**SECCION 1 Y 2**

**20/5/2022 INICIO DEL CURSO**

|  |  |
| --- | --- |
| **COMANDOS** | **NOTAS** |
| git --version | Ver mi version de Git |
| git help, git help commit | Nos tira los comandos mas usados |
| clear | Limpiar consola |
| q | Para salir |
| git config --global user.name "CrisRmrz" &&  git config --global user.email "cris\_23-jun-2001@hotmail.com" | Establecer a git cual es mi nombre de usuario y mi email |
| git config –global -e | Nos muestra el nombre que pusimos y el email  De ahi podemos editar presionamos la letra ” a ” si quieremos. Luego para salir y guardar ESC DESPUES ” :wq! ” o SOLO SALIR CON :q! |
| git init | Inicializar repositorio dentro de mi proyecto, esto crea una carpeta oculta llamada .git |
| git status | Nos muestra las carpetas a las que no le estamos dando seguimiento, por lo que si queremos darles seguimiento debemos hacer un add |
| git add css por ejemplo &&  git add . | para darle seguimientos a la carpeta css && con el punto para añadir todo |
| git reset scss/style.css por ejemplo | Para quitarlo o no darle seguimiento, pasarlo de verde a rojo por decirlo asi. Y igual si queremos añadirlo lo hacemos con el add |

**NOTAS EXTRAS:**

Por ejemplo, en el comando “git -- version” **ese doble menos significa que viene una palabra completa**. En cambio, cuando es **solo un menos, significa que viene una abreviatura** como en “git commit –am".

En las consolas normales se limpia la consola con cls

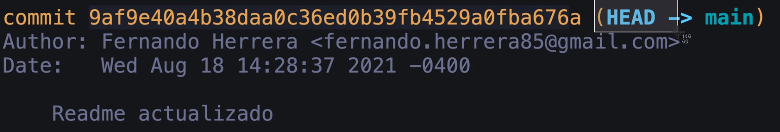
**COMANDOS DE COMMIT**

|  |  |
| --- | --- |
| **COMANDOS** | **NOTAS** |
| git commit –m “Primer commit” | Esto crea el primer commit y esa m significa el mensaje que ponemos al commit. LO NORMAL ES QUE DESPUES SI HACEMOS UN GIT STATUS NOS DIGA QUE NO HAY NADA PARA HACER COMMIT PORQUE ESTAMOS TRABAJANDO EN UN ARBOL LIMPIO |
| git log | Para ver la fecha del commit y su autor y la fecha y hora que se hizo el commit |
| git checkout -- . | ESTO ES MUY IMPORTANTE. ESTO RECUPERA EL ULTIMO COMMIT QUE HICE EN CASO DE QUE MI PROYECTO SE DESTRUYA O OCURRA ALGUN ERROR FATAL. Y RECORDAR QUE SOLO A LOS ARCHIVOS A LOS QUE LES DI SEGUIMIENTO CON EL GIT ADD |
| git branch | nos dice en que rama estamos |
| git branch –m master main | CON ESTO ESTAMOS **RENOMBRANDO** LA RAMA MASTER A MAIN |
| git config –global init.defaultBranch main | **ESTO SOLO SE HACE UNA VEZ, esto lo que hace es que cada que hagamos un nuevo proyecto SIEMPRE en vez de master se va a llamar main nuestro branch. Simplemente renombra de master a main** |
| git commit –am “Hice un commit y un add al mismo tiempo” | Para hacer el commit y el add al mismo tiempo. Si el archivo esta untrackted obvio no va a servir, primero tengo que darle seguimiento |
| Por ejemplo git add \*.html O SI SON ARCHIVOS ADENTRO DE OTROS ARCHIVOS SERIA COMO JS/\*.js | Esto añadiria al stage todos los archivos que terminen con .html |

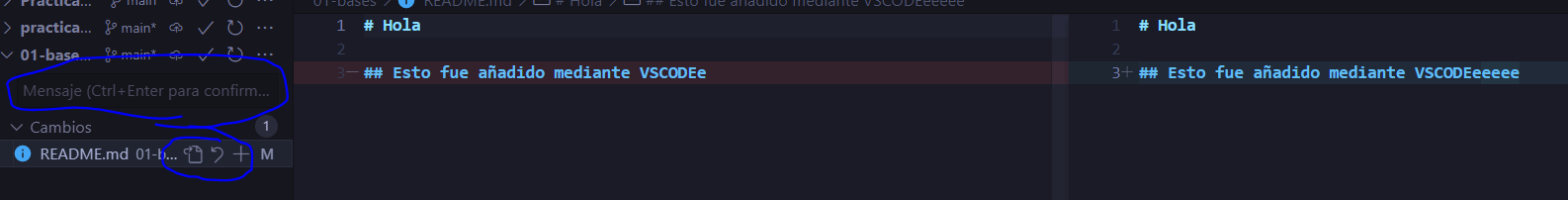
**NOTAS EXTRAS:**

**CUANDO SE CREA UN NUEVO ARCHIVO EN MI PROYECTO Y APARECE UNA “ U “ SIGNIFICA UNTRACKTED, O SEA QUE NO SE LE ESTA DANDO SEGUIMIENTO , POR ESO ES IMPORTANTE CADA QUE SE HACER UN NUEVO ARCHIVO HACERLE EL RESPECTIVO GIT ADD**

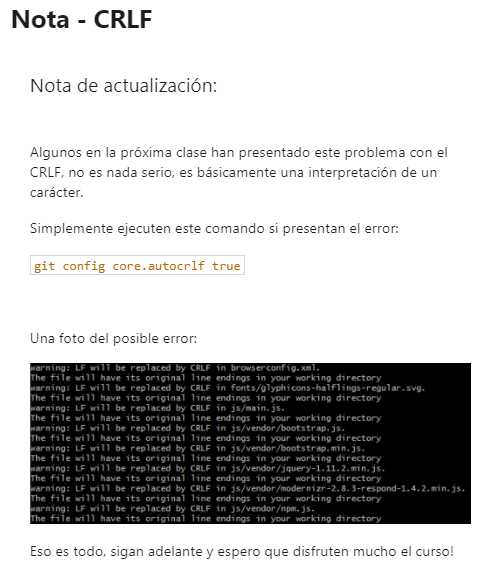
**Lo siguiente es el identificador del commit (** ESTO LO PUEDO VER CUANDO HAGO EL git log para ver los commits hechos **) :**



**CAMBIOS DESDE VSCODE**

**WARNING CRLF**

**git config core.autocrlf true**



**21/5/2022 Video #16 ( Diferentes formas de añadir al stage )**

|  |  |
| --- | --- |
| **git status --short** | **Nos dira una manera corta del git status** |
| **MANERAS DE CREAR ALIAS A NUESTROS COMANDOS, EJEMPLO:**  **git config –global alias.s “status --short”** | **LE ESTAMOS DICIENDO QUE DE AHORA EN ADELANTE CON SOLO DIGITAR “ git s “ sera como digitar git status –short**  **Si me equivoco puedo corregirlo en git config**  **--global –e, ya que ahi me aparecen los alias que hice y puedo modificarlos** |
| **git log --oneline** | **Es como hacer un git log pero nos tira la info de manera mas condensada** |
| **git log –online –decorate –all --graph** | **Lo musmo de arriba pero decorado de manera qe sea mas visible** |

DOS ALIAS QUE FERNANDO RECOMIENDA MUCHO PORQUE SON MUY LARGOS:

git config --global alias.lg "log --graph --abbrev-commit --decorate --format=format:'%C(bold blue)%h%C(reset) - %C(bold green)(%ar)%C(reset) %C(white)%s%C(reset) %C(dim white)- %an%C(reset)%C(bold yellow)%d%C(reset)' --all"

Entonces de ahora en adelante con solo git lg ejecuto todo eso de arriba

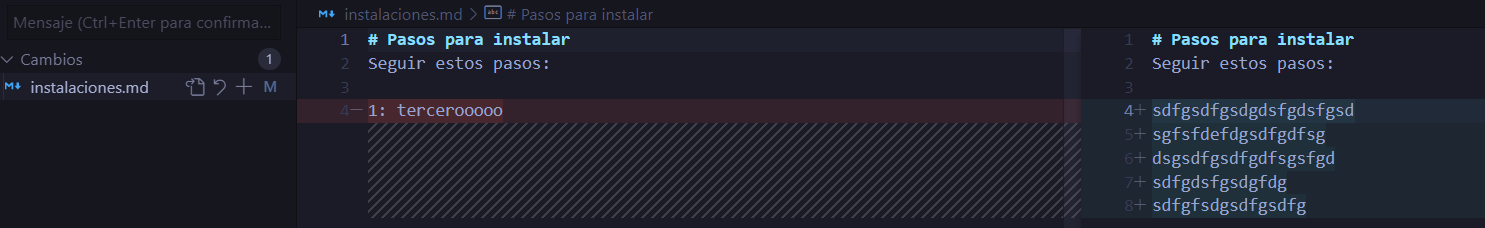
**SECCION 3**

**23/5/2022**

|  |  |
| --- | --- |
| **git diff** | **Si modificamos un archivo, esto nos indicara como estaba en el ultimo commit comparado a ahora. OJO, si todo sigue igual esto no nos tira nada. OJO, y si esta en el stage tampoco sirve. En resumidas cuentas esto compara los cambios con las cosas QUE NO ESTAN EN EL STAGE** |
| **git diff --staged** | **Este es lo contrario al anterior, nos muestra las modificaciones pero de las cosas que tenemos en el stage** |

**NOTA**

**Esto es mucho mejor hacerlo con la herramienta de VSCODE:**



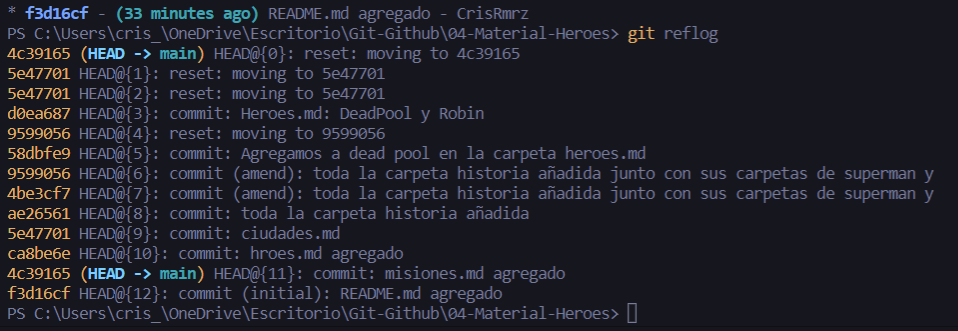
**24/5/2022**

|  |  |
| --- | --- |
| git commit –-amend –m “mensaje de commit actualizado” | Para enmendar o corregir el mensaje que puse en el ultimo commit |
| git reset –soft HEAD^  git reset –soft HEAD^2 | Para eliminar ese ultimo mensaje que corregimos y dejar el original. Para corregir lo que hicimos con el comando anterior.  ALT+94 para el ^ |
| git commit --amend | Nos tira en la consola para nosotros editarlo a mano:Despues ESC + :wq! para guardar o solo :q para salir y no guardar |

**Viajes en el tiempo**

|  |  |
| --- | --- |
| **git reset --soft 5e47701** | **Para volver y dejar todo como estaba hasta el index que le especifiquemos, ese numero raro es:** |
| **git reset –hard 5e47701** | **Este es destructivo, es destructivo pero igual en caso de que fuera un error lo puedo recuperar** |
| **git reflog** | **Para recuperar todo, esto muestra todo lo que se a hecho en orden cronologico:** |

**Foto del git reflog:**

**Entonces de esa manera podemos recuperar CUALQUIER COSA**

**Simplemente hariamos un git reset –hard a cualquier lugar del tiempo en el que queramos regresar**

|  |  |
| --- | --- |
| **git mv destruir-mundo.md salvar-mundo.md** | **Renombramos el nombre del archivo, aparte recordar hacer el add y commit para guardar del todo** |
| **git rm nombrearchivo.md** | **eliminar archivo. Pero esto igual queda en el stage. Entonces para eliminarlo del todo despues solo hacemos commit** |

**IGNORAR ARCHIVOS**

**Creando el .gitignore y ahi metiendo archivos**

**AL .gitignore SI HAY QUE DARLE SEGUIMIENTO**

**SECCION 4: RAMAS**

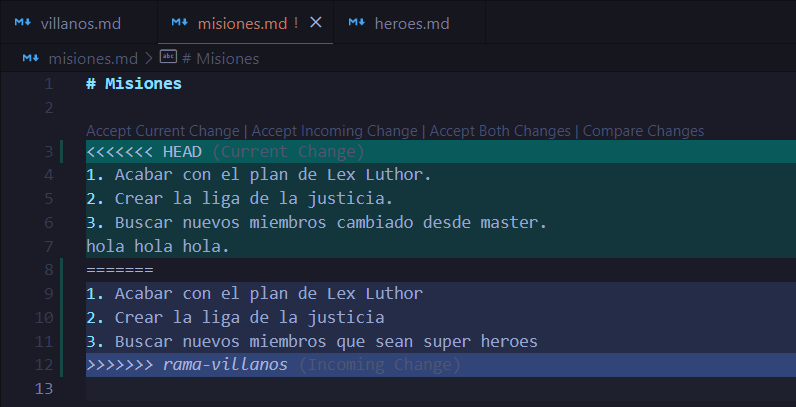
**Hay 3 tipos de merge( uniones ) :**

1. **Fast-Forward: Git logró establecer todos los cambios sin ningun conflicto**
2. **Union automatica**
3. **Union con conflictos**

|  |  |
| --- | --- |
| **git branch nombreDeLaRama** | **Crear nueva rama** |
| **git checkout nombreDeLaRama** | **Mover a la nueva rama** |
| **git branch** | **Ver en cual rama estoy y cuales ramas existen** |
| **git merge nombreDeLaRama** | **Agregar la rama a nuestro master ( a nuestra rama principal )** |
| **git branch –d nombreDeLaRama** | **eliminar nuestra rama** |
| **git branch –d nombreDeLaRama -f** | **eliminar nuestra rama de una manera forzada, esto pasa solo si tenemos archivos que no hemos añadido a ninguna rama** |
| **git checkout –b nombreDeLaRama** | **creamos la rama y ademas nos movermos a ella de una vez** |

**Esto ocurre en un conflicto:**

**Por cierto, los archivos que no dan conflictos si van a pasar sin problemas**

**Despues de arreglar el problema se debe hacer el commit**

**TAGS**

**Los tags son las versiones de nuestra aplicacion, por ejemplo la version 2.1.1.1**

|  |  |
| --- | --- |
| **git tag nombreDelTag** | **Poner un tag pero solo al lugar donde estamos** |
| **git tag –a v1.0.0 -m “Version 1.0.0 lista”** | **Esta es igual que la de arriba pero mejor** |
| **git tag** | **ver todos los tags** |
| **git tag –d nombreDelTag** | **borrar el tag** |
| **git tag -a v0.1.0 4177ffa -m "Version aplha lista"** | **4177ffa es el hash, esto para agregar un tag a una linea del tiempo anterior. Ver linea del tiempo con git lg para ver los hash** |
| **git tag -n** | **ver los mensajes completos de nuestros tags** |
| **git show v1.1.1** | **Esto para mostrar toda la info del tag** |

**25/5/2022**

**SECCION 5 Git stash y Git rebase para cambios de emergencia**

**Git stash es una bobeda, en caso de emergencia por si tenemos que darle al jefe la aplicacion con el ultimo commit, para no perder los cambios con los que estabamos trabajando y sus modificaciones, movemos todo al stash, aqui se almacena hasta los archivos a los que no les estabamos dando seguimiento**

**Al hacer el stash volveriamos al ultimo commit que teniamos y las modificaciones que estabamos haciendo desaparecen de pantalla pero no desaparecen del todo, solo se almacenan en el stash para luego recuperar todo**

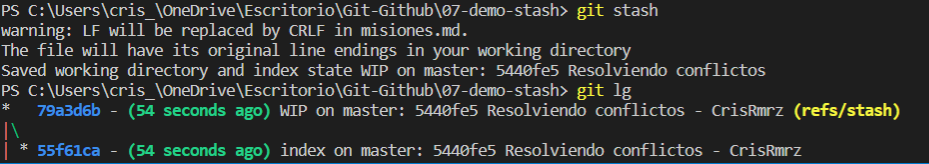
**En resumidas cuentas el stash esta hecho para meter todos los cambios y modificaciones a los que no les hemos hecho commit**

**\*\*RECOMENDACION\*\***

**Cada modificacio nque hacemos la movemos al stash y ya luego la integramos. NO hacer un monton de modificaciones y meterlas al stash porque luego puede dar problemas integrarlas**

**Git rebase : Sirve para unir commits, separar commits. Es una tecnica**

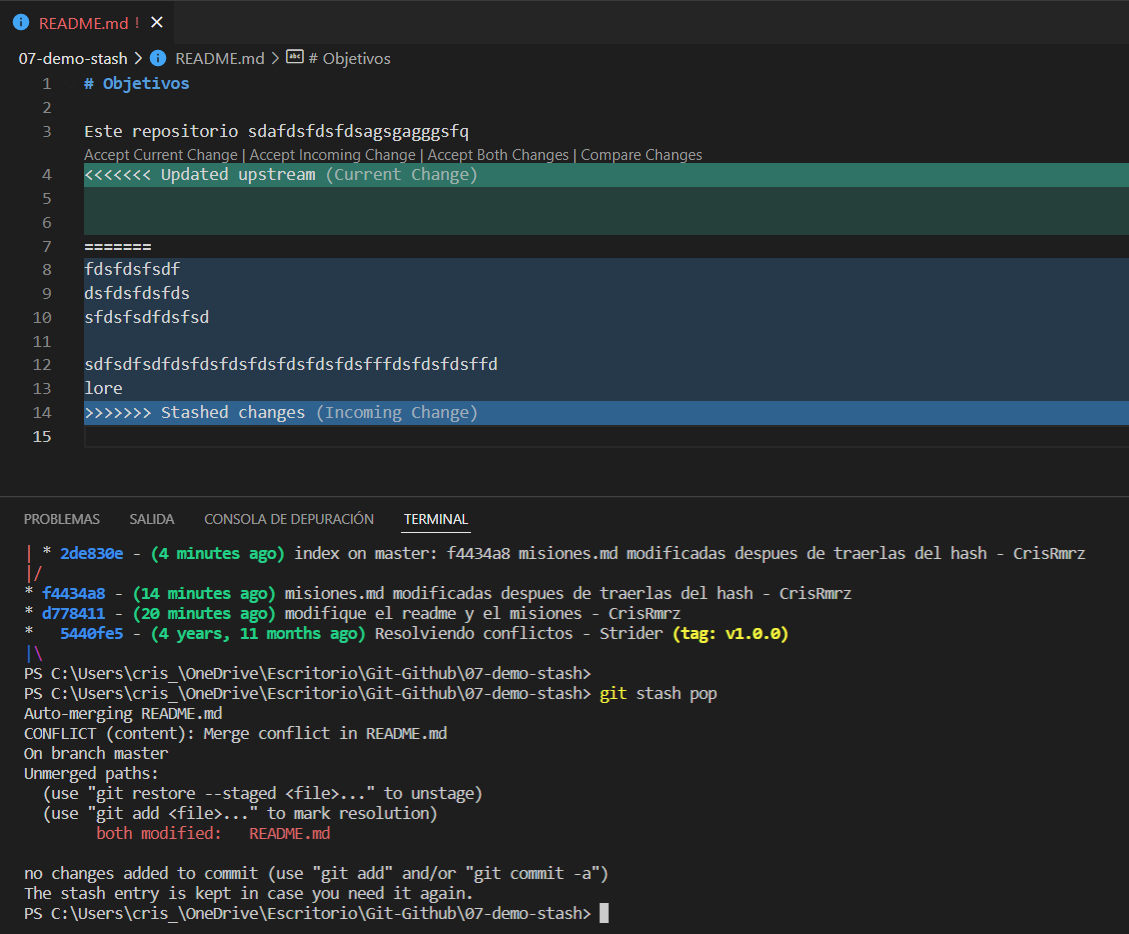
|  |  |
| --- | --- |
| **git stash** | **creamos el stash** |

**Se quitan todas las modificaciones quee estabamos haciendo pero se guardan en el stash, se guardan en ese hash que nos dio git**

|  |  |
| --- | --- |
| **git stash list** | **Nos da la lista de los stash que tenemos junto con su respectivo hash** |
| **git stash pop** | **Recuperamos esas modificaciones y archivos que tenemos en el stash y ADEMAS CONSERVAMOS LOS CAMBIOS QUE TENEMOS HECHOS**  **OJO cuando recuperamos los archivos haciendo el git stash pop, el stash se borra, nos dice Dropped refs/stash@{0}** |

**Conflictos con el stash**

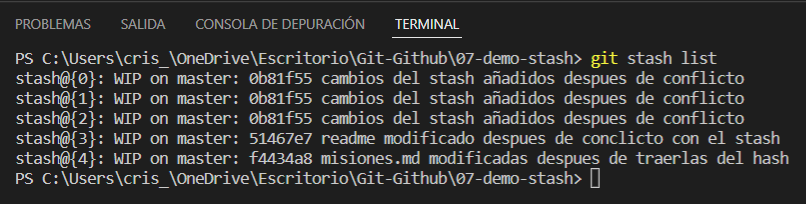
**Esto pasa por ejemplo cuando modificamos un archivo, llamamos al stash y el stash detecta que hay dos lineas que chocan. entonces nos pone a decidir en como deberian estar esas lineas como en la siguiente imagen:**



**A veces nos puede quedar el stash despues de hacer el pop, para resolver esto digitamos:**

|  |  |
| --- | --- |
| **git stash clear** | **OJO, esto borra TODOS los stash que tenemos. De todas maneras si esto fue un error simplemente hacemos un viaje en el tiempo con reflog, viendo el hash donde queremos ir y con git reset –hard yElHash** |

**Si hacemos varios stash tenemos algo asi en la lista**

**El problema seria que ahi no me dice que cambios hice en cada hash**

**Pero nos tira los numeros para identificarlos, el ultimo hash que hicimos siempre va a ser el numero 0 o stash@{0}**

**Si hacemos el pop agarra ese ultimo hash, o sea el 0**

**Pero si queremos ver un hash en especifico es con git stash apply “stash@{yElNumeroQueLeIndiquemos}”**

|  |  |
| --- | --- |
| **git stash apply "stash@{2}"** | **Recordar ponerlo entre comillas, sino no sirve, eso es por una actualizacion. Esto no borra el stash como con el pop** |
| **git stash drop "stash@{0}"** | **Borrar un stash en especifico** |

**LA MEJOR MANERA DE HACER UN STASH ES LA SIGUIENTE:**

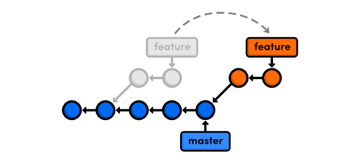
|  |  |
| --- | --- |
| **git stash save "MENSAJE EN ESPECIFICO PARA IDENTIFICAR EL STASH"** | **Este es mejor que el git stash** |
| **git stash show** | **mostrar cuantas lineas de agregaron o eliminaron** |
| **git stash list --stat** | **La mejor manera de mostrar los stash ya que muestran los mensajes que pusimos** |

**Recomendacion**

**Stash que hacemos, stash que eliminamos**

**Porque tener una lista de stash no es buena practica y despues se producen conflictos**

**REBASE**

**Coge los cambios que tenemos en una rama aparte y ponemos esos cambios en la master pero desde el puro final del tiempo( o sea lo mas actual)**

|  |  |
| --- | --- |
| **git merge rama-misiones-completadas** | **esto estando en el master para traer todo lo de esa rama** |
| **git rebase -i HEAD~4** | **ese simbolo se pone ctrl+126** |

**Tipos quash, reword y edit. VER VIDEOS DE UDEMY PARA RECORDAR COMO SE USAN**