COMANDOS DE GIT

1- CONFIGURACION DE NOMBRE Y MAIL:

git config --global user.name "Fabricio"

git config --global user.email "estudiodellsanta@gmail.com"

Para chequear o editar los cambios: git config --global -e

2- AÑADIR ARCHIVOS O CARPETAS A REPOSITORIO

git add . Añade todo

git add \*.extension o nombre del archivo

git add <nombre de la carpeta>/\*.extension

git add <nombre de la carpeta>/ añade todos los archivos de la carpteta

git add -u (Agrega lo updated)

git add --all (Agrega todos los arhivos que se hayan modificado)

git reset <nombre del archivo> desagrega un archivo añadido

git commit –m <”mensaje”> Toma la fotografía a almacenar, esta es la forma abreviada de git commit –-message <”mensaje”> que cumple la misma función. El –- es seguido de una palabra completa el – es seguido por una abreviatura. Ejemplo –am es la abreviatura de add messaje.

Si el archivo ya tiene seguimiento se puede utilizar la forma abreviada git commit -am "Comentario" El formato -am incluye la adicion mas el commit en un solo paso

2.1 Git no sigue carpetas vacias

Para que siga la carpeta dentro de ella se añade el archivo .gitkeep se añade mediante git add <nombre de la carpeta>/.gitkeep

3- ELIMINAR ARCHIVOS AÑADIDOS

git reset <nombre del archivo> (funciona si el achivo esta siendo seguido)

4- INCORPORAR COMMITS

git commit -m "Comentario"

4.1 Corregir commits

git commit --amend -m "<mensaje>" Modifica el mensaje del ultimo commit

4.2 Unificar commits git reset --soft HEAD^ (^ Apunta al ultimo commit antes del HEAD

Tambien se puede agregar HEAD^1,2,3,n segun cuantos commits antes del HEAD se quiera llegar)

Tambien se puede poner el hash del commit

4.3 git reset --mixed <hash> (Saca todo del stage sin eliminar y los cambios quedan listos para que se puedan añadir)

4.4 git reset --hard <hash> (Saca todo del stage y elimina no definitivamente todo lo que esta sobre ese hash)

4.5 git reflog (va al historial de todas acciones ejecutadas y se puede recuperar todo)

4.6 git reset --hard <hash> (restaura todo hasta el hash que se coloque)

**Nota: error CRLF**

**Tipear el siguiente comando: git config core.autocrlf true**

5- RECUPERAR PROYECTO A COMMIT ANTERIOR

git checkout -- .

6- SABER EN QUE RAMA ESTOY

git branch

6.1- CAMBIAR NOMBRE RAMA MASTER A MAIN

git branch -m master main

git config --global init.defaultBranch main (Para cambio global)

7- LISTADO DE SUCESOS EN EL PROYECTO

git log

git lg

8- RAMAS

Para saber en que rama estoy: git branch

8.1 MERGE git merge une las ramas a la principal o a la que se indique de tres formas distintas

8.1.1 Fast forward (git intenta hacer la union automaticamente cuando no hay confilictos de por medio esta opcion se puede activar o desactivar)

8.1.2 Uniones automaticas (cuando existe alguna diferencia entre las ramas pero no son incompatibles ni se superponen por lo cual no hay conflicto que evite la union)

8.1.3 Union manual (git no lo puede resolver de manera automantica porque hay conflicto o superposiciones en tal caso se hace un commit nuevo, el merge commit)

8.2 CREAR RAMAS

git branch <nombre de la rama>

8.3 MOVERSE ENTRE RAMAS

git checkout <nombre de la rama a la que me quiero mover>

8.4 UNIR RAMAS

por ejemplo una rama crada al rama main)

Primero me posiciono sobre la rama que va a recibir lo realizado en otra rama git merge <nombre de la rama> (en principio si no hay inconvenientes git ejecuta un ffw

8.5 ELIMINAR RAMAS

git branch -d <nombre de la rama> (Si eventualmente algo no se unio git pide confirmación antes de borrar la rama) En tal caso se ejecuta el comando git branch -d <nombre de la rama> -f (para forzar el borrado

8.6 CREAR RAMA Y MOVERSE A LA MISMA CON UN SOLO COMANDO

git checkout -b <nombre de la rama>

8.7 COMANDO RAPIDO PARA VER EN QUE RAMA ESTOY

git s -b

8.8 TRAER EVENTUALES RAMAS CREADAS POR OTROS A MI REPO LOCALES

git pull --all

8.9 VER OTRAS RAMAS CREADAS EN GITHUB

git branch --all o abreviado git branch -a

8.10 LIMPIAR RAMAS EN DESUSO

Una vez borradas las ramas en desuso en github, no se borran en el editor de texto local y es necesario borrarla manualmente para que no quede basura el comando seria:

git push - origin :<nombre de la rama> pero no se borran porque no existen mas en el repositorio remoto (si lo haria si existieran)

El comando correcto es:

git remote prune origin (esto revisa las rama que ya no existen en el remoto y las elimina)

Al ejecutar nuevamente git branch o git branch -a lo comprobamos

9- ABREVIATURAS (ALIAS)

Ejemplo: git status --short se genera una abreviatura mediante el siguiente comando

git config --global alias.s "status --short"

Para verificar o editar el alias generado git config --global -e

Otro ejemplo para git log abreviando a git lg

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"

Se puede realizar cualquier tipo de abreviatura con diversos parametros que se pueden encontrar en la documentacion de git

10- GIT DIFF

El git diff me muestra los cambios en el archivo aunque no lo haya subido

al stage ni le haya hecho el commitPara ver los cambios con lo que ya esta en el staged el comando es:

git diff --staged

11- CAMBIAR NOMBRE A ARCHIVO SIN PEDER DEL HISTORIAL LO HECHO CON EL NOMBRE ANTERIOR

git mv <nombre del archivo a modificar completo incluyendo la extension> <nombre del archivo nuevo>

Nota: mover al mismo directorio implica cambiar el nombre. Aparce señalado con la letra R de rename

Queda directamente en el stage y para confirmar el cambio se hace el commit

12- PARA ELIMINAR UN ARCHIVO: git rm <nombre del archivo>

Para confirmar la eliminacion que aparece identificada con D de deleted, para confirmar la eliminacion hay que hacer el commit, caso contrario con git reset --hard se recupera.

El git reset --hard es muy similar el git checkout -- .

13- NO DARLE SEGUIMIENTO A DETERMINADOS ARCHIVOS

En la raiz se agrega el archivo .gitignore y dentro del archivo se detallan las carpetas o

archivos que no hay que seguir.

Como todo archivo el .gitignore se agrega y se hace el commit

14- TAGS

14.1 CREAR UN TAG

git tag <nombre del tag> (Crea un tag en el ultimo commit)

14.2 VER LOS TAGS CREADOS

git tag

Para ver mas info de algun tag en particular:

git show <nombre del tag> (ejemplo v1.0.0)

14.3 BORRAR UN TAG

git tag -d <nombre del tag>

14.4 git tag -a <version ejemplo v1.0.0> -m "<mensaje>" (lo coloca en el ultimo commit o donde este parado en ese momento)

v1.0.0 el 1 version, el primer 0 alguna funcionalidad añadida sin que sea una nueva versión el segundo 0 correccion de algun error

14.5 Añadir un tag en un commit que no sea el ultimo:

git tag -a v0.1.0 <hash> -m "<mensaje>"

15- STASH (no es necesario si se trabaja con ramas)

(Pone una rama en suspenso mientras se trabaja en otra cosa incluyendo archivos a los que todavia no se les da seguimiento)

15.1 git stash (con los cambios guardados pero no añadidos ni commiteados) Guarda todo el working directory. Guarda todo lo hecho posterior al utlimo commit

15.2 VER LOS STASH

git stash list

15.3 RECUPERAR EL ULTIMO STASH

git stash pop

15.4 BORRAR UN STASH YA UTILIZADO

git stash clear (Borra todos los stash) Por supuesto que se pueden recuperar con reflog

git stash drop <identificador> (si no se identifica borra el cero el primero de la lista)

15.5 RECUPERAR UN STASH EN PARTICULAR CUANDO HAY VARIOS

git stash apply y el identificador del stash (por ejemplo satsh@{2}

15.6 VER INFO DE ALGUN STASH EN PARTICULAR

git stash show <identificador>

15.7 SALVAR UN STASH CON NOMBRE

git stash save "<mensaje>"

15.8 VER INFO DETALLADA DE LOS STASH

git stash list --stat

16- REBASE (no modificar archivos ya pusheados pq se pueden generar conflictos con alguien que descargo el archivo)

Cuando se generan archivos paralelos en ramas el rebase los ordena, corrige mensajes de los commits, une commits y separa commits

16.0 UTILIDADES: Ordenar commits, corregir mensajes de commits, Unir commits, Separar commits

16.1 REBASE INTERACTIVO

git rebase -i HEAD~3 (El 3 refiere al numero de commits y el HEAD puede reemplazarse por un hash)

16.2

git rebase master (me posiciono sobre otra rama y traigo informacion del master) luego me cambio a la rama main y hago un merge de la rama y se ejecuta un ffw, luego borro la rama

16.3 GIT SQUASH (fusiona)

Une commits similares

git rebase -i HEAD~4 (Entra en modo de edicion)

Ahi se puede aplicar el squash. Fusiona el seleccionado con el inmediato superior

16.4 Para modificar mensaje de commits se aplica el reword (pueden ser varios(wq!) y ahi entra en la edicion

Se presiona a para comenzar la edicion y se modifica el texto. Luego wq! y entra a la siguiente edicion y asi hasta que se modifican

todos los que se habian seleccionado para editar.

16-5 CONCLUIR UN REBASE

git rebase --continue (para terminar el rebase interiactivo)

17- GITHUB REMOTE PUSH AND PULL

17.1 Desde el repositorio local agregar un origen remoto

git remote add origin <url>

17.2 Para saber los remotos que tenemos en el repositorio

git remote -v

17.3 Para subir a una rama determinada, el -u indica una rama por defecto luego cada push sube ahi sin necesidad de especificar

git push -u origin master (solo la primera vez luego solo git push)

18- TAGS

18.1 Para consultar los tags: git tag

18.2 Para subir tags: git push --tags

19- CONFIGURACIONES GLOBALES Y LOCALES

19.1 PULL

git config --global pull.ff only

19.2 VERIFICAR LA CONFIGURACION ACTIVA

git config --global -e

19.3 Si eventualmente hay conflicto con el ffw (se configura localmente)

git config pull.rebase true (al hacer el pull muestra el conflicto a resolver)

20- CLONAR UN REPOSITORIO

20.1 Situarse en la carpeta donde debe caer el clon (cd <carpeta>

git clone <url> (se puede clonar todo o alguna rama en particular

21- FETCH

21.1 git fetch: detalla los cambios hechos antes de ejecutar un pull por ejemplo y actualiza mi detalle local (lo veo con un git lg) de lo que todavia no se realizo un pull

22- FORK

22.1 Se presiona el boton de fork en github que trae un repositorio ajeno a mi cuenta.

Luego se clona el repositorio (git clone) y se puede trabajar en el.

23- AGREGAR UN REPOSITORIO DEL QUE HAGO FORK

Si es un repositorio del que solo traigo informacion generalmente se le llama UPSTREAM

git remote -v: me muestra lo existente.

El pull por defecto se hace desde el repo remoto nuestro, para hacerlo desde el upstream se lo debe agragar como remoto

23.1 PARA GENERAR UNO NUEVO

git remote add upstream <url del repo origen> (se copia la url igual que cuando se colona)

23.2 En este caso para hacer el pull

se puede previamente hacer un git fetch upstream para chequear cuales es el estado de las actualizaciones

git pull upstream master (se indica la rama de la que se quiere hacer el pull).

Luego aparece una pantalla similar a la de conflictos que nos deja en medio de un rebase para lo cual se hace el procedimiento del caso).

23.3 En el caso de crear una rama nueva desde la consola y empezar a trabajar en ella al hacer push desde esa rama falla. Se debe ejecutar el comando git push --set-upstream origin rama-villanos

24- PARA REVISAR EL TRABAJO DE OTROS APORTANTES

git fetch

git branch -a

git checkout <rama x> (pasar a la rama de otro aportante)

24.1 CUALQUIER APORTANTE PUEDE MERGEAR A MASTER PARA ACTUALIZAR EL REPOSITORIO

git checkout master (se pasa a la rama que recibe)

git merge <rama x>

git push

24.2 OTRA FORMA MEJOR ES CON PULL REQUEST PARA SUBIR LA RAMA A GITHUB Y PUEDA SER CHEQUEADA ANTES DE MERGEAR A MASTER

git push origin <rama x>

25- CERRAR ISSUES DESDE UN COMMIT

git commit -am "Fixes #<numero de la issue>: <mensaje del commit"

26- GITHUB AGREGAR COLABORADORES

Ir a settings luego manage acces e enivar la invitacion al colaborador

27- GITHUB COMENZAR UN PROYECTO

Incializar y generar las cards

28- GITHUB PAGE

Hosting gratuito para crear un sitio web con codigo html, css y javascript. No funciona con codigo que se ejecuta con node, phitoy, java, csharp (solo funciona si la app solo llama a endpoints desplegados en otro lugar)

Github page es bueno para crear contenido estatico o sitiosweb dinamicos que sean constuidos en base a javascript

como angular, react, nview, etc.

Se puede hacer un sitio web para un repositorio en particular o para un usuario determinado.

29- TRANSFERIR UNA ORGANIZACION Y RECUPERARLA

settings>trasferir>designar la organizacion

Para recuperar la organizacion trasnferida ejecutar un fork

Alias útiles:

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"