TEAM HANDBOOK · v0.3.0

Working With AI Agents

A handbook for people who don't write code

Lukáš Czerner

The current version is always at https://lczerner.github.io/agents-handbook/

Contents

Last updated: 12 August 2026

You have probably used ChatGPT or Claude in a browser. You type, it answers, you copy the answer somewhere else. That is a **chat assistant**.

This handbook is about something different: an **agent**. An agent runs on your computer, inside a directory. It can read the files in that directory, write new ones, edit existing ones, search the web, and use your other tools. You don't copy anything anywhere. You tell it what you want and it does the work in the directory, and the directory is the deliverable.

Tools that work this way include **Claude Code**, **OpenAI Codex CLI**, **opencode**, and **pi**. They are marketed at programmers. They are not only for programmers. A directory full of Markdown files is just as valid a project as a directory full of code, and everything in this handbook works the same way for an article, a campaign brief, or a content calendar.

You need one of them installed before any of this is useful. Before you start covers what to install, and how to check in a minute that you are really talking to an agent rather than to a chat window. After that, this handbook teaches the part that actually decides whether you get good work out of it: **what you put in the directory.**

Contents

Companion files in this kit:

- `WALKTHROUGHS.md` - four step-by-step exercises. Do these after reading sections 1–8.
- `PROMPTS.md` - a cheat sheet of things to type.

- `starter-kit/` - a complete example workspace, to look at when you want to see a finished one.

Before you start

Everything in this handbook assumes you are running an agent. That sounds too obvious to say. It is also the first thing people get wrong, and it costs them a week, so it is worth two minutes now.

The mistake

A walkthrough tells you to type this:

> **TYPE THIS**
>
> Create a directory called `knowledge` with subdirectories `channels`, `voice`, `entities`, and `library`.

You paste it into ChatGPT or Claude in your browser. You get a confident, well-formatted reply - a tidy list of directories, perhaps some commands you could run. Nothing has happened on your computer. Nothing will. You move on to the next step, and the kit quietly turns into a reading exercise about files that do not exist.

A browser chat window cannot reach your computer. It cannot see what is on it, cannot create anything on it, and cannot check anything it tells you about it. What it can do is describe all of that fluently, in the same confident voice it uses for everything else.

	BROWSER CHAT	AGENT
Where it runs	On the provider's servers, in a tab	On your computer, in one directory
What it can do	Produce text in the window	Read, write and edit files in that directory, run commands, search the web
Where the work ends up	You copy it out by hand	In the directory, as files
What it knows about your work	Whatever you paste in, that once	Whatever you left in the directory, every time

It is the same underlying AI in both columns. What differs is what it has been given to work with - and that is the entire subject of this handbook.

Model, harness, agent

Three words used as though they meant the same thing. Keeping them apart explains what went wrong above.

The model is the part that produces text - Claude, GPT, Gemini. On its own it does exactly one thing: text goes in, text comes out. It cannot open a file, run a command, or remember yesterday.

The harness is the program that runs on your computer around the model. It is the part with hands. It reads your files and shows them to the model, carries out what the model asks for - create this file, run this command, fetch this page - hands the result back, and goes round again until the job is done. **Claude Code, OpenAI Codex CLI, opencode and pi are harnesses.** The browser chat is one too, but a very thin one: its hands reach nothing outside the tab.

An agent is a harness and a model together, pointed at a directory on your computer.

The colleague in section 1 is the model: capable, and remembering nothing. The harness is the office they walk into - the desk, the hands, the way to the filing cabinet. What you write in your workspace is what is in that office when they arrive. You install the harness once and then never think about it again. What is in the office is written by you, and that is what everything from section 2 onwards is about.

What you need

Four things:

- **A terminal** - the text window you type commands into. You need about four commands in total, and someone can show you all four in five minutes.
- **One of the four tools, installed.** See below.
- **An account with whoever provides the model.** Usually a paid subscription. The tool walks you through logging in the first time you start it.
- **A directory to work in.** Walkthrough 1 creates it.

The four tools, checked August 2026:

Claude Code - from Anthropic. Install with `curl -fsSL https://claude.ai/install.sh | bash` on macOS or Linux, `brew install --cask claude-code` if you use Homebrew, or `irm https://claude.ai/install.ps1 | iex` in Windows PowerShell. Needs a Claude Pro, Max, Team or Enterprise subscription, or a Claude Console account. Start it by typing `claude`. Quickstart

OpenAI Codex CLI - from OpenAI. Install with `curl -fsSL https://chatgpt.com/codex/install.sh | sh`. Sign in with your ChatGPT account. Start it by typing `codex`. Quickstart

opencode - open source, and not tied to one provider: you bring an API key for whichever model you want to use. Install with `curl -fsSL https://opencode.ai/install | bash`. Start it by typing `opencode`. Docs

pi - open source, also provider-agnostic, and it can log in with a Claude Pro/Max, ChatGPT Plus/Pro or GitHub Copilot subscription you already pay for. Install with `npm install -g --ignore-scripts @earendil-works/pi-coding-agent`, which needs Node.js on your machine. Start it by typing `pi`. Quickstart

If you have no reason to prefer one, use whichever your company already pays for. Everything in this handbook works the same way in all four. Where they differ is a file name or a directory name, and Appendix A lists every difference.

Installing takes about five minutes. If those commands mean nothing to you, that is expected and it is not your job - send this section to whoever looks after laptops where you work, or ask them to sit with you for ten minutes. It is a one-time job. Install instructions also change. If a command fails, open the linked guide rather than fighting it.

One minute to check you are in the right place

Do this before Walkthrough 1. Open your terminal in any directory, start the tool, and type:

> **TYPE THIS**
>
> Create a file called `hello.md` containing one line: it worked.

If it asks permission to write the file, say yes - being asked is the agent working as intended. Then leave the terminal and open that directory in Finder on a Mac, or File Explorer on Windows.

- **`hello.md` is there.** You are running an agent. Carry on to section 1.
- **No file, but a nicely formatted answer telling you what the file would contain.** You are in a chat window. Nothing else in this handbook will work until that changes.

---

01 The one idea that makes everything else make sense

Imagine you hire a colleague who is fast, tireless, well-read and willing to do anything you ask. Every evening they **lose their entire memory**.

Every morning they arrive knowing nothing about your company, your brands, your tone of voice, what you decided last week, or what they themselves did yesterday. But they will read absolutely everything you leave on the desk before they start.

That is what an agent is. Every session starts from zero.

So the job is not "write a clever prompt." The job is **to leave the right things on the desk**. Everything in this handbook is a different kind of thing to leave on the desk:

WHAT YOU LEAVE	WHAT IT IS
House rules	A file called `AGENTS.md` that it reads every single time
Reference material	A directory of notes about your brands, products, people, style
A plan and a logbook	So a long job survives across many days
Written procedures	Step-by-step recipes it picks up when relevant ("skills")
Keys to other systems	Connections to Drive, WebOps, your CMS, analytics ("MCP")

Two consequences follow, and they surprise everyone at first:

Consequence 1: writing things down is the work. Time spent writing down your style rules is not overhead before the real work. It *is* the real work. It compounds - write it once, every future task benefits.

Consequence 2: a bad output is usually a missing file, not a stupid agent. When the agent writes something off-brand, the useful question is not "how do I re-prompt this" but "what did it not know, and where should that live so it never has to ask again?"

> THE DESK HAS A LIMITED SIZE.
>
> The agent can only hold so much in its head at once - this is called the **context window**. Think of it as the desk surface. Files on the shelf (your directory) are unlimited; what's on the desk right now is not. When a job is long, the desk fills up and older things get pushed off. Section 5 is entirely about working around this.

---

02 The five things you give an agent

Everything you will ever set up falls into one of five buckets. Learn these five names and you can stop guessing.

1. Rules - `AGENTS.md` Loaded every session, no exceptions. Short. This is who we are, what we never do, and how work gets done here. *Section 4.*

2. Knowledge - a `knowledge/` **directory** Facts the agent cannot guess: your websites, your voice, your products, your people, your past articles. Read on demand, when relevant. *Section 6.*

3. Plan and progress - `PLAN.md` **and** `PROGRESS.md` For any job bigger than one sitting. The plan is what we agreed to do. The progress log is what has actually happened. *Section 5.*

4. Procedures - skills (`SKILL.md` **)** "Here is exactly how we produce a launch announcement, in nine steps." The agent picks these up automatically when a task matches. *Section 7.*

5. Reach - MCP servers Connections to systems outside the directory: Google Drive, WebOps, Slack, your CMS, analytics. *Section 8.*

A useful way to hold it in your head:

```
AGENTS.md   = the employee handbook   → always read
knowledge/  = the filing cabinet      → read when relevant
skills      = the procedure manuals   → opened when the task matches
MCP         = keys to the building     → lets it leave the directory
PLAN.md     = this project's brief
PROGRESS.md = this project's logbook
```

You do not need all five on day one. **Start with AGENTS.md and two knowledge files.** That alone gets you 70% of the value. Add the rest when you feel the specific pain each one solves.

---

03 Your workspace

A workspace is just a directory on your computer. You open a terminal in it and start the agent there. Everything the agent does happens inside it.

Here is a layout that works for a content and campaigns team. Copy it, then delete what you don't need.

```
my-content-workspace/
│
├── AGENTS.md              ← house rules. The agent reads this every session.
├── CLAUDE.md              ← one line: @AGENTS.md   (only needed for Claude Code —
see Appendix A)
│
├── knowledge/             ← the filing cabinet
│   ├── INDEX.md           ← a map of what's in here and when to read it
│   ├── channels/          ← one file per website / newsletter / social channel
│   │   ├── lighthouse-blog.md
│   │   ├── the-signal-newsletter.md
│   │   └── linkedin.md
│   ├── voice/             ← how we sound
│   │   ├── house-voice.md
│   │   └── author-jana-novak.md
│   ├── entities/          ← things we make claims about
│   │   ├── products.md
│   │   ├── people.md
│   │   └── events.md
│   └── library/           ← examples of our own past work
│       ├── GOLD-STANDARD.md
│       └── articles/
│
├── projects/              ← one directory per job. This is where work happens.
│   └── 2026-09-atlas-launch/
│       ├── BRIEF.md       ← what we were asked for
│       ├── PLAN.md        ← what we agreed to do, in phases
│       ├── PROGRESS.md    ← logbook: what's done, what's next
│       └── drafts/
│
└── .claude/skills/        ← procedure manuals (directory name depends on tool —
Appendix A)
    ├── article-draft/SKILL.md
    ├── campaign-plan/SKILL.md
    └── style-check/SKILL.md
```

Three rules about the workspace:

Keep one workspace per team, not per person. The whole point is that the knowledge is shared. If everyone has their own private directory, you're back to everyone having their own private prompt tricks.

Put it somewhere it gets backed up and shared. A shared Drive/Dropbox directory is fine to start. If someone technical can put it in Git, that is better - you get a full history of who changed which rule and when, and you can undo mistakes. Don't let this block you. A synced directory is enough on day one.

Never put secrets in it. No passwords, no API keys, no customer personal data. Assume everything in the directory may be read by the agent and sent to the model provider. If you wouldn't paste it into a chat window, it doesn't go in the directory.

---

04 AGENTS.md - the house rules

`AGENTS.md` is a plain text file (Markdown) in the root of your workspace. Every supported agent reads it at the start of every session, before it does anything else. No other file changes the output as much.

There is no required format. No special syntax. No fields you must fill in. It's a memo to a new colleague. Headings and bullet points, because those are easier to follow than paragraphs - for the agent as much as for a human.

> NAMING NOTE.
>
> `AGENTS.md` is an open standard, originally published by OpenAI and now maintained under the Linux Foundation's Agentic AI Foundation. Codex, opencode, pi, Cursor, Copilot, Gemini CLI and others read it directly. **Claude Code reads `CLAUDE.md` instead** - so you create a second file, `CLAUDE.md`, containing the single line `@AGENTS.md`, and now both work from the same source. See Appendix A.

4.1 What goes in it

Seven sections. In this order.

1. WHAT THIS PROJECT IS

Two or three sentences. What the workspace is for, who the company is, what the agent is helping with.

```
## What this is

Lighthouse is a B2B software company selling project-tracking tools to
mid-size construction firms. This workspace is where our two-person content
team plans and writes everything we publish: the blog, the weekly newsletter,
LinkedIn, and campaign materials for product launches.

You are helping us research, plan, draft and edit. You are not publishing
anything — a human always does that.
```

That last sentence does a lot of work. Say what the agent is *not* doing as clearly as what it is.

2. WHO WE'RE WRITING FOR

The agent will otherwise write for "a general business audience," which reads like nothing anyone wrote on purpose.

```
## Who we write for
```

```
Primary reader: an operations manager at a construction firm with 50–500
employees. Time-poor, skeptical of software vendors, has been burned by a
failed rollout before. They are not technical. They care about whether their
site foremen will actually use a tool.

They are not the buyer of last resort – they usually have to convince a
finance director. Give them arguments they can forward.
```

3. WHERE THINGS LIVE

A short map. This is what lets the agent find your knowledge base without you naming files every time.

```
## Where things live

- `knowledge/INDEX.md` – start here; it lists everything below
- `knowledge/channels/` – one file per website/channel: audience, formats, rules
- `knowledge/voice/` – house voice, plus per-author voices
- `knowledge/entities/` – approved facts about our products, people, events
- `knowledge/library/` – our own past work, including gold-standard examples
- `projects/<date>-<name>/` – active work. Each has BRIEF, PLAN, PROGRESS.

Before writing anything for a specific channel, read that channel's file in
`knowledge/channels/` and the relevant file in `knowledge/voice/`.
```

4. GUARDRAILS

The most important section. Three lists: never, always, ask first.

Be specific enough that a person could check whether the rule was followed. "Write well" is unverifiable and therefore useless. "No sentence over 25 words" is checkable.

```
## Guardrails

### Never
- Never invent a statistic, a customer name, a quote, or a case study.
  If you need a number and don't have a source, write `[NEEDS SOURCE: what
  you need]` and keep going.
- Never state a product capability that isn't in `knowledge/entities/products.md`.
  That file is the only source of truth for what our product does.
- Never name a competitor in published copy.
- Never publish, post, send, or schedule anything. Draft only.
- Never use the words: "leverage", "seamless", "game-changing", "in today's
  fast-paced world", "delve", "it's not just X, it's Y".
- Never use em dashes. Use commas or full stops.

### Always
- Always write in British English.
- Always cite a source with a link for any claim about the industry, and
  include the date the source was published.
- Always save work to a file in the project directory. Don't print a long draft
  into the chat and stop there.
- Always end a draft with a short "Open questions" list of anything you
  guessed at.

### Ask me first
- Before starting to write, if the brief is missing the audience, the channel,
```

```
  or the desired length.
- Before restructuring an existing published article.
- Before using any statistic that would go in a headline.
- Before deleting or overwriting any file in `knowledge/` or `library/`.
```

Three things to notice:

- **The `[NEEDS SOURCE: ...]` convention.** Give the agent a legal way to not know something. Without one, its only options are to stop or to make something up, and it will often choose the second. A placeholder is a rule it can actually follow.
- **Word bans are worth it.** Every team has ten words that instantly read as AI-written. List yours. This single bullet will save you more editing time than anything else in the file.
- **"Ask me first" is a real category.** It's how you stay in control of the moments that matter without micromanaging every step.

5. HOW WORK GETS DONE HERE

Your default workflow. What the agent should do when you give it a task, without being told.

```
## How we work

For anything longer than a social post:

1. Read the brief and the relevant channel + voice files.
2. Write a plan to `PLAN.md` and stop. Do not start drafting. Wait for me
   to approve it.
3. Work through the plan one phase at a time. After each phase, update
   `PROGRESS.md` and tell me what changed.
4. Never do more than one phase without checking in.

For research tasks: collect sources into a `research/` file with links and
publication dates first; summarise second. Never summarise from memory.
```

6. WHAT "DONE" LOOKS LIKE

The agent's idea of finished is not yours unless you say so.

```
## Definition of done

A draft is done when:
- It has a headline plus two alternatives.
- It has a meta description under 155 characters.
- Every factual claim has a source link or a [NEEDS SOURCE] marker.
- It passes the checks in `knowledge/voice/house-voice.md`.
- It is saved as `projects/<project>/drafts/<slug>.md`.
- Open questions are listed at the bottom.
```

7. HOW TO TALK TO ME

Small section, big quality-of-life improvement.

```
## How to talk to me

- Be direct. Skip the preamble, skip "Great question!", skip summarising
  what I just said back to me.
```

```
- When you finish a task, tell me in three lines: what you did, what file
  it's in, what needs my decision.
- If you disagree with my instruction, say so once, briefly, then do what
  I asked.
- Internal notes and our conversation: English. Anything for publication:
  British English unless the channel file says otherwise.
```

4.2 The rules about the rules

Keep it under about 200 lines. This file is loaded into the agent's head every single time, competing for desk space with the actual work. A 900-line `AGENTS.md` makes the agent *less* likely to follow any given rule, not more. If a section is growing, move it into `knowledge/` and leave a pointer.

Never contradict yourself. If one line says "keep it short" and another says "aim for 2,000 words," the agent picks one arbitrarily and you'll never know which. Re-read the whole file after every edit.

Only write what it can't work out for itself. Don't describe your directory structure in detail - it can see the directories. Write down the things that exist only in your head: the preferences, the past mistakes, the reasons.

It's a living file. The rule of thumb: *the second time you correct the same thing, it goes in `AGENTS.md`*. First time is a one-off. Second time is a pattern, and a pattern belongs in the file. You can just say: "*Add that to AGENTS.md so you don't do it again*" and the agent will edit the file itself.

It's guidance, not a lock. This is important to understand honestly: `AGENTS.md` shapes behaviour, it does not enforce it. The agent reads it and tries to comply. Clear, specific, non-contradictory rules get followed reliably. Vague or buried ones sometimes don't. For anything where a mistake would be expensive - publishing, sending, deleting - don't rely on a written rule alone. Rely on the fact that you review before anything goes out.

4.3 The fastest way to write your first one

Don't write it from a blank page. Have the agent interview you:

> **TYPE THIS**
>
> I want to create an AGENTS.md for this workspace. Don't write it yet. First interview me: ask me one question at a time, up to fifteen questions, about what we publish, who reads it, what our rules are, what mistakes you should avoid, and what "done" looks like. When you have enough, show me a draft AGENTS.md and I'll correct it.

Twenty minutes of answering questions gets you a better file than two hours of staring at an empty document. Then edit it by hand - it's yours, not the agent's.

---

05 Plans, phases, and progress

5.1 The problem

Ask an agent for something big - "plan our Q4 launch campaign" - and one of two things happens.

Either it produces a shallow, generic version of everything at once, because it tried to hold the entire job in its head. Or it starts well, works for twenty minutes, and then quietly loses the thread: it forgets a decision you made earlier, contradicts its own outline, repeats a section.

This is the desk filling up. The context window is finite. Long jobs overflow it.

There is also the human version of the same problem: you close your laptop on Tuesday, come back Thursday, and the agent has no idea any of it ever happened.

5.2 The fix: plan → phases → logbook

Three files, one discipline.

`BRIEF.md` - what we were asked for. Written by you, once, at the start. The raw request, deadline, audience, constraints, what success looks like.

`PLAN.md` - what we agreed to do, broken into phases. Written by the agent, approved by you, changed rarely.

`PROGRESS.md` - what has actually happened. Updated by the agent at the end of every work session. This is the file that lets a new session pick up exactly where the last one stopped.

The discipline: **one phase per session.** Then stop, update the log, and start a fresh session for the next phase. A fresh session with a good logbook beats a tired session with a full desk, every time.

5.3 Making the plan

Step one is always: **ask for the plan, and explicitly forbid the work.**

> **TYPE THIS**
>
> Read `BRIEF.md`, `knowledge/INDEX.md`, and the channel files for blog and newsletter. Then write a plan to `PLAN.md`.
>
> Do not write any campaign content yet. The plan only.
>
> Break it into 4–7 phases. Each phase must have: a goal in one sentence, the inputs you need, the files you'll produce, and how I'll know it's done. Phases must be small enough that one is a single sitting of work.
>
> At the end, list anything you're unsure about or had to assume.

Then **read the plan and change it.** This is your main point of control over the whole project, and it costs five minutes. If the plan is wrong, everything downstream is wrong, and you'll spend far longer fixing drafts than you would have spent fixing the plan.

A good phase list for a campaign looks something like:

```
## Phase 1 — Research and positioning
Goal: Establish what we're claiming and why anyone should believe it.
Inputs: BRIEF.md, knowledge/entities/products.md, competitor sites
Outputs: research/positioning.md with 3 candidate angles, evidence for each
Done when: Lukas has picked one angle and it's marked CHOSEN in the file.

## Phase 2 — Channel plan and calendar
...
```

Notice that Phase 1's "done" condition includes **a human decision**. Build those in deliberately. They're your checkpoints.

5.4 The logbook

`PROGRESS.md` is the single most underrated file in this handbook. It is what turns a series of disconnected sessions into a project.

Put this in `AGENTS.md` so it happens automatically:

```
## Progress logging

At the end of every working session, and after finishing any phase, update
`PROGRESS.md` in the current project directory. Keep it in this format, newest
entry at the top, and keep the whole file under 100 lines by summarising
older entries:

## Status
Current phase: <number and name>
Next action: <the single next thing to do>
Blocked on: <what you need from a human, or "nothing">

## Decisions made
- <date> — <decision> — <why>

## Log
### <date> — Phase <n>
Did: ...
Produced: <file paths>
Learned: <anything that changed our understanding>
Open questions: ...
```

The **Decisions made** section matters more than it looks. Halfway through a project someone asks "why are we leading with the cost angle?" and the answer is written down with a date, instead of lost in a chat window nobody can search.

5.5 Starting and ending a session

Two short rituals. Type these until they're muscle memory.

Starting:

> **TYPE THIS**
>
> Read `PROGRESS.md` and `PLAN.md` in `projects/2026-09-atlas-launch/`. Tell me in five lines where we are and what the next action is. Don't start work yet.

Ending:

> **TYPE THIS**
>
> Stop here. Update `PROGRESS.md`: what you did, what files changed, what decisions we made and why, and the single next action for next time. Write it so someone who wasn't here today could pick it up cold.

That last sentence is the trick. "Someone who wasn't here today" is, in fact, the agent tomorrow.

5.6 When the desk fills up mid-session

Agents will tell you when they're compacting or summarizing, or you'll notice the quality drop - it forgets a decision, repeats itself, contradicts the outline. When that happens: **don't push through.** Say:

> **TYPE THIS**
>
> Update PROGRESS.md with where we are, then I'm starting a fresh session.

Then start a new session (in Claude Code, `/clear`). Reading a clean logbook is far more reliable than remembering a long conversation.

---

06 The knowledge base

`AGENTS.md` is what the agent reads *every* time, so it has to stay short. The knowledge base is everything else - read only when relevant. This is where you can be generous with detail.

Four categories.

6.1 Channels - one file per place you publish

Every website, newsletter, and social channel gets its own file. This is what stops the agent writing the same beige paragraph for your technical blog and your Instagram.

`knowledge/channels/lighthouse-blog.md`:

```
# Channel: Lighthouse Blog (lighthouse.com/blog)

## Purpose
Organic search acquisition. Every post targets a keyword a construction ops
manager would actually type. This is not a company news channel.

## Reader
```

```
Ops manager, 50–500 person construction firm. Arrives from Google with a
specific problem. Skims first, reads second. Often on a phone, on site.

## Format
- 1,200–1,800 words
- H2 every 250–350 words, sentence case
- Answer the title question in the first 100 words. Do not build up to it.
- One table or checklist minimum
- Ends with a single CTA to the relevant product page

## Rules
- Target keyword in title, first paragraph, and one H2. Nowhere else forced.
- Meta description under 155 characters, written as a promise not a summary.
- Internal-link to 2–3 existing posts (see knowledge/library/articles/)
- No stock-photo clichés in image briefs. Describe a real jobsite scene.

## Never on this channel
- Product announcements (those go to the newsletter)
- First-person company voice ("we're excited to...")

## Good examples
- knowledge/library/articles/why-site-diaries-fail.md
- knowledge/library/articles/rfi-turnaround-benchmarks.md
```

Write one of these for each channel. It takes twenty minutes each and it is the difference between "the agent writes okay copy" and "the agent writes copy that fits."

6.2 Voice - how we sound

Separate from channel, because voice is often shared across channels and authors are not.

The trick to a voice guide that actually works: **contrast pairs.** Abstract adjectives ("confident, warm, human") mean nothing to an agent and, if we're honest, not much to a new copywriter either. Do this-not that pairs mean everything.

`knowledge/voice/house-voice.md`:

```
# House voice

## In one line
We sound like an experienced site manager explaining something to a
colleague over coffee. Direct, specific, slightly dry. Never a vendor.

## Rules
- Second person. "You" not "companies" or "organisations".
- Active voice. Name who does the thing.
- Sentences under 25 words. Vary the length or it reads like a manual.
- Concrete numbers over vague scale. "cut RFI turnaround from 9 days to 3"
  not "dramatically improves efficiency".
- One idea per paragraph. Three sentences maximum.
- British English.

## Do this, not that
✗
   "Leveraging our innovative platform, teams can seamlessly streamline
   their documentation workflows."
```

```
✅ "Your site team files an RFI from their phone. The office sees it the
   same minute."

❌ "In today's fast-paced construction environment, staying ahead is
✅  more important than ever."
   "Most delays don't start on site. They start in an inbox."

❌ "Our solution empowers stakeholders to optimise outcomes."
✅ "Foremen stop chasing paperwork. Project managers stop chasing foremen."

## Banned words and phrases
leverage · seamless · game-changing · robust · delve · in today's
fast-paced world · it's not just X, it's Y · unlock · elevate · journey
(unless literal) · revolutionary · cutting-edge

## Punctuation
- No em dashes. Comma or full stop.
- No exclamation marks in body copy.
- Serial comma: no.
```

Then per-author files for anything published under a byline:

`knowledge/voice/author-jana-novak.md`:

```
# Author voice: Jana Novák, Head of Operations

## Who she is
15 years in construction project management before joining Lighthouse.
Writes from experience, not from research. This is her main credibility.

## How she writes
- Opens with a specific thing that happened, not a general claim.
- Uses "I" and tells stories from sites she worked on.
- Skeptical of software claims, including ours. Will name trade-offs.
- Short paragraphs. Occasional one-line paragraph for emphasis.
- Never uses statistics she hasn't personally checked.

## Phrases she actually uses
"In practice, that means..." · "The honest version is..." ·
"That worked on paper."

## Never in her voice
- Marketing superlatives of any kind
- Third-person corporate ("Lighthouse believes...")
- Anything that sounds like it came from a product page
```

A POWERFUL SHORTCUT:

if you already have twenty good articles, don't write the voice guide from scratch. Put them in `knowledge/library/articles/` and ask: "*Read every article in this directory. Derive our voice guide: sentence patterns, structure, vocabulary we use and avoid, how we open and close. Include at least ten do-this-not-that pairs taken from real sentences in these articles. Save to* `knowledge/voice/house-voice.md`." Then edit what it produces. It will find patterns you didn't know you had.

For even better results, have the finished text rewritten with Hapax MCP.

6.3 Entities - the facts you're allowed to state

This is your defense against confident invention. If it's not in these files, the agent is not permitted to claim it.

`knowledge/entities/products.md`:

```
# Products — the only source of truth for what we sell

If a capability is not listed here, we do not claim it. If asked about
something not here, write [NEEDS PRODUCT INPUT: question] and continue.

---

## Atlas
**One-liner:** Site documentation that works without signal.
**Launched:** March 2024 · **Current version:** 4.2 (June 2026)
**Price:** From €39/user/month, annual. Minimum 10 users.

### What it does
- Photo and note capture on site, offline, syncs when back in range
- RFI creation and tracking with automatic reminders
- Daily site diary generated from the day's captures
- Exports to PDF and to Procore, Autodesk Build

### What it does NOT do — do not imply otherwise
- No scheduling or Gantt functionality
- No cost or budget tracking
- No BIM model viewing
- Android and iOS only. There is no desktop app for site use.

### Approved claims (use verbatim or close)
- "Works with no signal and syncs later."
- "Average RFI turnaround across our customers dropped from 9 days to 3."
    — source: internal 2026 customer study, n=41. Always say "across our
  customers", never "in the industry".

### Claims we must NOT make
- Anything about ROI in a specific timeframe. Legal has said no.
- "Fastest" or "most-used" anything. Unsubstantiated.
-   Any comparison naming a competitor.

### Boilerplate (press releases, footer)
Atlas is Lighthouse's site documentation tool for construction teams
working in low-connectivity environments. [50 words, approved by legal
2026-04-02, do not edit without asking Legal.]
```

Do the same for **people** (name spelling, exact job title, bio at three lengths, what they're allowed to be quoted on) and **events** (dates, venue, registration link, official name and its exact capitalization, key messages, what's not announced yet).

6.4 Library - your own past work

Two things live here, and they do different jobs.

Gold standard examples. Three to five pieces per format that you'd be happy to see repeated. Not your whole archive - your *best*. In `GOLD-STANDARD.md`, say what each one gets right:

```
# Gold standard

## Blog: why-site-diaries-fail.md
Why it works: opens with a specific failure, not a definition. The table in
the middle is the thing people screenshot. Ends without a hard sell.
Copy: the structure and the willingness to say what doesn't work.

## Newsletter: 2026-05-14-the-signal.md
Why it works: one idea, 400 words, a clear opinion. No round-ups.
Copy: the opinion-first structure. Don't copy the jokes.
```

The archive. Everything else you've published. This is what lets the agent internal-link properly, avoid re-covering the same ground, and notice that you already said the opposite of this in 2024. If your CMS can export to Markdown, export everything. If it can't, even a `catalogue.md` with title, URL, date, topic and a one-line summary per article is worth having. Ideally, you can give your agent access to an MCP server with your archive, if you have it.

6.5 Making the knowledge base findable

Two habits keep it working as it grows.

Write an `INDEX.md`. One line per file saying what's in it and *when to read it*. The agent reads the index cheaply, then opens only what it needs.

```
# Knowledge index

Read the file that matches your task. Don't read everything.

## Channels — read before writing for that channel
- `channels/lighthouse-blog.md` — SEO blog. Format, keyword rules, examples.
- `channels/the-signal-newsletter.md` — weekly newsletter. Opinion-led, 400w.
- `channels/linkedin.md` — company page + founder profile. Different rules.

## Voice — read alongside the channel file
- `voice/house-voice.md` — applies to everything. Read this one always.
- `voice/author-jana-novak.md` — only for pieces bylined Jana.

## Entities — read before making any factual claim
- `entities/products.md` — THE source of truth on capabilities and claims.
- `entities/people.md` — names, titles, pronouns, bios, quote permissions.
- `entities/events.md` — dates, venues, official names, embargo status.

## Library
- `library/GOLD-STANDARD.md` — our best work and why it works.
- `library/articles/` — full archive. Search here before proposing a topic.
```

Date everything and mark what's uncertain. A fact without a date rots silently. Put `Last verified: 2026-06-30` at the top of every entity file, and make a habit of it. If something is provisional, say so in the file: `Status: not yet announced, do not reference before 15 September.`

---

07 Skills - teaching it your procedures

7.1 What a skill is

A skill is a directory containing a file called `SKILL.md`. Inside is a step-by-step procedure for one kind of task, written in plain language.

The clever part is **how it loads**. At startup the agent reads only the *name and description* of each skill - a couple of lines each, costing almost nothing. When your request matches a description, it opens the full file and follows it. Everything else stays on the shelf.

The bookshelf analogy: `AGENTS.md` is pinned to the wall and always in view. Skills are procedure manuals on a shelf. The agent reads the spines constantly and only takes one down when the job calls for it. This is why you can have thirty detailed skills without slowing anything down, but you cannot have a thirty-page `AGENTS.md`.

Skills are an open standard (originally from Anthropic, now developed in the open) supported by Claude Code, Codex, opencode, pi, Cursor, Copilot, Gemini CLI, and a couple of dozen other tools. The directory they live in differs per tool - see Appendix A.

7.2 When to make one

Make a skill when:

- You've explained the same multi-step process three times.
- A section of `AGENTS.md` has turned from a *fact* into a *procedure*.
- You want a process to run identically no matter who asks for it.
- You want to type `/campaign-plan` and have it just happen.

Don't make a skill for a one-off, and don't make one for a simple fact - that's a knowledge file.

The clean test: **a fact goes in `knowledge/`. A rule goes in `AGENTS.md`. A sequence of steps goes in a skill.**

7.3 The format

```
.claude/skills/
└── article-draft/
    ├── SKILL.md          ← required: two lines of metadata + the procedure
    ├── scripts/          ← optional: small programs the agent can run
    │   └── check-copy.py
    ├── references/       ← optional: longer material, loaded only if needed
    │   └── seo-checklist.md
```

```
    └── assets/           ← optional: templates
        └── outline-template.md
```

`SKILL.md` needs exactly two things at the top, between `---` lines:

```
---
name: article-draft
description: Produce a publish-ready article draft from a brief, following our
channel and voice rules. Use when asked to write, draft, or outline a blog post,
article, or newsletter piece.
---
```

Rules for those two fields:

- `name` - lowercase letters, numbers and hyphens only, max 64 characters, and it **must match the directory name**.
- `description` - max 1024 characters. This is the only thing the agent sees until it decides to open the skill, so it has to say **what it does and when to use it**, using the words you would actually type. "Helps with articles" will never trigger. The version above will.

Then the body: the procedure. Keep it under about 500 lines. Push long reference material into `references/` files that the skill points to.

THE THREE OPTIONAL DIRECTORIES

A skill can be a single `SKILL.md` file. The directories are there for when written instructions alone can't carry everything.

`scripts/` - small programs the agent can run. This is the one people don't expect to need, and then can't work without.

Language models are unreliable at mechanical checks. Ask one whether a meta description is under 155 characters and it will tell you 148 when it's actually 163. It isn't being careless - counting simply isn't what it does. A four-line script counts correctly every time, and counts the same way on Monday as on Friday.

So put in `scripts/` anything that is a **check** rather than a **judgement**:

- Title and meta description length, character-exact
- Banned words and phrases from your voice file, so none get missed
- Sentences over your word limit, listed with line numbers
- Required fields present on every article: title, slug, date, author, category
- Internal links pointing at files that don't exist
- Filenames and slugs matching your convention
- A CSV or JSON export being valid before you hand it to anyone

Then the skill just says when to run it:

```
## Phase 6 — Self-review
First run `scripts/check-copy.py drafts/<slug>.md` and fix everything it
reports. Then do the judgement pass: read the draft against
```

```
`knowledge/voice/house-voice.md` and the channel file, and produce a
checklist with   /   per rule.
```

That division is the whole point. **The script checks what's countable. You and the agent judge what isn't.** "No sentence over 25 words" belongs in a script. "Does this sound like us" never will.

You don't have to write the script yourself. Describe the check and ask:

> **TYPE THIS**
>
> Write `scripts/check-copy.py` for the `article-draft` skill. It takes a markdown file path and reports: every sentence over 25 words with its line number, every banned word from `knowledge/voice/house-voice.md`, the meta description length, and any sentence containing a number but no link. Print a plain list and exit with an error if anything failed. Standard library only, nothing to install.

Two cautions. A script is real code, so it can be wrong in ways prose can't - read what it reports for a week before you trust it silently. And whether a skill's scripts actually run depends on your tool and your permission settings. If yours never seems to run them, that's the first thing to check.

`references/` - material too long to sit in the instructions. An SEO checklist, a legal wording guide, the full brand book. The agent opens these only when the task needs them, so length costs you nothing until it does.

`assets/` - fixed files the skill uses. Outline templates, an approved boilerplate paragraph, a spreadsheet layout. Things to be used as they are, rather than read for guidance.

7.4 A real one

`.claude/skills/article-draft/SKILL.md`:

```
---
name: article-draft
description: Produce a publish-ready article draft from a brief, following our
channel and voice rules. Use when asked to write, draft, or outline a blog post,
article, or newsletter piece.
---

# Article draft

Work through these phases in order. Stop at each ⏸ and wait for me.

## Phase 1 — Load context
1. Read the brief.
2. Read `knowledge/channels/<the target channel>.md`.
3. Read `knowledge/voice/house-voice.md`, plus the author file if bylined.
4. Search `knowledge/library/articles/` for anything we've published on
   this topic. List what you found and how this piece differs.

If the brief doesn't name a channel, a target reader, or a length: ask.
Don't guess. ⏸
```

```
## Phase 2 — Angle
Propose three angles. For each: the promise to the reader in one sentence,
who it's for, why we're credible on it, and what evidence we'd need.
Recommend one and say why. ⏸ Wait for me to choose.

## Phase 3 — Evidence
Collect what the piece needs: sources with links and publication dates,
approved product claims from `knowledge/entities/products.md`, internal
link targets.
Save to `research.md` in the project directory.
Anything you can't source: `[NEEDS SOURCE: ...]`. Never fill a gap with a
plausible-sounding number. ⏸

## Phase 4 — Outline
H2s with one line on what each section does for the reader, plus the
opening paragraph written in full — the opening sets the voice and is the
cheapest thing to fix now. ⏸

## Phase 5 — Draft
Write it. Save to `drafts/<slug>.md`. Follow the channel file's format
rules exactly.

## Phase 6 — Self-review
Re-read your own draft against `knowledge/voice/house-voice.md` and the
channel file. Produce a checklist with ✅ / ❌ per rule. Fix every ❌ and
say what you changed. Be genuinely critical — a review that finds nothing
is a review you didn't do.

Check specifically: banned words, sentences over 25 words, paragraphs over
three sentences, passive voice, unsourced claims, em dashes.

## Phase 7 — Package
Add at the top of the file: three headline options, meta description
(<155 characters), suggested slug, two internal links, one image brief.
Add at the bottom: open questions and every [NEEDS SOURCE] marker,
collected in one list.

Then tell me: what's done, where it is, what you need from me.
```

Phase 6 deserves a note. **Asking an agent to criticize its own work against a written checklist works** - much better than asking it to "write well" in the first place. Generating and evaluating are different jobs, and separating them into different phases produces better results than trying to do both at once. Build a self-review phase into every skill you write. Better still, use a specialized sub-agent with a clean context window as the reviewer.

7.5 Skills worth building first

In rough order of payoff for a content team:

1. **`article-draft`** - the one above.
2. **`style-check`** - takes any text and audits it against your voice files, line by line, with fixes. Run it on anything, including things a human wrote.
3. **`campaign-plan`** - brief in, multi-channel plan out: messaging spine, channel calendar, asset list, owners, measurement.

4. `repurpose` - one article into a newsletter, five LinkedIn posts, and an Instagram carousel, each in that channel's voice, no lazy copy-paste.
5. `brief-intake` - interrogates a vague request until it's a real brief. Refuses to proceed on "write something about the launch."
6. `weekly-roundup` - the recurring thing your team does every Monday.

A good way to create the first one: do the task manually with the agent once, paying attention to every correction you make. Then say: "*Turn everything we just did into a skill at* `<skills directory>/article-draft/SKILL.md`*, including every correction I made along the way.*" Your skills directory depends on the tool - the four are in Appendix A.

---

08 MCP - giving it reach into other systems

8.1 What it is

By default the agent can only see the directory it's running in, plus the web. **MCP** (Model Context Protocol) is the standard that lets it reach into other systems: Google Drive, WebOps, Slack, your CMS, your analytics, your project tracker.

You install a small connector called an **MCP server** - one per system - and the agent gains a set of new abilities: *search Drive, read this WebOps page, post to this Slack channel, pull last month's GA4 numbers.*

It's an open standard, donated by Anthropic to the Linux Foundation in December 2025 and now supported by essentially every major agent tool, with tens of thousands of connectors available. You will not usually build one. You install existing ones.

Practically, MCP replaces the export-and-paste step. Instead of downloading a CSV, tidying it, and pasting it in, you say "*pull last month's blog traffic and tell me which three posts to update.*"

8.2 What's worth connecting for a content team

Start with two. Seriously - two. Every connector adds tools the agent must consider on every request, and a stack of fifteen makes it slower and less accurate, not more capable.

CONNECTOR	WHAT IT GIVES YOU
Google Drive / Workspace	Read briefs, transcripts, decks and sheets your team already keeps there
WebOps or Confluence	If your knowledge lives there instead of in files
Slack	Read a channel for context; post drafts for review
Analytics (GA4) / Search Console	"Which posts are decaying?" answered with real numbers
SEO tool (Ahrefs, Semrush)	Keyword and competitor research without tab-switching
CRM (HubSpot, Salesforce)	Real customer language for campaign copy

CONNECTOR	WHAT IT GIVES YOU
Your CMS	Pull published articles into the library; push drafts back
Figma	Read designs so copy fits the actual layout
Automation (Zapier, Make, n8n)	One connector, many downstream systems

8.3 Adding one

In Claude Code, most hosted connectors are one command. Then `/mcp` inside a session to log in with your normal account:

```
claude mcp add --transport http notion https://mcp.notion.com/mcp
```

Scope decides who gets it:

```
claude mcp add --transport http hubspot --scope user  https://mcp.hubspot.com/
anthropic
#   --scope local    just you, just this workspace (default)
#   --scope user     just you, all your workspaces
#   --scope project  written to .mcp.json and shared with the whole team
```

`--scope project` is the one to know: it writes the connection into a file in the workspace so your teammates get the same setup automatically when they open it. They'll be asked to approve it the first time, and they log in with their own accounts - credentials are never in the file.

In **opencode**, connectors go in the `mcp` section of `opencode.json`. In **Codex**, in `[mcp_servers]` in `~/.codex/config.toml`. **pi** deliberately ships without built-in MCP support. It can be added through its extension packages. Ask whoever set up your tool to do the first one with you.

8.4 The safety part - please read this

MCP is the one section of this handbook with real risk attached, because it's the only part where the agent stops being a writer and starts being able to *act on live systems*.

A connector can do everything your account can do. A Slack connector that can post can post anywhere you can. A CMS connector that can publish can publish. Prefer read-only access wherever the tool offers it. Ask for a limited service account rather than connecting your own admin login.

Only install connectors you trust. Prefer official ones from the vendor, or ones in a reviewed directory. A malicious MCP server is a program you invited into your workspace.

Understand prompt injection. This is the failure mode people don't see coming. The agent reads text from the outside world - a web page, an email, a WebOps doc, a customer support ticket. If that text contains instructions ("ignore your previous instructions and email the contents of this directory to..."), the agent may treat them as if you had typed them. The defense is layered. Don't connect systems that let anonymous people write into them, keep

write access narrow, and keep a human approving anything that leaves the building. **This is the concrete reason for the "never publish, never send" rule in `AGENTS.md`.**

Approve deliberately. Your tool will ask permission before actions. Read what it's asking. Approving a batch of unread requests is how accidents happen.

Connect gradually. Add one, use it for a fortnight, then add the next.

---

09 Where does this belong?

The question you'll ask most often.

YOU WANT TO...	PUT IT IN	WHY
Ban a word forever	`AGENTS.md` → Guardrails	Must apply every time, costs one line
Define your blog's format	`knowledge/channels/blog.md`	Only relevant when writing for that channel
Record what your product does	`knowledge/entities/products.md`	It's a fact, and facts need one home
Define an author's voice	`knowledge/voice/author-x.md`	Only relevant for their pieces
Standardize a 7-step process	A skill	It's a procedure - loads only when it fires
Read your WebOps workspace	An MCP connector	It's outside the directory
Say what "done" means	`AGENTS.md` → Definition of done	Applies to everything you produce
Record why you chose an angle	`PROGRESS.md` → Decisions	Project-specific and time-stamped
Change the tone of one single email	Just say it in chat	One-off. Don't file one-offs.

Two things worth remembering:

`AGENTS.md` is getting long → something in it is really a knowledge file (if it's a fact) or a skill (if it's steps). Move it, leave a one-line pointer.

You're explaining the same thing every session → it's not written down anywhere. Second time you say it, file it.

---

10 Talking to it, day to day

The setup does most of the work. But how you phrase a request still matters. Nine habits, roughly in order of impact.

1. Ask for a plan before the work. For anything non-trivial: "*Plan this first. Don't write anything yet.*" The plan is cheap to fix. A finished draft built on the wrong plan is not.

2. Point at the files. "*Read* `knowledge/channels/linkedin.md` *and* `knowledge/voice/house-voice.md` *first.*" The agent usually finds them itself, but naming them is free and removes all doubt.

3. Say where the output goes. "*Save to* `projects/atlas-launch/drafts/announcement.md`." Otherwise you get a wall of text in the terminal that you have to copy somewhere - and you've lost the main advantage of working this way.

4. Edit in place, don't regenerate. Once a draft exists, say "*in the draft file, tighten section 3 and cut the last paragraph.*" Never "*here's the draft again, rewrite it.*" Iterating on the file keeps everything else stable.

5. Give it a role and a constraint, not just a task. "*You're the skeptical finance director this proposal has to get past. Read the draft and list every claim you'd challenge.*" Role plus constraint beats a bare instruction almost every time.

6. Make it check its own work. "*Now audit that draft against* `house-voice.md`*, rule by rule, with* ✅ / ❌ *and a fix for each* ❌*.*" Separating writing from reviewing is the cheapest quality upgrade available.

7. Ask for options, then choose. "*Give me three openings with different angles, one line each on why it works.*" You get better material and you stay the editor.

8. Correct once, then file it. After a correction: "*Add that to* `AGENTS.md` *so it holds from now on.*" Corrections that don't get filed will be needed again on Thursday.

9. Start fresh sessions often. New topic, new session. A long conversation carries all the earlier turns as clutter. Update the logbook, clear, continue.

And one thing that isn't a habit but a rule:

Verify anything a number, name, date or quote depends on. Agents are extremely good at producing text that has the shape of a fact. The guardrails in section 4 exist to reduce this, and they help a lot, but they don't eliminate it. Any statistic, any quote, any date, any spelling of a person's name that reaches a published page: check it against the source yourself. Your `[NEEDS SOURCE]` markers tell you where to look first.

---

11 When things go wrong

SYMPTOM	WHAT'S ACTUALLY HAPPENING	FIX
Ignores a rule in `AGENTS.md`	The rule is vague, buried in a 600-line file, or contradicted elsewhere	Make it specific and checkable. Cut the file down. Look for the contradiction - there usually is one.
Confidently invents facts	It had no source and no permitted way to say "I don't know"	Add the `[NEEDS SOURCE: ...]` rule. Put the real facts in `knowledge/entities/`. Say "only claims in that file are permitted."

SYMPTOM	WHAT'S ACTUALLY HAPPENING	FIX
Forgot everything from yesterday	Normal. Every session starts empty.	`PROGRESS.md`, and the start-of-session ritual in §5.5.
Went off and did far too much	No approval gate in the instruction	"Plan first, don't write." "One phase, then stop." Put both in `AGENTS.md`.
Output drifted mid-session	Desk full - context overflowed	Update the logbook, start a fresh session, continue. Don't push on.
Sounds like generic AI	It's writing from its general training, not your voice	Voice file with do/don't pairs, banned word list, and a self-review phase. All three.
Won't stop being enthusiastic	Default assistant register	"How to talk to me" section in `AGENTS.md`. Be blunt: no preamble, no flattery.
Edited a file you didn't want touched	It had permission and no instruction not to	"Ask before touching anything in `knowledge/`." Keep the workspace in Git so anything is undoable.
Rewrites the whole draft on a small ask	You asked for a rewrite without meaning to	"Edit the existing file. Change only section 3. Leave everything else."
Slow, or picks odd tools	Too many MCP connectors loaded	Turn off what you're not using this week.
Two rules conflict and it picks wrong	Genuinely ambiguous instructions	Fix the source. If two rules can both be true, the agent's choice is a coin flip.

A general debugging move: **just ask it.** "*You didn't follow the rule about sentence length. Look at `AGENTS.md` and `house-voice.md` and tell me why that rule might have been unclear or contradicted.*" It is often right about what confused it, and the answer tells you what to rewrite.

---

12 Rolling this out to a team

Week 1 - one person, one workspace. One person sets up `AGENTS.md`, two channel files, and one voice file. They use it for real work for a week. Nothing else. Resist building the whole system before you know which parts you need.

Week 2 - the first skill. Take the task you did most often last week and turn it into a skill. Do the task with the agent, then ask it to write the skill from what you just did together.

Week 3 - bring in the team. Share the workspace. Everyone uses the same `AGENTS.md`. Book 30 minutes at the end of the week: what did it get wrong, and which file should have prevented it? Edit the files together. This meeting is the whole rollout - it's where the system actually gets built.

Week 4 - one connector. Add the single MCP connector that removes the most copy-paste from your week. Just one.

Then, ongoing:

- **One owner for `AGENTS.md`.** Not a committee. Anyone can propose, one person edits. Otherwise it grows contradictions. Repository owner on GitHub.
- **Version it.** Git if you can, a synced directory if you can't. You want to be able to see what changed when quality drops.
- **Review the knowledge.** Stale facts are worse than missing facts, because the agent states them with total confidence. Date-stamp everything and check the dates.
- **Write down what you learn about the tool itself.** Which prompts worked, which failed. That's a knowledge file too.

A Which file does my tool read?

All four tools work with the same workspace. They differ in file names and directory locations. *(Checked August 2026. These things move - if something doesn't load, check your tool's current docs.)*

Instruction file

TOOL	READS	WHERE IT LOOKS
Claude Code	`CLAUDE.md`	Project root or `.claude/CLAUDE.md`; also `~/.claude/CLAUDE.md` for personal rules. **Does not read `AGENTS.md` directly.**
Codex CLI	`AGENTS.md`	`~/.codex/AGENTS.md` globally, then every directory from the repo root down to where you are, concatenated. `AGENTS.override.md` takes precedence in a directory. 32 KB cap by default.
opencode	`AGENTS.md`	Project root, walking up; then `~/.config/opencode/AGENTS.md`. Falls back to `CLAUDE.md` if there's no `AGENTS.md`.
pi	`AGENTS.md` or `CLAUDE.md`	`~/.pi/agent/AGENTS.md` globally, then parent directories, then the current one. `AGENTS.override.md` wins in a directory.

To make one file serve all four: write your real content in `AGENTS.md`, then create `CLAUDE.md` next to it containing one line:

```
@AGENTS.md
```

That's Claude Code's import syntax - it pulls in the whole file. You can add Claude-specific notes underneath. (A symlink works too: `ln -s AGENTS.md CLAUDE.md` - but not on Windows without developer mode, so the import line is the safer choice.)

A subdirectory can have its own `AGENTS.md` - useful if one client or brand needs different rules. What happens then depends on the tool. **Codex** and **pi** read every directory from the top down to the one you started the agent in, so the file in the subdirectory applies on top of the root one. **opencode** stops at the first file it finds walking up, so the file in the subdirect-

ory replaces the root one instead of adding to it. **Claude Code** reads a nested `CLAUDE.md` when it opens a file in that subdirectory, but never a nested `AGENTS.md`.

Skills directory

TOOL	PROJECT SKILLS	PERSONAL SKILLS
Claude Code	`.claude/skills/<name>/SKILL.md`	`~/.claude/skills/<name>/SKILL.md`
Codex CLI	`.agents/skills/<name>/SKILL.md`	`~/.agents/skills/<name>/SKILL.md`
opencode	`.opencode/skills/<name>/SKILL.md`	`~/.config/opencode/skills/<name>/SKILL.md`
pi	`.pi/skills/` or `.agents/skills/`	`~/.pi/agent/skills/` or `~/.agents/skills/`

The `SKILL.md` file itself is identical across all of them - same format, same two required fields. Only the directory differs. If your team uses more than one tool, keep the skills in one directory and create links to the others, or just copy them.

To run one on demand: `/skill-name` in Claude Code, `$skill-name` in Codex. opencode has no command for it - ask for the skill by name and the agent loads it itself. Or say nothing and let the description trigger it.

MCP connectors

TOOL	HOW
Claude Code	`claude mcp add --transport http <name> <url>`, or `.mcp.json` in the project for team-wide. `/mcp` in a session to log in and manage.
Codex CLI	`[mcp_servers]` section in `~/.codex/config.toml`
opencode	`mcp` section in `opencode.json` (project) or `~/.config/opencode/opencode.json` (global)
pi	No built-in MCP. Available through extension packages.

B Glossary

Agent - an AI that runs in a directory on your computer and can read, write and edit files and use tools, rather than only chatting.

AGENTS.md - the instructions file every agent reads at the start of every session. The open standard for this. Claude Code uses `CLAUDE.md` instead.

Chat assistant - ChatGPT or Claude in a browser tab. The same model, with no access to your computer. It produces text you copy out by hand. Not an agent - see Before you start.

Context window - how much the agent can hold in mind at once. The desk surface. Finite, and the reason long jobs need a logbook.

Compacting - what happens when the desk fills up: the tool summarizes the earlier conversation to make room. Detail is lost. A signal to save your progress and start fresh.

Harness - the program that runs on your computer around the model and gives it hands: it reads your files, carries out what the model asks for, and repeats until the job is done. Claude Code, Codex CLI, opencode and pi are harnesses.

Markdown - plain text with `#` for headings and `–` for bullets. What all these files are written in. That's the whole syntax you need.

MCP (Model Context Protocol) - the standard that lets an agent reach systems outside the directory: Drive, WebOps, Slack, your CMS.

MCP server / connector - one such connection. You install it once.

Model - the part that produces the text: Claude, GPT, Gemini. Text in, text out, and nothing else. Everything it can do to your files, it does through the harness.

Prompt injection - when text the agent reads from the outside world contains instructions, and the agent follows them as if you'd typed them. The reason to keep a human between the agent and anything that publishes or sends.

Session - one continuous conversation. Ends when you close the tool or clear it. The next one starts with no memory of it.

Skill - a directory with a `SKILL.md` inside: a written procedure the agent picks up when the task matches its description.

Progressive disclosure - the mechanism behind skills. The agent reads only names and descriptions until something matches, then loads the full file. Why you can have many skills cheaply.

Repository / repo - a directory tracked by Git, so every change is recorded and undoable. Nice to have, not required.

Terminal - the text window you type commands into. You need about four commands total. Someone will show you.

Where to learn more

Five things from outside this kit, in the order that makes most sense to take them in.

- Large Language Models explained briefly - what a model is actually doing when it answers you: predicting the next word, over and over, from the text in front of it. This is why the colleague in section 1 remembers nothing about yesterday.
- Agent Harness explained in 8min.. - the program around the model: the part that hands it your files, runs the tools it asks for, and decides when the job is done. Claude Code, Codex, opencode and pi are all this. The same distinction, in one page, is in Before you start.
- What AI Agent Skills Are and How They Work - what goes inside a `SKILL.md` and how the agent decides to load it. Read alongside section 7.

- MCP vs Skills: Which Is Right for Your AI Agent and LLMs? - the same question as section 9, answered by someone else.
- Best practices for Claude Code - Anthropic's own guide. Written for programmers, but most of it is not about code: keeping the context window clear, planning before doing, and giving the agent a way to check its own work.

Where to go next

1. Read `WALKTHROUGHS.md` and do **Walkthrough 1**. It takes an hour and produces your real `AGENTS.md`.
2. Keep `PROMPTS.md` open for the first fortnight.
3. Look at `starter-kit/` once you have your own workspace, and take a file from it when you find you need that file. Don't copy it over what you built - every answer in it is Lighthouse's.

The whole system is plain text files in a directory. Nothing here is fragile, nothing is hidden, and anything you break you can fix by editing a file. Start with `AGENTS.md` and two knowledge files, use it for real work, and add the rest when you feel the need for it.

> **THESE ARE EXERCISES, NOT A METHOD.**
>
> This is not a guide to writing articles or running marketing campaigns. They are examples, and they may have nothing to do with your job. The point is to let you get a feel for how an agent behaves, and to show you some of what can be customized.
>
> Nothing here is off-limits. Ideally, swap the brief for something you actually want to produce - but keep the idea of the exercise intact.

Do these in order. Everything in `> blockquote` is meant to be typed to the agent, more or less as written - adapt the names to your own work. Everything in a `code block` is a file or a terminal command.

#	WALKTHROUGH	TIME	WHAT YOU END UP WITH
1	Set up your workspace	60 min	A real `AGENTS.md` and four knowledge files
2	Write an article	90 min	A publish-ready draft, and a voice guide derived from your own work
3	Plan a campaign	3 sessions over 2 days	A full multi-channel campaign plan, built across sessions
4	Turn it into a skill	30 min	A reusable procedure you invoke with one word

Before you start, you need an agent. Not ChatGPT or Claude in a browser tab - those cannot create a single file on your computer, and every exercise here ends in a file. Claude Code, OpenAI Codex CLI, opencode or pi, installed, with a terminal you can open in a directory and run it from. If that sentence is doing a lot of work, read Before you start in the handbook and get whoever set it up to sit with you for the first ten minutes of Walkthrough 1.

W1 Set up your workspace

Goal: a directory with house rules the agent follows, and enough knowledge to write something that sounds like you. **Time:** about an hour.

Step 1 - Make the directory

Make a new directory somewhere sensible - `Documents/content-workspace` is fine. Open your terminal in it and start your agent (`claude`, `codex`, `opencode`, or `pi`).

Then, to check it can actually see and write files:

> **TYPE THIS**
>
> Create a directory called `knowledge` with subdirectories `channels`, `voice`, `entities`, and `library`. Also create a directory called `projects`. Then show me the directory structure.

You should see the directories appear on disk. If it asks permission to create files, say yes. **This is the whole trick of working with agents: you are not asking for text, you are asking for changes to a directory.**

If nothing appeared on disk, stop here. An answer describing the directories is not the same as the directories existing. Check in Finder or File Explorer, not in the reply. If they are not there, you are not running an agent - go back to Before you start.

Step 2 - Let it interview you

Don't write `AGENTS.md` from a blank page. Type this:

> **TYPE THIS**
>
> I want to create an `AGENTS.md` for this workspace. This is where our team will plan and write everything we publish.
>
> Don't write the file yet. First, interview me. Ask me **one question at a time**, up to about fifteen questions, covering:
>
> - what our company does and who we sell to
> - what we publish and where
> - who reads it
> - our writing rules and the words we hate
> - mistakes you could make that would embarrass us
> - what you should never do without asking
> - what a finished piece of work looks like
>
> Wait for my answer before each next question. When you have enough, say so and stop.

Then answer honestly, in short sentences. Don't try to sound polished - this is raw input, not a document.

Where people get this wrong: answering vaguely. "We're professional but friendly" is worthless. "We sound like a site manager explaining something to a colleague, never like a vendor" is worth an hour of editing later. When it asks about words you hate, actually list ten.

Step 3 - Get the draft, then take it over

> **TYPE THIS**
>
> Now write `AGENTS.md` based on my answers. Use these sections in this order: What this is · Who we write for · Where things live · Guardrails (Never / Always / Ask me first) · How we work · Definition of done · How to talk to me.
>
> Keep it under 150 lines. Make every rule specific enough that I could check whether you followed it - no "write well", no "be professional".

Now **read it yourself and edit it by hand.** Open it in any text editor. This is your file, not the agent's. Three things to check:

1. **Is every rule checkable?** Delete or sharpen anything you couldn't verify with a highlighter.
2. **Does anything contradict anything else?** "Keep it concise" plus "aim for 1,800 words" is a coin flip. Pick one.
3. **Is the "Never" list honest?** Add the things that would actually get you in trouble: inventing statistics, naming competitors, claiming a product does something it doesn't, publishing anything at all.

Make sure these two lines are in there, whatever else is:

```
- Never invent a statistic, quote, customer name, or case study. If you need
  one and don't have a source, write [NEEDS SOURCE: what you need] and continue.
- Never publish, post, send, or schedule anything. You draft, a human ships.
```

Step 4 - If you use Claude Code, add one more file

Claude Code reads `CLAUDE.md`, not `AGENTS.md`. Ask for it:

> **TYPE THIS**
>
> Create a `CLAUDE.md` in the project root containing exactly one line: `@AGENTS.md`

Now both work from the same source and you never maintain two files. (Codex, opencode and pi read `AGENTS.md` directly - nothing else to do.)

Step 5 - Your first channel file

Pick the one channel you publish to most.

> **TYPE THIS**
>
> Interview me about our blog, one question at a time, so you can write a channel file. Ask about: its purpose, who arrives there and why, format and length, structure rules, SEO rules, what we never do on this channel, and which past pieces are good examples.
>
> Then write it to `knowledge/channels/blog.md`.

Step 6 - Your voice file

If you already have published work, use it - this produces a far better result than describing your voice from memory:

> **TYPE THIS**
>
> I'm going to put five of our best published articles in `knowledge/library/articles/`. Read all of them, then write `knowledge/voice/house-voice.md` containing:
>
> - our voice in one sentence
> - concrete rules (sentence length, person, tense, paragraph length)
> - **at least ten "do this, not that" pairs using real sentences from those articles**
> - a banned words list of anything that would sound wrong in our voice
> - punctuation conventions you can see us following
>
> Base it only on what's actually in the articles. Don't invent rules that sound good.

(Copy the articles into that directory first - as `.md` or `.txt` files. If your CMS exports HTML, that's fine too, just say so.)

If you have nothing published yet, have it interview you instead, and insist on the do/don't pairs.

Step 7 - Your product fact sheet

The voice file keeps the agent from sounding wrong. This one keeps it from making things up about your product.

> **TYPE THIS**
>
> Interview me about our main product, one question at a time, then write `knowledge/entities/products.md`. Ask about: what it does, **what it explicitly does not do**, the claims we're allowed to make and where each one comes from, the claims we must never make, pricing, and our approved boilerplate description.
>
> Put `Last verified: <today's date>` at the top of the file.

The list of what the product does not do is the part that earns its place. Ask about something your files don't cover and the agent fills the gap with something that sounds right. It reads exactly as well as a true sentence. That is why nobody catches it.

Walkthrough 2 tells the agent to use no product claim that isn't in this file. Until the file exists, that rule has nothing to check against.

Step 8 - Write the index

Four files hardly need an index. Write one anyway - you write it now, or you write it when there are thirty of them and you no longer remember which holds what.

> **TYPE THIS**
>
> Write `knowledge/INDEX.md`. One line per file: what's in it, and **when you should read it**. Keep the whole thing under a screen.

The *when to read it* half is what does the work. The agent reads the index first and opens only what it needs, instead of reading the whole directory every time.

Add a line to it whenever you add a file to `knowledge/`.

Step 9 - Prove the whole thing works

Fresh session (in Claude Code, type `/clear`). Then:

> **TYPE THIS**
>
> Read `AGENTS.md` and everything in `knowledge/`. Then write me a 150-word LinkedIn post announcing that we've published a new guide about [any topic you actually cover].
>
> After you write it, audit your own post against `knowledge/voice/house-voice.md`, rule by rule, with ✅ or ❌ and a fix for each ❌.

Read the output. It won't be perfect. That's the point - **whatever is wrong with it tells you exactly which file is missing something.**

Then close the loop:

> **TYPE THIS**
>
> The tone is off in the second paragraph - it sounds like a press release, and we never do that. Add a rule to the right file so this doesn't happen again, and tell me which file you chose and why.

✅ **Done when:** you have `AGENTS.md`, one channel file, a voice file, a product fact sheet, an index, and you've watched the agent follow a rule you wrote.

W2 Write an article

Goal: a publish-ready draft, produced in phases with you as the editor at each gate. **Time:** about 90 minutes. **Needs:** Walkthrough 1 finished.

Step 1 - Set up the project

> **TYPE THIS**
>
> Create `projects/2026-08-guide-article/` with a `drafts/` subdirectory. In it, create `BRIEF.md` and interview me to fill it in: what we're writing, for which channel, target reader, the one thing they should take away, target length, deadline, and what success looks like.

A brief is the cheapest thing to get right and the most expensive thing to get wrong. Five minutes here.

Step 2 - Angles, not a draft

> **TYPE THIS**
>
> Read `BRIEF.md`, the channel file, and `knowledge/voice/house-voice.md`.
>
> If there's anything in `knowledge/library/articles/`, search it for what we've already published on this topic and tell me what you found. If that directory is empty, say so and move on.
>
> Then give me **three angles** for this piece. For each: the promise to the reader in one sentence, why we're credible on it, and what evidence we'd need to make it stand up.
>
> Do not write the article. Recommend one angle and say why.

Pick one. Push back if none are right - "*None of these. The interesting thing is X. Give me three angles on that.*" Two rounds here beats fixing a finished draft.

Step 3 - Evidence before prose

> **TYPE THIS**
>
> Angle 2, please. Now collect the evidence.
>
> Search the web for supporting data. For every source: the link, the publication date, and the exact claim it supports. Prefer primary sources - original research, official statistics, named studies. Skip anything you can't date.
>
> Pull the approved product claims from `knowledge/entities/products.md`. Do not use any product claim that isn't in that file.
>
> Save it all to `research.md` in the project directory. Anything we need but can't source, list as `[NEEDS SOURCE: ...]`.

Then - and this is the step people skip - **open `research.md` and click two of the links.** You are checking whether its sources are real and say what it claims. Do this every time for the first month. It calibrates your trust, and it shows you exactly where the weak points are.

Step 4 - Outline

> **TYPE THIS**
>
> Write an outline: H2 headings, one line under each on what that section does for the reader, plus **the opening paragraph written out in full**.
>
> Follow the structure rules in the channel file. Don't draft the rest.

The opening in full is deliberate. It's where voice lives, and it costs nothing to redo now. If the opening is right, the draft usually follows.

Step 5 - Draft

> **TYPE THIS**
>
> Good. Write the full draft to `drafts/<slug>.md`, following the outline and the channel format rules.

Then leave it alone. Don't read it yet - the next step will fix half of what you'd have commented on.

Step 6 - Make it review itself

> **TYPE THIS**
>
> Now audit your own draft against `knowledge/voice/house-voice.md` and the channel file.
>
> Produce a table: each rule, ✅ or ❌, and the specific fix for each ❌. Check especially: banned words, sentences over 25 words, paragraphs over three sentences, passive voice, em dashes, and any claim without a source.
>
> Be genuinely critical. A review that finds nothing is a review you didn't do.
>
> Then apply every fix to the file and tell me what you changed.

This step consistently produces a bigger improvement than any amount of re-prompting. Generating and judging are different jobs. Doing them separately works better than doing them at once.

Step 7 - Now you read it

Open the draft. Edit as an editor, not a prompter - ask for surgical changes, in place:

> **TYPE THIS**
>
> In the draft file: cut the third section entirely, it repeats section one. Tighten the intro to four sentences. Leave everything else exactly as it is.

Note "leave everything else exactly as it is." Without it you sometimes get a full rewrite and lose the parts you liked.

Then a harder pass:

> **TYPE THIS**
>
> Read the draft as our most skeptical reader - an ops manager who's been sold bad software before. List every sentence they'd roll their eyes at, and why. Don't fix anything yet.

Step 8 - Package it

> **TYPE THIS**
>
> Add to the top of the draft file: three headline options, a meta description under 155 characters, a suggested slug, and two internal links from `knowledge/library/articles/` if there's anything there to link to.
>
> Add at the bottom: an image brief, and every `[NEEDS SOURCE]` marker collected into one list.
>
> Then tell me in three lines: what's done, where it is, what you need from me.

Step 9 - Bank the learning

> **TYPE THIS**
>
> Which corrections did I make today that you should have got right the first time? For each, tell me which file should have prevented it, and make the edit.

Do this after every piece for the first month. It is how the workspace gets good.

✅ **Done when:** a draft exists in the directory, every source is checked, and at least one rule got added to a file because of something that went wrong today.

W3 Plan a media campaign

Goal: a full multi-channel campaign plan, built across three separate sessions, proving the plan/phase/logbook system. **Time:** three sessions of ~45 minutes, ideally on different days. **Needs:** Walkthroughs 1–2.

This one is deliberately spread over days. The point is to watch the agent pick up cold from a logbook - the skill that makes long projects possible.

Session 1 - Brief and plan

Step 1. Set up:

> **TYPE THIS**
>
> Create `projects/2026-09-atlas-launch/` with `drafts/` and `research/` subdirectories.
>
> Create `BRIEF.md` and interview me one question at a time: what we're launching, the date, the audience, the one message, which channels, budget, who's involved, what success looks like in numbers, and any hard constraints (embargoes, legal, approvals).

Step 2. Get a plan and nothing else:

> **TYPE THIS**
>
> Read `BRIEF.md`, `knowledge/INDEX.md`, and the channel file for every channel in the brief that has one. Tell me which channels in the brief have no channel file yet.
>
> Write a phased plan to `PLAN.md`. **Do not produce any campaign content.**
>
> 5–7 phases. For each: goal in one sentence, inputs needed, files produced, and how we'll know it's done. Every phase must be small enough to finish in one sitting. At least two phases must end with a decision I have to make.
>
> At the end, list your assumptions and anything the brief doesn't answer.

Step 3. Read the plan properly. This is your one big steering moment. Typical corrections:

> **TYPE THIS**
>
> Three changes. Phase 2 is too big - split it into messaging and calendar. Move the press list to Phase 1, we need it earlier for embargoes. And add a phase for the sales enablement one-pager, I forgot it in the brief. Update `PLAN.md` and `BRIEF.md`.

Step 4. Do exactly one phase:

> **TYPE THIS**
>
> Now do Phase 1 only. Stop when it's done.

Step 5. Close the session - this step is the whole exercise:

> **TYPE THIS**
>
> Stop here. Create `PROGRESS.md` with:
>
> **Status** - current phase, the single next action, anything you're blocked on **Decisions made** - dated, with the reason for each **Log** - what you did today, which files you produced, what you learned, open questions
>
> Write it so someone who wasn't here today could pick this up cold with no other context.

Then close the tool.

Session 2 - The cold start

Step 6. New day, fresh session. Type only this:

> **TYPE THIS**
>
> Read `PROGRESS.md` and `PLAN.md` in `projects/2026-09-atlas-launch/`. Tell me in five lines where we are and what the next action is. Don't start work yet.

Watch what happens. It will tell you exactly where you left off, including why you made the decisions you made. **This is the moment the system clicks.** No re-explaining, no scrolling back through a chat.

If it's confused or vague, that's diagnostic: `PROGRESS.md` wasn't specific enough. Fix the log format now, while it's obvious what was missing.

Step 7. Continue:

> **TYPE THIS**
>
> Correct. Do Phase 2 only, then stop.

Step 8. Same closing ritual. Every time:

> **TYPE THIS**
>
> Stop here. Update `PROGRESS.md` - status, next action, decisions with reasons, and today's log entry. Keep the whole file under 100 lines by summarizing older entries.

Session 3 - Finish and package

Step 9. Cold start again, then work through the remaining phases - still one at a time, still logging at the end of each.

Step 10. When the plan is complete:

> **TYPE THIS**
>
> The plan is finished. Produce `CAMPAIGN.md` in the project root: the full campaign on one page - messaging spine, channel-by-channel calendar with dates, the asset list with file paths and owners, dependencies and deadlines, and how we'll measure it.
>
> Then list everything still blocked on a human, and who needs to do what.

Step 11. Stress-test it before anyone else sees it:

> **TYPE THIS**
>
> Now be our CFO reading this for the first time. What are the three weakest points? What would you refuse to approve and why?

Step 12. Bank it:

> **TYPE THIS**
>
> What did we learn about how we plan campaigns that isn't written down anywhere? Propose additions to `AGENTS.md` or a new knowledge file. Show me the changes before making them.

✅ **Done when:** you've had at least one cold start that worked, and `CAMPAIGN.md` is something you'd actually send to a colleague.

W4 Turn a repeated task into a skill

Goal: the process from Walkthrough 2 becomes something you invoke with one word, that works identically for everyone on the team. **Time:** 30 minutes. **Needs:** Walkthrough 2 - you need to have done the task manually once.

Step 1 - Find the right directory

Depends on your tool (see Appendix A of the handbook):

TOOL	DIRECTORY
Claude Code	`.claude/skills/`
Codex CLI	`.agents/skills/`
opencode	`.opencode/skills/`
pi	`.pi/skills/` or `.agents/skills/`

The prompt in the next step says `<skills directory>`. Put your row from this table there before you send it.

Step 2 - Have it write the skill from what you actually did

> **TYPE THIS**
>
> Look back at how we produced the article in `projects/2026-08-guide-article/`, including every correction I made along the way.
>
> Turn that into a skill at `<skills directory>/article-draft/SKILL.md`.
>
> Requirements:
>
> - YAML frontmatter with `name: article-draft` and a `description` that says what it does **and when to use it**, using words I'd actually type - write, draft, article, blog post, outline, newsletter.
> - The body is the procedure, in numbered phases.
> - Mark with ⏸ every point where you must stop and wait for me.
> - Include the self-review phase where you audit your own draft against the voice file with ✅ / ❌ and fix every ❌.
> - Include every correction I made during the walkthrough as an explicit rule.
> - Under 200 lines. Push anything long into `references/`.

Step 3 - Check the description

Open `SKILL.md` and look at the `description` line. It is the only part the agent sees until it decides to open the skill, so it has to contain the words you'd naturally use.

❌ `description: Helps with writing articles.` ✅ `description: Produce a publish-ready article draft from a brief, following our channel and voice rules. Use when asked to write, draft, or outline a blog post, article, or newsletter piece.`

Also check: `name` is lowercase-with-hyphens, and it **matches the directory name exactly**. That's the most common reason a skill silently doesn't load.

Step 4 - Test both ways

Directly - fresh session, then type `/article-draft` (Claude Code) or `$article-draft` (Codex). opencode has no command for it: type `use the article-draft skill` instead. It should start at Phase 1.

Automatically - fresh session, then type something natural:

> **TYPE THIS**
>
> I need a blog post about how site diaries actually get filled in.

If the skill was written well, it starts following the procedure without being told. If it doesn't, the `description` is the problem - rewrite it with the words you just used.

Step 5 - Now build the rest

Same pattern, in order of payoff:

- **`style-check`** - audits any text against your voice files, line by line, with fixes. Run it on human-written copy too.
- **`campaign-plan`** - the process from Walkthrough 3.
- **`repurpose`** - one article into a newsletter, five LinkedIn posts and an Instagram carousel, each in its channel's voice.
- **`brief-intake`** - refuses to proceed on a vague request until it's a real brief.

Never write a skill from imagination. **Do the task manually once, note every correction, then have the agent write the skill from what happened.** Skills written from a real session describe how you actually work, including the corrections - and those corrections are most of the value.

✅ **Done when:** you type one word and a multi-step process runs the way you'd run it yourself.

What to do after all four

You now have the full system. The habit that keeps it working is a single question, asked at the end of every piece of work:

> **TYPE THIS**
>
> What did I correct today that you should have known already - and which file should have told you?

Ask it every time for a month. The workspace will teach itself your job.

If you want to see a finished one

`starter-kit/` is a complete workspace belonging to a fictional company called Lighthouse. Every file in it is filled in, including the ones these exercises never asked you to write: a people file with pronouns and quote permissions, an events file with an embargo status, a list of gold-standard pieces, a second voice file for a named author.

Read it for the shape of those files, and take one across when you find you need it. Don't copy the directory over the workspace you just built - every answer in it is Lighthouse's, not yours.

Keep this open for your first fortnight. Copy, paste, adapt.

> **STARTING POINTS, NOT BATTLE-TESTED PROMPTS.**
> These are simple examples to get you going. Change them to fit your work and keep whatever works best for you.

Every session

Start of a session on an existing project

> **TYPE THIS**
> Read `PROGRESS.md` and `PLAN.md` in `projects/<name>/`. Tell me in five lines where we are and what the next action is. Don't start work yet.

End of every session - never skip this

> **TYPE THIS**
> Stop here. Update `PROGRESS.md`: status, the single next action, decisions made today with the reason for each, and today's log entry with the files you produced. Write it so someone who wasn't here today could pick it up cold.

When quality drops mid-session *(it repeats itself, forgets a decision, contradicts the outline)*

> **TYPE THIS**
> Update `PROGRESS.md` with where we are. I'm starting a fresh session.

Then clear the session (`/clear` in Claude Code) and start again.

Getting set up

Write your AGENTS.md

> **TYPE THIS**
> I want to create an `AGENTS.md` for this workspace. Don't write it yet. First interview me, one question at a time, up to fifteen questions, about what we publish, who reads it, our rules, the mistakes you could make that would embarrass us, what you should never do without asking, and what "done" looks like. Wait for my answer before each next question.

Derive a voice guide from your own published work

> **TYPE THIS**
>
> Read every article in `knowledge/library/articles/`. Write `knowledge/voice/house-voice.md`: our voice in one sentence, concrete rules, at least ten "do this, not that" pairs using real sentences from those articles, a banned words list, and punctuation conventions. Base it only on what's actually there - don't invent rules that sound good.

Write a channel file

> **TYPE THIS**
>
> Interview me about our <channel>, one question at a time, then write `knowledge/channels/<name>.md`: purpose, who arrives and why, format and length, structure rules, SEO rules, what we never do here, and which past pieces are good examples.

Build a product fact sheet

> **TYPE THIS**
>
> Interview me about <product> and write `knowledge/entities/products.md`. Include what it does, **what it explicitly does not do**, approved claims with their sources, claims we must never make, pricing, and approved boilerplate. Add "Last verified: <today>" at the top.

Planning

Get a plan, not a draft

> **TYPE THIS**
>
> Read `BRIEF.md` and the relevant knowledge files. Write a phased plan to `PLAN.md`. **Do not do any of the work yet.** 5–7 phases; for each: goal in one sentence, inputs needed, files produced, how we'll know it's done. Each phase small enough for one sitting. At least two must end with a decision I have to make. List your assumptions at the end.

Work one phase

> **TYPE THIS**
>
> Do Phase 2 only. Stop when it's done.

Change the plan

> **TYPE THIS**
>
> Three changes to `PLAN.md`: <...>. Update the file and tell me what else those changes affect.

Writing

Angles before drafting

> **TYPE THIS**
>
> Give me three angles. For each: the promise to the reader in one sentence, why we're credible on it, and what evidence we'd need. Recommend one and say why. Don't write the piece.

Research with real sources

> **TYPE THIS**
>
> Search the web for supporting data. For every source give the link, the publication date, and the exact claim it supports. Prefer primary sources. Skip anything you can't date. Save to `research.md`. Anything we need but can't source: `[NEEDS SOURCE: ...]`.

Outline with the opening written out

> **TYPE THIS**
>
> Write an outline: H2s with one line each on what that section does for the reader, plus the opening paragraph in full. Don't draft the rest.

Draft to a file

> **TYPE THIS**
>
> Write the full draft to `drafts/<slug>.md`, following the outline and the channel format rules.

Self-review - the one prompt here that improves a draft most

> **TYPE THIS**
>
> Audit your own draft against `knowledge/voice/house-voice.md` and the channel file. A table: each rule, ✅ or ❌, and the specific fix for each ❌. Check especially banned words, sentences over 25 words, paragraphs over three sentences, passive voice, em dashes, and unsourced claims. Be genuinely critical - a review that finds nothing is a review you didn't do. Then apply every fix and tell me what you changed.

Surgical edits

> **TYPE THIS**
>
> In the draft file: cut section 3, it repeats section 1. Tighten the intro to four sentences. **Leave everything else exactly as it is.**

Package it

> **TYPE THIS**
>
> Add to the top: three headline options, a meta description under 155 characters, a suggested slug, two internal links from our library. Add at the bottom: an image brief and every `[NEEDS SOURCE]` marker collected in one list.

Pressure-testing

Hostile reader

> **TYPE THIS**
>
> Read this as our most skeptical reader - <describe them>. List every sentence they'd roll their eyes at, and why. Don't fix anything yet.

The approver

> **TYPE THIS**
>
> You're the CFO / Legal / our head of sales reading this for the first time. What would you refuse to approve, and why?

Find the weak claims

> **TYPE THIS**
>
> List every factual claim in this draft. For each: the source, or mark it unsourced. Then tell me which unsourced claim would be most damaging if it were wrong.

Kill the AI smell

> **TYPE THIS**
>
> Find every sentence that sounds like it was written by AI rather than by us. Quote it, say why, and rewrite it in our voice.

Repurposing

> **TYPE THIS**
>
> Turn `drafts/<slug>.md` into: a 400-word newsletter piece, five LinkedIn posts, and an Instagram carousel outline. Read each channel's file first and write each one properly for that channel - do not paste the same paragraphs across formats. Save each to `drafts/repurposed/`.

Maintaining the system

After every piece of work

> **TYPE THIS**
>
> What did I correct today that you should have got right the first time? For each, tell me which file should have prevented it, and make the edit.

File a correction immediately

> **TYPE THIS**
>
> Add that to the right file so it holds from now on. Tell me which file you chose and why.

Health check on your setup

> **TYPE THIS**
>
> Read `AGENTS.md` and everything in `knowledge/`. Tell me: anything that contradicts anything else, anything too vague for you to actually follow, anything out of date, and the three things missing that would most improve your output.

Turn a session into a skill

> **TYPE THIS**
>
> Turn what we just did into a skill at `<skills directory>/<name>/SKILL.md`, including every correction I made along the way. Frontmatter with `name` matching the directory and a `description` saying what it does and when to use it, using the words I'd actually type. Mark with ⏸ every point where you must stop and wait for me.

Phrases worth memorizing

SAY THIS	TO GET THIS
"**Don't do the work yet.**"	A plan you can fix cheaply
"**One phase, then stop.**"	Control over a long job
"**Save it to `<path>`.**"	Output in a file, not lost in the terminal
"**Leave everything else exactly as it is.**"	Surgical edits instead of a full rewrite
"**Give me three options and recommend one.**"	You stay the editor
"**Audit that against `<file>`, rule by rule.**"	Real self-correction
"**Ask me before you…**"	A checkpoint where it matters
"**Which file should have prevented that?**"	A system that improves itself
"**Read `<file>` first.**"	No guessing

SAY THIS	TO GET THIS
"**If you don't have a source, write `[NEEDS SOURCE: ...]`.**"	Honest gaps instead of invented facts

Two things no prompt can fix

Check the facts yourself. Every statistic, quote, date, price, and spelling of a person's name that reaches a published page. The guardrails reduce invention a great deal. They don't eliminate it. Your `[NEEDS SOURCE]` markers tell you where to look first, but they are not a complete list of what to check.

Nothing publishes without a human. Keep "never publish, post, send, or schedule" in `AGENTS.md`, and keep it true - especially once you've connected MCP tools that could technically do it.