



# GitHub

Stash

Clean

Cherry-pick

Lucero Guadalupe Domínguez Pérez

# STASH

El comando `git stash` te permite almacenar temporalmente los cambios que hayas efectuado en el código en el que estás trabajando para que puedas trabajar en otra cosa y más tarde, regresar y volver a aplicar los cambios.



Guardar los cambios en stashes resulta práctico si tienes que cambiar rápidamente de contexto y ponerte con otra cosa, pero estás en medio de un cambio en el código y no tienes todo listo para confirmar los cambios.

# EJEMPLO

El `stash` nos permite cambiar de rama o branch, hacer cambios, trabajar en otras cosas y, más adelante, retomar el trabajo con los archivos que teníamos en staging, pero que podemos recuperar, ya que los guardamos en el `stash`.

```
$ git status
On branch master
Changes to be committed:
  new file:   style.css

Changes not staged for commit:
  modified:   index.html

$ git stash
Saved working directory and index state WIP on master:
5002d47 our new homepage
HEAD is now at 5002d47 our new homepage


$ git status
On branch master
nothing to commit, working tree clean
```

# COMANDOS MAS FRECUENTES



```
git stash
```

Guarda los cambios temporalmente en memoria cuando no quieres hacer un commit aun.



```
git stash save "mensaje"
```

Guarda un stash con mensaje.




```
git stash list
```

Muestra la lista de cambios temporales.



```
git stash pop
```

Trae de vuelta los cambios que teníamos guardados en el ultimo stash.




```
git stash apply stash@{n}:
```

Trae el stash que necesites con indicar su número dentro de las llaves.



```
git stash drop
```

Borra el ultimo stash.

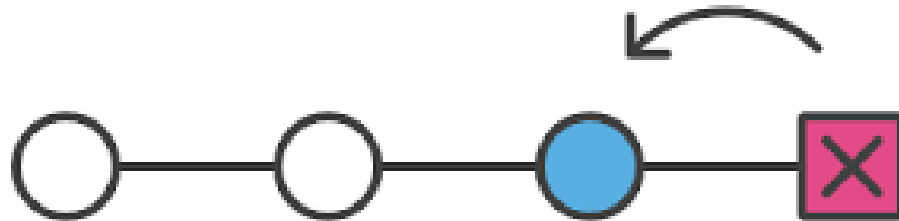


```
git stash clear
```

Borra todos los stash.

# CLEAN

Mientras estamos trabajando en un repositorio podemos añadir archivos a él, que realmente no forma parte de nuestro directorio de trabajo, archivos que no se deberían de agregar al repositorio remoto.




# COMANDOS COMUNES

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top left corner. The text 'git clean' is written in a light blue monospace font.


```
git clean
```

El comando clean actúa en archivos sin seguimiento, este tipo de archivos son aquellos que se encuentran en el directorio de trabajo, pero que aún no se han añadido al índice de seguimiento de repositorio con el comando add.

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top left corner. The text 'git clean --dry-run' is written in a light blue monospace font.

```
git clean --dry-run
```

**Revisar que archivos no tienen seguimiento**  
Para saber qué archivos vamos a borrar cuando están repetidos o no son de nuestro proyecto

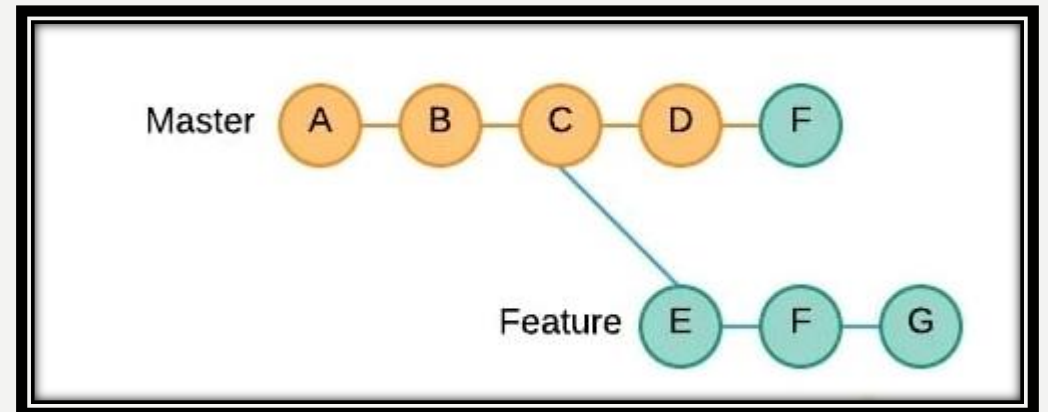
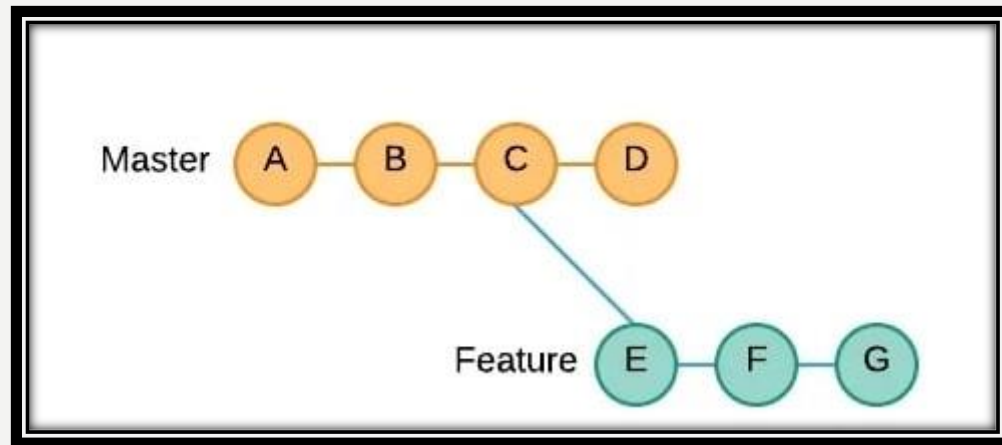
A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top left corner. The text 'git clean -f' is written in a light blue monospace font.

```
git clean -f
```

Para borrar todos los archivos listados (No borra las carpetas y lo que esta en .gitignore)

# CHERRY-PICK

Git cherry-pick es un comando en Git que selecciona y aplica commits específicos de una rama o branch a otra. Facilita la incorporación precisa de cambios, optimizando la colaboración y el mantenimiento en proyectos de desarrollo de software.





# COMANDOS COMUNES



```
git checkout rama-principal
```

Primero, debemos estar en la rama principal.



```
git cherry-pick commitSha
```

Para ejecutar o crear Cherry-pick, es el siguiente comando. Aquí, commitSha es una referencia al commit que deseas aplicar.



```
git cherry-pick --abort
```

Supongamos que estás usando GitHub para colaborar con un equipo en un proyecto y has realizado un cherry-pick de un commit de otra rama en tu rama local, pero ocurren conflictos durante este proceso y deseas detenerlo y volver al estado anterior.