Correct incorrect answers in 10 seconds in the Claude Code session.

Use the /rewind command to undo Claude's incorrect response without losing the good work you performed earlier.

Use the /rewind command to undo Claude's incorrect response without losing the good work you performed earlier.

In the next 10 minutes, you will fix a failed session.

Here's what we're going to do: You'll open Claude Code , ask it to do something, intentionally let it malfunction, and then rewind to fix the error—retaining useful context and removing unnecessary things. The whole process takes less than 10 minutes.

Eventually, you'll develop a new reflex. Instead of typing "that doesn't work, try this," you'll press Esc and neatly request it again. Thariq Shihipar at Anthropic calls the rewind command "the only habit that signals good context management." Let's see why!

The problem with bug fixing.

Recall from Lesson 1: Everything Claude does fills your context window—reading files, responding, calling tools, and sending your messages. When you debug Claude, all the failed operations remain within the context:

Read file ? try A (failed) ? "no, try B" ? try B (failed) ? "no, try C" ? try C ?

Your context now contains: Useful file reads + two failed attempts + two error messages + the final fix. That's 5 noise items for each subsequent prompt that needs to be processed.

Now let's see what happens to the rewind function:

Read file ? [Esc Esc to return here] ? "skip A and B, go straight to C because." ? try C ?

Context : Useful file reads + an informative prompt + a bug fix. Everything is clean and focused. Claude doesn't waste attention on deadlock methods.

? **Quick check** : In the debugging example above, how many "noise" items appear in your context?

Answer : 4 - two failed attempts plus two error messages

Step-by-step guide for your first rewind

Open Claude Code in the terminal. We're going to do this for real.

Step 1. Ask Claude to read a file in your project:

```
??c file ??u vào chính c?a d? án này và gi?i thích ch?c n?ng c?a nó
```

Wait for Claude to finish. He reads the file and provides a summary. Good - this is useful context that we want to retain.

Step 2. Now, let's ask for something that might result in an error (or at least produce suboptimal results):

```
Tái c?u trúc hàm chính ?? s? d?ng m?t ki?n ??trúc hoàn toàn khác - có th?
 là m?t h? th?ng plugin
```

Claude will produce something that may or may not be correct. For this exercise, let's pretend the result is wrong – because in real-world work, this happens constantly.

Step 3. Instead of typing "correct error," double-press the **Escape** key .

You will see a list of previous messages. Select the section immediately after Claude finishes reading the file—the point where the useful context ends and the incorrect method begins.

Step 4. Review what you have learned:

```
File s? d?ng [c?u trúc mà Claude ?ã mô t?]. ??ng c? g?ng s? d?ng ki?n ??
trúc plugin - thay vào ?ó, ch? c?n trích xu?t vi?c load c?u hình vào m?
t mô-?un riêng bi?t. Gi? cho nó ??n gi?n.
```

Claude can now read the file (helpful) and has your prompt (specifically). The failed plugin refactoring has disappeared from context.

Things to note

After rewinding, check the contextual usage indicator. It should be significantly lower than after the failed attempt. That's true – you've just released tokens that would have been a burden for the rest of the session.

Also, pay attention to the quality of Claude's feedback. With clear context and specific prompts, you'll often get better results on the first try than with a debugging loop.

Tip: "Summarize from here"

Sometimes a failed approach can teach you something useful—like which libraries don't support a particular feature, or which API endpoints are outdated. You don't want to lose that knowledge when you come back.

Before returning, please enter:

```
Tóm t?t nh?ng gì chúng ta ?ã h?c ???c t? l?n th? này - ?i?u gì ?ã hi?u qu?
, ?i?u gì không, và ?i?u gì chúng ta nên th? khác ?i
```

Claude wrote a brief summary of the lessons learned. Copy that text. Then go back and paste the summary into your new prompt:

```
D?a trên l?n th? cu?i cùng c?a chúng ta: [dán b?n tóm t?t]. Bây gi? hãy th?
 ph??ng pháp B v?i nh?ng ràng bu?c này.
```

All the lessons learned, without any unnecessary context. It's like sending a note from Claude's future self to his past self.

Troubleshooting

"I don't see the previous messages after pressing Esc" - Make sure you're pressing the Escape key twice in quick succession, not holding it down. You need to be at the input prompt (not in the middle of a response). If Claude is still generating, wait for it to finish first.

"I rewinded too far and lost something essential" - Check the terminal's scroll bar - the text is still visible even though it's gone from Claude's context. You can manually copy the relevant sections into the next prompt.

"Rewinding doesn't seem to save much context" - The level of savings is proportional to the amount of work Claude has done after the rewind point. A quick response with just one line will save very little. A refactoring attempt with 20 tool calls will save a lot. This habit is most effective for complex tasks.

A swift victory

If that works, you've just done something that 90% of Claude Code users never do: You've cut the context midway without losing any useful work. File reading operations remain intact. The dead end is gone. And your next prompt has yielded better results because Claude isn't distracted by its own failed attempt.

This is the core habit of session management. Every other technique in this course is built on this instinct: Before you add context, ask yourself whether you should trim it first.

Key points to remember

1. The rewind key (Esc Esc or /rewind) removes everything after the selected message from context.
2. The error correction operation retains failed attempts in context; rewinding removes them.
3. Always rewind to right after the last useful task—not all the way to the beginning.
4. "Summarize from here" before rewinding to grasp the key points without distractions.
5. This habit, if practiced consistently, will prevent most contextual errors.

1. Question 1:

What is the keyboard shortcut for the /rewind command in Claude Code?

1. A. Ctrl + Z on all platforms
2. B. Esc - double-press
3. C. Press Ctrl + R anywhere in the session
4. D. Cmd + Shift + R (only on current Macs)

EXPLAIN:

Double-tapping the Escape key activates the /rewind command. It's designed for quick operation – you can get used to it in a day.

2. Question 2:

   Claude read 5 files, tried method A (which failed), and you want to try method B. Which step is the best?

   1. A. Rewind previous file readings, prompting you to try method B.
   2. B. Run the /clear command and restart the entire process from the beginning.
   3. C. Run the /compact command to summarize everything and continue.
   4. D. Enter 'That didn't work, please try method B instead'

   EXPLAIN:

   Rewinding to immediately after the file read will retain those reads in context (they are still useful) while discarding the faulty method A. Fixing the error will cause the deadlock at the end of method A to clutter your context. The /clear command will also discard the file reads.

3. Question 3:

   What happens to the messages AFTER the rewind point when you use the /rewind command?

   1. A. Completely removed from context
   2. B. Summarized and retained in the context below.
   3. C. Retain in context marked as obsolete

   EXPLAIN:

   The /rewind command removes everything after the point you rewind. That's the whole point – it removes the failed method and context pollution, keeping only the useful work from before.

Submit your work

Training results

You have completed **0** questions.

-- / --

Review the lesson

You finished reading the article "**Correct incorrect answers in 10 seconds in the Claude Code session.**" edited by the **TipsMake** team. We hope this article has provided you with many useful tech tips and tricks. You can search for similar articles on tips and guides. Thank you for reading and for following us regularly.