CONFIGURACION GLOBAL

1. Git config –global -e

Commad to user git:

Examples of command to know the sintaxis of each command

1. Git help
2. Git help Commit

Command to config git (when we execute the command to be success it must not response to be ok)

1. Git config --global user.name “Luis Vargas”
2. Git config --global user.email “luge.vargas14@gmail.com”

Command to see the configurations that you have been configurated

1. Git config --global -e

**Key “a” to insert o update the configuration then “:wq!” to commit the update**

**COMMAND TO INITIAL THE FIST REPOSITORY**

1. Cd C:\Cursos-Git\01-bases

**<C:\Cursos-Git\01-bases>git init = create local repository (.git (folder))**

*New way to create a repository*

*“git config –global init.defaultBranch “main””*

*Use this command when use init for the first time else*

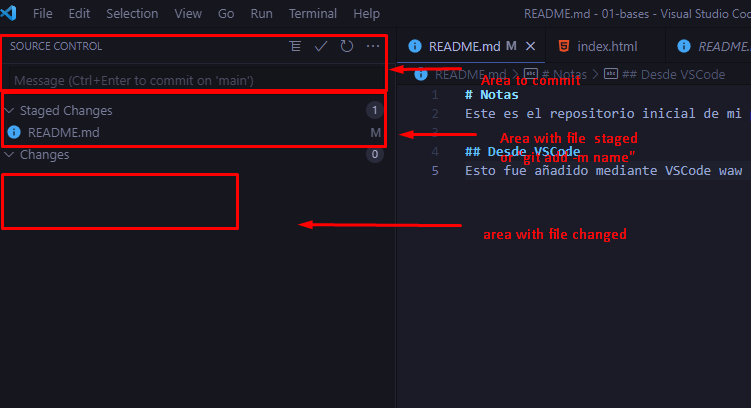
*Use git Branch -m master main*

1. Git status = to see the file that are ready to commit
2. Git add “name”= to add the file to be ready to commit
3. Git add . = to add all files that are ready to commit
4. Git reset “filename” = to remove from the list that are ready to commit
5. Git commit -m (or --message) “message” (take state of the repository)

COMMAND TO RECOVERY THE REPOSITORY

1. Git checkout -- . = recovery the repository at the last commit

USE VSCODE TO CHANGE, TO STOGED AND COMMIT



**Different way to add files and folders**

1. Git add fileName
2. Git add \*.extensionName or .extensionName(when the file doesnt have name)
3. Git add folderName/fileName or \*. extensionName or or .extensionName(when the file doesnt have name)
4. Git add folderName/ (to add everything in the folder)
5. Git add . = to add all files that are ready to commit
6. Through visual studio code

**Make Alias to own commands**

1. **Git status –short = show short piece of resumen**
2. **Git config --global alias.s** “status --short” = to créate alias
3. **Git config –global alias.l “log –oneline”**

**Change and edit files**

1. Git diff
2. Better option the VSCode to compare line before it send to the stage

**Update commit and Remove**

1)git commit -am “Instalaciones Actualiz”

2) git commit –amend -m “Instalaciones Actualizadas”

3) git reset –-soft HEAD^”index of commit” = like 1,2,3 surfing throgh the commit index to make the change or update of the commit name

**Travels throgh the time**

1) git reset --soft d2cf669 = te manda a este punto pero realmente el que editas es el siguiente, te posiciona en el lugar donde estaba sin modificaciones (NO MODIFICA EL HEAD)

2) git reset –mixed be0c63b = te manda a este punto pero realmente el que editas es el siguiente, te posiciona en el lugar donde mezcla todo y vuelves a comenzar (NO MODIFICA EL HEAD)

3) git reset -hard “hash” = te manda a este punto pero destruye lo que está adelante (ósea manda el HEAD AL main al punto solicitado)

4) git reflog = devuelve todo el historial de los eventos realizados tales como commit, reset, etc

**RENAME FILE and Delete AND .GITIGNORE**

1. **Git mv “filename” “newfilename” = Renombra un archivo**
2. **Git rm “filename” = borra archivo en especifico**
3. **git rm -r “one-of-the-directories/” = Borra los archivos dentro de la carpeta**
4. **git rm -r --cached myFolder**
5. **git checkout “hash” “fileName” = para regresar al commit previo al cambio pero solo de un archivo**
6. **.GITIGNORE**

**Archivos modo de ignore**

**- dist/**

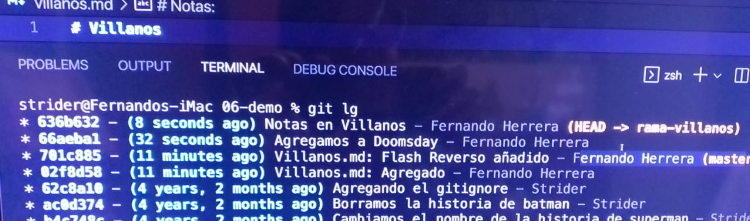
**- node\_modules/**

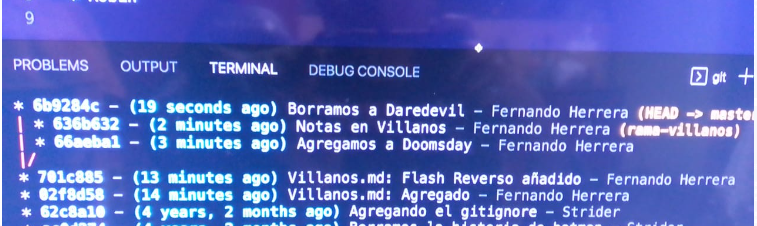
**- \*.log**

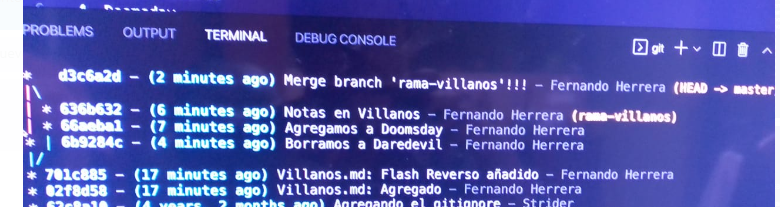
**RAMAS , UNIONES CONFLICTOS Y TAGS**

1. **Git Branch “BranchName” = to créate a new branch**
2. **Git checkout “BranchName” = to move throght Branch**
3. **Git merge “BranchName” = to join the branches**
4. **Git Branch -d “BranchName” = to delete the Branch to forcé the sentence use “-f” at the end of the sentence**
5. **Git checkout -b “ramaName” = para crear y posicionarse en la rama creada (LA MAS USADA QUE LA 3)**
6. **Git merge “ramaName” = hay que posicionarse en la rama receptora y ref a la emisora**

**GRAFICOS DE RAMAS CON MODIFICACIONES**

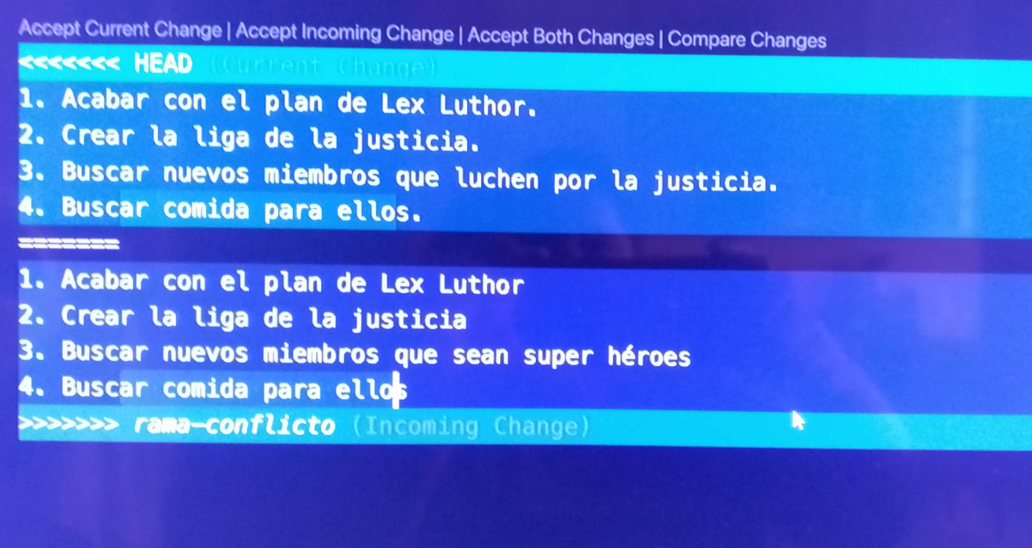
**RAMA CON UNA SOLA MODIFICACION “RAMA VILLANOS”**

**RAMA CON DIFERENTE MODIFICACIONES CADA RAMA CONSTA CON 1 O MÁS MODIFICACIONES , RAMA-VILLANOS – RAMA MASTER**

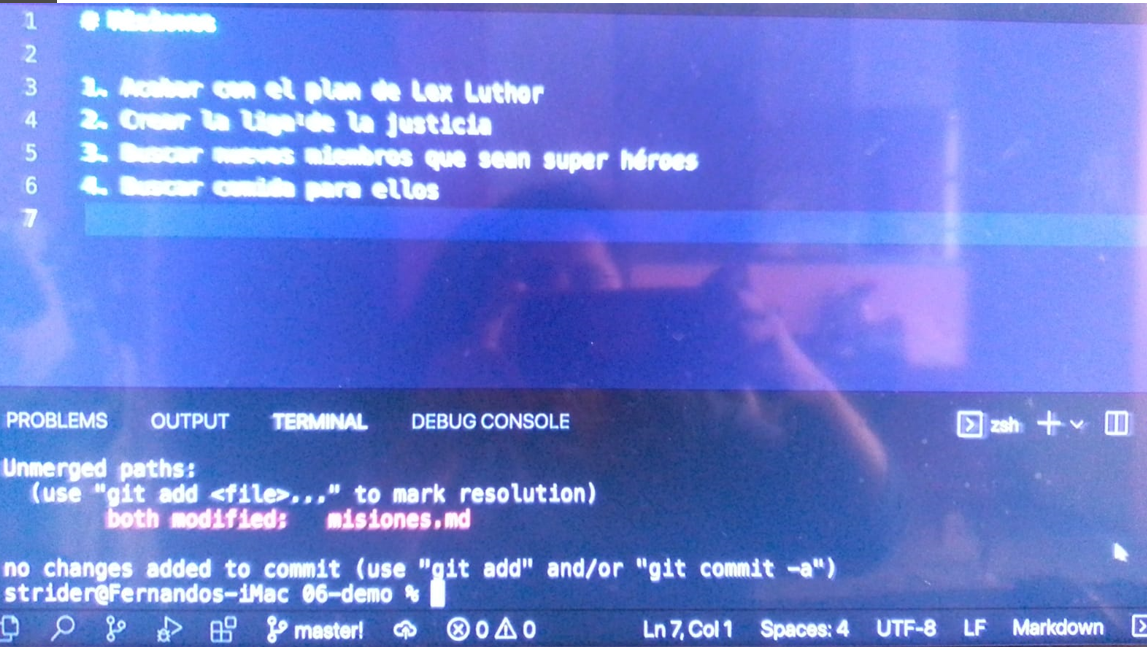
**RAMA UNIFICADA**

**MERGE**

**CONFLICTO AL HACER MERGE**



**RESOLUCION DE CONFLICTOS**



**TAGS**

1. **Git tag “tagName” = to créate tag**
2. **Git tag -d “tagName” = delete a tag**
3. **Git tag -a(anotation)”VersionName” ”Hash(optional)” -m(message) “Message”**

**git tag -a v0.1.0 4e809d4 -m"Version Alfa de tag"**

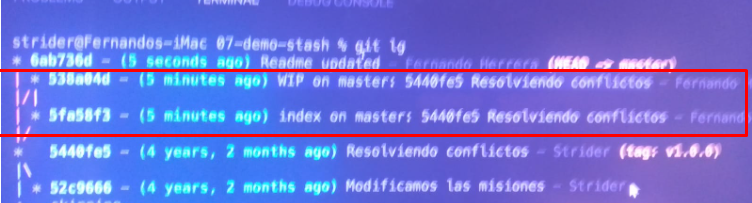
1. **Git show “tagName” (-a (anotate))**
2. **Git push –tags = to update tag in the origin branch (ósea al git hub para subir los tags)**

**GIT STASH Y GIT REBASE PARA REALIZAR CAMBIOS DE EMERGENCIA**

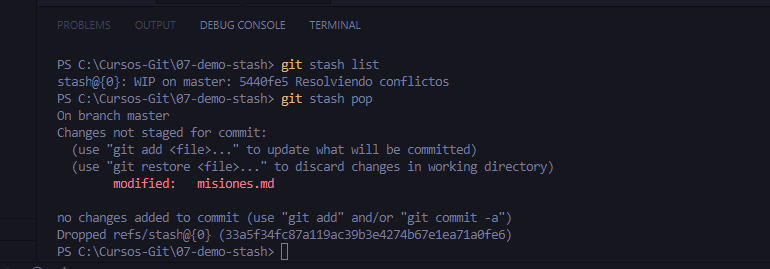
**GIT STASH**

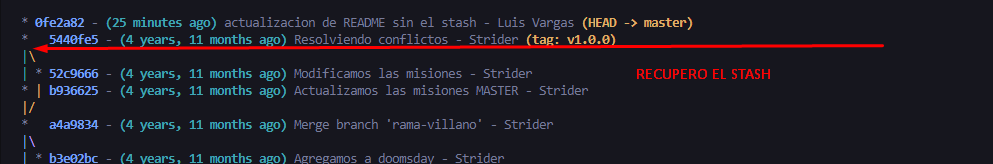
1. **Git stash = to sabe the change after the the last commit**

**Y DESPUES DE UN COMMIT DE README**

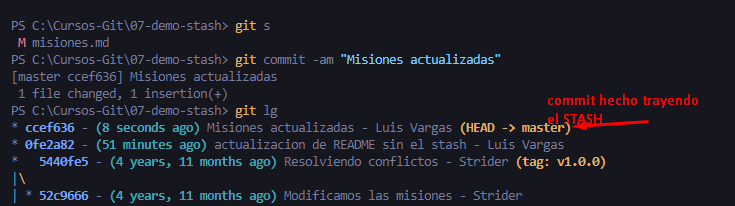


1. **Git stash list = para ver cuando stash tenemos**
2. **Git stash pop = esto saca los cambios del stash y los regresa junto con lo avanzado**





1. **EL STASH DESPUES DEL COMMIT**



1. **Git stash apply “hashNamedeStash ejem stash@{3}” = para ver el detalle del stash es poco usado pero puede ver el detalle**
2. **Git stash drop “stash@{3}”**

**Solo te permite borrar el ultimo stash no se puede con los anteriores**

**No olvidar las comillas**

1. **Git stash show “stash@{3}” = muestra el detalle de los stash No olvidar las comillas**
2. **Git stash save “agregamos a loki”**
3. **Git stash list – stat = para ver el detalle de cada stash**
4. **Git stash clear = para limpiar todos los stashs**

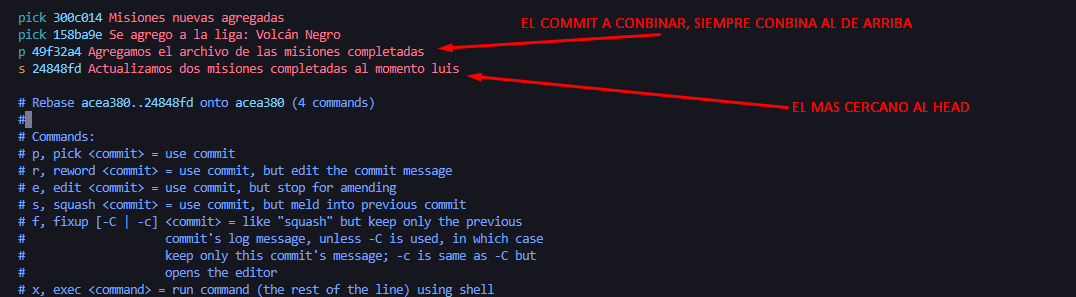
**REBASE**

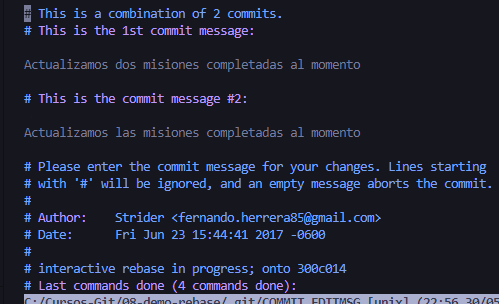
1. **Git rebase “RanaName” = pasa todos los commit a la rama en la que estamos posicionado recibimos los commit de “RanaName”**

**REBASE SQUASH (solo desaparece el nombre y actualiza el evento NO HAY COMBINACION del commit pero el commit realizado desaparece (el nombre o mensaje))**

1. **Git rebase -i HEAD~4 = este comando trae los 4 commit del HEAD y los manipula en este caso usamos el SQUASH**

**Los ordena al revés los commit de arriba**



**NO HACER CAMBIOS SI LOS COMMIT FUERON HECHOS EN OTRA COMPUTADORA, SI LOS COMMIT SON DE LA MISMA COMPUTADORA NO GENERARÁ ERRORES**

**REBASE REWORD**

1. **Git rebase -i HEAD~4 = este comando trae los 4 commit del HEAD y los manipula en este caso usamos el REWORD**

**REBASE EDIT**

1. **Git rebase -i HEAD~4 = este comando trae los 4 commit del HEAD y los manipula en este caso usamos el EDIT**

**CASO:**

**Queremos deshacer lo guardado, ojo antes del stage usamos:**

1. **Git checkout –- “FileName”**

**CASO: SEPARAR COMMITS**

**Se realizó un commit que no queremos y queremos deshacer el nombre del commit y además hacer los commit por separado**

1. **Git rebase -i HEAD~3 = entrar al interactivo >> editamos los commit que queremos y después usar**

**Las 2 opciones para terminar el proceso porque aún tu estado está en PROCESS**

**GIT COMMIT –AMEND**

**GIT REBASE –CONTINUE**

**PARA DESHACER LOS STAGE USAMOS EL SIGUIENTE COMMAND**

1. **Git reset HEAD^**

**PARA TERMINAR USAMOS EL SIGUIENTE COMANDO**

1. **Git rebase –CONTINUE**

**EN CASO NO APUNTE AL MASTER**

1. **Git checkout master**

**HACER UN PULL DEL UPSTREAM**

1. **Git remote add upstream “URL” = para setear la url del upstram**
2. **Git pull upstream “BranchName” o master**

**PARA AGREGAR UNA NUEVA RAMA AL GIT HUB**

1. **git push --set-upstream origin "ramaName"**

**PARA TRAER RAMAS DESD GIT HUB PENDIENTES DE MERGE**

1. **git pull –all**

**PARA VER LAS RAMAS ESCONDIDAS**

1. **git Branch -a**

**PARA POSICIONARSE EN LA RAMA ESCONDIDA**

1. **git checkout “ramaNameEscondida”**

**PARA BORRAR RAMA ESCONDIDA EN LOCAL Y GIT HUB**

1. **git push origin :"ramaName"**

**PARA BORRAR RAMAS ESCONDIDAS**

1. **git remote prune origin**

**PARA RECUPERAR UNA RAMA BORRADA DE GIT HUB PODEMOS IR A TAG Y DESPUES HACER UN CHECKOUT -B PARA crear la rama y POSICIONAR EL HEAD EN EL TAG Y DESPUES HACER UN PUSH A GIT HUB**



1. **git checkout v1.0.0**
2. **git checkout -b rama-kitkat**
3. **git push**

**PARA CERRAR UN ISSUE DESDE GIT LOCAL**

1. **git commit -am "Fixes #5: Hecho"**

**PARA DESPLEGAR UNA WEB DESDE EL LOCAL DE GIT**

**-PRIMERO CREAMOS NUESTRA CARPETA DOCS DESPUES CREAMOS LOS ARCHIVOS PARA LA WEB Y POR ULTIMO LO SUBIMOS Y EN LA SUBIDA NOS**

**VAMOS A CONFIGURACION => PAGE=> Y DESPLEGAMOS LA CARPETA DOCS**