**CURSO GIT**

**COMANDOS BÁSICOS POR CONSOLA**

1. **Configuración Inicial:**
   1. git config –-held
   2. git config -–global user.name “Eddy Ramirez”
   3. git config -–global user.email [eddy.ramirez.garcia.01@gmail.com](mailto:eddy.ramirez.garcia.01@gmail.com)
   4. git config -l
2. **Primer Repositorio:**
   1. Git init (Ejecutar este comando dentro del directorio raíz).
   2. (dir /a:h) este comando muestra el contenido oculto de la carpeta raíz. Nos mostrara la carpeta oculta (.git)
3. **Los 3 estados de Git:**
   1. **Espacio de trabajo:** Es todo el contenido que vallamos creando dentro de la carpeta raíz. A excepción de la carpeta oculta (.git). En esta área Git no hace seguimiento al contenido que exista.
      1. **Comando (git status):** Para verificar el contenido que está en el área de trabajo
   2. **Área de preparación:** Es el contendido seleccionado del área de trabajo los cuales se preparan para pasar al repositorio de Git, que es la carpeta oculta que se crea (.git).
      1. **Comando (git add):** Con este comando movemos el contenido del área de trabajo a el área de preparación.
   3. **Repositorio (carpeta .git):** Es el contenido que se preparó y ya forma parte del repositorio de Git.
      1. Comando git commit: Con este comando movemos el contenido del área de preparación al repositorio de Git.
      2. Dentro de la consola luego de ejecutar el comando (git commit) se abre el editor (Vim) donde tenemos que agregar un mensaje para confirmar el (commit), para esto presionamos la letra (i) para inserta dicho mensaje.
      3. Una vez escrito el mensaje presionamos la tecla (Esc) y seguidamente donde aparece el cursor escribimos (:wq) que confirma la escritura del mensaje y luego sale del editor (Vim).
      4. Por último, podemos usar el comando (git log) el cual nos muestra el registro de las confirmaciones (commit) del repositorio Git.
      5. Para salir de (git log) cuando la terminal muestre al final del mensaje (END) se debe presionar la tecla (q) y así retornamos al directorio raíz.
4. **Añadir y restaurar archivos:**
   1. Esta vez sí realizamos cualquier tipo de modificación a los archivos que están dentro del repositorio de Git pasaran nuevamente al área de trabajo y pasa lo mismo con los archivos que vallamos creando.
   2. Entonces podemos pasar archivo por archivo al área de preparación usando el comando (git add) o todos los archivos nuevos como los modificados con el comando (git add .).
   3. Luego que todos estos archivos están en el área de preparación podemos realizar el (git commit) que nos llevara al editor (Vin) para agregar el mensaje deseado. Pero en este caso podemos realizar un atajo para agregar el mensaje sin utilizar el editor (Vim) y es el comando (git commit -m “Aquí va el mensaje de forma directa”).
5. **Modificando la estructura del proyecto:**
   1. Modificamos el contenido del proyecto o carpeta que está siendo monitoreada por Git y realizamos nuevamente los comandos básicos para pasar toda esta información al repositorio de Git. Los cuales mencionamos a continuación en un orden secuencial:
      1. **git status:** Si existe se mostrará la modificación de los archivos ya creados y de los elementos nuevos agregados al proyecto. Estos se encuentran aún en el área de trabajo.
      2. **git add .:** Con este comando agregaremos directamente todo lo que se encuentra en el área de trabajo hacia el área de preparación.
      3. **git commit -m:** Con este comando pasaremos todo lo que está en el área de preparación al repositorio de Git. Pero este comando también nos da la posibilidad de escribir directamente una descripción para el commit sin la necesidad de abrir el editor de la consola (Vim).
      4. **git log:** Este comando se utiliza para ver todos los commit realizados en el repositorio actual de Git.
6. **Archivos nuevos y archivos modificados:**
   1. Se puede realizar en un solo paso el movimiento de **(solo los archivos modificados)** del área de trabajo hacia el repositorio de Git, sin la necesidad de enviarlos primero al área de preparación.
      1. **git commit -am “Descripción”:** Este comando me permite añadir y hacer el commit con su respectiva descripción de forma directa hacia el repositorio de Git. Por lo tanto, si se crea contenido nuevo se debe ejecutar los comandos de Git en el orden correspondiente para moverlos de las diferentes áreas hasta el repositorio de Git.
7. **Deshaciendo cambios realizados:**
   1. Cuando realizamos algunas modificaciones a nuestros archivos que ciertamente no nos sirve, podemos deshacer estos cambios realizados con el siguiente comando:
      1. **git checkout <name file>:** Deshace los cambios realizados según el archivo especificado que se encuentra en el área de trabajo.
      2. **git checkout .:** Deshace los cambios realizados a todos los archivos que se encuentran en el área de trabajo.
      3. **Git checkout -**f: Fuerza y deshace los cambios realizados a todos los archivos que se encuentran en el área de trabajo.
8. **Quitar archivos del área de preparación:**
   1. Para quitar los archivos o elementos que ya los pasamos al área de preparación y no deseamos hacer un commit, se usa el siguiente comando:
      1. **git restore –-staged <name file>:** Retornamos el archivo o elemento del área de preparación hacia el área de trabajo nuevamente.
      2. **Git restore –-staged .:** Retornamos de golpe todos los archivos o elementos del área de preparación hacia el área de trabajo nuevamente.
9. **Ver cambios realizados en cada archivo:**
   1. **git diff <name file>:** Este comando nos muestra la modificación realizada en el archivo que lo indiquemos.
   2. **git diff –-stat <name file>:** Este comando nos muestra un resumen de la modificación realizada en el archivo que lo indiquemos.
   3. **git diff –-unmstat <name file>:** Este comando nos muestra aún más resumido la modificación realizada en el archivo que lo indiquemos.
10. **Historial de cambios:**
    1. **git log –-oneline:** Muestra una lista en una sola línea de todos los commit realizados
    2. **git show <name commit>:** Podemos ver que cambios se realizaron en un commit en específico indicándole su identificador resumido.
    3. **git checkout <name commit>:** Me mueve a un commit en específico y también cambia la posición de la cabecera. En otras palabras, me muevo en el tiempo de acuerdo a la fecha de creación del commit.
    4. **git checkout master:** Con este comando retornamos al último commit realizado, en otras palabras, nos posicionamos en el tiempo actual.
11. **Opciones de visualización del historial:**
    1. Para ver más opciones de visualización al momento de mostrar los commit que tiene un repositorio de Git, es conveniente revisar la documentación de Git: <https://git-scm.com/book/es/v2>
12. **Ramas locales:**
    1. Desde un punto de la línea original de tiempo podemos sacar una o varias ramas que sigan su propia línea de tiempo, en otras palabras, muy independiente a la línea master.
    2. **git branch:** Este comando nos permite ver todas las ramas que existen dentro de nuestro repositorio de Git y el nombre de la rama que lleva un asterisco hacia su izquierda es en la cual se está posicionado.
    3. **git log -–oneline –-all:** Ya sabemos hasta cierto punto que este comando nos muestra los commit del repositorio Git en una sola línea, pero agregando (--all) nos garantiza que se mostrara todos los commit que hay en el repositorio Git tanto de la línea master como de las ramas.
    4. **git log -–oneline –-all --graph:** Muestra por medio de caracteres la línea de tiempo original y a partir de qué punto se van creando las ramas.
    5. **git checkout <name Branch> | git switch <name branch>:** Estos comandos sirven para movernos entre las distintas ramas que existan en nuestro contenedor Git, el primer comando es de una versión antigua de Git mientras que el segundo es un comando más actual de Git.
    6. **git checkout -b <name branch>:** Crea una nueva rama y automáticamente me posiciona en ella sin la necesidad de moverme a esta usando otro comando de Git.
    7. **git branch -d <name branch>:** Elimina la rama especificada y es una opción segura porque Git evita eliminar las ramas que aun tienen cambios y no se han fusionado.
    8. **git branch -D <name branch>:** Fuerza la eliminación de la rama especificada, incluso si tienen cambios sin fusionar. Este comando se utiliza para eliminar de forma permanente todas las confirmaciones asociadas a una línea concreta de desarrollo.
    9. **git branch -m <new name>:** Con este comando podemos cambiar el nombre de la rama, y tendrá efecto sobre la rama en la que se está posicionado actualmente.
13. **Fusionando ramas:**
    1. Ahora que tenemos 2 ramas la principal en la línea de tiempo llamada (master) y la rama que nace a partir de la línea principal llamada (development) lo que también podemos es fusionarlas en una sola, para esto usamos el siguiente comando:
       1. **git merge <name merge branch>:** Tener en cuenta que si uno no toca los mismos archivos de las ramas creadas en este caso (master) o (development) el proceso de fusionar ramas se realizara sin problemas. Pero si es el caso contrario donde si se manipularon los mismos archivos tenemos que ayudarlo en el proceso un poco a Git.
       2. Aplicar el comando anterior significa que estamos posicionados en la rama (master) y esta se unirá con otra rama, para este caso (development).
       3. **git reset --merge ORIG\_HEAD:** Este comando es conveniente utilizar en el caso se presente errores al momento de realizar una fusión de ramas, por tal motivo es muy importante no tocar los mismos archivos creados en cada rama independiente.
14. **Resolviendo conflictos:**
    1. En este apartado reutilizamos algunos comandos de Git para volver a crear y fusionar ramas.
       1. **git status:** Muestra el estado de los archivos y elementos que se encuentran en un directorio raíz del repositorio Git, exactamente en el área de trabajo.
       2. **git add .:** Mueve todos los elementos y archivos tanto los nuevos como los modificados al área de preparación.
       3. **git commit -am <descripción>:** Mueve todo los elementos y archivos que se hallan en el área de preparación hacia el repositorio de Git. Este comando te permite agregar la descripción del commit de forma directa.
       4. **git log --oneline --graph –all:** Te muestra el registro de todos los commit realizados, cada uno de ellos colocados en una sola línea y con una ayuda grafica para ver la posición y forma de las ramas.
       5. **git checkout -b <branch>:** Crea una nueva rama y te posiciona en ella de forma automática.
       6. **git branch -d <branch>:** Elimina una rama de forma segura, protegiendo a las ramas que aún no se han fusionado y tienen modificaciones o elementos por agregar al repositorio de Git.
       7. **git switch <branch>:** Me permite mover entre las ramas que existen en un repositorio de Git.
       8. **git merge <branch>:** Estando posicionado en una rama en específico cuya información se muestra en la ruta de la terminal, esta se podrá fusionar con el nombre de la rama que se incluye en el comando que se va a ejecutar.
15. **Ignorando archivos:**
    1. Dentro de un repositorio de Git uno puede crear ciertos archivos e incluir elementos que no deseamos que Git les haga seguimiento, con esta finalidad se crea un archivo (.gitignore) en la raíz del proyecto el cual nombrara los elementos o archivos que no se le realizara ninguna clase de seguimiento por no prescindir de ellos.
16. **Manejo de ramas remotas:**
    1. Para guardar nuestros proyectos de forma remota haremos uso de la plataforma GitHub. Previamente para hacer esto tenemos que crearnos una cuenta en GitHub para pasar a crear un nuevo repositorio dentro de él.
    2. Este repositorio se almacenará en los servidores de GitHub en la nube. Pasamos a crear el nuevo repositorio y para esto llenamos los parámetros que GitHub nos pide.
    3. Luego que se crea el repositorio en GitHub tenemos 2 formas de descargarlo por medio de (HTTPS) o (SSH).
    4. Elegimos una de las 2 opciones y copiamos la dirección que se genera que nos sirve para clonar el repositorio remoto y usarlo de forma local.
    5. **git clone <dirección (HTTPS)>:** Ejecutando este comando clonamos el repositorio remoto para trabajar con el de local. En este caso como estamos empezando, el repositorio este vacío, pero normalmente los proyectos que se clonan no lo están eso significa que el proyecto descargado tendrá todo su historial (commit, autor, etc.). *Usar la dirección (HTTPS) es la forma directa que tenemos para clonar un repositorio.*
    6. **git clone <dirección (SSH)>:** *Para utilizar esta dirección tenemos que realizar un poso adicional, el cual es almacenar la llave publica de nuestra cuenta de GitHub para realizar esto ingresamos al apartado de configuración y seleccionamos (SSH and GPG keys).* Como no tenemos ninguna (SSH Key) pasamos a crear una usando el siguiente comando:
    7. **ssh-keygen -t ed25519 -C "mi email de GitHub" / ssh-keygen -t rsa -b 4096 -C "mi email de GitHub":** Utilizamos este comando en la terminal de Windows en cual nos crea una Key con un ID predeterminado (Se recomienda dejar el ID predeterminado), pero nos da la opción de cambiar el nombre del ID. Luego se nos pedirá agregar una contraseña la cual ingresamos 2 veces, una vez realizado esto se generan 2 archivos en la ubicación que nos indican, el archivo que nos interesa tiene la extensión .pub y es el cual abriremos para copiar su contenido y pegar dentro del apartado (SSH Key / Add new). Colocamos un titulo y luego pegamos la Key así el proceso estar terminado.
    8. **Todos los pasos realizados hasta el punto (g) son para la creación y clonación de repositorios ubicados en la nube de GitHub, pero ahora pasaremos a subir nuestros repositorios locales de Git hacia los repositorios de la nube de GitHub.** Usaremos los siguientes comandos:
    9. Los agregamos en el siguiente orden de acuerdo a lo sugerido por GitHub, en este caso sería utilizando el enlace de (Key SSH) por lo que ya tenemos una llave creada con nuestro ordenador.
       1. **git remote add origin git@github.com:eddRamirez/curso-git.git**
       2. **git branch -M main**
       3. **git push -u origin main**
    10. Una vez que témenos nuestro proyecto local de Git subido al repositorio remoto de GitHub ya tenemos una comunicación entre ambos y cada vez que actualicemos nuestro proyecto local de Git y queremos nuevamente subir esos cambios a GitHub basta con ejecutar el siguiente comando:
        1. **git push:** Como mencionamos antes una vez exista la comunicación entre (Git local) y (GitHub remoto) solo basta con ejecutar este comando para subir los cambios a la nube de GitHub.
        2. **git remote -v:** Con este comando sabremos hacia donde apunta la cabecera del repositorio remoto (HEAD -> main, origin/main).
        3. **git fetch:** Como ya tenemos la comunicación entre (GitHub) y (Git) utilizamos este comando para descargar los cambios que se hallan hecho en (GitHub) hacia (Git). Pero solo podemos ver los commit nuevos creados en (GitHub) y para poder visualizar los archivos con las modificaciones y elementos nuevos si es que los hay dentro de nuestro proyecto local (Git) utilizamos el siguiente comando:
        4. **git pull:** Este comando va ha jalar el contenido nuevo de remoto (GitHub) hacia local (Git).
17. **Creación de una rama remota:**
    1. Cuando se crea una nueva rama en (Git) local, podemos trabajar con ella y después cuando creamos conveniente fusionarla con el repositorio master, pero todo esto se da a nivel local.
    2. Ahora lo que queremos es tener esta nueva rama creada en local subida a (GitHub) para que este disponible en la nube, entonces aplicamos el siguiente comando:
    3. **git push origin <branch name>:** Este comando ya se utilizo y como sabemos sube los cambios de (Git) local hacia (GitHub) en la nube. Pero como vemos se agregó (origin <branch name>), así lo indicamos que el origen será la rama creada la cual se subirá a (GitHub).
18. **Eliminación de una rama remota:**
    1. Lo primero que tenemos que hacer es fusionar la rama creada con la rama principal, lo haremos estando en (Git) local y usando el comando:
       1. **git merge <branch name>**
    2. Ahora si realizamos un (git log) veremos que la rama se fusiono con la rama principal pero aun esta presente a nivel (Git) local y (GitHub) remoto. Por lo tanto, primero pasaremos a eliminar la rama a nivel (Git) local con el siguiente comando:
       1. **git branch -d <branch name>**
    3. Volvemos a ejecutar un (git log) y veremos que solo sigue estando presenta la rama a nivel remoto (GitHub). Ahora solo queda eliminar esa rama remota y para ello utilizamos el siguiente comando:
       1. **git push –-delete origin <branch name>**
    4. Al ejecutar (git log) vemos que nuestro proyecto solo tiene la rama principal a nivel local (Git) donde la cabecera se encuentra posicionada. Lo que significa que tenemos un proyecto a nuestro gusto y a partir del cual podemos seguir ejecutando las mismas operaciones de (Git) y (GitHub) para seguir teniendo un historial en orden cronológico de crecimiento de nuestro proyecto ya que de seguro lo tendrá en el tiempo.

**NOTA FINAL:** Hacemos mención que todos estos comandos descritos hasta el momento son los más necesarios y elementales para trabajar con (Git) local y (GitHub) remoto y así tener un historial de crecimiento y en orden cronológico de nuestros proyectos, sobre todo respaldados en la nube.