

GitHub

Stash, Clean,

Cherry Pick

Bryan Ivan Montiel Ortega

Noviembre 2023

Índice

Índice.....	1
Stash.....	2
Uso del Stash.....	2
git stash list.....	2
git stash show.....	3
git stash show NOMBRE-DEL-STASH.....	3
Recuperar cambios de stash.....	3
Borrar los cambios guardados.....	3
Clean.....	5
Otras opciones en clean.....	5
Cherry-Pick.....	7
Características.....	7
Limitaciones.....	7
¿Cómo utilizar git cherry-pick?.....	7
Otros ejemplos de uso con cherry-pick.....	8
Bibliografía.....	10

Stash

Puede darse un contexto de cuando se puede usar, Git no te permite cambiar de rama por medio de `'git switch branch'` si aún no se ha hecho un `'commit'` de los cambios hechos, esto es porque sino el switch haría que los cambios se pierdan. Para ello se utiliza el `'stash'`

```
git stash
```

Este comando guarda temporalmente una versión de una rama sin mandarlo a `'commit'`, puede pensarse que es algo como `'me voy, pero deja este proyecto donde lo dejé'`. La ventaja es que lo hace de manera local y no afecta al árbol.

Por parte de *'freeCodeCamp'* tenemos lo siguiente:

Git tiene un área llamada "stash" donde puedes almacenar temporalmente una captura de tus cambios sin enviarlos al repositorio. Está separada del directorio de trabajo (working directory), del área de preparación (staging area), o del repositorio.

Esta funcionalidad es útil cuando has hecho cambios en una rama que no estás listo para realizarle commit, pero necesitas cambiar a otra rama.

Uso del Stash

Para utilizar de la mejor manera posible los stash es bueno recordar los siguientes puntos:

- no se está limitado a guardar un solo stash, cada vez que se reserva un stash va a la parte superior de la pila de stash
- es posible recuperar el último stash guardado quitándolo de la pila con `git stash pop` o dejándolo en la pila con `git stash apply`
- es posible recuperar uno de los stash anteriores usando su índice `git stash pop stash@{2}` (0 el más reciente)
- en el stash van a parar como default solo los cambios realizados en los archivos rastreados, pero es posible usar la opción `git stash -u` para incluir también los archivos untracked

```
git stash list
```

La pila de los stash nos permite saber contra qué branch y commit ha sido creado el único stash. Si tienes múltiples conjuntos de cambios guardados en stash, cada uno tendrá un índice diferente.

(nota: la pila de los stash es del repository, entonces se pueden encontrar stash desde branch locales diferentes)

```
$ git stash list
stash@{0}: WIP on main: 5002d47 our new homepage
stash@{1}: WIP on main: 5002d47 our new homepage
stash@{2}: WIP on main: 5002d47 our new homepage
```

git stash show

Si se olvidaron los últimos cambios hechos en el stash, se puede ver un resumen utilizando este comando:

git stash show NOMBRE-DEL-STASH

Si quieres ver el típico diseño "diff" (con los + y - en las líneas con los cambios), puedes incluir la opción -p (para parche o patch). La siguiente imagen es un ejemplo:

```
git stash show -p stash@{0}

# Ejemplo de un resultado:
diff --git a/PathToFile/fileA b/PathToFile/fileA
index 2417dd9..b2c9092 100644
--- a/PathToFile/fileA
+++ b/PathToFile/fileA
@@ -1,4 +1,4 @@
-What this line looks like on branch
+What this line looks like with stashed changes
```

Recuperar cambios de stash

Para recuperar los cambios del stash y aplicarlos a la rama actual en la que estás, tienes dos opciones:

1. `git stash apply NOMBRE-DEL-STASH` aplica los cambios y deja una copia en el stash
2. `git stash pop NOMBRE-DEL-STASH` aplica los cambios y elimina los archivos del stash

Puede haber conflictos cuando se aplican los cambios. Puedes resolver los conflictos de forma similar a un 'merge'.

Borrar los cambios guardados

Si quieres remover los cambios guardados en stash sin aplicarlos, ejecuta el comando:

```
git stash drop NOMBRE-DEL-STASH
```

Para limpiar todo del stash, ejecuta el comando:

```
git stash clear
```

Clean

`git-clean` - Elimina los archivos no rastreados en el árbol de trabajo.

En Git, los archivos no rastreados son aquellos que no están bajo control de versiones. Este comando elimina los archivos no rastreados del árbol de trabajo de forma recursiva, comenzando desde el directorio actual.

Si deseas eliminar también los directorios no rastreados, puedes agregar la opción `-d`.

Si deseas forzar la eliminación, puedes agregar la opción `-f`.

Una opción combinada puede verse como:

```
git clean -d -f
```

Otras opciones en clean

Así como `-d` y `-f` existen otros parámetros que se pueden añadir a `git clean` que se explicarán a continuación.

Para mostrar los archivos de una forma interactiva, estar mencionando si de verdad los borras o no de un archivo en un archivo se usa el parámetro `-i`. Solicitará a los usuarios que confirmen cada archivo que Git haya detectado como sin seguimiento:

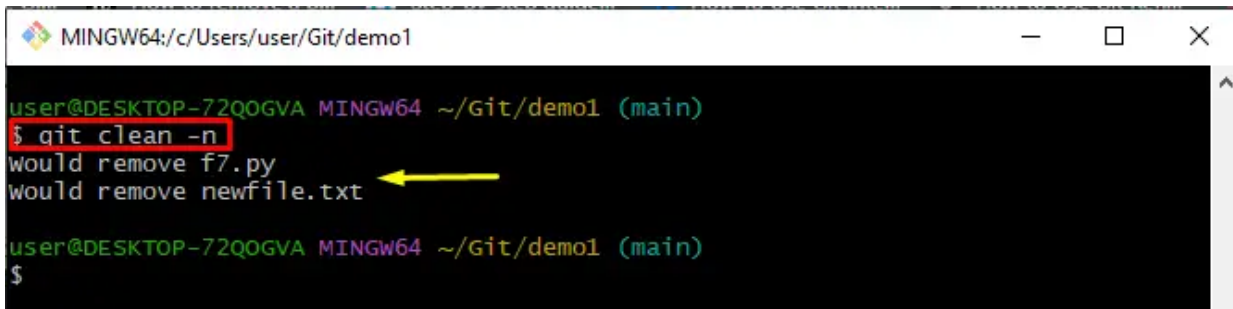
```
git clean -i
```



```
MINGW64: c:/Users/user/Git/demo1
user@DESKTOP-72QOGVA MINGW64 ~/Git/demo1 (main)
$ git clean -i
would remove the following item:
  newfile.txt
*** Commands ***
  1: clean          2: filter by pattern  3: select by numbers
  4: ask each       5: quit             6: help
what now> a
Remove newfile.txt [y/N]? N
user@DESKTOP-72QOGVA MINGW64 ~/Git/demo1 (main)
$
```

Para ver una lista de archivos sin seguimiento que se eliminarán, ejecute el comando:

```
git clean -n
```

A screenshot of a terminal window titled 'MINGW64:/c/Users/user/Git/demo1'. The prompt is 'user@DESKTOP-72QOGVA MINGW64 ~/Git/demo1 (main)'. The command '\$ git clean -n' is entered and highlighted with a red box. The output shows 'would remove f7.py' and 'would remove newfile.txt', with a yellow arrow pointing to the second line. The prompt '\$' is shown again at the bottom.

```
MINGW64:/c/Users/user/Git/demo1
user@DESKTOP-72QOGVA MINGW64 ~/Git/demo1 (main)
$ git clean -n
would remove f7.py
would remove newfile.txt
user@DESKTOP-72QOGVA MINGW64 ~/Git/demo1 (main)
$
```

Para silenciar la salida, solo informar errores, pero no los archivos que se eliminaron correctamente:

```
git clean -q
```

El siguiente usa el patrón de exclusión dado además de las reglas de ignorar estándar.

```
git clean -e <patrón>
```

El siguiente no usa las reglas de ignorar estándar, pero aún usa las reglas de ignorar dadas con opciones '-e' desde la línea de comandos. Esto permite eliminar todos los archivos no rastreados, incluidos los productos de compilación.

Esto se puede usar (posiblemente en conjunto con `git restore` o `git reset`) para crear un directorio de trabajo prístino para probar una compilación limpia.

Por últi, usando '-X' se utiliza con el comando `git clean` para eliminar solo los archivos ignorados por Git. Esto puede ser útil para reconstruir todo desde cero, pero mantener los archivos creados manualmente.

Cherry-Pick

Git `cherry-pick` se define como un importante comando del sistema de Git que se encarga de elegir uno o más `commit` o confirmaciones de cambios de una rama específica para luego aplicarla a otra rama. Es decir, `git cherry-pick` toma los cambios de una confirmación y los aplica en la rama actual.

Este comando puede ser útil para deshacer cambios, por ejemplo, si una confirmación se aplica accidentalmente en la rama equivocada. En ese caso, puedes cambiar a la rama correcta y ejecutar `'git cherry-pick'` en la confirmación para aplicarla donde debería estar.

Características

Dentro de las propiedades y características del comando Git `cherry-pick`, se encuentra que es de fácil ejecución, hasta el punto en el que, para utilizarlo, solo se necesita conocer el `commit` determinado que se desea aplicar en la rama.

Para el uso de este comando también se requiere que no se hayan llevado a cabo modificaciones del `commit` de HEAD, para poder tener un espacio de trabajo limpio.

El comando Git `cherry-pick` también se caracteriza por su utilidad para la rápida corrección de errores en el sistema, evitando que impacten sobre más usuarios. Al mismo tiempo, esta herramienta puede utilizarse para el trabajo independiente en un mismo proyecto de desarrollo, gracias a que permite ir avanzando en los procesos de manera sencilla y práctica.

Git `cherry-pick` también ofrece múltiples opciones que extienden sus funciones, como, por ejemplo:

- `-e` o `--edit`: solicita un mensaje de `commit` o confirmación antes de llevar a cabo la ejecución del comando.
- `--no-commit`: en vez de realizar una confirmación nueva, mueve el contenido al directorio de trabajo de la rama actual.

Limitaciones

Git `cherry-pick` también se caracteriza por su gran potencia de ejecución, por lo que su uso no es siempre recomendable, y debe ser cuidadoso para evitar inconvenientes como `commits` duplicados. Además, aunque la ejecución de este comando se realizará de forma exitosa, en algunos casos es preferible trabajar con fusiones tradicionales de Git.

¿Cómo utilizar `git cherry-pick`?

Para demostrar cómo utilizar `git cherry-pick`, supongamos que tenemos un repositorio con el siguiente estado de rama:


```
a - b - c - d   Main
      \
      e - f - g Feature
```

Usar `git cherry-pick` es sencillo y se puede ejecutar de la siguiente manera:

```
git cherry-pick commitSha
```

En este ejemplo, `commitSha` es una referencia de confirmación. Puedes encontrar una referencia de confirmación con el comando `git log`. En este caso, imaginemos que queremos aplicar la confirmación 'f' a la rama principal. Para ello, primero debemos asegurarnos de que estamos trabajando con la rama principal.

```
git checkout main
```

A continuación, ejecutamos `cherry-pick` con el siguiente comando:

```
git cherry-pick f
```

Una vez ejecutado, el historial de Git se verá así:

```
a - b - c - d - f   Main
      \
      e - f - g Feature
```

La confirmación f se ha introducido correctamente en la rama principal

Otros ejemplos de uso con `cherry-pick`

`git cherry pick` también se puede combinar con algunas opciones de ejecución.

```
-edit
```

Al combinar la opción `-edit`, Git solicitará un mensaje de confirmación antes de aplicar la operación `cherry-pick`.

```
--no-commit
```

La opción `--no-commit` ejecutará el comando `cherry-pick`, pero en lugar de hacer una nueva confirmación, moverá el contenido de la confirmación de destino al directorio de trabajo de la rama actual.

```
--signoff
```

La opción `--signoff` añadirá una línea de firma 'signoff' al final del mensaje de confirmación de cherry-pick.

Además de estas útiles opciones, `git cherry-pick` también admite una variedad de opciones de estrategia de fusión.

Además, `git cherry-pick` admite la introducción de opciones para la resolución de conflictos de fusión, incluidas las siguientes: `--abort`, `--continue` y `--quit`.

Bibliografía

1. Atlassian. (s/f). Git cherry pick. Atlassian. Recuperado el 10 de noviembre de 2023, de <https://www.atlassian.com/es/git/tutorials/cherry-pick>
2. Carrillo, J. (2021, febrero 6). Git Stash Explicado: Cómo Almacenar Temporalmente los Cambios Locales en Git. freecodecamp.org. <https://www.freecodecamp.org/espanol/news/git-stash-explicado/>
3. El comando Git stash en Git. (s/f). aulab.es. Recuperado el 10 de noviembre de 2023, de <https://aulab.es/articulos-guias-avanzadas/73/el-comando-git-stash-en-git>
4. Git - git-clean documentation. (s/f). Git-scm.com. Recuperado el 10 de noviembre de 2023, de <https://git-scm.com/docs/git-clean>
5. ¿Qué es Git cherry-pick? (2022, julio 19). KeepCoding Bootcamps. <https://keepcoding.io/blog/que-es-git-cherry-pick/>
6. Thomas, J. (2023, junio 30). ¿Qué es el comando “git clean”? LinuxPedia. <https://www.linuxpedia.net/git/que-es-el-comando-git-clean/>