En git tenemos tres estados:

1. **Working directory** aquí es donde normalmente estamos y trabajamos con nuestros proyectos es decir, aquí es donde abrimos nuestro editor de código y empezamos a hacer magia.
2. **Staging area** aquí es donde escogemos que archivos están listos para pasar al último estado y también decidimos que archivos no lo están listos por ejemplo: supongamos que tenemos un archivo html que ya esta listo y un archivo css que aun no lo esta listo. En el staging area elegimos que nuestro html pasa a la siguiente fase pero nuestro css no.
3. **Repository** aquí se registró todo los commit que hicimos a lo largo de nuestro proyecto.

Lo primero es configurar nuestro nombre. Para ello solo tenemos que ejecutar:

git config --global user.name "aqui tu nombre"

Lo siguiente es configurar nuestro email. Igual que antes ejecutamos el siguiente comando:

git config --global user.email "aquí tu email"

Si queremos ver la configuración siempre podremos ejecutar:

git config --global user.name

git config --global user.email

Si queremos ver toda nuestras configuraciones globales podemos ejecutar el comando:

git config --global --list

y esto nos va a dar un listado de nuestras configuraciones de lo que acabamos de hacer.

Primero no vamos a la terminal, nos movemos al directorio de nuestro proyecto y ejecutamos el primer comando de git:

git init

Con este comando le indicamos a git que inicialice un repositorio nuevo.

Para saber el estado que se encuentra nuestro proyecto ejecutamos:

git status

Como vimos en el punto anterior teníamos archivos y directorios que aun no teníamos añadidos en nuestro repositorio. Para añadir un archivo en el repositorio simplemente ejecutamos:

git add index.php

Ahora si volvemos a ejecutar:

git status

## Añadir archivos a la zona de escalonamiento

El comando git add command añade los archivos nuevos o modificados en tu directorio de trabajo al área de preparación de Git.  
Añade algún archivo:

git add somefile.js

Añade todos los archivos:

git add .

Tal vez en tu proyecto tengas muchos fichero y añadir uno a uno puede ser un poco coñazo. Para evitar eso git tiene un comando para añadir todo de golpe:

git add -A

ó

git add --all

## Cambios en la pantalla

git diff muestra los cambios entre commits, commit y árbol de trabajo.

git diff

Especifique el nombre del archivo para mostrar los cambios en curso de su archivo:

git diff somefile.js

Muestra los cambios entre las ramas master y develop:

git diff master..develop

## Cambio de nombre de los archivos

Puedes cambiar el nombre de un archivo o carpeta con el comandomv. Debes especificar una ruta de origen y otra de destino. El origen es un archivo o carpeta real, y el destino es una carpeta existente.

git mv directory1/somefile.js directory

## Tomar una captura de los cambios

Grabar los cambios en el repositorio. Este comando se utiliza para guardar los cambios en el repositorio local. Se puede utilizar con algunas claves de git, como:

* - m para añadir un mensaje a su confirmación
* - a para poner en escena todos los archivos en su commit
* --amend para reescribir la última confirmación con los cambios que se estén realizando o con un nuevo mensaje de confirmación

git commit -m "Commit message"

git commit --amend

git commit --amend -m "New message"

Bueno aquí empieza una de las partes más importante de tener un repositorio git y es poder commitear el estado actual de nuestro repositorio. Para hacerlo están sencillo como ejecutar:

git commit -m "esto es un mensaje"

Esto lo que hará es hacer una foto del estado actual de nuestro repositorio. El mensaje es muy importante y tiene que ser lo más descriptivo posible.

Ahora si de nuevo ejecutamos:

git status

## Comprobar el historial de git

Mostrar los registros de confirmación. También como usuario de git puedes usar el comando git log de forma más avanzada, simplemente añadiendo algunas claves a tu comando git log

git log

Oneline flag, muestra cada confirmación en una sola línea:

git log --oneline

Shortlog agrupa cada confirmación por autor y muestra la primera línea de cada mensaje de confirmación:

git shortlog

La opción --graph dibuja un gráfico ASCII que representa la estructura de ramas del historial de confirmaciones. Se suele utilizar junto con los comando –oneline y –decorate para que sea más fácil ver qué confirmación pertenece a qué rama:

git log --graph --oneline --decorate

`  
También puedes limitar el número de commits a la salida del registro:

git log -5

Soportar el filtrado del historial de git, por ejemplo por fecha, por autor, por archivo, por mensaje:

git log --after="yesterday" --before="2022-10-10"

git log --author="Michael"

git log -- somefile.js

git log -S "fix"

Este comando de git es muy interesante si queremos ver el historia de nuestros commits. Si ejecutamos:

git log

**git reset –soft**

Primero vamos a ver el estado actual de nuestro repositorio con:

git log

Imaginemos que el commit f41fb613902f9b7cc47f1fdb23de392d6238a57a queremos deshacerlo, pues ejecutamos:

git reset --soft 48cb8d5c37c0298b0ddb9a037996d8df27ac931f

veras que ahora solo tenemos un commit y si ejecutamos:

git status

tenemos todos los cambios que se hicieron en el commit que hemos desecho.

**git reset –mixed**

La diferencia entre git reset soft y git reset mixed es que mixed saca los ficheros nuevos, que tuviese ese commit, fuera de nuestro repositorio, es como si deshiciese el git add.

Podemos comprobar que ha deshecho el commit perfectamente pero, si verfificamos el status:

git status

esta totalmente limpio. Eso significa que git ha deshecho el commit pero, ha borrado todos los cambios que tenia el commit.

## Función de ramificación

Una rama representa una línea de desarrollo independiente. Las ramas sirven como una abstracción para el proceso de edición/etapa/compromiso. El comando git branch te permite crear, listar, renombrar y eliminar ramas.  
Para crear una nueva rama:

git branch branch\_name

También puedes pasar algunas claves al comando git branch:

* git branch -m <branch> para renombrar la rama actual
* git branch -d <branch> para eliminar la rama localmente
* git push origin --delete <branch> para empujar los cambios a remoto informando de la eliminación de la rama al repositorio de origen remoto (requiere su uso con el comando anterior)
* git branch -a para mostrar la lista de todas las ramas

Este comando creará una rama **localmente**. Para insertar la nueva rama en el repositorio remoto, debes usar el siguiente comando:

git push -u <remote> <branch-name>

## Deshacer cambios en los archivos

El comando git restore ayuda a deshacer o incluso descartar los cambios locales no comprometidos. El comando puede ser utilizado para deshacer los efectos de git add y deshacer los cambios que ha añadido previamente, también puede ser utilizado para descartar los cambios locales en un archivo, restaurando así su último estado comprometido.

git restore somefile.js

git restore --staged index.js

## Trabajar con comandos remotos

git remote gestionar un conjunto de repositorios remotos rastreados.  
Para mostrar la lista de todas las conexiones remotas:

git remote -v

Para cambiar la url remota:

git remote set-url <url> <new\_url>

Para renombrar las conexiones actuales, se puede utilizar el siguiente comando:

git remote rename <old\_name> <new\_name>

Para eliminar la conexión:

git remote remove <remote\_name>

## Guardar los cambios en el portapapeles

git stash Guarda los cambios en un directorio de trabajo sucio. Este comando toma los cambios no comprometidos (tanto staged y unstaged) y los guarda para su uso posterior.

git stash

Se pueden añadir varias teclas al comando:

* git stash para guardar los archivos rastreados
* git stash -u para almacenar los archivos sin seguimiento
* git stash -a para almacenar todos los archivos (incluidos los ignorados)

El comando Stash guarda tus cambios en una especie de lista de cambios, a la que puedes acceder simplemente usando:

git stash list

Además, puedes añadir mensajes a tu stash, anotándolos con el comando:git stash save "message"

git stash save "some comment"

También admite la visualización de los diferenciales del stash:

git stash show

Para aplicar los cambios guardados en el stash (se aplicará el último stash de la lista de stash):

git stash apply

Y poder limpiar todos los stash:

git stash clear

## Etiquetado

git tag etiqueta puntos específicos en la historia de un repositorio.

git tag v1.1

Para acceder a la lista de etiquetas utilice git tag -l. Para borrar sólo hay que pasar una clave específica git tag -d v1.0. Para listar las etiquetas remotas: git ls-remote --tags. Para reetiquetar Para reetiquetar (cambiar el nombre de la etiqueta existente) sólo tienes que enviar con la tecla de fuerza: git tag -f v1 v1.1 , en este caso renombrando v1 con el nuevo v.1.1.

## Obtén los últimos cambios remotos

Para obtener los últimos cambios en su local hay 2 comandos git: git pull y git fetch. La principal diferencia entre ellos es que git fetch descargará el contenido remoto pero no actualizará el estado de trabajo de su repo local, dejando su trabajo actual intacto. Personalmente utilizo un comando git fetch con una llave --prune, que es la mejor utilidad para limpiar las ramas obsoletas. Antes de la obtención, elimina cualquier referencia de seguimiento remoto que ya no exista en el remoto. git pull en cambio, descargará el contenido remoto de la rama local activa y ejecutará inmediatamente la fusión en sus archivos. También git pull se puede utilizar con la clave común rebase: git pull -r para tirar y volver a basar.

## Deshacer los últimos commits

En git existen dos formas de deshacer los últimos cambios: git revert y git reset. El comando git revert crea una nueva confirmación que deshace los cambios de una confirmación anterior. Este comando añade un nuevo historial al proyecto. git reset se utiliza para deshacer los cambios en tu directorio de trabajo que aún no han sido cometidos. El comando Reset se puede utilizar con argumentos --soft, --mixed, --hard. Por defecto git utiliza reset con --mixedkey (anular el compromiso + anular los cambios). Los desarrolladores utilizan con frecuencia la opción --hard (anular el compromiso + anular la etapa + eliminar los cambios). Cuando se le pasa --hard los punteros del historial de confirmaciones se actualizan a la confirmación especificada. Y --soft es una forma más precisa si se desea descomprometer los cambios, en este caso los cambios se dejan escalonados.

## Cambio entre commits o ramas

El comando git checkout es usado. Puedes cambiar entre commits y ramas, simplemente pasando branch\_name/commit\_sha a el comando git checkout. También puedes crear una nueva rama no existente utilizando el comando checkout, se creará una nueva rama y se cambiará a ella:

git checkout -b new\_branch

## 1. Git clone

Git clone es un comando para descargar el código fuente existente desde un repositorio remoto (como Github, por ejemplo). En otras palabras, Git clone básicamente hace una copia idéntica de la última versión de un proyecto en un repositorio y la guarda en su computadora.

Hay un par de formas de descargar el código fuente, pero sobre todo prefiero él clone con https:

git clone <https://name-of-the-repository-link>