CLAUDE FOR TRAINERIZE COACHES · THE COMPLETE GUIDE

Everything from the series, **in one place**

The words, the shape and every cheat sheet from Videos 1 and 2. Read Part 1 once, keep Part 2 open next to the Claude desktop app.

Inside

PART 1 · GOES WITH VIDEO 1

The Operator's Vocabulary 37 AI terms for coaches, in plain English 4

What Runs Where Claude chat, Cowork and Claude Code side by side 22

The Source of Truth One place where your business is true 41

PART 2 · GOES WITH VIDEO 2

The Claude Code Coach Packet every cheat sheet from Video 2 53

- Claude Output Styles 55
- Plan Mode 57
- Skills, Commands, Hooks and Plugins 62
- Teach Claude to Remember 72
- Stay Organized in Claude Code 76
- My Software Stack 80

> **The skills and hooks from Video 2** come in the Skills Kit, a separate download in the same email as this guide. Unzip it, drag it into the Claude desktop app's Code tab and ask Claude to install it. Then type **/fmm-learn-claude** to start.

CLAUDE FOR TRAINERIZE COACHES · PART 1

The words and **the shape**

Goes with Video 1. The vocabulary, a map of what runs where, and the source of truth your assistant works from. The videos are the build.

FITNESS MARKETING MACHINE
SCALE WITHOUT LIMITS

The words you need before the tools make sense.

Every AI glossary is alphabetical, which is useless: you cannot understand *context rot* before you understand what a *token* is. This one is ordered the way the ideas actually stack. Read it top to bottom once and the rest of this world stops sounding like a foreign language.

37 TERMS 8 LEVELS ~20 MIN READ FOR COACHES, NOT ENGINEERS

HOW TO READ IT

Each level uses only words defined in the levels above it. **Nothing is explained using a term you have not met yet.** That constraint is the entire design, and it is why this works when other glossaries don't.

Every entry gives you four things: the plain-English definition, the same idea in coaching terms, and the part that actually matters most: **the one behavior it should change**. If you only skim one line per entry, skim that one.

You do not need to memorize any of this. You need to have *heard* it once, so that when it shows up in a video or a settings screen, it lands as familiar instead of intimidating.

01 What this thing actually is

Six words that explain why AI behaves the way it does. Almost every frustration people have with AI traces back to not knowing one of these.

BUILDS ON **NOTHING** · START HERE

01

LLM

Large Language Model · say it "ell-ell-em"

The AI itself. A program trained on an enormous amount of text that predicts what should come next, one small piece at a time.

IN YOUR WORLD

It is not a database you search. It is closer to an extremely well-read intern who has read most of the internet and has never once seen your client roster.

> WHY YOU CARE
>
> **It does not look anything up unless you give it a way to. Everything it says about *your* business is a guess until you hand it your actual data.**

02

Token

The unit of measurement for all of this

The chunk of text the AI reads and writes in, roughly three quarters of a word. "Coach" is one token. "Periodization" is three or four.

IN YOUR WORLD

Tokens are to AI what reps are to a set. Everything gets counted in them: what you can send, what it can say back, and what you pay.

> WHY YOU CARE
>
> **Every limit and every bill in AI is measured in tokens, not messages. A 40-page PDF is not "one message". It is thousands of tokens.**

03

Context window

The single most important idea on this page

The maximum number of tokens the AI can hold in mind at one time: your instructions, the conversation so far, every file you have given it, and everything it has already said back.

IN YOUR WORLD

It is working memory, not a file cabinet. Like a client who can hold their whole plan in their head during the session and remembers none of it on Thursday.

> WHY YOU CARE
>
> **This is the ceiling on everything. Nearly every "the AI got dumber" moment is really "we filled up the context window."**

04

Model

Opus · Sonnet · Haiku

A specific version of the AI, with a name, a capability level and a price. Claude comes in a few: Opus is the most capable, Sonnet is the balanced default, Haiku is the fastest and cheapest.

IN YOUR WORLD

Same logic as programming a session. You do not send every client to failure on back squats. You match the tool to the job in front of you.

> WHY YOU CARE
>
> **Reaching for the biggest model on everything is like doing max singles every workout: expensive and slow for no benefit. Use the small fast one for simple, repetitive work.**

05

Hallucination

The failure mode you must plan around

When the AI states something false with complete confidence: an invented statistic, a study that does not exist, a feature that was never built.

IN YOUR WORLD

It is not lying to you. It is the client who tells you they hit 10,000 steps every day and genuinely believes it. The confidence is real. The fact is not.

> WHY YOU CARE
>
> **Confidence is not accuracy. Anything touching money, a medical claim, or a client's personal data gets checked by a human being. Every single time.**

06

Non-deterministic

Why it never answers the same way twice

Ask the identical question twice and you get two different answers. Both can be completely correct. This is by design, not a defect.

IN YOUR WORLD

Two excellent coaches write two different programs for the same client. Neither of them is wrong.

> WHY YOU CARE
>
> **Stop treating it like a calculator. If you need byte-identical output every time, you need software, or you need the AI to *write* you that software, which it is very good at.**

---

02 How you talk to it

The gap between people who get astonishing results and people who get mediocre ones is almost never the model. It is these five words.

BUILDS ON **TOKEN** · **CONTEXT WINDOW**

07

Prompt

What you send it

The question, instruction, or raw material you hand over. Everything you type is a prompt.

IN YOUR WORLD

It is a client intake. "Get me in shape" and "I'm 42, three hours a week, bad left knee, want to pull 315 by June" produce very different programs.

> WHY YOU CARE
>
> **Most disappointing AI output is a vague brief, not a weak model. Specificity is the entire game.**

08

Prompt engineering

Writing the brief on purpose

Deliberately shaping your instructions to get better output: being specific, showing examples, defining what "good" looks like, and stating what to avoid.

IN YOUR WORLD

The difference between telling a client "do some cardio" and prescribing a 40-minute zone-2 session with a heart-rate range and an RPE target.

> WHY YOU CARE
>
> **The highest-leverage skill on this entire page, and the cheapest to learn. One great example of what you want beats three paragraphs describing it.**

09

Context engineering

The step up from prompt engineering

Deliberately managing everything that occupies the context window: which files are loaded, which history is carried forward, which standing instructions apply, so that the AI has exactly what it needs and nothing that distracts it.

IN YOUR WORLD

Prompt engineering is writing a good workout. Context engineering is designing the whole training environment: what equipment is in the room, what is on the whiteboard, and what the client already walked in knowing.

> WHY YOU CARE
>
> **This is where the people getting remarkable results actually differ. It is not a cleverer prompt. It is a better-loaded context.**

10

Context rot

Why long conversations get worse

The gradual decay of a long conversation. As the context window fills with old detours, abandoned ideas and superseded information, the AI starts confusing what is still true with what was true an hour ago.

IN YOUR WORLD

The client you have coached for three years through six goals, four injuries and two nutrition rewrites. Somewhere in there you lost track of what is actually true today.

> WHY YOU CARE
>
> **Long conversations are not free. When answers start drifting, do not argue with it. Start a fresh conversation and carry over only what still matters.**

11

Compaction

The automatic summary you should not fully trust

When a conversation runs up against the context window, the AI summarizes what came before and carries on from the summary rather than the full transcript.

IN YOUR WORLD

A handover note between coaches. Genuinely useful, and lossy: it keeps the decisions and loses the texture.

> WHY YOU CARE
>
> **After a compaction, assume detail was dropped. Restate anything load-bearing instead of trusting that it survived.**

---

03 How it learns your business

Generic AI gives generic answers. These four terms are how it stops guessing about you and starts working from what is actually true.

BUILDS ON **CONTEXT WINDOW** · **HALLUCINATION**

12

Grounding

The fastest quality upgrade available to you

Giving the AI real source material to work from, instead of letting it answer out of general knowledge.

IN YOUR WORLD

The difference between asking "how is my client doing?" and handing over their check-ins, their weigh-ins and their actual training log.

> WHY YOU CARE
>
> **Ungrounded AI is an opinion. Grounded AI is analysis. Before you blame the model, ask what you actually gave it.**

13

Source of truth

The boring discipline everything else depends on

The one place a fact officially lives. When two systems disagree, the source of truth is the one that wins by definition.

IN YOUR WORLD

Client bodyweight sits in a spreadsheet, in your coaching app, and in a DM from Tuesday. Which one is real? Until you decide, none of them are.

> WHY YOU CARE
>
> **AI faithfully multiplies whatever you feed it, including your mess. Naming the source of truth for each kind of data is unglamorous work that makes every clever thing downstream possible.**

14

Memory

Not the same thing as the context window

What the AI retains between separate conversations. The context window resets every time; memory is written down somewhere and deliberately reloaded.

IN YOUR WORLD

The context window is what is in your head during the session. Memory is what is written in the client's file.

> WHY YOU CARE
>
> **By default it remembers nothing about you. Anything you want it to still know next Tuesday has to end up in a file it reads on startup.**

15

Project instructions

often a file literally named CLAUDE.md

A file of standing instructions the AI reads at the start of every single session: who you are, how you work, what it must never do, and where things live.

IN YOUR WORLD

The onboarding document you write once for a new assistant coach, instead of explaining your standards on day one for the rest of your life.

> WHY YOU CARE
>
> **This is the highest-return file you will ever write. Every correction you find yourself repeating belongs in it. Permanently, and that same day.**

04 How it touches your other software

This is the level where AI stops being a writing assistant and starts being staff. It is also where most people get stuck for the first time.

BUILDS ON **GROUNDING** · **SOURCE OF TRUTH**

16

API

Application Programming Interface · say it "ay-pee-eye"

The defined way one piece of software talks to another without a human clicking anything. Every serious tool you pay for has one.

IN YOUR WORLD

Your gym's front desk. There is a defined way to ask for things, a defined way answers come back, and you do not get to wander into the back office and help yourself.

> WHY YOU CARE
>
> **"Does it have an API?" is the single question that decides whether a tool can ever be automated. Ask it *before* you buy the software, not after.**

17

Authentication

often shortened to "auth"

Proving to a system that you are who you claim to be, and that you are permitted to do the thing you are asking to do.

IN YOUR WORLD

The key fob at the 24-hour gym. The door does not know you or care about you. It only cares whether your fob is valid, current, and allowed through that particular door.

> WHY YOU CARE
>
> **This is where nearly everyone gets stuck first. When a connection "just stops working," it is almost always an expired or wrongly-permissioned credential, not a broken tool and not your fault.**

18

API key

also: token, secret, credential

The secret string of characters that authenticates you. It looks like `sk-ant-a1b2c3…` Whoever is holding it can act as you.

IN YOUR WORLD

A key to the gym, not a membership card. It does not identify you. It simply opens the door for whoever happens to be holding it.

> WHY YOU CARE
>
> **Treat it exactly like a password, because that is what it is. Never paste one into a chat, a screenshot, a screen share, or a video. If it is ever exposed, replace it that hour.**

19

Tool use

also called function calling

The AI's ability to *do* things rather than only describe them: read a file, run a search, send a request, update a record. It decides which tool to reach for and when.

IN YOUR WORLD

The jump from a consultant who writes you a beautiful plan to a coach who is on the floor moving the plates.

> WHY YOU CARE
>
> **This is the exact line between AI that advises and AI that operates. Everything genuinely valuable is on the far side of it.**

20

MCP

Model Context Protocol · say it "em-see-pee"

A shared standard for plugging tools into an AI. Instead of every AI needing custom code for every app, an app publishes one MCP server and any AI can use it.

IN YOUR WORLD

The 45-pound plate. Once the industry agreed on the standard, any plate fits any bar in any gym on earth. MCP is that agreement, for AI tools.

> WHY YOU CARE
>
> **When you ask "can Claude talk to my CRM?", the real question is "does it have an MCP server?" That is what turns AI from a chat box into something wired into the business you already run.**

---

05 The Claude Code vocabulary

Eight words that describe the actual machinery. Learn these and you can follow any tutorial, any video, and any settings screen in this world.

BUILDS ON **TOOL USE · MCP · PROJECT INSTRUCTIONS**

21

Claude Code

the thing this whole page is preparing you for

Claude with hands and a workspace. It runs on your own computer, reads and writes your files, runs commands, and uses connected tools, rather than only answering questions in a browser tab.

IN YOUR WORLD

The difference between texting a contractor photos of your kitchen and handing them the keys to the house.

> **WHY YOU CARE**
>
> **Despite the name, it is not only for code. It is the best tool available for running a business made of files, data, and connected software, which is every online coaching business.**

22

Terminal

also: command line, CLI, shell

The plain text window where you type commands instead of clicking buttons. CLI stands for command-line interface.

IN YOUR WORLD

Free weights versus machines. More intimidating on day one, fewer guardrails, dramatically more range once you are in.

> **WHY YOU CARE**
>
> **This is the one real hurdle on this page, and it is far smaller than it looks. You need about five commands to start, and the AI will hand you every one after that.**

23

Agent

the word that changes your job description

An AI given a defined role, a set of tools, and a goal, which then works through the steps on its own rather than answering one question at a time.

IN YOUR WORLD

The difference between an intern who answers exactly what you ask and a staff coach who owns client onboarding end to end.

> **WHY YOU CARE**
>
> **You stop writing prompts and start writing job descriptions. That shift is the entire point of everything above.**

24

Subagent

the practical cure for context rot

A fresh agent that the main one spins up to handle a chunk of work in its own clean context, reporting back only the conclusion.

IN YOUR WORLD

Delegating properly. Your assistant reads all 40 check-ins and brings you the three that need you. You never load the other 37 into your head.

> WHY YOU CARE
>
> **Heavy reading happens somewhere else and only the answer comes back. This is how you work on big things without poisoning your own context window.**

25

Skill

your SOPs, made executable

A written procedure the AI loads only when it is relevant: a small folder holding the instructions for doing one specific thing your way, every time.

IN YOUR WORLD

A standard operating procedure. Written once, followed identically whether it is 9am Monday or 11pm Friday.

> WHY YOU CARE
>
> **Every process you currently carry in your head is a skill waiting to be written down. Once written, it is executable: by the AI, by a new hire, or by both at once.**

26

Slash command

named for the / you type first

A shortcut you type, like `/debrief` or `/invoice`, that fires a saved instruction instead of you retyping the same request.

IN YOUR WORLD

A saved workout template. You built it carefully once; now it is two taps forever.

> WHY YOU CARE
>
> **The moment you catch yourself typing roughly the same request for the third time, that is the signal to make it a slash command.**

27

Hook

the difference between a habit and a system

A rule that fires automatically on an event, with no prompting from you at all: before a tool runs, after a file changes, when a session ends.

IN YOUR WORLD

The email that goes out automatically when a client misses two check-ins. Nobody decides to send it. It simply goes.

> WHY YOU CARE
>
> **Prompts are things you remember to do. Hooks are things that happen whether you remember or not. Anything you must never forget belongs in a hook, not in your head.**

28

Plugin

skills, commands, agents and hooks in one box

A bundle of all of the above, packaged so that someone else can install the whole kit in a single step.

IN YOUR WORLD

Buying a complete program rather than writing every session yourself.

> WHY YOU CARE
>
> **You do not have to build everything from scratch. And once you have built something genuinely good, this is how you hand it to your team, or sell it.**

---

06 Letting it work without you

The part everyone wants and almost everyone rushes. These four words are what separate leverage from a mess you have to clean up.

BUILDS ON **AGENT** · **TOOL USE** · **HALLUCINATION**

29

Agentic

the adjective, used constantly, rarely explained

Describes AI that chooses its own next step. You supply the outcome; it works out the sequence.

IN YOUR WORLD

"Get her ready for a half marathon in 16 weeks" instead of "write Tuesday's run."

> WHY YOU CARE
>
> **The word marks a change in what *you* are responsible for. You stop supervising steps and start defining outcomes and boundaries.**

30

Autonomous work

the overnight-run promise

The AI running a long task through to completion without you watching: many steps, sometimes hours, sometimes while you sleep.

IN YOUR WORLD

Handing off a project on Friday afternoon and reviewing the finished thing Monday morning.

> WHY YOU CARE
>
> **The real constraint is not capability, it is scope. Autonomy works beautifully when the task is clearly bounded and the AI can check its own work. It fails when the goal is vague.**

31

Human in the loop

what makes autonomy safe enough to actually use

A deliberate design where the AI prepares the action but a person approves it before it takes effect.

IN YOUR WORLD

Your assistant drafts every client email and you press send. All of the speed, none of the "who on earth sent this?"

> WHY YOU CARE
>
> **Decide up front which actions are automatic and which need your thumb. Money, client communication, and anything that deletes something are the usual line.**

Verification

the habit that separates the pros

Confirming the work is actually finished and correct, rather than accepting the AI's report that it is.

IN YOUR WORLD

The client says they hit their macros all week. The progress photo says otherwise.

> WHY YOU CARE
>
> **AI reports on itself optimistically. "Did it run?" and "did it produce the right result?" are two different questions, and only the second one has ever mattered.**

07 The files all of this lives in

Three formats. You can learn all three in under ten minutes, and they are what every skill, instruction and note in this world is made of.

BUILDS ON **SKILL** · **PROJECT INSTRUCTIONS**

33

Markdown

`file extension .md`

Plain text with a handful of simple marks for formatting. `#` makes a heading, `**like this**` makes bold, `-` starts a bullet. That is genuinely most of it.

IN YOUR WORLD

Writing a program on a napkin instead of in design software. Zero friction, readable by anyone, opens on any device, still works in twenty years.

> WHY YOU CARE
>
> **It is the native language of AI-readable documents. Every skill and instruction file in this world is markdown, and you can learn the whole format in about four minutes.**

34

Frontmatter

`the block between the --- lines`

A small block at the very top of a markdown file holding facts *about* the file: its name, what it is for, and when it should be used.

IN YOUR WORLD

The label on the outside of the folder, as distinct from the paperwork inside it.

> WHY YOU CARE
>
> **It is how the AI decides whether a file is even relevant before reading it. A skill with a vague description never gets picked up, no matter how good the instructions inside are.**

35

Repo & Git

`repository · the undo button for your business`

A repo is a folder that tracks its own history. Git is the system doing the tracking: every change, who made it, when, and exactly how to undo it.

IN YOUR WORLD

Version history for your entire operation, with an undo button that reaches back years and never expires.

> WHY YOU CARE
>
> **It is the safety net that makes it reasonable to let AI edit your real files. If something goes wrong, you roll it back with one command and lose nothing.**

08 What it actually costs

Two terms that will change how you use this daily, and stop you being needlessly stingy in exactly the wrong place.

BUILDS ON **TOKEN** · **CONTEXT WINDOW** · **MODEL**

36

Input vs. output tokens

the two halves of every bill

You are charged for what you send *and* what comes back, at different rates. Output typically costs several times more than input.

IN YOUR WORLD

Reading a client's check-in is cheap. Writing their next twelve-week block is not.

> WHY YOU CARE
>
> **Sending a big document is far cheaper than most people assume. Asking for a big document back is not. Ask for the decision, not the essay, unless you genuinely need the essay.**

37

Prompt caching

why being thorough is affordable

When you send the same context repeatedly (your instruction file, your knowledge base), it can be cached and re-read at a steep discount instead of charged in full every time.

IN YOUR WORLD

A membership versus a day pass.

> WHY YOU CARE
>
> **It is why loading real, substantial context on every session is affordable. Being thorough with your setup is cheaper than being clever about trimming it.**

JUST AS IMPORTANT

What you do not need to know

Half of feeling behind is people talking about things that have nothing to do with you. These are real, they are genuinely technical, and you can operate at a very high level without ever touching one of them.

- **Fine-tuning.** Retraining a model on your own data. Enormously expensive, and good context beats it for essentially everything you would want to do.
- **RAG.** A specific, formal way of doing grounding. Worth recognising the acronym; not worth building one yourself in 2026.
- **GPUs, parameters, training runs.** How models get built. Fascinating. Completely irrelevant to using one.
- **Vector databases & embeddings.** The plumbing behind AI search. Real, useful, and someone else's job. The tools you will use handle it invisibly.
- **Writing code.** You need to read it well enough to sanity-check, not write it. The AI writes it. You direct it.
- **Which model is beating which on the leaderboard.** It changes monthly. Pick a good one, learn context engineering, and stop reading the scoreboard.

THE SHORT VERSION

If you remember nothing else, remember these eight.

Every term on this page collapses into one of these. Screenshot this section.

01 **Context is the whole game.**

Not the model, not the prompt. What is loaded into the window when you ask.

02 **Confidence is not accuracy.**

It will be wrong in exactly the same tone it is right. Check anything that matters.

03 **Ground it, or it is guessing.**

Give it your real data. An ungrounded answer about your business is fiction with good grammar.

04 **Write it down once.**

Every correction you repeat is a file you have not written yet. Instructions, skills, memory.

05 **Tools are the line.**

AI that can only talk gives advice. AI with tools does the work. Everything valuable is past that line.

06 **Delegate, do not accumulate.**

Send heavy reading to a subagent. Protect your own context like you protect your calendar.

07 **Automatic beats remembering.**

If it must never be forgotten, it belongs in a hook, not in your head and not in a prompt.

08 **Verify the result, not the report.**

"I have completed the task" is a claim, not evidence. Go look.

None of this requires you to become a developer. It requires you to stop being a tourist. Thirty-seven words is the whole toll.

What changed on 16 September 2026

Cowork is not a separate place any more. Anthropic merged it into Claude chat, so the three surfaces in this chapter are now two. Everything this chapter says about *the work* still stands. Read this page first, then read the chapter with one substitution in mind.

CHAPTER TWO ERRATUM · VERIFIED 18 SEPT 2026 · APPLIES TO "THREE DOORS. WHICH ONE DO YOU OPEN?"

What happened

- Claude went from three modes to two: **Claude chat** and **Claude Code**.
- Cowork's abilities did not go away. They moved into chat, and Claude decides for itself when a request needs that heavier, longer-running treatment instead of asking you to pick a tab.
- Docs and Slides arrived at the same time, and Design now works inside a conversation.
- It is reaching Pro and Max plans first over the following weeks, then Team and Free. Enterprise administrators are told at least 30 days before anything changes for their organisation.

> WHY THIS MATTERS MORE THAN A MENU CHANGE
>
> Anthropic's stated reason for merging them is that people could not tell where a task belonged. That is the exact problem this chapter was written to solve. The answer is now partly built into the product — which is good news, and it does not remove the judgement, it moves it.

How to read this chapter now

Where the chapter says	Read it as
"Switch to Cowork"	Stay in chat. Hand Claude the folder and describe the finished thing.
"The Cowork column"	A description of how Claude behaves on project work, not of a separate tab.

“The borrowed room”	Still accurate. Project work runs in a fresh workspace on Anthropic's servers.
“Three doors”	Two doors, three jobs. The jobs are unchanged.

What did not change

- **Claude Code is still its own thing**, and still the only one with your files, your installed software and your logins. Every limit this chapter describes on that boundary still holds.
- **The decision rule survives.** It was never really about tabs. Ask whether the work needs your machine, your credentials and your tools. If it does, that is Claude Code. If it does not, it is chat.
- **Never paste an API key, a password or an MCP link into a chat.** Unchanged, and more relevant now that one surface does more.

> THE HONEST LESSON
>
> This is what moving fast looks like. A chapter written in September was overtaken by a product change two days before it shipped. That is the argument for learning the shape of the work rather than the layout of the buttons: the buttons moved, and nothing you learned here was wasted.

Sources · claude.com/blog/cowork-is-now-claude · techcrunch.com/2026/09/16/anthropic-merges-claude-chat-and-cowork-in-one-interface · fortune.com, 16 Sept 2026. Read and verified 18 Sept 2026. If you are reading this months later, check the current docs — that is the habit this whole series is trying to build.

CHAPTER TWO

Three doors. Which one do you open?

Claude, Cowork and Claude Code are not three versions of the same thing, and they are not a beginner, intermediate and advanced ladder. They are three different working arrangements. Once you can tell them apart, you stop asking "can Claude do this?" and start asking the only question that matters: *where does this particular job belong?*

3 SURFACES **1** DECISION RULE VERIFIED **SEPT 2026** CHECK THE DOCS BEFORE QUOTING

The thirty-second version

If you read nothing else on this page, read this table. Everything after it is explanation.

THE THREE SURFACES, SIDE BY SIDE

	CLAUDE	COWORK	CLAUDE CODE
In one line	Thinks *with* you	Does the work *for* you	Runs the machinery *behind* you
The relationship	A very sharp colleague you talk to	An assistant you hand a project to	An operations team with keys to the building
Best at	Answers, drafts, thinking out loud, decisions	Multi-step deliverables built from your own files	Anything repeatable, scheduled, or wired into your systems
Where work happens	In the conversation	In folders you point it at	On your machine and inside your accounts
Keeps working if you close the laptop	No	Yes in the cloud	Yes
Do you need to be technical	No	No	A little, and less every month
Cost	Works on the free plan	Paid plans	Paid plans
Remembers you as	Facts it picks up	The same facts, plus per-project memory	A system of files you own
Reads your CLAUDE.md	No	No not documented	Yes every session
From your phone	The app	Dispatch	Remote Control, iMessage
Start here if you are	New, or thinking	Drowning in documents	Ready to automate the business

THE TRAP TO AVOID

Most coaches pick one, get comfortable, and try to make it do everything. That is how you end up pasting the same spreadsheet into a chat window forty times. The skill is not mastering one door. It is recognising, in about three seconds, which one this job goes through.

Claude: the one you talk to

USE IT FOR · THINKING, DRAFTING, DECIDING

What it actually is

The chat window at claude.ai, the desktop app, or the phone app. You type, it answers. You paste something in, it works on it. Turn by turn, you and it, in a conversation.

This is the surface almost everyone starts on, and a great deal of people never need to leave. Do not let anyone make you feel behind for living here. A coach who is genuinely fluent at conversation with Claude gets more done than one who half-configured a terminal and gave up.

What it is genuinely best at

- **Thinking out loud.** Working through a pricing decision, a difficult client conversation, a positioning problem.
- **Drafting.** Emails, posts, programme descriptions, sales pages, scripts.
- **Making sense of a thing you paste in.** A contract, a spreadsheet of check-ins, a long transcript.
- **Answering, with your own connected accounts as context.** More on connectors below.

Where it runs out

It is turn by turn. It waits for you. If a job takes forty steps, you are personally supervising all forty. That is fine for a decision and miserable for a process. The moment you catch yourself doing the same conversation for the fourth time, you have outgrown this door for that particular job.

Cowork: the one you hand a project to

USE IT FOR · MULTI-STEP WORK ON YOUR OWN FILES

What it actually is

Anthropic describe Cowork as bringing Claude Code's agentic capability to knowledge work beyond coding. In plain terms: you point it at a folder, describe the outcome you want, and it works through the steps on its own until the thing is finished.

It reads, edits and creates files in the folders you give it access to. It can run commands in an isolated environment on Anthropic's servers. It can drive a browser, opening pages, clicking, typing and filling forms. It can run several streams of work in parallel. When it runs in the cloud, the work continues after you close your laptop, and scheduled work can run with no device of yours online at all.

Why this matters to a coach

This is the first door where you stop supervising and start delegating. "Take these eleven consult transcripts, pull out every objection anyone raised, group them, and write me a one-page summary with the three I should build content around" is a Cowork job. In chat it is an afternoon of babysitting. Here it is a sentence and a coffee.

The honest limits

- It is **in beta** on several platforms and the rollout differs by plan. Desktop for macOS is the most established; web and mobile are newer.
- It is **paid only**. There is no free tier.
- It works in **a borrowed room**, not at your desk. More on that below, because it is the single most useful thing to understand about Cowork.

The borrowed room

Same Claude, different desk. Cowork runs in an isolated, temporary environment on Anthropic's servers, separate from your computer. Each session gets a fresh one, and it is deleted when the session ends. It reaches your files only through the Claude Desktop app, and only the folders you have connected.

The part that matters: **it cannot run software installed on your computer, and it cannot use credentials stored on your computer.** Nothing you set up there survives the session, and nothing of yours is there unless you hand it in.

Claude Code is the opposite. It works at your desk: your installed tools, your logins, your saved keys, still sitting there tomorrow. That is the line between them.

A real example: sending a client their intake form

Here is a skill we run for our own coaching business. You say "send Sarah the intake form for the 12-week package," and it creates or updates her contact in the CRM, builds the right link so she

lands on the contract rather than just the waiver, and fires the automation that emails her. Walk it through both doors:

WHY THIS JOB CANNOT RUN IN COWORK

STEP	WHAT IT NEEDS	COWORK	CLAUDE CODE
Fetch the CRM key	A tool installed on the computer, plus a key file saved on it	No	Yes
Find or create the contact	That key	No	Yes
Build the right link	Just logic: which package, is it a re-sign	Yes	Yes
Fire the email automation	That key again	No	Yes

Cowork could *write* this skill for you. It could not *run* it. The one step it can do is the one that does not matter. Notice it is not the internet that blocks it: the CRM is a public website and Cowork can browse the web. What blocks it is that the key and the tool live on your machine.

So which jobs go where

SORTING REAL COACHING WORK

COWORK: FILES IN, FINISHED THING OUT	CLAUDE CODE: NEEDS YOUR DESK
Eleven consult transcripts into one objections summary	Sending the intake form: CRM key, contact write, email trigger
Your StoryBrand workbook into a messaging draft	Pulling this week's check-ins out of Trainerize
A transcript plus intake answers into a client one-sheet	Updating your client tracker in Google Sheets
A competitor's website into a comparison doc	Rules that fire every time something happens
A folder of receipts into an expenses summary	Reorganising a year of files and keeping every link working

THE RULE

If the job reaches your stuff through a connector or a folder you hand it, Cowork can do it. If it needs a key you manage or a tool you installed, it is Claude Code. For a Trainerize trainer that splits cleanly: Gmail and Calendar have connectors, so they are Cowork or chat jobs. Trainerize, your intake forms and your Google Sheet client tracker have APIs but no connector, which makes them Claude Code jobs. That is why onboarding, the thing you most want automated, lands on the Claude Code side.

SAY THIS TO A NERVOUS TRAINER

The borrowed room is a benefit, not just a limit. Cowork physically cannot wreck your laptop or leak a password, because it never has them. That makes it the right first step, and Claude Code the move once you have outgrown it.

Claude Code: the one with keys to the building

USE IT FOR · SYSTEMS, SCHEDULES, ANYTHING WITH AN API

What it actually is

Claude running on your own machine, with your files, your terminal, your credentials and your connected systems. It reads and writes anything you let it, runs commands, talks to databases, calls APIs, and remembers how you work through instruction files you write once.

It runs in a terminal, in a desktop app, on the web, and inside code editors. Despite the name, **it is not only for code.** It is the best available tool for operating a business made of files, data and connected software, which is exactly what an online coaching business is.

What only this door gives you

- **Skills** that encode your processes, and **hooks** that fire automatically on events, so a thing happens whether you remember it or not.
- **Subagents**, so heavy reading happens somewhere else and only the answer comes back to you.
- **Scheduled and recurring work** against your real systems.
- **The command line**, which is the part that quietly unlocks everything else. The rest of this chapter is about that.

The honest limits

There is a real setup cost, and it is the only door with one. You install something, you meet a terminal, and for about an evening you feel out of your depth. That is the whole toll. It is smaller than it looks, and it is the last time you pay it.

Remembering: facts versus a system

The most common belief about this is slightly out of date, and the correction matters.

You will hear that chat "forgets everything" and you have to re-explain yourself every time. That was true once. Chat now has **memory**: it saves things as you talk, you can read and edit every one of them in Settings under Memory, and you can simply say "remember this." It is on by

default on the Free, Pro and Max plans, and **chat and Cowork share the same memory**, so what one learns the other knows. Cowork projects add their own project memory on top, scoped so one project does not leak into another.

So chat is not weak at remembering *facts*. What it cannot do is maintain a *system*.

WHAT EACH SURFACE ACTUALLY REMEMBERS

	CLAUDE AND COWORK	CLAUDE CODE
What it keeps	A list of facts and topics it picked up	Files you own: instructions, memory, skills, client notes, docs
Who decides the structure	Claude	You. The seven drawers in Chapter Three.
Standing instructions	Typed into Settings, or into a project	A `CLAUDE.md` file, read every session
Your procedures	A skill you create in chat	A skill in a folder, loaded when relevant
Rules that fire automatically	No	Hooks
Undo yesterday's change	No	Yes version history
Updates its own reference files as it works	No	Yes

Does Cowork read your CLAUDE.md?

Not according to Anthropic's documentation. Cowork takes standing context from **global instructions** you type once in Settings, **project instructions** scoped to one project, and the folders or links you point it at. None of Anthropic's Cowork pages mention `CLAUDE.md`. Treat it as no.

That is a bigger difference than it sounds. Your `CLAUDE.md` is a file you own, version and can hand to anyone. Cowork's instructions live inside the app. Improve one and the other does not change.

> WHERE "LEARNING AS IT GOES" REALLY LIVES
>
> If you want Claude to *get better at your business every week*, writing down each correction, updating its notes on how Trainerize behaves, sharpening a skill after it fumbles, that is Claude Code. In chat, the best move is to capture a repeated procedure as a skill the moment you notice you are explaining it again. That works, but it is you doing the maintenance. In Claude Code, keeping the files current is part of the job.

Voice: the feature coaches underrate most

Of everything on this page, this is the one that changes a coach's day fastest, and it is the one almost nobody sets up.

Voice mode lets you speak to Claude and listen to it answer. In hands-free mode it listens continuously and replies at the natural pauses in your speech. You can switch between talking and typing inside the same conversation and the context carries over, so nothing is lost when you move from the car to your desk.

It is a beta feature, it is available on **every plan including Free**, and it runs on the mobile apps, desktop and web. It is built to work best from your phone.

> THE PART THAT MAKES IT POWERFUL
>
> Voice mode can use web search *and the tools you have connected* during the conversation. That means you can be walking between clients and say "what does my Thursday look like, and did that intake form ever come back from Sarah?" and it can actually go and look. It is not a dictation toy. It is your whole connected setup, operated by talking.

Why this fits coaching specifically

- **Your hands are busy and your brain is not.** Between sets, on the gym floor, driving between sessions, walking the dog after a consult.
- **The ideas arrive at the worst moments.** The programme tweak, the content hook, the thing you meant to tell a client. Voice catches them before they evaporate.
- **Post-consult debriefs.** Five minutes of talking in the car turns into a written summary and a task list by the time you are home.

> NOT THE SAME THING AS DICTATION
>
> Dictation just turns your speech into text in the message box, and you still read the reply. Voice mode is a spoken conversation in both directions. Both are useful. Dictation is better when you want a long, precise brief on the record. Voice is better when you want to think with something.

On the Free plan voice gives you Haiku and one connected tool. Paid plans open up the bigger models and all of your connected tools. Voice conversations count against your normal usage limits.

Running it from your phone

This is the part that changes how the work feels. Your computer keeps working at home, and you reach it from the gym floor.

There are three ways to do it, and they are not interchangeable. The simplest way to keep them straight:

- **Remote Control is steering.** You drive one Claude Code session on your computer, live, from your phone.
- **Dispatch is delegating.** You describe an outcome, it splits the work into tasks, runs them, and tells you when they are done.
- **iMessage is texting.** You message your running Claude Code like a person, and the reply comes back in the same thread.

THREE WAYS TO REACH YOUR COMPUTER FROM YOUR PHONE

	REMOTE CONTROL	DISPATCH	IMESSAGE CHANNEL
Part of	Claude Code	Cowork	Claude Code, as a plugin
You get	The live session, from any device	A manager that runs tasks for you	A chat thread with your session
Your full setup: files, CLAUDE.md, skills, hooks	Yes	For coding tasks. Knowledge tasks go to Cowork, which does not read CLAUDE.md.	Yes
Start new work from the phone	Yes, in server mode	Yes, that is the point	Yes, into the running session
Approvals	Push notification, approve on the phone	Forwarded to you; auto-denied after 10 minutes	Pauses unless the channel relays them
Must stay running	The Claude Code process	Claude Desktop open, computer awake	A Claude Code session with the channel on
If the laptop sleeps	Reconnects when it wakes, messages queued	Nothing runs until it wakes	Messages only arrive while it is running
Plans	All plans; Team and Enterprise need an admin	Pro and Max, limited beta	Pro and Max; Team and Enterprise need an admin
Status	Available now	Beta	Research preview, Mac only

Which one, when

- **You started something at your desk and want to keep going from the couch.** Remote Control. You get the exact session, mid-thought, with everything it already knows.
- **You want to hand off a pile of work and forget about it.** Dispatch. "Summarise this week's check-ins and draft replies for the three who went quiet" is a Dispatch brief.
- **You have a quick question between clients.** iMessage. It is the lowest-friction of the three because you are already in Messages all day.

> THE ONE THING THEY ALL NEED
>
> **Your computer has to be on.** All three do the work on your machine, which is exactly why they can use your files and your keys. A laptop that sleeps in your bag is a laptop that is not working for you. The setups that feel magical are the ones with a desktop or a mini at home that never sleeps.

> THE DISPATCH GOTCHA
>
> Dispatch sends coding tasks to Claude Code and everything else to Cowork. Your `CLAUDE.md`, your hooks and your installed tools come along for the first kind and not the second. If a dispatched task ignores a rule you are sure you wrote down, that is almost always why.

The browser, and using your screen

When there is no connector and no API, Claude can use the website or the app the way you do: by looking at it and clicking.

Claude reaches for things in a fixed order, and knowing the order explains almost everything about speed and reliability. **A connector first. The browser second. Your screen last.** Each step down is slower and more fragile than the one above it, so it is used only when the step above is not available.

WHO CAN DRIVE A BROWSER, AND WHO CAN DRIVE YOUR SCREEN

	CLAUDE CHAT	COWORK	CLAUDE CODE
Search the web	Yes	Yes	Yes
Drive a browser: open, click, fill forms	Through the Chrome extension, which runs as a Cowork session	Yes: a built-in browser, or your Chrome	Yes, through the Chrome extension
Use your logged-in accounts in Chrome	Yes	Yes	Yes
Control your whole screen and your apps	No	Yes, Pro and Max	Yes, Pro and Max

The Chrome extension

Claude in Chrome reads, clicks and navigates websites alongside you, on every paid plan. It opens its own tabs but **shares your browser's login state**, so it can reach any site you are already signed into. That is what makes it useful, and it is also why you should be deliberate about what you let it near. Claude Code can connect to the same extension to test a web page, fill a form or pull data out of a site.

Computer use

The last resort, and sometimes the only option: Claude opens apps, clicks and types on your actual screen. It works in both Cowork and Claude Code, on Pro and Max only, on macOS and Windows through the Claude Desktop app, and your computer has to be awake. It is in beta, and it is honest about its limits: screen control is slower than a connector, and complex tasks sometimes need a second try.

> WHERE IT EARNS ITS PLACE FOR A TRAINER
>
> The legacy studio software with no API. The admin screen of a platform that only exists as a website. The report you can export only by clicking through four menus. For those, screen control turns "I have to do this by hand" into "I asked for it." For everything that has a connector or an API, use that instead. It is faster, and it does not break when someone moves a button.

Connectors and CLIs: reach versus depth

This is the section that explains why serious operators end up in Claude Code, and it is not the reason most people assume.

What a connector is

A connector plugs one of your accounts into Claude so it can act there without you copying and pasting. You authorise it once in settings and it works everywhere your Claude account goes, including on your phone and inside voice mode. Google Workspace connectors are available on Claude and Claude Desktop.

They are more capable than people expect. Claude can search, read and analyse Gmail, and it can **draft, send, reply to and forward** email, asking your approval before anything goes out. It can check availability across you and your attendees, **create** calendar invites with Meet links and room booking, **update** events, **delete** them with cancellation notices, and RSVP for you. Drive is read-and-search by default, and you can tighten or loosen each permission in the connector's settings.

> CORRECTING A COMMON CLAIM
>
> You will still hear people say the Google connectors are read-only and that you need Claude Code to send an email. That was true once. It is not true now. If your only Google jobs are conversational, the connector is genuinely enough, and it works on your phone, which Claude Code does not.

So where do connectors actually run out?

Not on capability. On **breadth and repetition**. A connector exposes a curated set of actions across a few services. A command-line tool exposes the whole API.

GOOGLE WORKSPACE: THE CONNECTORS VERSUS A CLI, MEASURED

	GOOGLE CONNECTORS	GWS COMMAND-LINE TOOL
Google services covered	3 Gmail, Calendar, Drive	17 the list below
Sheets	No	Yes
Forms	No	Yes
Contacts	No	Yes
Docs and Slides, writing	No	Yes
Tasks, Chat, Meet, Keep	No	Yes
Depth per service	A curated set of actions	Every method the API exposes
Bulk operations	Not designed for it	Built for it, with auto-pagination
Runs on a schedule, unattended	No	Yes
Works on your phone	Yes	No
Works in voice mode	Yes	No
Setup	Click authorise, once	Install and authenticate

The seventeen services the CLI reaches: `drive`, `sheets`, `gmail`, `calendar`, `docs`, `slides`, `forms`, `tasks`, `people`, `chat`, `meet`, `keep`, `classroom`, `admin-reports`, `events`, `modelarmor` and cross-service `workflow`.

Look at that list as a coach. **Sheets, Forms and Contacts are not connector-reachable.** Your client tracker, your intake forms and your actual client list are three of the most important things you own, and they sit outside the curated set.

> THE RULE WORTH REMEMBERING
>
> **Connectors are for conversation. Command-line tools are for operations.** "Did Sarah reply?" is a connector question. "Find every client who has not been emailed in 30 days, cross-reference the training log, and draft the check-ins" is a CLI job. One is a question. The other is a process.

The command line, explained properly

This is the concept that unlocks the rest, and it is taught badly almost everywhere. Here is the version that actually lands.

What it is

Every application you use has a menu of things it can do. Send this. Delete that. List these. When you click a button, you are picking an item off that menu with your mouse.

A **command-line tool** is that same menu, typed instead of clicked. Nothing more mysterious than that.

```
# "list my next 10 calendar events"
$ gws calendar events list --params '{"maxResults": 10}'
```

That is a menu selection. Service, thing, action, details. Once you can see that shape, the intimidation drops away, because every command you will ever meet is that same four-part sentence.

Why typed beats clicked

Not because it is faster for one action. It usually is not. It wins because of what a typed instruction can do that a click never can:

- **It can be repeated exactly.** A thousand times, identically, with no drift and no fatigue.
- **It can be scheduled.** Run at 6am every Monday without anyone being awake.
- **It can be combined.** The output of one becomes the input of the next, so a chain of tools does a job no single app offers.
- **It can be written down and version-controlled.** A process that lives in a file survives you forgetting it.
- **It can be handed to an AI.** This is the one that matters. Claude cannot click your mouse. It can absolutely type.

> THE POINT MOST PEOPLE MISS
>
> You are not learning the command line so you can type commands. You are learning it so that **Claude can**. Your job is to describe the outcome. Its job is to know the syntax. You need to recognise roughly what is happening and when to say stop, which is a far smaller skill than writing it yourself.

Why this means "almost any software you use"

Chapter One defined an **API**: the defined way one piece of software talks to another without a human clicking. Nearly every serious tool you pay for has one, including Trainerize, your CRM, your booking system and your email platform.

An API on its own is not much use to you, because talking to one normally takes code. A **CLI wraps that API in typed commands.** And Claude Code can run typed commands. Chain those two facts together and you get the sentence this whole chapter exists to deliver:

> **If a platform has an API, it can have a CLI. If it has a CLI, Claude can operate it.**
>
> That is the entire unlock. Not a specific integration someone else decided to build for you. Any platform, including the niche one your business depends on that no integration directory has ever heard of.

And if it does not have a CLI, you build one

This is where it stops being a concept and starts being an afternoon's work.

Most platforms a coaching business runs on have an API and no command-line tool. Trainerize has an API. So does Amelia, the WordPress booking plugin. So does Formaloo. None of them shipped a CLI, so none of them were reachable.

So we wrote a **skill** that builds them. You describe the platform once, its base address, how it authenticates, where the credentials live, and it generates a complete working command-line tool in the same shape every time: consistent commands, consistent output formats, credentials pulled from secure storage rather than pasted anywhere.

What that has produced so far

COMMAND-LINE TOOLS BUILT IN-HOUSE FOR PLATFORMS THAT SHIPPED NONE

TOOL	WHAT IT REACHES	WHY IT MATTERED
`trainerize`	The Trainerize API	Programmes, clients and messaging, operable by Claude
`amelia`	WordPress booking	Appointments and availability across every coach site
`formaloo`	Forms and submissions	Intake data out of a walled garden
`mindbody`	Studio management	A platform with no useful automation surface
`n8n`	The automation platform itself	Automations that manage automations
`ploi`	Server management	Infrastructure without a control panel visit

Six tools, for six platforms that never intended to be operated this way. That is not a large engineering project. It is a repeatable pattern applied six times, and the sixth took an hour.

> **WHAT THIS MEANS FOR YOU**
>
> Stop asking whether a platform "integrates with AI". Ask whether it has an API. If it does, it is reachable, and the gap between reachable and reached is much smaller than it used to be.

Which door, in three seconds

The decision rule

1. **Am I thinking, deciding or drafting?** That is Claude. Use voice if your hands are busy.
2. **Is there a pile of my own material that needs turning into a finished thing?** That is Cowork.
3. **Will I need this to happen again, on a schedule, or against a system nobody built an integration for?** That is Claude Code.

Almost every job you have sorts cleanly into one of those three. When a job seems to want two doors, it is usually two jobs.

A worked week

- **Monday, driving:** voice, on the phone. "What is on today and did anyone cancel?"
- **Monday, at the desk:** Claude. Think through a pricing change for the new programme.
- **Tuesday:** Cowork. Eleven consult recordings into one objections summary.
- **Wednesday:** Claude Code. Build the check-in chase so it runs itself every Monday from now on.
- **Thursday, on the gym floor:** a text to your Claude Code over iMessage. "Send Marcus the intake form for the 12-week." It runs at home, on your machine, with your keys.
- **Friday, before bed:** Dispatch. "Summarise this week's check-ins and draft replies for anyone who went quiet." It is waiting for you in the morning.
- **From then on:** Wednesday's work happens without you. That is the whole point.

> WHERE TO START IF THIS IS ALL NEW
>
> Turn on voice mode and one connector this week. Do not touch a terminal yet. Get fluent at the conversation first, notice which jobs you keep repeating, and let that list tell you when it is time for the third door. The list will build itself.

Capabilities here were verified against Anthropic's documentation in September 2026. These products move quickly and several are in beta, so check the current docs before betting a business process on any single row.

CHAPTER THREE

Your business needs one place to be true.

Everyone who quits AI quits at the same point. Not because it could not do the work, but because the work felt like it was scattered across forty conversations and none of them remembered the others. That is not an AI problem. It is a filing problem, and it has six drawers.

7 DRAWERS **2** LOAD EVERY TIME **5** LOAD ON DEMAND BUILD IT IN **AN AFTERNOON**

Why it feels chaotic

The discouragement is real and it is predictable. It goes like this.

Week one is magic. You ask for a programme description and get something better than you would have written. Week two you ask again and it has forgotten your tone. Week three you paste your brand colours in for the fourth time. Week four you are re-explaining a client's shoulder injury to something that helped you with that exact shoulder a fortnight ago.

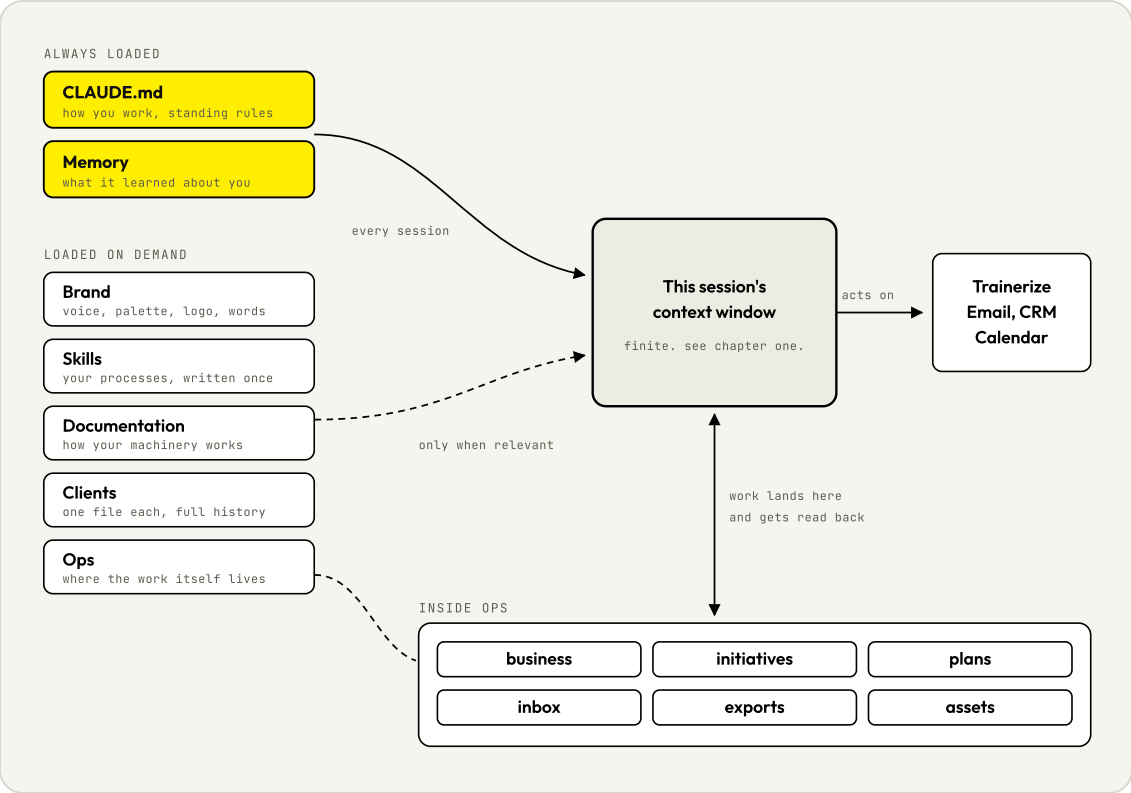
So you conclude the tool is unreliable, and you go back to doing it yourself.

Nothing was unreliable. You were starting from zero every time, because nothing you told it was written anywhere it could read. Chapter One called this the difference between the *context window*, which is working memory and empties every session, and *memory*, which has to be written down somewhere and deliberately loaded back.

> THE REFRAME
>
> You are not "using an AI". You are onboarding a very capable new hire who has total recall of anything written down and zero recall of anything that was not. Everything in this chapter is the staff handbook you would have written for a human, except this one gets read in full, every single time, forever.

The shape of it

Seven drawers. Two open automatically, five open only when the job calls for them. Six hold what Claude needs to know about you, and the seventh holds the work itself. That is the whole structure.



SEVEN DRAWERS. TWO LOAD THEMSELVES, FIVE WAIT TO BE CALLED. SIX HOLD WHAT CLAUDE NEEDS TO KNOW ABOUT YOU. OPS IS THE SEVENTH, AND IT HOLDS THE WORK ITSELF.

> **WHY ON-DEMAND IS A FEATURE, NOT A LIMITATION**
>
> A beginner's instinct is to load everything so Claude "knows it all". That is exactly how you cause context rot. If all forty client files are open, the one client you are actually discussing is competing for attention with thirty-nine who are irrelevant. **A good source of truth is not the one that loads the most. It is the one that loads the right thing.**

The seven drawers

WHAT EACH DRAWER IS, AND THE QUESTION IT ANSWERS

DRAWER	THE QUESTION IT ANSWERS	LOADS
CLAUDE.md	How does this person work, and what must never happen?	Every session
Memory	What have I learned about them that is not written anywhere else?	Every session
Brand	What do they sound like and look like?	Anything client-facing
Skills	How do they do this particular job?	When the task matches
Documentation	How does my machinery actually work, and where are the keys?	When touching it
Clients	Who is this person and what is their history?	When that client is named
Ops	Where is the work itself, and everything we are producing?	When working on it

1 · CLAUDE.md, the standing instructions

One file, read at the start of every session. Who you are, what business you run, how you like things done, what is off limits, and where everything else lives.

This is the highest-return file you will ever write, and the rule for filling it is simple: **every correction you find yourself repeating belongs in it, the same day you notice.** If you have told it twice not to use exclamation marks, that is not a preference any more, it is a rule, and rules go in the file.

2 · Memory, the running record

Distinct from CLAUDE.md. That file is what you deliberately wrote. Memory is what got learned along the way, one fact per note, indexed so the index loads cheaply and the detail loads only when it matters. Covered properly further down, because this is the drawer people manage worst.

3 · Brand, so it sounds like you

Voice and tone with real examples of both. Words you use and words you refuse. Colours with their hex codes and where each is allowed. Fonts. Logo files and the rules for using them. Your promise, your positioning, who you are for and who you are not for.

Skip this drawer and every piece of copy will be competent, generic and unmistakably not yours. Fill it once and everything downstream inherits it.

Where it lives. Brand is the one drawer that sits inside ops, at `ops/assets/brand/`, because most of it is files: logos, a social kit, a website theme. Start with `guidelines/brand-guidelines.md`. If you already have brand guidelines (a page on your site, or a PDF), you don't need to rewrite them. Keep one text copy there so Claude can read it quickly, put the link to the original at the top, and say the original wins if they ever disagree.

4 · Skills, your processes made executable

Every repeatable job you do, written down as instructions. Onboarding a new client. Writing a check-in. Building a twelve-week block. Handling a cancellation. Publishing a video.

Written as a skill, the process runs identically at 9am Monday and 11pm Friday, whether you run it, Claude runs it, or the person you hire next year runs it. This is the drawer that turns a coach into a business.

> **THE GOTCHA THAT SILENTLY BREAKS THIS**
>
> Claude Code only discovers skills in a hidden folder called `.claude/skills/`. Put them anywhere else and they simply never load, with no error and no warning. But a hidden folder is a miserable place to keep something you edit weekly.
>
> The fix is one line, and it gets you both: keep the folder visible at `skills/` and point Claude Code at it with a shortcut.

```
# run once, from your business folder
$ mkdir -p .claude && ln -s ../skills .claude/skills
```

Now the drawer is where you can see it and your assistant still finds it. If you ever move the folder, recreate that link or your skills will quietly stop working.

5 · Software documentation, how your stack works

One note per platform you depend on. What it does for you, how it authenticates, where the credentials live, which parts of it have an API, and the specific gotchas you have hit and solved.

The gotchas are the valuable part. Every hour you have lost to a platform quirk is an hour you will lose again unless it is written down. Write the quirk and the fix, and it becomes an hour you never lose again.

6 · Clients, one file each

Goals, history, injuries and limitations, preferences, what has been tried and what happened, decisions you made and the reasoning behind them.

This is the drawer that changes how coaching feels. "Draft Sarah's check-in" stops being a prompt you have to load with context and becomes an instruction that already has it. The file

makes the difference between an assistant who knows your clients and one who has to be told every time.

7 · Ops, where the work itself lives

The first six drawers are things Claude reads to understand you. Ops is different: it is where the actual work sits, and **everything Claude produces for you lands here.**

Claude writes into other drawers too, and should. It records what it learns into memory, and it writes up how a platform behaves into documentation. The difference is not read versus write, it is knowledge versus work.

It is one drawer with subfolders, not seven more drawers, and that matters. You are not memorising a filing system. You are opening one drawer and finding the obvious compartment inside it. Covered in full below.

The client one-sheet, and where it comes from

The Clients drawer is the one that sounds like admin until you see what fills it. You do not write these. Your sales calls write them.

Zoom is the input you already have

If you run consults on Zoom, you are already generating the single richest document about every client you have, and almost certainly throwing it away. The recording holds their exact words: the real reason they came, the thing they are embarrassed about, the last three things that failed and why they think it failed.

Connect Zoom and that transcript becomes readable. Point Claude at it with a template and you get a filled-in client profile instead of a two-hour recording nobody will ever rewatch.

> WHY THIS BEATS NOTES YOU TOOK YOURSELF
>
> Your consult notes capture what you thought was important while you were also selling. The transcript captures what *they* said, in their words. Those words are the ones that should show up later in their check-ins and their renewal conversation, and you will not remember them in March.

The one-sheet template

One file per client. Filled from the consult transcript and the intake form, then updated as things change. This is the shape worth standardising on:

```
# Sarah Chen · client profile
## Identity
Started        2026-09-01     Package  12-week 1:1
Timezone       PST            Check-in Sundays
Contact        sarah@...      Source   Instagram, saw the knee video

## Why she came (her words, from the consult)
"I've lost the same 20 pounds four times. I don't want another diet,
I want to stop doing this every two years."
Emotional driver: her daughter's wedding, but she led with the pattern.

## Goal
Primary      Body recomposition, hold it past 12 months
Measurable   Down 15 lb, deadlift bodyweight, 3x/week consistency
Deadline     Wedding, 2027-06-14

## History and limits
Training age   ~3 years, on and off
Injuries       L knee, meniscus 2019, no surgery. Pain on deep flexion.
Medical        Hypothyroid, medicated and stable
Equipment      Commercial gym, full access
Schedule       Mon/Wed/Fri early, 45 min hard ceiling

## Nutrition baseline
Start weight   168 lb
Initial macros P 140 / C 180 / F 60  (~1900 kcal)
Restrictions   No dairy, dislikes fish
Notes          Skips breakfast most days. Not negotiable, worked around.

## Training baseline
Squat 135x5 · Bench 85x5 · DL 165x5 · no overhead pressing yet

## What has failed before, and why
Two coached programs, both abandoned around week 7.
Her read: "the check-ins stopped feeling personal."
⇒ Flag: week 6-8 is the risk window. Do something human there.

## Decisions
2026-09-01  Leg press over back squat until knee tested. Revisit wk 4.
2026-09-01  Macros set conservative on purpose. She has dieted hard
            before and needs room to move down later.
```

Every section on that sheet answers a question you would otherwise ask again, or worse, guess at. The two most valuable blocks are the ones most coaches never write down: **what failed before and why**, and **the decisions with their reasoning**. Six months from now you will not remember why her macros started conservative, and without the note you will read it as a mistake instead of a plan.

> THE COMPOUNDING BIT
>
> Once these exist, "draft Sarah's week 6 check-in" is a complete instruction. It already knows week 6 is her risk window, that the knee is being tested, that fish is out and that her real fear is the fourth relapse. That is the moment coaching with AI stops feeling generic.

The memory layer, and how to keep it honest

This is the drawer people manage worst, because it is the only one that gets actively worse if you ignore it.

The other five drawers are things you write. Memory is what accumulates. Left alone it becomes a landfill: contradictory facts, notes about a client who churned last year, a rule you changed your mind about in March that is still quietly steering things in September.

A wrong memory is worse than no memory. No memory makes Claude ask. A wrong one makes it confidently proceed.

Six rules that keep it useful

1. **One fact per note.** Not a diary entry, not a session log. One thing that is true, in a file named after it. Notes that hold five facts get half-read and half-updated.
2. **Write the why, not just the what.** "Use 1900 calories for Sarah" rots the moment it changes. "Started conservative because she has dieted hard before and needs room to move down" survives, because it explains itself.
3. **Keep a cheap index.** One line per memory, in one file that loads every session. The detail sits behind it and loads only when relevant. The index is what makes the whole layer affordable.
4. **Date anything that can expire.** Prices, packages, platform quirks, team members, tools. A fact true in March is a liability in September if nothing says when it was written.
5. **Do not store what is already written somewhere.** If it is in the client file, the brand doc or the skill, it does not belong in memory too. Two copies means one of them is wrong and you will not know which.
6. **Delete confidently.** Pruning is maintenance, not loss. A memory that is no longer true should be gone, not annotated.

> THE TRIGGER TO WATCH FOR
>
> If you have corrected the same thing twice, stop and write it down. Not the third time, the second. That single habit is most of what separates a setup that gets sharper every week from one that plateaus in a fortnight and gets abandoned in a month.

Inside the ops drawer

Six drawers hold what Claude knows. This one holds what you are doing. Six compartments inside it, and you create each one the first time you actually have something to put in it.

> THE QUESTION THAT SORTS EVERYTHING
>
> **"Does Claude need to know this, or are we working on it?"** Knowledge goes in one of the first six drawers. Work goes in ops. That single question resolves almost every "where does this go" moment, and it is why the structure still holds up once there are a few hundred files in it.

business

The business as a business. Contracts and templates, pricing, policies, insurance, registration, anything you would hand an accountant or a lawyer. Named `business` rather than `documents` on purpose, so it never gets confused with the documentation drawer.

initiatives

One folder per thing you are actively building. The website rebuild. The group programme launch. The certification you are studying for. Each holds its own notes, drafts and decisions, and gets archived whole when it finishes.

This is the folder that stops long projects leaking into everything else. Without it, a three-month build scatters across forty conversations and you can never answer "where did we get to on that?"

plans

Thinking before doing. Rough, dated, disposable. Every plan either graduates into an initiative or gets deleted, and both outcomes are fine. Keeping plans separate from initiatives is what stops half-formed ideas being mistaken for commitments.

inbox

Capture. Voice notes from the car, links, half-thoughts, the thing you must not forget. It has exactly one rule: **it gets emptied.** An inbox you never process is just a pile with a nicer name.

This is the smallest folder on the list and the one that changes your week most, because it is what lets you stop holding things in your head between clients.

exports

Your Claude conversation history, saved as files so it is searchable later. Sessions are not permanent by default, and the good ones contain reasoning you will want again. Exporting them turns your own past work into something you can search, quote and reuse instead of half-remember.

assets

The files. `brand/` holds your guidelines, logos and social kit (the brand drawer lives here). `images/` holds headshots and photos, and `copy/` holds finished text like bios and profile copy. Kept together so your text folders stay light and quick to search. Videos are too big for this folder, so keep them on a drive or in your video host.

> **DO NOT BUILD ALL SIX ON DAY ONE**
>
> Create a compartment the first time you have something to put in it. An ops drawer with three used folders beats one with six empty ones, and empty folders are how a structure starts feeling like homework instead of a tool.

What it looks like on disk

Seven drawers, one folder. This is genuinely the whole thing.

```
my-coaching-business/
│
├── CLAUDE.md              ← read every session. start here.
├── memory/
│   ├── MEMORY.md          ← the cheap index
│   └── ...                  one fact per file
├── skills/
│   ├── onboard-client/
│   ├── write-checkin/
│   └── build-program/
├── documentation/         ← how your machinery works
│   ├── software/
│   │   ├── trainerize/      how you use it, where the keys are
│   │   ├── stripe/
│   │   └── zoom/
│   ├── blueprints/          what each automation does
│   └── workflows/
├── clients/
│   ├── sarah-chen.md      ← the one-sheet
│   ├── marcus-webb.md
│   └── _template.md         so every new one matches
│
└── ops/                   ← the work itself. the only drawer
    │                        Claude writes back into.
    ├── business/            contracts, pricing, policies, legal
    ├── initiatives/
    │   ├── client-journey/← first touch to renewal
    │   └── website-rebuild/
    ├── plans/               dated, rough, disposable
    ├── inbox/             ← capture. gets emptied.
    ├── exports/             saved conversations, searchable
    └── assets/              the files
        ├── brand/         ← the brand drawer lives here
        │   ├── guidelines/  start with brand-guidelines.md
        │   ├── logos/
        │   └── social/
        ├── images/          headshots, photos
        └── copy/            bios, profile copy, finished text
```

That is a filing cabinet, not a software project. If you can keep a client folder in Dropbox, you can keep this. The only genuinely new idea is that something now reads it.

> **ONE FOLDER, BACKED UP, VERSIONED**
>
> Keep the whole thing in a single folder that syncs and has version history. That way the entire operating system of your business is one directory you could hand to a new hire, restore onto a new laptop, or roll back if something goes wrong. Chapter One called this a *repo*. It is the safety net that makes it reasonable to let Claude edit real files.

How to actually build it

In order, smallest first. Do not try to fill six drawers in one sitting. Every coach who tries that finishes none of them.

1. **CLAUDE.md, twenty minutes.** Who you are, what you do, three things you always want and three you never want. It will be bad. Ship it anyway; it improves every time you correct something.
2. **One client file.** Your next consult. Record it, transcribe it, fill the one-sheet. One file proves the whole idea faster than any amount of planning.
3. **Brand voice.** Paste in five things you have written that sound like you and ask it to describe the voice. Edit what comes back. That is your voice doc, and it took ten minutes.
4. **Your first skill.** The job you do most and enjoy least. Write down how you do it. That is a skill.
5. **Documentation, as you go.** Do not sit down to document your stack. Write the note the next time a platform wastes an hour of your life. Write the client journey the first quiet afternoon you get; it is the one worth scheduling.
6. **Memory, continuously.** Never a task. Just the twice-corrected rule, forever.
7. **Ops compartments, on first use.** Make `ops/inbox/` today because you will use it today. Make the rest the first time you have something to put in one.

The whole chapter in four lines

1. **Seven drawers.** Standing instructions, memory, brand, skills, documentation, clients, ops.
2. **Two load every session. Five load when called.** That is context management, not filing.
3. **Ops is the seventh and the only one Claude writes into.** Ask "does Claude need to know this, or are we working on it?"
4. **Your consults already contain the client files.** You are just not keeping them yet.
5. **Corrected twice means write it down.** The habit matters more than the folder.

It stops feeling chaotic the moment you can name where a thing belongs. That is the whole trick, and you now have every name on the list.

A source of truth is not a project you finish. It is a place you put things, starting with the next thing you would otherwise have to explain twice.

CLAUDE FOR TRAINERIZE COACHES · PART 2

The Claude Code **Coach Packet**

Every cheat sheet from Video 2, in the order you saw them. Keep it open next to the Claude desktop app.

Inside

01 — 3

Claude Output Styles

Pick how Claude talks to you

02 — 5

Plan Mode

Design before you build, with the prompt I used

03 — 10

Skills, Commands, Hooks and Plugins

What each one is and when to use it

04 — 20

Teach Claude to Remember

The words that make it stick

05 — 24

Stay Organized in Claude Code

Rename, pin, group, and run more at once

06 — 28

My Software Stack

What I run my business on, and why

> **The skills and hooks from the video** come in the Skills Kit, linked in the same email as this packet. Unzip it, drag it into the Claude desktop app's Code tab and ask Claude to install it. Then type **/fmm-learn-claude** to start.

CLAUDE CODE · DESKTOP APP

Pick how Claude talks to you

An output style changes how Claude writes its replies. It doesn't change what Claude can do or which tools it has. Same assistant, different voice.

STYLE	WHAT CLAUDE DOES	BEST FOR	WHAT YOU'LL NOTICE
Default Get it done	Does the task and reports back briefly. No extra teaching.	Everyday work when you already know what you want.	Short, direct answers. Fastest to read.
Explanatory Show your work	Does the task, then explains the why behind its choices.	Understanding your own setup, or onboarding to a new system.	"Insight" boxes with 2 to 3 key points alongside the answer.
Learning Coach me	Works with you and leaves small parts for you to do yourself.	Building a skill you want to own, not just a result.	Invites to take a step yourself, then feedback on how it went.
Proactive Take the lead	Takes initiative: suggests next steps and moves work forward with fewer check-ins.	Bigger jobs where you want momentum and fewer questions.	More "here's what I did next" and fewer "want me to…?" messages.

REAL EXAMPLE FROM A COACHING BUSINESS

When Learning is worth the extra time

Explanatory teaches while Claude does the work. Learning has you do the key steps yourself. Here's how a coach might choose between them.

USE LEARNING

Building a skill for weekly check-in replies

Claude sets up the file and the structure, then hands you the parts only you can write:

- **The trigger:** the words you'd actually say that should make it run.
- **Your judgment calls:** when a check-in needs a call instead of a message, and what "progress before intensity" means for a client who missed three workouts.
- **Your voice:** one example reply that sounds like you.

Skills are how your business runs without you in the room. Do this once and you can write the next skill yourself.

SKIP LEARNING

Routine admin

Sending an intake form, tagging a contact, cleaning up disk space, publishing a page like this one.

The process is already written down, and nothing in it depends on your judgment. Making you do the steps only slows you down. Use Default.

Use Learning when the knowledge is the asset. Use Default when the result is the asset.

How to switch

1. Open any chat in the Code tab.
2. Click the chat title at the top and choose **Output style**.
3. Pick a style. It takes effect on Claude's next reply, in that chat only.

Try the same question in two styles side by side. The difference shows up right away.

CLAUDE CODE · DESKTOP APP

Plan Mode

In plan mode, Claude thinks before it acts. It reads, asks you questions and writes a plan, but it doesn't change anything in your files or your apps until you approve that plan.

What happens in plan mode

1

You describe the goal

A new product, a new onboarding flow, a new automation.

2

Claude interviews you

A few questions at a time about what you deliver, how you charge and which tools you use.

3

Claude writes the plan

What gets built, in which app and in what order.

4

You approve it

Or ask for changes. Claude only starts building once you say yes.

When to use it

USE PLAN MODE

- Creating a new product or coaching offer
- Anything that touches more than one app, like billing plus your coaching app plus your CRM
- Anything your clients will see or pay for

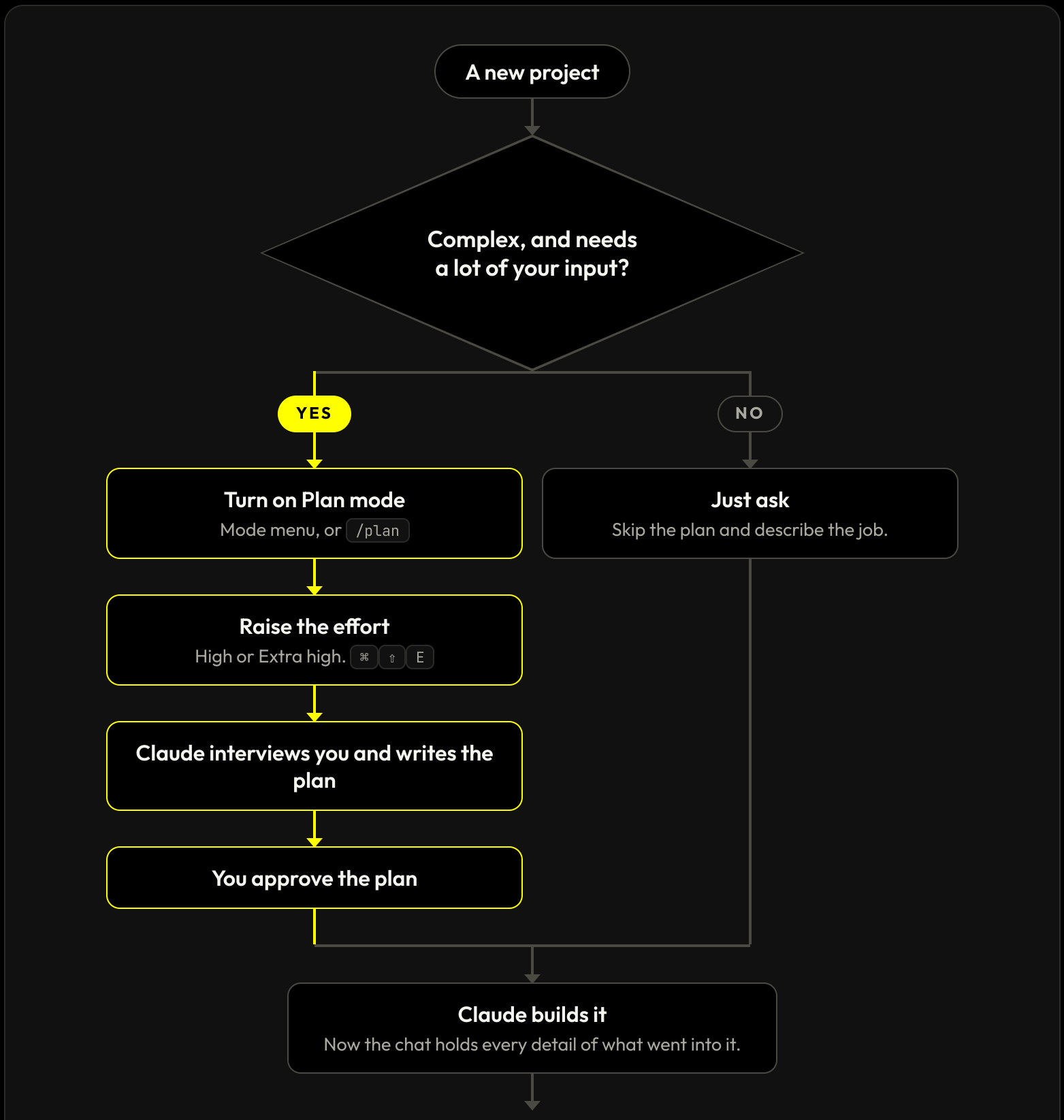
SKIP IT

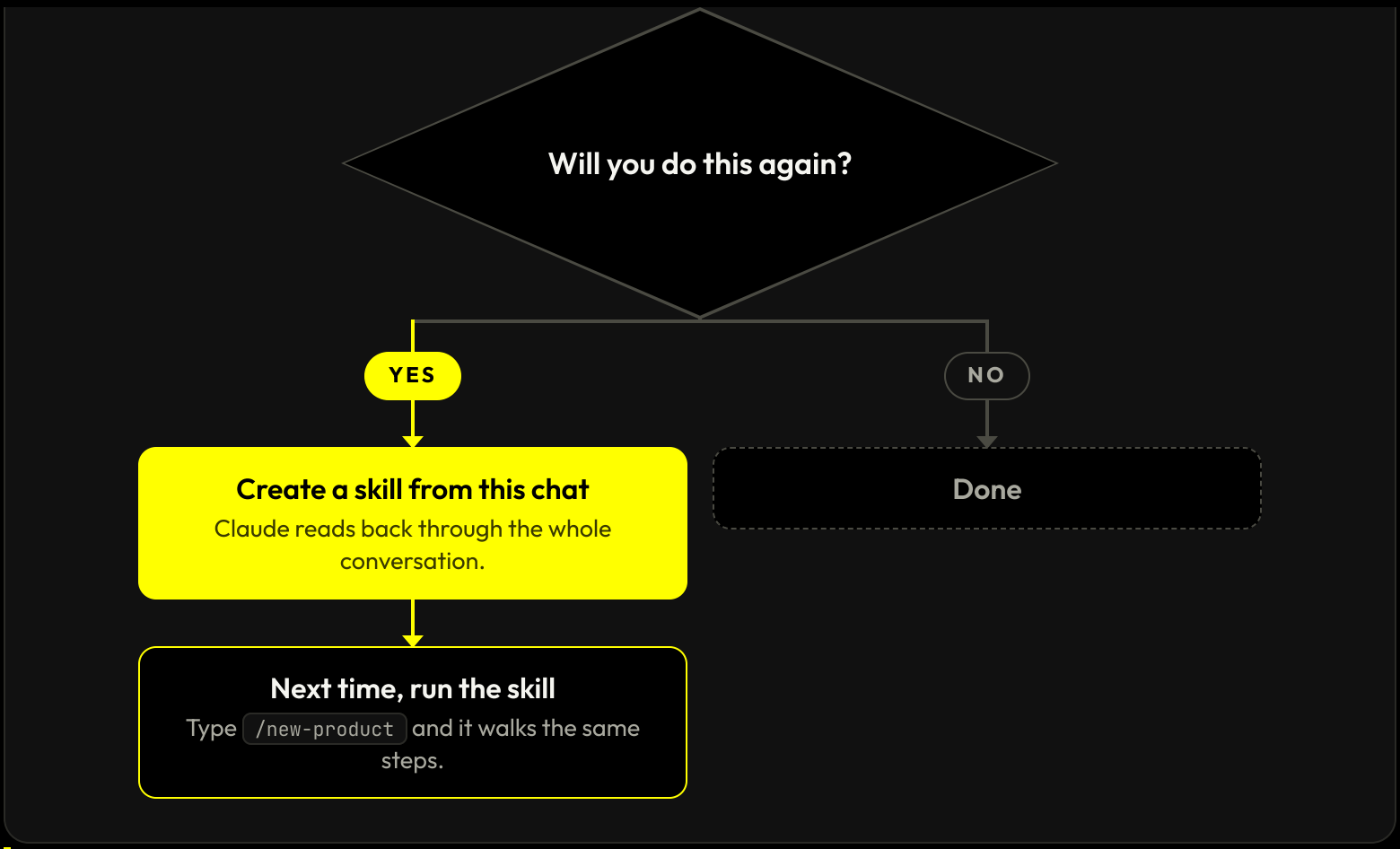
- Quick jobs you already know how to describe
- Sending a form, tagging a contact, fixing a typo

> **Planning takes ten minutes. Undoing a product that's already live in four apps takes a lot longer.**

Plan it, build it, keep it

The first time you do something big, the whole chat becomes a record of how it's done: your answers, every decision, the order things got built. If you'll do it again, turn that chat into a skill before you close it.





> The first time, you teach Claude how you do it. **Every time after, it's one command.**

How to turn it on

1. In the Claude desktop app's **Code** tab, open the mode menu next to the message box and choose **Plan mode**.
2. In the terminal, press `Shift` + `Tab` until you see **plan mode on**.
3. Or skip the menu: the prompt below tells Claude to plan first and not build anything yet.

Copy-and-paste prompt: plan a new product

Fill in the tools you use, delete the ones you don't, and paste it into a new chat. Claude will ask the right questions, one small batch at a time.

PROMPT

I'm an online fitness coach and I want to plan a new product (a coaching offer I can sell). Don't build or change anything yet. Use plan mode, interview me, and help me turn it into a clear plan.

MY SETUP (fill in what you use, delete what you don't):
- Coaching app (programs, check-ins, messaging): [e.g. Trainerize]
- Website / CRM (sales pages, contacts, tags, emails): [e.g. WordPress + my CRM]
- Payments: [e.g. Stripe]
- Bookings / scheduling: [e.g. Amelia, Calendly]
- Intake forms and waivers / agreements: [e.g. Gravity Forms, WP E-Signature]
- Task list: [e.g. ClickUp]

HOW TO INTERVIEW ME:
- Ask 2 to 4 short questions at a time, then wait for my answers. Skip anything I've already told you.
- I may be talking loosely or dictating. Say the numbers back to me so I can confirm them.
- If a term might be new to me (renewal, rollover, add-on, payment plan), explain it in one plain sentence when it comes up.

WHAT TO COVER, IN THIS ORDER:
1. The offer: name, who it's for, the result they get, and a one-paragraph "what's included" in my voice.
2. Deliverables: everything the client gets (training program, check-ins, 1:1 sessions, group calls, nutrition, habits, community). For each one: how many, how often, how long, and how it gets delivered (automated in my app, or something I do by hand).
3. The agreement: how long it lasts, and whether it renews automatically or they sign again.
4. Pricing: 1 to 3 ways to pay (pay in full, payment plan, monthly). For each one, write the math out: number of payments × amount = total. Ask which option shows first.
5. Onboarding: what happens from "they paid" to "first workout" (intake form, waiver, agreement, welcome message, booking a kickoff call).
6. Selling it: where it's sold (sales page, store, private payment link), CRM tags, and the emails that go out.

WHEN WE'RE DONE, GIVE ME:
- A one-page summary of the product I can read in 60 seconds.
- A build checklist grouped by app (what to create in each tool I listed, in order).
- Open questions or anything that doesn't fit my tools, flagged clearly.
Don't create anything in any app until I approve the plan.

Copy-and-paste prompt: turn this chat into a skill

Paste this at the end of the same chat, right after the build is finished, while every detail is still in the conversation. Change the name to fit the job.

PROMPT

```
This is something I'll do again. Create a skill from this whole conversation so next time I can run it with one command.

LOOK BACK THROUGH THIS CHAT AND CAPTURE:
- The questions you asked me, in the order that worked.
- The decisions we made, and why, so you can suggest them next time.
- The steps you took in each app, in the order they actually worked.
- Anything that went wrong or had to be redone, so the skill avoids it.

MAKE IT REUSABLE:
- Keep what stays the same every time.
- Turn what changes (names, prices, dates, the client or product) into questions the skill asks me at the start.
- Plan first and wait for my approval before building anything, the same way we did today.

Name it [new-product] and save it as a skill in this workspace. When it's done, show me a short summary of what it does and how to run it.
```

CLAUDE FOR TRAINERIZE COACHES · GUIDE PART TWO

Skills, Commands, Hooks and Plugins

Four ways to teach Claude how your business works, so you stop repeating yourself. What each one is, the skills that come with this guide, the plugins worth installing, and the command-line tools I use every day.

SECTION 1

The four, in plain English

COMMAND

A saved prompt

Type `/` and its name, and Claude gets the instructions you saved. **One job, no extra files.** If you keep pasting the same prompt, make it a command.

`/weekly-recap` writes this week's client recap in your format.

SKILL

A process Claude can run

A folder with step-by-step instructions plus anything the job needs: scripts, templates, reference notes. **Handles multi-step work across your apps.** Claude picks it up on its own when it fits.

Send an intake form: find the client, pick the right form, send it, log it.

HOOK

A rule that always fires

Runs on its own at a set moment, every time. **A rule in CLAUDE.md is something Claude usually follows. A hook runs no matter what.**

Before messaging a client, check for anything they sent that you haven't read.

PLUGIN

A bundle you install

Skills, commands, hooks and connectors packaged together, so one install gives Claude a whole new ability. **Browse them in the app.**

hookify: say a rule out loud, get a hook.

SECTION 2

How Claude decides to use a skill

Every skill has a name and a short description. Claude reads those at the start of every session. When what you ask matches a description, it opens that skill and follows it. You can also call one yourself by typing `/` and its name. **So write the description in the words you'd actually say.**

YOU, TO CLAUDE

```
Make a task to follow up with Jane about her renewal.
```

NO CLICKUP SKILL

- Claude uses the ClickUp connector however it sees fit.
- The task might land in a different list each time.
- Tags, due dates and custom fields are a guess.

WITH YOUR CLICKUP SKILL

- Renewals always go to **your Renewals list**.
- It always adds **your tags** and the **client's name** field.
- It sets the due date **the way you do**, every time.

> The connector gives Claude the keys. **The skill tells it how you want things done.**

SECTION 3

Skills that come with this guide

RUN THIS FIRST

FMM Learn Claude

Picks one small, real project for your business and walks you through it step by step, telling you which control to use and why. In 20 to 30 minutes you've used everything from the video and **built your first skill**.

Start: open the Code tab in your folder and type `/fmm-learn-claude`.

SKILL

Coach Credentials

Lets Claude use your API keys and passwords **without ever seeing them**. Keys go from your password manager straight to the app that needs them, never into the chat. Asks which password manager you use.

Start: say "set up my credentials."

SKILL

Sync Conversations

Saves every Claude Code conversation as a searchable file in one folder, including ones you came back to later. **Now Claude can look up exactly what you decided last month.**

Start: type `/sync-conversations`. Needs the Claude Conversation Extractor (section 7).

Also in the download: the **Skill Cleaner** prompt. Paste it before you install any skill you found online. It gives you a safety verdict and a version reworked for your own tools.

SECTION 4

How a tool, a skill and a hook work together

Here's how the three layer up, using your conversation history as the example.

1 · THE TOOL

Claude Conversation Extractor

A free command-line tool that exports your Claude Code chats. **It can do it. It doesn't know how you want it done.**

2 · THE SKILL

Sync Conversations

Runs the tool **my way**: every session, updated ones included, saved to one folder with a name I can search.

3 · THE HOOK

Look before you leap

Before Claude changes anything in one of my apps, it searches past conversations on that topic and reads my notes for that software, so it works from what we already learned.

SECTION 5

Hooks: the "whens"

Every hook is "when this happens, do that." These are the moments you can hook into. **The one you'll use most is right before a tool, because that's where guardrails live.**

WHEN	EXAMPLE
A session starts `SessionStart`	Load today's context: who has a check-in today.
You send a message `UserPromptSubmit`	Add a reminder, or catch something you shouldn't have pasted, like a password.
Right before Claude uses a tool `PreToolUse`	The guardrail. Check for unread messages before messaging a client. Ask before deleting anything.
Right after Claude uses a tool `PostToolUse`	Clean up after it, like formatting a file it just wrote.
Claude finishes answering `Stop`	"Did we decide anything that isn't written down?"
Before a long chat gets summarized `PreCompact`	Save your place, or ask "should any of this become a skill?"
A session ends `SessionEnd`	Wrap up, like syncing your conversations.
Also there `Notification` · `SubagentStop`	When Claude needs your attention, and when a helper agent finishes its part.

Hook ideas for a coaching business

Look before you leap

Before changing anything in an app, search past conversations and your notes for that software.

Unread-message guard

Before messaging a client, check for anything they sent that you haven't read.

Ask before deleting

Anything, anywhere. Covers what the connector toggles don't.

Confirm before offboarding

Ask before removing clients whose contracts have run out.

Key scan before GitHub

Block anything shaped like a password or API key from being backed up.

Formatting before posting

Check your style rules before a comment, email or post goes out.

Nothing decided goes unrecorded

When Claude finishes, check that decisions made it into

Save before summary

Before a long chat gets summarized, update your task with

You don't have to write hooks yourself. Install the hookify plugin and say the rule out loud: `/hookify warn me before deleting any video file`. Plain English in, guardrail out. Most of these need your own tools (Trainerize, ClickUp), so pick the ones that fit your business.

SECTION 6

Plugins worth installing

In the desktop app, open **Customize → Browse plugins**, search the name and install. Or type the line below into Claude Code.

PLUGIN	WHAT IT DOES FOR A COACH	INSTALL LINE
superpowers	Brainstorms with you, then writes a plan before anything gets built. Planning, done properly.	`/plugin install superpowers@claude-plugins-official`
hookify	Say a rule out loud, get a hook. No expertise needed.	`/plugin install hookify@claude-plugins-official`
skill-creator	Explain a process out loud, or point it at a finished chat, and get a skill you can rerun.	`/plugin install skill-creator@claude-plugins-official`

PLUGIN	WHAT IT DOES FOR A COACH	INSTALL LINE
obsidian	Claude writes proper Obsidian notes, links and canvases in your vault. Add its marketplace first.	`/plugin marketplace add kepano/obsidian-skills` `/plugin install obsidian@obsidian-skills`
playwright	Claude drives a real browser: tests your forms and checkout, clicks through every step and takes screenshots.	`/plugin install playwright@claude-plugins-official`
stripe	Help with payments, subscriptions and Stripe errors.	`/plugin install stripe@claude-plugins-official`

SECTION 7

My favorite command-line tools

A command-line tool lets Claude operate an app by typing instead of clicking. Install it once in Terminal, and Claude does the typing from then on. **Connectors are for conversation. Command-**

line tools are for getting work done.

TOOL	WHAT CLAUDE DOES WITH IT	INSTALL
Claude Conversation Extractor `claude-extract`	Turns your Claude Code sessions into searchable files. The Sync Conversations skill runs it. Free and open source.	`pip install claude-conversation-extractor` GitHub page
Google Workspace CLI `gws`	Gmail, Calendar, Drive, Sheets and Docs from one tool. Reaches far more of Google than the connector. Not an official Google product.	`npm install -g @googleworkspace/cli` GitHub page
Bitwarden CLI `bw`	Fetches your keys for the Coach Credentials skill, without Claude seeing them.	`brew install bitwarden-cli`
GitHub CLI `gh`	Backs up your workspace and grabs skills and tools from GitHub.	`brew install gh`
Firecrawl `firecrawl`	Web search and whole pages as clean text.	firecrawl.dev
ffmpeg	Converts, trims and compresses video and audio. Great for a YouTube workflow.	`brew install ffmpeg`
Whisper `whisper`	Transcribes your recordings on your own computer.	`pip install openai-whisper`

TOOL	WHAT CLAUDE DOES	INSTALL
`brew`	these come from. Install it first.	

Don't want to type any of this? Paste the install line into Claude Code and ask it to install the tool for you. On Windows, ask Claude for the Windows version of the line.

Plugin names and install lines checked September 2026. Hooks run in Claude Code only, not in Chat.

CLAUDE FOR TRAINERIZE COACHES · GUIDE PART TWO

Teach Claude to remember

Claude keeps notes on what it learns about you and your business, and reads them every session. Here's what it keeps, where it keeps it, and the words that make it save something **the moment you say it**, so you never have to say it twice.

What Claude remembers, and how

1

It's automatic

Claude decides on its own when something is worth keeping. The trigger words just make sure it does.

2

It loads every session

Every new chat starts by reading your CLAUDE.md and the index of Claude's notes. It opens a full note when it's relevant.

3

One note per file

Each thing it learns is its own small text file, kept in a hidden folder on your computer.

About you

Your role, your business, how you like answers. "I'm a solo coach, keep replies short."

How you like things done

Your corrections, and what you said yes to. "Never send without showing me the draft."

What you're working on

Projects, deadlines and the decisions you made. "New program launches March 1."

Where things live

Which folder, list or link something belongs in. "Renewals go in the Renewals list."

Find the files on a Mac They're hidden because the folder name starts with a dot.

1. In Finder, press `⌘` `⇧` `G` (Go to Folder).
2. Paste `~/.claude/projects/` and press Return.
3. Open the folder named after your workspace, then `memory`. `MEMORY.md` is the index. Every other file is one note.

On Windows: paste `%USERPROFILE%\.claude\projects\` into File Explorer's address bar. To see hidden files in any Finder window, press `⌘` `⇧` `.`

Trigger it on the fly: the words that make it stick

Always...

"Always use the client's first name in check-in replies."

`Saved: check-in replies use the first name`

Never...

"Never message a client without showing me the draft first."

`Saved: show drafts before any client message`

Remember...

"Remember that Jane is training for a half marathon in March."

`Saved: Jane Doe, half marathon, March`

Don't forget...

"Don't forget to tag new leads "Consult booked" in my CRM."

`Saved: new leads get the "Consult booked" tag`

From now on...

"From now on, save every new program in the Programs folder."

`Saved: new programs go in Programs/`

Whenever...

"Whenever a client misses two check-ins, draft a nudge for me."

`Saved: 2 missed check-ins → draft a nudge`

Talk to it like a new hire on day one. Say the rule once, out loud, and it's kept.

Make it reliable: add this to your CLAUDE.md

Claude already saves a lot on its own. This one paragraph in your CLAUDE.md makes it save every time you use those words, and tell you what it saved, so you know it stuck.

PASTE INTO CLAUDE.MD

```
## Memory
When I say "always", "never", "remember", "don't forget", "from now on", "rule:", or "whenever this happens, do that", save it to memory right away. Don't wait until the end of the session.
Then confirm in one line: "Saved to memory: [what you saved]."
If what I said changes something you already saved, update that note instead of adding a second one.
```

Check it, fix it, tidy it

CHECK

"What do you remember about my check-in process?"

FIX

"Update that: nudges go out after **three** missed check-ins, not two."

FORGET

"Forget what I said about the Programs folder."

TIDY

"Review your memory. Merge duplicates and remove anything out of date."

Where each thing lives

CLAUDE.MD

Your standing instructions

A file **you** write. Claude reads it at the start of every session. Keep it short and link out to other files.

MEMORY

What Claude learns

Notes **Claude** writes as you work: your rules, your clients, your preferences. Plain files you can read, change and take with you.

HOOK

What must happen every time

A rule in memory is something Claude **usually** follows. If it can never be missed, make it a hook.

Coming from ChatGPT or another AI? You don't start from zero. Use **Import memory** to bring over what it already knew about you.

CLAUDE FOR TRAINERIZE COACHES · GUIDE PART TWO

Stay organized

Once Claude is doing real work, you'll have a lot of conversations going. Here's how to keep them findable, run several at once, and check in from your phone.

SECTION 1

Keep your sidebar tidy

Rename

Give every conversation a name you'd search for later, like "March program build."

How: click the title, right-click it, or type `/rename`

Group

Sort conversations by folder, by project, or into groups you make, like "Clients" and "Content."

How: the filter icon, then Folder, Project or Custom groups. Right-click a chat for **Move to group**.

Pin

Keep the conversations you use every day at the top, like your weekly check-ins or your content planner.

How: right-click the conversation, then **Pin**

Archive

Done with it? Archive it. It's out of the way but not deleted, so you can always find it again.

How: right-click the conversation, then **Archive**

SECTION 2

Read the sidebar at a glance

Each conversation shows what it's doing. **The two yellow ones are waiting on you.**

Working
Claude is busy. Let it go.

Needs input
It asked you a question.

Needs approval
It wants your OK before it acts.

Idle
Waiting for your next message.

Done

Finished the job.

SECTION 3

Do more than one thing at once

Queue a message

Type your next ask while Claude is still working. It goes in line, or jumps the line.

How: **Send now** with `Enter`. **Queue it** with `⌘` `Enter` (`Ctrl` `Enter` on Windows)

Ask a side question

A quick question that doesn't derail the main job.

How: type `/btw`, or press `⌘` `;`

Side by side

Give a conversation its own window, then put two next to each other. Build a program on the left, write the welcome email on the right.

How: right-click a conversation, then **Open in → New window**

Jump to the files

Open the folder a conversation is working in, to see or grab what Claude made.

How: right-click, then **Open in → Finder** (or **VS Code**)

SECTION 4

Run it from your phone

Remote Control lets your phone drive the Claude Code session on your computer. Your computer still does the work, with your files, skills and tools. The phone is just the remote.

1 Check it's on

Settings, then Claude Code. It's on for new sessions by default.

2 Open the Claude app on your phone and tap **Code**

3 Pick the same session you left open on your computer

Same conversation, same folder, picking up right where you were.

4 Type or dictate your next ask

It runs on your computer. You watch it happen on your phone.

FROM YOUR PHONE

- **Works through Remote Control:** your skills, your files, your command-line tools.
- **Doesn't work in the regular phone chat:** skills. That chat can't reach your computer.
- **Your computer has to be on and awake.**

> **A laptop asleep in your bag isn't working for you. A computer at home that stays on is.**

SECTION 5

Let it work overnight

Routines

Jobs that run on a schedule while you sleep: sort your inbox, draft replies to client messages, clean up yesterday's files.

How: sidebar, then **Routines**, then **New routine**. Pick Local to run on your computer, or cloud.

Continue in the cloud

Hand a conversation off to run in the cloud, so it keeps going even when your computer is off.

How: right-click, then **Open in → Cloud**

An always-on computer

My Mac mini stays on most of the time and runs routines overnight. My laptop works off it remotely, so nothing gets interrupted when I close the lid.

CLAUDE FOR TRAINERIZE COACHES · GUIDE PART TWO

My Software Stack

The apps I run my coaching business on, and what each one is for. Start with the free column. Nearly everything here has a free way in, and Claude can operate most of it for you.

Free, or a generous free tier START HERE

TOOL	WHAT I USE IT FOR	COST	LINK
Obsidian Notes and knowledge	Your business brain as plain files on your own computer: SOPs, offers, client notes, and the instructions Claude reads every session.	Free. Sync and Publish are paid add-ons.	obsidian.md
VS Code Code editor	The higher-tech version of Obsidian. Same folders and files, plus a built-in terminal, git and extensions. Switch when you start building your own tools.	Free	code.visualstudio.com
GitHub Backup and version control	A private backup of your whole workspace, with every version saved, so any change can be undone. Also where you'll find skills and tools to install.	Free, private repos included	github.com
Puzzle Process mapping	A visual map of how your business runs: every step of the client journey, who does it and which tool powers it.	Free tier. Paid plans per editor.	puzzleapp.io
ClickUp Project management	Tasks, docs and projects in one place: onboarding checklists, your content calendar and every to-do Claude creates for you.	Free forever plan. Paid plans for more.	thefmm.click/pm affiliate
n8n Automation	When this happens, do that. Follow-ups, reminders and syncs that run while you coach, sleep or train.	Free to self-host. Paid cloud plans.	thefmm.click/automate affiliate

TOOL	WHAT I USE IT FOR	COST	LINK
Supabase Database	One home for client, sales and check-in data, so Claude can answer questions across all of it.	Generous free tier. Idle free projects pause.	supabase.com
Firecrawl Web research	Lets Claude search the web and read whole pages as clean text: competitor offers, research, your own site.	Free tier. Paid plans for volume.	firecrawl.dev
Bitwarden Password manager	Holds your logins and API keys. With the credentials skill, Claude uses your keys without ever seeing them.	Free for one person. Team plans are paid.	bitwarden.com
DaVinci Resolve Video editing	A full video editor for YouTube, reels and course lessons. Claude can build and rework your timelines for you: pick the best takes, make the cuts, and hand you a file to import.	Free. Studio is a one-time purchase.	blackmagicdesign.com
Wavebox Browser for web apps	Trainerize, email, ClickUp and your socials grouped in one window, each logged in and in its place.	Free tier. Paid Pro for more.	thefmm.click/wavebox `affiliate`

Paid WORTH THE MONEY

TOOL	WHAT I USE IT FOR	COST	LINK
Claude Your AI	The brain that ties everything else together. Chat thinks and writes with you. Claude Code works on your computer, inside your apps.	Free tier for Chat. Claude Code needs a paid plan.	thefmm.click/claude `referral invite`
Trainerize Coaching app	Where your clients train, track and message you: programs, habits, nutrition and check-ins under your own brand.	Paid. Free for one client.	thefmm.click/trz `affiliate`
Google Workspace Email, calendar, files	Email on your own domain, your calendar, Drive and Sheets. With the gws command-line tool, Claude works in almost all of it.	Paid, per user	workspace.google.com
Stripe Payments	Takes payments, payment plans and subscriptions. Everybody uses it, and Claude can read it to answer "who paid, who didn't."	No monthly fee. A fee on each payment.	stripe.com

> **Where to start.** Day one: Claude, Obsidian and Trainerize. Month one: ClickUp, Bitwarden, GitHub and Puzzle. When you automate: n8n, Supabase and Firecrawl. When you publish: DaVinci Resolve.

Disclosure. Links marked affiliate may earn me a commission if you sign up, at no extra cost to you. Wavebox gave me free licenses. The Claude link is a referral invite, not a commission.

Plans change, so check each site before you buy. Fitness Marketing Machine · Claude for Trainerize Coaches · thefmm.click/guide