

MINI PACK

Two AIs, One Project

Get Claude and ChatGPT working together: plan in one, build in the other, and check the work.

Written for use with ChatGPT, Claude, Claude Code, Cursor or Codex · Not yet tested against them

9 prompts · 4 sections

Welcome! How to Use This Pack

Maybe you plan in ChatGPT and build in Claude Code. Maybe you build in Cursor and want Claude to double-check the work. Either way, you've probably hit the same wall: each AI starts from zero. You copy a wall of text from one chat to the other, half the context gets lost, and soon neither AI knows what the other one did.

The fix isn't a secret connection between the two. It's a simple routine. Give each AI one clear job. Pass short, organized notes between them instead of whole conversations. Keep one shared file in your project that says where things stand. This pack gives you a prompt for each step.

Is there a built-in way to connect them? This pack doesn't rely on one. You pass short notes between the two yourself, which you can do with any pair of tools. There is one official option if you use Claude Code: OpenAI's Codex plugin for Claude Code. It lets Claude Code ask Codex (OpenAI's coding agent) to review your changes or take on a task. It needs Node.js and a ChatGPT account or an OpenAI API key, and it counts toward your Codex usage limits. None of the prompts in this pack need it. If you want to try it, follow the setup steps in its README on GitHub (openai/codex-plugin-cc).

One safety rule before you start: never paste API keys, passwords, or the contents of your `.env` file into either AI. When you copy code or notes between them, check for secrets first.

Using our other packs too? One Rules File for Every AI Tool keeps both tools following the same project rules. Fix My Bugs (Prompt 1.2) writes a fresh-chat handoff when you're stuck on one bug. This pack is about using two AIs at once, on purpose, every day.

How each prompt is laid out

- **Title:** what the prompt is.
- **Use this when...:** one or two sentences on what the prompt is for and when to reach for it.
- **Use in:** which AI to paste it into.
- **Fill in:** the blanks to replace before you paste.
- **The prompt:** the box you copy.
- **Tips to customize:** optional ways to change it.

Anything in **[BRACKETS IN CAPS]** is a blank to fill in.

Written for use with: ChatGPT or Claude in a normal chat for planning and reviews, and Claude Code, Cursor's Agent, or Codex for building in your project. Each prompt says which kind of AI it's for. Not yet tested against them.

The words, in one line each. The **planner** is the AI that thinks through what to build and checks the results. The **builder** is the AI that edits your code. A **handoff** is a short note that tells one AI what

the other one did. **HANDOFF.md** is a shared file in your project where that note lives, so both AIs can read it. A **diff** is a list of the exact lines that changed in your code. A **rules file** (like AGENTS.md or CLAUDE.md) holds instructions your coding tool reads at the start of a session.

Quick Start: Find Your Problem

- **Not sure which AI should do what?** Prompt 1.1.
- **Tired of re-explaining your project every time you switch?** Prompt 1.2.
- **Want one AI to plan a feature so the other can build it?** Prompt 2.1.
- **Ready to hand that plan to your builder?** Prompt 2.2.
- **Got a report back from the builder and not sure it's right?** Prompt 2.3.
- **Want the other AI to check the code before you keep it?** Prompt 3.1.
- **The two AIs suggest different fixes?** Prompt 3.2.
- **Finishing for the day?** Prompt 4.1.
- **Starting a new session and want both AIs caught up?** Prompt 4.2.

Golden rule: Only one AI edits your code at a time. The other one plans, reviews, or waits.

What's Inside

1. Give Each AI a Job (2 prompts)
2. Plan in One, Build in the Other (3 prompts)
3. Get a Second Opinion (2 prompts)
4. Keep Both on the Same Page (2 prompts)

SECTION 1

Give Each AI a Job

Two AIs doing the same job just make twice the mess. Decide who does what, then give them one place to share notes.

1.1 Pick a Job for Each AI

Use this when you have two AI tools and keep switching between them without a plan. It's for deciding which one plans, which one builds, and when to switch.

Use in: ChatGPT or Claude, in a normal chat.

Fill in: [THE AI TOOLS I HAVE] (e.g. "ChatGPT in the browser and Claude Code in my terminal"), [WHAT I'M BUILDING] (e.g. "a habit tracker web app with Supabase"), [WHERE I GET STUCK MOST] (e.g. "the AI builds things I didn't ask for, and I lose track of what changed").

Help me set up a simple routine for using two AI tools on one project. I'm a beginner, so keep it plain.

The AI tools I have: [THE AI TOOLS I HAVE]

What I'm building: [WHAT I'M BUILDING]

Where I get stuck most: [WHERE I GET STUCK MOST]

Please:

1. Ask me up to 3 short questions if anything above is unclear. Wait for my answers before going on.
2. Give each tool one main job: planner (thinks through what to build and checks results) or builder (edits the code). Only one tool should edit my code. Explain your choice in one or two sentences, based on what each tool can actually do in my setup. For example, a normal chat can't see my project files unless I paste or upload them.
3. Write my routine as 5 to 7 short steps, from "I have an idea" to "the change is done and checked."
4. Tell me exactly what I should pass from one tool to the other at each step, and what I should NOT paste (like whole conversations, secrets, or my .env file).
5. List 3 signs that I should stop and switch tools, and 3 signs I'm switching too much.

Keep the whole answer short enough to fit on one page.



Tips to customize: If you only have chat AIs (no coding tool), make one the builder by pasting in the files it needs, and keep the other as the checker. If both of your tools can edit code, still pick one builder. Two AIs editing the same files is how changes get lost.

1.2 Set Up a Shared HANDOFF.md

Use this when you're tired of re-explaining your project every time you switch AIs. It's for creating one short file in your project that says where things stand, so either AI can catch up fast.

Use in: your builder (Claude Code, Cursor's Agent, or Codex), opened in your project.

Fill in: [MY TOOLS] (e.g. "Claude Code builds, ChatGPT plans and reviews"), [MY RULES FILE] (e.g. "AGENTS.md", "CLAUDE.md", or "I don't have one"), [WHAT I'M WORKING ON NOW] (e.g. "adding a weekly streak chart").

```
I use two AI tools on this project, and I want a shared handoff file so either one
can catch up quickly. Don't change any app code.
```

```
My tools: [MY TOOLS]
```

```
My rules file: [MY RULES FILE]
```

```
What I'm working on now: [WHAT I'M WORKING ON NOW]
```

```
Please:
```

1. Look around the project so you understand it, then create HANDOFF.md in the project root with these short sections:
 - Goal: what we're building right now, in one or two sentences
 - Current state: what works, what's half-done, what's broken
 - Files touched recently: with one line on each
 - Decisions made: and why, one line each
 - Next step: the single next thing to do
 - Open questions: anything waiting on me
 - Don't touch: files or areas to leave alone
2. Fill it in from what you can see in the code and recent git history. Mark anything you're guessing with "(guess)".
3. Keep it under 60 lines. Plain words, no long explanations.
4. Suggest one short line to add to my rules file that asks the AI to read HANDOFF.md at the start of a session and update it at the end. If I don't have a rules file, tell me which one my tools read.
5. Show me the file and the suggested line before saving anything.

 **Tips to customize:** Codex and Cursor read AGENTS.md. Claude Code reads CLAUDE.md, and its docs say newer versions read AGENTS.md when your project has no CLAUDE.md. Claude Code's docs also say that a file your rules only mention in words gets read only if Claude decides to open it. Writing `@HANDOFF.md` on its own line in the rules file Claude Code reads (not inside backticks) imports the file, so it loads at the start of every session. Keep HANDOFF.md short, because an imported file loads into every session and uses up context. Rules files are guidance, not guarantees, so if an AI skips the file, just say "Read HANDOFF.md first." Our One Rules File pack helps if your rules are spread across several files.

SECTION 2

Plan in One, Build in the Other

The planner writes small, clear steps. The builder does one step at a time and reports back.

2.1 Turn My Idea Into a Build Plan the Other AI Can Follow

Use this when you want one AI to think a feature through before the other one touches your code. It's for getting a plan made of small steps, each with a clear "done when" check.

Use in: your planner (ChatGPT or Claude, in a normal chat).

Fill in: [THE FEATURE] (e.g. "users can set a reminder time for each habit"), [MY APP AND STACK] (e.g. "a Next.js app with Supabase for login and data"), [WHAT ALREADY WORKS] (e.g. "login, adding habits, checking them off"), [MY BUILDER TOOL] (e.g. "Claude Code"), [HANDOFF NOTES] (paste your HANDOFF.md, or write "none").

```
Help me plan a feature that a different AI tool will build. You're the planner.
You can't see my code, so ask me for anything you need instead of guessing.
```

```
The feature: [THE FEATURE]
My app and stack: [MY APP AND STACK]
What already works: [WHAT ALREADY WORKS]
My builder tool: [MY BUILDER TOOL]
Handoff notes: [HANDOFF NOTES]
```

```
Please:
```

1. Ask me up to 5 questions first if anything important is missing. Wait for my answers.
 2. Write the plan as 3 to 7 numbered steps. Each step should be small enough to build and check in one sitting.
 3. For each step, include: what to build, which parts of the app it probably touches (mark these as guesses, since you can't see my files), and "Done when:" a check I can do myself, like "I click Save and the reminder time still shows after a refresh."
 4. Add a short "Don't" list for the builder, like "don't change login" or "don't add new packages without asking."
 5. Add "Questions for the builder": things it should check in the real code before starting.
 6. Put the whole plan in one code block so I can copy it in one go.
- Keep the plan simple. Don't add features I didn't ask for.



Tips to customize: If the plan has more than 7 steps, the feature is probably too big. Ask the planner to split it into two features and plan the first one only. Read the "Done when" checks

yourself. If you can't picture doing one, ask the planner to make it more concrete.

2.2 Build One Step and Report Back

Use this when you have a plan from your planner and want the builder to follow it without running ahead. It's for building one step, checking it, and writing a short report you can carry back.

Use in: your builder (Claude Code, Cursor's Agent, or Codex), opened in your project.

Fill in: [THE PLAN] (paste the plan from Prompt 2.1), [STEP NUMBER] (e.g. "1").

Another AI wrote the plan below. You're the builder. Work on step [STEP NUMBER] only.

The plan:
[THE PLAN]

Please:

1. Before writing any code, read the files this step touches. Answer the plan's "Questions for the builder." Tell me if anything in the plan doesn't match the real code. If something's wrong, stop and tell me instead of working around it.
2. Tell me in 3 to 5 lines what you're going to change. Wait for my OK.
3. Build step [STEP NUMBER] only. Follow the plan's "Don't" list. Don't start the next step.
4. Check your work against that step's "Done when" line. Run any tests, the build, or the linter this project uses. Tell me exactly how I can check it myself.
5. Write a short report for the planner in one code block, under 150 words:
 - Step: the number and name
 - What changed: one line per file
 - How it was checked: and the result
 - Surprises: anything that didn't match the plan
 - Questions: anything the planner needs to decide

If you also update HANDOFF.md, show me the change first.

 **Tips to customize:** Commit your work before each step (or ask the builder to), so you can undo one step without losing the rest. If the builder says the plan doesn't match the code, that's good news. Take its note back to the planner with Prompt 2.3 instead of letting the builder improvise.

2.3 Check the Builder's Report

Use this when the builder says a step is done and you want the planner to check it before you move on. It's for comparing the report with the plan, catching gaps, and deciding the next instruction.

Use in: your planner (the same chat where you made the plan, if you still have it).

Fill in: [THE PLAN] (the plan from Prompt 2.1), [THE REPORT] (the builder's report from Prompt 2.2), [WHAT I SAW WHEN I TRIED IT] (e.g. "Save works, but the time shows in the wrong time zone").

You're the planner. The builder finished a step and sent the report below. Check it before I move on.

The plan:
[THE PLAN]

The builder's report:
[THE REPORT]

What I saw when I tried it:
[WHAT I SAW WHEN I TRIED IT]

Please:

1. Compare the report and what I saw with the step's "Done when" check. Say plainly: done, not done, or not sure.
2. List anything the builder changed that wasn't in the step, and anything in the step it skipped.
3. Answer the builder's questions, or tell me which ones I need to decide myself.
4. If the builder found surprises, update the remaining steps to fit. Show only the steps that changed.
5. Write my next message to the builder in one code block: either "fix this first" with the exact problem, or "go ahead with the next step," naming which step. Only use what's in the report and what I saw. Don't assume the code does anything the report doesn't say.

 **Tips to customize:** What you saw counts more than what the report says. If they don't match, tell the planner so. If the chat has grown very long, start a new chat with the plan, the latest report, and your HANDOFF.md, so the planner isn't working from an old version of the plan.

SECTION 3

Get a Second Opinion

A second AI is most useful as a careful reviewer, not a second builder.

3.1 Review My Changes: Bugs and Risks Only

Use this when one AI wrote some code and you want the other AI to check it before you keep it. It's for a focused review that finds real problems without rewriting everything.

Use in: the AI that did NOT write the code (a normal chat is fine).

Fill in: [WHAT THE CHANGE IS SUPPOSED TO DO] (e.g. "let users set a reminder time for each habit"), [THE DIFF] (paste the changed lines; see the tips for how to get them), [ANYTHING I'M WORRIED ABOUT] (e.g. "I'm not sure the database part is safe", or "nothing specific").

Another AI wrote the change below. Review it, but don't rewrite it. I only want bugs and risks.

What the change is supposed to do: [WHAT THE CHANGE IS SUPPOSED TO DO]

Anything I'm worried about: [ANYTHING I'M WORRIED ABOUT]

The diff:

[THE DIFF]

Please:

1. Tell me in one or two sentences whether the change looks like it does what it's supposed to.
2. List real problems only: bugs, security risks (like exposed secrets or missing permission checks), data that could be lost, and cases that would break (empty input, logged-out users, slow connections). For each one, quote the line, explain the problem in plain words, and rate it high, medium, or low.
3. Separately, list anything you can't judge without seeing more code, and which file you'd need to see.
4. Don't suggest style changes, renames, or rewrites unless they fix a real problem.
5. If you find nothing serious, say so. Don't invent problems to have something to say.

End with a short list of fixes I can paste back to the AI that wrote it.

 **Tips to customize:** To get the diff, ask your builder to "show me the git diff of everything you changed." If you use the terminal, `git diff` shows changes you haven't staged yet, and `git diff HEAD` shows everything since your last commit. Brand-new files the AI created don't show up in either one until you run `git add` on them. After that, `git diff HEAD` includes them. Check the

diff for secrets before pasting it anywhere. If you use Claude Code with the official Codex plugin, its `/codex:review` command runs a read-only Codex review of your changes without any copy and paste.

3.2 The Two AIs Disagree: Settle It With the Facts

Use this when the two AIs suggest different fixes and you don't know which one to trust. It's for testing both ideas against the real error instead of picking whichever sounds more confident.

Use in: either AI. A fresh chat is a good idea, so neither one is defending its own answer.

Fill in: [THE PROBLEM] (e.g. "reminders save but never show up"), [THE EXACT ERROR] (paste the error message or what you see, word for word), [FIX FROM THE FIRST AI], [FIX FROM THE SECOND AI], [WHAT I'VE ALREADY TRIED].

Two AI tools gave me different fixes for the same problem. Help me figure out which one is right, using the evidence. Don't just pick the one that sounds better.

The problem: [THE PROBLEM]

The exact error: [THE EXACT ERROR]

Fix from the first AI: [FIX FROM THE FIRST AI]

Fix from the second AI: [FIX FROM THE SECOND AI]

What I've already tried: [WHAT I'VE ALREADY TRIED]

Please:

1. For each fix, explain in plain words what it assumes is causing the problem.
2. Check each assumption against the exact error. Which one does the error actually support? Quote the part of the error you're relying on.
3. Suggest one small check I can do in a few minutes that would show which assumption is true, like adding a log line, checking one setting, or trying one input. Tell me what result would point to each fix.
4. Tell me which fix to try first and why. If both might be wrong, say so, and tell me what to look at instead.
5. Tell me what to undo if the first fix doesn't work, so I'm not stacking fixes on top of each other.

Don't write a third fix unless the evidence rules out both.

 **Tips to customize:** Paste the error exactly as it appears, with no summary. Small details like a file name or a line number often decide the answer. Try one fix at a time, and undo it if it doesn't help before trying the other. If you're stuck on the same bug for a while, our Fix My Bugs pack has more prompts for breaking the loop.

SECTION 4

Keep Both on the Same Page

Two minutes at the end of a session saves twenty at the start of the next one.

4.1 End-of-Session Handoff Update

Use this when you're about to stop for the day, or before you switch from one AI to the other. It's for updating HANDOFF.md so it shows where things stand now, plus a short summary you can paste into the other AI.

Use in: your builder (Claude Code, Cursor's Agent, or Codex), opened in your project.

Fill in: [HANDOFF FILE] (e.g. "HANDOFF.md"), [WHAT I WORKED ON TODAY] (e.g. "steps 1 and 2 of the reminders plan").

```
I'm finishing this session. Update [HANDOFF FILE] so the next AI, or me tomorrow,
can pick up without asking anything. Don't change any app code.
```

```
What I worked on today: [WHAT I WORKED ON TODAY]
```

Please:

1. Check what actually changed using git status, the recent commits, and the files we edited. Trust what you see in the project over what we said in chat.
2. Update every section of [HANDOFF FILE]: Goal, Current state, Files touched recently, Decisions made, Next step, Open questions, Don't touch.
3. Replace old information instead of adding to the bottom. Remove anything that's no longer true. Keep it under 60 lines.
4. Be honest about anything half-done or untested. Write "not tested" where it applies.
5. Make "Next step" one specific action, not a vague goal.
6. Show me the changes before saving.
7. Then write a summary of the handoff in one code block, under 120 words, that I can paste into my other AI.



Tips to customize: Commit HANDOFF.md along with your code, so the note always matches the code it describes. If you use Claude Code, Prompt 4.2 in our Make Your Claude Code Limit Last pack is for wrapping up before you hit your usage limit. This prompt is different: it keeps one shared file up to date for two tools.

4.2 Catch Up Before You Start

Use this when you open a new session in either AI and want it caught up before it does anything. It's for having the AI read the handoff, check it against the project, and repeat the plan back before working.

Use in: either AI. In a coding tool, it can read the file itself. In a normal chat, paste the file.

Fill in: [HANDOFF FILE] (e.g. "HANDOFF.md"; in a normal chat, paste its contents instead), [WHAT I WANT TO DO TODAY] (e.g. "finish step 3 of the reminders plan").

```
We're starting a new session. Before doing anything, catch up on where this project stands.
```

```
Handoff notes: [HANDOFF FILE]
```

```
What I want to do today: [WHAT I WANT TO DO TODAY]
```

```
Please:
```

1. Read the handoff notes.
 2. If you can see my project, check them against reality: run `git status` and look at the last 10 commits. Tell me anything that changed since the notes were written, or anything the notes say that doesn't match the code. If you can't see my project, skip this and list what you'd want me to check.
 3. Repeat the current state and the next step back to me in your own words, in 5 lines or fewer.
 4. Tell me whether what I want to do today fits the next step, or whether something else should come first.
 5. List any open questions from the notes that I need to answer before we start.
- ```
Wait for my OK before making any changes.
```



**Tips to customize:** If the AI's summary gets something wrong, fix HANDOFF.md now, not just the chat, so the other AI doesn't make the same mistake later. If the notes and the code disagree, trust the code and update the notes.

## Bonus: Magic Phrases for Working With Two AIs

- "Only one of us edits the code. You're the [PLANNER / BUILDER]."
- "Do step [NUMBER] only, then write a report for the planner."
- "Bugs and risks only. Don't rewrite it."
- "Check it against the exact error, not what sounds right."
- "Trust what's in the project over what we said in chat."
- "Read HANDOFF.md first, and repeat the next step back to me."

## Two AIs, One Project

That's the whole pack: **9 prompts across 4 sections**. Give each AI one job, pass short notes between them, keep one shared file up to date, and let the second AI check the work instead of redoing it.

**Happy building!**