

`./CLAUDE.md`

Your new colleague reads one file before every shift.

Claude Code starts each session with a blank memory. CLAUDE.md is the briefing it reads first: what the project is, how you work, and where the lines are.

This handout gives you the essentials, a template you can copy, and ready-made prompts so Claude Code can help you write your own.

What it is

CLAUDE.md is a plain Markdown file that Claude Code loads into its context automatically at the start of every session. Anything you write there acts as standing instructions: your conventions, the commands to run, the things it should never do. You no longer have to repeat yourself each time you open a new session.

Think of it as the onboarding note you would hand a capable new team member on day one. Short, specific, and kept up to date.

Where it can live

<code>./CLAUDE.md</code>	Project memory. Lives in the root of your project and is shared with your team through git. This is the one you'll write today. (<code>./.claude/CLAUDE.md</code> works too.)
<code>~/.claude/CLAUDE.md</code>	Your personal memory, applied to every project on your machine. Good for your language, how you like answers, your general working style.
<code>CLAUDE.local.md</code>	Personal notes for one project that you don't want to share. Keep it out of git.
<code>sub/folder/CLAUDE.md</code>	Extra rules for one part of a larger project. Claude picks these up when it works in that folder.

Claude Code also keeps its own auto memory of things it learns along the way. If you want something to become a rule in CLAUDE.md, say so explicitly: "add this to CLAUDE.md".

Get started in four steps

- 1 Let Claude look first.** Open your project folder in Claude Code and type `/init`. Claude scans the project and writes a first draft of CLAUDE.md for you.
- 2 Shape it with the template.** Paste the "interview me" prompt (page 2) followed by the template (page 4). Claude combines what it found with what only you know, asking one question at a time.
- 3 Read it and cut.** Open the file with `/memory` or your editor. Delete anything vague, obvious or wrong. A short file that's true beats a long file that's half true.
- 4 Commit and keep it alive.** Put it in git so your team shares it. Every time you correct Claude, ask yourself whether that correction belongs in the file.

Prompts to copy into Claude Code

Select the text in this PDF and paste it straight into Claude Code.

Interview me and write my CLAUDE.md

The best starting point. Paste this, then paste the template from page 4 right below it.

Prompt · interview

I want to create a CLAUDE.md for this project. Below is a template.

1. First explore the project yourself (only files tracked by git) and fill in everything you can work out on your own: purpose, structure, tools, commands.
2. Then interview me about the rest, ONE question at a time. Offer a suggested answer where you can, so I can just say "yes" or correct you.
3. Leave out any section that doesn't apply. Don't invent rules I didn't give you.
4. Keep the result concise: aim for under 150 lines, bullet points over prose.
5. Show me the draft before writing it to ./CLAUDE.md.

Template:

Review and tighten an existing CLAUDE.md

Prompt · review

Review our CLAUDE.md as if you were a new colleague reading it on day one.

- Which instructions are vague, contradictory or out of date compared to the actual project?
- What do you keep having to guess that should be written down?
- What could be removed because it's obvious or never used?

Propose concrete edits as a list. Don't change the file until I approve.

Turn a correction into a rule

Use this right after you've corrected Claude on something you'll want next time too.

Prompt · capture

What I just corrected is a standing preference, not a one-off.

Add it to CLAUDE.md in the most relevant section, as one short, specific rule.

If a similar rule already exists, refine that one instead of adding a duplicate.

Show me the exact line you added.

Tip: you don't have to fill every template section on day one. A CLAUDE.md with just a purpose, your commands and a few golden rules already makes a noticeable difference.

Best practices

Do

- Be specific and checkable. Rules Claude can actually verify get followed.
- Write down the commands you type every day, so Claude stops guessing how to build or test.
- Point to good examples ("follow the pattern in `orders.ts`") instead of pasting them in.
- Say what Claude must ask before doing. A clear do/ask-first table prevents most surprises.
- Treat it like code: commit it, review changes, update it when the project moves.
- Use `@path/to/file.md` to import longer docs instead of copying them in.

Don't

- Put secrets, passwords or tokens in it. It gets committed and shared.
- Write a novel. Every line costs context in every session; long files dilute the rules that matter.
- Repeat what Claude can see in the code anyway, like a full file listing.
- Shout everything. Save "IMPORTANT" or "NEVER" for the two or three rules that truly are.
- Let it go stale. An outdated rule is worse than no rule, because Claude will trust it.

Vague versus specific

~~Write clean code:~~

Use 2-space indentation; run `npm run validate` before saying a task is done.

~~Be careful with data:~~

Only run scripts against dev or test. Ask me before anything touches acceptance.

~~Keep me in the loop:~~

End every task with a three-line summary: what changed, what you checked, what's still open.

A small challenge for today

By the end of the hackathon, have a CLAUDE.md with at least a purpose, your three most-used commands, three golden rules and a do/ask-first table. Then start a fresh session and ask Claude: "Based on CLAUDE.md, what are you allowed to do here, and what should you ask me first?" If the answer surprises you, your file needs another pass.

Read more: docs.claude.com/en/docs/claude-code/memory · Made for our Claude Code hackathon.

The template

A general skeleton, stripped down from a CLAUDE.md that works well for one of our own test automation teams. Usable for any kind of work: code, documents, data, analysis. Fill in the [brackets], delete what you don't need.

CLAUDE.md template

```
# CLAUDE.md – [Project name]

> Standing instructions for Claude in this project. Loaded at the start of
> every session. Keep it short, specific and true.

## 1. What this project is
| Field | Value |
|-----|-----|
| Purpose | [What this project does / produces, in one sentence] |
| Audience | [Who uses the result] |
| Tools / stack | [Languages, frameworks, apps, data sources] |
| Environments | [e.g. dev / test / acceptance; which are off-limits] |
| Owner | [Who decides on scope and direction] |

## 2. How we work together
Claude acts as a skilled [junior colleague / assistant]; I am the lead.
- Do the work proactively and to a high standard.
- Propose improvements with your reasoning; don't silently decide.
- Say how confident you are. Ask instead of guessing when it's ambiguous.

| Claude may do on its own | Claude asks me first |
|-----|-----|
| [e.g. edit files in this repo] | [e.g. delete files, bulk changes] |
| [e.g. run checks / tests] | [e.g. anything touching production] |
| [e.g. draft proposals] | [e.g. anything outward-facing or costly] |

### 2.1 Learning protocol
When I correct you or reveal a new preference that should last, add it to
this file in the same turn (refine an existing rule rather than duplicating),
and tell me what you recorded. One-off instructions stay in the conversation.

## 3. Where things live
- `[folder/]` – [what's in it]
- `[folder/]` – [what's in it]
- [Where a newcomer should look first]

## 4. How to run, build and check
- `[command]` – [what it does]
- `[command]` – [what it does]
- Before calling work done: [e.g. run lint/tests, re-read the brief]

## 5. Golden rules
1. [The thing that must never go wrong, e.g. never touch production data]
2. [A quality bar, e.g. every change gets a test / a source]
3. [A consistency rule, e.g. follow existing patterns before inventing new ones]

## 6. Conventions
- [Naming, style, language (e.g. code in English, UI text in Dutch)]
- [Formatting / templates / tone of voice]
- [Patterns to copy: "see [file] as the example"]

## 7. Common pitfalls
| Symptom | Likely cause | What to do |
|-----|-----|-----|
| [What goes wrong] | [Why] | [How to handle it] |

## 8. Boundaries
- Secrets never enter git, logs or output. They live in [.env / vault].
- Don't read files excluded by .gitignore unless I name them explicitly.
- [Git: e.g. never push; only commit when asked]

## 9. Communication
- [Language to respond in, level of detail, format preferences]
- [e.g. "Summarise what you changed at the end of each task"]

---
*Last reviewed: [date]. Update this file when the project changes.*
```