* **git init:** lo usamos para determinar la carpeta en la que vamos a trabajar.
* **git status:** lo usamos para saber si tenemos un archivo añadido o borrado en nuestro proyecto, para saber en la rama en la que estamos y si tenemos commits.
* **git add:** es para añadir un archivo a nuestra rama seguidamente ponemos entre comillas el nombre de nuestro archivo o poner un punto para añadir todos los archios de nuestra carpeta.
* **git rm:** lo usamos para borrar un archivo que hayamos añadido, para eliminarlo por completo de nuestra rama usamos **git rm --cached**.
* **git commit:** se usa para añadir un commit a nuestra rama, también podemos ponerle un **-m** seguidamente ponemos entre comillas nuestro ensaje.
* **git config:** muestra configuraciones de git también podemos usar **–list** para mostrar la configuración por defecto de nuestro git y si añadimos --show-origin inhales nos muestra las configuraciones guardadas y su ubicación.
* **git config --global**[**user.name**](http://user.name/)**:** cambia de manera global el nombre del usuario, seguidamente ponemos entre comillas nuestro nombre.
* **git config --global user.email:** cambia de manera global el email del usuario, seguidamente ponemos entre comillas nuestro nombre.
* **git log:** se usa para ver la historia de nuestros archivos, los commits, el usuario que lo cambió, cuando se realizaron los cambios etc. seguidamente ponemos el nombre de nuestro archivo.
* **git show:** muestra el historial de cambios
* **git diff # del commit : muestras la diferencias entre las versiones**
* **git reset** : regresa a una versión anterior, tiene dos parametros - -hard (Regresa todo el archivo) y - -soft
* **git log --**stat: muestra detalladamente las versiones, indicando los bits que fueron cambiados
* **git checkout #Version nombrearchivo:** regresa el archivo a la versión seleccionada
* **git checkout master nombrearchivo:** regresa el archivo a la versión master

**git rm**

Este comando nos ayuda a eliminar archivos de Git sin eliminar su historial del sistema de versiones. Esto quiere decir que si necesitamos recuperar el archivo solo debemos “viajar en el tiempo” y recuperar el último commit antes de borrar el archivo en cuestión.

Recuerda que git rm no puede usarse así nomás. Debemos usar uno de los flags para indicarle a Git cómo eliminar los archivos que ya no necesitamos en la última versión del proyecto:

* git rm --cached: Elimina los archivos del área de Staging y del próximo commit pero los mantiene en nuestro disco duro.
* git rm --force: Elimina los archivos de Git y del disco duro. Git siempre guarda todo, por lo que podemos acceder al registro de la existencia de los archivos, de modo que podremos recuperarlos si es necesario (pero debemos usar comandos más avanzados).

**git reset**

Este comando nos ayuda a volver en el tiempo. Pero no como git checkout que nos deja ir, mirar, pasear y volver. Con git reset volvemos al pasado sin la posibilidad de volver al futuro. Borramos la historia y la debemos sobreescribir. No hay vuelta atrás.

Este comando es **muy peligroso** y debemos usarlo solo en caso de emergencia. Recuerda que debemos usar alguna de estas dos opciones:

Hay dos formas de usar git reset: con el argumento --hard, borrando toda la información que tengamos en el área de staging (y perdiendo todo para siempre). O, un poco más seguro, con el argumento --soft, que mantiene allí los archivos del área de staging para que podamos aplicar nuestros últimos cambios pero desde un commit anterior.

* git reset --soft: Borramos todo el historial y los registros de Git pero guardamos los cambios que tengamos en Staging, así podemos aplicar las últimas actualizaciones a un nuevo commit.
* git reset --hard: Borra todo. Todo todito, absolutamente todo. Toda la información de los commits y del área de staging se borra del historial.

**¡Pero todavía falta algo!**

* git reset HEAD: Este es el comando para sacar archivos del área de Staging. No para borrarlos ni nada de eso, solo para que los últimos cambios de estos archivos no se envíen al último commit, a menos que cambiemos de opinión y los incluyamos de nuevo en staging con git add, por supuesto.

**¿Por qué esto es importante?**

Imagina el siguiente caso:

Hacemos cambios en los archivos de un proyecto para una nueva actualización. Todos los archivos con cambios se mueven al área de staging con el comando git add. Pero te das cuenta de que uno de esos archivos no está listo todavía. Actualizaste el archivo pero ese cambio no debe ir en el próximo commit por ahora.

¿Qué podemos hacer?

Bueno, todos los cambios están en el área de Staging, incluido el archivo con los cambios que no están listos. Esto significa que debemos sacar ese archivo de Staging para poder hacer commit de todos los demás.

¡Al usar git rm lo que haremos será eliminar este archivo completamente de git! Todavía tendremos el historial de cambios de este archivo, con la eliminación del archivo como su última actualización. Recuerda que en este caso no buscábamos eliminar un archivo, solo dejarlo como estaba y actualizarlo después, no en este commit.

En cambio, si usamos git reset HEAD, lo único que haremos será mover estos cambios de Staging a Unstaged. Seguiremos teniendo los últimos cambios del archivo, el repositorio mantendrá el archivo (no con sus últimos cambios pero sí con los últimos en los que hicimos commit) y no habremos perdido nada.

**Flujo de trabajo básico con un repositorio remoto**

**No veas esta clase** a menos que hayas practicado todos los comandos de las clases anteriores.

Por ahora, nuestro proyecto vive únicamente en nuestra computadora. Esto significa que no hay forma de que otros miembros del equipo trabajen en él.

Para solucionar esto están los **servidores remotos**: un nuevo estado que deben seguir nuestros archivos para conectarse y trabajar con equipos de cualquier parte del mundo.

Estos servidores remotos pueden estar alojados en GitHub, GitLab, BitBucket, entre otros. Lo que van a hacer es guardar el mismo repositorio que tienes en tu computadora y darnos una URL con la que todos podremos acceder a los archivos del proyecto para descargarlos, hacer cambios y volverlos a enviar al servidor remoto para que otras personas vean los cambios, comparen sus versiones y creen nuevas propuestas para el proyecto.

Esto significa que debes aprender algunos nuevos comandos:

* **git clone url\_del\_servidor\_remoto**: Nos permite descargar los archivos de la última versión de la rama principal y todo el historial de cambios en la carpeta .git.
* **git push**: Luego de hacer git add y git commit debemos ejecutar este comando para mandar los cambios al servidor remoto.
* **git fetch**: Lo usamos para traer actualizaciones del servidor remoto y guardarlas en nuestro repositorio local (en caso de que hayan, por supuesto).
* **git merge**: También usamos el comando git fetch con servidores remotos. Lo necesitamos para combinar los últimos cambios del servidor remoto y nuestro directorio de trabajo.
* **git pull**: Básicamente, git fetch y git merge al mismo tiempo.

# Introducción a las ramas o branches de Git

Las ramas son la forma de hacer cambios en nuestro proyecto sin afectar el flujo de trabajo de la rama principal. Esto porque queremos trabajar una parte muy específica de la aplicación o simplemente experimentar.

La cabecera o HEAD representan la rama y el commit de esa rama donde estamos trabajando. Por defecto, esta cabecera aparecerá en el último commit de nuestra rama principal. Pero podemos cambiarlo al crear una rama (git branch rama, git checkout -b rama) o movernos en el tiempo a cualquier otro commit de cualquier otra rama con los comandos (git reset id-commit, git checkout rama-o-id-commit).