FEBRUARY 2026 EDITION

Ofir's Claude Code Field Manual

A practical guide for non-developers who want to build real things — dashboards, websites, automations — without writing code or waiting for IT.

Ofir Bloch | 10 Project Playbooks | Zero Code Required

What's inside (and who it's for)

This guide is for marketing people, customer success managers, sales ops leads, founders, and anyone else who has ideas and projects but no programming background. You've been told you need a developer to build things. That changed. Now it's mostly not true.

Claude Code is a tool that builds things when you describe what you want in plain English. Websites. Dashboards. Automated reports. Internal tools. The kinds of things that used to require a developer, a sprint planning meeting, and six weeks of "it's in the backlog."

This guide starts with the easy stuff (Cowork, which works like chatting with a very competent assistant) and progresses to the powerful stuff (Claude Code Desktop, which actually builds and deploys real software). You don't need to know what a terminal is. You don't need to know what "git" means. I'll explain every term the first time it shows up.

CONTENTS

No coding experience required. Seriously.

LEGAL NOTICE

This guide is for informational and educational purposes only. The author is not affiliated with Anthropic beyond being a user. All product names and features are based on publicly available information as of February 2026 and may change.

Results will vary. The author assumes no liability for outcomes. Do not use AI tools for sensitive data without consulting a developer and legal counsel.

Table of Contents

PART ONE

What Is This Thing?

No jargon. No hype. Just what it does and why you'd care.

- The one-paragraph version
- What's the difference between Claude, Cowork, and Claude Code?
- Why should a non-developer care?
- What you need before we start

PART 1 — WHAT IS THIS THING?

The one-paragraph version

Anthropic (the company that makes Claude, the AI you might already chat with) built a tool called Claude Code. It does more than answer questions. It can actually build things: websites, apps, dashboards, automated workflows, internal tools. You describe what you want in plain English, and it writes the code, creates the files, tests them, and can even put the finished product on the internet. You don't write code. You describe outcomes.

What's the difference between Claude, Cowork, and Claude Code?

Think of them as three levels of the same AI:

	CLAUDE CHAT	COWORK	CLAUDE CODE
What it is	AI chatbot	Claude with hands	Claude with a workshop
What it does	Answers questions, writes text, analyzes documents	Creates files, builds spreadsheets, processes data on your computer	Builds full apps, websites, dashboards, deploys to the internet
What it can't do	Touch your files or build anything	Put things on the internet or connect to databases	Nothing major — this is the full toolkit
Who it's for	Everyone	Anyone who works with documents and data	Anyone who wants to build real tools

This guide covers Cowork and Claude Code. You'll start with Cowork (easy, low-risk, useful immediately) and graduate to Claude Code (powerful, slightly more involved, life-changing for your team's velocity).

Why should a non-developer care?

Because the bottleneck in your work has never been ideas. It's been execution. You know exactly what dashboard your team needs. You know the landing page that would convert better. You know the weekly report that should be automated. But every one of those things requires "dev resources." And dev resources are always allocated to something else.

Claude Code changes the math. Not for everything. But for a surprising number of the things that are stuck in someone's backlog, you can now build them yourself. In hours, not sprints.

REALITY CHECK

Claude Code won't replace your dev team. But it will let you stop waiting for things that don't actually need a developer. That distinction is worth understanding.

What you need before we start

- A computer (Mac or Windows). Chromebook works for Cowork only.
- A Claude account on a paid plan. Pro ($20/month) works. Max ($100/month) is better if you'll use this daily.
- The Claude Desktop app installed. Download it at claude.ai/download.
- Patience for about 30 minutes of setup.
- No coding experience. No terminal experience. No git experience. None of that.

PART TWO

Start Here: Cowork

Your comfort zone. No code. No terminal. Just results.

- What Cowork is
- How to open Cowork
- Your first task: messy doc to clean report
- Ten things Cowork is great at
- Cowork limitations

PART 2 — START HERE: COWORK

What Cowork is

Cowork is a feature inside the Claude Desktop app. You open it by clicking the "Cowork" tab at the top of the app. It looks like a regular Claude chat, but it can do things regular Claude can't: access files on your computer, create documents, build spreadsheets with working formulas, run long tasks in the background, and work on multiple things at once.

JARGON BUSTER

Sandbox — A secure, protected area on your computer where Cowork does its work. It can't accidentally delete your important files or do anything destructive. Think of it as a playpen for AI.

How to open Cowork

1. Open the Claude Desktop app (the orange icon in your dock or taskbar).
2. Look at the top of the app. You'll see tabs: **Chat** , **Code** , and **Cowork** .
3. Click **Cowork** .

That's it. You're in.

WORTH KNOWING

If you don't see the Cowork tab, make sure your Claude Desktop app is updated. Go to claude.ai/download and install the latest version.

Your first task: turn a messy document into a clean report

Let's do something practical. Say you have a messy Google Doc full of notes from a customer call, and you need to turn it into a clean summary for your team.

1. In Cowork, drag the file into the chat area. Or click the paperclip icon and select it.
2. Type your prompt (feel free to copy and paste it).
3. Press Enter (or click the send button).
4. Cowork will read the file, extract the important parts, and create a formatted Word document.
5. When it's done, you'll see a download link. Click it to save the file.

COPY THIS EXACTLY

"Read this document. Extract the key points, decisions made, and action items. Create a clean, formatted summary document with sections for: Key Takeaways, Decisions, Action Items (with owners if mentioned), and Open Questions. Make it professional enough to share with my team."

Ten things Cowork is great at

These are real tasks non-developers use Cowork for every day:

1. **Clean up messy documents** — Turn raw notes into polished reports, briefs, or summaries.
2. **Build spreadsheets with formulas** — Describe the logic you need; Cowork writes the formulas.
3. **Analyze data from files** — Drop in a CSV and ask questions about it.
4. **Create presentation outlines** — Draft slide decks from research or meeting notes.
5. **Write and format long documents** — SOPs, handbooks, proposals with consistent formatting.
6. **Compare documents** — Find differences between contract versions or policy updates.
7. **Extract data from PDFs** — Pull tables, figures, and key data from PDF reports.
8. **Generate charts and visualizations** — Basic charts from your data, downloadable as images.
9. **Batch-process files** — Apply the same transformation to 50 files at once.
10. **Draft email sequences** — Write multi-touch campaigns with consistent voice.

KNOW THE LIMITS

Cowork works with files, not the internet. It can't deploy websites, connect to databases, or pull live data. When you hit those walls, move to Claude Code Desktop (Part 3). Until then, stay here — simpler is better.

PART THREE

Level Up: Claude Code Desktop

Where you start building things that live on the internet.

- What Claude Code Desktop is
- Opening it, step by step
- The three modes (Plan / Code / Act)
- The permission system
- The diff view
- Your first project: a landing page
- The 5-minute project setup
- Your project's memory: CLAUDE.md
- Skills: reusable workflows
- Chrome integration
- Running multiple sessions

PART 3 — LEVEL UP: CLAUDE CODE DESKTOP

What Claude Code Desktop is

Claude Code Desktop is the "Code" tab in the same Claude Desktop app you already have. When you click it, you get a different interface. Instead of chatting about files, you're working inside a project. Claude can create files, organize them into folders, run them, test them, and deploy them. It's the difference between writing a recipe (Cowork) and actually cooking the meal (Claude Code).

JARGON BUSTER

Project folder — A folder on your computer where all the files for one project live. Think of it as a filing cabinet drawer dedicated to one thing. Claude Code works inside this folder.

Opening Claude Code Desktop, step by step

1. Open the Claude Desktop app.
2. Click the **Code** tab at the top.
3. It will ask you to select a folder. This is where your project will live.
4. If you don't have a folder yet: On Mac, open Finder, go to Documents, right-click and choose New Folder. Name it something like **"my-first-project"** (no spaces, use dashes instead). On Windows, open File Explorer, go to Documents, right-click and choose New > Folder.
5. Select that folder in Claude Code Desktop.
6. You'll see a chat-like interface with your project name at the top. This is your workspace.

PREVENT A HEADACHE

Use dashes instead of spaces in folder names. "my-project" works. "My Project" might cause problems later. It's a small thing that prevents headaches.

The three modes

Claude Code Desktop has three modes. You switch between them using the dropdown in the chat area.

Mode	What it does	When to use
Plan	Claude reads your files and describes what it would build.	Always start here. Review before building.
Code	Claude creates and edits files, asks permission for each action.	After you've approved the plan. Your main building mode.
Act	Claude runs commands, installs tools, deploys. More autonomy.	Only when you're comfortable and have safeguards in place.

DO NOT SKIP THIS

Start in Plan mode. Always. Describe what you want. Let Claude explain how it would build it. Review that plan. Then switch to Code mode and say "Go ahead." This saves you from Claude enthusiastically building the wrong thing.

The permission system (your safety net)

In Code mode, every time Claude wants to create a file, change a file, or run a command, it shows you what it's about to do and asks: allow or deny? You get three choices:

OPTION	WHAT IT MEANS
Allow Once	Do this one thing, then ask me again next time.
Allow Always	Do this type of thing without asking for the rest of this session.
Deny	No. Don't do that.

This means Claude literally cannot do anything without your say-so. If you see something you don't understand, click Deny. Nothing bad happens. Claude will explain what it was trying to do and suggest an alternative.

WORTH KNOWING

When in doubt, use "Allow Once." You can always switch to "Allow Always" later once you're comfortable with what Claude is doing. Start cautious.

The diff view (seeing what changed)

After Claude makes changes, you'll see a "diff" view. This shows you exactly what was added (highlighted in green) and what was removed (highlighted in red). You don't need to understand the code. You're just looking for:

- **Does the amount of change seem right?** If you asked for a small fix and see 500 lines changed, something's off.
- **Are there any red lines that look important?** Claude sometimes removes things it shouldn't.

> **JARGON BUSTER**
>
> **Diff view** — A display that shows what changed in a file. Green lines were added. Red lines were removed. It's like Track Changes in Word, but for code.

If something looks wrong, you can undo it. Claude Code Desktop saves a snapshot before every change. Click the rewind button in the toolbar to go back.

> **THIS WILL BREAK THINGS**
>
> Never switch to Act mode unless you understand what your project contains. Act mode gives Claude more autonomy. It's for experienced users who have set up proper safeguards.

Your first project: a simple landing page

Let's build something real. A one-page website for a product, event, or team.

1. Open Claude Code Desktop. Select your empty project folder.
2. Start in **Plan mode**. Type your prompt describing the landing page you want.
3. Claude will describe the plan. If you see unfamiliar terms, type: "Explain [term] like I'm in high school."
4. Once the plan makes sense, switch to **Code mode**.
5. Type: "Go ahead and build it. After each major step, stop and show me what it looks like."
6. Claude will create files and tell you how to preview the site. Usually a link like http://localhost:3000. Click it.
7. Look at the preview. Tell Claude what to change.
8. When you're happy, say: "How do I put this on the internet so other people can see it?"

The 5-Minute Project Setup

Why this matters

Most frustration with Claude Code happens in the first 10 minutes. Wrong folder. No preferences saved. Claude builds before you're ready. Files get mixed up. You don't know when to call it done.

This checklist prevents all five of those problems. It takes 5 minutes. Do it once per project.

> **SAVE YOURSELF 15 MINUTES**
>
> These steps take 5 minutes. They prevent the top 5 mistakes: wrong folder structure, Claude forgetting preferences, building without a plan, mixing up files, and not knowing when to stop.

The checklist

Step 1: Create your project folder. Give it a short, clear name using dashes instead of spaces:

```terminal
my-webinar-page
```

Not `My Webinar Page` or `webinar final v2 FINAL`. Dashes, lowercase, no spaces. This prevents a whole category of technical problems.

Step 2: Open Claude Code in that folder.

```terminal
cd my-webinar-page && claude
```

"cd" means "change directory" (go to this folder). "&&" means "then do this next." Or in Claude Code Desktop: use File > Open Folder.

Step 3: Run /init to generate a CLAUDE.md. Type `/init` and hit Enter. Claude analyzes your folder and creates a starter CLAUDE.md file. Open it, review it, and add your preferences.

Step 4: Create a .claudeignore file. This file tells Claude which files to ignore completely. Just type:

```prompt
Create a .claudeignore file that ignores
node_modules, .env files, and any files with
"secret" or "key" in the name.
```

Step 5: Start in Plan Mode. Press Shift+Tab twice to switch to Plan Mode. Now describe what you want to build:

```prompt
I want to build a landing page for our Q3 webinar.
Interview me about what I need before you start.
```

Claude asks questions, you answer, and together you create a plan before any building happens. (See the Interview-Plan-Build Method for the full workflow.)

Step 6: Define "done." Before Claude starts building, tell it what success looks like:

```prompt
This project is done when:
- The page loads without errors
- The registration form submits to our Google Sheet
- The design matches our brand colors
- It looks good on mobile phones
```

Without a "done" definition, projects drift. Claude keeps adding features you didn't ask for, and you're never sure when to stop.

MINI CHECKUP

- ☐ I have a project folder with a clear, dashed name
- ☐ I ran /init and customized my CLAUDE.md
- ☐ I have a .claudeignore file
- ☐ I started in Plan Mode
- ☐ I defined what "done" looks like

Your Project's Memory: The CLAUDE.md File

What it is (plain English)

CLAUDE.md is a text file you put in your project folder. Claude reads it automatically at the start of every session. Think of it as a sticky note on your desk that Claude checks before every conversation.

Without it, every new session starts from zero. Claude forgets your brand colors. It forgets your preferences. It forgets you told it to "always ask before deleting anything." You spend the first five minutes of every session re-explaining yourself.

Why it matters

Here's the problem: Claude has no memory between sessions. When you close a session and open a new one, it's a blank slate. Every preference, every correction, every "no, use our blue, not that blue" is gone.

CLAUDE.md solves this. It's the one file Claude reads first, every time, automatically.

> ⚠ **GOLDEN RULE**
>
> If you change one thing about how you use Claude Code, make it this: create a CLAUDE.md file. Users report "night and day" quality differences.

How to create one

You have two options. Both take less than two minutes.

Method 1: Let Claude generate one. Type `/init` in Claude Code and hit Enter. Claude analyzes your project folder and creates a CLAUDE.md with smart defaults. Review it, then customize with your preferences.

Method 2: Ask Claude to create one from scratch.

```prompt
Create a CLAUDE.md file for this project.
My brand colors are #2563EB (blue) and #F59E0B (gold).
Our company name is always written as "YourCompany" (one word).
Our tone is professional but warm.
Always ask before deleting files.
Always show me a plan before making changes.
```

TWO MINUTES NOW, HOURS SAVED LATER

Without a CLAUDE.md, you'll spend 5-10 minutes per session re-explaining preferences. Over 5 sessions, that's nearly an hour of repeating yourself. A CLAUDE.md takes 2 minutes to set up.

The non-developer template

Every CLAUDE.md template online is written for software developers. Here's one for the rest of us:

CLAUDE.md

```
# Project: [Your Project Name]

## What This Project Is
[One sentence: "A landing page for our Q3 webinar" or
"A dashboard showing monthly sales data"]

## My Preferences
- Brand colors: [hex codes or color names]
- Font preferences: [if any]
- Tone of voice: [professional, casual, playful, etc.]
- My company name: [name and how to spell it]

## Important Rules
- Always ask before deleting any files
- Show me a plan before making changes
- Use simple, non-technical language in any text content

## Files That Matter
- [List key files and what they do, once your project grows]
```

Copy this, fill in the blanks, save it as CLAUDE.md in your project folder. That's it.

The "keep it short" rule

CLAUDE.md works best under 150 lines. If it gets longer, Claude starts ignoring parts of it (the same way you skim a long email). Think one-page cheat sheet, not a manual.

If you need to reference longer documents (a brand guide, a style sheet), point to the file instead of pasting it in:

CLAUDE.md

```
For brand guidelines, see @brand-guide.md
```

WARNING

Don't paste your entire brand guide into CLAUDE.md. Point to the file instead. Pasting long documents into CLAUDE.md makes Claude less effective, not more. It fills up Claude's memory before you've even started working.

I LEARNED THIS THE HARD WAY

My first CLAUDE.md had 12 rules with full justifications for each one. Claude ignored half of them. Now I keep it to five or six rules, written as direct commands: "Always ask before deleting files." Not "It's important to ask before deleting files because..." Delete the justifications. Claude doesn't need to know WHY — it needs to know WHAT.

When to update it

Update your CLAUDE.md after any session where you corrected Claude on something it should have known.

If you said "no, use our blue, not that blue" three times, add your hex code to CLAUDE.md so you never have to say it again. If Claude kept using a word you hate ("leverage," "utilize"), add a rule: "Never use the words leverage or utilize."

Your CLAUDE.md should grow slowly, one correction at a time. Every update makes every future session better.

COMMUNITY TIP

"I restructured my context files into smaller, focused files instead of one big CLAUDE.md. It improved efficiency by 40-60%." — Reddit user, r/ClaudeCode. This is an advanced move. Start with a single CLAUDE.md. Once your project grows beyond 100 lines, you can split it into focused files.

CLAUDE.md vs Skills

Not sure if something belongs in CLAUDE.md or a Skill? The Skills section has a decision table to help you choose.

MINI CHECKUP

- ☐ I created a CLAUDE.md in my project folder (or ran /init)
- ☐ It includes my brand preferences and project description
- ☐ It's under 150 lines
- ☐ I know to update it when Claude keeps making the same mistake

Skills: Reusable Workflows Claude Remembers

What Skills are

Skills are saved recipes that Claude pulls up when it recognizes a matching task. Unlike CLAUDE.md (which Claude reads every time), Skills only load when they're relevant.

Why they matter

You create a Skill once. From then on, Claude uses it automatically whenever a matching task comes up. Built a great process for creating landing pages? Save it as a Skill. Next time you ask for a landing page, Claude follows the same process without you repeating the instructions.

Skills also save space in Claude's memory. A detailed 50-line workflow in CLAUDE.md loads every session whether you need it or not. As a Skill, it loads only when relevant.

Where Skills live

Skills can be personal (available across all your projects, just for you) or project-level (shared with anyone working on this project). Most people start with personal Skills. Claude handles the file locations.

How to create your first Skill

Step 1: Ask Claude to create it.

prompt

```
Create a skill called brand-check in my personal skills folder.
It should check any content against our brand guidelines.
Our primary color is #2563EB, our accent is #F59E0B,
our tone is professional but warm, and our company name
is always written as "YourCompany" (one word, capital Y and C).
```

Step 2: That's it. Type `/skills` and hit Enter to confirm it worked. You can also invoke it directly by typing `/brand-check`.

Updating and deleting Skills. To update: "Update my brand-check skill to add our new secondary color #10B981." To delete: "Delete my brand-check skill." Claude handles the file changes.

If Claude seems to ignore your Skill, invoke it directly by typing its name as a slash command: `/brand-check`.

UNDER THE HOOD

Inside, the Skill file has a description (which tells Claude when to use it) and a checklist (which tells Claude what to do). You'll never need to edit this file by hand. Just tell Claude what to change.

SAVE YOURSELF 20 MINUTES

Create a Skill for any workflow you've explained to Claude more than twice. The 5 minutes creating the Skill saves 10+ minutes per future session.

CLAUDE.md vs Skills: the decision rule

PUT IT IN CLAUDE.MD	MAKE IT A SKILL
Stable, always-true rules	Specialized workflows
Brand colors, naming conventions	"How to check brand compliance"
"Always ask before deleting"	"How to format a weekly report"
Project description	"How to structure a landing page"

Rule of thumb: if it applies to every task, put it in CLAUDE.md. If it applies to a specific type of task, make it a Skill. CLAUDE.md is your constitution. Skills are your recipes.

Skills vs MCP: capabilities vs behavior

You'll meet MCP (Model Context Protocol) later in this guide. Here's the difference in one sentence: **MCP gives Claude the ability to use your tools. Skills teach Claude how to use them well.**

An MCP connection to Google Drive lets Claude create, read, and edit files. A Skill teaches Claude how to write a great product requirements doc inside a Google Drive doc — the structure, the best practices, the sections to include. MCP is the hands. Skills are the brain. You'll often use both together.

WORTH KNOWING

MCP loads all of its tool definitions into Claude's memory the moment you start a session. Skills only load their name and description up front — the full instructions load when Claude actually needs them. This is why connecting too many MCP servers causes context bloat, while having dozens of Skills barely costs anything.

MCP (THE HANDS)	SKILLS (THE BRAIN)
Gives Claude access to external tools	Teaches Claude how to do a task well
Runs a server that connects to APIs	A folder of instructions Claude reads
Everything loads at startup	Only loads when relevant
"Read this Notion doc"	"Write a great PRD in this Notion doc"

Manual-only Skills (the safety valve)

Some Skills shouldn't run automatically. Anything that publishes to a live site, sends an email, or modifies customer data should only run when you explicitly ask for it. Add one line to the top of the Skill file:

SKILL.md

```
disable-model-invocation: true
```

Now Claude won't use this Skill automatically. You have to type the slash command to trigger it yourself.

Skills from other people

You can download Skills others have created. Treat them like apps from unknown developers: check what access they request. Look for this line in the SKILL.md file:

SKILL.md

```
allowed-tools: Bash(*)
```

That means the Skill has full access to run any command on your computer. Be cautious. A Skill with `allowed-tools: Read, Grep` is much safer.

WARNING

Treat Skills from GitHub or strangers like apps from unknown developers. Check what tools they request access to. A Skill with `allowed-tools: Bash(*)` means full computer access.

REAL EXAMPLE

"I created a 'weekly-report' Skill that formats our team's status updates. Before: I'd spend 10 minutes each week re-explaining the format. After: I just say 'generate the weekly report' and Claude follows the exact same template every time."

MINI CHECKUP

- ☐ I understand the difference between CLAUDE.md and Skills
- ☐ I created at least one Skill for a workflow I repeat
- ☐ I know Skills live in .claude/skills/ (project) or ~/.claude/skills/ (personal)
- ☐ I know to review tool access in Skills from external sources

Chrome Integration: See What Claude Builds, Instantly

What it does

Claude Code can open your Chrome browser, look at what it built, read error messages, and fix problems automatically. Instead of you switching back and forth between Claude and Chrome to check the output, Claude does the checking itself. This feature is in beta.

Why it matters

Without Chrome integration: Claude builds a page. You open it in Chrome. You notice a broken button. You describe the problem to Claude. That's four context switches and a lot of typing.

With Chrome integration: Claude builds a page, opens it in Chrome, spots the broken button, fixes it, and verifies the fix. You watch.

REAL EXAMPLE

"I told Claude to build a landing page, then said 'Open it in Chrome and check for mobile layout issues.' Claude opened the page, identified that the hero image wasn't scaling on mobile, fixed the layout code, and refreshed. Total time: 90 seconds."

What you need before starting

- **Google Chrome** (or Microsoft Edge). Not supported on Brave, Arc, or other browsers yet.
- **Claude in Chrome extension** from the Chrome Web Store (version 1.0.36 or higher)
- **Claude Code** version 2.0.73 or higher
- A direct Anthropic plan (Pro, Max, Teams, or Enterprise)

How to set it up

Step 1: Install the Claude in Chrome extension from the Chrome Web Store.

Step 2: Start Claude Code with the Chrome flag:

```terminal
claude --chrome
```

Step 3: Inside the session, type `/chrome` to confirm the connection is working.

Step 4: Now you can say things like: "Open the landing page in Chrome and check if everything looks right."

SAVE YOURSELF 15 MINUTES

To avoid typing `--chrome` every time, run `/chrome` inside a session and select "Enabled by default." Chrome integration will be on whenever you start Claude Code.

What Claude can do in the browser

Once connected, Claude can:

- **Debug live pages** — read console errors and fix the code that caused them
- **Check designs** — build a UI, then open it to verify it matches your mockup
- **Test forms** — submit forms with test data to check if validation works
- **Work with web apps you're logged into** — Google Docs, Notion, Gmail, your CMS
- **Extract data** — pull information from web pages and save it as spreadsheets

Important: Claude sees what you see

Chrome integration shares your browser's login state. If you're logged into Gmail, Google Docs, or your company admin panel, Claude can see those tabs too. This is powerful and risky.

> **WARNING**
> Chrome integration shares your browser's login state. If you're logged into your company's admin panel, Claude can see it. Use a dedicated browser profile for Claude Code work.

Safety patterns

Follow these four rules:

1. **Keep it off by default.** Only turn it on for the specific task that needs browser access.
2. **Use a dedicated browser profile.** Create a Chrome profile just for Claude Code work.
3. **Restrict site permissions.** In the Claude in Chrome extension settings, limit which sites Claude can interact with.
4. **Expect pauses.** Login pages and CAPTCHAs require you to handle them manually.

Troubleshooting

If the connection drops, type `/chrome` and select "Reconnect extension." If Chrome becomes unresponsive, restart Chrome and reconnect. If Claude says "Extension not detected," make sure the extension is enabled in `chrome://extensions`.

> **WORTH KNOWING**
>
> Chrome integration adds to your context usage. Every page Claude reads in the browser takes up space in its memory (see Context Rot). Keep it off when you don't need it.

MINI CHECKUP

- □ I installed the Claude in Chrome extension
- □ I know how to start Claude Code with --chrome and use /chrome
- □ I set up a dedicated browser profile for Claude Code
- □ I restricted site permissions in the extension settings
- □ I keep Chrome integration off when I don't need it

COPY THIS EXACTLY

"I want to build a one-page website for [your product/event/team]. The page should have: a hero section with a headline and subheadline, a section explaining what this is, a section with three key benefits, a testimonial or quote section, and a call-to-action button. Use a modern, clean design. My brand colors are [your colors]. Before building anything, explain your plan step by step."

Claude will walk you through deployment. For most simple sites, it will suggest Vercel or Netlify.
Both are free for basic use. Claude handles the setup.

I LEARNED THIS THE HARD WAY

The first time I used Act mode, Claude helpfully reorganized my entire project folder structure "for clarity." It took an hour to undo. Stick with Code mode until you know what you're doing.

Running multiple sessions

The sidebar in Claude Code Desktop shows your sessions. Each session is an independent workspace. You can have one session building a website and another session fixing a spreadsheet automation. They don't interfere with each other.

Click "+ New session" to start a new one.

WORTH KNOWING

Keep different projects in different folders and different sessions. Don't try to build two unrelated things in the same session. Claude gets confused when the context is mixed.

PART FOUR

Talking to Claude So It Actually Gets It

The difference between a frustrating 45 minutes and a working prototype in 10 isn't the tool. It's how you ask.

- Bad ask vs. good ask (with examples)
- The five rules for good prompts
- The interview-plan-build method
- The prompt template
- How to give feedback mid-project
- The loop problem and how to fix it

PART 4 — TALKING TO CLAUDE SO IT ACTUALLY GETS IT

The single most important skill in this guide

How you ask matters more than what you ask. The same request produces wildly different results depending on how you phrase it. This is true for Cowork and Claude Code.

Think of Claude as a very talented freelancer who just started working with you today. They're smart. They're fast. But they know nothing about your company, your preferences, or your definition of "good." The clearer your brief, the better the work.

Bad ask vs. good ask

The bad version

bad-prompt.txt

```
"Make me a landing page."
```

What kind? For whom? What should it do? What color? Claude will guess. It will guess wrong.

The good version

good-prompt.txt

```
"I need a landing page for our Q2 webinar on AI in marketing.
The audience is marketing directors at mid-size companies.
It should have: a hero with the event title and date,
a section with three speaker bios, a registration form,
and a FAQ section. Use our brand colors: navy (#1B2332)
and teal (#0987A0). Keep it minimal and professional.
Before building anything, explain your plan."
```

Same request. Ten times better output. The extra two minutes of writing save an hour of back-and-forth.

REALITY CHECK

You don't need to learn "prompt engineering." You need to communicate clearly. If you can write a decent brief for a freelancer, you can prompt Claude.

The five rules

1. Start with the outcome, not the method

Say "I need a dashboard that shows our monthly sales by region" not "Build me a React app with a Recharts component." You don't need to know the tools. Claude does.

2. Describe your audience

Who will use this? Your boss? Your customers? Your ops team? Claude makes different design decisions depending on the answer.

3. List what you need, specifically

Vague: "Make it look good." Specific: "Use our brand colors. No more than three sections. Include a call-to-action button above the fold."

4. Say what you DON'T want

Claude is enthusiastic. Left unsupervised, it will add features you didn't ask for. If you want something simple, say so: "Keep this minimal. No extra features beyond what I described."

5. Ask for a plan before execution

Always start with: "Before building anything, explain your plan." Review the plan. Adjust. Then give the green light. This alone prevents most bad outcomes.

WORTH KNOWING

Claude remembers everything in the current session but starts fresh each time you open a new one. That's why your CLAUDE.md file (page 16) is so important. It automatically loads your preferences into every new session so you don't have to repeat yourself.

WORTH KNOWING

Keep a document with prompts that worked well. When you find a phrasing that gets great results, save it. You'll reuse it more than you think.

YOU DON'T EVEN NEED TO TYPE

I use Wispr Flow to speak to Claude Code directly — I just talk, and it types. Mac also has built-in dictation (press the microphone key twice). If typing feels like a barrier, remove it. The quality of your prompts comes from clarity of thought, not typing speed.

The Interview-Plan-Build Method

The problem with "just build it"

When you say "build me a landing page," Claude makes hundreds of decisions without asking. What color is the header? Where does the form go? What happens when someone clicks submit? Claude will answer all of these questions itself. And many of those answers won't match what you had in mind.

Then you spend 30 minutes fixing things that were wrong from the start. There's a better way.

The three-step slowdown

Step 1: DESCRIBE (let Claude interview you)

Instead of jumping straight to building, ask Claude to ask YOU questions first:

```prompt
I want to build a landing page for our Q3 webinar.
Before you write any code, interview me.
Ask me questions about what I need, what it should
look like, who will use it, and anything else you
need to know.
```

Claude will ask things like: What's the webinar topic? Who's the audience? Do you have brand colors? This takes 3-5 minutes and eliminates 80% of wrong decisions.

THE ONE PROMPT THAT CHANGES EVERYTHING

Ask Claude to "interview me about what I need before you start building." This single prompt eliminates the most common source of wasted rebuilds.

Step 2: PLAN (make Claude show its work)

After the interview, tell Claude to write a plan:

prompt

```
Based on our conversation, write a plan.
List every step you'll take, in order.
Save it as PLAN.md so we can reference it.
```

Claude creates something like this:

PLAN.md

```
# Plan: Q3 Webinar Landing Page

## Steps
1. Create page structure: hero section with title, subtitle, and CTA button
2. Add speaker section with headshots and bios
3. Build registration form (name, email, company)
4. Add agenda section with time slots
5. Style everything using brand colors (#2563EB, #F59E0B)
6. Make it mobile-friendly
7. Add form validation (check that email is real)
```

Read it. Does anything look wrong? Is something missing? Fix it now, before Claude writes a single line of code. Changing a plan takes 10 seconds. Changing finished work takes 10 minutes.

Step 3: BUILD (one step at a time)

Now tell Claude to execute the plan, but one step at a time:

```prompt
Execute step 1 of the plan.
Stop after completing it so I can review
before you move to step 2.
```

Claude builds the hero section. You look at it. It's good? Say "looks good, move to step 2."
Something's off? Fix it now, before the rest of the page is built on top of it.

> **GOLDEN RULE**
> Never let Claude build without showing you a plan first. The 5 minutes reviewing a plan saves 30 minutes fixing wrong decisions.

Why step-by-step matters

Think of it like building a house. You wouldn't let a contractor finish the whole house before checking if the foundation is level. Each step is a checkpoint. Problems caught early are easy to fix. Problems caught late mean tearing things apart.

Real example: webinar landing page

The interview prompt:

```prompt
I want to build a landing page for our Q3 webinar
called "AI for Marketing Teams." Before you build
anything, interview me about what I need.
```

Claude asks (abbreviated): What's your brand? Colors? Logo? Who's the target audience? How many speakers? What information should the registration form collect?

The plan output (saved as PLAN.md):

PLAN.md

```
1. Hero section: Title, date, CTA button
2. Speaker cards: 3 speakers with photos and bios
3. Agenda: 4 time blocks with descriptions
4. Registration form: name, email, company, role
5. Brand styling: blue/gold palette, Inter font
6. Mobile responsive layout
7. Form sends data to Google Sheets
```

Step-by-step execution: "Execute step 1 of the plan. Stop when done." You review. Approve. Move to step 2. By step 7, you have a finished page and you approved every decision along the way.

Permission mode cycling

Here's a handy trick. Press Shift+Tab twice to cycle Claude into Plan Mode. In Plan Mode, Claude describes what it would do instead of doing it. This is perfect for Step 2 of the method.

When you're ready for Claude to start building, press Shift+Tab again to switch back to normal mode.

REAL EXAMPLE

"I used to say 'build me a dashboard' and spend an hour fixing it. Now I start with the interview prompt, get a plan, and execute step by step. My dashboards come out right the first time, and the whole process takes less time than the old fix-it loop."

MINI CHECKUP

- ☐ I know the three steps: Describe, Plan, Build
- ☐ I start projects with "interview me" instead of "build me"
- ☐ I ask Claude to save plans as PLAN.md
- ☐ I execute plans step-by-step instead of all at once
- ☐ I know Shift+Tab cycles between Plan Mode and normal mode

PART FIVE

The Safety Stuff

Five minutes now saves a very bad day later.

- The permission system protects you
- Protect your sensitive files
- Undo button: checkpoints
- The golden rules
- Lock your doors: security & permissions

PART 5 — THE SAFETY STUFF

Claude Code can create files, delete files, and run programs on your computer. That's what makes it powerful. It's also why you need to understand the guardrails.

The permission system protects you

In Code mode (the default), Claude asks before doing anything destructive. It shows you what it's about to do, and you approve or deny. This is your safety net. Use it.

> ⚠ **DO NOT SKIP THIS**
>
> If Claude suggests running a command you don't understand, ask: "Explain what this command does in plain English before I approve it." Never approve something you don't understand.

Protect your sensitive files

Claude reads every file in your project folder by default — including passwords and API keys. The "Lock Your Doors" section on the next page shows you exactly how to block access. If you followed the 5-Minute Setup, you already have a .claudeignore file. Now add permission deny rules for full protection.

Undo button: checkpoints

Claude Code saves a snapshot of your files before every change. If something goes wrong, click the rewind button in the toolbar. It restores your files to the state before Claude's last change.

> ⚠ **DO NOT SKIP THIS**
>
> Rewind only undoes file changes on your computer. If Claude did something "out in the world" — like deploying a website or sending an API request — rewind cannot take that back.

The golden rules

1. **Start in Plan mode. Always.** (See the Interview-Plan-Build Method.)
2. **Use "Allow Once" until you're comfortable.**
3. **Lock your sensitive files.** See the next section for deny rules.
4. **Never approve a command you don't understand.** Ask Claude to explain it first.
5. **If anything feels wrong, hit Stop.** Nothing bad happens when you stop.

Lock Your Doors: Security & Permissions

The one thing you must do

Block Claude from reading files that contain passwords, API keys (codes that give access to paid services), and credentials (login information). These files exist in most projects, and Claude reads them by default.

Open Claude Code and type `/permissions`. Then add these deny rules:

permissions

```
Deny: Read(.env*)
Deny: Read(.ssh/*)
Deny: Read(.aws/*)
Deny: Read(**/*key*)
Deny: Read(**/*secret*)
```

The exact format may vary by Claude Code version. If these don't work as typed, run `/permissions` and Claude will walk you through setting deny rules interactively.

What these mean in plain English: `.env` files are where most passwords and API keys live. `.ssh` files contain keys that give access to remote servers. `.aws` files contain Amazon cloud service credentials. Files with "key" or "secret" in the name almost always contain sensitive data.

GOLDEN RULE

If you do one thing: block access to .env files and secrets paths. This 2-minute setup prevents the most commonly reported security incident.

Understanding the three permission levels

Allow means Claude does it without asking. Use this for safe, everyday commands you trust completely.

Ask means Claude pauses and asks for your approval each time. Use this for commands that could change things outside your project.

Deny means Claude can never do it, period. Use this for anything involving sensitive files or dangerous operations. Deny rules always win.

Three layers of protection

Think of your security like a building. Three layers, each doing a different job.

Layer 1: File boundaries. You already know about `.claudeignore` (the "don't even look at this" list). Add permission deny rules on top. The two work together: `.claudeignore` hides files completely, deny rules block specific operations on files Claude can still see.

Layer 2: Command boundaries. Pre-approve safe commands so Claude doesn't interrupt you constantly, but require approval for anything that deploys, publishes, or sends data. Here's a starter set. Type `/permissions` and add these:

permissions

```
Allow: Bash(npm run *)
Allow: Bash(git status)
Allow: Bash(git diff)
Allow: Bash(git add *)
Allow: Bash(git commit *)
Ask: Bash(git push *)
Ask: Bash(npm install *)
Deny: Read(.env*)
Deny: Read(.ssh/*)
Deny: Read(.aws/*)
Deny: Bash(rm -rf *)
Ask: Bash(curl *)
```

THE HARD WAY VS THE EASY WAY

Hard way: Approve every single action Claude takes, 47 times per session. **Easy way:** Pre-approve safe commands (git, npm run) and deny dangerous ones (rm -rf). Set it once, work in peace.

Layer 3: Habit boundaries. Good habits are your last line of defense: start in Plan Mode for new projects, keep Chrome integration off by default, disable MCP servers you're not actively using, and don't install Skills or plugins from unknown sources without reviewing what tools they access.

The supply chain warning

Skills, plugins, and MCP servers from other people can contain code you didn't write. A Skill with `allowed-tools: Bash(*)` has full access to your terminal. Read the permissions before installing, just like you would for a phone app.

WARNING

Claude reads .env files by default. Users have reported API key exposure. Add deny rules before your first real project.

If something goes wrong

Don't panic. Most incidents are accidental exposure, not malicious. Stop the session immediately, rotate any exposed credentials, clear MCP authentication with `/mcp`, and review your project's change history.

COMMUNITY TIP

"I deny Claude Code access to **/.env* and it has never been an issue. Takes 30 seconds to set up." — Reddit user, r/ClaudeAI

MINI CHECKUP

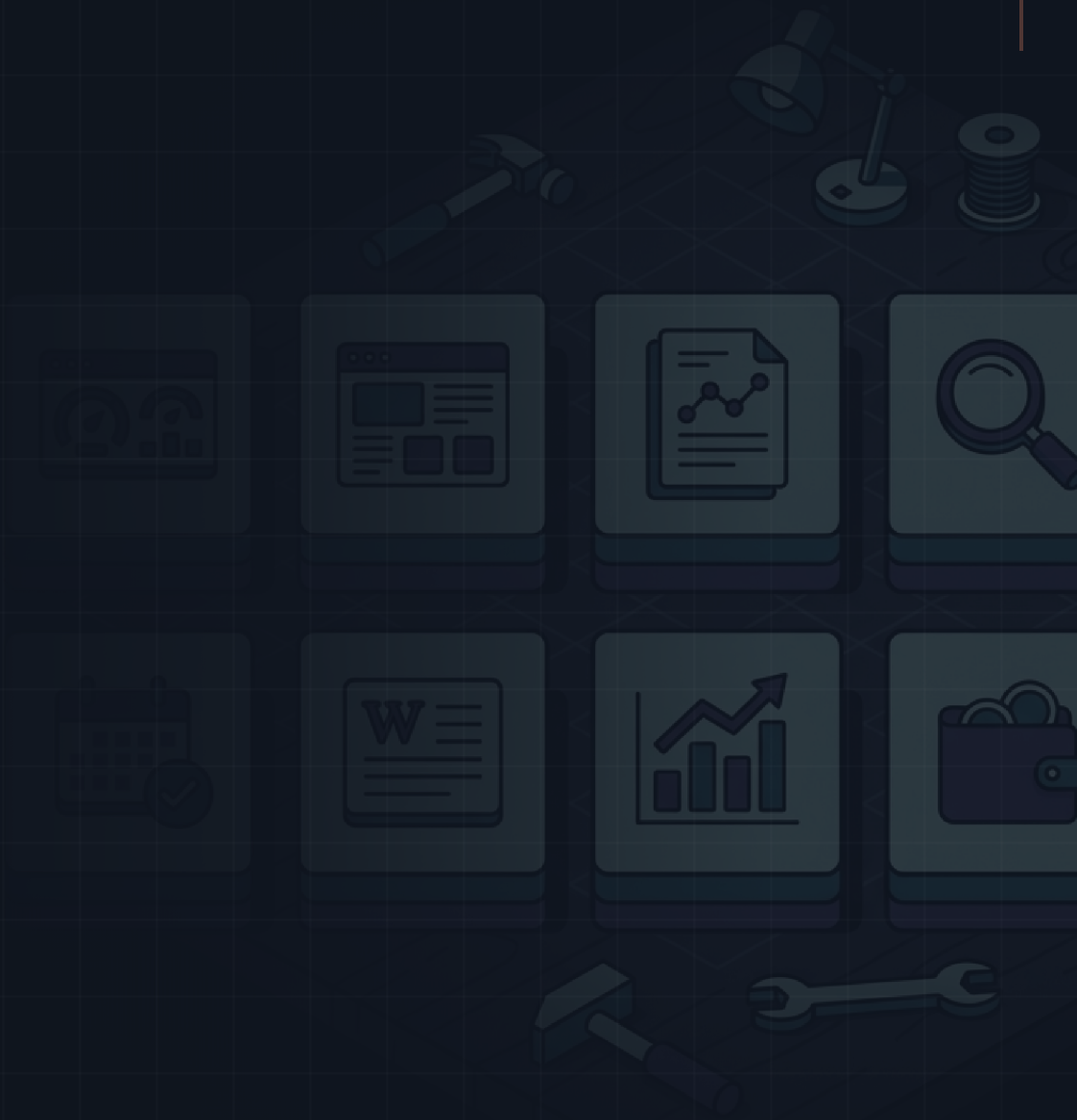
- ☐ I added deny rules for .env, .ssh, .aws, and key files
- ☐ I pre-approved safe commands so Claude doesn't interrupt constantly
- ☐ I keep Chrome integration off by default
- ☐ I review tool access before installing Skills or plugins from others
- ☐ I know to rotate credentials if something seems wrong

PROJECTS 01–10

The Playbooks

Real projects, step by step. Each one designed for non-developers.

- 01 — Spreadsheet to Dashboard
- 02 — Marketing Website
- 03 — Monday Morning Report
- 04 — Competitive Intel Tracker
- 05 — Customer FAQ Bot
- 06 — Event Landing Page
- 07 — Team Wiki / Knowledge Base
- 08 — Lead Scoring Prototype
- 09 — Finance / Budget Tracker
- 10 — Email Campaign Builder



PROJECTS 01–10 — THE PLAYBOOKS

Each playbook tells you: who this is for, what you'll end up with, the exact first prompt to send Claude, what to watch for, and how long it takes. Follow them in order or jump to whichever one matches your need.

WORTH KNOWING

Difficulty ratings: ● ○ ○ = Cowork only, no code. ● ● ○ = Claude Code Desktop, straightforward. ● ● ● = Claude Code Desktop, a few moving parts.

PROJECT 01 ● ● ○

Turn a Spreadsheet into a Real Dashboard

Your boss wants a dashboard. You have the data in Excel. Let's skip the six-week dev request.

DETAIL	INFO
Who	Marketing ops, sales ops, CS leads, anyone reporting metrics
What you get	A live dashboard with charts and filters that opens in any browser. Your team bookmarks it and checks it whenever they want.
Tool	CLAUDE CODE DESKTOP
Time	30–60 minutes including back-and-forth on styling

First prompt (copy this):

COPY THIS EXACTLY

"I have a spreadsheet with our monthly sales data. I want to turn it into a dashboard with: a bar chart showing revenue by month, a line chart showing deal count trend, a dropdown to filter by region, and a summary row at the top showing total revenue, average deal size, and number of deals. I have zero coding experience. Walk me through the plan before building anything."

What happens next

Claude will ask you to share the spreadsheet (drag it in or attach it). It will look at the columns and data types, then propose a plan. Review the plan. Say "Go." Claude builds it. You get a link to preview it on your computer.

WORTH KNOWING

Column names matter. If your spreadsheet has a column called "Rev" and you said "revenue" in your prompt, Claude might get confused. Use the exact column names from your file.

To put it online: Ask Claude: "How do I deploy this so my team can access it?" It will walk you through Vercel (free) or a similar service.

PROJECT 02 ●●○

Build Your Team's Marketing Website

A professional site you control, without waiting for the web team.

DETAIL	INFO
Who	Marketing managers, founders
What you get	Multi-page website with home, about, contact. Looks professional. Loads fast. Works on phones.
Tool	CLAUDE CODE DESKTOP
Time	1–2 hours

First prompt (copy this):

COPY THIS EXACTLY

"I need a professional marketing website for [your company/product]. It should have: a home page with hero section, value props, and CTA button; an about page; a contact page with a working form. Style: clean, modern, mobile-friendly. Our brand colors are [your hex codes]. Before building anything, explain your plan."

What happens next

Claude will propose a framework (usually Next.js or Astro), outline the pages, and describe the layout. Review the plan. Once you approve, Claude builds all the pages, styles them, and gives you a local preview link.

WORTH KNOWING

Claude will suggest Next.js or Astro. Both are fine. Say "pick whichever is simpler."

PROJECT 03 ●○○

Automate the Monday Morning Report

The report your boss reads every Monday, built automatically from your data.

DETAIL	INFO
Who	Anyone reporting weekly metrics
What you get	Formatted executive summary from raw data
Tool	COWORK
Time	15 minutes first run, 2 min after

First prompt (copy this):

COPY THIS EXACTLY

"Here's last week's raw data [attach file]. Turn this into a formatted executive summary with: a one-paragraph overview of key metrics, a table comparing this week vs last week, a bullet list of the top 3 highlights and top 3 concerns. Format it as a Word document I can send to my VP."

What happens next

Cowork reads your data, identifies the key metrics, calculates week-over-week changes, and produces a polished document. The second time, just drop in the new data and say "same format as last week."

DO THE MATH

This saves 30–45 minutes every week. Over a year, that's 26–39 hours.

PROJECT 04 ●○○

Competitive Intelligence Tracker

Know what your competitors are doing. Organized, searchable, shareable.

DETAIL	INFO
Who	Product marketing, strategy, sales enablement
What you get	Organized competitive tracker spreadsheet
Tool	COWORK
Time	45 min setup, 15 min monthly

First prompt (copy this):

COPY THIS EXACTLY

"I need a competitive intelligence tracker. Create a spreadsheet with columns for: competitor name, product/feature, pricing, positioning, last updated, and source URL. Start with these competitors: [list 3-5 names]. Then I'll drop in research materials for you to extract and organize."

What happens next

Cowork creates the spreadsheet structure, then waits for your research materials. Drop in pricing pages, feature pages, press releases, or screenshots. Cowork reads them all and fills the tracker.

WORTH KNOWING

Update monthly by dropping in new materials and saying "update the tracker."

PROJECT 05 ●●●

Customer FAQ Bot Trained on Your Docs

A chatbot that answers customer questions using your actual documentation, not made-up answers.

DETAIL	INFO
Who	Customer success, support, product teams
What you get	Chat widget answering from your docs
Tool	CLAUDE CODE DESKTOP
Time	3–5 hours

First prompt (copy this):

COPY THIS EXACTLY

"Build me a customer FAQ chatbot that answers questions using our actual documentation. I'll upload our docs. The bot should: search the docs for relevant answers, cite which document the answer came from, say 'I don't know' when the docs don't cover the question, and offer to connect the user with support. Include a clean chat widget interface."

What happens next

Claude will ask for your documentation files. Upload them all. Claude builds a search layer, a chat interface, and connects them. You test it by asking questions you know the answers to.

REALITY CHECK

This is the most complex project in the guide but the most valuable.

PROJECT 06 ●●○

Event Landing Page with Registration

A page for your next webinar, conference, or product launch. Live in an afternoon.

DETAIL	INFO
Who	Event marketers, demand gen, community managers
What you get	Landing page with registration form
Tool	CLAUDE CODE DESKTOP
Time	1–2 hours

First prompt (copy this):

COPY THIS EXACTLY

"I need a landing page for our upcoming [event type: webinar/conference/product launch]. Details: [event name], [date], [time], [speakers if any]. Include: hero with event name and date, agenda or speaker section, registration form that collects name and email, countdown timer. Style: professional but energetic."

What happens next

Claude builds the page with all sections, a working countdown timer, and a registration form. You preview it, tweak the copy, and deploy.

WORTH KNOWING

After the event, tell Claude: "Turn this into a post-event page with recordings and slides."

PROJECT 07 ●●○

Internal Team Wiki / Knowledge Base

Stop answering the same question in Slack for the fifteenth time.

DETAIL	INFO
Who	Team leads, ops managers, anyone onboarding
What you get	Internal wiki with search and categories
Tool	CLAUDE CODE DESKTOP
Time	2–3 hours

First prompt (copy this):

COPY THIS EXACTLY

"Create an internal team wiki / knowledge base. It should have: a home page with search, category navigation (Onboarding, Processes, Tools, FAQ), individual article pages with a clean reading layout, and a simple way to add new articles. Start by creating the structure and 2-3 sample articles. I'll fill in the real content."

What happens next

Claude builds the site structure with navigation, search, and sample articles. You replace the sample content with real information. Add articles by saying "add a new article about [topic]."

WORTH KNOWING

Claude writes fast once it knows the format.

PROJECT 08 ●●○

Lead Scoring Prototype

Score your leads based on your own criteria. No expensive tool required.

DETAIL	INFO
Who	Sales ops, revenue ops, marketing ops
What you get	Scored and ranked lead list from CSV
Tool	CLAUDE CODE DESKTOP
Time	1–2 hours

First prompt (copy this):

COPY THIS EXACTLY

"Build a lead scoring tool. I want to score leads based on these criteria: company size (1-10 points), industry match (1-10 points), engagement level based on email opens and page visits (1-10 points), budget indicated (1-10 points). The tool should take a CSV of leads and output a scored, ranked list. Use simple rules, not machine learning."

What happens next

Claude builds a scoring engine that reads your CSV, applies the rules you defined, and outputs a ranked list. You can tweak the weights by saying "make industry match worth more."

REALITY CHECK

Start with simple rules. Don't let Claude build a machine learning model. Rule-based scoring is transparent and debuggable.

PROJECT 09 ●●○

Personal Finance / Budget Tracker

Track your spending without giving your bank login to another app.

DETAIL	INFO
Who	Anyone tracking personal spending
What you get	Budget tracker with charts and categories
Tool	CLAUDE CODE DESKTOP
Time	1–2 hours

First prompt (copy this):

COPY THIS EXACTLY

"Build me a personal finance tracker. I'll upload my bank statement as CSV. I want: automatic categorization of transactions (groceries, dining, utilities, etc.), a monthly spending summary with charts, a budget comparison (actual vs target for each category), and a simple dashboard I can check weekly. Everything stays local on my computer."

What happens next

Claude reads your CSV, categorizes each transaction, builds charts, and creates a dashboard. You can adjust categories by saying "merge dining and takeout into one category."

DO NOT SKIP THIS

This runs entirely on your computer. Your financial data never leaves your machine.

PROJECT 10 ●○○

Email Sequence / Campaign Builder

Plan and draft an entire email sequence with your brand voice.

DETAIL	INFO
Who	Email marketers, demand gen, lifecycle marketing
What you get	Complete 5-email nurture sequence
Tool	COWORK
Time	20–30 minutes

First prompt (copy this):

COPY THIS EXACTLY

"I'm planning an email nurture sequence for [product/audience]. Create a 5-email sequence with: Email 1: welcome + value prop, Email 2: educational content / how-to, Email 3: case study / social proof, Email 4: objection handling, Email 5: CTA / offer. Write in a [tone: professional but friendly] voice. Target audience: [describe audience]."

What happens next

Cowork drafts all five emails with subject lines, body copy, and CTAs. You review, tweak the voice, and copy them into your email platform.

WORTH KNOWING

Attach a previous email that performed well and say "Match this tone and style."

PART SIX

When Things Go Wrong

It happens. Here's what to do.

- Claude is making things up (hallucination)
- How to check Claude's work
- It keeps making the same mistake
- The preview doesn't load
- You're lost and don't know what happened
- You want to start over
- Context rot: why Claude gets dumber

PART 6 — WHEN THINGS GO WRONG

Claude is making things up

Claude sometimes invents features that don't exist, references files that aren't there, or claims something works when it doesn't. This is called "hallucination." It's not lying. It's pattern-matching gone wrong.

JARGON BUSTER

Hallucination — When an AI confidently states something that isn't true. It's not deliberately lying; it's generating text that sounds right but isn't grounded in reality. Always verify claims that seem surprising.

Fix: If something Claude says doesn't seem right, say: "Are you sure? Show me the actual file/output/evidence." Claude will often correct itself.

It keeps making the same mistake

Claude tries a fix. It doesn't work. It tries again, slightly differently. Still broken. Third attempt. Same result.

COPY THIS EXACTLY

"Stop. The approach of [describe what it's doing] isn't working. Let's try a completely different approach."

Fix: Stop it (click Stop or press Escape). Give it new context or constraints. Don't say "try again."

How to check Claude's work (when you can't read code)

Claude produces impressive-looking output fast. But impressive-looking and correct are not the same thing. Developers can read the code to verify. You can't. So here are four techniques that work without any technical knowledge.

1. The "show your sources" prompt

After Claude finishes a task, don't immediately accept it. Ask it to explain its own work first:

COPY THIS EXACTLY

"Before I use this, show me: which files you read, what assumptions you made, and anything you're uncertain about."

This forces Claude to trace its own logic. If it says "I assumed the prices were in USD" and they weren't, you just caught a problem that would have cascaded through the entire output.

2. The visual check

If Claude built something visual (a dashboard, a landing page, a report), don't just read the code. Use Chrome integration (page 24) to make Claude check its own work visually:

COPY THIS EXACTLY

"Open this in the browser and check: does it match what I asked for? List anything that's missing or wrong."

3. The hostile reviewer trick

This is the most powerful technique. Ask Claude to critique its own work as if it were reviewing a colleague's output:

COPY THIS EXACTLY

"Now pretend you're reviewing a colleague's work. What would you criticize? What's fragile? What did you skip?"

Claude will often identify problems it didn't mention the first time. This works because generating output and evaluating output use different reasoning paths.

4. The number check

If the output contains data, calculations, or statistics, ask Claude to verify each one against the source: "Show me where each number in this report came from. Create a table with two columns: the claim and the source file or calculation."

WARNING

The most dangerous output is the one that LOOKS right. A dashboard with the wrong formula gives you confident-looking wrong numbers. Always verify the data, not just the design.

MINI CHECKUP

- ☐ I ask Claude to show its sources before accepting output
- ☐ I use the visual check for anything with a user interface
- ☐ I use the hostile reviewer trick for important deliverables
- ☐ I verify numbers against original data, not just the design

The preview doesn't load

Claude says the site is running but you see a blank page or error.

Fix:

- Check the address bar. It should say **localhost:something**. If it says something else, ask Claude for the correct URL.
- Ask Claude: "The preview isn't loading. Check for errors and fix them."
- If that doesn't work: "Stop the current server, clear any errors, and restart it from scratch."

You're lost and don't know what happened

The project has files everywhere. You're not sure what's working and what's broken.

COPY THIS EXACTLY

"Explain the current state of this project to me like I'm new here. What files exist, what works, what's broken, and what should we do next?"

Claude will audit the project and give you a status report.

You want to start over

Sometimes the cleanest fix is a fresh start. That's fine. It's not failure. It's software development.

1. Create a new empty folder.
2. Open a new session in Claude Code Desktop pointing to that folder.
3. Paste your original prompt (this is why you saved your good prompts).
4. This time, add any lessons: "Do it the same way but don't use [library that caused problems]. Use [alternative] instead."

I LEARNED THIS THE HARD WAY

I spent two hours trying to fix a project that was fundamentally broken. Starting over with the same prompt took 15 minutes and worked perfectly. Don't fall for the sunk cost fallacy.

REALITY CHECK

Professional developers start over all the time. It's called "spiking" — build a quick version to learn, throw it away, build the real version with what you learned. You're not failing. You're iterating.

Context Rot: Why Claude Gets Dumber the Longer You Talk

What's happening

Context rot is when Claude gets worse the longer your session goes. Not because it's tired. Because it ran out of room.

Think of Claude's memory like a whiteboard. Every message you send, every file Claude reads, every command output gets written on that whiteboard. Eventually, it fills up. Claude starts erasing the oldest stuff to make room for the newest stuff. Your instructions from the beginning? Gone.

This is the number one reason non-developers get frustrated and quit. They think Claude is broken. It isn't. It's just full.

How to spot it

Quick test: if you've been in the same session for more than 30 minutes and Claude's responses feel off, you probably have context rot. Here's how to confirm:

- Claude ignores rules you set at the beginning of the session
- It suggests changes you already rejected
- The quality of its work drops noticeably
- It "forgets" your brand colors, file structure, or preferences
- It repeats work it already did
- It gets confused about which files it already changed

GOLDEN RULE

Context is your most precious resource. Protect it. Start fresh for each new topic. Use /compact when things get slow.

The fix: three commands

`/context` — checks how full your whiteboard is. If that number is above 70%, it's time to act.

`/compact` — squeezes the whiteboard. Claude summarizes everything and throws away the details. You can even focus it: `/compact Focus on the dashboard layout changes`.

`/clear` — wipes the whiteboard completely. Use it between unrelated tasks.

WORTH KNOWING

Claude's memory (called the "context window") is about 200,000 tokens. Tokens are roughly words and pieces of words. 200,000 tokens is about 150,000 words, or a 500-page book. That sounds like a lot, but file reads, command outputs, and long conversations fill it fast. A single large file can use 10% of your available space.

The fresh session rule

If Claude is still acting confused after compacting, start a completely new session. Close the current one. Open a fresh one. This is not failure. This is the professional workflow. Your CLAUDE.md file makes sure the new session picks up your preferences automatically. Experienced users start new sessions every 20-30 focused exchanges.

THE HARD WAY VS THE EASY WAY

Hard way: Push through a degraded session, getting frustrated as Claude ignores your instructions. **Easy way:** Start a new session. CLAUDE.md brings back all preferences automatically. Time lost: 30 seconds.

The handoff trick

Before closing a long session, type this:

prompt

```
Write a summary of everything we've done, what's finished,
and what still needs to be done. Save it as SESSION-NOTES.md.
```

Claude creates a handoff document. Next session, just say:

prompt

```
Read SESSION-NOTES.md and continue where we left off.
```

THE 60-SECOND HANDOFF

This handoff trick takes 60 seconds. Without it, you'll spend 15-30 minutes re-explaining where you left off in a new session.

WORTH KNOWING: SUBAGENTS

When Claude needs to read a long document, don't let it fill your main whiteboard. Type: "Use a subagent to read brand-guide.pdf and give me a summary." The subagent runs in its own separate space and sends back a short summary. Your main session stays clean. You can also say: "Use a subagent to explore the project folder and tell me what's in it" — useful when you're returning to a project after time away.

WARNING

If Claude starts contradicting itself, do NOT keep correcting it. Every correction makes the context problem worse. Start fresh.

COMMUNITY TIP

"Context is like milk. It's best served fresh and condensed." — YK Sugi, "40+ Claude Code Tips"

MINI CHECKUP

- ☐ I know what context rot is and can spot the symptoms
- ☐ I know how to use /context, /compact, and /clear
- ☐ I write a SESSION-NOTES.md before ending long sessions
- ☐ I start new sessions for new topics instead of reusing old ones

APPENDIX

Quick Reference & Glossary

Everything you need to look up, in one place.

- Glossary of terms
- Copy/paste shortcuts
- Pricing (February 2026)
- Useful links
- When to stop and get a developer
- Connecting your tools (MCP)

APPENDIX — QUICK REFERENCE & GLOSSARY

Glossary of terms used in this guide

TERM	MEANING
API	Application Programming Interface. A way for two programs to talk to each other. You don't need to understand this to use Claude Code.
CLI	Command Line Interface. A text-based way to interact with your computer. Claude Code handles this for you.
Deploy	Putting your project on the internet so other people can see/use it.
Diff view	A display showing what changed in a file. Green = added, red = removed.
Git	A version control system that tracks changes to files. Claude Code uses it behind the scenes.
Hallucination	When AI confidently states something untrue. Not lying — pattern-matching gone wrong.
Localhost	Your own computer, acting as a temporary web server. When you see localhost:3000, the site is running on your machine only.
Sandbox	A protected area where Cowork does its work. Can't affect your real files.
CLAUDE.md	A text file in your project folder that stores your preferences. Claude reads it automatically at the start of every session.
Context rot	When Claude gets worse the longer your session goes because its memory (context window) fills up.
Context window	Claude's working memory. About 200,000 tokens. Files, messages, and commands all use space in it.
MCP	Model Context Protocol. Lets Claude Code talk directly to tools like Notion, Slack, and Figma.
Session	One conversation/workspace with Claude. Starts fresh each time.
Subagent	A helper that runs in its own memory space. Useful for reading large files without filling your main session.
Terminal	A text-based interface for giving commands to your computer. Claude Code interacts with it for you.

Pricing (February 2026)

All paid plans include access to Cowork and Claude Code Desktop.

PLAN	PRICE	BEST FOR
Pro	$20/month	Occasional use, trying things out
Max	$100/month	Daily use, real projects, fewer rate limits
Team	$30/user/month	Organizations, shared billing

Useful links

- **Claude Desktop download:** claude.ai/download
- **Claude documentation:** docs.anthropic.com
- **Vercel (free hosting):** vercel.com
- **Netlify (free hosting):** netlify.com

When to stop and get a developer

This guide helps you build a lot. But some things genuinely need a professional developer:

- **Anything that handles payment processing or credit card data.**
- **Systems that connect to your company's production database.**
- **Healthcare, legal, or financial tools with compliance requirements.**
- **Anything with complex user authentication and permissions** (admin/viewer roles across organizations).
- **Infrastructure that other production systems depend on.**

REALITY CHECK

The skill isn't building everything yourself. It's knowing when you can and when you should ask for help. That judgment is what makes you valuable — not the ability to do every single thing alone.

You've reached the end of the guide.
Now go build something.

Connecting Your Tools: MCP Integrations

What MCP is (one sentence)

MCP (Model Context Protocol) lets Claude Code talk directly to tools you already use: Notion, HubSpot, Google Sheets, Slack, Figma, Zapier. Instead of copying data between tools and Claude, you connect them. Claude reads your data directly and writes back to it. No copy-paste. No switching tabs.

Why it matters

Without MCP: You open your Notion doc, copy the brand guidelines, paste them into Claude, then tell Claude to build a landing page based on them.

With MCP: You say "Read our brand guidelines from Notion and build a landing page that follows them." Claude pulls the guidelines directly. No copy-pasting. That's the difference.

WORTH KNOWING

MCP connections persist across sessions. Set up once, available forever. Like installing an app on your phone.

The six integrations your audience cares about

TOOL	WHAT CLAUDE CAN DO
Notion	Read docs, update pages, manage tasks
HubSpot	Access CRM data, manage contacts
Figma	Read designs, turn mockups into code
Zapier	Connect 8,000+ apps, automate workflows
Slack	Send messages, search conversations
Airtable	Read/write records, manage bases

Install commands

Copy and paste the command for the tool you want to connect.

Notion:

terminal

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

HubSpot:

terminal

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

Figma:

terminal

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

Zapier:

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

Slack: Check Slack's MCP docs for the current install method.

Airtable:

```terminal
claude mcp add --transport stdio airtable \
  --env AIRTABLE_API_KEY=YOUR_KEY \
  -- npx -y airtable-mcp-server
```

The Airtable command is slightly more advanced. It requires an API key from your Airtable account settings (found under Account > API). Replace YOUR_KEY with your actual key.

What you can say after connecting

- "Read my Q3 planning doc from Notion and use it to build a dashboard."
- "Pull our top 50 leads from HubSpot and create a prioritization spreadsheet."
- "Build this landing page to match the Figma design."
- "When a form is submitted, add the lead to our Airtable and send a Slack notification."

How to install (Notion walkthrough)

Let's connect Notion as an example. The process is similar for all tools.

1. Open Claude Code and paste the Notion install command above. Hit Enter.
2. Claude opens a login page. Sign into your Notion account and approve the connection.
3. That's it. Now try: "Read my project brief from Notion." Claude pulls the document directly.

Three scopes (where connections live)

You don't need to memorize file paths. Just know that there are three levels:

- **Personal connections** — Yours alone. Good for your personal Notion, your HubSpot account.
- **Project connections** — Shared with anyone working on the project. Good for team tools.
- **Managed connections** — Set by your IT admin. Company-wide rules about which connections are allowed.

Managing connections

Three commands to remember:

- `claude mcp list` — Shows everything that's connected.
- `claude mcp remove notion` — Disconnects a tool.
- `/mcp` — Inside a session, shows connection status and manages authentication.

Context overhead warning

Each MCP connection adds tool definitions that take up space in Claude's memory (see Context Rot). Don't connect everything just because you can. Connect only what you need for your current project.

REAL EXAMPLE

"I connected Figma and said 'Build this page to match the mockup.' Claude read the design file, extracted the colors and spacing, and built a pixel-close version in 10 minutes. It even flagged where my CLAUDE.md brand colors didn't match the Figma file."

WARNING

MCP connects Claude to your real accounts with real data. Don't connect production databases or accounts with sensitive financial data until you're comfortable with how Claude uses them. Also keep secrets out of `.mcp.json` if your project folder is shared (on GitHub, for example) — use environment variables (written as `${VAR_NAME}`) so the file can be committed safely without exposing passwords or API keys.

MINI CHECKUP

- ☐ I understand what MCP connections do
- ☐ I connected at least one tool I already use
- ☐ I know how to list and remove connections
- ☐ I disabled MCP servers I'm not actively using

Ofir's Claude Code Field Manual

A practical guide for non-developers who want to build real things — dashboards, websites, automations — without writing code or waiting for IT.

Ofir's Claude Code Field Manual · February 2026

Compiled by Ofir Bloch