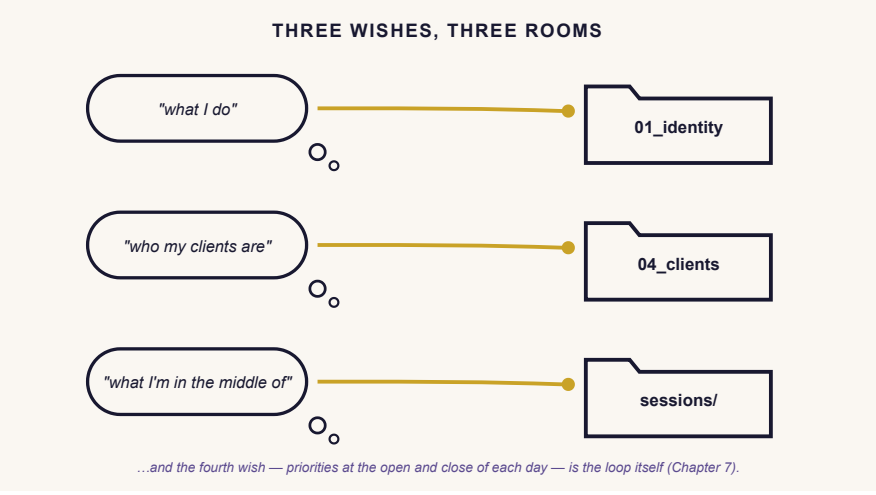
Do this now: the audit question, in your main AI, today. Write down its top three suggestions — they're Monday's shopping list, and Monday is the next chapter.

CHAPTER 7

Your First Week

A shell you built in an afternoon is a promising start. A shell that's still alive in a month is a different thing entirely — and the difference is decided in the first week, while the momentum is real and the habit is soft. So here is your first week, planned to the day. Each day is twenty to forty minutes. Nothing here requires inspiration; all of it survives interruption.

Remember the three things you wished your AI could permanently remember — back in Chapter 1 you wrote them down. For nearly every operator I've asked, they're some version of: *what I do, who my clients are, and what I'm in the middle of* — and, from the schedule-driven, a fourth: *my priorities, at the open and close of every day.* Look at how precisely the week maps onto them; the fourth one gets its answer on Friday.



Monday — what you do. Open a chat the Chapter 6 way and build your first identity file together: `context/01_identity/who-we-are.md`. Talk plainly; let the AI draft; correct it where it flattens you — those corrections are the most valuable sentences in the file. If you have brand-voice opinions (words you'd never use, a tone you refuse), give them their own file. Recall Riverstone's banned-words list from Chapter 3: that file did more work than any slogan. End the day with the ritual — you know it from Station 4: **Commit to main** with a note, then **Push origin**. Every day this week ends with those two clicks.

Tuesday — who your clients are. Two files, not ten: your most important current client, and your most promising prospect, each as `context/04_clients/their-name.md`. What they hired you for, what they said when they were happiest, what they're touchy about, where things stand as of today. Plain sentences. (And keep the Chapter 5 rule in view: what they buy and what they said belongs here — account passwords and payment numbers never do. The guard is watching, but don't make it work.)

Wednesday — the thing you do every week. One workflow, written as steps: your client onboarding, your monthly invoicing, your session-prep checklist — whichever routine you'd hate to re-derive. Riverstone's roast-cycle file is the model: times, steps, what goes wrong, what each failure means. Ask your AI to interview you for it: *"Ask me questions one at a time until you can write out my onboarding process as a checklist."* That interview trick works for every file in the shell, forever.

Thursday — what you decided. Think of one real decision from the last month — a price change, a service you stopped offering, a tool you switched. Write it as `context/07_decisions/2026-08-13_short-name.md` using the decision-record template that

shipped in your shell's `templates/` folder: the situation, the decision, the reasoning, what it commits you to. One page, dated. Future-you will read this file in a year, mid-relitigation, and stop.

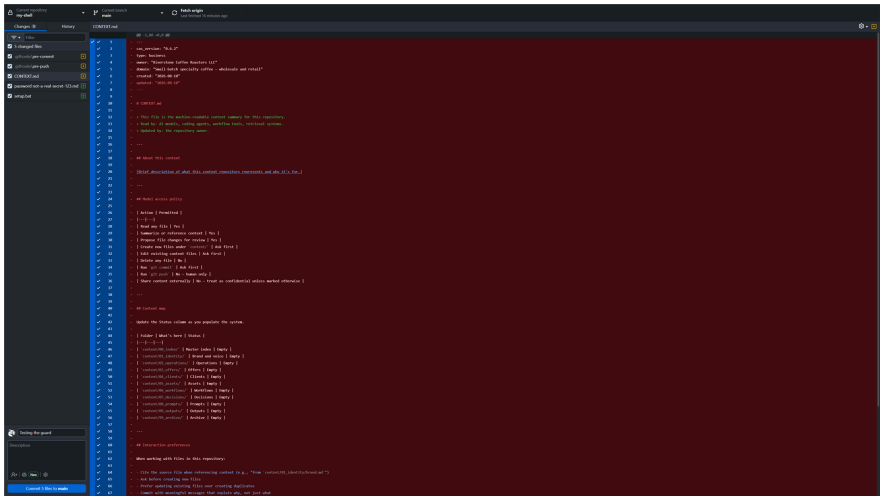
Friday — what you're in the middle of. The third wish is the one that makes Monday mornings different. Copy `templates/session-log.md` into `context/sessions/` — name it with the date, like `2026-08-14_first-week.md` — and fill in its sections for this first week: what was worked on, decisions made, what was delivered, and the one that earns its keep: **open threads** — what's unfinished and what should happen next. That section is tomorrow's starting point, written while you still know it. From now on, one of these per working session, and no AI you brief ever again asks "so where were we?" — the answer is a file. And if your Chapter 1 wish was the daily rhythm — priorities surfaced in the morning, a summary at close — notice that this file *is* that wish granted: open threads read over coffee, the log written at shutdown. First thing and last thing, attended to, every day the loop runs.

Break something on purpose

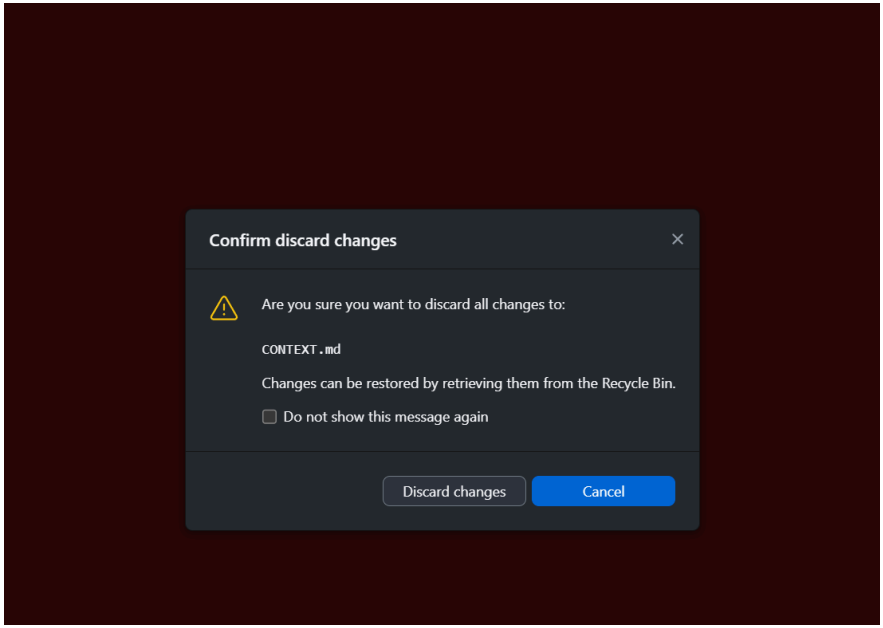
One more Friday task — thirty seconds that will pay for the whole week. It's time to *use* the undo machine, so that the safety net you've been trusting becomes one you've felt.

Open Monday's identity file in Notepad. Select everything. Delete it. Save the ruined, empty file and feel the small, authentic horror.

Now open GitHub Desktop. There's the damage in the changes panel — your file, all red.

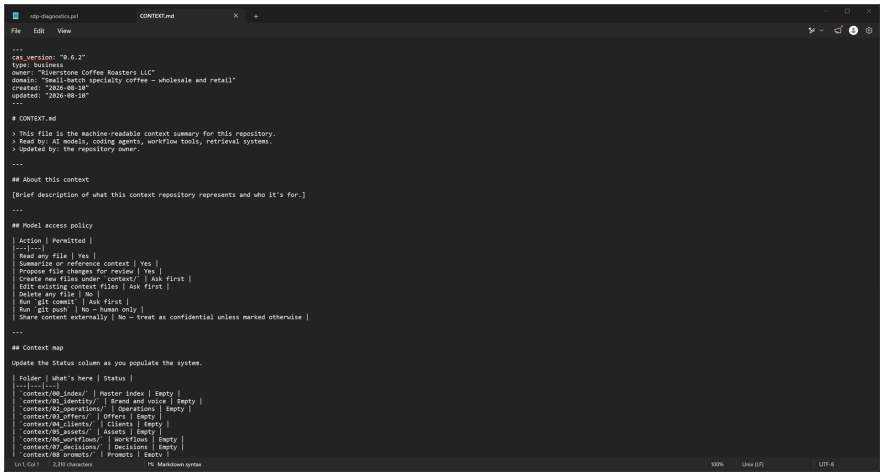


Right-click it and choose **Discard changes**, and confirm.



Open the file again.

Everything is back.



What you just did — reverting to the last snapshot — handles nearly every "oh no" you will ever produce. For the rarer, older cases, the **History** tab holds every snapshot you've pushed all week; click through them and watch your shell's whole week replay in labeled steps. Chapter 4 made a claim: *from the first snapshot on, you can only add versions, never destroy them.* You've now tested it against deliberate destruction. That fear is retired.

The loop, from here on

Zoom out and the week reveals its shape — the loop you'll run for years: **open warm** (your AI reads the shell, or your workspace already holds it) → **work** (the real work of your business, with a briefed colleague) → **log** (the session file: what happened, what's open) → **commit and push** (two clicks, snapshot forever) → and the next session opens warm again, one file smarter than the last. Around the loop, the shell only accretes. Miss a day, miss a week — nothing decays. The files wait.

One boundary, so the habit stays light: the shell is a *working file cabinet, not a diary*. If a file wouldn't help a briefed colleague do better work, it doesn't need to exist. Empty rooms are still fine. Thin files that get one honest paragraph richer each month will quietly beat any ambitious system you'd abandon by October.

Do this now: put five twenty-minute blocks on next week's calendar, labeled Mon through Fri as above. The week is designed; all it needs is the appointments.

CHAPTER 8

The Business That Remembers Itself

Count what exists now that didn't when you opened this book.

A folder, on your machine and mirrored on the internet, that states what your business knows: who you are, who you serve, how your routines run, what you decided and why, and what's mid-flight this week. A history in which nothing can be lost, which you've tested by breaking something on purpose. A guard on the door that you've watched catch a planted secret. House rules that make any AI a well-behaved colleague inside it. And the paste — one page — that briefs any model on the market in under a minute, which you've proven against two different brands.

The re-explaining tax is gone. That was the promise on the cover, and it's kept. But I told you in Chapter 1 that the AI wouldn't be the biggest thing you got out of this, so let me tell you what the second month looks like, because I've watched it from the consulting side of the table.

An artist who teaches — decades of mastery, a business run from memory and momentum — sat with me for one long discovery conversation. It went into the shell as a transcript and came back out as files: an identity document, an avatar of the one person his marketing speaks to, a map of everything he sells at every price. Nothing we generated afterward — pages, plans, posts — started from a blank chat. It started from the files, and when a draft rang false, we didn't argue with the draft; we fixed

the *file* it came from, and every later draft inherited the correction. Compounding, not repeating. That's the mechanism, and it's now installed in your business too.

Because here's what you actually built this week. The files brief your AI — that's the trick that got you here. But a business that writes down what it knows becomes *legible to its owner*. Decisions stop being vibes with amnesia. The voice stops drifting. The open threads stop living in the 3 a.m. wakeups. On the day you bring on a collaborator — human or artificial — the onboarding is a folder they read, not a month of your sentences. And on the day some distant future version of you winds the business down, or sells it, or hands it on: what you're handing over *exists*. Most businesses your size can't say that. Yours can, as of this week.

The shell you have now is a starter home, and it will grow two ways. The first is the loop from Chapter 7, running quietly for months — no further instruction needed; the files accrete on their own schedule. The second is deliberate practice: there is more system than an introduction can hold — the full discipline of decision records, the operations and account inventories that saved my clients from vendor hostage-taking, running an AI *against* a mature shell for large jobs with verification to match, and the ten principles applied as daily judgment calls rather than a list in an appendix. That practice is the subject of this book's full-length companion, **Context Architecture** — where the starter home becomes the permanent one.

Beyond that, three doors, in the order I'd walk through them:

Build with what's free. Everything this book used stays free and open: the standard, the template, the Riverstone example, the validator. Start at `contextarchitecturesystem.com` — and the

template your shell came from lives at `github.com/context-arcana-llc/cas-shell-template`, where the starter kit is maintained as the standard evolves.

Get a guide for an afternoon. If you'd rather walk the next stretch with the person who's built these for working businesses — your shell, your folders, a plan for your particular mess of heads, scrolls, and tabs — that's what I do at `contextarcana.com`: a single Roadmap Session, transparent pricing on the page, everything built in accounts you own. (You'd expect no less by now — it's Principle 8, and it's the house style.)

Or simply run the loop. Honestly the strongest option of the three. Twenty minutes a day, commit and push, and let compounding do what compounding does.

One last thing. Somewhere in your shell's history sits a snapshot from this week labeled something like *Claim the shell — fill in the manifest*. Years from now — different AI on the throne, different tools, maybe a different business than the one you're running today — that snapshot will still open, and the file inside will still say your name. That's the quiet thing you bought with an afternoon: not software, which expires, but a record, which doesn't.

Your business remembers itself now. Go give it something worth remembering.

Appendix — The Ten Principles

The Context Architecture System is governed by ten principles. You met most of them in passing through this book; here they are in one place, condensed. The full text lives in the standard's repository, linked from `contextarchitecturesystem.com`. Cite a principle by number — "CAS Principle 5" — the numbering is stable.

1. The user owns the context. The context layer belongs to whoever's work it describes — not a vendor, not a model provider, not a consultant. Every other principle follows from this one.

2. The context lives in portable files. Plain Markdown, plain folders, plain text. No proprietary format, no database, no app required to read it. If a text editor can open it, it qualifies.

3. The context is versioned with Git where possible. Git provides history, attribution, and replication without requiring a service. Where Git isn't an option, the context is still valid — the framework recommends it, never requires it.

4. The context layer outlives any model, vendor, workflow engine, or consultant. Models get replaced; vendors change pricing or shut down. The context must remain readable and usable through all of it.

5. Models may read context by default, but may not destructively edit it by default. An AI proposes; the owner

reviews and applies. Deletion, overwriting, and reorganization require explicit human instruction.

6. Every meaningful change should be committed. A change worth making is worth recording. The history is the long-term log of what changed, when, and why.

7. Secrets do not belong in context files. Keys, passwords, tokens, and identity-grade personal data live in a password manager — never in the context layer. The starter kit's guard hooks enforce this automatically.

8. Accounts remain under the user's control. Every account referenced in the context is registered in the owner's name. When a relationship with any third party ends, nothing of value leaves with them.

9. Infrastructure should be replaceable. Any tool, host, or runtime the context touches can be swapped for an equivalent without rewriting the context. Replaceability is a design feature, not an emergency exit.

10. Operational continuity is the outcome. The point is not the files — it's that work continues through model changes, vendor changes, team changes, and time. A context system that survives ten years is the goal.

Appendix — Working From an iPad

Chapter 5 assumes a Windows or Mac computer, because that's where the richest version of the toolkit lives. But some operators run their whole working life from an iPad — and one of the people this book was written with does exactly that. This appendix is the honest iPad path: what works beautifully, what works differently, and the one thing that doesn't work at all, said plainly.

The good news first: **everything essential in this book can be done from an iPad**, because your shell's safe copy lives on a website — and an iPad is excellent at websites. The difference is *where the work happens*. On a desktop, you keep a local working copy and send snapshots up. On an iPad, you work directly on the safe copy itself, in the browser. The filing cabinet and the reading room turn out to be the same room.

> **Version stamp:** verified against github.com in Safari on iPadOS, mid-2026.

Stations 1 and 2 — account and template — work as written. Safari on iPad handles `github.com/signup` and the template page the same way a desktop browser does. One wrinkle: mobile layouts sometimes tuck buttons away. If the green **Use this template** button isn't visible on the template page, you're either not signed in (the usual cause — see Station 2) or Safari is showing you the compact mobile layout: tap the **aA** (or puzzle-piece) menu in Safari's address bar and choose **Request Desktop Website**, and the full page returns.

Stations 3 and 4 — the working copy and the editing — happen in the browser instead. Skip GitHub Desktop entirely; it doesn't exist for iPadOS. Instead:

1. On your shell's page, tap `CONTEXT.md`, then tap the **pencil icon** (top right of the file view — its job is "edit this file," wherever a redesign puts it). 2. Make your Chapter 5, Station 4 edits right there — owner, domain, dates, type. 3. Tap **Commit changes**. GitHub asks for the same thing Desktop's Summary box asks for: a short note to your future self. Type it, confirm, done.

That *is* the snapshot ritual — same commit, same permanent history, one step shorter because there's nothing to push: you were already working on the safe copy. The **History** for any file (and the whole shell) is a tap away, and every restore Chapter 7 promises is available there too — every committed version of every file remains viewable and recoverable.

Station 5 — the guard — is the honest limitation. The secret-scanning guard installs into a *local* copy of the shell, and on an iPad there isn't one, so **the automatic guard does not run on this path.** No workaround in an app store changes that today. What to do instead: adopt the guard's rule as your own discipline — the shell records *which* accounts exist and who holds access; passwords, keys, and card or account numbers never enter any file — and make your AI a standing deputy: the setup prompt already orders it to flag anything that looks like a secret. If you ever borrow a desktop computer for twenty minutes, running Station 5 there installs the guard for that machine's copy; the iPad path itself stays guard-free, and knowing that is the protection.

Chapters 6 and 7 work as written, with one advantage: on an iPad, the easiest way to get files to your AI is often the best way.

The AI apps' persistent-workspace features (Chapter 6) run fine on iPad, and several AI tools can now connect directly to a GitHub account — which suits this path perfectly, since GitHub is where your working copy already lives. See the next appendix for the tool-by-tool routes.

One reframe to carry with you: the iPad path isn't the junior version. It's the same standard — the same files, the same history, the same portability — reached through a browser instead of an installed app. The shell doesn't care. That indifference is Principle 9, and it's the whole idea working as designed.

Appendix — Your AI, Three Ways: Claude, ChatGPT, Cursor

This book plays no favorites among AI tools, for a reason that is the book's own thesis: **the tools churn; your context layer doesn't.** Models leapfrog each other every season, apps rename their features, and the folder of files you built is the one piece that never has to care (Principles 4 and 9). So this appendix teaches one universal method that works in everything — then the better route in each of the three tools this book explicitly supports. The universal method is forever; the better routes are version-stamped and will drift. When they do, the durable line under each one tells you what you're looking for.

> **Version stamp:** tool features described here verified mid-2026. Feature names change; the *durable line* under each route is the part to trust.

The universal method — works in every AI, every year

1. Open a new conversation. 2. Paste the block from your shell's `SETUP_PROMPT.md`. 3. Get the relevant files to the AI: attach them, or paste their contents — `CONTEXT.md` first, then whatever today's work touches.

That's it. Any AI that can read text can run your shell this way, including ones that don't exist yet. Everything below is optimization.

Claude

Better route: Claude's persistent workspace feature (called *Projects* as of mid-2026) accepts your shell's files once and holds them across every conversation in that workspace — put the setup prompt's text in the workspace's standing instructions, add your context files, and every new chat starts already briefed. Claude can also connect to a GitHub account and read a repository directly — ideal for the iPad path, where GitHub already holds your working copy. **Durable line:** you're looking for Claude's way to keep files and standing instructions attached across conversations, and its way to connect GitHub.

ChatGPT

Better route: ChatGPT's equivalent persistent workspace (*Projects*, and custom GPTs, as of mid-2026) does the same job: upload your shell files as knowledge, put the setup prompt in the instructions, and the briefing persists. ChatGPT also offers connectors that can link cloud services — including GitHub — for direct reading. **Durable line:** you're looking for ChatGPT's way to attach knowledge files and instructions permanently, and its connectors list.

Cursor

Cursor is different in kind, and worth knowing about even if you never use it: it's a full working environment (built for programmers, usable by anyone) that opens your shell *folder itself* — no attaching, no uploading. The AI inside it reads your files in place and, under your Principle-5 rules, proposes edits to them directly. Desktop only (Windows/Mac). **Better route:** open your cloned shell folder in Cursor and work there; the CAS

toolkit can even generate a Cursor-native rules file from your manifest (the full-length companion book covers the export system). **Durable line:** Cursor-class tools are "the AI comes to your folder" instead of "your files go to the AI" — the shell works both directions without changing a single file.

The point, restated once

Notice what these three routes have in common: *your files never changed.* Same folder, same manifest, same plain text — read three different ways by three competing products. When one of them ships a redesign, raises prices, or falls behind, you re-read this appendix's durable lines, spend ten minutes, and continue. That's not a workaround. That's the architecture doing its job.

About the Author

Michael Purvis is an independent technologist and the founder of Context Arcana LLC. He is the designer of the Context Architecture System, the open standard this book teaches, and he spends his working week doing exactly what these pages describe: building and operating context systems for real businesses — artists, practitioners, consultants, and partnerships — as their implementation partner. He works from Fort Collins, Colorado, usually with an AI on the other half of the screen and the session log open.

CAS is free and open by design; Context Arcana is the consulting practice built beside it — the same relationship WordPress has to the company that grew around it. If this book built your shell, the standard costs you nothing, forever. If you want a guide, that's the practice.

Resources

Every address from this book, in one place. Electronic editions: these are clickable.

The standard

- `contextarchitecturesystem.com` — the Context Architecture System: the ten principles, the full specifications, the Riverstone Coffee example shell, and the validator.
- `github.com/context-arcana-llc/cas-shell-template` — the starter template your shell is built from (Chapter 5, Station 2).

The tools

- `github.com/signup` — create your free GitHub account (Station 1).
- `desktop.github.com` — GitHub Desktop, the point-and-click app that drives the undo machine (Station 3).
- `git-scm.com/downloads` — Git for Windows / Mac, the toolkit the secret-scanning guard runs on (Station 5).

The author

- `contextarcana.com` — implementation consulting, the Roadmap Session, and the commercial shells. Transparent pricing on the page.

The companion volume

- ***Context Architecture: The complete guide to a business that remembers itself*** — the full-length practice book: the complete build, the session discipline, decision records, running AI against a mature shell, and case studies from working businesses.