The Non-Developer's OpenClaw Playbook - Free Sample

Nick Rae

The Non-Developer’s OpenClaw Playbook

How a Commercial Pilot Built an AI-Powered Life

By Nick Rae

For the tinkerers, the side-hustlers, and the people who were told “you need to learn to code first.”

You don’t.

Published by Nick Rae | nickrae.net

First Edition: April 2026

This book reflects the author’s personal experience with OpenClaw as of April 2026. Software changes. Commands may evolve. The principles won’t.

Table of Contents

Chapter 1: Getting Started Right

Foundation: Hardware, Installation & Basic Setup

I'll be honest with you. I'm a 46-year-old commercial pilot. Six months ago, I couldn't tell you the difference between JSON and JavaScript. But today? My AI assistant manages my smart home, tracks my investments, schedules my flights, and even wrote parts of this book.

The difference wasn't learning to code. It was learning to set up OpenClaw correctly from the start.

Most people get this wrong. They rush through installation, skip the identity setup, choose the wrong hardware, and then wonder why their expensive AI does nothing useful. This chapter fixes that.

The hardware decision: Mac Mini vs VPS

My recommendation: Start with a Mac Mini

I know, I know. Everyone talks about cloud VPS servers because they're "always on" and "scalable." But unless you're running a Fortune 500 company, you want the Mac Mini. Here's why:

Mac Mini advantages (what I use)

- **It just works**: macOS handles hardware conflicts for you
- **Local access**: No SSH fumbling when things break (and they will)
- **Performance**: M2/M3 chips handle AI workloads beautifully
- **Integration**: iMessage, FaceTime, macOS apps all available
- **Familiar**: If you use a Mac laptop, this feels like home
- **Power efficient**: Mine runs 24/7 for about $3/month in electricity

VPS when it makes sense

- You need geographic distribution (different time zones)

- Your internet connection is unreliable
- You're running 10+ simultaneous agents
- You already live in terminal/SSH land

Bottom line: Mac Mini for 95% of people. VPS only if you have specific technical requirements.

What you actually need

Hardware shopping list

- **Mac Mini** (M2 base model is plenty)
- **External monitor** (any will do, even an old one)
- **Keyboard and mouse** (for initial setup)
- **Ethernet cable** (Wi-Fi works but wired is more reliable)
- **External storage** (optional, for backups)

Software prerequisites

You probably already have these, but double-check: - **macOS Monterey or later** (check Apple menu > About This Mac) - **Admin access** to your Mac (you can install software) - **Internet connection** (obviously, but needs to be solid)

Installation: the right way

Most guides skip the boring stuff and jump to the fun parts. That's why they fail. We're doing this properly.

Step 1: download and install

Open Terminal (press Cmd+Space, type "Terminal", press Enter) and run:

```
curl -fsSL https://openclaw.ai/install.sh | bash
```

What this does: - installs the OpenClaw CLI - puts `openclaw` on your PATH - prepares the default `~/.openclaw/` home directory

If Terminal still says `command not found` afterward, close it and reopen it once. Shell path updates are sometimes the only thing being dramatic.

Step 2: run onboarding

Use the built-in wizard:

```
openclaw onboard --install-daemon
```

You'll be asked about: - your model provider and API key - gateway/network setup - workspace location, usually the default `~/.openclaw/workspace` - optional channel setup

This is the cleanest path because the wizard reflects the current release better than random screenshots on the internet.

Step 3: choose a model without overthinking it

If you're new, start with one reliable paid model and move on. Anthropic Sonnet, OpenAI, or Gemini all work. Specific model IDs change over time, so trust the current onboarding recommendations and docs more than any frozen screenshot in a book, including this one.

If you want to change models later, do it after the system is already working. First goal: get a reply, not win the benchmark Olympics.

Step 4: verify the Gateway and open the dashboard

Check that the service is running:

```
openclaw gateway status
```

If it isn't running yet, start it:

```
openclaw gateway start
```

Then open the Control UI:

```
openclaw dashboard
```

Send a simple first message in the chat, like:

> Hello. Can you tell me what time it is?

If you get a reply, **you're in business**.

If you get errors, check: - onboarding completed successfully - your API key/ provider choice is valid - `openclaw gateway status` shows the service running - `openclaw status --all` for a deeper read on what's broken

Connecting your chat app

Here's where OpenClaw gets really useful. Instead of typing commands in Terminal, you'll chat with your agent like texting a friend.

Telegram setup (recommended)

Why Telegram: Works on all devices, reliable message delivery, supports file sharing, voice messages, and has the best OpenClaw integration.

1. **Create a bot**:
   - Message @BotFather on Telegram
   - Send `/newbot`
   - Choose a name: "YourName's Assistant"
   - Choose a username: "yournamebot" (must be unique)
   - Copy the token (looks like `123456789:ABCdefGHIjklMNOpqrSTUvwxyz`)
2. **Add the token to OpenClaw**:

```
openclaw channels add --channel telegram --token "your-bot-token-
        here" --name "Telegram"
```

1. **Start or restart the Gateway**:

```
openclaw gateway start
```

1. **Approve your first DM**:
   - Find your bot on Telegram and send it any message
   - In Terminal, run:

```
openclaw pairing list telegram
openclaw pairing approve telegram <CODE>
```

   - Pairing codes expire, so don't wander off and make coffee first
2. **Test it**:
   - Send "What time is it?"
   - Your agent should respond

Discord/WhatsApp alternatives

Discord and WhatsApp both work, but Telegram is still the easiest first setup for a personal assistant. Get one channel working before you start collecting integrations like Pokemon cards.

Your first automation

Let's create something immediately useful: a morning weather report.

Open your Telegram chat with your agent and send something like:

> "Set up a cron job that runs every morning at 7am and sends me a weather briefing via Telegram. Include temperature, conditions, whether I need an umbrella, and any severe weather alerts."

That's it. Your agent will create the cron job for you. No YAML files, no terminal commands.

If you're curious what happens under the hood, OpenClaw stores a structured cron definition with a schedule, a payload, and a delivery mode. It looks more like this now:

```
{
  "name": "morning-weather",
  "schedule": { "kind": "cron", "expr": "0 7 * * *" },
  "payload": {
    "kind": "agentTurn",
    "message": "Send me a morning weather briefing for my area.
        Include temperature, conditions, whether I need an
        umbrella, and any severe weather alerts."
  },
  "delivery": { "mode": "announce" }
}
```

The exact JSON can vary by channel and version. The important part is that you don't have to hand-write any of it.

What this does: - Runs every morning at 7 AM - Gets weather for your location (OpenClaw figures this out) - Sends a summary to your Telegram - Includes practical advice (umbrella, jacket, etc.)

Tomorrow morning, you'll wake up to your first automated message. And that's when it clicks: *this thing actually works.*

Choosing your model: the honest comparison

This matters more than you think. Different models have different personalities and tradeoffs, but the exact names and version numbers change constantly.

Anthropic Sonnet

- **Best for**: strong reasoning, coding help, long conversations
- **Cost**: mid-range paid option
- **Personality**: measured, usually clear, sometimes a little too careful
- **Weaknesses**: not the cheapest choice for constant background volume

OpenAI GPT family

- **Best for**: broad general use, creative work, flexible tool use
- **Cost**: varies a lot by model
- **Personality**: fast, capable, sometimes too confident for its own good
- **Weaknesses**: easy to overspend if you pick the wrong tier for automation

Gemini / lower-cost models

- **Best for**: cheaper background jobs, lightweight automation, experimentation
- **Cost**: often lower than flagship models
- **Personality**: good enough for a lot of recurring work

- **Weaknesses**: less consistent on harder reasoning tasks

My recommendation: Start with one reliable paid model, get the system working, then optimize cost later. Premature model shopping is nerd catnip, not progress.

The reality check

By the end of this chapter, you should have:

✅ OpenClaw installed and running
✅ An AI model connected and responding
✅ Telegram bot sending you messages
✅ Your first automation scheduled
✅ Confidence to move forward

If any of these aren't working, **stop here** and troubleshoot. The rest of the book builds on this foundation.

What's next

In the next chapter, we'll teach your agent who you are. This is where OpenClaw transforms from a fancy chatbot to a personal assistant that actually knows you.

The identity setup takes 30 minutes but saves hours every week. It's the difference between "generic AI responses" and "feels like it knows me personally."

Troubleshooting common issues

"Command not found: openclaw"

- Check installation: `which openclaw`
- Restart Terminal once
- Re-run the install script if needed

"Invalid API key" or model auth errors

- Re-run onboarding: `openclaw onboard`
- Check for extra spaces or a bad pasted key
- Confirm the provider account actually has billing or credits enabled

"The Gateway isn't responding"

- Check service status: `openclaw gateway status`
- Start it if needed: `openclaw gateway start`
- Run `openclaw status --all` for a fuller diagnosis

“Telegram bot doesn’t respond”

- Confirm the bot token was added with `openclaw channels add`
- Make sure you sent the bot a DM first
- Run `openclaw pairing list telegram` and approve the pending code

“Something is still weird”

- Open the dashboard: `openclaw dashboard`
- Run `openclaw status --all`
- Check whether the problem is channel setup, model auth, or the Gateway service

Remember: Every expert was once a beginner who got stuck on these same issues. The difference is they kept going.

In the next chapter, we’ll make your agent personal.

Chapter 2: Your First 24 Hours

Identity: Teaching Your Agent Who You Are

When I first got OpenClaw running, I did what I always do with new technology: I jumped straight to the “cool stuff.” I started asking it to check my Kalshi positions, help with my FeedFare app, manage my TikTok pipeline. And it answered, sort of. The responses were… fine. Technically correct. But they felt like talking to a customer service rep who had to ask for your account number every single time.

“What’s your preferred format for updates?”

“Could you clarify what kind of investment you’re referring to?”

“I don’t have context for FeedFare. Could you describe it?”

Every single conversation. Over and over. I was re-introducing myself to my own assistant daily, which kind of defeats the purpose of having an assistant.

The fix was embarrassingly simple, and it’s what this chapter is about. Identity files. Thirty minutes of setup that changes everything.

Why your agent doesn’t know you (yet)

Here’s something I had to wrap my head around: your AI agent doesn’t have a persistent brain the way you do. Every time it starts a new session, it wakes up fresh. No memory of yesterday, no idea who you are, no understanding of your life or preferences.

That's not a bug. It's how large language models work. The "memory" they have is whatever is in their context window right now: the active conversation plus any files they've been told to load.

Think of it like this: imagine hiring a brilliant assistant who, through some weird medical condition, loses all their memories every morning. They're still brilliant (all their skills and knowledge intact). But they need a briefing every single day about who you are, what you do, and how you like things done.

The identity file system is that briefing. You write it once, and OpenClaw loads it automatically at the start of every session. Your agent wakes up already knowing you.

The identity file system

OpenClaw uses a set of structured markdown files stored in your workspace at `~/.openclaw/workspace/`. Each file serves a specific purpose, and together they form a complete picture of who your agent is and who it's working for.

One naming note before we go further: **OpenClaw is the platform. Talos** is just the name of my personal agent inside it. You can name yours whatever you want. The system matters. The agent name is flavor.

Here's what each file does:

SOUL.md: Your agent's personality and values IDENTITY.md: Your agent's name, role, and operating parameters USER.md: Information about you, the human AGENTS.md: Definitions for any sub-agents your system uses

You don't have to create all of these on day one. Start with USER.md and IDENTITY.md. Those two alone will transform your experience.

SOUL.md: who your agent is

This is the most interesting file to write. SOUL.md defines your agent's character: its communication style, its values, how it handles uncertainty, what it cares about.

I named my agent Talos. Here's a simplified version of my SOUL.md to show you what I mean:

```
# Talos: Soul & Values

## Core Character
I am Talos, a personal AI assistant. I am direct, practical, and
        honest.
I don't pad responses with pleasantries when they aren't needed. I
        don't
say "Certainly!" or "Great question!" I just answer.

## Communication Style
- Short responses unless depth is actually needed
- Bullet points for lists, prose for explanations
- Admit uncertainty immediately. Never bluff.
```

```
- Use plain language. Nick is smart but not technical.

## What I Care About
- Nick's time is valuable. Get to the point.
- Accuracy over confidence. A wrong answer stated confidently is
        worse
  than an honest "I'm not sure."
- Proactive heads-up when something seems off

## Tone
Professional but not stiff. Like a competent colleague who skips
        the
small talk. Not cold, just efficient.
```

Notice what's in there: specific instructions about *how* it talks, not just what it knows. This is the part most people skip, and it makes a real difference. An agent without a SOUL.md will default to the base model's personality: helpful but generic, excessively enthusiastic, padded with unnecessary filler.

Write the personality you actually want to work with every day.

IDENTITY.md: the role and boundaries

Where SOUL.md is about character, IDENTITY.md is about role definition. What is this agent's job? What is it authorized to do? What should it refuse?

```
# Talos: Identity & Role

## Role
Personal AI assistant to Nick Rae. Primary operator of the
        OpenClaw
system. Handles scheduling, automation, research, app development
support, and smart home control.

## Operating Scope
- Full read/write access to workspace files
- Can execute cron jobs and automation tasks
- Can interact with connected integrations (Telegram, smart home,
        etc.)
-
        NOT authorized to: make financial transactions, post
        publicly to social
  media without review, or access systems outside the defined
        workspace

## When in Doubt
Ask before acting on anything irreversible. It's better to confirm
        once
than to undo something that can't be undone.

## Name & Persona
Name: Talos
```

```
I respond to "Talos," "hey Talos," or messages sent via Telegram.
I do not pretend to be human. If asked, I acknowledge I'm an AI.
```

The “when in doubt” section is something I added after an early incident where my agent was a little too enthusiastic about executing things. The boundaries you set here are real. OpenClaw respects them.

USER.md: everything about you

This is the file that makes your agent actually useful. USER.md is your profile, the briefing document your agent reads every morning before your “workday” starts together.

Mine is several hundred lines long at this point, but here’s the structure:

```
# Nick Rae: User Profile

## Who I Am
- Name: Nick Rae
- Age: 46
- Location: Merced, CA
-
        Occupation: Mobile electronics retail manager, Mon-Thu
        9-6 / Fri 8:30-5, commercial pilot on the side
-
        Technical level: Non-developer, can follow instructions
        and edit files

## My Projects
### FeedFare
iOS screen time management app, a "walk to unlock" concept where
        kids
earn screen time by walking. Currently on Expo SDK 55. Stack:
        React
Native, TypeScript, Supabase backend. I don't write the code
        (Talos
and Claude Code do). I manage direction and testing.

### Kalshi Weather Prediction Bot
Automated weather trading bot on Kalshi prediction markets.
        Located at
~/dev/prediction-bot. Live mode is only allowed with explicit
        written
approval and defined risk limits. Never change KALSHI_LIVE_MODE,
        bankroll
caps, or risk controls without explicit approval.

### KDP Books
Self-published books on Amazon Kindle Direct Publishing.
Current project: The Non-Developer's OpenClaw Playbook.

### TikTok Marketing Pipeline
Automated content pipeline for FeedFare marketing. Uses Claude to
generate scripts, TTS for voiceover, automated posting workflow.
```

```
## My Schedule
- Work Mon-Thu 9-6, Fri 8:30-5 (mobile electronics retail)
- Fly when schedule allows (369.5 total hours, commercial
      certificate)
- Most productive: mornings 7-11am PST
- Check Telegram throughout the day
- Prefer brief async updates over real-time interruptions

## Preferences
- Short answers by default, detailed when I ask for more
- Metric for currencies: USD always
- Date format: Month DD, YYYY
- When suggesting code changes: explain what the change does and
      why
- Flag anything that costs money or is irreversible before doing
      it

## Important Constraints
- Never npm/bun/pnpm install in ~/Documents/ (iCloud EDEADLK risk)
-
      Work copies go in ~/dev/ or /tmp/, never ~/Documents/ for
      active dev
- Never git push --force or delete branches
```

I learned this the hard way: the more specific you are in USER.md, the less you'll ever repeat yourself. I used to explain my project structure in every single session. I'd re-explain the iCloud restriction at least once a week after watching my agent cheerfully try to run npm install in a directory that would freeze everything. I even re-explained my own name. Now it just *knows*, because I wrote it down.

AGENTS.md: your crew

If you eventually run sub-agents (specialized agents for specific tasks), AGENTS.md is where you define them. This is more advanced, but here's a simple example of what it looks like:

```
# Sub-Agent Definitions

## research-agent
Role: Web research and data gathering
Tools: Tavily search, web fetch
When to invoke: Any task requiring current information from the
      web
Behavior: Summarize findings concisely, include sources

## code-agent
Role: Software development tasks
Tools: Full filesystem access, terminal execution
When to invoke: Any coding task for FeedFare or prediction-bot
Behavior: Explain changes before making them, commit with
      conventional
commits format
```

You don’t need this on day one. Get comfortable with the main identity files first.

The onboarding conversation

Here’s the part nobody tells you about: writing the files is only half the job. The other half is an onboarding conversation with your agent where you talk through things that are hard to capture in structured files.

Once your identity files are in place, start a fresh session and say something like:

> “Talos, I’ve set up your identity files. Read them, then ask me any clarifying questions you have. I want to make sure you actually understand my situation before we start working together.”

What happens next is worth paying attention to. A good agent will ask follow-up questions that reveal gaps you didn’t know were there. Mine asked about my flying schedule and how it should handle urgent messages when I might be in the cockpit. It asked whether my KDP books were technical or non-fiction (both, as it turns out). It asked about my risk tolerance for the Kalshi bot.

These are questions I should have anticipated in USER.md but hadn’t. The conversation surfaced them, and I added the answers directly to the file during the conversation.

Do this once when you set things up, and then repeat it whenever your life changes significantly. New project? Update USER.md and have the conversation. Changed jobs? Same thing. Your identity files should be living documents, not set-and-forget.

Teaching preferences without overwhelming the system

One mistake I see people make: they try to document everything at once. They write these massive USER.md files that cover every possible scenario, every preference, every exception. The file becomes a novel, and the agent gets confused trying to hold it all.

My approach: start lean, add specifics as they come up naturally.

When your agent does something you don’t like, don’t just correct it in the conversation. Add the preference to the appropriate file. Over a few weeks, your files naturally grow to cover the things that actually matter, not the things you imagined might matter.

For example, I didn’t write “use bullet points for lists” in my SOUL.md on day one. I added it after noticing my agent was writing long paragraph responses when I asked for quick summaries. One note in the file, problem solved permanently.

A good rule of thumb: if you’ve corrected the same behavior twice, it belongs in a file.

Creating your first reusable prompt library

You do not need a special prompt registry for this to work. The current, low-drama approach is to keep reusable prompts in a markdown file inside your workspace and tell the agent to read the one you want. Think of it as a prompt library, not a magic command.

Create a file like `~/.openclaw/workspace/prompt-library.md` and keep simple blocks in it:

```
## daily-briefing
Give me a quick rundown: weather in Merced, any urgent items in my
        workspace, Kalshi position status, and one thing I should
        handle today.

## flight-prep
Check weather along my planned route, flag delays or ugly winds,
        and tell me if this looks like a go/no-go day.

## end-of-day
Summarize what got done, what's pending, and what I should hit
        first tomorrow.
```

Then in chat, say something like:

> Read `prompt-library.md` and run the `daily-briefing` prompt.

What this does: gives you reusable instructions without depending on a CLI feature that might move around on you. The prompts live in plain text, you can edit them anytime, and your agent can pull the exact block you asked for.

These are tiny automations, but they add up fast.

Setting realistic expectations

Let me be straight with you: even with perfect identity files, there are things your agent won't be able to do well.

It will still make mistakes. It might misunderstand a request, give you outdated information, or confidently do the wrong thing. Identity files reduce this; they don't eliminate it.

It doesn't actually "learn" the way you do. When you add something to USER.md, the agent knows it next session. But between sessions? It doesn't accumulate experiences the way a human would. We'll get into this more in the memory chapter.

It can't do things outside its tools. If you haven't connected it to a service, it can't interact with that service. No amount of instruction in SOUL.md can make it read your email if email isn't set up.

What it *will* do, with proper identity files, is feel dramatically more like a real assistant and less like a search engine that talks back. The generic "how can I help you?" feeling disappears. In its place: something that feels like it actually knows you.

Reality check

By the end of your first 24 hours with identity files, you should be able to check off all of these:

- [ ] `~/.openclaw/workspace/USER.md` exists and includes your name, location, projects, and key preferences
- [ ] `~/.openclaw/workspace/SOUL.md` exists and defines your agent's name, communication style, and tone
- [ ] `~/.openclaw/workspace/IDENTITY.md` exists with role definition and any hard limits
- [ ] You've had the onboarding conversation and addressed any gaps
- [ ] You've added at least one preference to a file after seeing a behavior you didn't like
- [ ] Your agent refers to you by name without being prompted

If you're still getting generic responses that don't reference your projects or preferences, your identity files aren't loading. Verify:

```
ls ~/.openclaw/workspace/
```

You should see your files listed there. If they're in a different location, OpenClaw won't find them.

What's next

The identity files solve the "who are you?" problem. But there's a related problem we haven't touched: "what happened last Tuesday?"

Your agent knows who you are now. But it doesn't remember yesterday's conversation, last week's decisions, or the context behind ongoing projects. That's the memory problem, and it's what Chapter 3 is about.

Memory is where a lot of people give up because it seems complicated. I promise you it's not. Once you understand what's actually happening under the hood, it's just file organization. Pilot stuff, essentially.

Troubleshooting

“My agent still doesn’t know my name or projects”

First, verify the files are in the right place:

```
ls ~/.openclaw/workspace/
```

If the files exist but the agent isn’t reading them, check whether your workspace path is configured correctly:

```
openclaw config get workspace
```

The output should match where your files live. If not:

```
openclaw config set workspace ~/.openclaw/workspace
```

Then restart your agent:

```
openclaw restart
```

“The agent’s personality hasn’t changed”

SOUL.md changes take effect at the start of a new session, not in the middle of an ongoing conversation. Close your current chat and start fresh. If you’re using Telegram, try `/reset` to start a new session.

“My agent ignores the boundaries I set in IDENTITY.md”

Boundaries in IDENTITY.md work best when they’re specific and unambiguous. “Don’t do risky things” is too vague. “Never execute git push –force or any command with –force flag” is something the agent can actually check against. Rewrite vague rules as concrete ones.

“I wrote a huge USER.md and now responses seem scattered”

Long files can dilute focus. Try breaking USER.md into sections with clear headers and keeping each section tight. If the file is over 500 lines, consider splitting project-specific details into separate files (like `FEEDFARE.md`, `KALSHI.md`) and referencing them from USER.md. The agent can be told to load specific files when working on specific projects.

“The onboarding conversation just feels like normal chatting”

Push back on surface-level responses. Ask your agent directly: “What aspects of my situation are you still unclear on?” or “What would you need to know to handle my projects without asking me every time?” Good agents will surface the gaps. If it says it has everything it needs, test it: ask it to explain your main projects back to you in its own words. The gaps will show themselves fast.