EcomLabs

GUIDE AND CHEAT SHEET

Claude Code rewind: when to use it, when git

How to open the rewind menu, what each of its six options does, what it cannot undo, and which git command covers you in each case. With a cheat sheet to keep next to you while you work with Claude Code.

FOR	Anyone working with Claude Code in a git repository
TESTED WITH	git 2.54 · Windows 11
UPDATED	November 3, 2026
DOCS	code.claude.com/docs/en/checkpointing
ARTICLE	ecomlabs.dev/g/rewind-en
AUTHOR	Luis Morales · @luismoralesdev

Contents

Step 01	Leave a safe point in git
Step 02	Open the rewind menu
Step 03	Pick the action
Step 04	Check what rewind cannot see
Step 05	Rewind or git
Cheat sheet	Rewind and git at a glance
When done	Checklist

STEP 01

Leave a safe point in git

Rewind is for the last few minutes of a session; the real safe point is a commit. Before the first prompt, save your own work and give the agent its own branch. Real output from a scratch repository with git 2.54:

```
$ git status --short
 M README.md
$ git add -A
$ git commit -m "Document shipping"
[main 80ae09a] Document shipping
 1 file changed, 1 insertion(+)
$ git switch -c agent/free-shipping
Switched to a new branch 'agent/free-shipping'
```

The `80ae09a` hash is a point you can return to even when the session is gone or months have passed.

STEP 02

Open the rewind menu

- Type `/rewind`. It also answers to `/checkpoint` and `/undo`.
- Or press `Esc` twice while the prompt input is **empty**.
- If the input has text, double `Esc` clears it instead of opening the menu. The text is saved to your input history: press Up to get it back.

The menu lists the prompts you sent in the session. Claude Code saves a checkpoint before every prompt that starts a turn, with copies of the files Claude changes through Write, Edit or NotebookEdit. It keeps the 100 most recent per session, they are saved with the conversation (so they work after `claude --continue` or `claude --resume`), and the retention sweep deletes the snapshots about 30 days later, per `cleanupPeriodDays`.

STEP 03

Pick the action

OPTION (AS SHOWN)	WHAT IT DOES	USE IT WHEN…
Restore code and conversation	Code and conversation go back to that point	Claude went the wrong way from the start
Restore conversation	Only the conversation goes back; code stays	The code is fine, the conversation drifted
Restore code	Only the code goes back; conversation stays	The code is wrong, but the discussion is useful
Summarize from here	Summarizes from that point to the end	You need context space and the latest part was noise
Summarize up to here	Summarizes what came before, keeps what came after	You want the recent messages kept intact
Never mind	Changes nothing	You opened the menu by mistake

The code options only appear when that checkpoint has file changes to revert. After restoring the conversation or **Summarize from here**, the original prompt goes back into the input. A message sent while Claude was working can show up as **No code restore**: pick the prompt that started that turn.

> CAUTION
>
> If you restore the conversation and quit without sending another prompt, the messages you removed can come back when you resume. Send a prompt before you exit.

STEP 04

Check what rewind cannot see

A checkpoint holds files Claude edited with its own tools. These do not come back:

- Files only a Bash command changed: `rm`, `mv`, `cp`, `sed -i`, a formatter, `npm install`. (Once Claude has edited a file with its tools, what Bash does to it afterward is undone.)
- Subagent edits to files Claude never touched, except a `context: fork` skill running in the foreground.
- Your manual edits and edits from another session running at the same time, unless they hit the same files.
- Symlinks and hard links: skipped with `Restored the code, but skipped N files`.
- Creating, moving or deleting directories, and anything that is not a file: a commit, a push, a migration, a deploy.

So after every rewind, check git. In the demo, Claude edited `shipping.js` and created `shipping.test.js`; Bash deleted `coupons.js` and changed `cart.js`:

```
$ git status --short
 M src/cart.js
 D src/coupons.js
 M src/shipping.js
?? src/shipping.test.js
$ git restore src/cart.js src/coupons.js
$ git status --short
 M src/shipping.js
?? src/shipping.test.js
```

What is left in the second list is what **Restore code** does cover: the file Claude edited and the new file it created with its tools.

STEP 05

Rewind or git

SITUATION	USE
Claude went wrong a few prompts ago, in this session	`/rewind`, **Restore code**
The conversation drifted, the code is fine	`/rewind`, **Restore conversation**
Bash deleted, moved or reformatted files	`git status` and `git restore <file>`
A subagent changed other files	`git restore <file>`
Shelve the attempt and start clean	`git stash push -u -m "..."`
The change is good	`git add -A` and `git commit -m "..."`
Another session, another person, or 30+ days later	git
The agent already committed	`git reset` on its branch, or `git revert` if shared

```
$ git stash push -u -m "free shipping attempt"
Saved working directory and index state On agent/free-shipping: free shipping attempt
$ git stash list
stash@{0}: On agent/free-shipping: free shipping attempt
```

`-u` includes new files git does not track yet. `git stash pop` brings it back; `git stash drop` throws it away.

CHEAT SHEET

Rewind and git at a glance

YOU WANT TO…	DO
Open the menu	`/rewind` · `Esc` `Esc` (empty input)
Undo code and conversation	**Restore code and conversation**
Undo only the code	**Restore code**
Undo only the conversation	**Restore conversation**
Free up context	**Summarize from here** · **Summarize up to here**
Try another path, keep this one	`/branch` · `claude --continue --fork-session`
Keep snapshots longer	`"cleanupPeriodDays": 60` in `settings.json`
See what changed	`git status --short` · `git diff`
Safe point	`git add -A` · `git commit -m "..."`
Branch for the agent	`git switch -c agent/<task>`
Bring a file back	`git restore <file>`
Shelve everything, new files too	`git stash push -u` · `git stash pop`

WHEN DONE

Checklist

- [ ] A commit and the agent's branch before the first prompt.
- [ ] You open the menu with `/rewind`, or `Esc` twice on an empty input.
- [ ] You pick the option by what went wrong: code, conversation or both.
- [ ] After a rewind, you check `git status` for anything Bash left behind.
- [ ] You review and revert background subagent changes with git.
- [ ] Anything worth keeping ends up in a commit: a checkpoint is not history.