**HOLA MUNDO**

**COMANDOS GIT TERMINAL**

**CONCEPTOS**

**GIT:** Sistema de control de versiones más utilizado que trabaja de forma descentralizada donde cada miembro de un equipo tiene una copia del proyecto y permite registrar el histórico de trabajo de nuestro código, generar copias de seguridad, y trabajar en equipo de forma rápida y sin errores.

**Github** es un servicio de alojamiento en la nube de código fuente basado en el sistema de control de versiones que Git ofrece para manejar repositorios. Añade además otras funcionalidades muy útiles para el desarrollo, como una UI muy amigable, GitHub Actions, Webhooks o Dependabot. Otras alternativas a github son **GitLab o BitBucket**

Ventajas:

* Historial código de cada desarrollador
* Almacenamiento de código a modo de nube
* Trabajar en equipo
* Conocimiento de cuando se hizo un error por medio de los commits

Un **sistema de control de versiones** es una herramienta que se utiliza para mantener un registro de los cambios que se hacen en un proyecto. Permite a los desarrolladores trabajar en un proyecto de manera colaborativa, manteniendo un historial de cambios y documentando cada uno de ellos. De esta forma, se puede seguir el progreso del proyecto y volver a versiones anteriores si es necesario, entre muchas otras funcionalidades.

**¿Qué es una rama?**

Una rama es simplemente una versión de la colección de directorios y archivos del repositorio. Cada vez que se crea una nueva rama, se crea una copia de la colección de archivos actual.

Una **rama** es un nuevo camino asociado al código que se está desarrollando en un proyecto. Las ramas permiten a los desarrolladores trabajar en diferentes partes del proyecto al mismo tiempo, sin interferir en el trabajo de los demás. Una vez que completamos una tarea en una rama, podemos fusionarla con otra rama del proyecto y combinar sus cambios.

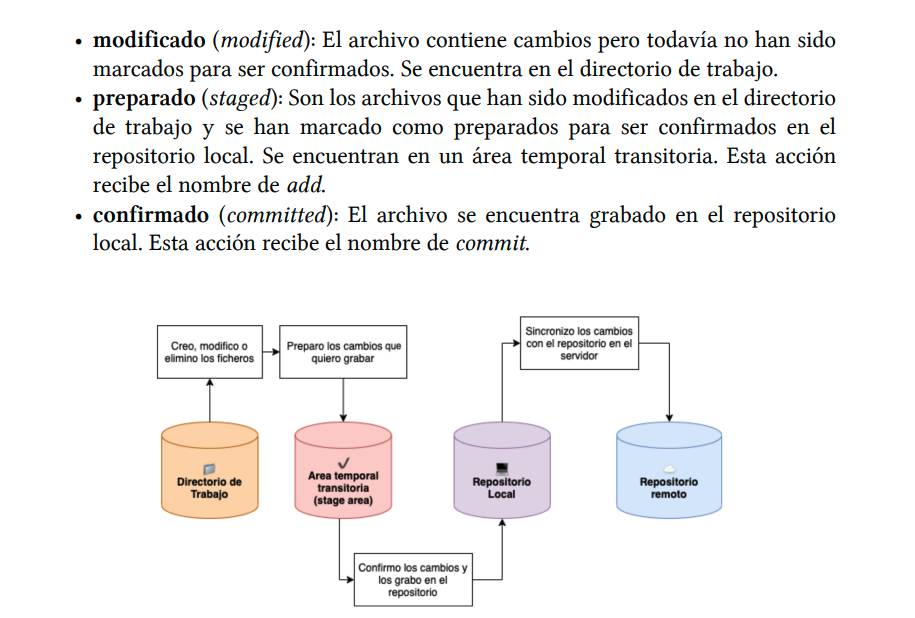
**¿Para qué sirven las ramas?**

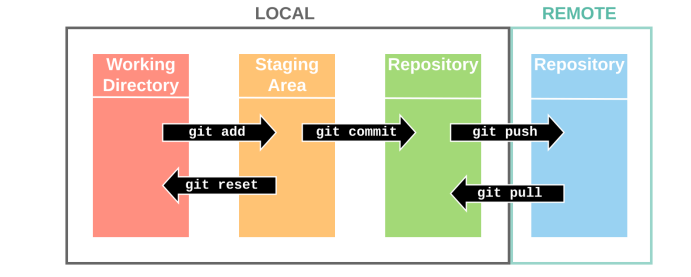
En un entorno de colaboración, donde diferentes personas están trabajando en un mismo código, se genera una evolución del código en paralelo. Mientras alguien está trabajando en añadir una nueva característica al proyecto, otra persona puede estar arreglando un bug y otra en añadir alguna documentación. De esta forma, partiendo de un mismo código, se generan diferentes ramas. Esto sirve para aislar el trabajo de cada persona y que, una vez concluido, se pueda integrar en el tronco de nuestro repositorio que será, dicho de otro modo, la rama principal.

**Rama master o main es la rama principal.**

HEAD es el puntero que referencia el punto actual del historial de cambios del repositorio en el que estás trabajando. Normalmente será el último commit de la rama en la que te encuentres pero como también puedes moverte entre commits… es posible que HEAD no sea el último commit.

**FLUJO DE TRABAJO CON GIT**

****

****

**Ordenador :** Directorio de trabajo – Servidor local

* Cuando estamos en el computado podemos empezar a agregar o modificar lo que queramos, pero al área de stage solo subiremos los que nosotros seleccionemos con git add fichero/archivos.

**Stage: "preparado" (staged):** Estado virtual listo a ser subido (hacer commit)..

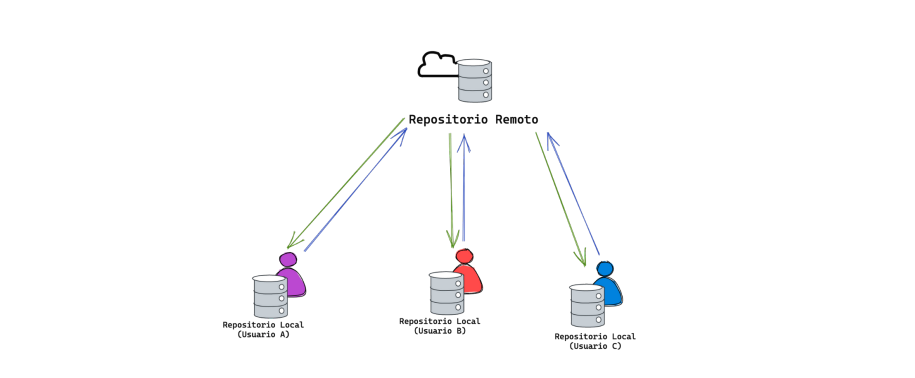
* Una vez revisado lo que tenemos en stage si queremos podemos subir todo o sacar algo de ahí.

**Repositorio local:** Hecho commit se agrega al repositorio local.

* Aquí podemos ya hacer commit y mandar al repositorio local que es lo que subiremos con git push al repositorio remoto.

**Repositorio remoto o servidor en la nube:** Lo que tenemos commited en el repo local se sube al repositorio remoto (Github).

* Hecho el commit, hacemos la subida al github con git push, pueden ser otros servidores en la nube bitbucket, etc
* Si al final una vez subido todo, se elimina algún archivo en nuestro directorio, seguimos el mismo flujo cada vez

****

**COMANDOS DE CONFIGURACIÓN GIT Y EDITOR DE TEXTO VSCODE**

1. **ABRIR TERMINAL Y CONFIGURAR GIT Y COMPROBAR QUE LO TENEMOS INSTALADO**

git –versión

1. **NOS POSICIONAMOS EN CARPETA TRABAJO / CREAMOS DIRECTORIO / FICHERO**

**mkdir + nombreDirectorio:** Crear carpetas o directorios desde terminal

Nos movemos dentro del directorio con CD directorio

1. **CONFIGURACIÓN GLOBAL NOMBRE Y CORREO USADO EN GIT**

**(NO POR PROYECTO)**

git config --global user.name “FerDotCode”

git config --global user.email [f.romero.uni@gmail.com](mailto:f.romero.uni@gmail.com)

Si tienes que cambiar esta configuración para un repositorio en concreto, sólo tienes

que obviar el parámetro --global. Esto te permitirá poder configurar tu nombre y

correo para cada repositorio

.

$ cd <tu directorio de repositorio>

$ git config user.name "midudev"

$ git config user.email "work@midu.dev"

1. **CONFIGURACIÓN EDITOR DE CODIGO POR DEFECTO QUE ABRE - VSCODE**

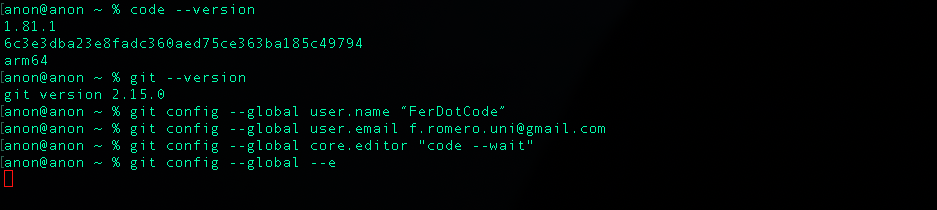
*La opción code --wait es par que nuestro terminal se quede esperando hasta que cerremos el editor de texto.*

git config --global core.editor "code --wait

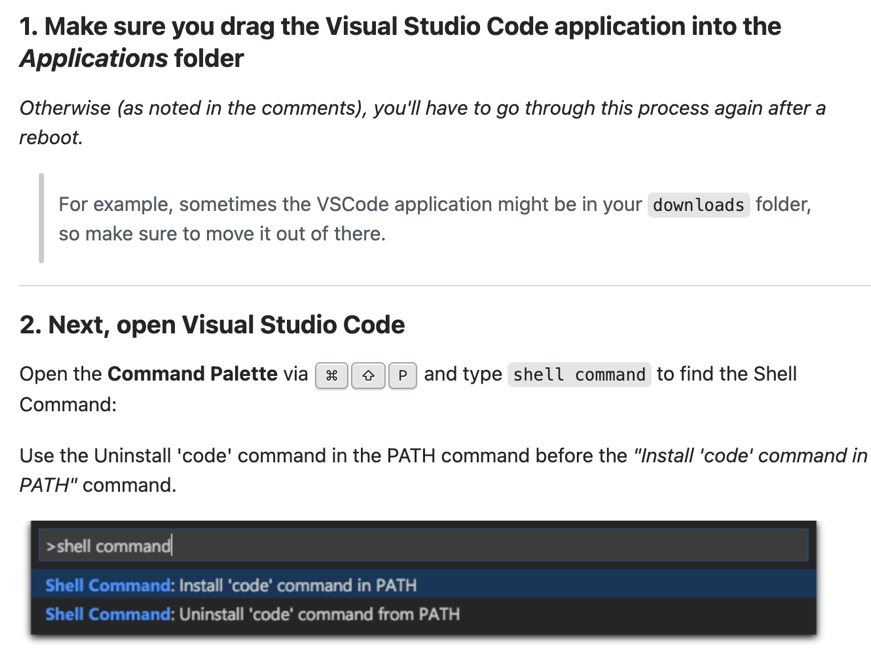
git config --global core.editor "code"

**Ver archivo de configuración global y comprobar que se abre el editor de texto**

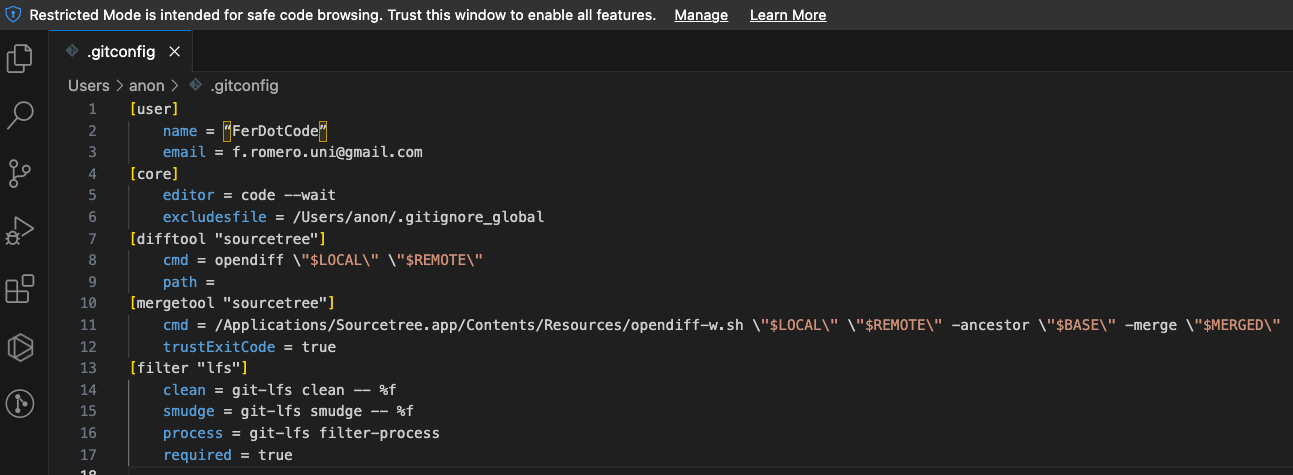
git config --global –e



Puede que vscode no tenga el path instalado para poder enlazar con el terminal

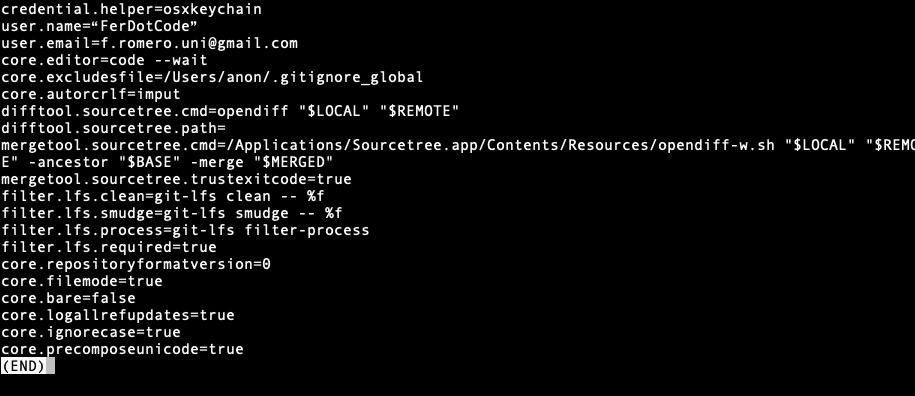


Ahora se abrirá el vscode con la configuración global que tenemos.



Si cerramos el vscode editor volvemos a tener control en la terminal

git config --list

****

**# para mostrar sólo la configuración global de git**

$ git config --global --list

**# para mostrar la configuración del repositorio local (si existe)**

$ git config --local --list

**# muestra la configuración del git del sistema (si existe)**

$ git config --system --list

**Alternativamente, puedes usar el parámetro --show-scope para saber de dónde**

**proviene cada configuración:**

$ git config --list --show-scope

global user.name=midudev

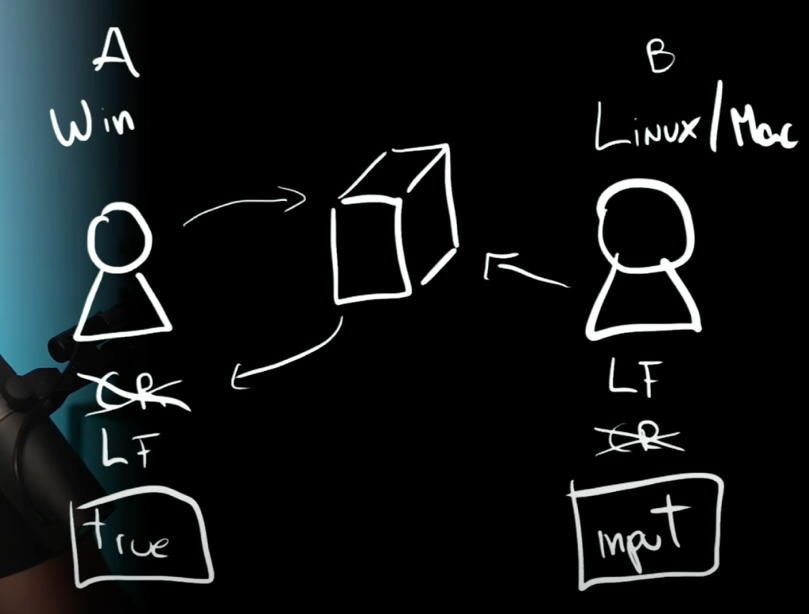
global user.email=miduga@gmail.com

global pull.rebase=false

local core.repositoryformatversion=0

local core.filemode=true

1. **CONFIGURACIÓN DE CORE.AUTOCRLF (SALTOS DE LÍNEA)**

****

**Dejamos configurado como git tratará los saltos de línea**

git config –global core.autorcrlf imput (LINUX O MAX)

**Si queremos ver todas las configuraciónes**

git config -h

1. **ABRIR EDITOR DE CÓDIGO (dentro de la carpeta a trabajar)**

code .

**COMANDOS BÁSICOS DE TERMINAL (en minúsculas)**

**NAVEGACIÓN**

**ls:** Ver listado de carpetas o directorio donde estoy

**ls -a:** Ver todos directorios incluidos ocultos

**pwd:** Ver la ruta carpeta o directorio donde estoy

**cd + ruta directorio:** Moverse entre directorios. Si pulsamos TAB y hemos puesto una inicial nos muestra las opciones y rellena el nombre

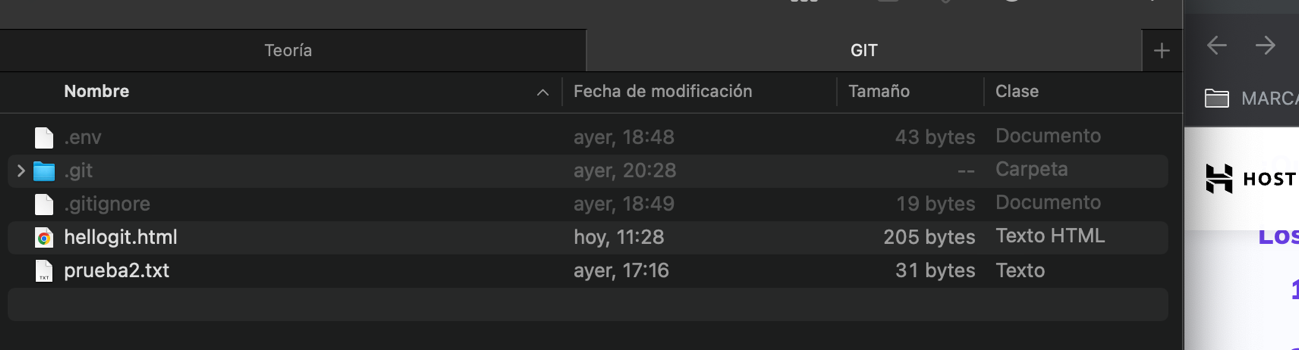
**cd .. :** Moverse hacia atrás un salto de directorio

**CREAR DIRECTORIOS**

**mkdir + nombreDirectorio:** Crear carpetas o directorios desde terminal

Nos movemos dentro del directorio con CD directorio

Al crear el directorio ya vemos como GIT ha creado una carpeta oculta que significa que ese repositorio es el local que trabaja con control del versiones



Esto ya nos posiciona en la rama master o main.





Pensar en el concepto de que cada vez que hacemos un cambio hay que hacer una fotografía del repositorio para guardarlo.

**CREAR FICHERO CUALQUIER EXTENSIÓN**

**touch** fichero.html

1. echo "Esta es la primera frase." > miarchivo.txt (crear)
2. echo "Esta es la segunda frase." >> miarchivo.txt (añadir cosas)
3. echo "Esta es la tercera frase." >> miarchivo.txt

**Estos son comandos comunes de Git usados en varias situaciones:**

comenzar un área de trabajo (mira también: git help tutorial)

clone Clonar un repositorio dentro de un nuevo directorio

init Crear un repositorio de Git vacío o reinicia el que ya existe

trabajar en los cambios actuales (mira también: git help everyday)

add Agregar contenido de archivos al índice

mv Mover o cambiar el nombre a archivos, directorios o enlaces simbólicos

restore Restaurar archivos del árbol de trabajo

rm Borrar archivos del árbol de trabajo y del índice

examinar el historial y el estado (mira también: git help revisions)

bisect Usar la búsqueda binaria para encontrar el commit que introdujo el bug

diff Mostrar los cambios entre commits, commit y árbol de trabajo, etc

grep Imprimir las líneas que concuerden con el patrón

log Mostrar los logs de los commits

show Mostrar varios tipos de objetos

status Mostrar el estado del árbol de trabajo

crecer, marcar y ajustar tu historial común

branch Listar, crear, o borrar ramas

commit Grabar los cambios al repositorio

merge Juntar dos o más historiales de desarrollo juntos

rebase Volver a aplicar commits en la punta de otra rama

reset Reiniciar el HEAD actual a un estado específico

switch Cambiar de branch

tag Crear, listar, borrar o verificar un objeto de tag firmado con GPG

colaborar (mira también: git help workflows)

fetch Descargar objetos y referencias de otro repositorio

pull Realizar un fetch e integra con otro repositorio o rama local

push Actualizar referencias remotas junto con sus objetos asociados

**COMANDOS GIT**

git <verb> --help para ver una descripción de lo que hace ese comando y

una lista de todas las opciones disponibles.

**INICIALIZACION**

1. **INICIALIZACIÓN EN DIRECTORIO**

git init

git crea una carpeta oculta que podemos ver con COMMAND+MAYUS+.

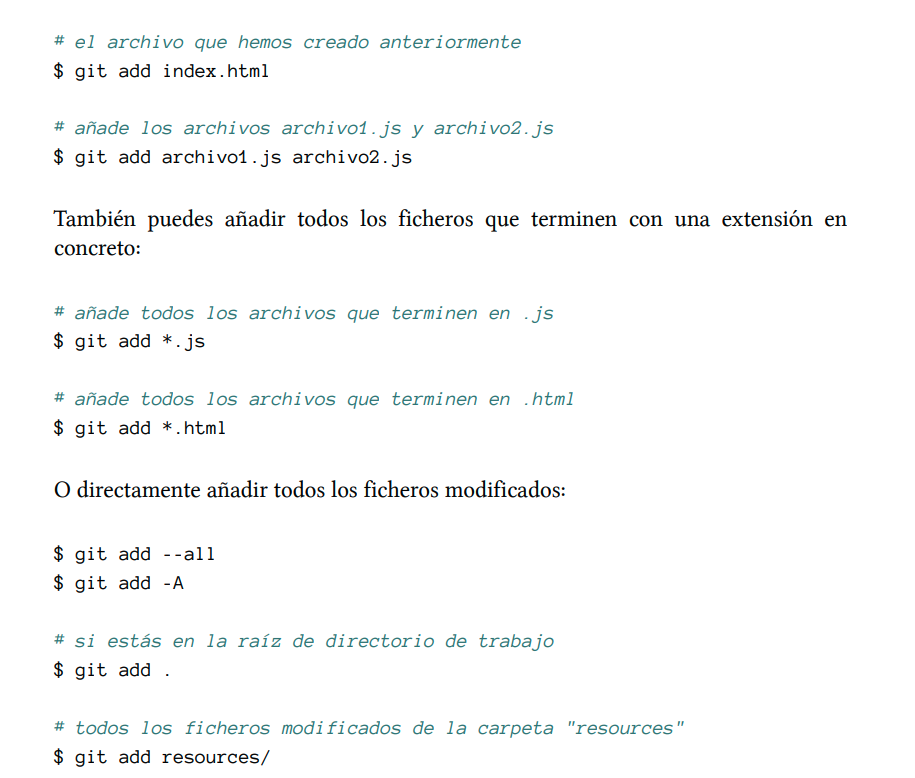
Esto indica que en este directorio hemos creado un enlace para trabajar con un sistema de control de versiones.

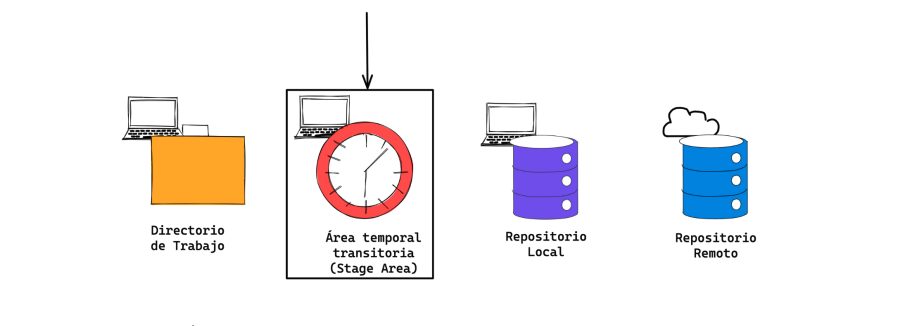
En este momento ya estaremos en MASTER.

**AÑADIR A STAGE**

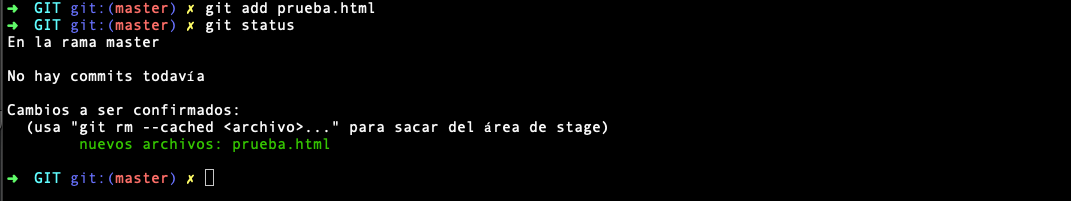
1. **AGREGAR ARCHIVOS AL AREA STAGE (HACER FOTOGRAFÍA)**

git add [nombre.ext] {nombre.ext}.. los que queramos añadir





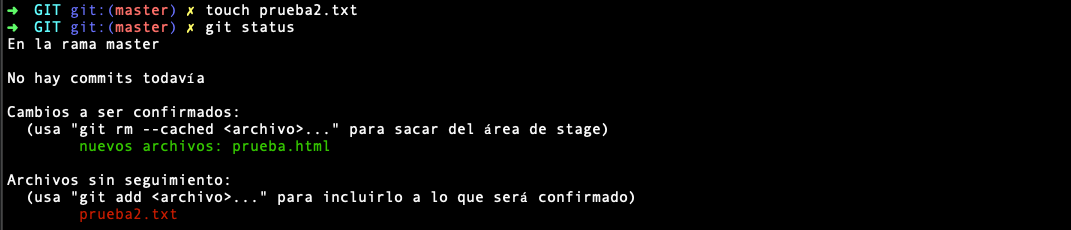
Poner . punto es una mala práctica porque podemos olvidar cosas que no queremos subir, mejor usar el archivos espécifico o los de una extensión en concreto etc.



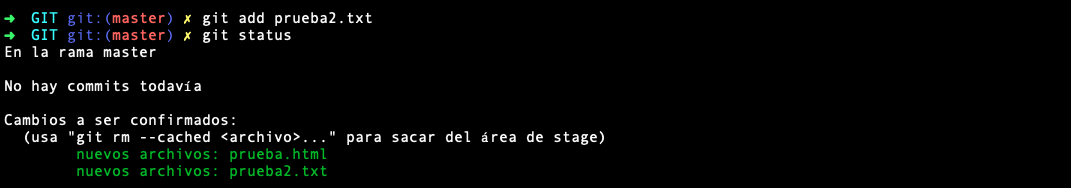
Ya vemos cambios realizados y nos indica que tenemos algo en stage.

Nos indica que estamos en la rama master.

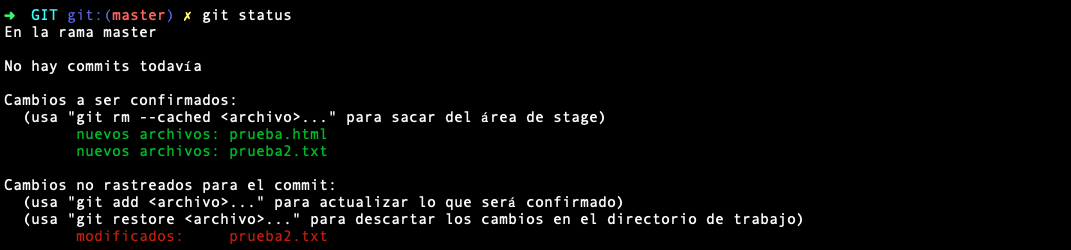
Si creamos algo nuevo en la carpeta vemos como aparece en rojo porque no está en stage.



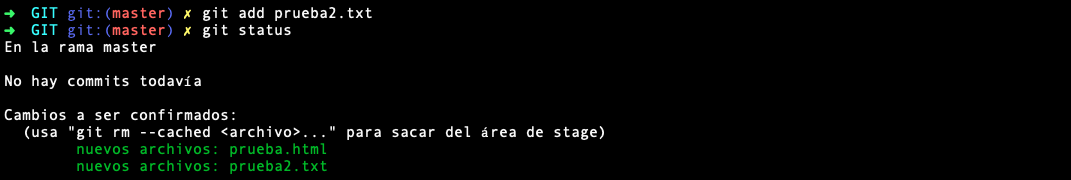
Agrego el segundo archivo a stage y aparecen en verde todos.



Si hiciera un cambio en algún fichero veríamos como me lo indica, que no está el cambio en el área de stage.



Habría que hacer un nuevo git add.



1. **SACAR ARCHIVO DEL AREA STAGING O DESHACER UN ARCHIVO MODIFICADO**

git checkout nombrefichero (deja el fichero como estaba anteriormente)

git reset (anula cambios realizados y vuelve al último git add)

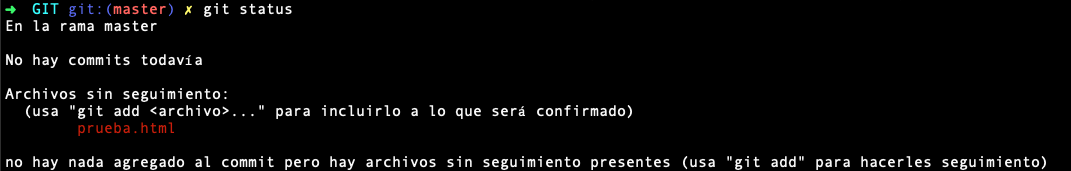
git reset –hard (número de hash) vuelvo al momento concreto

**VER ESTADO**

1. **VER EL ESTADO DE LA CARPETA**

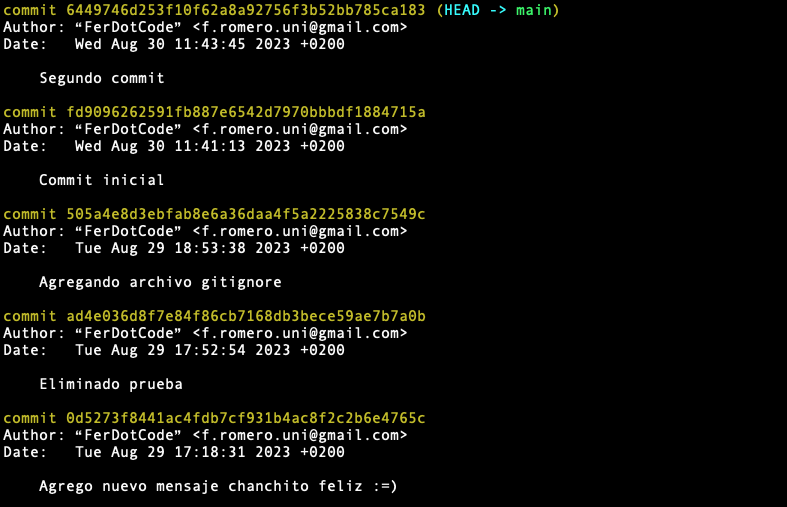
git status

Los que salgan en rojo son los disponibles en el directorio pero no están en stage aún.



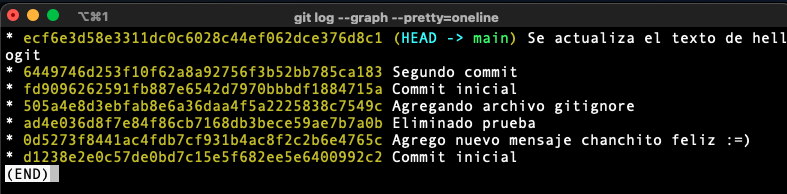
1. **VER EL CAMINO O RUTA DE LOS COMMIT REALIZADOS – VER EL LOG**

git log o git log –graph (ver con ramas)





git log --graph --pretty=oneline



PULSAR TECLA.Q PARA SALIR

gif reflog

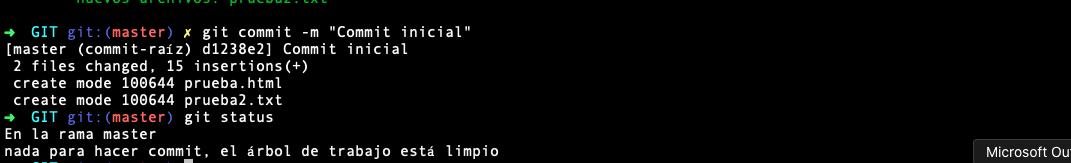
Sirve para ver todo el historial de cambios y modificaciones que hemos hecho

****

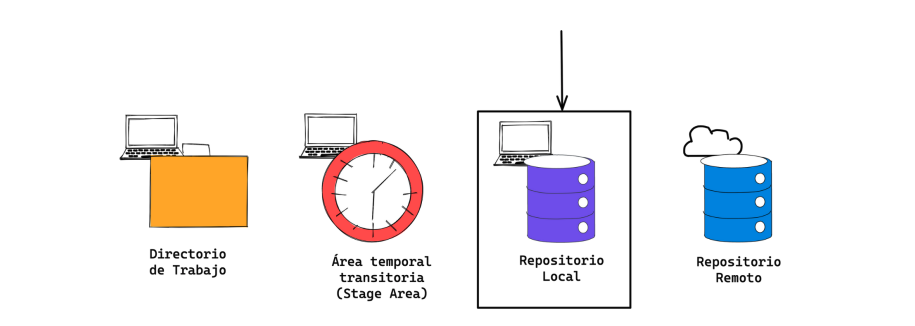
**HACER UN COMMIT Y DEJARLO LISTO PARA SUBIR AL REPO REMOTO**

1. **SUBIR AL REPOSITORIO LOCAL O COMMIT(LANZAR FOTOGRAFÍA A REPO LOCAL – DEJARLA LISTA PARA SUBIR A NUBE)**

git commit -m “mensaje del commit – Commit inicial”



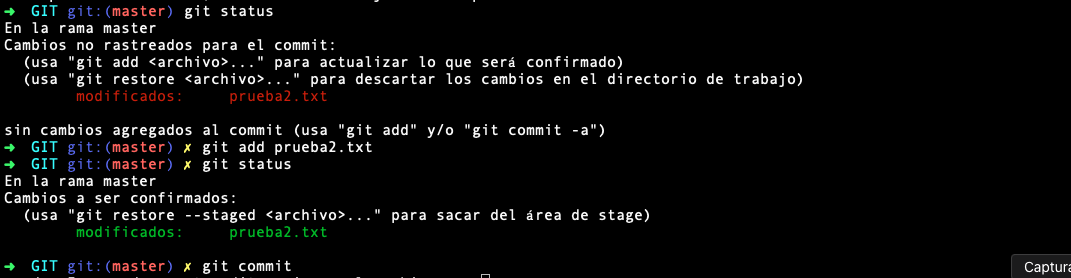
Al hacer commit, y git status vemos que seguimos en la rama master y ya no tenemos nada en el área stage para hacer commit



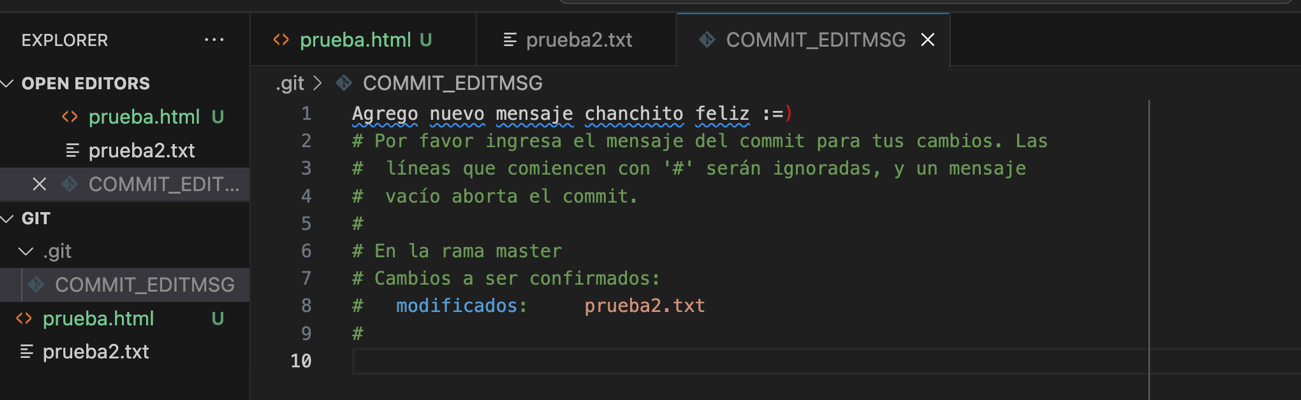
**GRABAR LOS CAMBIOS ANTES DE HACER COMMIT SINO NO APARECEN NADA QUE COMITEAR**

**HACER UN CAMBIO NUEVO EN LOCAL Y HACIENDO COMMIT DESDE VSCODE**

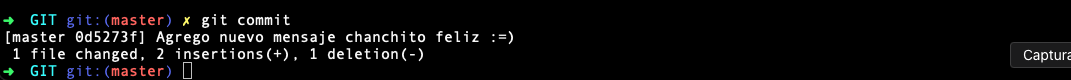
Se ha realizado un cambio, se ha agrado a stage con git add y se ha hecho git commit solamente.

****

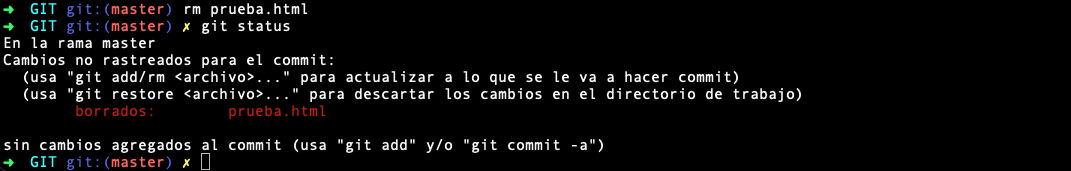
Se abre en el vscode y se agrega el mensaje en el editor



1. Se guarda
2. Se cierra la pestaña y se vuelve al terminal
3. Vemos que se ha agregado el commit con el mensaje

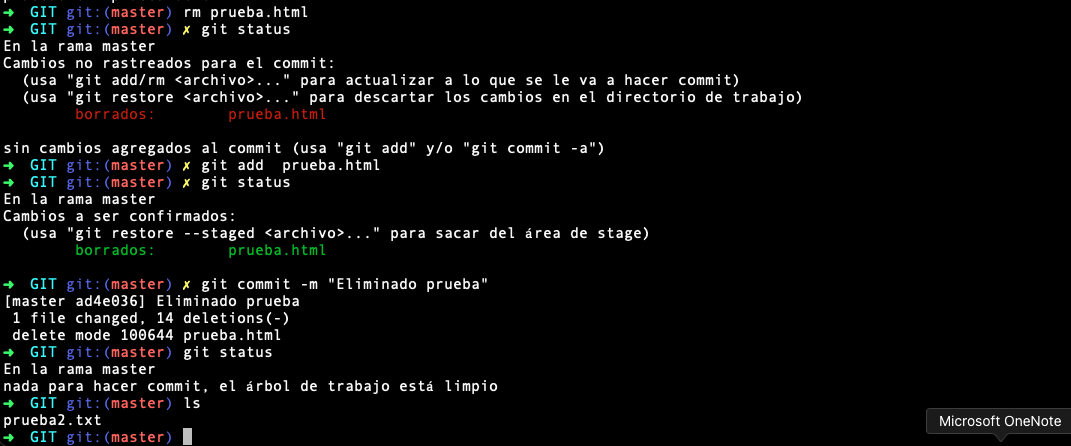


**ELIMINAR ARCHIVOS EN NUESTRO REPOSITORIO**



Si hemos borrado un archivo tenemos que modificarlo también del área del stage para ello, tenemos que volver a hacer un git add.

* Básicamente cada cambio que hagamos en el directorio local tenemos que hacer un git add fichero modificado, para que queda reflejado en el stage.
* Y después un commit para que quede reflejado el cambio también en el repositorio local

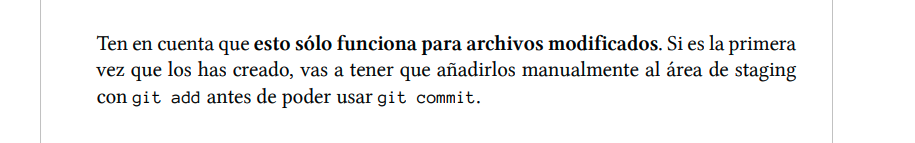
****

Se puede usar git rm nombreArchivo.ext para ahorrarnos el paso de borrar y agregar al stage.

**HACER COMMIT SALTANDONOS EL AREA STAGE (GIT ADD)**

$ git commit -a -m 'Add new files without staging'

$ git commit -am 'Add other files without staging'

****

**DESHACER EL ÚLTIMO COMMIT**

**Si quieres mantener los cambios**

$ git reset --soft HEAD~1

**Si no quieres mantener los cambios**

$ git reset --hard HEAD~1

**MODIFICAR EL ÚTLIMO COMMIT**

**Sólo quieres arreglar el mensaje que has usado para el último commit:**

$ git commit --amend -m "Este es el mensaje correcto"

**Quieres añadir más cambios al último commit:**

$ git add src/archivo-con-cambios.js

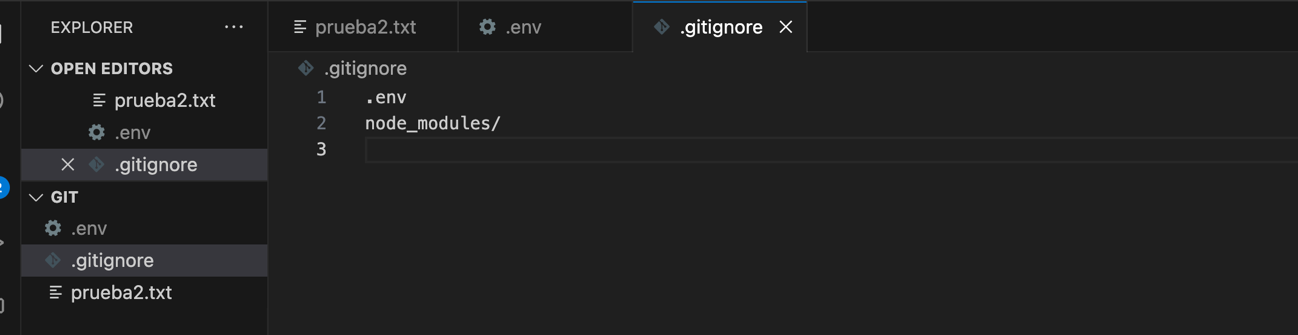
$ git commit --amend -m "Mensaje del commit"

**IGNORAR ARCHIVOS**

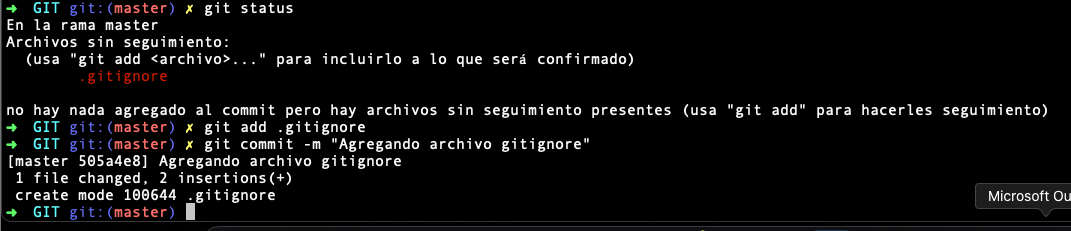
1. **IGNORAR ARCHIVOS PARA NO INCLUIR NUNCA EN GIT (GITIGNORE)**

Por ejemplo tenemos un archivo o varios con x extensión que no queremos que se suban al stage. Los incluimos en el archivo .GITIGNORE

1. Creamos un archivo .gitignore dentro de la carpeta donde estemos
2. En el archivo escribimos las .extensiones o rutas de carpetas que no queremos dentro de vscode tambien con \*\*/nombre archivo
3. Guardamos archivo
4. Si vemos git status vemos que solo tenemos el gitignore

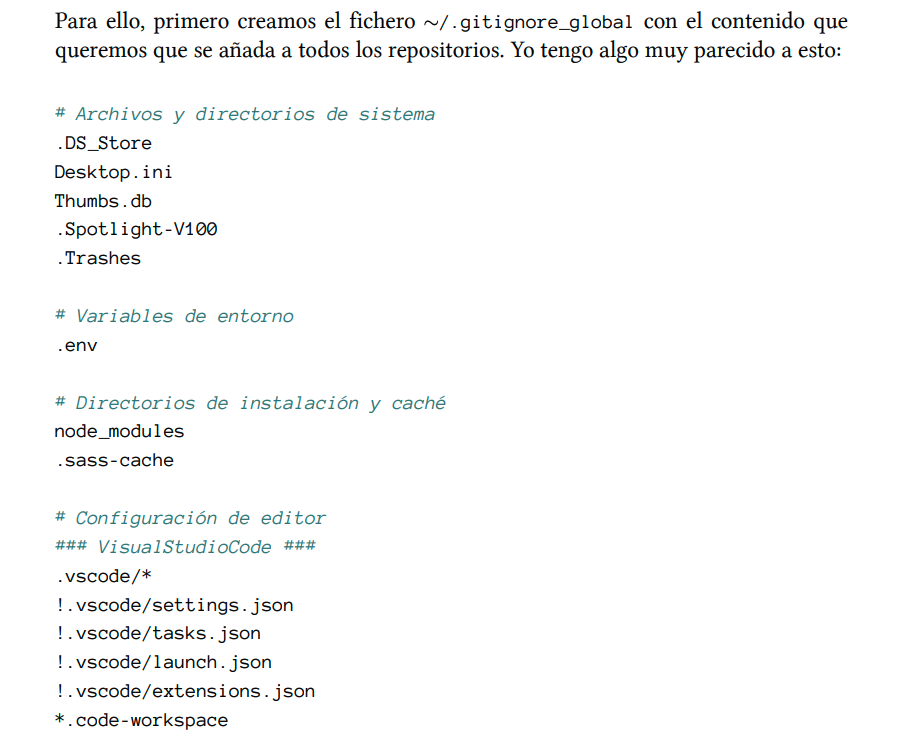
****

1. Le hacemos el commit tb

****



**¿Cómo consigo ignorar siempre los mismos archivos en todos mis repositorios?**



Ahora, vamos a actualizar la configuración de core.excludesfile para que lea este archivo de forma global en todos los repositorios locales.

git config --global core.excludesfile ~/.gitignore\_global

**¿Cómo le indico a Git que deje de hacer el seguimiento de un archivo (o varios archivos)?**

Imagina que has hecho commit de un fichero pero resulta que, más tarde, te has dado cuenta que en realidad no quieres que este archivo sea parte de tu repositorio. ¿Ahora que hacemos?

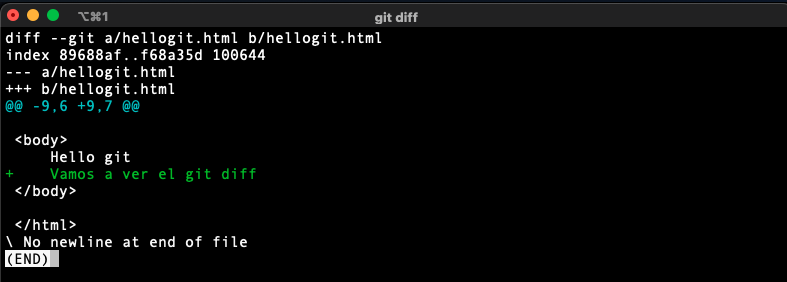
Ya hemos visto que deberíamos haberlo añadido al .gitignore… pero a veces nos damos cuenta tarde.

$ git rm config.local.js

$ git commit -m "Remove config.local.js to ignore it"

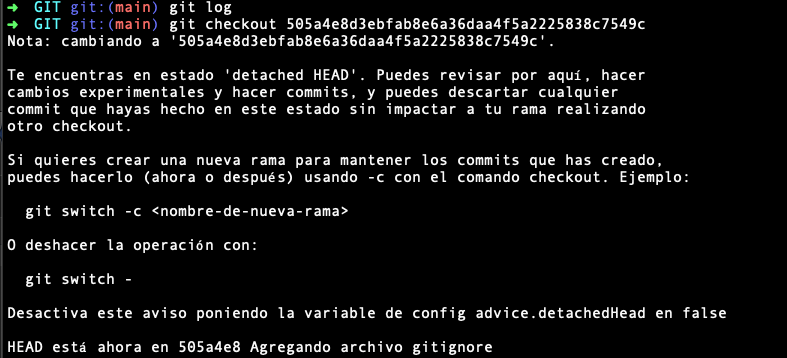
1. **BUSCAR QUE HE CAMBIADO**

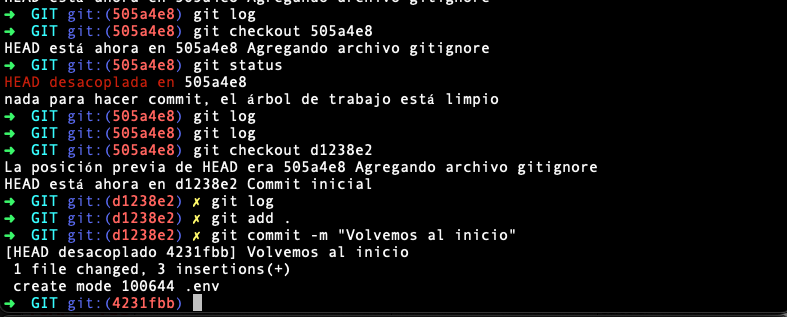
Modifico algo y sin hacer add o commit quiero ver los cambios porque me he perdido

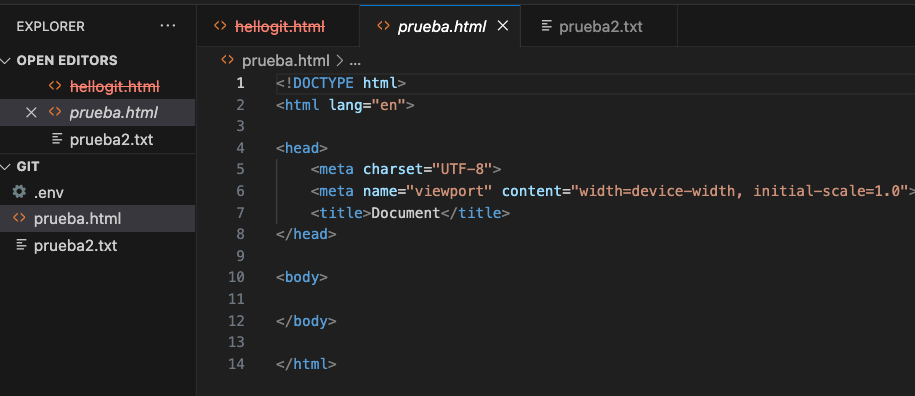
****git diff

1. **DESPLAZAMIENTOS ENTRE LOG CON EL HASH**

git checkout número de hash del log o las 7 primeros dígitos del hash

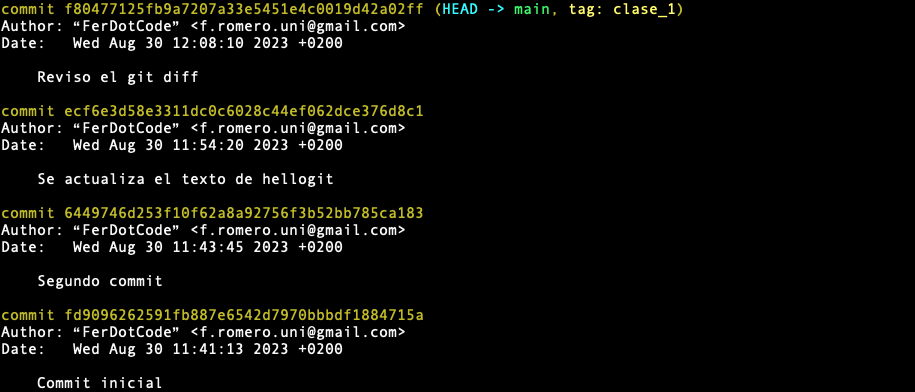






1. **CREAR UN TAG**

git tag {nombre identificación}



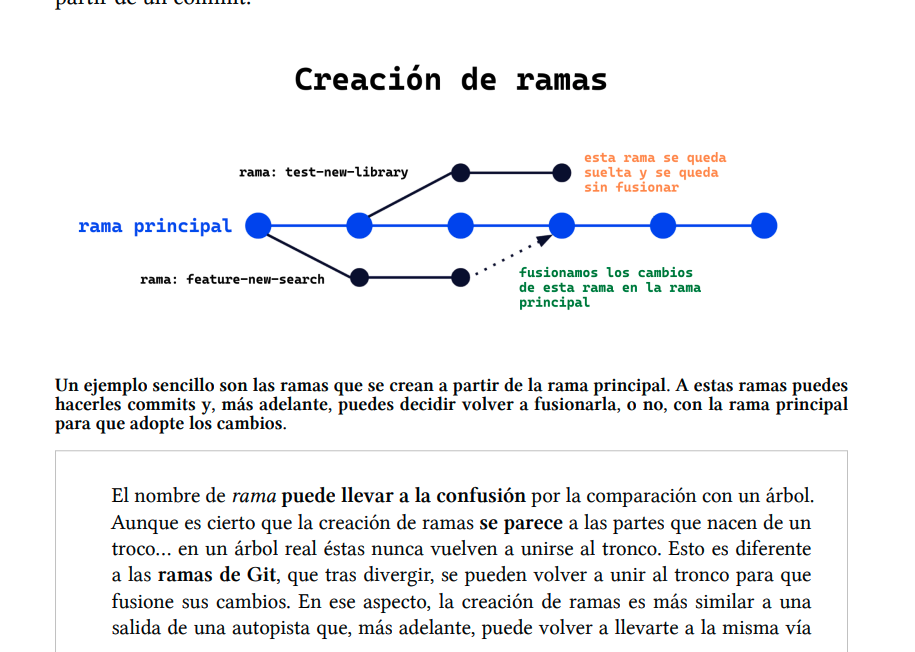
Sirve para dejar marcado un momento de la sesión y nos podemos mover entre tgs

git tag/nombreTag

Se utiliza para establecer diferentes flujos de trabajo por ejemplo que me he dejado un error pasado.

Se utiliza para crear puntos concretos a lo largo del trabajo.

**RAMAS EN GIT**

****

La creación de ramas nos permite el trabajo en paralelo sobre una misma base

de código o, dicho de otra forma, una rama representa una línea completamente

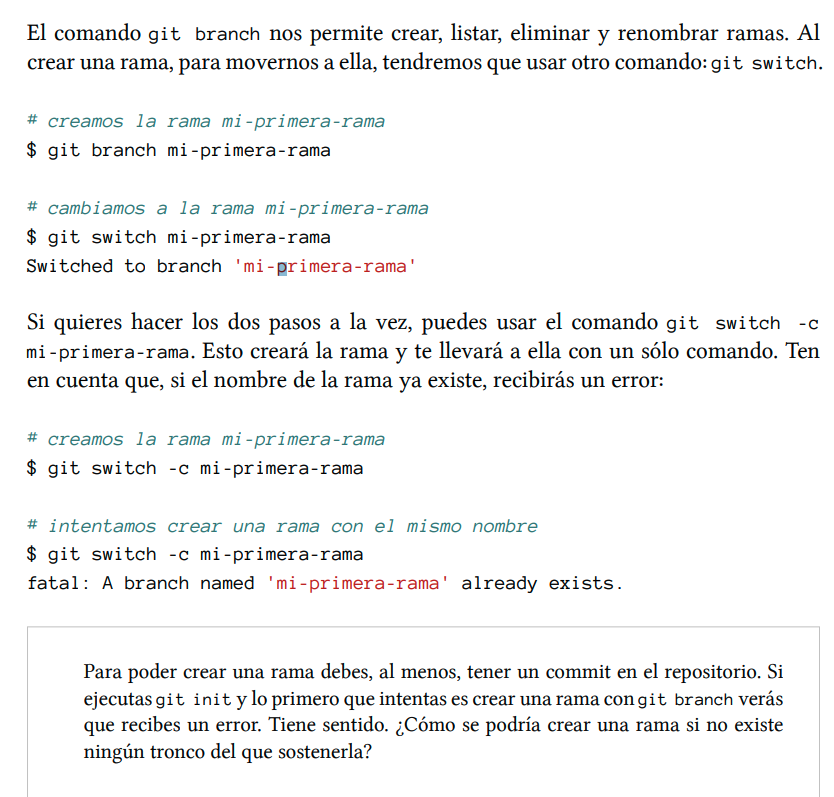
independiente de desarrollo.

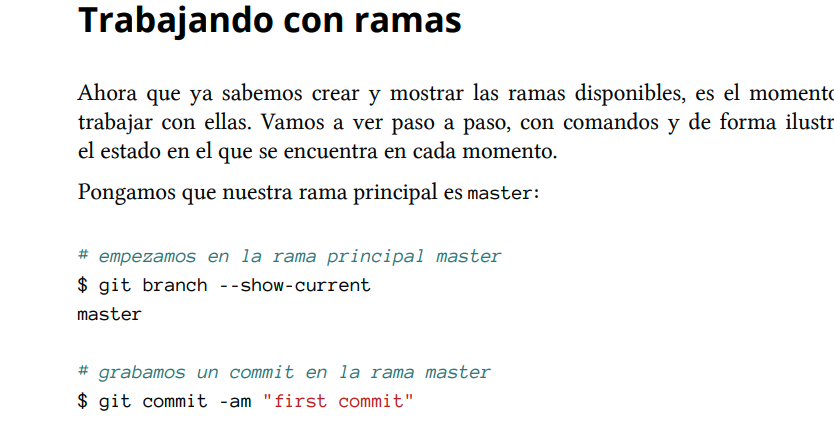
Al grabar cambios con commits en una rama, se genera una bifurcación en el historial

de cambios del proyecto. Estos cambios pueden ser más adelante integrados en

otra rama (normalmente la rama principal) o se puede eliminar la rama y dejar los

cambios sin efecto.





1. **CREAR RAMA**

git branch nombreRama

# creamos nuestra primera rama y # cambiamos a ella para empezar a trabajar

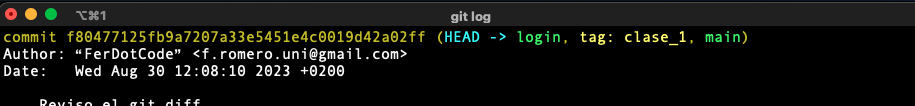
$ git switch -c my-branch

1. **MOVERSE POR LAS RAMAS**

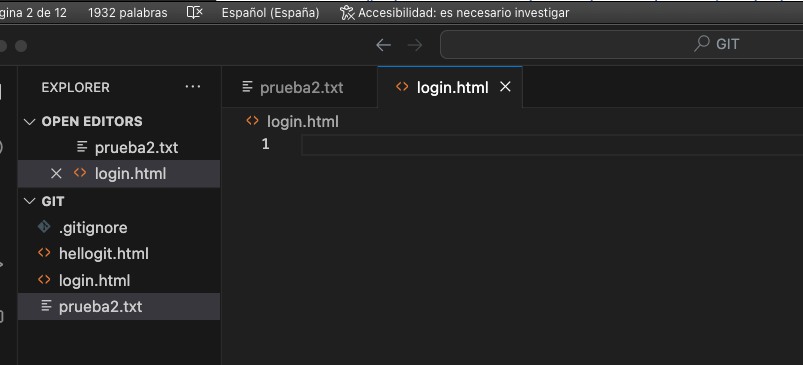
git switch nombreRama



Head apunta a la rama donde estés

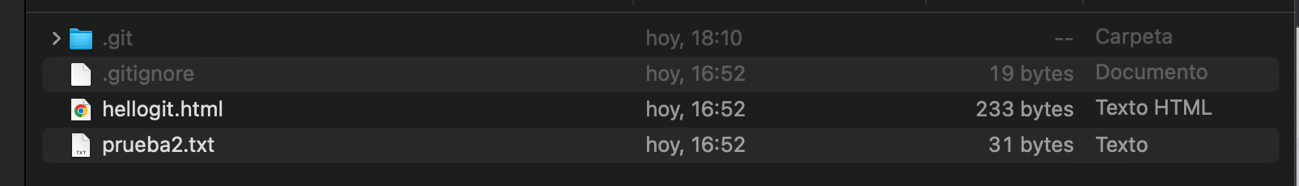


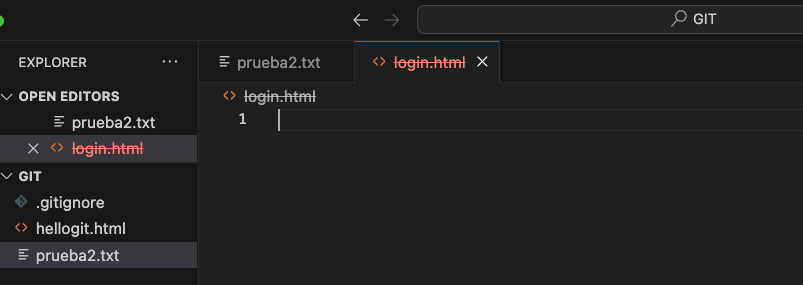
Si nos ponemos en la rama login y hacemos un add y commit vemos lo siguiente:





En la carpeta y en vscode tenemos los cambios realizados pero si cambiamos a la rama main no los tenemos:



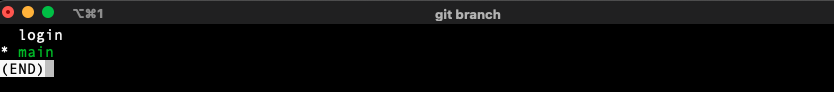


Para ello hay que hace run merge o fusión de ramas.

**Listando las ramas disponibles**

git branch

Verás que una de las ramas tiene, al principio, un asterisco. Esto significa que actualmente te encuentras en esa rama.



**Ver la rama en la que estoy**

git branch --show-current

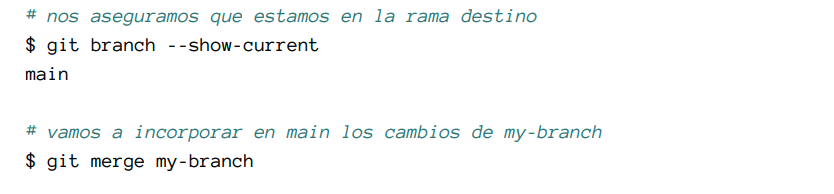
main

**Listando las ramas ordenadas por orden de creación**

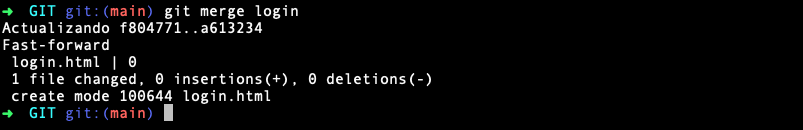
git branch --sort=-committerdate

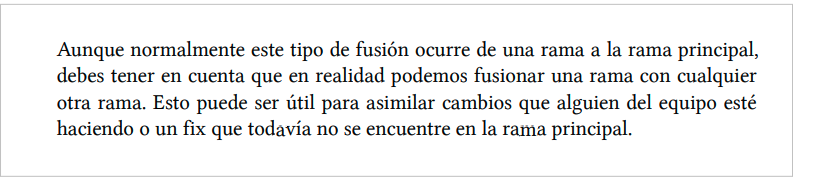
1. **MERGE FUSIONAR RAMAS** merge commit

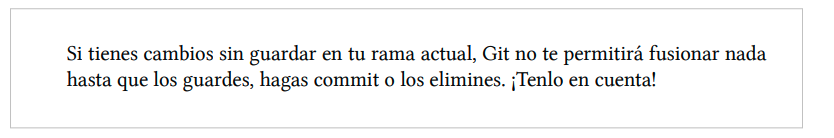
**Posicionados en la rama a donde queremos fusionar cambios o añadir.**

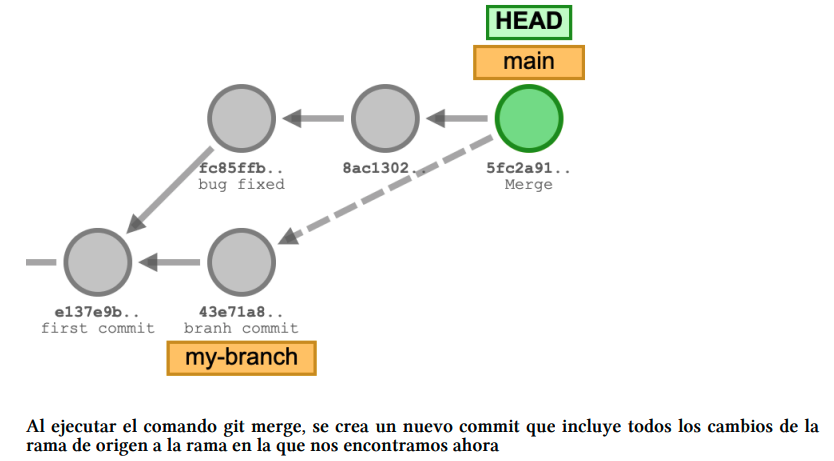


git merge [RAMA QUE QUEREMOS FUSIONAR]









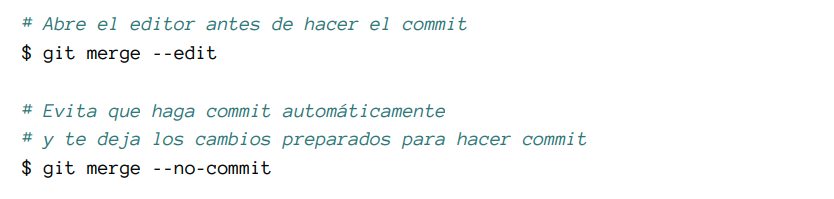
**OTROS TIPOS DE MERGE**

Fast-forward

No fast-forward

Squash

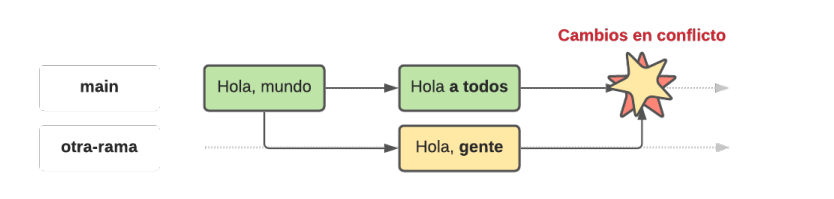
**Modificando el mensaje de Merge commit**



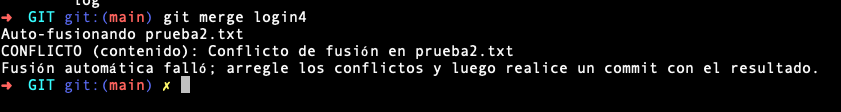
**CONFLICTOS AL USAR RAMAS Y MERGE**

Un conflicto es una situación en la que Git no es capaz de determinar qué cambio es el que tiene que prevalecer una vez ocurra la fusión y, por lo tanto, requiere que el usuario lo

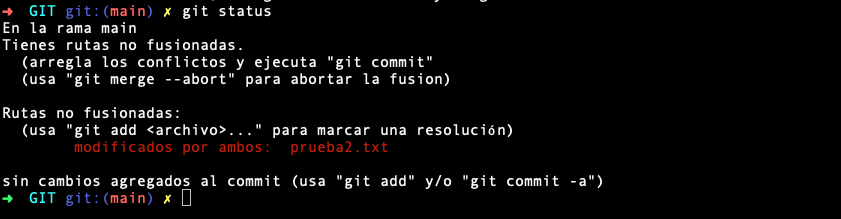
resuelva.

****

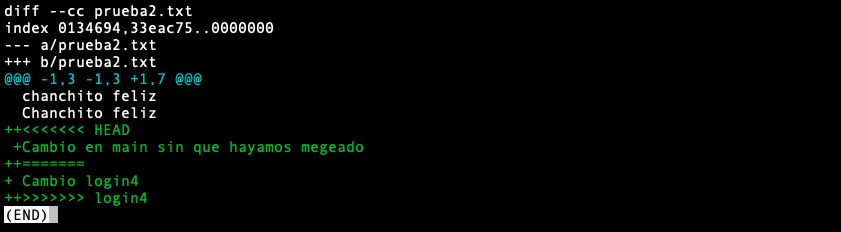
Se abre el editor y debemos decidir qué cambio dejamos. Cuando hay un conflicto resuelto en vscode nos marca el archivo con M para indicarnos que ha sido modificado, pero hay que hacer un commit para dejarlo todo grabado en el repo local y así desaparece la indicación.



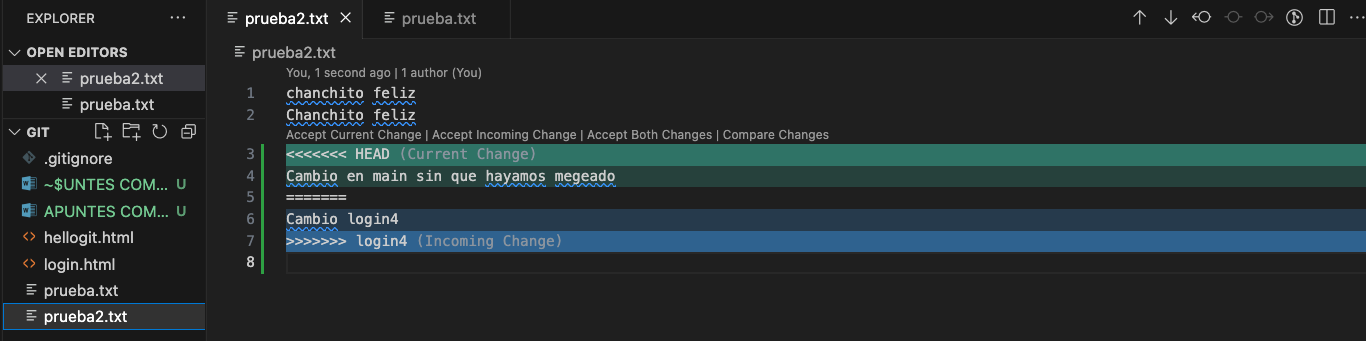
1. git status para ver que fichero tiene conflicto

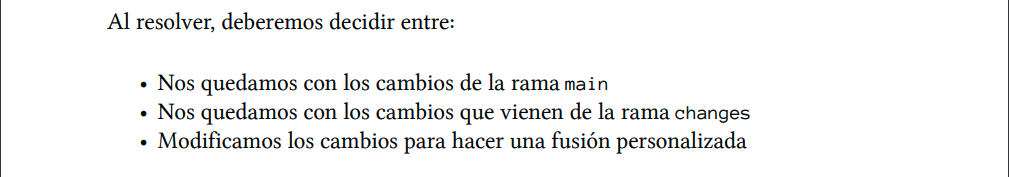
****

1. Nos da dos opciones merge –abort o solucionar el conflicto y hace otro merge.
2. Ver el contenido del archivo con conflicto git diff

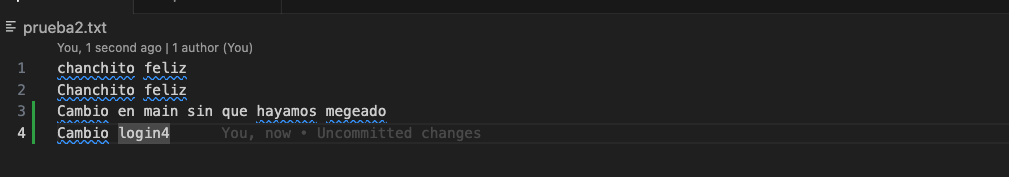
****

Arriba tenemos el contenido que ya existía en la rama de destino de la fusión (la rama main) y debajo de la línea divisoría ======= tenemos el contenido de la rama de origen (la rama login4), que queremos incorporar a la rama main.





1. Borramos o modificamos manualmente para tener el control desde el vscode



1. git status para ver lo que tenemos pendiente aún ahora
2. git add nombreArchivo
3. git commit -m “mensaje”
4. conflicto resuelto

**ELIMINAR RAMAS**

**Si la rama ha sido fusionada**

$ git branch --delete nombreRama

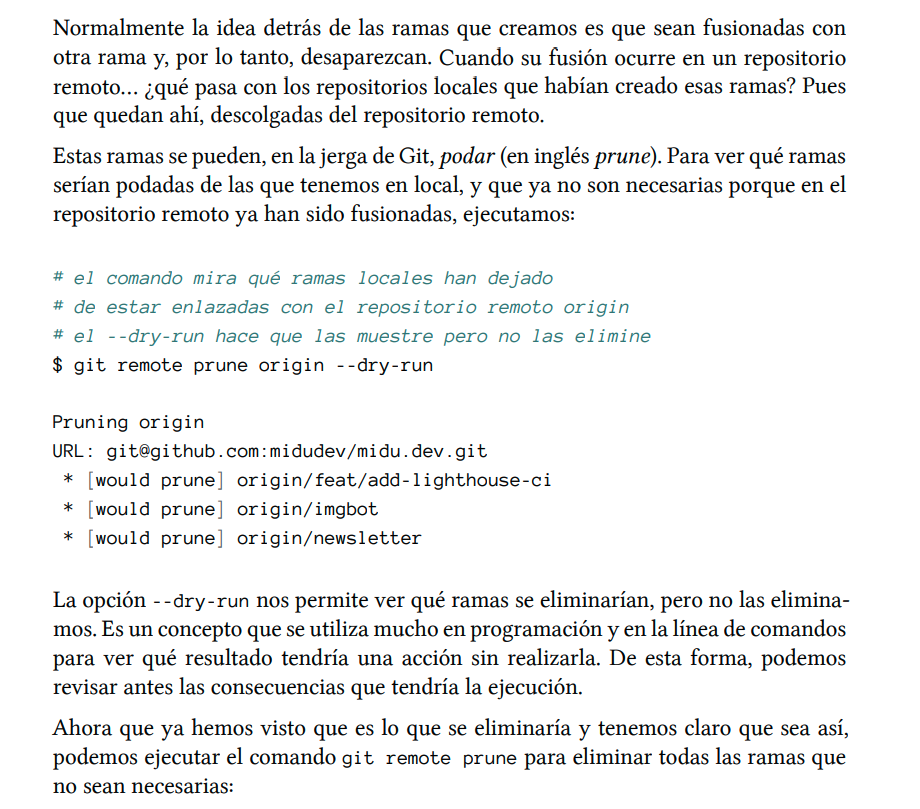
Sino dará error y nos dará el comando para borrarla de todas formas

git branch -D nombreRama

**¿Cómo eliminar ramas de mi repositorio local que ya no se usan?**

$ git remote prune origin --dry-run

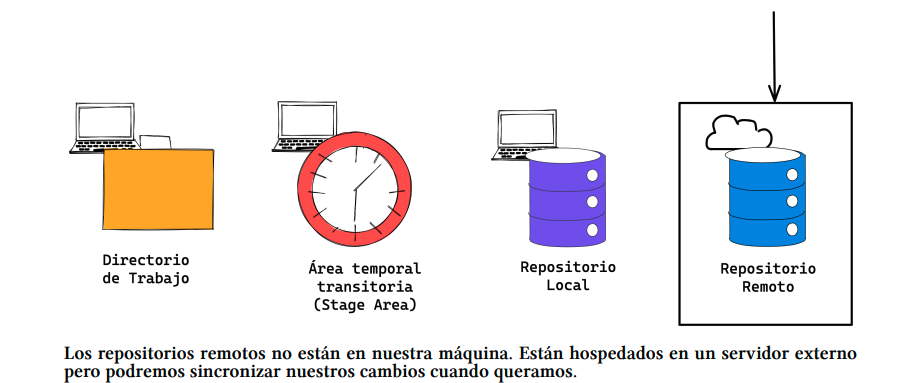
git remote prune origin PODAR RAMAS DEL REPOSITORIO REMOTO NO USADAS



**REBASE (merge pero reordena commits) Es peligroso de usar**

El rebase es una operación que nos permite reescribir el historial de commits de una rama. Esto significa que podemos añadir, mover, ordenar o eliminar commits directamente de un historial de commits. Es como tener el superpoder de viajar en el tiempo y cambiar el pasado, presente o futuro de nuestra rama.

**GIT EN REPOSITORIO REMOTO**

****