Crash course: build with an AI agent

Everything you need before you start. Then we build.

AI builders society
GREENWICH

Five parts, one build

10 min	**1 · Understanding the machine**	How the agent works, so you know why it behaves as it does.
15–20 min	**2 · Vibecoding**	How to steer an agent: spec, plan, build, review.
20 min	**Build session**	Everyone builds their own one-page site.
15–20 min	**3 · Levelling up**	Git, secrets, rules files and other extras.
10 min	**Wrap up**	Cheat sheet, show and tell, what's next.

Understanding the machine

What’s going on under the hood, so you know why it behaves the way it does.

1 · Understanding the machine
10 min

2 · Vibecoding
15–20 min

Build session
20 min

3 · Levelling up
15–20 min

Wrap up
10 min

You already use AI every day

AI is software that learns patterns from examples, not rules written by hand.

Autocomplete

Your keyboard guesses your next word.

Maps and traffic

Arrival times come from patterns in millions of journeys.

Recommendations

Your feed, Netflix and Spotify pick what you'll like next.

Spam filters

Your inbox has learned what junk looks like.

Face unlock

Your phone knows your face, and sorts photos by person.

Subtitles and translation

Speech becomes text, and one language becomes another.

ChatGPT and Claude are the same idea, **scaled up: predicting what comes next.**

A chatbot talks. An agent does.

Look

Reads your files and the error on screen.

→

Act

Edits files, runs commands, searches the web.

→

Check

Runs it again to see if it worked, then loops.

Round again until it thinks it’s done

You set the goal. **Your job is to check that it really is done.**

Same brain, different toolbox

The model is the brain

Claude, GPT, Gemini and others. It reasons and writes, but on its own it can only talk.

The harness is the toolbox

OpenCode or Claude Code. It hands the brain tools, rules and memory.

What the tools let it do:

- Read files
- Edit files
- Run commands
- Search your code
- Browse the web
- Ask you questions

In plain English: The model is the AI itself. The harness is the app that lets it touch your computer. *Like a driver (the model) and the car they're given (the harness).*

Model, harness, agent

Model

The brain. It reasons and writes, but on its own it can only talk.

Claude, GPT, Gemini

+

Harness

The toolbox and rules around the brain: files, commands, memory.

OpenCode, Claude Code, Codex

=

Agent

The two working together, looping until the job is done.

What you actually talk to

Same model in a different harness, **different results.**

The agent works on a desk, and the desk has limited space

Everything it can see has to fit on the desk

- Your chat
- Files it opened
- Command output
- Your rules file

When the desk fills up, older things get squashed into a summary and details fall off. **That's why long chats get worse.**

Vibecoding

How to get an agent to build what you actually want.

1 · Understanding the machine
10 min

2 · Vibecoding
15–20 min

Build session
20 min

3 · Levelling up
15–20 min

Wrap up
10 min

Tell it you're not technical

Start every session with

I'm not technical. Explain everything in plain English, and if you use a technical word, tell me what it means.

Still lost? Ask

What does that mean, in plain English?

Keep the desk clear

One task, one session	Start fresh for each new task: `/new` in OpenCode, `/clear` in Claude Code.
Squash when it’s long	`/compact` summarises the chat so far and frees up space.
Two strikes, start over	Corrected it twice and it’s still wrong? Start fresh with a clearer prompt.
Lost? Ask it	Any word you don’t know: “explain that in plain English.”

Spec. Plan. Build. Review. Repeat.

Every feature goes round this loop until you're happy with it.



Let it interview you first

Most bad results start with a vague request. Start with this instead:

I want to build **[your idea]**. Before writing any code, interview me one question at a time until you understand exactly what I want. Then write it up as SPEC.md.

It asks you

One question at a time, until it knows exactly what you want.

→

It writes SPEC.md

A short written brief of what you're building.

→

You correct it

It becomes the brief every later step works from.

Let it plan, then check the plan

Read SPEC.md and make a plan to build it. Stop at each checkpoint, report back in plain English, and wait for me before carrying on.

You approve it

Read the plan. Change anything that isn’t what you want before it starts.

Skip it for tiny things

If you can describe the change in one sentence, just ask for it.

Checkpoints: stop, report, check

Give it one step from the plan. It stops at each checkpoint, and you check everything still holds up before it carries on.

At a checkpoint

```
What did you just do, and what works now?
Show me.
```

If something's off

```
That's not right: [what you see]. Fix it
before moving on.
```

Regular moments where you look, not tiny steps.

“Done” is a claim. Ask for proof.

Show me	• The test results • The command it ran and what came back • A screenshot
Explain it	`Explain what you changed and why, in plain English, for someone who doesn’t code.`
Look yourself	Open it and click around. If it looks wrong to you, it is wrong.

The explanation is what it *thinks* it did. **The proof shows it worked.**

Iterate until you’re happy

Say what you see

“The button is hidden on my phone” works better than guessing at the fix.

Show, don’t describe

Drop in a screenshot of the problem. It can read images.

One change per message

Five requests at once makes it hard to check any of them.

Update the spec

Changed your mind? Change SPEC.md too, so the next session knows.

Things that don’t work, and what does

“Make no mistakes.”	**Instead:** tell it how to check. “If you’re not sure, say so. Run it and show me.”
“You are a world-class engineer.”	**Instead:** give it facts. What it’s for, who uses it, what must not break.
“CRITICAL!!! YOU MUST…”	**Instead:** write it calmly and specifically. Shouting, threats and tips don’t help.
“Fix it.”	**Instead:** say what’s wrong, where, and what “done” looks like.
“No, fix it again…” ×10	**Instead:** after two tries, start fresh with a better prompt.
“Don’t make it long.”	**Instead:** say what you want. “Short, plain-English paragraphs.”

Now you build

Open OpenCode, or any AI coding tool you prefer, in an empty folder. Your first prompt is on the next slide.

1 · Understanding the machine
10 min

2 · Vibecoding
15–20 min

Build session
20 min

3 · Levelling up
15–20 min

Wrap up
10 min

Your first build

Open OpenCode, or any AI coding tool you prefer, in an empty folder and paste this in. It doesn’t know you yet, so it has to ask.

> I’m not technical, so explain anything technical in plain English. I want a one-page website about me. You don’t know anything about me yet, so interview me first, one question at a time. Then show me a plan before you build.



A little trick

Site looks a bit generic? Pick a style you like at **styles.refero.design** and hand the link to your agent.

> Restyle my site to match this design: **[paste the link]**. Keep all my content. Show me before and after.

Don’t like it? **Try another style.** It takes one message.

Levelling up

The extras that make bigger projects easier and safer.

1 · Understanding the machine
10 min

2 · Vibecoding
15–20 min

Build session
20 min

3 · Levelling up
15–20 min

Wrap up
10 min

AGENTS.md: the induction pack

Notes the agent reads at the start of every session. Type **/init** and it writes the project one for you.

Global

About you. One file for every project.

- How you like it to talk to you
- How it should prove work is done
- What it must ask before doing, like deleting

Project

About this project. One per project.

- How to run and test it
- Rules it can't guess from the code
- Things it must never do here

Leave out anything it can see in the code, long tutorials and vague advice like "write clean code". **Keep it short:** long files get partly ignored.

In plain English: AGENTS.md is a notes file the agent reads every time it starts. One is about you, one is about the project. *Like the induction pack a new starter reads on day one, plus the project brief.*

Skills: a recipe binder

A saved how-to for one job, like “how we write a report” or “our launch checklist”.

1

It only sees the recipe titles

2

Opens one when the job needs it

3

Follows it step by step

A skill is a folder with a **SKILL.md** file. The same skill works in OpenCode and Claude Code.

In plain English: A skill is a saved set of instructions for one job, opened only when needed. *Like a recipe card you only pull out when you’re cooking that dish.*

Sub-agents: helpers with their own desk

Hand a big job to a helper. It works in its own session and brings back a short summary, so **your main chat stays clean**.

`@general`

Any multi-step job

`@explore`

Reads your code, changes nothing

`@scout`

Researches docs online, changes nothing

OpenCode sends jobs to them by itself. Or call one by typing its name, or just say **“use a subagent to…”**

In plain English: A sub-agent is a second AI you hand a side job to. It reports back in a few lines. *Like sending an intern to read a report and bring you the summary.*

Plugs and guard rails

MCP: plugs for other apps

Connects the agent to things like Notion, GitHub or a web browser.

Each plug takes up desk space, so connect only what you need.

Hooks: automatic checks

Run something every time, like tidying the code after each edit.

A rules file is a request. A hook always happens.

In OpenCode, hooks are plug-ins written in code. In Claude Code they’re short commands in a settings file.

In plain English: MCP connects the agent to other apps. A hook is a rule that runs automatically. *Like a plug adapter, and a smoke alarm that goes off without being asked.*

It works on any file, not just code

Emails

Draft replies in your tone from a few bullet points.

Slides

Turn rough notes into a deck you can present.

Spreadsheets

Clean up messy data, add formulas, make charts.

Documents

Reports, CVs and cover letters, properly formatted.

Research

Read a pile of PDFs and pull out what matters.

Chores

Rename, sort and tidy hundreds of files at once.

If it lives in a file, an agent can help. **Plugs (MCP) connect it to apps like your email and calendar.**

Git: save points for your project

Every commit is a snapshot you can jump back to. You don’t need to learn the commands. Just ask:

Save

`Commit this with a clear message.`

Check

`What changed since the last commit?`

Rescue

`Something broke. Go back to the last working commit.`

Push to GitHub to keep a backup online, and to show employers what you’ve built.

In plain English: Git keeps save points of your project. A commit is one save point. GitHub keeps a copy online. *Like checkpoints in a video game, backed up to the cloud.*

Keep secrets secret

A secret is anything that unlocks an account or can cost you money: passwords, API keys, tokens.

Put them in .env

A file just for secrets. Your code reads from it, so they never sit in the code.

Keep .env out of git

List it in .gitignore so it never reaches GitHub. Ask the agent to set it up.

Leaked one? Cancel it

Deleting it isn't enough. Revoke the key on the provider's site and make a new one.

Never paste a secret into the chat. OpenCode already refuses to read .env files unless you allow it.

In plain English: .env is a private file for passwords and keys. .gitignore is a list of files git must never save. *Like keeping your house key in your pocket, not taped to the door.*

Leave a note for next time

Every new session starts with a blank memory. A HANDOFF.md file carries the work over, like a shift-change note.

Before you stop

Write HANDOFF.md: what's done, what's broken, and the exact next step.

Next session

Read HANDOFF.md first, then carry on.

Three headings are enough: **done, broken, next.** Short and specific beats long.

In plain English: HANDOFF.md is a short note so the next session knows where you left off. *Like a nurse's shift-change note.*

Stay safe

Share links are public

/share makes a link anyone can open. Don’t share chats with private info.

It can delete things

Commit before big changes, so nothing is ever lost for good.

Free models have a price

Some free trial models use your prompts for training. Keep private data out.

Work in a project folder

Open your agent inside one project’s folder, not your whole computer. That keeps what it can touch small.

Scroll past the hype merchants

“This one prompt changes everything”	Hooks are made to get views, not to teach. If it sounds like magic, it isn’t.
“Secret commands”	If it’s not in the tool’s own docs, it doesn’t exist. Type / to see the real list.
“My 50-skill mega setup”	Piles of skills and plugs fill the desk. Add one only when you keep repeating a job.
“Comment AI and I’ll DM you the link”	That’s to game the algorithm or sell you something. Good resources are linked openly.
“£99 prompt pack”	Everything in this deck is free. The official docs beat any paid bundle.

What actually works

The official docs, a simple setup you understand, and building things. Lots of things.

Cheat sheet

To do this	OpenCode	Claude Code
Start a fresh session	`/new`	`/clear`
Squash a long chat	`/compact`	`/compact`
Write the rules file	`/init` (AGENTS.md)	`/init` (CLAUDE.md)
Undo a step	`/undo`	`/rewind` or `Esc Esc`
Point at a file	`@filename`	`@filename`
Call a helper	`@general` or “use a subagent to…”	“use a subagent to…”
Save a checkpoint	“commit this”	“commit this”

Vibecoding is the tip of the iceberg

Logins and payments	Who can do what, and getting paid safely.
Privacy and security	Keeping data and secrets safe, and following the law (GDPR).
Staying up	Monitoring, backups, and coping with lots of users.
Testing and shipping	Checking every change before it goes live.

Fine for learning and showing off. **Learn what's below the waterline** before real users or real money.

Before you go

Show one thing

Share what you built with the person next to you.

Keep building

Pick one small change and do it at home this week.

Free docs: opencode.ai/docs and code.claude.com/docs
Tips and tutorials: claude.dev
Free courses from Anthropic: anthropic.skilljar.com

AI builders society
GREENWICH

Thank you

Keep building. See you next session.