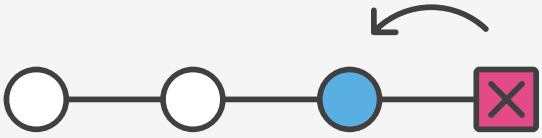
Aprende Git
Principiante
Empieza ya
Configuración de un repositorio
Guardar cambios
Examen de un repositorio
Deshacer cambios
git checkout
git clean
git revert
git reset
git rm
Reescribir el historial
Colaboración
Migración a Git
Consejos avanzados



git reset

git checkout / git clean / git revert / git reset / git rm

El comando `git reset` es una herramienta compleja y versátil para deshacer cambios. Se invoca principalmente de tres formas distintas, que se corresponden con los argumentos de líneas de comandos `--soft`, `--mixed` y `--hard`. Cada uno de los tres argumentos se corresponde con los tres mecanismos de gestión de estados internos de Git: el árbol de confirmaciones (`HEAD`), el índice del entorno de ensayo y el directorio de trabajo.

git reset y los tres árboles de Git

Para entender correctamente cómo utilizar `git reset`, primero tenemos que entender los sistemas de gestión de estados internos de Git. A veces, a estos mecanismos se les llama los "tres árboles" de Git. Puede que este nombre sea poco apropiado, ya que no son estrictamente estructuras de datos tradicionales en forma de árbol. Sin embargo, son estructuras de datos de nodos y basadas en punteros que Git utiliza para monitorizar un cronograma de ediciones. La mejor manera de demostrar estos mecanismos es crear un conjunto de cambios en un repositorio y seguirlo a lo largo de los tres árboles.

Para empezar, crearemos un nuevo repositorio con los siguientes comandos:

```
$ mkdir git_reset_test
$ cd git_reset_test/
$ git init .
Initialized empty Git repository in /git_reset_test/.git/
$ touch reset_lifecycle_file
$ git add reset_lifecycle_file
$ git commit -m"initial commit"
[main (root-commit) d386d86] initial commit
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 reset_lifecycle_file
```

El código de ejemplo anterior crea un nuevo repositorio de Git con un único archivo vacío: `reset_lifecycle_file`. En este punto, el repositorio de ejemplo tiene una única confirmación (`d386d86`) de añadir `reset_lifecycle_file`.

El directorio de trabajo

El primer árbol que examinaremos es "el directorio de trabajo". Este árbol está sincronizado con el sistema de archivos local y representa los cambios inmediatos que se realizan en el contenido de los archivos y los directorios.

```
$ echo 'hello git reset' > reset_lifecycle_file
 $ git status
 On branch main
 Changes not staged for commit:
 (use "git add ..." to update what will be committed)
 (use "git checkout -- ..." to discard changes in working
 modified: reset_lifecycle_file
```

En nuestro repositorio de ejemplo, modificamos y añadimos contenido a `reset_lifecycle_file`. Al invocar `git status`, se muestra que Git está al corriente de los cambios en el archivo. En ese momento estos cambios forman parte del primer árbol: "el directorio de trabajo". `git status` puede utilizarse para mostrar los cambios en el directorio de trabajo. Se mostrarán en rojo con el prefijo "modified".

Índice del entorno de ensayo

El siguiente es el árbol del "índice del entorno de ensayo". Este árbol monitoriza los cambios en el directorio de trabajo, que se han aplicado con `git add`, para que se almacenen en la siguiente confirmación. Este árbol es un complejo mecanismo de almacenamiento en caché interno. Por lo general, Git intenta ocultar al usuario los pormenores de la implementación del índice del entorno de ensayo.

Para ver con exactitud el estado del índice del entorno de ensayo, debemos utilizar un comando de Git menos conocido: `git ls-files`. El comando `git ls-files` es, básicamente, una utilidad de depuración que sirve para inspeccionar el estado del árbol del índice del entorno de ensayo.

```
git ls-files -s
100644 e69de29bb2d1d6434b8b29ae775ad8c2e48c5391 0   reset
```

Aquí hemos ejecutado `git ls-files` con la opción `-s` o `--stage`. Sin la opción `-s`, el resultado de `git ls-files` es simplemente una lista de nombres de archivos y rutas que forman parte del índice en ese momento. La opción `-s` muestra metadatos adicionales de los archivos del índice del entorno de ensayo. Estos metadatos son los bits de modo, el nombre de objeto y el número de entorno del contenido preparado. A nosotros nos interesa el nombre de objeto, el segundo valor (`d7d77c1b04b5edd5acfc85de0b592449e5303770`). Se trata de un hash SHA-1 de objeto de Git estándar. Es un hash del contenido de los archivos. El historial de confirmaciones almacena sus propios SHA de objeto para identificar los punteros de confirmaciones y de referencias, y el índice del entorno de ensayo tiene sus propios SHA de objeto para monitorizar versiones de archivos en el índice.

A continuación, pasaremos el archivo `reset_lifecycle_file` modificado al índice del entorno de ensayo.

```
$ git add reset_lifecycle_file
$ git status
On branch main Changes to be committed:
(use "git reset HEAD ..." to unstage)
modified: reset_lifecycle_file
```

Aquí hemos invocado `git add reset_lifecycle_file` que añade el archivo al índice del entorno de ensayo. Al invocar `git status`, ahora aparece `reset_lifecycle_file` en verde debajo de "Changes to be committed". Conviene resaltar que `git status` no es una representación verdadera del índice del entorno de ensayo. El resultado del comando `git status` muestra los cambios entre el historial de confirmaciones y el índice del entorno de ensayo. Llegados a este punto, vamos a examinar el contenido del índice del entorno de ensayo.

```
$ git ls-files -s 100644 d7d77c1b04b5edd5acfc85de0b592449
```

Podemos ver que el SHA de objeto de `reset_lifecycle_file` se ha actualizado de `e69de29bb2d1d6434b8b29ae775ad8c2e48c5391` a `d7d77c1b04b5edd5acfc85de0b592449e5303770`.

Historial de confirmaciones

El último árbol es el historial de confirmaciones. El comando `git commit` añade cambios a una instantánea permanente que reside en el historial de confirmaciones. Esta instantánea también incluye el estado que tenía el índice del entorno de ensayo en el momento de efectuar la confirmación.

```
$ git commit -am"update content of reset_lifecycle_file"
[main dc67808] update content of reset_lifecycle_file
1 file changed, 1 insertion(+)
$ git status
On branch main
nothing to commit, working tree clean
```

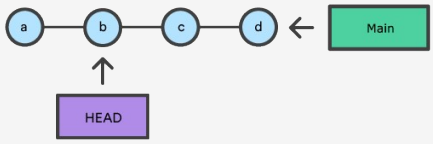
Aquí hemos creado una nueva confirmación con el mensaje `"update content of resetlifecyclefile"`. El conjunto de cambios se ha añadido al historial de confirmaciones. Al invocar `git status` en este momento, se muestra que no hay cambios pendientes en ninguno de los árboles. Al ejecutar `git log` se mostrará el historial de confirmaciones. Ahora que hemos seguido la trayectoria de este conjunto de cambios por los tres árboles, podemos empezar a utilizar `git reset`.

Funcionamiento

A nivel superficial, `git reset` tiene un comportamiento parecido a `git checkout`. Mientras que `git checkout` solo opera en el puntero de referencia `HEAD`, `git reset` moverá el puntero de referencia `HEAD` y el puntero de referencia de la rama actual. Para demostrar mejor este comportamiento, vamos a analizar el siguiente ejemplo:

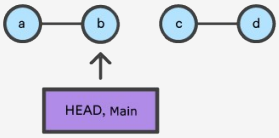
Este ejemplo demuestra una secuencia de confirmaciones en la rama `main`. La referencia `HEAD` y la referencia de la rama `main` en estos momentos apuntan a la confirmación d. Ahora vamos a ejecutar y a comparar tanto `git checkout b` como `git reset b`.

git checkout b



Con `git checkout`, la referencia `main` sigue apuntando a `d`. La referencia `HEAD` se ha movido y ahora apunta a la confirmación `b`. Ahora el repositorio se encuentra en un estado de "`HEAD` desasociado".

git reset b



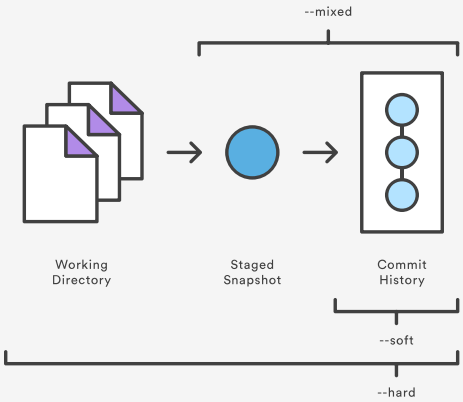
En comparación, `git reset` mueve tanto las referencias de `HEAD` como las de la rama a la confirmación especificada.

Además de actualizar los punteros de referencia de las confirmaciones, `git reset` modificará el estado de los tres árboles. La modificación del puntero de referencia sucede siempre y es una actualización del tercer árbol, el árbol de confirmaciones. Los argumentos de las líneas de comandos `--soft`, `--mixed` y `--hard` indican cómo modificar los árboles del índice del entorno de ensayo y del directorio de trabajo.

Opciones principales

La invocación predeterminada de `git reset` tiene argumentos implícitos de `--mixed` y `HEAD`. Esto significa que ejecutar `git reset` equivale a ejecutar `git reset --mixed HEAD`. De esta forma, `HEAD` es la confirmación especificada. En vez de `HEAD`, se puede usar cualquier hash de confirmación SHA-1 de Git.

The scope of git reset's modes



--hard

Esta es la opción más directa, PELIGROSA y habitual. Cuando se pasa `--hard`, los punteros de referencia del historial de confirmaciones se actualizan a la confirmación especificada. A continuación, el índice del entorno de ensayo y el directorio de trabajo se restablecen para reflejar la confirmación especificada. Todos los cambios pendientes anteriores del índice del entorno de ensayo y del directorio de trabajo se restablecen para reflejar el estado del árbol de confirmaciones. Esto significa que se perderá cualquier trabajo pendiente que haya quedado en el índice del entorno de ensayo y en el directorio de trabajo.

Para demostrarlo, continuemos con el repositorio de ejemplo de los tres árboles que hemos visto antes. En primer lugar, hagamos unas cuantas modificaciones en el repositorio. Ejecuta los siguientes comandos en el repositorio de ejemplo:

```
$ echo 'new file content' > new_file
$ git add new_file
$ echo 'changed content' >> reset_lifecycle_file
```

Estos comandos han creado un nuevo archivo llamado `new_file` y lo han añadido al repositorio. Además, se modificará el contenido de `reset_lifecycle_file`. Ahora que se han aplicado estos cambios, examinemos el estado del repositorio usando `git status`.

```
$ git status
On branch main
Changes to be committed:
   (use "git reset HEAD ..." to unstage)

new file: new_file

Changes not staged for commit:
   (use "git add ..." to update what will be committed)
   (use "git checkout -- ..." to discard changes in worki

modified: reset_lifecycle_file
```

Podemos ver que ahora hay cambios pendientes en el repositorio. El árbol del índice del entorno de ensayo tiene un cambio pendiente que se corresponde con la incorporación de `new_file`, y el directorio de trabajo tiene un cambio pendiente que se corresponde con las modificaciones de `reset_lifecycle_file`.

Antes de seguir adelante, examinemos también el estado del índice del entorno de ensayo:

```
$ git ls-files -s
100644 8e66654a5477b1bf4765946147c49509a431f963 0 new_fil
```

```
100644 d7d77c1b04b5edd5acfc85de0b592449e5303770 0 reset_l
```

Podemos ver que `new_file` se ha añadido al índice. Hemos efectuado modificaciones en `reset_lifecycle_file`, pero el SHA del índice del entorno de ensayo (`d7d77c1b04b5edd5acfc85de0b592449e5303770`) sigue siendo el mismo. Este es un comportamiento previsto, ya que no se ha usado `git add` para aplicar estos cambios en el índice del entorno de ensayo. Estos cambios existen en el directorio de trabajo.

Ahora vamos a ejecutar un comando `git reset --hard` y a examinar el nuevo estado del repositorio.

```
$ git reset --hard
HEAD is now at dc67808 update content of reset_lifecycle_
$ git status
On branch main
nothing to commit, working tree clean
$ git ls-files -s
100644 d7d77c1b04b5edd5acfc85de0b592449e5303770 0 reset_l
```

Aquí hemos ejecutado un "hard reset" utilizando la opción `--hard`. Git muestra el resultado indicando que `HEAD` apunta a la última confirmación `dc67808`. A continuación, comprobamos el estado del repositorio con `git status`. Git indica que no hay cambios pendientes. Examinamos también el estado del índice del entorno de ensayo y vemos que se ha restablecido a un punto anterior a que se añadiera `new_file`. Nuestras modificaciones en `reset_lifecycle_file` y la adición de `new_file` se han destruido. Esta pérdida de datos no se puede deshacer. Es esencial que tomemos buena nota de ello.

--mixed

Este es el modo de funcionamiento predeterminado. Se actualizan los punteros de referencia. El índice del entorno de ensayo se restablece al estado de la confirmación especificada. Todos los cambios que se hayan deshecho en el índice del entorno de ensayo se mueven al directorio de trabajo. Vamos a continuar.

```
$ echo 'new file content' > new_file
$ git add new_file
$ echo 'append content' >> reset_lifecycle_file
$ git add reset_lifecycle_file
$ git status
On branch main
Changes to be committed:
    (use "git reset HEAD ..." to unstage)

new file: new_file
modified: reset_lifecycle_file


$ git ls-files -s
100644 8e66654a5477b1bf4765946147c49509a431f963 0 new_fil
100644 7ab362db063f9e9426901092c00a3394b4bec53d 0 reset_l
```

En el ejemplo anterior, hemos hecho unas cuantas modificaciones en el repositorio. De nuevo, hemos añadido un `new_file` y modificado el contenido de `reset_lifecycle_file`. A continuación, estos cambios se aplican al índice del entorno de ensayo con `git add`. Con el repositorio en este estado, ejecutaremos ahora el restablecimiento.

```
$ git reset --mixed
$ git status
On branch main
Changes not staged for commit:
    (use "git add ..." to update what will be committed)
    (use "git checkout -- ..." to discard changes in work
```

```
    modified: reset_lifecycle_file

    Untracked files:
        (use "git add ..." to include in what will be committ

    new_file


    no changes added to commit (use "git add" and/or "git com
    $ git ls-files -s
    100644 d7d77c1b04b5edd5acfc85de0b592449e5303770 0 reset_l
```

Aquí hemos ejecutado un "mixed reset". Para que quede claro, `--mixed` es el modo predeterminado y surte el mismo efecto que ejecutar `git reset`. Al examinar el resultado de `git status` y `git ls-files`, se ve que el índice del entorno de ensayo se ha restablecido a un estado en el que `reset_lifecycle_file` es el único archivo del índice. El SHA de objeto de `reset_lifecycle_file` se ha restablecido a la versión anterior.

Lo importante que debemos destacar aquí es que `git status` nos muestra que hay modificaciones en `reset_lifecycle_file` y que hay un archivo sin seguimiento: `new_file`. Este es el comportamiento `--mixed` explícito. Se ha restablecido el índice del entorno de ensayo y se han movido los cambios pendientes al directorio de trabajo. Solo tienes que compararlo con el caso del `--hard` reset, en el que se restablecieron tanto el índice del entorno de ensayo como el directorio de trabajo, lo que hizo que se perdieran estas actualizaciones.

--soft

Cuando se pasa el argumento `--soft`, se actualizan los punteros de referencia y el restablecimiento se detiene ahí. El índice del entorno de ensayo y el directorio de trabajo permanecen intactos. Puede ser difícil demostrar claramente este comportamiento. Vamos a continuar con nuestro repositorio demo y a prepararlo para un soft reset.

```
$ git add reset_lifecycle_file
$ git ls-files -s
100644 67cc52710639e5da6b515416fd779d0741e3762e 0 reset_l
$ git status
On branch main
Changes to be committed:
(use "git reset HEAD ..." to unstage)
modified: reset_lifecycle_file
Untracked files:
(use "git add ..." to include in what will be committed)
new_file
```

Aquí hemos utilizado otra vez `git add` para pasar el `reset_lifecycle_file` modificado al índice del entorno de ensayo. Confirmamos que el índice se ha actualizado con el resultado de `git ls-files`. El resultado de `git status` ahora muestra "Changes to be committed" en verde. El `new_file` de nuestros ejemplos anteriores está flotando por el directorio de trabajo como un archivo sin seguimiento. Vamos a ejecutar rápidamente `rm new_file` para eliminar el archivo, puesto que no lo necesitaremos para los siguientes ejemplos.

Con el repositorio en este estado, ejecutaremos ahora un restablecimiento parcial (soft reset).

```
$ git reset --soft
$ git status
On branch main
Changes to be committed:
    (use "git reset HEAD ..." to unstage)

modified: reset_lifecycle_file
$ git ls-files -s
100644 67cc52710639e5da6b515416fd779d0741e3762e 0 reset_l
```

Hemos ejecutado un "soft reset". Al examinar el estado del repositorio con `git status` y `git ls-files`, se muestra que no ha cambiado nada. Es un comportamiento esperado. Un soft reset solo restablecerá el historial de confirmaciones. De manera predeterminada, `git reset` se invoca con `HEAD` como la confirmación objetivo. Como nuestro historial de confirmaciones ya se encontraba en `HEAD` y restablecemos implícitamente a `HEAD`, no ha ocurrido nada en realidad.

Para entender y utilizar mejor `--soft`, necesitamos una confirmación objetivo que no sea `HEAD`. Tenemos a `reset_lifecycle_file` en espera en el índice del entorno de ensayo. Vamos a crear una nueva confirmación.

```
$ git commit -m"prepend content to reset_lifecycle_file"
```

En este punto, nuestro repositorio debería tener tres confirmaciones. Retrocederemos en el tiempo hasta la primera de ellas. Para ello, necesitaremos el ID de la primera confirmación. Se puede saber viendo el resultado desde `git log`.

```
$ git log
commit 62e793f6941c7e0d4ad9a1345a175fe8f45cb9df
Author: bitbucket
Date: Fri Dec 1 15:03:07 2017 -0800
prepend content to reset_lifecycle_file

commit dc67808a6da9f0dec51ed16d3d8823f28e1a72a
Author: bitbucket
Date: Fri Dec 1 10:21:57 2017 -0800

update content of reset_lifecycle_file

commit 780411da3b47117270c0e3a8d5dcfd11d28d04a4

Author: bitbucket
Date: Thu Nov 30 16:50:39 2017 -0800

initial commit
```

Ten en cuenta que los ID del historial de confirmaciones serán únicos en cada sistema. Esto significa que los ID de confirmación de este ejemplo serán diferentes de los que veas en tu dispositivo. El ID de confirmación que nos interesa para este ejemplo es `780411da3b47117270c0e3a8d5dcfd11d28d04a4`. Es el ID que corresponde a la confirmación inicial (initial commit). Una vez que hayamos localizado este ID, lo utilizaremos como objetivo de nuestro soft reset.

Antes de retroceder en el tiempo, vamos a comprobar primero el estado actual del repositorio.

```
$ git status && git ls-files -s
On branch main
nothing to commit, working tree clean
100644 67cc52710639e5da6b515416fd779d0741e3762e 0 reset_l
```

Aquí ejecutamos un comando combinado de `git status` y `git ls-files -s` que nos muestra que hay cambios pendientes en el

repositorio y que `reset_lifecycle_file` del índice del entorno de ensayo se encuentra en una versión de `67cc52710639e5da6b515416fd779d0741e3762e`. Teniendo esto en cuenta, vamos a ejecutar un soft reset a nuestra primera confirmación.

```
$git reset --soft 780411da3b47117270c0e3a8d5dcfd11d28d04a
$ git status && git ls-files -s
On branch main
Changes to be committed:
    (use "git reset HEAD ..." to unstage)

modified: reset_lifecycle_file
100644 67cc52710639e5da6b515416fd779d0741e3762e 0 reset_l
```

El código anterior ejecuta un "soft reset" y también invoca el comando combinado `git status` y `git ls-files`, que muestra el resultado del estado del repositorio. Podemos examinar el resultado del estado del repositorio y sacar algunas observaciones interesantes. En primer lugar, `git status` indica que hay modificaciones en `reset_lifecycle_file` y las resalta indicando que son cambios preparados para la siguiente confirmación. En segundo lugar, la información de `git ls-files` indica que el índice del entorno de ensayo no ha cambiado y conserva el SHA 67cc52710639e5da6b515416fd779d0741e3762e que teníamos antes.

Para explicar mejor lo que ha ocurrido en este restablecimiento, vamos a examinar el `git log`:

```
$ git log commit 780411da3b47117270c0e3a8d5dcfd11d28d04a4
```

El resultado del registro ahora muestra que hay una única confirmación en el historial de confirmaciones. Esto ayuda a ilustrar claramente qué ha hecho `--soft`. Como sucede en todas las invocaciones de `git reset`, lo primero que hace el restablecimiento es restablecer el árbol de confirmaciones. Los dos ejemplos anteriores con `--hard` y `--mixed` han apuntado a `HEAD` y no han hecho que el árbol de confirmaciones retroceda en el tiempo. Durante un soft reset, esto es lo único que sucede.

Entonces, podríamos preguntarnos por qué `git status` indica que hay archivos modificados. `--soft` no toca el índice del entorno de ensayo, por lo que las actualizaciones de este nos han acompañado en el tiempo a lo largo del historial de confirmaciones. Podemos confirmarlo con el resultado de `git ls-files -s` , que muestra que no ha cambiado el SHA de `reset_lifecycle_file`. Como recordatorio, `git status` no muestra el estado de "los tres árboles", sino que, en esencia, muestra una comparación entre ellos. En este caso, muestra que el índice del entorno de ensayo va por delante de los cambios del historial de confirmaciones como si ya los hubiéramos preparado.

Diferencia entre restablecer y revertir

Si git revert es una forma "segura" de deshacer los cambios, podríamos considerar `git reset` como el método peligroso. Corremos el riesgo real de perder trabajo con `git reset`. `git reset` nunca eliminará una confirmación. Sin embargo, las confirmaciones pueden quedarse "huérfanas", es decir, sin una ruta directa desde una referencia para acceder a ellas. Normalmente estas confirmaciones

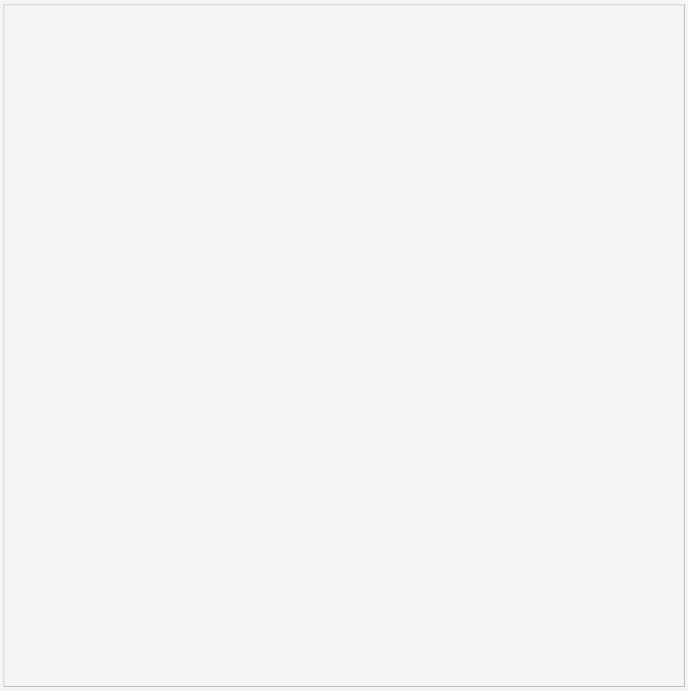
huérfanas pueden localizarse y restaurarse utilizando git reflog. Git eliminará para siempre las confirmaciones huérfanas tras ejecutar el recolector de basura interno. De manera predeterminada, Git está configurado para ejecutar el recolector de basura cada 30 días. El historial de confirmaciones es uno de los "tres árboles de Git"; los otros dos, el índice del entorno de ensayo y el directorio de trabajo, no son tan permanentes como las confirmaciones. Se debe tener cuidado al utilizar esta herramienta, ya que es uno de los únicos comandos de Git que puede hacerte perder el trabajo realizado.

Mientras que la reversión se ha diseñado para deshacer de forma segura una confirmación pública, `git reset` se ha diseñado para deshacer los cambios locales en el índice del entorno de ensayo y el directorio de trabajo. Dado que sus objetivos son diferentes, los dos comandos se implementan de forma distinta: el restablecimiento elimina por completo un conjunto de cambios, mientras que la reversión conserva el conjunto de cambios original y utiliza una nueva confirmación para aplicar la acción de deshacer.

No restablezcas el historial público

No deberías utilizar nunca `git reset` si cualquier instantánea posterior a se ha enviado a un repositorio público. Después de publicar una confirmación, tienes que dar por sentado que el resto de los desarrolladores dependen de ella.

Eliminar una confirmación que otros miembros del equipo han seguido desarrollando supone un problema grave para la colaboración. Cuando intenten sincronizarse con tu repositorio, parecerá que un pedazo del historial del proyecto ha desaparecido repentinamente. En la secuencia que aparece abajo se muestra lo que ocurre cuando intentas restablecer una confirmación pública. La rama `origin/main` es la versión de tu rama `main` local del repositorio central.



En cuanto añadas nuevas confirmaciones tras el restablecimiento, Git creerá que tu historial local se ha desviado de `origin/main`, y es probable que la confirmación de fusión necesaria para sincronizar tus repositorios confunda y frustre a tu equipo.

Lo importante es que te asegures de que estás utilizando `git reset` en un experimento local que salió mal, no en cambios publicados. Si

tienes que arreglar una confirmación pública, el comando `git revert` se ha diseñado específicamente para este fin.

Ejemplos

```
git reset <file>
```

Elimina el archivo especificado del entorno de ensayo, pero el directorio de trabajo se mantiene intacto. Deshace la preparación de un archivo sin sobrescribir ninguno de los cambios.

```
git reset
```

Restablece el entorno de ensayo para que refleje la confirmación más reciente, pero el directorio de trabajo se mantiene intacto. Deshace la preparación de todos los archivos sin sobrescribir ninguno de los cambios, lo que te da la oportunidad de volver a compilar la instantánea preparada desde cero.

```
git reset --hard
```

Restablece el entorno de ensayo y el directorio de trabajo para que reflejen la confirmación más reciente. Además de deshacer la preparación de los cambios, el indicador `--hard` le indica a Git que sobrescriba todos los cambios en el directorio de trabajo también. Dicho de otra manera: borra todos los cambios no confirmados, así que asegúrate de que realmente quieres descartar tus desarrollos locales antes de utilizarlo.

```
git reset
```

Retrocede el extremo de la rama actual a `commit`, restablece el entorno de ensayo para que coincida, pero no toques el directorio de trabajo. Todos los cambios realizados desde se encontrarán en el directorio de trabajo, que te permite volver a confirmar el historial del proyecto utilizando instantáneas más limpias y más atómicas.

```
git reset --hard
```

Retrocede el extremo de la rama actual a y restablece tanto el entorno de ensayo como el directorio de trabajo para que coincidan. Esta acción elimina no solo los cambios no confirmados, sino también todas las confirmaciones posteriores.

Cómo deshacer la preparación de un archivo

El comando `git reset` suele aparecer mientras se prepara la instantánea preparada. En el siguiente ejemplo, se da por supuesto que tienes dos archivos llamados `hello.py` y `main.py` que ya has añadido al repositorio.

```
# Edit both hello.py and main.py
```

```
# Stage everything in the current directory
git add .

# Realize that the changes in hello.py and main.py
# should be committed in different snapshots

# Unstage main.py
git reset main.py

# Commit only hello.py
git commit -m "Make some changes to hello.py"

# Commit main.py in a separate snapshot
git add main.py
git commit -m "Edit main.py"
```

Como puedes ver, `git reset` te ayudar a mantener tus confirmaciones con un enfoque muy claro al permitirte deshacer la preparación de los cambios que no están relacionados con la siguiente confirmación.

Eliminación de las confirmaciones locales

El siguiente ejemplo muestra un caso práctico más avanzado. Demuestra qué sucede cuando has estado trabajando en un nuevo experimento durante un tiempo, pero decides descartarlo por completo tras confirmar unas cuantas instantáneas.

```
# Create a new file called `foo.py` and add some code to

# Commit it to the project history
git add foo.py
git commit -m "Start developing a crazy feature"

# Edit `foo.py` again and change some other tracked files

# Commit another snapshot
git commit -a -m "Continue my crazy feature"

# Decide to scrap the feature and remove the associated c
git reset --hard HEAD~2
```

El comando `git reset HEAD~2` retrocede la rama actual dos confirmaciones, con lo que se elimina de forma eficaz las dos instantáneas que acabamos de crear a partir del historial del proyecto. Recuerda que este tipo de restablecimiento solo debe utilizarse en las confirmaciones no publicadas. No hagas nunca la operación anterior si ya has enviado tus confirmaciones a un repositorio compartido.

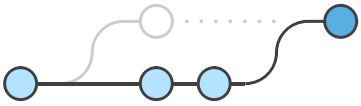
Resumen

Como resumen, `git reset` es un comando potente que sirve para deshacer los cambios locales en el estado de un repositorio de Git. `git reset` actúa en "los tres árboles de Git". Estos árboles son el historial de confirmaciones (`HEAD`), el índice del entorno de ensayo y el directorio de trabajo. Hay tres opciones de líneas de comandos que se corresponden con los tres árboles. Las opciones `--soft`, `--mixed` y `--hard` se pueden pasar a `git reset`.

En este artículo, hemos utilizado unos cuantos comandos distintos de Git para ayudar a mostrar los procesos de restablecimiento. Amplía la información sobre esos comandos en las siguientes páginas individuales: git status , git log , git add , git checkout , git reflogy git revert .

Prueba este tutorial interactivo.

Comienza ahora



Siguiente paso:

git rm

EMPEZAR EL SIGUIENTE TUTORIAL

Con tecnología de

Bitbucket

Recomendaciones

¿Quieres enterarte de los próximos artículos?

Introduce tu dirección de correo

Sitio alojado por

ATLASSIAN