

Generative AI · [Follow publication](#)

🌟 Member-only story

How to Give Claude Code a Memory

Turn your forgetful AI into a reliable coding partner with just four files

7 min read · Jul 22, 2025



Nishad Ahamed

Follow

▶ Listen

📄 Share

⋮ More



Intro

Ever opened Claude Code, only to feel like you're talking to a brand-new intern who has *no idea* what you're working on? You patiently explain the project again. Then it creates a file that you already had. Or worse, skips a critical step because it "forgot."

Claude Code is brilliant but has the memory of a goldfish. 🐟 Each session starts fresh, so context vanishes. The result? Wasted time, duplicated work, and a lot of “Wait, didn’t we already do this?”

But here’s the good news: you can give Claude Code a brain. A simple **four-file framework** stores your project’s memory, so every session feels like you’re working with the *same* developer who knows exactly where things left off.

Let’s see how it works.

Why Claude Code Makes Mistakes

Claude Code isn’t really “aware” of your ongoing project. It doesn’t know what’s been built, what’s still pending, or which tools you’ve chosen.

So when you ask it to keep coding, it might:

- Recreate files it already been made
- Miss dependencies or steps
- Forget decisions about tech stack or architecture
- Leave tasks incomplete without realising it

It’s like having a helpful assistant who... just keeps forgetting what you said yesterday. The solution? **Persistent external memory** — a place where Claude can “relearn” everything before starting work.

The Simple 4-Part Framework

Here’s the fix: you create **four markdown files** that live in your repo. Claude reads these files every session before it touches a single line of code.

- **PRD** → The big picture
- **CLAUDE.md** → Rules for how Claude should behave
- **PLANNING.md** → Your project’s architecture and stack
- **TASKS.md** → A living checklist of what to build

Once you have these, Claude *always* knows what you're building, how you're building it, and what's next. No re-explaining. No confusion.

Part 1: PRD (Your North Star)

Begin with a Product Requirements Document — your narrative compass. It answers three anchoring questions: *What are we building? Who is it for? Why does it matter now?* Keep the language plain and executable rather than dreamy. Open with a one-line vision that could sit at the top of a README. Follow with a short problem paragraph that frames the user's pain in concrete terms, not buzzwords. Describe the core users (two or three succinct personas are plenty), then translate the vision into a handful of well-structured user stories. Add clear success criteria: activation%, response latency targets, retention, maybe a qualitative adoption threshold (“First 20 beta users use chat >3 days in week one”).

Don't overstuff. The PRD should *inform* the other files, not be copied verbatim into them. Treat it like a stable strategic artefact; when scope shifts, append a dated changelog note at the bottom instead of silently mutating prior intent. That visible evolution helps both you and Claude interpret later architectural shifts as deliberate choices rather than random drift.

Part 2: CLAUDE.md (Behaviour Contract)

Where the PRD states *intent*, CLAUDE.md governs *conduct*. This file tells Claude *how to behave inside this repository*. Think of it as codified ritual plus guardrails. A good CLAUDE.md is concise enough that every directive is salient, yet rich enough that Claude doesn't improvise unsafe patterns.

Open with a short purpose line ("This file defines session workflow, coding standards, and logging conventions for the Clarity Coach project."). Then define the invariant session loop in imperative form: load planning context, scan tasks, pick exactly one unblocked atomic task (unless there's an explicit chain), propose changes, await confirmation, implement, mark completion with timestamp, optionally surface follow-up suggestions. Include a modest section on style: naming conventions, module boundaries, testing expectations ("Each new service function accompanied by a minimal happy-path test"), and a principle like "Favour incremental edits; avoid whole-file rewrites unless structural refactor." Provide a structured template for session summaries so the log stays uniform over weeks.

Prompt to create CLAUDE.md

Generate a CLAUDE.md file from this PRD that will guide Claude Code sessions on this project.

Mandatory instruction to include inside CLAUDE.md

Always read PLANNING.md at the start of every new conversation, check TASKS.md before starting your work, mark completed tasks to TASKS.md immediately, and add newly discovered tasks to TASKS.md when found.

That single block is the mechanical spine. Without it, Claude reverts to amnesic improvisation. After generation, prune excess adjectives or duplicated rationale — economy here improves compliance.

Parts 3 & 4: PLANNING.md + TASKS.md (Project Memory Core)

PLANNING.md is the distilled architectural worldview. It rephrases (not repeats) the PRD in an implementation lens: current architecture diagram (described textually if no image), stack choices and why ("Postgres over MongoDB for relational integrity and analytic queries"), primary data models, critical request/response flows, latency or security constraints, and any active open questions parked at the bottom. This file should feel *current*; if a decision changes, update *this* first so the next session

inherits reality. Resist the temptation to dump exploratory brainstorming here — keep it authoritative, not speculative.

Prompt to create PLANNING.md

Create a PLANNING.md file that includes vision, architecture, technology stack, and required tools list for this app.

TASKS.md is operational oxygen. It encodes momentum as a linear, visible progression. Organize tasks under milestones that reflect value slices (Setup, Data Layer, Retrieval, Chat UI, Testing & Hardening, Deployment). Each task must be atomic enough to be completed in a single Claude iteration: if you'd naturally say "Part 1" or "Phase A," it's probably two tasks. Avoid vague verbs ("handle auth") — instead, say "Implement JWT issuance endpoint (login)". When a task completes, it is converted in place to `[x]` with a date, optionally adding a brief artefact note ("created /src/lib/embed.ts").

Prompt to create TASKS.md

Create a TASKS.md file with bullet point tasks divided into milestones for building this app.

Example fragment (simple is fine):

```
## Milestone 1
- [ ] Initialize repo
- [ ] Set up React + Tailwind
- [ ] Health check endpoint
```

Because Claude will both read and edit this document, clarity of formatting matters more than aesthetic cleverness. Consistency lowers friction.

How to Use It (Daily Loop)

Every working session begins the same way: you instruct Claude to reload the canonical memory (planning + behaviour + operational state) before touching code. This enforces determinism and reduces divergence.

Prompt to initiate building

Please read *PLANNING.md*, *CLAUDE.md*, and *TASKS.md* to understand the project. Then complete the first task on *TASKS.md*.

Claude parses context, selects the top uncompleted task (or you can direct a specific one), proposes changes, implements upon approval, and marks completion. If, during execution, it uncovers missing prerequisites, it should propose and append them rather than silently compensating. That keeps the backlog an honest ledger rather than an after-the-fact fiction.

Before you clear the conversation, snapshot progress into the behavioural log so the next session cold-start fast:

Prompt to add context before clearing

Please add a session summary to *CLAUDE.md* summarizing what we've done so far.

A good summary lists: tasks completed, files created/modified, any key decisions (e.g., “Chose 1024-dim embeddings to reduce token cost”), and proposed next steps. Over time, *CLAUDE.md* becomes the chronological narrative; *PLANNING.md* remains the architectural present tense; *TASKS.md* reflects both trajectory (future) and history (checked items); the PRD stays a mostly stable intent artefact.

If something feels off, Claude reimplements a file unnecessarily or skips tests — tighten the directive language in *CLAUDE.md* or adjust task granularity. Think of the four files as tunable levers: clarity in one compensates for ambiguity elsewhere.

In Short

PRD defines *why and what*. *CLAUDE.md* enforces *how we work*. *PLANNING.md* captures *how it's built today*. *TASKS.md* encodes *what moves next*. Together, they simulate persistent memory and institutional knowledge inside a stateless AI environment. The payoff is fewer repetitions, fewer hallucinated rewrites, cleaner diffs, and a calmer development cadence.

Want a ready-to-commit template bundle or a fully worked sample for a different project type? Just say the word.

Final Thoughts

With just four simple files — PRD, *CLAUDE.md*, *PLANNING.md*, and *TASKS.md* — you turn Claude Code from a forgetful assistant into a reliable partner.

- It always remembers what the project is about
- It never redoes completed work
- It follows the same structure every time
- You have a living roadmap of progress

No more explaining the same thing. No more duplicate files. No more “uhh, what were we building again?” moments.

Next time you start a Claude Code project, set up this mini-memory system first. It takes five minutes and saves *hours* of headaches later.

This story is published on [Generative AI](#). Connect with us on [LinkedIn](#) and follow [Zeniteq](#) to stay in the loop with the latest AI stories.

Subscribe to our [newsletter](#) and [YouTube](#) channel to stay updated with the latest news and updates on generative AI. Let's shape the future of AI together!

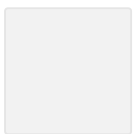
Claude

Ai Agent

Software Development

Vibe Coding

Agentic Ai



Follow

Published in Generative AI

61K followers · Last published 2 days ago

Stay updated with the latest news, research, and developments in the world of generative AI. We cover everything from AI model updates, comprehensive tutorials, and real-world applications to the broader impact of AI on society. Work with us: jimclydegm@gmail.com



Follow

Written by Nishad Ahamed

178 followers · 64 following

Hi, I am Nishad Ahamed, an IT undergrad. I am passionate about web development, Data Science, and Artificial intelligence.

Responses (10)



Bgerby

What are your thoughts?



Chris Howard
Jul 31



I've started telling it to save plans to md files. e.g I ask it to plan implementation of the rendering methods of the project I'm working on. Once we've settled on the plan, I ask it to write it to and md.

Then I can clear or exit or whatever and... [more](#)



7



1 reply

[Reply](#)



Andreas Sigloch
Aug 22



Do you really need the To Do List? since chat compression was introduced, it keeps way longer the right path. Also I am handing over right after defining requirements, to do list is agents task.



12

[Reply](#)



Alex E he/him
Jul 30




I'm not a developer.

The things I've added to Claude (and on to Claude code) as MCPs are Context7 for documentation, Git for source control, Memory to provide a journal on work completed and Task Master for planning (but not 100% on the value of... [more](#)

 3 [Reply](#)


See all responses



More from Nishad Ahamed and Generative AI


 In Towards AI by Nishad Ahamed

Build Your Own AI Sidekick with Claude Agents SDK (Beginner-Friendly Guide)

Ever wished your editor could build features from notes, fix bugs, surf unfamiliar repos, and even spin up a web server — while you sip...

 Oct 4  36


 In Generative AI by Dr. GenAI

Hierarchical Reasoning Model (HRM): a tiny brain that embarrasses giant LLMs

HRM is a 27M-parameter model trained in hours that beats giant LLMs on reasoning puzzles, proving architecture can trump scale.

★ Sep 7 🤝 306 💬 6

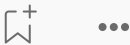
 


 In Generative AI by AI Rabbit

16 Claude Coding Traps (and the claude.md That Fixes Them)

Claude is a beast at generating code—no doubt. But if you expect perfect, secure, production-ready code on the first try... you're...

★ Sep 16 🖱️ 329 💬 9



 In Dev Genius by Nishad Ahamed

Stop Over-Engineering Subagents: A One-Scenario Playbook That Actually Ships

Short, punchy, and practical. One orchestrator. A few focused subagents. Files as truth. A tiny public widget that proves the whole...

★ Sep 6 🖱️ 7



See all from Nishad Ahamed

See all from Generative AI

Recommended from Medium


 Sevak Avakians

Spec Coding Tips for Claude Code

Helpful bits to up your Spec Coding game.

Sep 28  23  2



 Reza Rezvani

Mastering Claude Code: A 7-Step Guide to Building AI-Powered Projects with Context Engineering

From Chaos to Code: How I Reduced Development Time by 70% Using Claude Code's Hidden Power



Sep 9



159



4



Chris Evans

Choosing a Terminal on macOS (2025): iTerm2 vs Ghostty vs WezTerm vs kitty vs Alacritty

Developer-first guide with practical setups.

Sep 20



37



7




Daniel Avila

Complete Guide to setting up Git Flow in Claude Code

In this article, I'll show you how to build an automated Git Flow system using Claude Code.

Oct 1

 45

 1







Tosny

7 Websites I Visit Every Day in 2025

If there is one thing I am addicted to, besides coffee, it is the internet.



Sep 23

 3.2K

 129







In Coding Nexus by Algo Insights

How to Build Your Own AI Rig for Running Local LLMs (Gemma, Mistral, Qwen, GPT-OSS and Llama)

About three months ago, I realised I was utterly dependent on companies that didn't care about anything except power, Money, and control.



4d ago



305



6



See more recommendations