

MINI PACK

Fix My Bugs

Break the loop when your AI keeps failing on the same bug.

For Cursor · Claude Code · Claude · ChatGPT · Windsurf · Lovable · Bolt · v0 · Replit

9 prompts · 3 sections

Welcome! How to Use This Pack

You paste the error. The AI "fixes" it. Same error. You paste it again. It "fixes" it a different way, and now something else is broken too. Sound familiar?

This pack is for when normal debugging has stopped working: the AI keeps going in circles, fixes create new bugs, or nothing crashes but the app still does the wrong thing. These prompts slow things down, make the AI test its guesses, and give it a clean start when the chat has gone stale.

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.
- **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. If you don't know, write **"not sure"**.

Using the Vibe Code Starter Pack too? Start with its Section 5 for everyday errors. Use this pack when those fixes aren't sticking.

Quick Start: Stuck Right Now?

1. **Stop retrying.** Use **Prompt 1.1 (Break the Loop)** in your current chat.
2. **Still stuck?** Use **Prompt 1.2 (Fresh-Chat Handoff Brief)**, then open a new chat and paste the brief.
3. **Something that used to work broke?** Go to **Section 2**.
4. **No error, just wrong results?** Go to **Section 3**.

Golden rule: If the same fix has failed twice, don't ask a third time. Change the approach, not the wording.

What's Inside

1. Breaking the Loop (3 prompts)
2. Regressions: "It Worked Yesterday" (3 prompts)
3. It Runs, But It's Still Wrong (3 prompts)

SECTION 1

Breaking the Loop

When the AI keeps trying the same kind of fix and nothing changes.

1.1 Break the Loop

Use this when you've gone back and forth on the same error two or more times. It's for stopping the AI from repeating failed fixes and making it list what's been tried and why it failed.

Fill in: [NUMBER], [ERROR OR BEHAVIOR].

```
Stop. We've tried to fix this [NUMBER] times and it's still failing. Don't write any more code yet.
```

```
The error or problem (latest version, copied exactly):  
[ERROR OR BEHAVIOR]
```

```
Please:
```

1. List every fix we've tried in this chat, in one line each, and why each one didn't work.
2. Tell me what assumption all those fixes had in common. Which of those assumptions might be wrong?
3. Give me 3 completely different explanations for the problem, including at least one that isn't in the file we keep editing (for example config, environment variables, versions, caching, or data).
4. For each explanation, give me one quick check (a command, a log line, or something to click) that would prove or rule it out WITHOUT changing code.
5. Tell me which check to run first, and wait for my result.



Tips to customize: If the AI keeps editing the same file, add: "Assume the bug is NOT in [FILE NAME] until a check proves otherwise."

1.2 Fresh-Chat Handoff Brief

Use this when the chat is long, the AI seems confused or forgetful, you've hit a usage limit, or you want to try another tool. It's for packing everything a new chat (or a different AI tool) needs into one short message, so you can start clean without losing what you've learned.

Fill in: Nothing. Paste it as is.

```
I'm going to start a new chat to fix this bug. Write a handoff brief I can paste into it. Keep it under 300 words and leave out anything that turned out to be irrelevant.
```

Include:

1. The app in one sentence, and the tech stack with versions if you know them.
2. The goal: what should happen.
3. The bug: what happens instead, with the exact latest error message.
4. How to reproduce it, step by step.
5. What we've already tried that did NOT work (one line each), so the new chat doesn't repeat it.
6. What we've ruled out, and how.
7. The files that matter most (names only), and any we should not touch.
8. The best current theory, marked as a theory.

End the brief with: "Before writing code, ask me for any file you need and suggest the first check to run."

 **Tips to customize:** In the new chat, attach or tag only the files the brief lists. A smaller, focused context often works better than a long history. Save good briefs in your project; they're useful for the next bug too.

1.3 Make a Minimal Reproduction

Use this when the bug is buried in a big app and nobody (you or the AI) can see where it starts. It's for shrinking a bug down to the smallest example that still fails, which often reveals the cause on its own.

Fill in: [TECH STACK], [WHAT GOES WRONG], [PAGE, COMPONENT, OR ROUTE].

Help me build a minimal reproduction of this bug: the smallest possible piece of code that still shows the problem.

Tech stack: [TECH STACK]

The bug: [WHAT GOES WRONG]

Where it shows up: [PAGE, COMPONENT, OR ROUTE]

Please:

1. Suggest what to remove first (unrelated components, styling, extra data, other routes) while keeping the bug.
2. Walk me through cutting things out in small steps. After each step I'll tell you whether the bug is still there.
3. If the bug disappears after a step, tell me that the last thing we removed is involved and what to look at there.
4. Once we have the smallest failing version, explain the cause and the fix.
5. Then help me apply the fix to the real app.

Do this on a new Git branch or a copy of the project, so the real app isn't touched.

SECTION 2

Regressions: "It Worked Yesterday"

Something that used to work is broken now, often after an AI fix somewhere else.

2.1 Find the Change That Broke It

Use this when a feature worked before, you don't know what broke it, and you use Git. It's for tracking down exactly which commit or change broke a feature that used to work.

Fill in: [SKILL LEVEL], [WHAT USED TO WORK], [WHAT HAPPENS], [ROUGHLY WHEN IT LAST WORKED] (e.g. "yesterday", "3 commits ago", "not sure").

A feature that used to work is now broken. Help me find exactly which change broke it. I'm a [SKILL LEVEL], so give me exact commands.

The feature: [WHAT USED TO WORK]

What happens now: [WHAT HAPPENS]

Roughly when it last worked: [ROUGHLY WHEN IT LAST WORKED]

Please:

1. Show me how to find a commit where it still worked (for example with `git log` and checking out an older commit on a separate branch).
2. Walk me through `git bisect` step by step: start, mark good and bad, test, repeat, and finish with `git bisect reset` so I'm back where I started.
3. Tell me what quick test I should do at each step to decide "good" or "bad".
4. Once we find the commit, show me its diff and explain which part most likely caused the problem.
5. Suggest a fix that keeps whatever that commit was trying to do, instead of just undoing it.

 **Tips to customize:** No Git history? Ask instead: "Help me compare the current files with my last backup and list every difference in [FEATURE]-related files."

2.2 Fix Without Breaking Something Else

Use this when every fix the AI makes seems to break a different feature. It's for getting a fix that doesn't cause new bugs, by mapping what else uses the code first.

Fill in: [BUG], [FILE NAME / FUNCTION OR "TELL ME"].

Before you fix this bug, I want to make sure the fix doesn't break anything else.

The bug: [BUG]

The file or function you want to change: [FILE NAME / FUNCTION OR "TELL ME"]

Please:

1. List everything else in the app that uses this code (other components, routes, functions, database tables).
2. For each one, tell me how the change might affect it.
3. Propose the smallest possible fix and explain why it's safe for those other places.
4. Give me a short checklist of features to click through after the fix, including the ones most at risk.
5. Make the change, then list every file you changed.

Don't rename, move, or "clean up" anything that isn't needed for this fix.

2.3 Lock It In With a Test

Use this when a bug keeps coming back, or you want proof that a fix actually works. It's for writing a small automated test that fails while the bug exists and passes once it's fixed, so the same bug can't quietly come back.

Fill in: [TECH STACK], [BUG], [STEPS].

Help me write a test that proves this bug is fixed and stops it from coming back. I've never written tests, so keep it simple.

Tech stack: [TECH STACK]

The bug: [BUG]

How to trigger it by hand: [STEPS]

Please:

1. Recommend the simplest test tool for my stack (or use the one already in my project) and give me the install command if needed.
2. Write ONE small test that describes the correct behavior. Run it (or tell me how) BEFORE fixing, and confirm it fails for the right reason.
3. Then fix the bug and run the test again to show it passes.
4. Show me the actual test output both times. If you can't run it, say so and tell me the command.
5. Tell me how to run all my tests with one command before each commit.

 **Tips to customize:** Add to your rules file: "When you fix a bug, add a test that would have caught it."

SECTION 3

It Runs, But It's Still Wrong

No red errors, but the app still misbehaves.

3.1 It Only Breaks Sometimes

Use this when the bug happens on some clicks, some devices, or some reloads, but not others. It's for tracking down bugs that come and go: timing problems, race conditions, and caching.

Fill in: [TECH STACK], [BEHAVIOR], [WHEN IT SEEMS TO HAPPEN] (e.g. "on slow internet", "on the first load", "when I click fast", "not sure").

I have a bug that only happens sometimes. Help me find the pattern.

Tech stack: [TECH STACK]

What goes wrong: [BEHAVIOR]

When it seems to happen: [WHEN IT SEEMS TO HAPPEN]

Please:

1. List the usual causes of "sometimes" bugs in my stack, in plain English: async code finishing in a different order, double clicks or double submits, stale cached data, state updated from an old value, effects running twice in development, and timeouts.
2. Tell me how to make the bug happen on purpose (for example, slow down the network in DevTools, click quickly, or clear the cache).
3. Add temporary logs with timestamps at the key points so we can see the order things happen in.
4. Once I share the logs, help me find the cause and fix it.
5. Remind me to remove the debug logs afterward.

3.2 Works for Me, Broken for Them

Use this when someone reports a bug you can't reproduce on your computer. It's for figuring out why a bug only happens for other people, on other devices, or on the live site.

Fill in: [TECH STACK], [WHAT THEY SAID, COPIED], [DEVICE, BROWSER, LOGGED IN OR NOT, LIVE SITE OR PREVIEW].

Someone reported a bug I can't reproduce. Help me figure out what's different about their situation.

Tech stack: [TECH STACK]

Their report: [WHAT THEY SAID, COPIED]

Their setup, if known: [DEVICE, BROWSER, LOGGED IN OR NOT, LIVE SITE OR PREVIEW]

Please:

1. List the differences between my setup and theirs that most often cause this: browser and device, screen size, logged in vs logged out, a new account with no data, a different time zone or language, slow internet, browser extensions, an old cached version, and live vs local environment variables.
2. Tell me how to recreate each one on my own computer (device mode, a private window, a fresh test account, network throttling).
3. Write a short, friendly message I can send them asking for exactly the details that would help most.
4. Suggest one small piece of error logging I could add so I see these problems myself next time.

3.3 Check the Data, Not Just the Code

Use this when the code looks right, but some users or records behave strangely. It's for finding bugs caused by the data itself: missing fields, wrong types, old records, or duplicates.

Fill in: [DATABASE], [BEHAVIOR], [DESCRIPTION OR "NOT SURE"].

I think my bug might be caused by the data, not the code. Help me check.

Database: [DATABASE]

The bug: [BEHAVIOR]

Which records or users seem affected: [DESCRIPTION OR "NOT SURE"]

Please:

1. Explain the usual data problems that look like code bugs: empty or null fields, numbers stored as text, dates in different formats or time zones, old records created before a feature existed, duplicates, and records pointing to deleted items.
2. Give me read-only queries (SELECT only) to look for each of these in the affected table.
3. Help me compare a record that works with one that doesn't.
4. Suggest a fix in the code so it handles bad or missing data gracefully.
5. If the data itself needs fixing, write the update separately, explain it, and remind me to back up first.

Don't run or suggest anything that changes or deletes data until I say so.

Bonus: Magic Phrases for Stubborn Bugs

- "Don't write code yet. Give me 3 explanations and a check for each."
- "What have we tried, and why did each attempt fail?"
- "Assume the bug is not where we've been looking."
- "Show me the actual output, not what it should be."
- "What would prove this theory wrong?"
- "Write me a handoff brief for a fresh chat."

You've Got This

That's the whole Fix My Bugs pack: **9 prompts across 3 sections**. When a bug won't budge, stop retrying, change the approach, and make the AI prove each guess.

Happy debugging!