
git

¿Qué es git?

Git es un sistema de control de versiones. ¿Qué es el "control de versiones"? Es un sistema que registra los cambios en un archivo o conjunto de archivos a lo largo del tiempo para que se puedan recuperar versiones específicas más adelante.

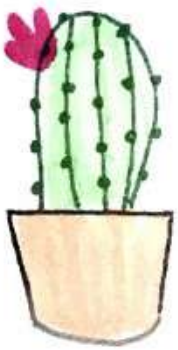
Git colabora con esto

⚠ GitHub es una plataforma para alojar proyectos utilizando el sistema de control de versiones git.

COMANDOS BÁSICOS

El primer paso es inicializar un nuevo repositorio de git. Se puede hacer con el comando `git init`.

Este comando creará un área conocida como "Staging" (donde se guardará temporalmente nuestros archivos) y un repositorio local (base de datos histórico del proyecto)



Cuando trabajamos con git, nuestros archivos pasan por diferentes **estados**.

→ **git status**: nos muestra en qué estado se encuentran nuestros archivos.

estado untracked — archivo sin rastrear (aún no vive en git)

estado unstaged — archivo que vive en git pero fue afectado por el comando add

git add

estado staged — el archivo aún no es guardado definitivamente.

git commit

estado tracked — archivo rastreado. al ser afectado por commit es enviado al repositorio. Los cambios son definitivos.

rm

reset HEAD

git status: ver el estado de nuestros archivos.

git add: añade el archivo a Staging.

git add nombre_archivo

git add . → añade todos los archivos.

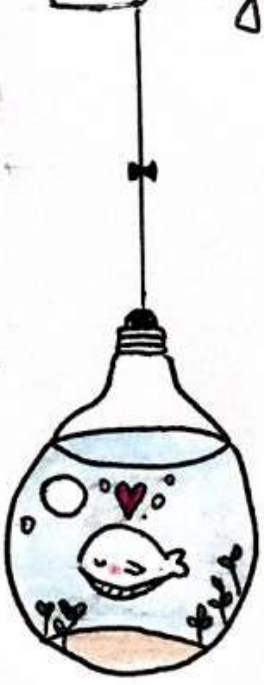
git commit: guarda los archivos en el repositorio local.

git commit -m "mensaje"

git reset HEAD: quita archivos de staged o los devuelve a su estado anterior.

git rm: elimina el archivo de tu directorio de trabajo y no elimina su historial del sistema de versiones.

- **git rm --cached**: mueve los archivos que indiquemos al estado untracked.
- **git rm --force**: elimina los archivos de git y del disco duro.



Ramas o branches

Todos los commits se aplican sobre una rama.

Por defecto, comenzamos en la rama MASTER.

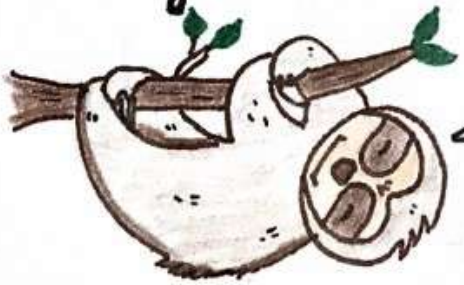
Crear una rama consiste en copiar un commit de una rama determinada pasarlo a otra rama y continuar el trabajo sin afectar el flujo de trabajo principal (rama master)

Crear una nueva rama

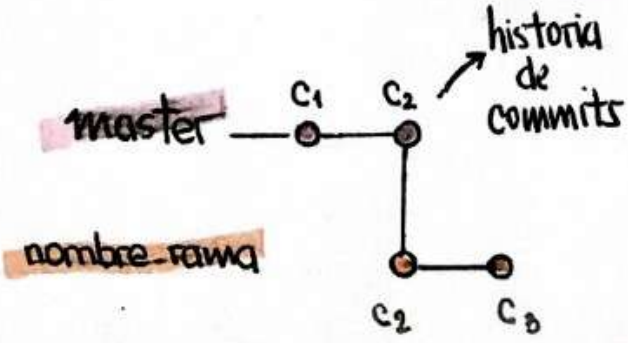
La rama se va a crear desde el lugar que estoy ubicado:

❤ **git status** : me muestra en que rama estoy, en qué versión del commit y si hay algún commit que enviar.

✨ **git branch** nombre_rama → copia el último commit a esta rama.



con **git show** vemos que el último commit está pegado a dos ramas!

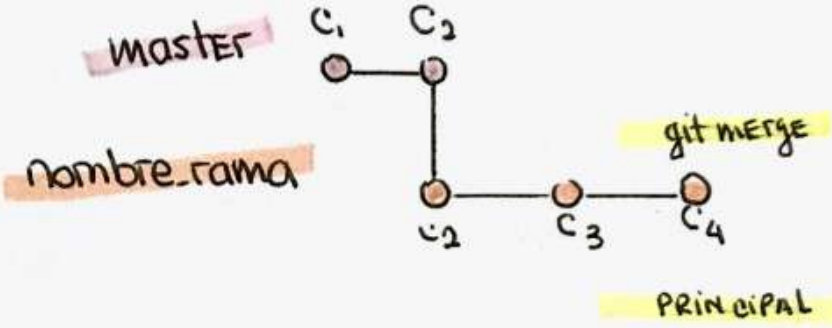


Para migrar de una rama a otra, usamos: **git checkout** nombre_rama.

⚠ No olvidar hacer 'git commit -am' para no perder los cambios que hicimos, previo a migrar.

fusión de ramas

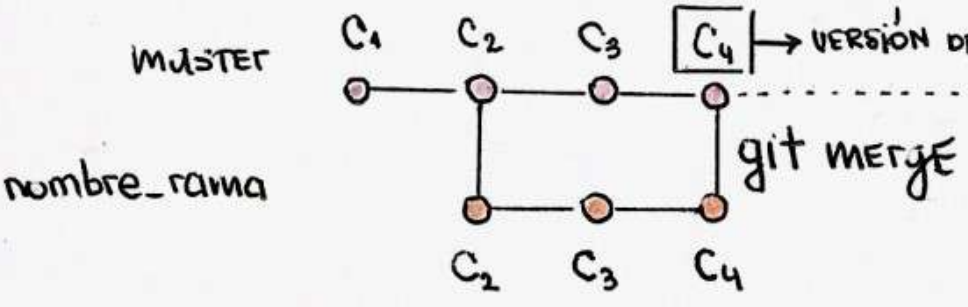
El merge (git merge) va a ocurrir donde estoy ubicado generando que el último commit de esa rama sea el principal.



Si queremos agregar a master un cambio que se hizo en otra rama, nos ubicamos en ella con **git checkout master**. Luego, **git merge** nombre_rama. (se abre un entorno para colocar un mensaje).

¡Listo, se autofusiona todo!

si no sabemos el nombre de la rama, usando **git branch** podemos ver las ramas existentes y en cuál estamos.



MÁS COMANDOS

☆ **git show**: muestra los cambios que han existido sobre un archivo.

☆ **git diff**: muestra la diferencia entre una versión y otra.

E.j: git diff commit1 commit2
Si usamos git diff --staged vemos los cambios entre versiones por etapas.

☆ **git log**: obtenemos el ID de nuestros commits. Muestra el historial de confirmaciones del repo local.
Si usamos git log -p, muestra el historial de confirmación incluyendo todos los archivos y sus cambios.

☆ **git checkout -b nombre_rama**: podemos crear una rama y cambiamos a ella, con un solo comando.

☆ **git checkout IDcommit**: podemos volver a cualquier versión anterior de un archivo. También nos permite movernos entre ramas.

☆ **git reset**: también volvemos a versiones anteriores pero además borramos todos los cambios que hicimos después de ese commit. ⚠

Hay dos formas de usar git reset:

- **git reset --hard**: Reestablece el árbol de trabajo y el índice. Cualquier cambio en el árbol desde el commit se descartan

→ --hard reestablece tu commit, zona de 'staging' y tu directorio de trabajo.

- **git reset --soft**: no reinicia el archivo índice o el árbol de trabajo, sino que reinicia el HEAD para commit. Cambia todos los archivos a "Cambios a ser commited"

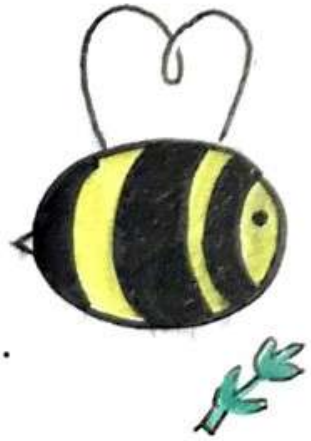
HEAD: versión del commit que estoy viendo.

☆ **git branch -d nombre-rama**: cuando haya terminado de trabajar con una rama y la hayamos fusionado, podemos eliminarla usando este comando.

☆ **git add remote URL**: agrega un repositorio remoto al repositorio local.

Flujo de trabajo

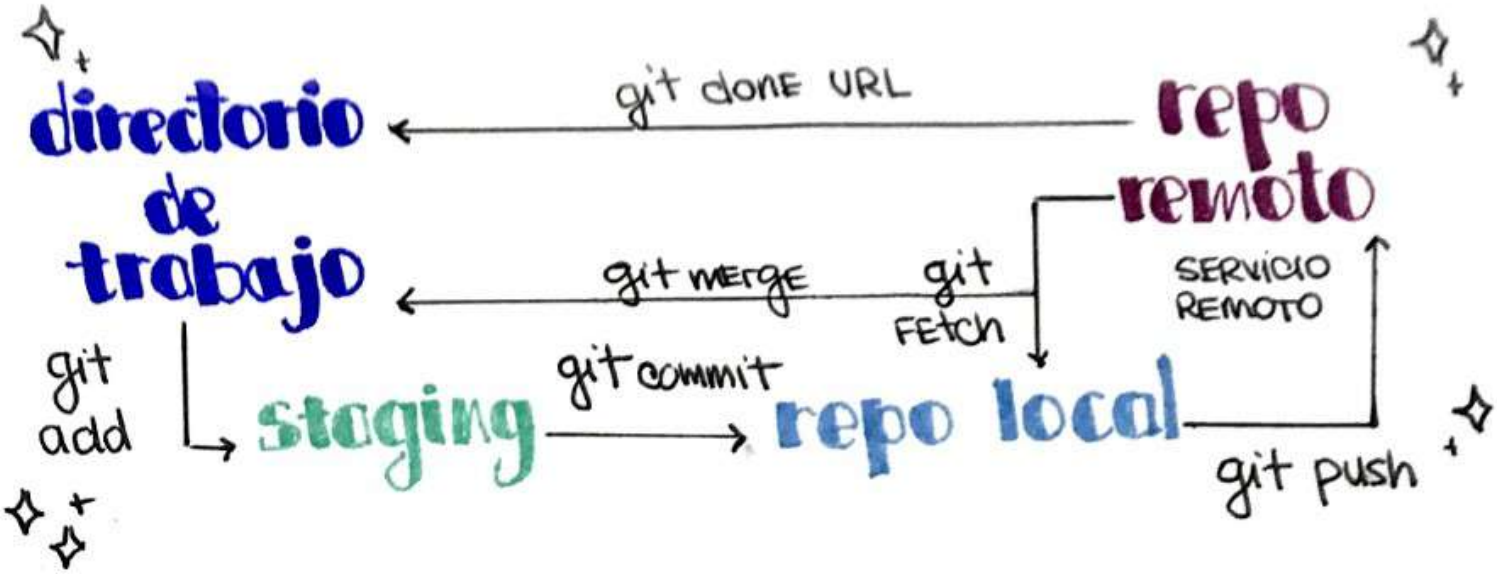
- Creamos un directorio de trabajo: donde están nuestros archivos.
- git init para crear nuestro staging (área de preparación) y repositorio local.
- agregamos archivos al área de staging con git add nombre_archivo o git add .
- Enviamos nuestro archivo al repositorio local con git commit -m



Nuestro proyecto puede estar alojado en un servicio remoto, como GitHub, GitLab, BitBucket, entre otros. Estos guardan nuestro proyecto y nos brindan una URL para poder acceder a él.

Para acceder a nuestro proyecto remoto, utilizamos el comando `git clone URL` (reemplazando 'URL' por nuestro link)

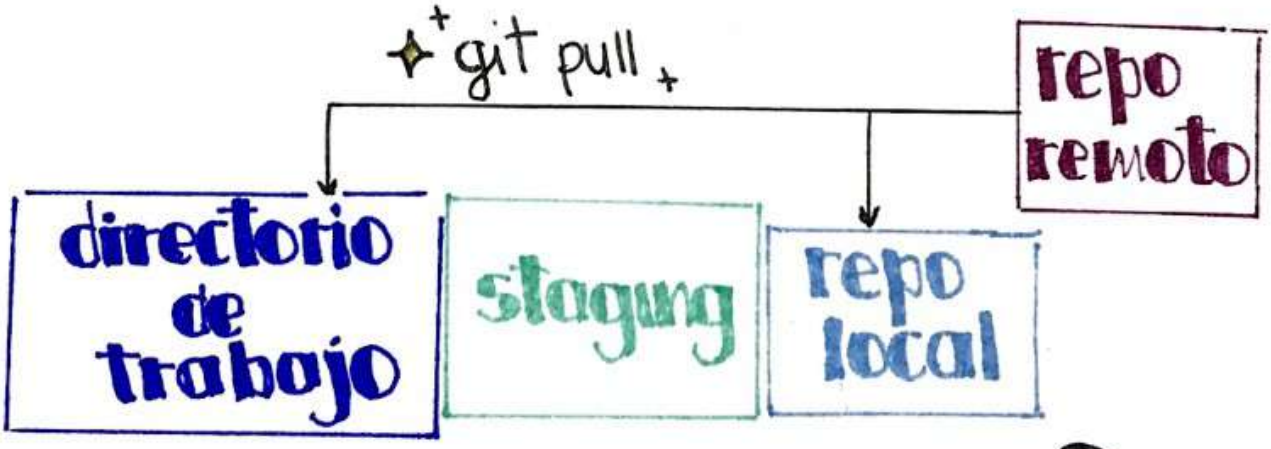
- Luego de hacer git add y git commit, debemos ejecutar `git push` para mandar los cambios al servidor remoto.



- utilizamos git FETCH para traer actualizaciones del SERVIDOR REMOTO y guardarlas en NUESTRO REPO local.
- También usamos git MERGE cuando NECESITAMOS combinar los últimos cambios del SERVIDOR REMOTO y EL DIRECTORIO DE trabajo.

↓

- git pull: BASICAMENTE ES git FETCH y git MERGE al mismo TIEMPO.



Más comandos

- git stash: guarda el trabajo actual de staging en una lista temporal (stash) para poder recuperarlo en un futuro.
  Para recuperar los últimos cambios desde el stash al staging: git stash pop.
  Para ver la lista de cambios guardados en stash: git stash list.
  Para eliminar cambios más recientes dentro del stash: git stash drop.
- git clean: limpia el proyecto de archivos no deseados.
  archivos listados (que no son carpetas): git clear -f
-  git cherry-pick: nos permite tomar uno o varios commits de otra rama sin tener que hacer un merge completo.
  ⚠ usá cherry-pick con sabiduría.
-  git commit -amend: se utiliza para agregar archivos nuevos o actualizar el commit anterior y no generar commits innecesarios.
  ⚠ ¡es mala práctica!