Curso Git

Configurar Git

1. Git tiene 3 tipos de jerarquía:  
   Sistema – Esto es para toda la computadora, para todos los usuarios, etc.
2. Global – Esto aplica para todos los repositorios de un usuario.
3. Local – todo lo que definamos en un bloque va a ser valido ahí.

Siempre el más pequeño es al que se le da prioridad (a lo más específico).

Para configurar el global es:

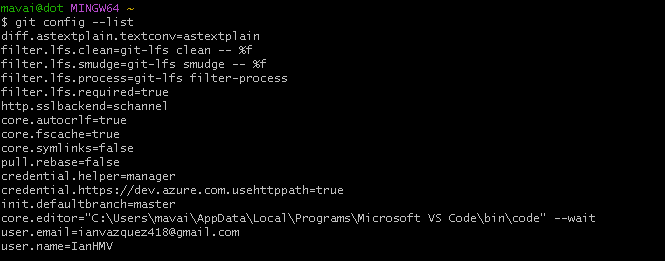
git config --global user.name “IanHMV”

git config --global user.email “ianvazquez@gmail.com”

(tomalo como que son campos del apartado “user”)

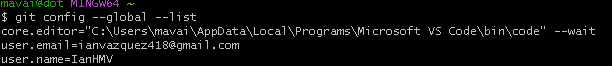
Para ver las configuraciones que tenemos hechas seria con:

git config --list



Para ver únicamente las de global es con:

git config --global --list



Para configurar editor de código:

En este caso en específico configuraremos visual studio code:

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

code = visual studio code

--wait = Esto hace que cada cambio va a esperar a que nosotros cerremos nuestro editor de código para ejecutarse (HASTA QUE CIERRES EL EDITOR SE CONFIRMAN LOS CAMBIOS).

Configurar el colerizado:  
git config --global color.ui true

De esta forma las salidas en pantalla que git nos devuelva van a ser más coloridas.

Configurar saltos de linea (necesario):

En Windows:

git config --global core.autocrlf true

En linux:

git config --global core.autocrlf input

autocrlf = auto(automatico) cr (carriage return) lf (line feed)

Esto es necesario por un que en Linux y en Windows es diferente el salto de línea.

En windos es asi: \r\n

En Linux es asi: \n

Entonces lo que hace esto en WINDOWS es que si lo descargamos el archivo en Linux no va a haber errores de salto de línea.

Primeros pasos en git:

* Repositorio: Lo que está en línea.
* Stationing área (área de preparación): Donde subimos los cambios que después serán subidos al repo.
* Área de trabajo: La carpeta o área en la que esta nuestro proyecto.

cd = entrar a una carpeta

cd ../ = salir de una carpeta

mkdir = crear un directorio (mk=make, dir=directory)

rmdir = eliminar una carpeta

pwd = verificar nuestra ruta

touch = crea un archivo

Inicializar git:

git init

Mostrar elementos ocultos:

ls -a

Subir archivos:

git add . (con esto se agregan todos mis archivos de mi área de trabajo los movemos al área de preparacion)

IMPORTANTE: No es una mala práctica, tampoco es recomendado, pero es mejor escribir el nombre del archivo a subir por si se llega a escapar un archivo que no.

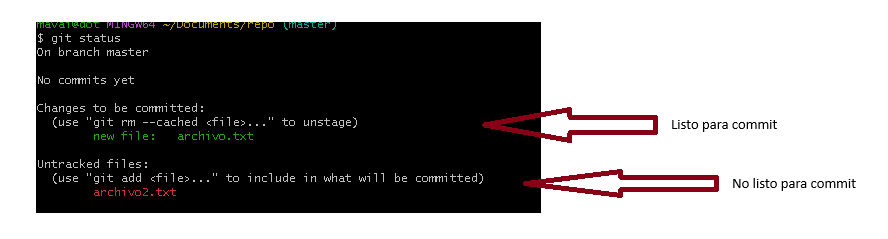
En este caso:  
git add archivo.txt

Como ver información acerca de nuestra área de trabajo y nuestro entorno de preparación:

git status

Remover un archivo del área de preparación:

git rm --cached <nombreDelArchivo>



Para subir mas de un archivo al área de preparación es con:

git add archivo.txt archivo2.txt

Hacer un commit (todo lo que tenemos en el área de preaparacion subirlo al repositorio):

git commit

Hacer un commit con un mensaje:

git commit -m “<mensaje>”

Si solamente hacemos “git commit” probablemente nos salga el editor de código pidiéndonos el mensaje que especifique que hace nuestro commit, en ese caso simplemente agregamos el puro mensaje (no tiene que tener comillas ni nada) y cerramos el editor de texto.

Como subir los archivos sin pasar el área de preparación:

git commit -a

Restore, Checkout y más

Como eliminar un archivo de un repositorio:

rm archivo2.txt

Como notificar a git que agregue la eliminación del archivo (es de la misma forma que notificamos que agregamos un archivo):

git add archivo2.txt

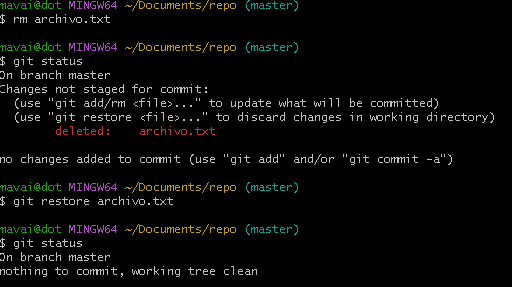
Como hacer para subirlo sin pasar por el área de preaparacion:

git commit -m "elimando archivo" -a

Como recuperar un archivo eliminado:

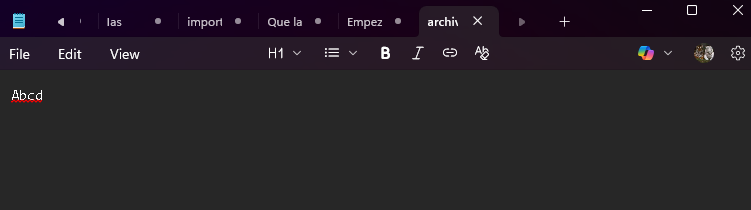
git restore <nombreDelArchivo>

Ejemplo:

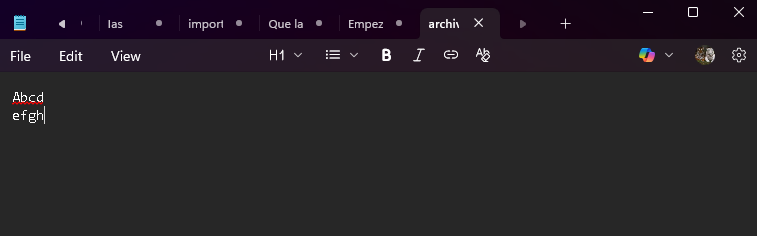


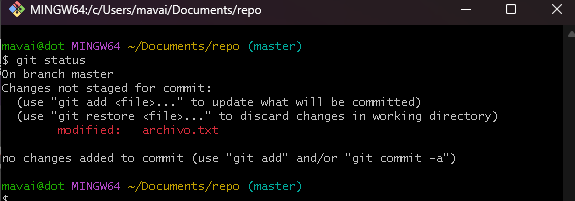
git checkout:

Si tenemos un archivo con estas letras (que ya este se le haya hecho un commit anteriormente):



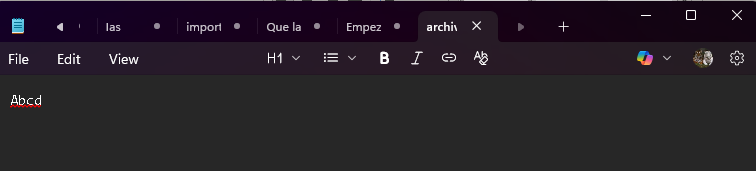
Y hacemos un cambio:



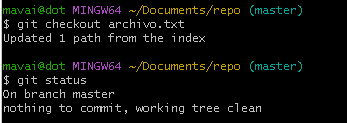


Y después le damos “git checkout <nombreDelArchivo>”:



Nos lo devuelve a como estaba antes:  


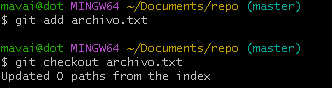
Y aparece sin cambios en la consola si le damos “git status”:



PERO, si hacemos un cambio y lo subimos al área de preparacion:



No me va a dejar regresarlo (dado a que ya lo subí al área de preparación):

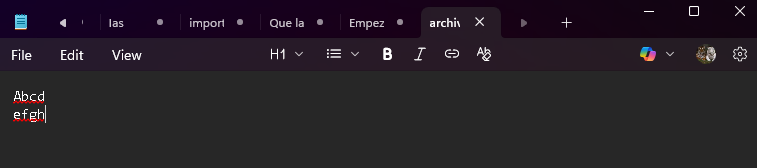


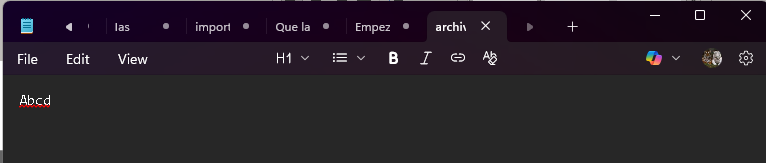
Git reset:

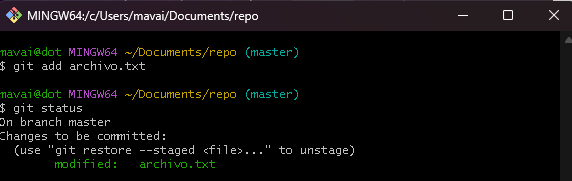
Como le hago para que una vez hice un cambio y lo subi el cambio al área de preparación poder bajar ese cambio y devolver el archivo a su estado original:

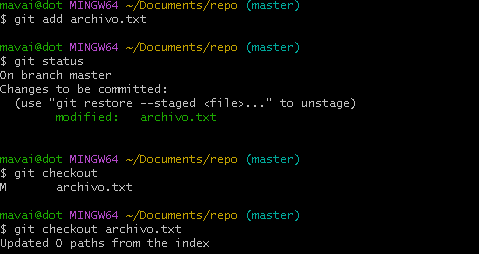
git reset --hard

Ejemplo:

Tenemos este archivo:  


Borramos la segunda fila y guardamos el cambio y lo subimos al área de preparacion:  


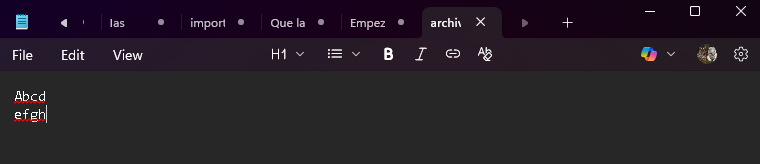


No podemos hacer un checkout para devolver el archivo a su estado original:  


Entonces para bajar ese cambio del área de preparación y regresar el archivo a su estado original es con:

git reset --hard

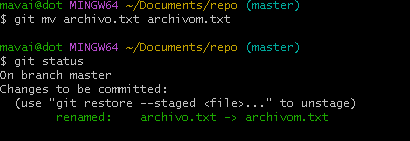


Y el archivo vuelve a estar en su estado original:  


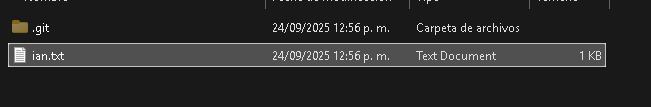
IMPORTANTE: puede ser una mala practica usarlo mal.

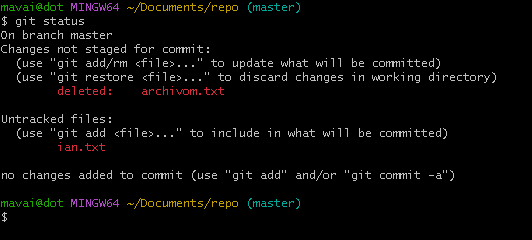
Como cambiar el nombre de un archivo:

git mv <archivoInicial.txt> <archivoModificado.txt>



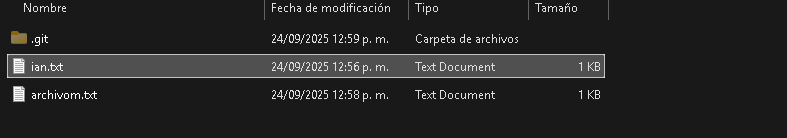
Si ahora cambiamos el nombre de “archivom.txt” a “ian.txt”:

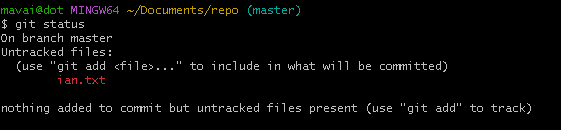




Y luego le damos:

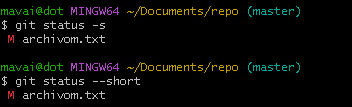
git checkout archivom.txt

Nos devuelve el archivo y nos deja el anterior que modificamos con el nombre:  




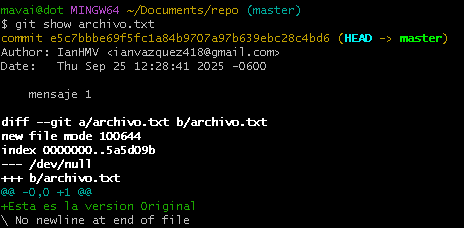
Como hacer que “git status” nos de algo más limpio:

git status -s ó git status --short (es lo mismo)



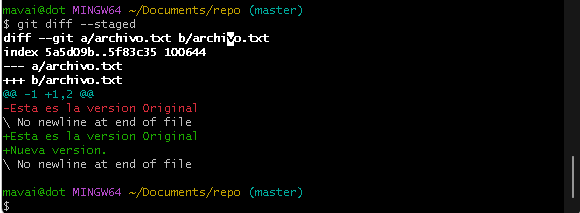
git diff:

Para poder ver lo que hay dentro de un archivo COMMITEADO, podemos usar el comando:  
git show <nombreArchivo>



NOTA: no muestra la modificación si no lo commiteas.

Como comprar algo que está en área de preparación vs algo que esta en le commit:  
git diff --staged



Como comparar entre commits:

1. Para eso necesitamos primero la identificación del commit, que se obtiene con:

git log

también para abreviar podemos usar:

git log --oneline

configuramos git para que nos de los caracteres que nosotros le indiquemos en el identificador:

git config --global core.abbrev <numeroDeCaracteres>

Ejemplo:  
git config --global core.abbrev 5

1. Para después para ver lo que cambio de un commit a otro es con:

git diff <hash1> <hash2>

Ejemplo:

git diff 51671 e5c7b

Para ver únicamente los nombres de los archivos que cambiaron seria con:

git diff --name-only 51671 e5c7b

Si quisiera ver a nivel de las líneas lo que cambio es con:  
git diff --word-diff 51671 e5c7b



Blanco: lo que estaba antes

Rojo: lo nuevo o ultimo

Modificar y deshacer commits:

Para modificar el comentario que nosotros le asignemos al último commit es con:

git commit --amend

Esto en dado caso de que solamente queramos corregir el mensaje y no subir nada

Pero en dado caso de que olvidáramos agregar algo y lo queramos modificar subiéndolo, tendríamos agregar los nuevos archivos con “git add” y ahora si darle nuevamente:

git commit --amend

NOTA: Cambia hasta el hash

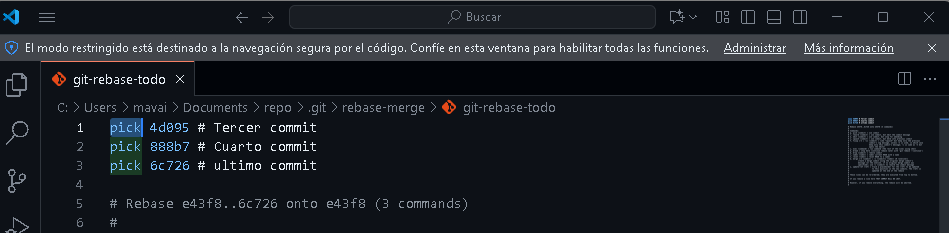
CONSEJO: Siempre intentar modificar SOLAMENTE el ultimo commit.

Como le hago para regresar a commits anteriores:

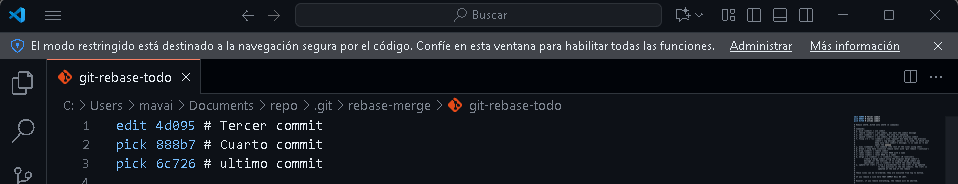
git rebase -i head~<numeroDeCommit>

Ejemplo:  
git rebase -i head~3

NOTA: es el numero de commits que queremos ir hacia atrás, después nos salta esto:



Y tenemos que colocar “edit” al que queramos modificar:



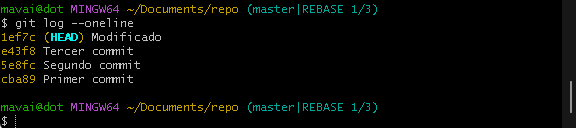
Y después lo cerramos.

Despues ya podemos colocar la instrucción para modificar:

git commit --amend

Modificamos el commit, guardamos y cerramos.

Lo cual ahora al momento de dar “git log --oneline" nos dará:

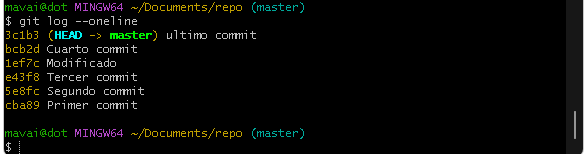


Esto significa que los próximos commits al tercero se borran.

Y para eso tenemos que hacer el “rebase”:

git rebase continue

Y nos devuelve los commits:



NOTA: esto modifica todos los hashes.

Tenemos 3 formas de utilizar el “git reset”:

Soft

Mixed

Hard

Como deshacer un commit:

git reset --soft <hash>

Lo que hace es agarrar los archivos que teníamos en ese commit y nos lo mueve al área de preparación.  
  
Tambien lo que podemos hacer para irnos 1 hacia abajo es con:  
git reset –soft head~1

(si usamos soft los archivos que tenemos en el área de espera se van a mantener, y los que necesita sobrescribir los sobrescribe)

Si usamos mixed:

git reset --mixed 5e8fc



Nos regresamos al commit del hash y se limpia el área de espera, pero el área de trabajo no le pasa nada.

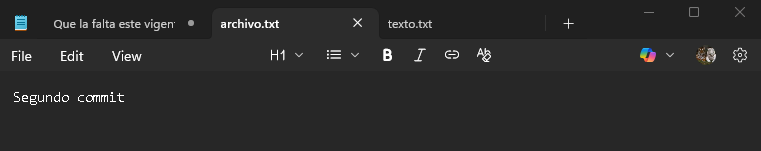
Y que pasa con hard?

Lo que pasa con hard es agarrar el utlimo commit y borrarlo completamente:

Descarta todos los cambios de nuestra área de trabajo y agarra los archivos que estaban en el commit que elimino y los coloca en el área de trabajo. Ademas descarta todo del área de espera.

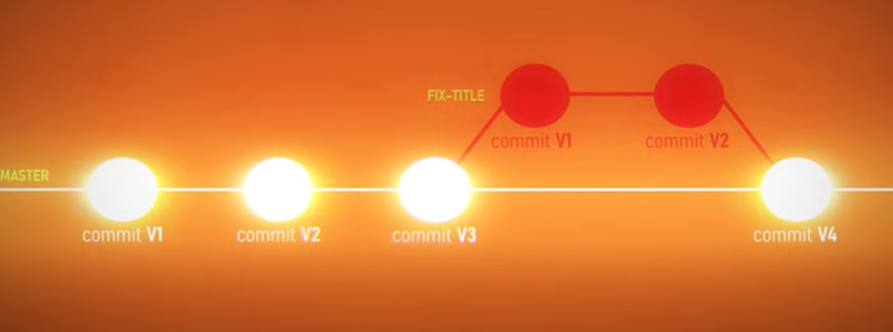
git reset --hard 5e8fc

Como se puede ver, el archivo ya no dice “ultimoo”, dice “segundo commit”



Si nosotros hacemos una modificación y después queremos usar “hard” tenemos que guardar primero los cambios que hicimos.

Branches (ramas):



Mostrar todas las ramas que tenemos creadas:

git branch

Como crear una rama:  
git branch <nombreDeRama>

Como moverme a la rama:

git chechout <nombreDeRama>

Forma recomendada de moverse entre ramas:

git switch <nombreDeRama>

Como crear una nueva rama y al mismo tiempo moverme a ella:

git checkout -b <nombreDeRama>

Como hacerlo con “switch”:

git switch -c <nombreDeRama>

Como eliminar una rama (se tiene que hacer desde otra rama):  
git branch -d <nombreDeRama> (-d: delete)

(1ra). Como modificar el nombre de la rama (se tiene que hacer desde otra rama):

git branch -m <nombreAnteriorRama> <nuevoNombreRama>

(2da). Como cambiar el nombre de la rama actual:  
git branch -m <nombreDeLaRama>

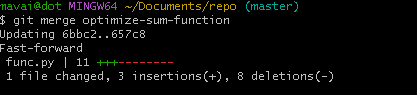
NOTA: para modificar el nombre desde otra rama es con la (1ra), si queremos modificar el nombre de la rama desde la misma rama es con la (2da)

Fusionar Ramas (merge):

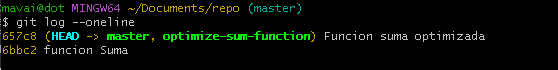
Primero tienes que colocarte en la rama a la que quieres que se apliquen los cambios:



Una vez ahí tienes que indicarle cual es la rama de los cambios que quieres agregar, para así aplicarlos:



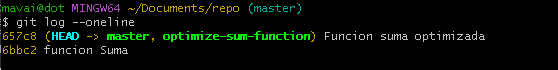
De esa forma se aplican los cambios a la rama en la que estas posicionado:



Si nos equivocamos y fusionamos una rama que no debimos haber fusionado es con:  
git reset –hard <Nombre de la versión del commit que estamos o a la que queramos volver>

Ejemplo:

Si tenemos estas ramas fusionadas:



Pero la optimización no funciono y queremos volver a la version anterior (función suma)

Entonces tenemos que colocar:

git reset --hard <hashDeVerisonAnterior>

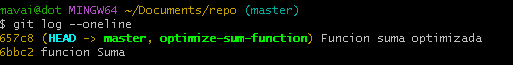
Esto desfusiona la rama, pero no borra el archivo, solo borra el commit:  

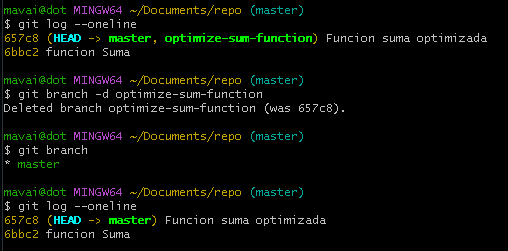

El archivo sigue estando en la rama que tu hiciste:  


Esto pasa porque las ramas muestran los commits de ellas mismas y de la rama padre

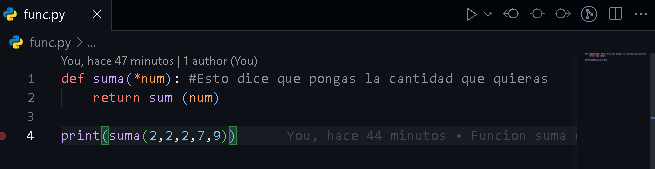
Por lo que si quiero volver a fusionarla tengo que hacer:

git merge <ramaQueQuieroFusionar>

Y de nuevo volvería a mostrar las 2 ramas:  


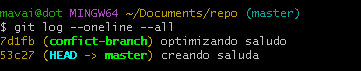
E incluso si borras la rama:  


Los cambios siguen estando en master:



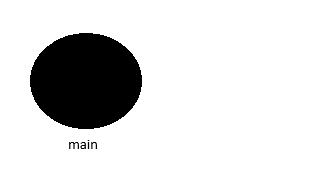
Merge Conflicts:

Como hacer para que nos muestre todos los commits (todos, los de las ramas también):  
git log --oneline --all

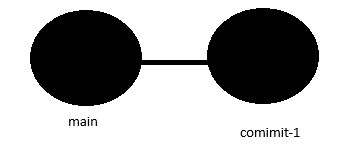


Ejemplo:

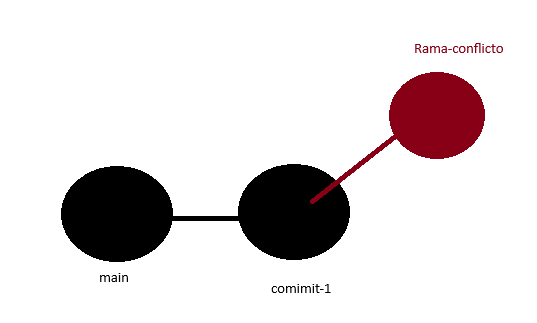
Digamos que principalmente tenemos esto:



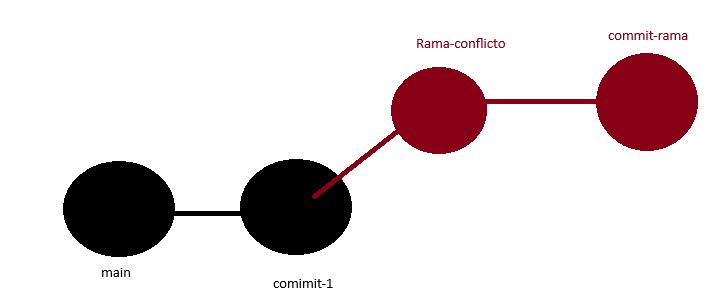
Y después hacemos un commit con un cambio:



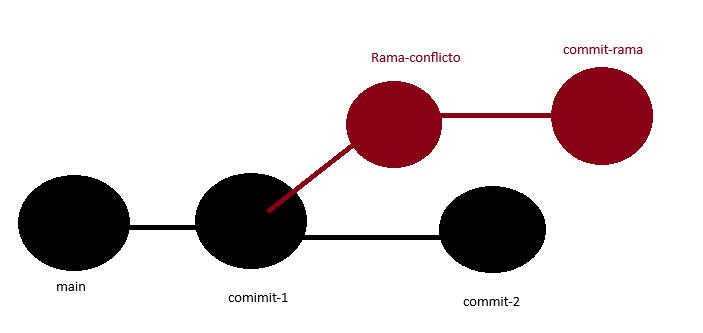
Después creo una nueva rama (sin mergear los cambios):

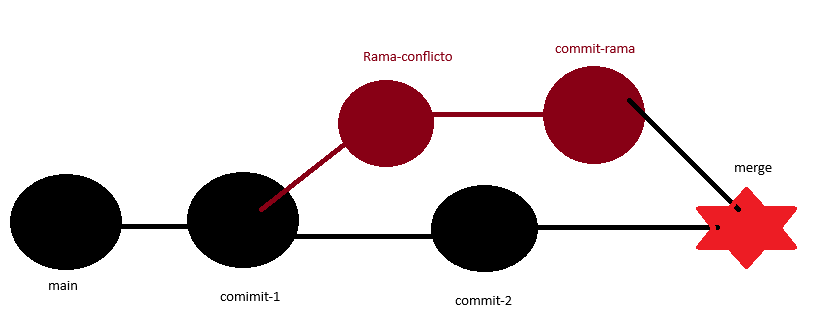


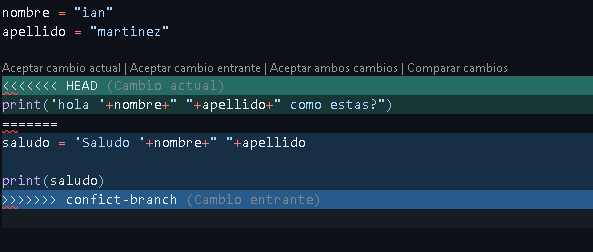
Y hacemos un commit:



Y nos regresamos a main y hacemos otro commit:

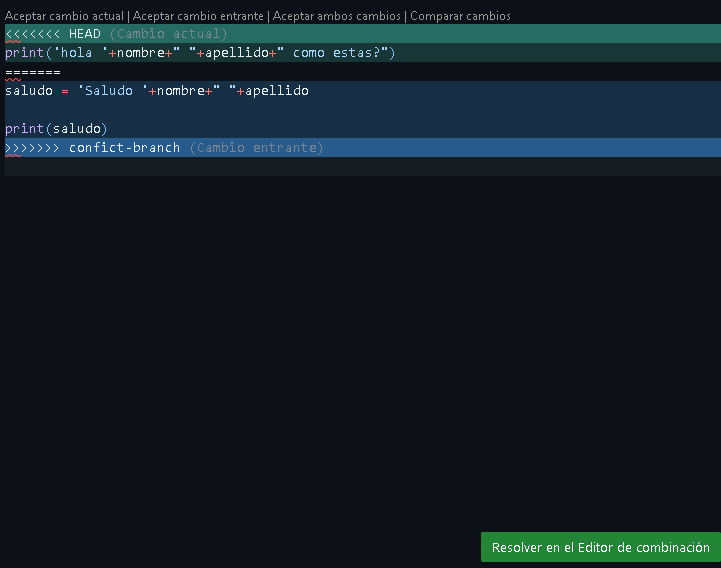


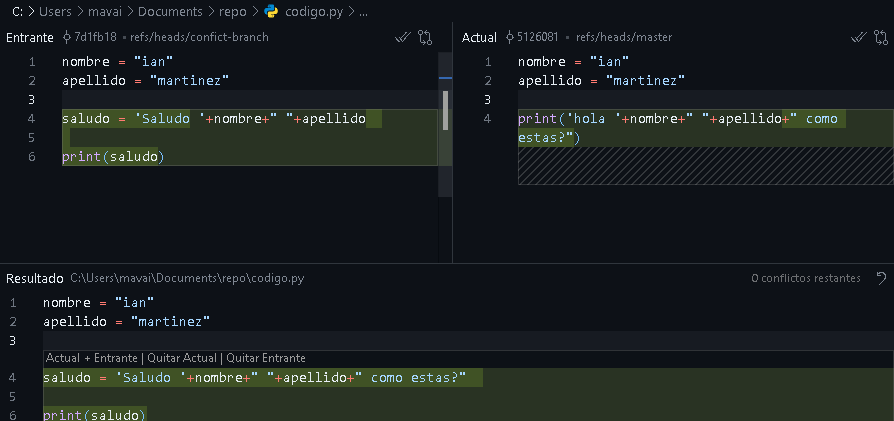
Al momento de fusionarlos nos va a marcar un error pese a que especifiquemos que queremos fusionar la rama, porque a final de cuentas los 2 son ramas:  
Por ejemplo colocamos: git merge conflicto-branch

Entonces nos va a saltar un conflicto en vsc donde tenemos que aclarar con cual cambio quedarnos, si con el “commit-2” o con el “commit-rama”:  


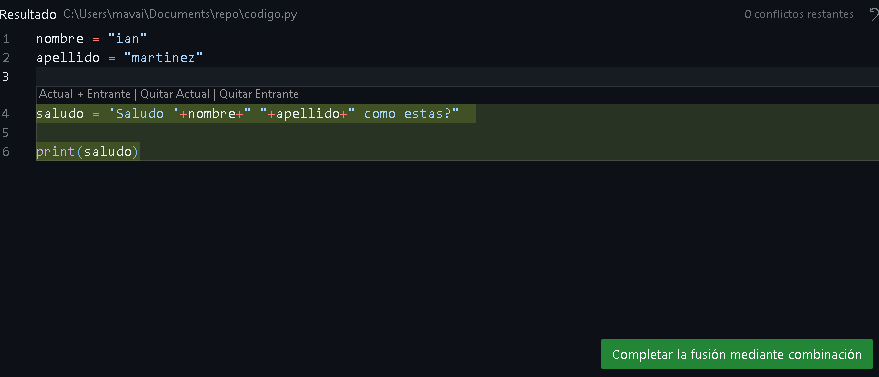
La rama actual es la rama master (antes de la fusión) (current change), y la rama entrante es la de en este caso “conflicto-branch” (incoming change).  
Y nosotros tenemos que elegir cual cambio hacer.

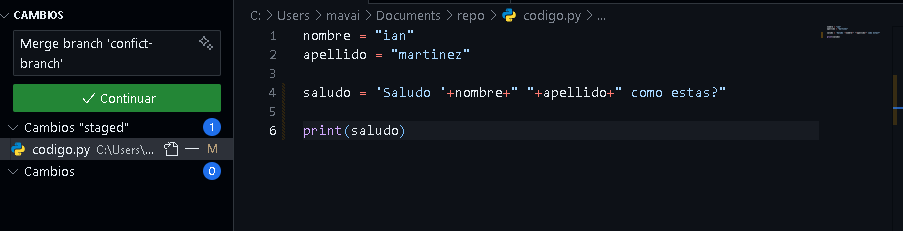
Para eso podemos elegir resolverlo en el editor de combinación (resolve in merge Editor)

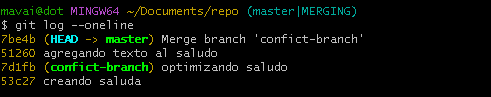




Una vez elegido el cambio podemos darle: completar merge.



Una vez hecho eso le damos en “commit” o en “continuar”:  


Y ya estaría:  


En dado caso de que no lo coloquemos con: continuar o commit, en consola tendríamos que colocar:

gir merge --continue

Archivo git ignore:

Tenemos que agregar un archivo “.gitignore” al principio del proyecto

Son archivos que no queremos que se suban al repositorio.

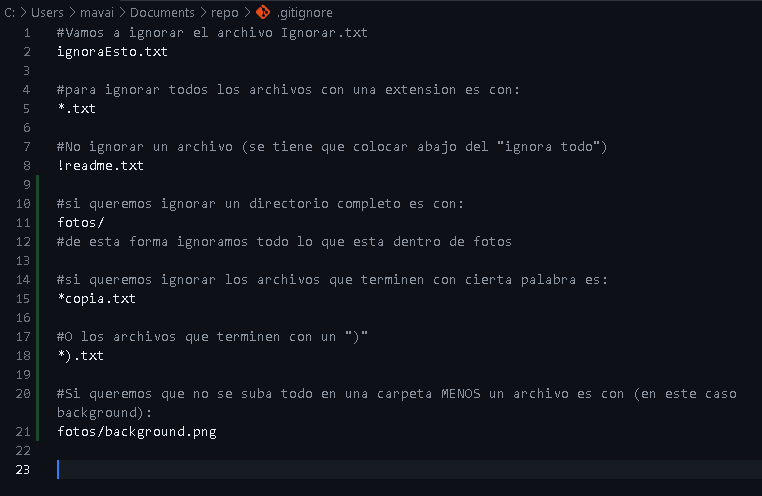
Al momento de utilizar “git add” git va a consultar si hay un archivo .ignore, si lo hay los ignora y si no los agrega.

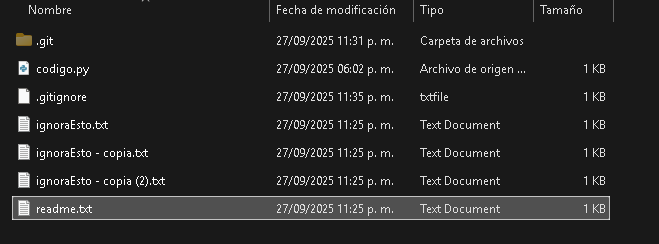
La siguiente línea se usa para mostrar todos los archivos que tienen un commit en específico:

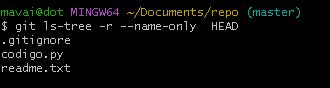
$ git ls-tree -r --name-only <hashDelCommit>

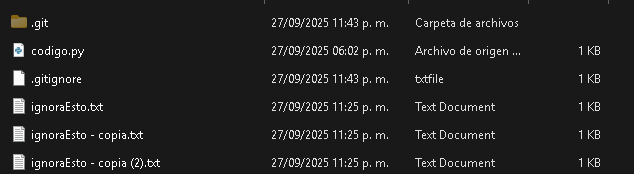
git ls-tree -r (para que te muestre los archivos del commit)

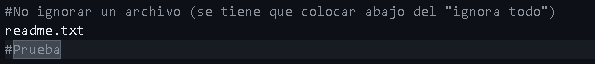
--name-only <hashDelCommit> (para que te muestre solamente el nombre del archivo)

Algunas sintaxis que se usan para ignorar archivos:  


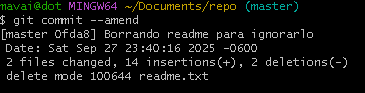
IMPORTANTE: Si anteriormente no colocamos que ignorara cierto archivo (en este caso readme.txt):  


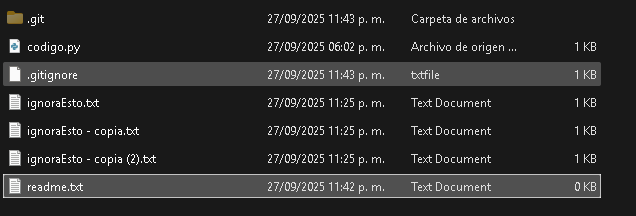
Y hacemos un commit donde lo muestra:  


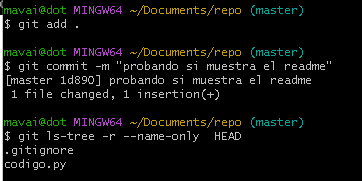
Tendriamos que eliminarlo y colocar la instrucción en .ignore:  




Despues hacer un “git add” y “git commit”



Y de esta forma si colocas el readme nuevamente y haces un ‘git add’ y ‘commit’:  


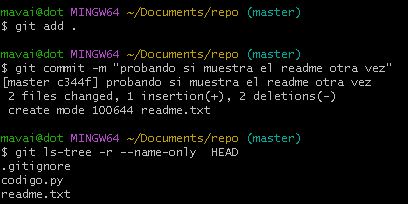


Ya no se muestra 😊

Curiosamente si modificas el ignore:



Y después haces un commit con la modificación:



Si te lo muestra, no ocupas hacer algo más aquí.

Git Ignore Global:

Para esto temenos que hacer una configuración global:

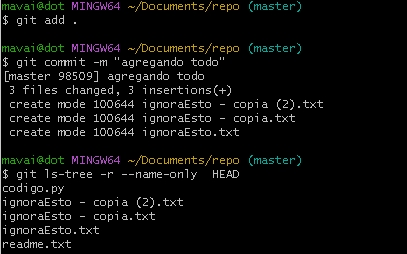
git config --global core.excludesfile <direccion>

Ejemplo:

git config --global core.excludesfile C:/Users/mavai/Documents/.gitignore\_global

Si quisieramos en un caso especial NO se ignorara un archivo, simplemente tendríamos que crear un gitignore en el repositorio y configurarlo, LO LOCAL>GLOBAL

Y en dado caso que queramos borrar el ignoreglobal lo podemos hacer sin ningún problema y subir los archivos:



Alias:

Para colocar alias tienes que hacer una configuración global:

sintaxis:  
git config --global alias.<nombreDelAlias> “<comando>”

Ejemplo:  
git config --global alias.log-mejorado "log --oneline --graph --all --pretty=format:'%C(auto)%h%d %s %C(black)%C(bold)%cr'"

Git reflog:

Es una especie de herramienta que lleva un registro de todos los commits que llevamos.

Lleva un registro de todas las veces, todos los movimientos, que apunto HEAD.

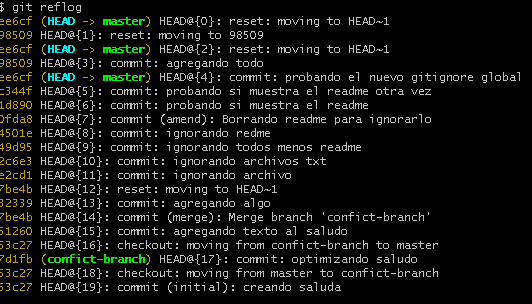
Podemos recuperar un commit eliminado:  
git reset --hard <hashDelCommitEliminado>

Cuando hacemos un “reset --hard" no borramos el commit, si no que borramos la referencia (es como borrar un archivo de tu computadora, no se borra, se borra la referencia).

Como recuperar un commit si no tengo la referencia:

git reflog

Esto lo que hace es mostrarte a donde “HEAD” se fue moviendo:



Introduccion a Github:

origin/master: Hace referencia a nuestro repositorio remoto

push:

Como hacer un “git push”, para esto hay que entender que origin hace referencia al repositorio de donde descargamos el repositorio en github.

El código es:  
git push origin <nombreRama>

Git pull y Git Fetch:

Lo que hace git pull es bajar los cambios que ya sabemos que están en el servidor.

Descargarlos cambios a nuestro repositorio:  
git pull

Que hace git pull?

Git pull en si lo que hace es un “git fetch” y después hace un “git merge”, baja los archivos del servidor y los fusiona directamente a la rama.

En si lo que hace es:  
1ro. Se trae los cambios (fetch)

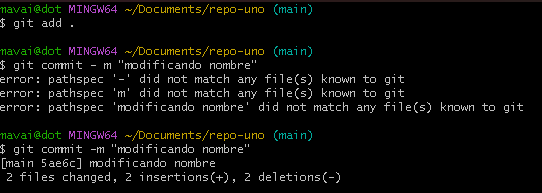
2do. los fusiona con la rama (merge)

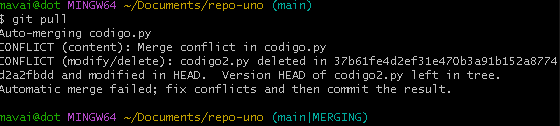
Si tenemos un error al momento de hacer un git pull, ya se porque estamos intentando hacer un cambio a un archivo eliminado en github u otra cosa.

Lo primero que debemos de hacer es:

Agregar nuestros cambios en local:  
git add .

Hacer un git commit:

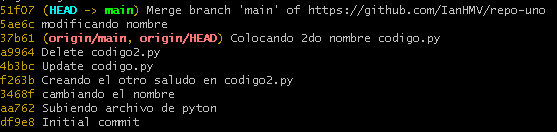


Y después darle git pull:  


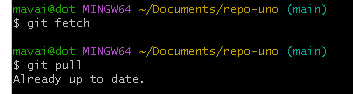
Lo cual nos pedirá solucionar los conflictos:



Los solucionamos y hacemos un commit:



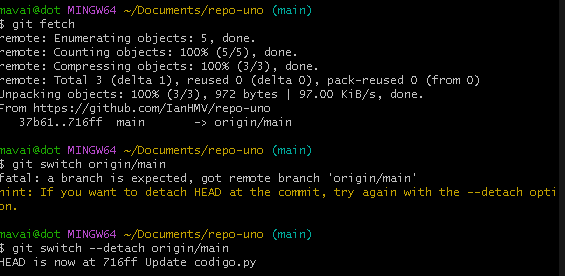
Despues le damos git fetch y git pull:

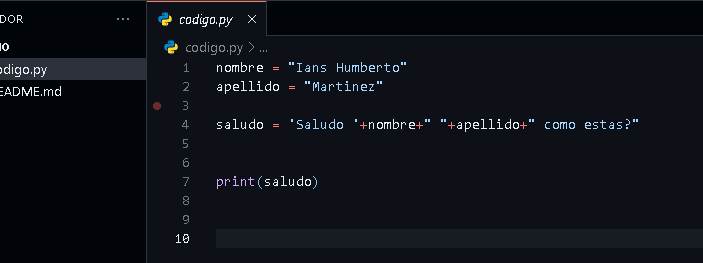


Si queremos ver los cambios que hay en el repositorio y posicionarnos con los cambios que hay en nuestro repositorio. Para eso hacemos un git fetch para bajar los cambios:

git fetch

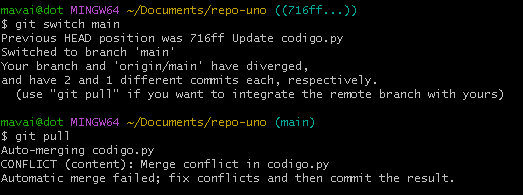
Despues cambiamos a la rama origen en un área temporal:  
git switch --detach origin/main

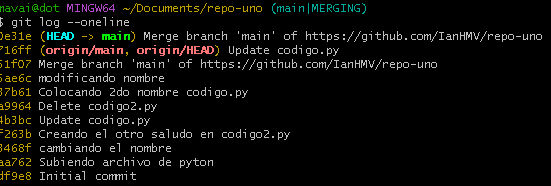


Y nos posicionaría en la rama con los cambios que hay en nuestro servidor:  


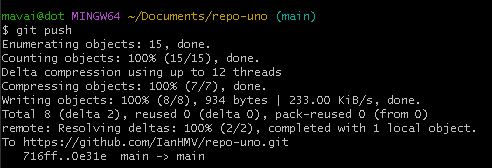
Una vez confirmamos que el cambio nos sirve y este bien volvemos a la anterior:  
git switch master

Y nos va a decir que si queremos integrarlos cambios demos un git pull.



Solucionamos los problemas y hacemos un commit:  


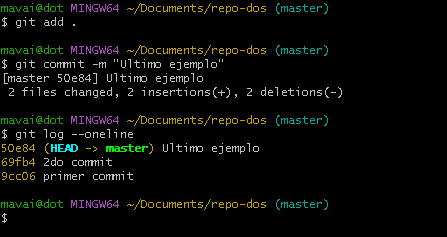
NOTA: HACER UN COMMIT NO ES LO MISMO QUE SINCRONIZAR LOS CAMBIOS.

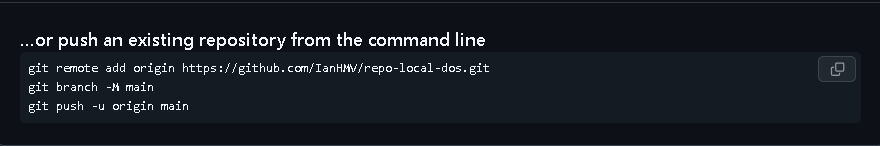
Y para sincronizar o mandar los cambios hacemos un git push:  


Migrar Repositorio Local:

Esto se puede dar ya habiendo hecho commits anteriormente en un proyecto, github solamente sirve para poder almacenar estos dichosos repositorios.

Un ejemplo si nosotros ya tenemos un proyecto con commits:



Ahora simplemente vamos a tener que ir a github y darle a “new” y crear nuestro nuevo repo, una vez hecho:  


git remote:

Es un comando que nos permite agregar repositorios remotos, cambiarles el nombre etc.

Para subir el comando tenemos que hacer en consola:  
git remote add origin <https://github.com/IanHMV/repo-local-dos.git>

En <origin> le podemos colocar cualquier nombre.

Despues colocamos:  
git branch -M main

**-M (move/rename)**  
Significa forzar el cambio de nombre.  
Si ya existe una rama llamada main, Git la sobrescribe con la rama actual (lo cual no permite con -m a menos que no haya conflicto).

Si tu proyecto inicia en la rama por defecto master:

git branch -M main

→ Cambia el nombre de master a main.

Diferencia entre -m y -M

git branch -m viejo nuevo → renombra, pero falla si ya existe nuevo.

git branch -M viejo nuevo → renombra, forzando aunque nuevo exista.

Y al final:

git push -u origin main

-u (o --set-upstream)

Esto es clave: le dice a Git que recuerde la relación entre tu rama local main y la rama remota origin/main.

(siempre cuando hagamos “push” se va a hacer directamente en origin main)

No es necesario cuando hacers un “git clone” dado a que ese proyecto ya esta vinculado a tu repositorio en la nube.

FORKS:

Diferencias entre un Fork y un git clone:

En git clone lo que se hace es desde github subirlo a nuestro repositorio local.

Y con fork lo que hacemos es subirlo desde github a NUESTRA cuenta de github.

Simplemente le damos al apartado de “Fork” en la parte superior derecha del repositorio en github:



Pull request:

Es descargar un repositorio (fork), hago una copia, le hago una mejora, y esa mejora queremos que el dueño la integre.

Investigar:

rebase y squach

branches strategies

git stach

Cherry pick

tags

milestones

submodulos

git hooks

git bisect

Ya una vez aprendido podemos trabajar con: git Flow o github flow