GUÍA Y CHULETA

# Claude Code rewind: cuándo usarlo y cuándo git

Cómo abrir el menú de rewind, qué hace cada una de sus seis opciones, qué no deshace y qué comando de git te cubre en cada caso. Con una chuleta para tener a la mano mientras trabajas con Claude Code.

| | |
|---|---|
| PARA | Quien trabaja con Claude Code en un repositorio git |
| PROBADO CON | git 2.54 · Windows 11 |
| ACTUALIZADA | 3 de noviembre de 2026 |
| DOCUMENTACIÓN | code.claude.com/docs/en/checkpointing |
| ARTÍCULO | ecomlabs.dev/g/rewind |
| AUTOR | Luis Morales · @luismoralesdev |

# Índice


| | |
|---|---|
| Paso 01 | Deja un punto seguro en git |
| Paso 02 | Abre el menú de rewind |
| Paso 03 | Elige la acción |
| Paso 04 | Revisa lo que rewind no ve |
| Paso 05 | Rewind o git |
| Chuleta | Rewind y git a la mano |
| Al terminar | Checklist |

PASO 01

## Deja un punto seguro en git

Rewind es para los últimos minutos de una sesión; el punto seguro de verdad es un commit. Antes del primer prompt, guarda lo tuyo y dale al agente su rama. Salida real de un repositorio de prueba con git 2.54:

```
$ git status --short
 M README.md
$ git add -A
$ git commit -m "Documenta el envío"
[main ca01a57] Documenta el envío
 1 file changed, 1 insertion(+)
$ git switch -c agente/envio-gratis
Switched to a new branch 'agente/envio-gratis'
```

El hash `ca01a57` es el punto al que vuelves aunque la sesión ya no exista o hayan pasado meses.

PASO 02

## Abre el menú de rewind

- Escribe `/rewind`. También responde a `/checkpoint` y `/undo`.
- O presiona `Esc` dos veces con el campo de texto **vacío**.
- Si hay texto escrito, el doble `Esc` lo borra en lugar de abrir el menú. El texto queda en el historial: recupéralo con la flecha hacia arriba.

El menú lista los prompts que enviaste en la sesión. Claude Code guarda un punto de control antes de cada prompt que inicia un turno, con copias de los archivos que Claude cambia con Write, Edit o NotebookEdit. Conserva los 100 más recientes de cada sesión, se guardan con la conversación (sirven tras `claude --continue` o `claude --resume`) y la limpieza borra las copias unos 30 días después, según `cleanupPeriodDays`.

PASO 03

## Elige la acción

| OPCIÓN (TAL COMO SALE) | QUÉ HACE | ÚSALA CUANDO… |
|---|---|---|
| **Restore code and conversation** | Código y conversación vuelven a ese punto | Claude tomó el camino equivocado desde el principio |
| **Restore conversation** | Solo la conversación vuelve; el código se queda | El código sirve, la conversación se desvió |
| **Restore code** | Solo el código vuelve; la conversación se queda | El código quedó mal, pero lo hablado sirve |
| **Summarize from here** | Resume desde ese punto hasta el final | Necesitas contexto libre y lo último fue ruido |
| **Summarize up to here** | Resume lo anterior y conserva lo posterior | Quieres conservar intactos los mensajes recientes |
| **Never mind** | No cambia nada | Abriste el menú por error |

Las opciones de código solo aparecen si ese punto tiene cambios de archivos que revertir. Tras restaurar la conversación o **Summarize from here**, el prompt original vuelve al campo de texto. Un mensaje enviado mientras Claude trabajaba puede salir como **No code restore**: elige el prompt que inició ese turno.

> **CUIDADO**
> Si restauras la conversación y sales sin enviar otro prompt, los mensajes que quitaste pueden volver al retomar la sesión. Envía un prompt antes de salir.

PASO 04

## Revisa lo que rewind no ve

Un punto de control guarda archivos que Claude editó con sus herramientas. No vuelven:

- Archivos que solo cambió un comando de Bash: `rm`, `mv`, `cp`, `sed -i`, un formateador, `npm install`. (Si Claude ya había editado ese archivo con sus herramientas, lo que Bash le hizo después sí se deshace).
- Ediciones de subagentes en archivos que Claude no tocó, salvo una skill con `context: fork` en primer plano.
- Tus ediciones a mano y las de otra sesión al mismo tiempo, salvo en los mismos archivos.
- Enlaces simbólicos y duros: se omiten con `Restored the code, but skipped N files`.
- Crear, mover o borrar carpetas, y lo que no es un archivo: un commit, un push, una migración, un despliegue.

Por eso, después de cada rewind, mira git. En la demo, Claude editó `envio.js` y creó `envio.test.js`; Bash borró `cupones.js` y cambió `carrito.js`:

```
$ git status --short
 M src/carrito.js
 D src/cupones.js
 M src/envio.js
?? src/envio.test.js
$ git restore src/carrito.js src/cupones.js
$ git status --short
 M src/envio.js
?? src/envio.test.js
```

Lo que queda en la segunda lista es lo que sí cubre **Restore code**: lo que Claude editó y el archivo nuevo que creó con sus herramientas.

---

PASO 05

## Rewind o git

| SITUACIÓN | USA |
|---|---|
| Claude se equivocó hace unos prompts, en esta sesión | `/rewind`, **Restore code** |
| La conversación se desvió, el código sirve | `/rewind`, **Restore conversation** |
| Bash borró, movió o reformateó archivos | `git status` y `git restore <archivo>` |
| Un subagente cambió otros archivos | `git restore <archivo>` |
| Apartar el intento y empezar limpio | `git stash push -u -m "..."` |
| El cambio sirve | `git add -A` y `git commit -m "..."` |
| Otra sesión, otra persona o más de 30 días | git |
| El agente ya hizo commit | `git reset` en su rama, o `git revert` si se compartió |

```
$ git stash push -u -m "intento envío gratis"
Saved working directory and index state On agente/envio-gratis: intento envío gratis
$ git stash list
stash@{0}: On agente/envio-gratis: intento envío gratis
```

`-u` incluye los archivos nuevos que git no sigue. `git stash pop` lo devuelve; `git stash drop` lo descarta.

CHULETA

## Rewind y git a la mano

| QUIERES… | HAZ |
|---|---|
| Abrir el menú | `/rewind` · `Esc` `Esc` (campo vacío) |
| Deshacer código y conversación | **Restore code and conversation** |
| Deshacer solo el código | **Restore code** |
| Deshacer solo la conversación | **Restore conversation** |
| Liberar contexto | **Summarize from here** · **Summarize up to here** |
| Probar otro camino sin perder este | `/branch` · `claude --continue --fork-session` |
| Guardar las copias más días | `"cleanupPeriodDays": 60` en `settings.json` |
| Ver qué cambió | `git status --short` · `git diff` |
| Punto seguro | `git add -A` · `git commit -m "..."` |
| Rama para el agente | `git switch -c agente/<tarea>` |
| Recuperar un archivo | `git restore <archivo>` |
| Apartar todo, con archivos nuevos | `git stash push -u` · `git stash pop` |

AL TERMINAR

## Checklist

- [ ] Commit y rama del agente antes del primer prompt.
- [ ] Abres el menú con `/rewind` o `Esc` dos veces con el campo vacío.
- [ ] Eliges la opción según lo que salió mal: código, conversación o ambos.
- [ ] Después de un rewind, revisas `git status` por lo que hizo Bash.
- [ ] Los cambios de subagentes en segundo plano los revisas y reviertes con git.
- [ ] Lo que sirve termina en un commit: el punto de control no es historial.