La **fusión en Git** es la forma en que este sistema une un historial bifurcado. El comando git merge permite integrar líneas de desarrollo independientes generadas por git branch en una sola rama. Con este comando, podemos crear un nuevo *commit* que combina dos ramas o [branches](https://platzi.com/clases/1557-git-github/19947-que-es-un-branch-rama-y-como-funciona-un-merge-en-/%22%20%5Ct%20%22_blank): la rama actual y la rama que se indica después del comando.

Estos comandos de fusión del merge afectan solo a la rama actual y no a la rama de destino. Por lo tanto, te recomendamos utilizar git checkout para seleccionar la rama actual y git branch -d para eliminar la rama de destino obsoleta.

**Funcionamiento de Git merge**

Git merge fusiona secuencias de confirmaciones en un solo historial, generalmente para combinar dos ramas. Busca una confirmación de base común y genera una confirmación de fusión que representa la combinación de las dos ramas hasta el resultado final.



**¿Cómo unir dos ramas en git?**

Ahora bien, para combinar ramas en tu repositorio local, usa git checkout para cambiar a la rama donde deseas fusionar. Por lo general, esta es la rama principal. Luego, emplea git merge y especifica el nombre de la otra rama que deseas traer a esta rama. Ten en cuenta que esto es una combinación de avance rápido.

**¿Cómo realizar un merge en git?**

Para hacer un merge en Git, primero asegúrate de estar en la rama correcta. Después, usa el comando git merge seguido del nombre de la rama que quieres combinar. Por ejemplo, si quieres crear un nuevo commit en la rama master con los cambios de la rama cabecera, usa este comando:

git checkout master

git merge cabecera

Es importante tener en cuenta que en caso de haber conflictos, debes guardar tus cambios antes de hacer git checkout para evitar perder tu trabajo. También es recomendable emplear los comandos básicos de GitHub, como git fetch, git push y git pull, para mantener actualizado tu repositorio.

En este ejemplo, vamos a crear un nuevo *commit* en la rama *master* combinando los cambios de una rama llamada *cabecera*: Otra opción es crear un nuevo *commit* en la rama *cabecera* combinando los cambios de cualquier otra rama:

Git es asombroso porque puede saber cuáles cambios deben conservarse en una rama y cuáles no. En casos de conflictos, asegúrate de guardar tus cambios antes de hacer git checkout para evitar perder tu trabajo.

**Comandos básicos de GitHub**

* git init: crear un repositorio.
* git add: agregar un archivo a staging.
* git commit -m “mensaje”: guardar el archivo en git con un mensaje.
* git branch: crear una nueva rama.
* git checkout: moverse entre ramas.
* git push: mandar cambios a un servidor remoto.
* git fetch: traer actualizaciones del servidor remoto y guardarlas en nuestro repositorio local.
* git merge: tiene dos usos. Uno es la fusión de ramas, funcionando como un *commit* en la rama actual, trayendo la rama indicada. Su otro uso es guardar los cambios de un servidor remoto en nuestro directorio.
* git pull: fetch y merge al mismo tiempo.
* git checkout “codigo de version” “nombre del archivo”: volver a la última versión de la que se ha hecho *commit*.
* git reset: vuelve al pasado sin posibilidad de volver al futuro, se debe usar con especificaciones.
* git reset --soft: vuelve a la versión en el repositorio, pero guarda los cambios en staging. Así, podemos aplicar actualizaciones a un nuevo *commit*.
* git reset --hard: todo vuelve a su versión anterior
* git reset HEAD: saca los cambios de staging, pero no los borra. Es lo opuesto a git add.
* git rm: elimina los archivos, pero no su historial. Si queremos recuperar algo, solo hay que regresar. se utiliza así:git rm --cached elimina los archivos en staging pero los mantiene en el disco duro.git rm --force elimina los archivos de git y del disco duro.
* git status: estado de archivos en el repositorio.
* git log: historia entera del archivo.
* git log --stat: cambios específicos en el archivo a partir de un commit.
* git show: cambios históricos y específicos hechos en un archivo.
* git diff “codigo de version 1” “codigo de version 2”: comparar cambios entre versiones.
* git diff: comparar directorio con *staging*.