TECH NOT TECHNICAL · RESOURCE GUIDE

Claude Code Guide for Non-Technical Professionals

Everything you need to understand Claude Code — the tools, the workflow, and how it all connects.

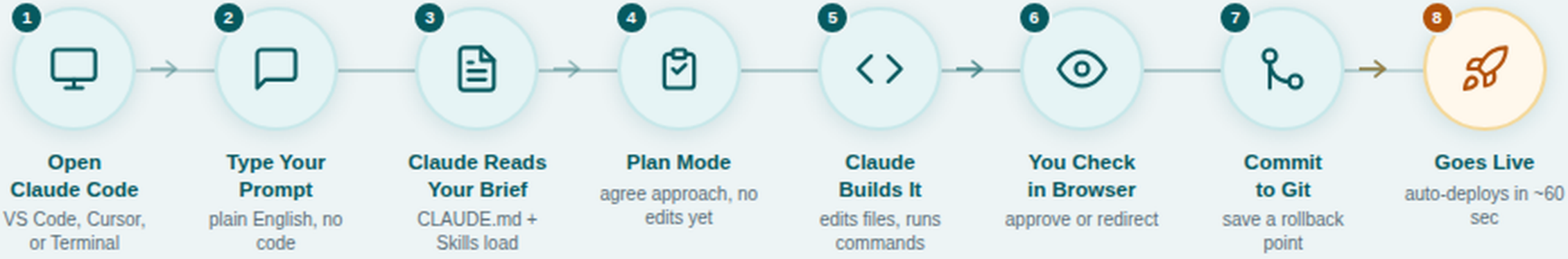
8-step overview | tools explained | step-by-step workflow | plain English

CLAUDE CODE · 30,000FT OVERVIEW

The Big Picture

Eight steps from idea to live feature — and the tools that power each one.

THE END-TO-END FLOW



COMPLI-MENTARY TOOLS

	VS Code or Cursor		VS Code watch edits	Browser review live	GitHub cloud backup	Vercel or Netlify

🔁 **Iteration loop** — if the result isn't right, redirect Claude and it rebuilds. Repeat until happy.

WHAT MAKES THIS WORK

You Stay in Charge

You define the goal in plain English. Claude handles the technical execution. Your job is to direct and review — not to code.

Claude Knows Your Project

CLAUDE.md and Skills files give Claude memory of your app, rules, and preferences — loaded fresh every single session.

Nothing Is Ever Lost

Git saves a snapshot after every working feature. If anything breaks, one command rolls you back to the last safe version.

CLAUDE CODE · COMPLETE REFERENCE GUIDE

Every Tool & Why It Exists

A plain-English guide to every component — what it does, why you need it, and what breaks without it.

CATEGORY 1 — WHERE YOU WORK: EDITORS & INTERFACES

VS Code
FREE VISUAL EDITOR — OPTIONAL

Microsoft's free code editor. Gives you a visual file tree, colour-coded panels, and a built-in terminal — all in one window. Claude Code runs inside the terminal panel at the bottom.

- ✓ See all project files in a visual sidebar
- ✓ Watch Claude edit files in real time
- ✓ Colour-coded code is easier to scan
- ✓ Terminal panel built right in — no separate window

Without it: You can't visually browse files or watch edits live. You'd open files manually to inspect them.

Cursor
AI-NATIVE EDITOR — OPTIONAL

VS Code rebuilt from the ground up for AI. Has its own AI chat panel baked in alongside full Claude Code integration — two AI modes in one environment.

- ✓ Everything VS Code does, plus AI chat built in
- ✓ Inline AI suggestions as you type
- ✓ Great for founders who want AI everywhere in the editor
- ✓ Free tier available; paid plans for heavy use

Without it: Same trade-off as VS Code. Use one or the other — or neither if you're comfortable in terminal only.

Terminal
REQUIRED

`Mac Terminal · iTerm2 · Windows Terminal`

The command-line window where Claude Code lives. You type `claude` here to start a session and have your full conversation. This is the only non-negotiable interface — VS Code and Cursor are just wrappers around it.

Without it: You cannot run Claude Code at all. Everything else builds on top of this.

Terminal Only — No Editor Needed
MINIMUM SETUP

`Simplest viable workflow`

You don't need VS Code or Cursor at all. Open Terminal → navigate to your project folder → type `claude`. Describe what you want in plain English. Check the results in your **browser**. That's the complete loop.

Best for: Non-technical founders getting started fast. Your browser is your feedback screen — not the code editor.

CATEGORY 2 — CORE DOCUMENTS: FILES CLAUDE READS EVERY SESSION

Project Brief
REQUIRED

`CLAUDE.md`

The single most important file in your project. Sits in the root folder and is automatically read by Claude at the start of every session. Think of it as the standing brief you'd hand a new developer on day one — your app's purpose, tech stack, naming rules, what to never touch, and how your team works. Without it, Claude starts every session with zero context and makes generic assumptions.

Without it: Claude has no memory of your project between sessions. It will make generic decisions instead of tailored ones. Always keep this updated as your app evolves.

WHAT TO INCLUDE IN CLAUDE.MD

```
App description → "PackSmart is a travel packing list app"
Tech stack → "React, Tailwind CSS, Supabase"
Colour palette → "Primary: #065A60, Accent: #F59E0B"
Naming conventions → "Components use PascalCase"
Folder structure → "Components live in /src/components"
Off-limits folders → "Never edit /legacy or /migrations"
Design principles → "Keep UI minimal, gender-neutral palette"
Current focus → "Working on packing list page now"
```

Skills
RECOMMENDED

`/skills/SKILL.md`

Reusable instruction sets for recurring tasks. A skill is a markdown file that teaches Claude exactly how to do something your way — build a component, write a query, structure a new page. Write it once; Claude follows it every time without you repeating yourself.

Without it: Claude improvises and may be inconsistent. Skills make output predictable and on-brand across every session.

Environment Variables
RECOMMENDED

`.env`

Stores private credentials — API keys, database passwords, third-party tokens. Claude can use these values without them ever appearing in your code or being accidentally uploaded to GitHub. This file must never be shared publicly.

Without it: Secrets end up hardcoded in your code — a serious security risk if the project is ever shared or made public.

Ignore Rules
OPTIONAL

`.claudeignore`

A list of folders and files Claude should never read or edit — like `node_modules/`, build outputs, or protected config. Keeps Claude focused and fast.

Without it: Claude may read irrelevant files, slow down, or accidentally edit things you didn't intend to change.

KEY — REQUIRED: Can't use Claude Code without this · RECOMMENDED: Strongly advised for safe, consistent builds · OPTIONAL: Enhances experience, not essential

TECH NOT TECHNICAL

Every Tool & Why It Exists

Continued — Power tools, version control, and how Claude manages your session.

CATEGORY 3 — POWER TOOLS: EXTEND WHAT CLAUDE CAN DO

MCP Servers

OPTIONAL

Model Context Protocol

MCP is Claude's plugin system for connecting to external services. Each MCP server gives Claude the ability to talk to a real-world tool — your database, calendar, GitHub, Slack, Stripe, Notion, and more. Instead of only editing code files, Claude can now take real actions: create a database record, fetch live data, push a commit, or send a Slack message. You configure which servers Claude can access.

Without it: Claude is limited to reading and writing local files. With MCP, Claude becomes a true agent that can interact with your entire business stack — not just your code.

VS Code Extensions

OPTIONAL

Extensions Marketplace

Add-ons that extend VS Code's capabilities — real-time error highlighting, auto-formatters, Git visualisers, Tailwind CSS autocomplete, colour themes, and framework-specific tools. Claude Code works perfectly without any extensions, but they make reviewing Claude's output faster and more informative for you.

Without it: VS Code works fine. Extensions are quality-of-life upgrades. Recommended ones to start: Tailwind CSS IntelliSense, GitLens, Prettier, ESLint.

CATEGORY 4 — SAFETY NET: VERSION CONTROL

Git

STRONGLY RECOMMENDED

Version control system · runs locally

Git is an infinite undo system for your entire project. Every time Claude builds something that works, you "commit" it — creating a permanent snapshot. If a future session breaks something, one command restores any previous version instantly. Claude Code can also run Git commands itself, committing completed work on your behalf.

Without it: There is no rollback if Claude makes a mistake. You could permanently lose working code. This is the single most important safety habit for non-technical founders building with AI.

GitHub

RECOMMENDED

Remote repository host · cloud backup

GitHub stores your Git history in the cloud — backed up offsite, accessible from any computer, shareable with collaborators or hired developers. You can also connect it directly to deployment platforms like Vercel or Netlify so your app publishes automatically when you push new code.

Without it: Your entire project history lives only on your laptop. If your machine is lost or breaks, everything is gone. GitHub is your offsite insurance policy.

CATEGORY 5 — INSIDE THE SESSION: HOW CLAUDE MANAGES CONTEXT

Context Window

ALWAYS ACTIVE

The active conversation

Everything Claude holds in working memory during a session — your messages, its replies, and files it has read. Context grows as the session progresses. When you start a fresh session, the context resets entirely. CLAUDE.md is what persists across sessions.

Tip: Type `/clear` to reset context if Claude starts getting confused or going in circles mid-session.

Auto-Correction Loop

BUILT-IN

Autonomous error recovery

When Claude runs code and hits an error, it reads the error message, diagnoses the problem, adjusts its approach, and tries again — automatically. This loop repeats until the task succeeds or Claude decides it needs your input. You don't have to intervene for most errors.

Tip: If Claude loops more than 3–4 times on the same error, redirect it with a fresh description of the goal or break the task into smaller pieces.

Session Memory

OPTIONAL

/memory · CLAUDE.md updates

You can instruct Claude to remember decisions by writing them to CLAUDE.md or a dedicated memory file. For example: *"Remember we chose Supabase over Firebase."* Claude updates the file and carries that decision into every future session automatically.

Without it: Claude forgets everything the moment your terminal session ends. Persistent memory lives in your files — not in Claude's mind.

CATEGORY 6 — GOING LIVE: DEPLOYMENT TOOLS

Vercel

OPTIONAL

vercel.com · auto-deploy

Connects to your GitHub repo and automatically publishes your app live every time you push new code. No server management needed. Your app gets a real URL the world can access within seconds of a commit.

Best for: React / Next.js apps. Free tier is generous. Claude Code can generate your Vercel config file automatically.

Netlify

OPTIONAL

netlify.com · auto-deploy

Similar to Vercel — connects to GitHub and deploys your app automatically on every push. Strong support for static sites and serverless functions. Good alternative if Vercel doesn't fit your stack.

Best for: Static sites, HTML/CSS projects, and apps with serverless backend functions. Also has a generous free tier.

Supabase

OPTIONAL

supabase.com · database + auth

A ready-made backend — gives your app a database, user authentication, file storage, and an API, all in one place. Claude Code can write Supabase queries and schema files directly. No backend developer needed for most use cases.

Best for: Apps that need to store user data, handle logins, or sync data across devices. The go-to backend choice for non-technical founders.

KEY — REQUIRED: Can't use Claude Code without this · RECOMMENDED: Strongly advised for safe, consistent builds · OPTIONAL: Enhances experience, not essential · ALWAYS ACTIVE: On by default, no setup needed · TECH NOT TECHNICAL

CLAUDE CODE · WORKFLOW DIAGRAM

Claude Code Workflow Explained

Every step from opening Claude Code to seeing your feature live in a browser — explained in plain English.

KEY · You set up · You plan · You prompt · Claude reads · Claude thinks · Claude builds · You review · You save · Goes live

STEP 01 · GETTING STARTED

Open Claude Code — in your editor or directly in Terminal

You have two options for how to open Claude Code. **Option A (recommended for beginners):** Open VS Code or Cursor, then open the built-in Terminal panel at the bottom of the screen (View → Terminal). Type `claude` and press Enter — Claude Code chat opens right there. **Option B (minimal setup):** Open your Mac Terminal app directly, navigate to your project folder, and type `claude`. Either way, once you see the Claude prompt, you're ready.

⬛ Option A — VS Code or Cursor

Best if you want to see your files and watch Claude edit them live. Open the editor first, then start Claude in its built-in terminal panel.

$_ Option B — Terminal Only

No editor needed. Just your Mac Terminal app. You'll test results in your browser. Simpler, and totally valid for non-technical founders.

STEP 02 · YOUR INPUT

You describe what you want — in plain English, no code needed

Now you give Claude its task. No technical language required — describe the outcome you want to see, not the code you want written. The more specific you are about the result, the better. Think of it like briefing a contractor: tell them what the finished room should look like, not which tools to use.

💡 **Good prompt:** "Add a filter bar below the header on the packing list page. It should have three buttons: All, Carry-on, and Suitcase. Tapping a button filters the list to show only items with that bag tag."

❌ **Vague prompt:** "Make the packing list better." — Claude has nothing concrete to act on.

`one task at a time` `describe the outcome` `not the code` `be specific`

STEP 03 · AUTO-LOAD · HAPPENS IN THE BACKGROUND

Claude silently reads your CLAUDE.md and Skills files

The moment you start a session — whether in Plan Mode or Build Mode — Claude automatically reads your `CLAUDE.md` file and any **Skills** you've set up in your `/skills/` folder. You don't trigger this; it happens every single time without you asking. This is how Claude knows your app's purpose, your tech stack, your colour palette, your naming rules, and your preferred way of doing things.

📋 CLAUDE.md — Your Standing Brief
One file that tells Claude everything about your project. Updated by you whenever something important changes. Think of it as onboarding docs for Claude at the start of every session.

⚡ Skills — Your How-To Guides
Markdown files in `/skills/` that teach Claude how to do recurring tasks your way. E.g. a "create-component" skill so every new UI component is built the same way every time.

Skills are also called during a task: When Claude is mid-build and realises a relevant skill exists (e.g. you asked it to create a new page and there's a "create-page" skill), it reads and follows that skill automatically — even mid-session.

STEP 04 · BEFORE YOU BUILD ANYTHING — USE PLAN MODE

Switch Claude to Plan Mode and think through the feature first

Before asking Claude to write any code, switch to **Plan Mode** by typing `/plan` in the chat. In Plan Mode, Claude cannot edit any files — it can only think, ask questions, and map out an approach. This is where you describe what you want to build and Claude tells you exactly what it would change and why. You read the plan, ask questions, refine it, and only approve it when it makes sense to you.

Why this matters: Jumping straight to building without planning is the #1 mistake non-technical founders make with AI. A bad plan executed perfectly still produces the wrong result. Plan Mode costs you 5 minutes and saves hours of unpicking mistakes. Always plan first on anything more complex than a small text change.

How to switch modes: Type `/plan` to enter Plan Mode (read-only, no edits). Type `/code` to switch back to Build Mode when you're happy with the plan and ready to go.

`/plan — enter plan mode` `/code — enter build mode` `no files touched in plan mode`

STEP 05 · CLAUDE READS · AUTOMATIC

Claude maps your codebase before touching anything

Before writing a single line, Claude reads the specific files relevant to your task — the page file, your component folder, your style definitions. It's building a precise map of what already exists so it edits in the right place and doesn't accidentally duplicate or overwrite something. You'll see it print out file paths in the chat as it explores. This is normal — it means Claude is being careful, not slow.

`reads relevant files` `checks folder structure` `reviews existing patterns` `re-reads skills if relevant`

STEP 06 · CLAUDE THINKS · VISIBLE IN CHAT

Claude tells you its plan — you can redirect before it acts

Claude will describe its intended approach in the chat before it starts editing. It explains which files it will change, what it will add or remove, and flags anything it's uncertain about. **This is your last chance to redirect before any changes are made.** Read it. If the approach sounds wrong, say so. This is far faster than undoing bad edits after the fact.

What to look for: Does it mention the right files? Does the approach match what you had in mind? Is it planning to change anything you didn't ask it to? If anything looks off, type your correction now — before it starts building.

`explains approach` `lists files to edit` `asks if anything is unclear` `redirect here if needed`

STEP 07 · CLAUDE BUILDS · WATCH IT HAPPEN

Claude writes and edits your files — in real time

Claude edits your actual project files. If you have VS Code or Cursor open, you'll see the changes appear live in your file tabs as Claude works. It can also run terminal commands on your behalf — installing a package, starting your development server, running tests. Every action is printed in the chat so you always know what it's doing.

Skills in action: If your task matches a skill (e.g. "create a new page"), Claude reads the relevant skill file mid-task and follows your defined pattern automatically — consistent output every time, no extra prompting needed.

`edits files live` `installs packages` `starts dev server` `follows skills automatically` `runs tests`

🔁 **Auto-correction loop:** If Claude hits an error (a package fails to install, code produces a crash), it reads the error, diagnoses the cause, adjusts its approach, and retries — automatically. You usually don't need to intervene. If it loops more than 3–4 times on the same problem, redirect it: describe the end goal again in fresh words, or break the task into smaller steps.

STEP 08 · YOU REVIEW

Check the result in your browser — approve or redirect

Claude finishes and summarises what it changed. Open (or refresh) your browser where your app is running — you'll see the result immediately. Does it look right? Work as expected? If yes, move to the next step. If something's off, type your feedback in the same chat window — no need to start over. Claude reads your correction and adjusts.

Your browser is your feedback screen — not the code. You don't need to read or understand what Claude wrote. Open the app, try the feature, and judge it on how it looks and behaves. That's it.

`test in browser` `approve or redirect` `no need to read the code`

STEP 09 · SAVE YOUR WORK

Commit to Git — lock in the working version forever

Once you're happy with the result, commit your changes to Git. This takes a permanent snapshot of the working code, labels it with a message, and saves it in your version history. If any future session ever breaks this feature, one command restores you to exactly this moment. You can ask Claude to do this for you: *"Commit this with the message: added filter bar to packing list."*

Good habit: Commit after every feature that works. Small, frequent commits give you a safety net at every step — not just at the end of a big session.

`git commit` `git push to GitHub` `ask Claude to do it for you` `permanent rollback point`

STEP 10 · GO LIVE · DEPLOY TO THE WORLD

Push to GitHub → your app updates automatically

Once your code is committed and pushed to GitHub, your deployment platform (Vercel or Netlify) detects the new code automatically and publishes it live — no manual upload needed. Within 1–2 minutes, anyone with your app's URL sees the updated version. Claude can also generate your deployment config files (`vercel.json` or `netlify.toml`) and write step-by-step setup instructions for you.

▲ Vercel
Best for React / Next.js apps. Connect once to your GitHub repo. Every push to the main branch auto-deploys in ~60 seconds. Free tier available.

🌿 Netlify
Great for static sites and serverless apps. Same auto-deploy from GitHub. Has a built-in form handler and edge functions. Also free to start.

Ask Claude to set this up: "Set up Vercel deployment for this project and create all the config files I need." Claude will generate everything and walk you through connecting your GitHub repo to Vercel in your browser.

`push to GitHub` `Vercel auto-deploys` `live in ~60 seconds` `Claude sets up config` `free tier available`

⚡ **Speed**
Features that once took developer days now take minutes of plain-English conversation.

🔍 **Transparency**
Claude tells you every file it touches and every decision it makes — you stay in control.

🛡️ **Safety**
Git gives you infinite undo. Plan Mode prevents bad builds before they start.

Still unsure?
Just ask Claude.

You don't need to be technical to get value from Claude Code. Start where you are, describe what you want to build, and trust the process.

Save this for later **Share with your team**

"Clarity rarely comes the first time. Give it time — it comes together. And if you're still stuck, **just ask Claude.**"