GIT

1. COMANDOS:

# 1.1. git help comand

## Con git help *comando* encontramos la información de cualquier comando que no recordemos como usar o que directamente no sepamos nada de él.

## [(ingresar) git help. Al usarlo sin adicionarle ningún nombre de algún comando se nos mostrarán todos los comandos principales de git.]-[(verificar) Aparecerán los comandos principales como config, add, etc.]

## [(ingresar) git help comand. Comand se refiere a cualquier comando en git por defecto.]-[(verificar) Aparecerá toda la información referente al comando.]

# 1.2. git config:

## [(ingresar)--global user.name “name”. Así se ingresa el nombre de un usuario.]-[(verificar) –global user.name. Deberá aparecer el nombre ingresado antes.]

## [(ingresar)--global user.email “name@email.com”. Así se ingresa el email.]-[(verificar) –global user.email. Deberá aparecer el email ingresado antes.]

## [(ingresar)--global color.ui true. Para ver colores en el cmd de git.]

## [(verificar todo lo hecho hasta el momento) --global –list. Así se verán todas las modificaciones hechas hasta el momento.]

# 1.3. cd …

## Muy importante porque nos permite ingresar la ruta donde se encuentran los archivos del espacio de trabajo

## [(ingresar) cd *pathOfTheFile.* Así se ingresa la ruta del archivo.(Verificar) En el bash al lado de MINGW debe aparecer la ruta especificada]

## [(ingresar) cd ../*pathOfTheFile.* El ../ permite decir a la ruta que retroceda una carpeta.(Verificar) En el bash al lado de MINGW debe aparecer la ruta especificada]

# 1.4. git init

## Marca el inicio y fin del proyecto. Permite inicializar un proyecto o reinicializarlo, por ejemplo, si cerramos el bash y queremos volver a trabajar con el repositorio. Para reinicializar el proyecto lo único que hay que hacer es ir a la ruta donde tenemos el proyecto (ósea, donde está el archivo .git) y poner *git init.*

# 1.5. git status

## Aclara el estado del proyecto respecto a las carpetas en las que se encuentra. Es decir, si se agregaron archivos nuevos al espacio de trabajo, si se modificó algún archivo o carpeta del espacio de trabajo, el porcentaje de modificación, entre otros.

# 1.6. git add

## Sirve para añadir archivos a el proyecto

## [(ingresar git add *nameOfTheFile*. Así se ingresa el archivo.]-[(verificar) con git status]

## [(ingresar) git add -A. Así se ingresan todos los archivos en la ruta o path.]-[(verificar) con git status.]

# 1.7. git commit -m “description”

## Así se guardan los commits o modificaciones con git.

## [(ingresar) git commit -m “description”. Al usar este comando es importante colocar la descripción del commit entre comillas, este identifica el commit y sin él, el comando no sirve.] -[(verificar) con git status]

## [(ingresar) git commit *--amend* -m –“nuevaDdescripción”. Cuando cometemos un error en la descripción del commit usamos este comando, primero hay que dirigirnos al commit con el comando 1.9.-1, luego usar nuestro comando cambiando la descripción.] – [ (verificar) usando el comando 1.8. veremos que nuestro commit ha cambiado de descripción]

# 1.8. git log

## Muestra todos los commits realizados hasta el commit en el que nos encontramos, esto quiere decir que, si no devolvemos a un commit anterior solo aparecerán los commits hasta ese punto. Además, aparecerá un mensaje con el código de identificación del commit de la siguiente manera:

## commit *códigoDeIdentificaciónDelCommit (A esto se le llama código SHA)*

## Author: *elNombreDeUsuarioDelAutor*

## Date: *Day Month Hour Year*

## *Descripción del Commit*

# 1.9. git checkout identificationOfText

## Nos permite volver a commits o ramas anteriores, esto se visualiza en el workSpace en el código, la visualización, etc.

## [(ingresar) git checkout *códigoSHA*. Así se regresa a un commit pasado]-[(verificar) con git status, los cambios mencionados anteriormente, en el bash pues aparece al lado el código de identificación (si es el commit actual debería decir el nombre de la rama en la que nos encontramos).]

## [(ingresar) git checkout *nombreDeLaRama* Así se regresa al último commit realizado en la rama.]-[(verificar) con git status, los cambios mencionados anteriormente, en el bash vuelve a decir el nombre de la rama y aparecen los commits correspondientes a esta.]

## [(ingresar) git checkout *textoDeIdentificaciónDeLaRama.* Así se navega entre ramas.]-[(verificar) Usando el comando 1.11.-1.]

## [(ingresar) git checkout -b *textoDeIdentificaciónDeLaRama.* Así se crea y navega a la rama creada.]-[(verificar) Usando el comando 1.11.-1.]

# 1.10. git reset (Peligro)

## El git reset elimina ciertos aspectos de los commits o los elimina completamente, por lo que es de MUCHO CUIDADO.

## [(ingresar) git reset *--soft.* Es el suavecito y no se mete con el workSpace o área de trabajo, ósea, lo que hace es borrar el registro del commit lo cual puede ser útil si se quiere cambiar la descripción en el comando 1.6. De esta manera solo se borra el registro del commit, pero los cambios relacionados al mismo se mantienen.]-[(verificar) Al poner git log nos aparecerá un commit menos, pero los últimos cambios relacionados al workSpace permanecen]

## [(ingresar) git reset *–mixed.* No se recomienda usarlo.]

## [(ingresar) git reset *–hard* (Peligro). Es el más peligros y BORRA TODO LO DEL COMMIT, es decir, se borrará el registro del commit, así como el trabajo realizado en el workSpace]

# 1.11. git branch nameOfTheBranch

## Nos muestra todas las ramas en el proyecto y nos permite crear las ramas del proyecto. Siempre aparecerá la rama *Master.*

## [(ingresar) git branch. Así se ven todas las ramas del proyecto]-[(verificar) Aparecerá *\*master* o la rama actual junto con los nombres de otras ramas.]

## [(ingresar) git branch *nombreDeLaRama*. Al poner el nombre de la nueva rama la crearemos, pero seguiremos en la rama master.]-[(verificar) Al poner *git branch* aparecerá *\*master* (indicándonos que seguimos en la rama master) luego rama creada y las otras ramas. Para viajar entre ramas revisar el comando 1.9-3]

## [(ingresar) git branch -D nombreDeLaRama. Así se borran las ramas.]-[(verificar) Al usar el comando 1.11.-1 dejará de aparecer la rama.]

# 1.12. git merge nameOfTheBranchToFuse

## Cuando se tienen dos ramas, y se quieren fusionar, hay que dirigirse primero a la rama1, a la que se van a agregar los commits de la otra rama (rama2) usan el comando 1.9-3. Luego se usa:

## [(ingresar) git merge *nombreDeLaRamaAFusionar*. De esta manera todos los commits de la rama2 se agregarán a la rama1]-[(verificar) Usando el comando 1.8. observaremos que todos los commits de la rama2 se agregaron a la rama1 y en el commit head aparecerá (rama2, rama1)]

## [(ingresar) git merge *nombreDelRepositorioRemoto/ramaDelRepositiorio. C*uando se usa el comando 1.17.-1, se crea una rama en nuestro repositorio local que es invisible hasta que es fusionada, con el comando aquí expresado podemos fusionar esa rama remota una rama de nuestro proyecto] – [(verificar) usar el comando 1.8. , se podrá observar que los commits que queríamos fusionar ahora se encuentran en nuestra rama]

## Es importante decir que una fusión puede ser de tipo *fast-forward o manual-forward*. La primera se hace de manera automática por git y, corresponde generalmente a modificaciones de pocas líneas de código o de un solo archivo. La segunda se hace de manera manual indicando en donde están los cambios realizados, esto sucede generalmente entre varios archivos o cuando se está trabajando con ramas remota, es decir, cuando se tiene un equipo de trabajo y alguien pasa su rama para fusionarla.

# 1.13. git clone url

## Permite bajar los archivos de un repositorio subido en la web para trabajar localmente en nuestro ordenador.

## [(ingresar) git clone *url*. En el url va el link del repositorio y así se clona]-[(verificar) En el bash de git aparecerá -*cloning into …-* y en la carpeta en la que se debe clonar el archivo aparecerá el archivo clonado.]

# 1.14. git remote

## Vincula el repositorio local con el remoto.

## [(ingresar) git remote add nombreDelRepositorio *url*. Así se le dice a git que repositorio local esta enlazado al repositorio de la url, ósea, son el mismo.]-[(verificar) Con el comando 1.14.-3.]

## [(ingresar) git remote remove nombreDelRepositiorio. Sirve para desenlazar el repositorio remoto, ejemplo, se nos cayó el link del repositorio y creamos uno nuevo.]-[(verificar) Aparecerá un mensaje en el bash que contiene: Una línea que dice “*origin url (fetch)”* y otra que dice “*origin url (push)”*.]

## [(ingresar) git remote -v..]- [(verificar para comando 1.13.-1) Aparecerá un mensaje en el bash que contiene: Una línea que dice “*nombreDelRepositorio url (fetch)”* y otra que dice “*nombreDelRepositorio url (push)”* ;(verificar para comando 1.13-2) No aparecerá nada en el bash indicando que no hay enlaces a repositorios remotos.]

# 1.1.5. git push

## git push sirve para subir nuestros archivos al repositorio.

## [(ingresar) git push *nombreDelRepositorio* nombreDeLaRama. Así se suben los commits de pertenecientes a la rama, si no hay commits no se puede hacer nada]-[(verificar) Primero en el git bash se pedirá la contraseña y nombre de usuario de github que va a subir la rama o commits, luego se podrán visualizar los archivos subidos en el url de los mismos.]

## [(ingresar) git push *nombreDelRepositorio nombreDeLaRama* -f. Con -f forzamos a que git suba una rama con el mismo commit, pero con la descripción cambiada (Sirve para cuando usamos el comando 1.7-2)] –[(Verificar) En el repositorio aparecerá el cambio].

## [(ingresar) git push nombreDelRepositorio *v.x.x(tag)* (ver comando 1.16.). Así se suben los tags o versiones al repositorio.] – [(verificar) En el repositorio debe haberse subido el tag]

## [(ingresar) git push nombreDelRepositorio –tags. Así se suben todos los tags en el repositorio local] – [(Verificar) Mirar el repositorio remoto]

# 1.16. git tag

## Un tag es un punto especificó en la historia de un repositorio o proyecto. Se puede usar para marcar versiones (versión 0.1, 1.0, 1.1.1, etc.) El tag le dice a los commits que versiones son.

## [(ingresar) git tag -a v.x.x –m “descripciónDelTag”. Un tag anotado es un tag que va a contar con un poco más de información que otras.]

## [(ingresar) git tag -a v.x.x -m “descripciónDelTag” *códigoSHA.* Así se agrega el tag a un commit anterior al que nos encontramos.]

## [(ingresar) git tag v.x.x. Esta es un tag ligero y contiene menos información.]

# 1.17. git fetch

## Cuando se está trabajando con compas en un repositorio, hay que saber añadir los commits o modificaciones que hagan a nuestras ramas de trabajo. Para esto usamos *git fetch.*

## [(ingresar) git fetch nombreDelRepositorio. De esta manera bajamos la rama llamada *nombreDelRepositorio/nombreDeLaRama.* Revisar el comando 1.12.-2 para saber cómo terminar de fusionar la rama del repositorio remoto a la nuestra]

2. Conceptos:

# 2.1. Commit:

## Un commit es un versión o punto de respaldo a partir de un cambio realizado a un proyecto.

# 2.2. Head:

## En git, *Head* se refiere al commit, versión, o modificación actual, en la que no encontramos. Esto es diferente del ultimo commitrealizado. Usualmente el head será el último commit, pero si usamos git checkout para viajar a un commit anterior, claramente el commit al que viajemos se convertirá en nuestro head. También podemos verificarlo al poner git log, pues este nos muestra las versiones anteriores al head y el head, pero no nos muestra el ultimo commit realizado necesariamente.

# 2.3. Ramas:

## La rama es la línea de commits que se tiene del proyecto, se puede ver como la “línea de tiempo”. En un proyecto puede haber varias ramas, pero siempre hay una rama *Master* que es la que tiene los cambios más importantes en el proyecto y es la más estable*.*

# 2.4. Repositorios:

## Los repositorios son las carpetas en las que guardamos y trabajamos. Hay repositorios locales(los que están en nuestras PCs y generalmente solo usamos individualmente) y los remotos (que se acceden por internet y se trabajan tanto en grupo como solitos :’v ).