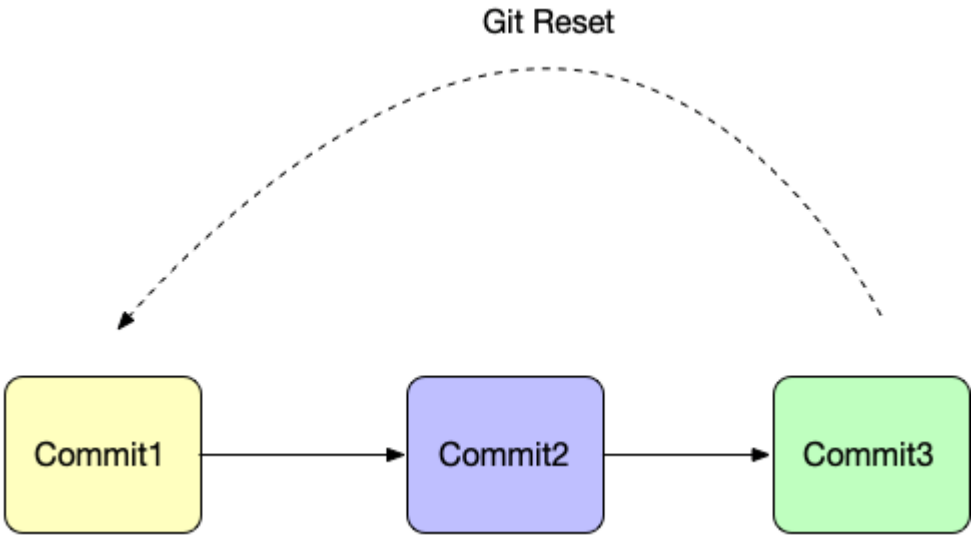
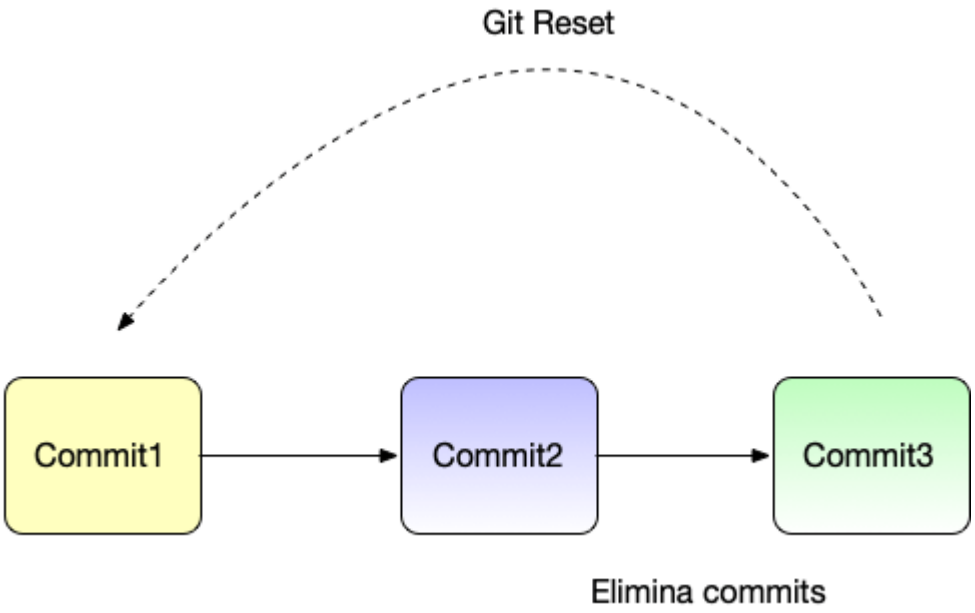
Si trabajas con Git a diario, tarde o temprano te vas a encontrar con una situación típica: hiciste un commit que no querías, agregaste archivos al staging por error o simplemente quieres volver unos pasos atrás. Para eso existe `git reset`, un comando muy útil, pero que también puede dar un poco de respeto al principio. En este artículo vamos a ver qué hace, cuáles son sus opciones principales y algunos ejemplos básicos para usarlo con más confianza.



Qué hace git reset y cuándo conviene usarlo

`git reset` sirve para mover el puntero de tu rama actual a otro commit. Dicho de una forma más simple: te permite “volver” a un estado anterior del historial. Dependiendo de la opción que uses, Git puede conservar tus cambios, sacarlos del área de preparación o eliminarlos por completo.

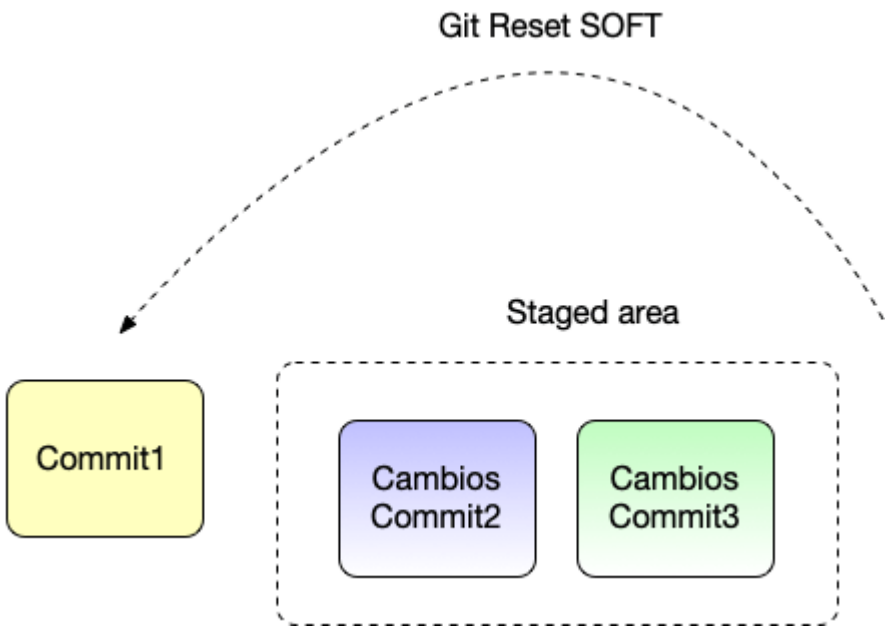


Es importante entender que `git reset` afecta principalmente a tres zonas de trabajo: el historial de commits, el área de staging y tu directorio de trabajo. El historial es donde están los commits, el staging es donde preparas los cambios antes de confirmar, y el directorio de trabajo son los archivos que estás editando en tu proyecto.

Conviene usar `git reset` cuando quieres corregir commits locales, quitar archivos del staging o deshacer cambios antes de subirlos a un repositorio remoto. Si ya hiciste `push` y otras personas pueden haber descargado esos cambios, hay que tener más cuidado, porque puedes reescribir el historial y causar conflictos al equipo.

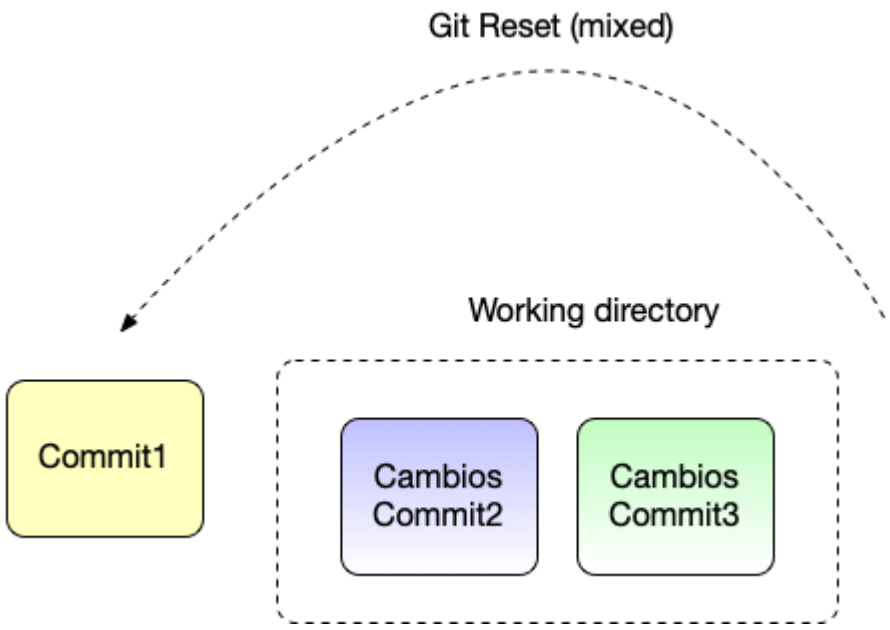
Las tres opciones clave: –soft, –mixed y –hard

La opción `--soft` mueve el historial al commit indicado, pero mantiene tus cambios en el área de staging. Es útil cuando hiciste un commit demasiado pronto y quieres rehacerlo, quizás cambiando el mensaje o agregando más archivos antes de volver a confirmar.



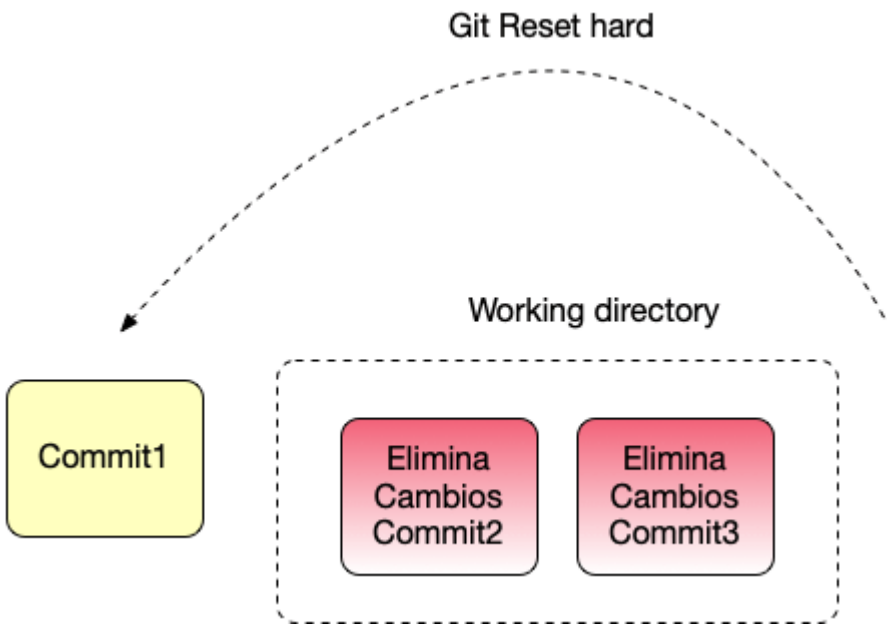

```
git reset --soft HEAD~1
```

La opción `--mixed` es la opción por defecto si no indicas ninguna. Esta mueve el historial y saca los cambios del staging, pero los conserva en tu directorio de trabajo. Es perfecta cuando hiciste `git add` o un commit por error, pero todavía quieres conservar los cambios para revisarlos con calma.




```
git reset --mixed HEAD~1
# Equivale a:
git reset HEAD~1
```

La opción `--hard` es la más potente y también la más peligrosa. Mueve el historial, limpia el staging y elimina los cambios del directorio de trabajo. Es decir, deja tu proyecto exactamente como estaba en el commit al que vuelves. Si tienes cambios sin guardar, se perderán.




```
git reset --hard HEAD~1
```

Ejemplos básicos para deshacer cambios sin miedo

Imagina que hiciste un commit, pero te diste cuenta de que el mensaje estaba mal o que te faltó incluir un archivo. Puedes usar `--soft` para deshacer el commit sin perder lo que habías preparado. Luego haces los ajustes necesarios y creas un nuevo commit.

```
git reset --soft HEAD~1
git add archivo-faltante.js
git commit -m "Mensaje correcto del commit"
```

Ahora supongamos que agregaste varios archivos al staging con `git add .`, pero no querías preparar todos. En ese caso puedes usar `git reset` sin opciones para quitar todo del staging, manteniendo los cambios en tus archivos. Después puedes agregar solo lo que realmente quieres confirmar.

```
git add .
git reset
git add archivo-correcto.js
git commit -m "Agrego solo el archivo correcto"
```

Por último, imagina que hiciste cambios de prueba, rompiste algo y quieres volver completamente al último commit. Si estás seguro de que no necesitas esos cambios, puedes usar `--hard`. Este comando eliminará las modificaciones locales y dejará el proyecto limpio.

```
git reset --hard HEAD
```

Consejos para evitar sustos al usar git reset

Antes de usar `git reset --hard`, revisa siempre el estado del repositorio con `git status`. Este comando te muestra qué archivos están modificados, cuáles están en staging y si tienes commits pendientes. Es una pequeña costumbre que puede ahorrarte muchos dolores de cabeza.

```
git status
```

También es buena idea usar `git log --oneline` para ver el historial de commits de forma resumida. Así puedes identificar mejor a qué commit quieres volver. En vez de usar siempre `HEAD~1`, puedes copiar el hash de un commit concreto y hacer reset hacia ese punto.

```
git log --oneline
git reset --mixed abc1234
```

Si tienes dudas, crea una rama temporal antes de hacer cambios arriesgados. De esta forma puedes experimentar sin miedo y volver atrás si algo sale mal. Git está pensado para ayudarte, no para complicarte la vida, pero conviene usar sus comandos con un poco de contexto.

```
git branch backup-antes-del-reset
git reset --hard HEAD~1
```

`git reset` es uno de esos comandos que parecen intimidantes al principio, pero que se vuelven muy útiles cuando entiendes qué pasa con cada opción. Como regla rápida: `--soft` conserva los cambios en staging, `--mixed` conserva los cambios pero los saca del staging, y `--hard` borra todo lo que no esté en el commit elegido. Úsalo con calma, revisa siempre con `git status` y, si tienes dudas, crea una rama de respaldo antes de tocar el historial.

- Git Commit y Buenas prácticas
- Eclipse GIT commit y como crear un repositorio
- Git Notes y Metadatos
- Eclipse Git , Repositorios locales y remotos
- Curso GIT y Buenas Prácticas