



Writing a good `claude.md`

The one file that keeps your AI coding agent from reinventing your project on every prompt. What it is, what belongs in it, how to write it, and how to stop it from rotting as your codebase grows.

STEP 1

Scaffold

Run `/init` to draft from your real codebase.



STEP 2

Harden

Use `PRD.md` to sharpen rules and conventions.



STEP 3

Maintain

Update the moment a convention changes.

WHO THIS GUIDE IS FOR

Beginners Never heard of this file? This explains what it does and why skipping it makes every session worse than the last.

Technical readers Already have a `CLAUDE.md` that turned into a junk drawer? This covers what to cut, what to keep, and how to keep it from rotting as the project grows.



01 · WHAT IT IS

Standing context, read **every** turn

CLAUDE.md sits at the root of your project and is read by Claude Code automatically, every session, before it does anything else. It's not documentation for humans — it's the context you'd otherwise re-explain at the start of every conversation.

WITHOUT IT, EVERY SESSION STARTS FROM ZERO

The agent doesn't know your folder structure, your naming conventions, which package manager you use, or that you decided weeks ago that all Firestore calls go through /hooks and nowhere else.

So it guesses. Sometimes the guess is fine. Sometimes it invents a second data-access pattern next to your real one — and you don't notice until you're debugging why two screens read the same data two ways.

PRD.md vs. CLAUDE.md — not the same file

PRD.md — the *what*

Features, target user, what's out of scope. You read it.
The agent reads it when planning a feature.

CLAUDE.md — the *how*

Stack, structure, conventions, rules — for this specific repo. The agent reads it on every single turn, automatically.

! **Why this changes how you write it.** CLAUDE.md is loaded into context constantly, so every line has an ongoing token cost — and a bloated file drowns out the instructions that actually matter.



02 · WHAT GOES IN IT

Short, specific, **answers** the guesses

A good CLAUDE.md answers the questions the agent would otherwise have to guess at — nothing more. Here's the actual list.

SECTION	WHAT TO WRITE
Stack declaration	Framework, language, key dependencies — stated explicitly. Don't make the agent infer your stack from package.json every session.
Folder structure	Not a full tree — just the rule. "Screens in /app, components in /components, data hooks in /hooks, nowhere else." The single highest-leverage section in the file.
Naming conventions	camelCase vs. snake_case, how you name hooks, how you name analytics events. Pick once, write it down, stop relitigating it.
Design language	Colors, type scale, spacing unit, corner radius — pasted straight in as a table. Makes every generated screen consistent without restating your palette.
Hard rules	The things you never want done without asking: "Never commit .env." "Firestore only through /hooks." "No new deps without asking." "Don't touch files outside the task."
"Before you finish"	The commands that must pass before a task is done — type-check, lint, tests. Turns "I think it works" into something the agent verifies itself.

- a** **The folder rule earns its keep.** It's the difference between a codebase that stays organized and one where a Firestore call quietly shows up inside a component six weeks in.



03 · HOW TO WRITE IT

Don't start from a **blank** page

Scaffold a draft from the codebase, then sharpen it against your PRD, then fold in your design language once it's locked.

STEP 1**Scaffold with /init**

Run `/init` at your project root. It reads your real deps, folder structure, and inferable conventions.

**STEP 2****Harden with PRD**

Once PRD.md exists, use it to sharpen CLAUDE.md so the two files agree.

**STEP 3****Add design language**

Once your palette, type, spacing and radius are set, paste them in as a compact table.

COPY-PASTE · HARDENING PROMPT

Read PRD.md, then rewrite CLAUDE.md so that a fresh agent session with no other context could correctly add a new feature to this project. Include folder structure, naming conventions, data-access rules, and a "before you finish" checklist. Keep it under 100 lines – it's read on every turn, so density matters.

! **Treat `/init` output as a first draft.** It'll get your stack right and probably miss the rules and conventions that only exist in your head.



04 · COMMON MISTAKES

Five ways it goes **wrong**

Every one of these makes the file worse at its actual job — constraining real decisions on every turn.

Write once, never touch again

A stale CLAUDE.md is worse than none — the agent confidently follows instructions that no longer match your codebase.

Letting it become a second PRD

Feature descriptions and product reasoning belong in PRD.md. Mixing *what* and *how* makes both files worse.

Writing it too long

A 400-line file doesn't inform the agent better — it buries the three rules that matter in noise. Every line costs context, every turn.

Being vague where specificity is cheap

"Use good naming" costs the same tokens as "Hooks are `use[Noun]`", e.g. `useHabits`" — only one constrains anything.

- 5 Forgetting it must work with zero other context.** The test: could a fresh session, no prior conversation, no other files open, correctly add a feature using only this file? If not, something's missing — usually the folder-structure rule or the hard rules.



05 · KEEPING IT MAINTAINED

Small, frequent, **event**-driven

Treat updates like updates to any shared infrastructure — triggered by real events, not a scheduled review.

- 1 **You add a folder or pattern** → update the folder-structure rule immediately, not "eventually."
- 2 **The agent makes the same mistake twice** → that's a missing or unclear rule. Add it.
- 3 **You finish a stage with new standing context** (design language, a schema pattern, an analytics convention) → fold the summary into CLAUDE.md, not just into your head.
- 4 **A rule stops being true** → delete it immediately. A wrong rule is more dangerous than a missing one — the agent follows it with total confidence.

A USEFUL HABIT

After any session where you corrected the agent's understanding of "how we do things here" more than once — stop and ask whether that correction belongs in CLAUDE.md. If you'll need to say it again in a future session, it does.



06 · A REAL EXAMPLE

Short enough to stay **dense**

Close to what a CLAUDE.md looks like for a small React Native app a few weeks into a build — specific enough to constrain real decisions.

COPY-PASTE · CLAUDE.MD STARTER

```
# CLAUDE.md

## Stack
- Expo SDK 52, React Native, TypeScript (strict mode)
- Firebase Auth + Firestore (@react-native-firebase)
- RevenueCat for subscriptions
- PostHog for analytics

## Folder structure
- Screens: /app (file-based routing)
- Reusable components: /components
- Data access: /hooks – ALL Firestore calls go here, never in components
- Analytics events: /lib/analytics.ts – one typed function per event

## Naming
- Hooks: use[Noun], e.g. useHabits, useHabitStreak
- Analytics events: object_action, snake_case, past tense, e.g. habit_created

## Design language
- Primary: #2F7D5B, neutral scale: gray-50 through gray-900
- Type: Inter, scale 12/14/16/20/24/32
- Spacing: 8px base unit, scale 8/16/24/32
- Radius: 12px everywhere / No drop shadows

## Rules
- Never commit .env – .env.example only
- No new dependencies without asking first
- Don't touch files outside the current task's stated scope
- Firestore listeners must unsubscribe on unmount

## Before you finish
- [ ] `npx tsc --noEmit` passes
- [ ] `npm run lint` passes
- [ ] Manually verified on device, not just simulator
```

! **Notice what's *not* there:** no product reasoning, no feature list, no explanation of why Firestore over Supabase. That context lives elsewhere. This file only answers "how do things work in this repo."



07 · RECAP

The workflow at a glance

Six moments in the life of a CLAUDE.md — from day one to every session after.

- 1 **Day one** — run `/init` in Claude Code to generate a first-draft CLAUDE.md from your actual codebase.
- 2 **Once PRD.md exists** — rewrite it so a fresh agent session could correctly build a feature from it alone.
- 3 **Once design language is set** — paste the color / type / spacing / radius table in.
- 4 **Ongoing** — update the moment a convention changes: new folder, new rule, deprecated pattern.
- 5 **Every session** — let it load automatically; never re-explain conventions it's supposed to know.
- 6 **Sanity check** — could a fresh session with zero other context act correctly on this file alone? If not, something's missing.

THE ONE-LINE TEST

If you'll need to say it again in a future session — it belongs in CLAUDE.md. If it doesn't change agent behavior in a way that matters — cut it.

Part of the AI Coding Guides series — covering the full path from ideation to App Store launch using AI-assisted development.