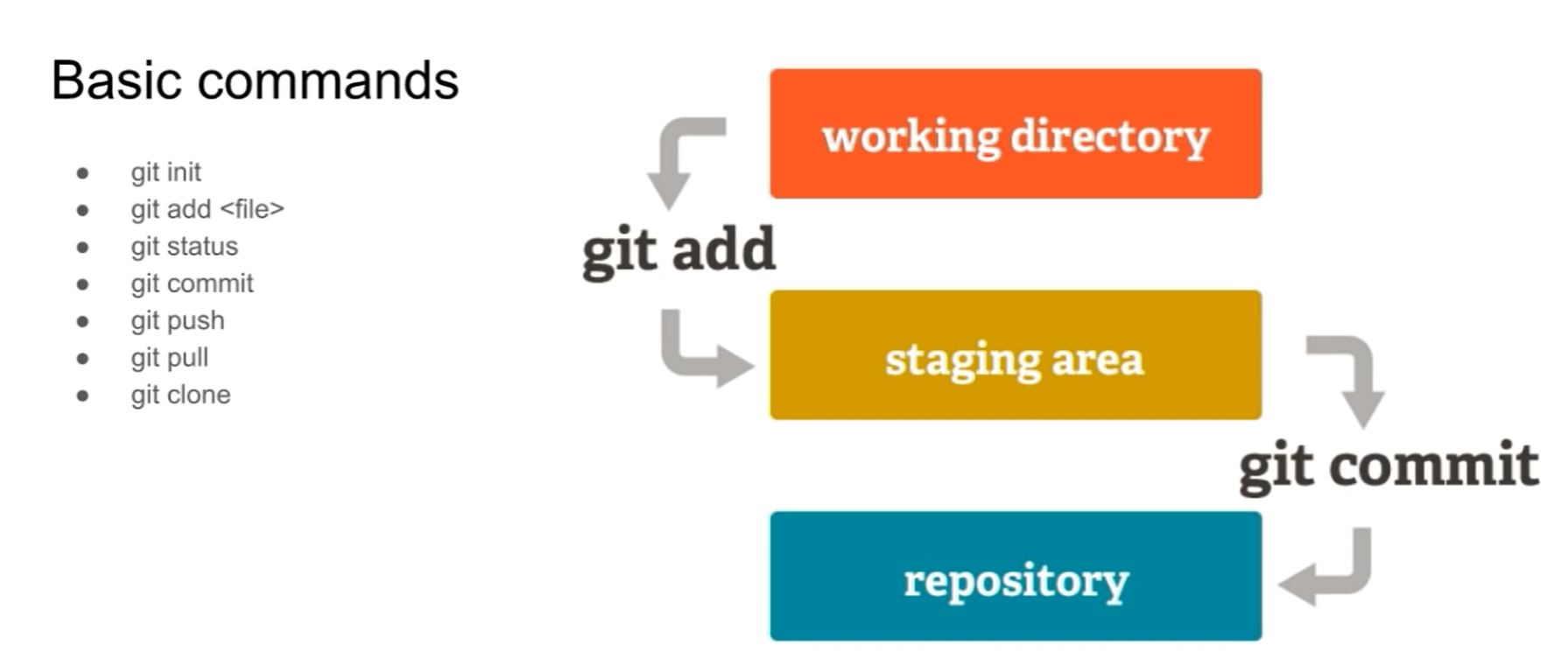
**Curso Git – GitHub**

Cuando hablamos de estas herramientas, hablamos de un usar un sistema de control de versiones.



Cuando hablamos de GIT se debe tener en cuenta que aquí contamos con tres estados.

**WORKING DIRECTORY**

Aquí es donde estaremos trabajando con todos nuestros estados

**STAGIN AREA**

Es el lugar donde se agregan los archivos que preparamos para el guardado.

**REPOSITORY**

El lugar donde se guardará cada una de nuestras versiones.

**Comandos básicos**

**GIT INIT**

Es como le decimos a nuestro sistema operativo que vamos a crear un proyecto nuevo.

**GIT ADD**

Es para pasar los archivos de WORKING DIRECTORY al STAGIN AREA

**GIT STATUS**

Es para ver en qué estado están nuestros archivos

**GIT COMMIT**

Es para pasar del STAGIN AREA al REPOSITORY es para crear una foto de nuestra versión de código.

**GIT PUSH**

Es para subir nuestro proyecto a un repositorio remoto (Servidor)

**GIT PULL**

Trae los cambios que han realizado los otros desarrolladores, en caso de que alguien este usando el proyecto.

**GIT CLONE**

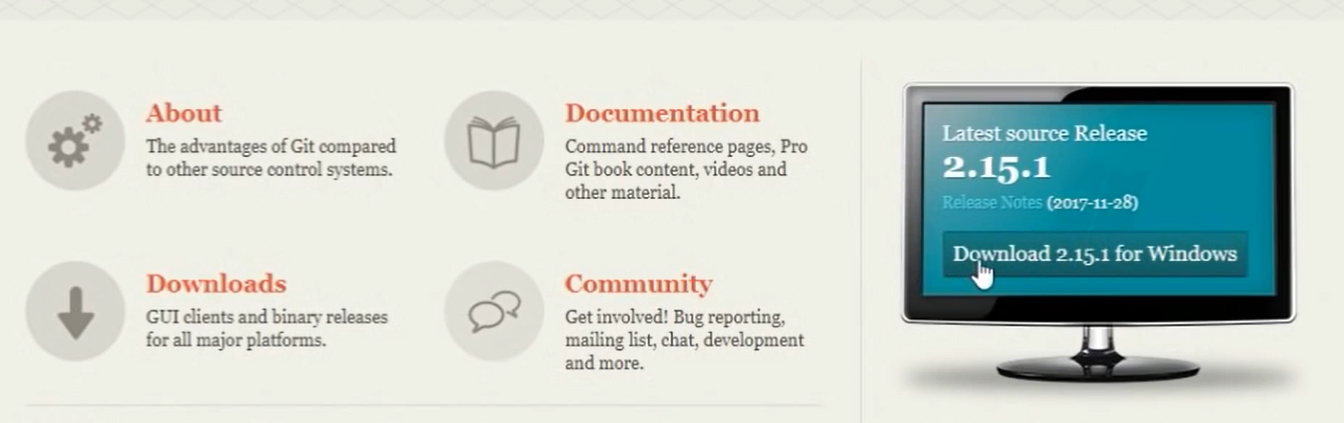
Lo que hace es hacer una copia desde el servidor central donde está el código hasta el computador.

**Descarga e instalación**

Primero vamos a la página principal de GIT

Enlace: <https://git-scm.com/>

Seleccionamos la versión y lo descargamos.

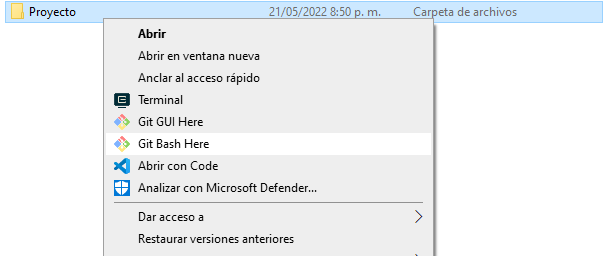


La misma página nos proporciona un libro (manual) en el que nos enseña como trabajar con Git, se recomienda descargar la versión en español.

EMPEZAR A TRABAJAR CON GIT

Cuando nosotros trabajamos con Git vamos a tener una forma de seguir los archivos, pero antes de empezar se debe inicializar el repositorio de código y para eso, lo primero que se debe hacer es abrir un terminal.

Vamos al proyecto y damos CLICK derecho y damos seleccionamos la opción: GIT BASH HERE

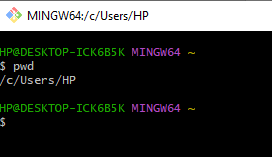


O también se puede ir al CMD, acceder al directorio y hacer uso del comando GIT INIT el cual va a permitirnos trabajar.

También podemos ir a nuestra consola de GIT BASH la cual fue instalado al momento de instalar GIT

Una vez dentro de la terminal de GIT podemos hacer uso de los comandos

**PWD**: el cual nos permite identificar el lugar donde nos encontramos.



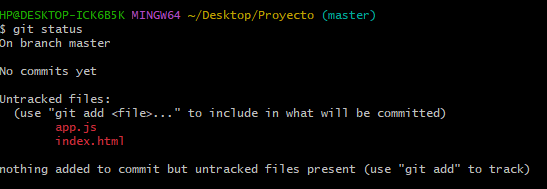
Posteriormente ingresamos a nuestro proyecto

**LS:** Este comando nos muestra los archivos que tiene la carpeta en la que nos encontramos.



Es aquí donde vamos a hacer uso de nuestro comando GIT INIT

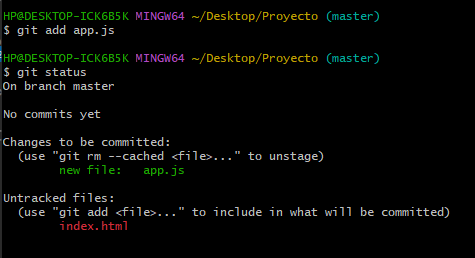
Una vez iniciado el comando, podemos hacer uso del comando GIT STATUS el cual nos permite ver que archivos estamos trabajando.



Tal como se ve, nos dice inicialmente que de momento no estamos trabajando con ningún archivo, además nos muestra que dos de nuestros archivos aún no han sido agregados a nuestra área de trabajo.

En la imagen anterior de igual forma nos dan el comando necesario con el cual podemos agregar archivos a nuestra área de trabajo.

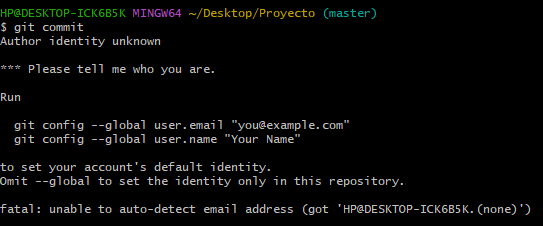
**GIT ADD**



Tal como se ve en la anterior imagen, hemos hecho uso del comando mencionado y hemos agregado uno de los archivos por lo que, al verificar nuevamente el estatus de nuestro proyecto, este nos retorna un nuevo valor agregado.

Hasta este punto ya hemos agregado uno de los archivos, pero es necesario agregarlos todos los archivos con los que vamos a trabajar dentro de nuestro proyecto. Una vez agregados, hacemos uso de nuestro comando el cual nos permite crear nuestro primer punto de control.

**GIT COMMIT**



Una vez aplicado el comando, en consola se nos presenta un mensaje el cual nos pregunta quienes somos. Esto se hace ya que cuando trabajamos con GIT muchas personas pueden realizar cambios sobre determinado proyecto.

De igual forma en consola se nos muestra la forma como podemos configurar nuestra identidad y especificar quien es la persona que realiza los cambios.

Usamos los comandos que se nos dan para agregar inicialmente al email



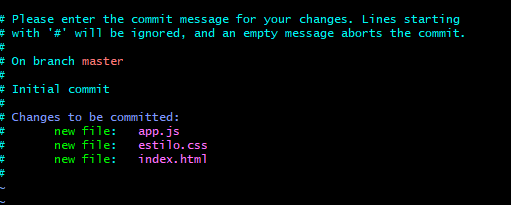
Después usamos el comando para configurar a nuestro usuario



Una vez hecho esto, ya tenemos configurados tanto nuestro correo como nuestro nombre.

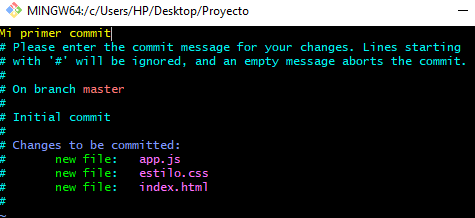
Ya que se registró quienes somos, podemos hacer uso nuevamente de GIT COMMIT y tener nuestra primera captura del proyecto.

Una vez ingresado el comando nos va a aparecer algo como esto.

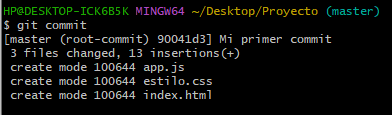


Lo que podemos ver en la anterior imagen es un editor de código dentro de la terminal, el editor es conocido como BIN

En la parte superior de este código se puede poner el nombre que queremos a nuestra versión de Git.



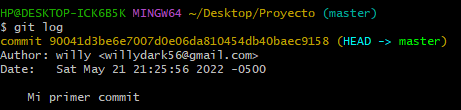
Una vez hecho esto, damos escape seguido de el comando: wq lo cual nos saca del editor y nos muestra.



Se puede ver en la imagen que se nos creó un ID para cada uno de los archivos, además nos informa de todos los cambios realizados. Ahora podemos hacer seguimiento la versión que hemos creado.

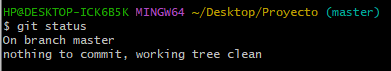
**GIT LOG**

Este comando nos permite ver todos los COMMITS que hemos creado.



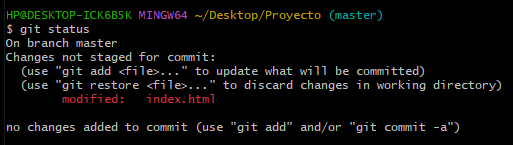
En la imagen podemos ver el ID asignado a nuestra primera versión de nuestro proyecto, de igual forma se puede ver quien la modifico, su nombre y correo, además nos muestra la fecha en que fue modificado.

Si en este punto damos el comando GIT STATUS el mensaje retorno nos va a decir que no hay nuevos cambios que guardar dentro de nuestro proyecto.



**GIT CHECKOUT**

Lo que hace este comando es revertir todos los cambios que se han hecho sobre el archivo.



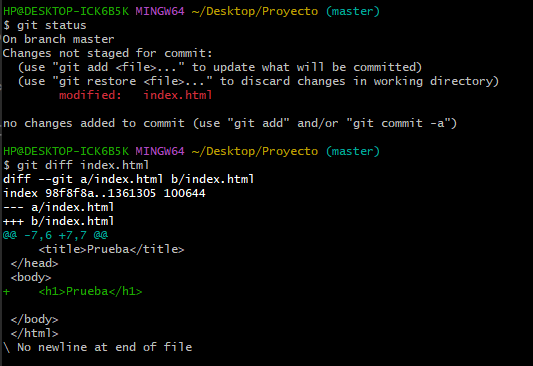
Cuando cambio uno de los elementos dentro de mi editor de texto y posteriormente escribo GIT STATUS dentro de la terminal de Git, nos va a decir que uno de los archivos fue cambiado. Si después de esto hago uso del comando GIT CHECKOUT los cambios que se habían realizado sobre el archivo van a ser revertidos.



Si vamos al archivo en cuestión vamos a ver que efectivamente los cambios han sido revertidos.

**GIT DIFF**

Este comando lo que hace es mostrar en consola los cambios que se han hecho sobre un documento en específico.



Tal como se ve en la imagen, inicialmente hemos usado GIT STATUS para identificar si se hicieron cambios sobre el proyecto. Git nos dice que se modificó y no se guardo un cambio dentro de INDEX.HTML posteriormente con el comando GIT DIFF se nos muestra en pantalla que fue lo que se agregó y no se guardó dentro del archivo en cuestión. En nuestro caso se agregó un H1 con un título. Esta parte se muestra de color verde en consola.

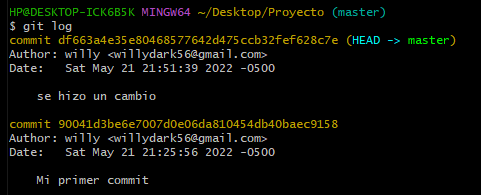
Si lo que queremos es agregar los nuevos cambios hechos sobre el archivo, debemos hacer nuevamente uso del comando GIT ADD y asignarle el archivo que deseamos.



Lo que hemos hecho es agregar los cambios, pero estos aún no han sido guardados dentro de nuestro proyecto. Para poder guardarlos debemos nuevamente usar el comando GIT COMMIT.

Una vez lo hacemos, proporcionamos nuevamente un título a nuestro cambio y nuevamente damos escape, comando: wq

Si ahora ingresamos el comando GIT LOG vamos a poder ver las versiones o capturas que tenemos de nuestro proyecto y de igual forma se va a ver sobre cual estamos trabajando ya que se va a mostrar el HEAD del proyecto.



Quedamos minuto 28

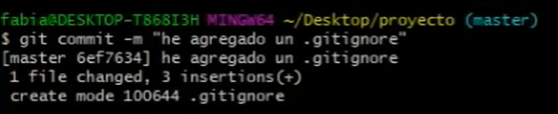
**No se quiere agregar nuevos archivos**

Qué pasa si nosotros agregamos nuevas carpetas o archivos dentro de nuestro proyecto, pero no queremos que estas sean incluidas dentro de nuestra versión interna de Git.

Si lo que queremos es ignorar cierta información, lo que hacemos es que a nuestro archivo le agregamos el nombre “.gitignore” dentro del archivo vamos a agregar los nombres de las carpetas y de los archivos que deseamos ignorar.

**Agregar un COMMIT sin ingresar a editor**

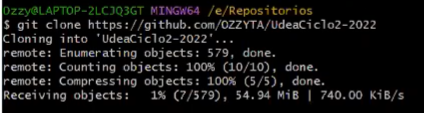
Si queremos agregar un nuevo COMMIT y no queremos hacer uso del editor, proporcionar un nuevo mensaje y demás, podemos hacer uso del comando.



En la imagen podemos ver que agregamos un COMMIT con la diferencia de que al poner -m especificamos que el mensaje lo proporcionaremos de forma directa, sin ingresar al editor.

**GIT CLONE**

Si lo que queremos es, trabajar sobre un proyecto de otro repositorio, podemos clonarlo. Para ello copiamos la URL donde se encuentra el repositorio y posteriormente, en consola usamos el comando GIT CLONE seguido de la URL y damos ENTER



En la imagen se puede ver que, efectivamente hemos clonado el repositorio, ahora lo podremos encontrar en nuestra carpeta.

**Llenar un repositorio vacío**

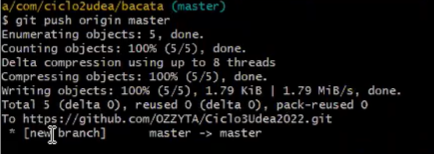
Si tenemos creado un repositorio en la nube, pero está vacío, podemos cargar datos desde nuestro computador.

Ingresamos a la carpeta sobre la que estamos trabajando, damos CLICK en abrir con GIT BASH y después de haber realizado el respectivo GIT INIT, GIT ADD y GIT COMMITE, agregamos al origen



En la anterior imagen podemos ver que, con la dirección del repositorio al cual queremos guardar la información, hemos usado nuevamente el comando GIT ADD, pero en esta ocasión lo que hicimos fue utilizar la palabra ORIGIN, especificando el lugar al cual lo vamos a enviar.

Posteriormente realizamos el respectivo PUSH, pero ahora usando nuevamente la palabra ORIGIN seguido de la especificación (master)

En nuestro caso usamos en lugar de MAIN – MASTER. El (-U) lo usamos una sola vez si el PUSH se hace desde la misma rama.

https://github.com/OZZYTA/Ciclo3Udea2022.git

**CREACION DE NUEVAS RAMAS**

Cuando son varios los programadores que están trabajando sobre un proyecto, existe la posibilidad de que, un cambio realizado en el repositorio, genere problemas para los demás, por esta razón, GIT permite la creación de diferentes ramas.

**Git BRANCH**

****

Para crear la nueva rama, lo que hacemos es utilizar la palabra reservada BRANCH seguido del nombre que queremos darle a la nueva rama.

Una vez creada la rama, teniendo en cuenta que no existe ninguna respuesta que indique la creación exitosa de forma inicial, podemos consultar si la rama fue o no creada.



Lo que hacemos es chequear la existencia de una rama con el nombre esperado y la respuesta nos indica que, efectivamente la rama si existe.

**NAVEGAR ENTRE RAMAS**

**GIT CHECKOUT**

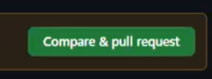
****

Este comando nos permite seleccionar la rama sobre la cual queremos trabajar, en este caso hemos pasado a la rama llamada RamaNatali

Una vez creada la rama, si lo que queremos es que los cambios se apliquen a esa rama podemos hacerlo usando los comandos anteriores de; GIT PUSH ORIGIN (‘ponemos el nombre de la rama’)



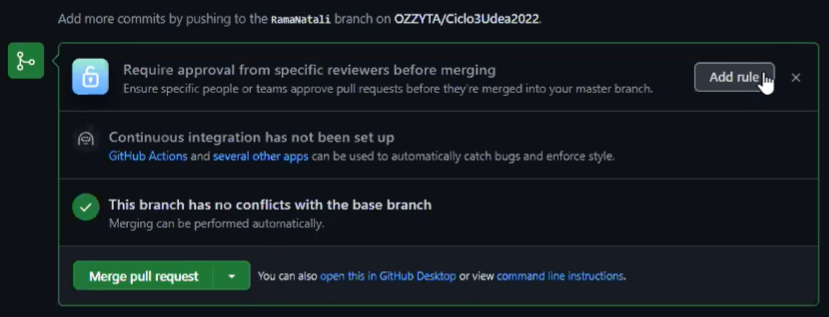
Una vez hecho esto y considerando que los cambios hechos en nuestra rama son los necesarios, podemos hacer un empalme con la rama original de nuestro repositorio, por lo que, nos dirigimos a nuestro repositorio en la nube para realizar un PULL REQUEST



El PULL REQUEST es una solicitud que se hace al dueño del repositorio para pedir que se permitan agregar los cambios al repositorio original, en caso de que sean compatibles.

Respuesta

Si los cambios no generan ningún conflicto, al administrador del repositorio le va a aparecer el siguiente mensaje.



De lo contrario, saldrán mensajes color rojo indicando los elementos incompatibles del código que generarán conflicto.

**GIT LOG**

Este comando nos permite ver la información de cada COMMIT generado dentro de nuestro proyecto, al igual que, la información de quien lo hizo y para que rama se hizo.

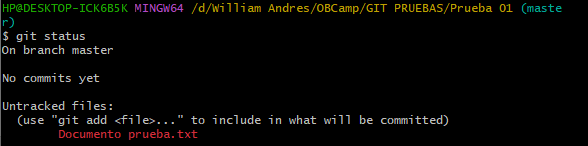
**GIT OPEN BOOT CAMP**

**GIT INIT**

Este comando nos genera un repositorio local en nuestro computador, en el lugar que hemos seleccionado.

**GIT STATUS**

Este comando nos va a mostrar los elementos que están dentro de la carpeta que hemos iniciado el repositorio local.



En la anterior imagen podemos ver nos presenta un documento dentro de la carpeta el cual no ha sido guardado aún.

**GIT ADD**

Este comando nos permite agregar los cambios realizados a partir de la adición de nuevos archivos

Usamos este comando para pasar los archivos de WORKING DIRECTORY al STAGIN AREA

Si queremos agregar un archivo en específico hacemos uso de el comando GIT ADD seguido del nombre del archivo y su extensión. Si lo que se quiere es que se agreguen todos los archivos nuevos, usamos el comando GIT ADD seguido de un.



NOTA: Una vez hecho esto, el cambio realizado se encuentra en un espacio conocido como STAGIN AREA, por lo que se encuentra a la espera de una confirmación que permita realizar el cambio de forma definitiva.

**Configuración de mis credenciales**

Se debe saber que cuando trabajamos desde nuestro GIT BASH utilizamos credenciales que nos identifican dentro del sistema, estas credenciales se presentan de tres tipos; Credenciales del sistema, globales y sssss

Ver mis credenciales del sistema

Para ver que credenciales tenemos dentro del sistema, hacemos uso del comando



Esto nos va a mostrar nuestras credenciales dentro del sistema

Ver mis credenciales globales



Como se puede ver, utilizando el anterior código podemos ver las configuraciones globales de nuestras credenciales dentro del sistema

Ver mis configuraciones y credenciales a nivel local



IMPORTANTE

Cada una de las opciones anteriormente mencionadas, modifican un fichero diferente dentro del GIT. La opción del sistema va a configurar el sistema, el comando global, estamos modificando el fichero GIT.CONFIG desde nuestro usuario y cuando usamos el comando local podemos configurar los archivos dentro del GIT.CONFIG.

**Modificar configuraciones**

Globales



El anterior código, lo que hace es modificar al usuario. Usamos GIT CONFIG –GLOBAL USER.NAME dejamos un espacio y seguido ponemos entre comillas dobles el nombre del usuario que queremos tener, pasa igual con el EMAIL

Locales

Podemos de igual forma configurar las opciones con respecto a nuestro repositorio local y para ello usamos el siguiente comando



El código es similar a la opción anterior, lo único que cambió es el lugar al cual se le aplicó la modificación.

NOTA: Las configuraciones locales se usarán cuando hagamos cambios dentro del repositorio local y las configuraciones globales se usarán cuando hagamos cambios sobre cualquier otro repositorio fuera.

**GIT TAG (etiqueta)**

Permite al usuario crear una etiqueta para diferenciar cierto espacio. Existe la posibilidad de que, los desarrolladores quieran diferenciar sus versiones no solo por el código que se asigna automáticamente dentro de cada COMMIT, sino que también se quiera diferenciar, por esta razón hacemos uso de este comando.



Con este comando se crean etiquetas ligeras



Con este comando usamos etiquetas más pesadas, las cuales contienen mucha más información que las etiquetas ligeras. Aquí encontramos metadatos como autor, número y fecha de la etiqueta.

NOTA: Se recomienda utilizar este comando cuando trabajamos dentro de proyectos importantes.

Para agregar nuestra etiqueta podemos hacer uso del código generado dentro del COMMIT que queramos.

Ejemplo



En la anterior imagen podemos ver que hemos usado el comando mencionado a la vez que agregamos el nombre de la etiqueta el cual en este caso es “v1.0” haciendo referencia a la versión, posteriormente adjuntamos el código del COMMIT sobre el cual queremos que se especifique el etiquetado. Si nosotros no agregamos el código del COMMIT, GIT va a agregar automáticamente la etiqueta a la versión actual sobre la cual estemos trabajando. Una vez damos ENTER sobre el cambio, se nos va a pedir ingresar un mensaje descriptivo sobre la etiqueta, ingresamos el mensaje y quedara listo. Una vez ingresado el mensaje debemos hacer GIT PUSH para cargar los cambios a nuestro repositorio en la nube, pero debemos saber que se necesita especificar el origen de los cambios.



Tal como se ve en la anterior imagen, hemos empujado los cambios de forma correcta.

**NAVEGAR ENTRE VERSIONES**

Una vez versionado un determinado COMMIT podemos ir hasta el especificando su nombre



Utilizamos el comando CHECKOUT para cambiar de ramas, en este caso estamos navegando entre versiones. Como se ve en la imagen hemos accedido a la versión “v1.0” la cual hemos creado en la parte de arriba. Cabe resaltar que esa versión va a contener el código que teníamos en ese punto.

Al hacer esto obtendremos una visualización del código que teníamos en aquel momento.

**GIT REVERT**

Este comando lo usamos para regresar a versiones antiguas.



Como se ve en la imagen, usamos el comando anteriormente mencionado, posteriormente agregamos parte del código del COMMIT. Resaltamos que no hace falta agregar todo el código del COMMIT solo una parte representativa que permita diferencial el COMMIT.

IMPORTANTE: Al retroceder en versiones, no se va a borrar el historial, únicamente nos posicionaremos en la versión que nos interesa.

**GIT RM**

Este comando nos permite eliminar algún fichero que tenemos dentro de nuestro proyecto de GIT.



Como se ve en la imagen, usamos el comando mencionado seguido del nombre del fichero que queremos eliminar. Se debe mencionar que, para que el cambio sea efectivo, debemos utilizar posteriormente el GIT COMMIT Y GIT PUSH.

**Abrir un proyecto desde un ID o editor de texto**

Algo importante y que debemos tener en cuenta es que, GIT permite conexión con IDE’S o editores de texto, de modo que, si tenemos una carpeta que ha sido inicializada con GIT en nuestro computador, podemos abrirla con nuestro editor preferido y trabajar desde el mismo, teniendo en cuenta que estará conectado con GIT.

Si lo que queremos es conectar nuestro IDE con un repositorio en la nube, podemos hacerlo de igual forma ya que, tanto los editores de texto como los IDE’S presentan la opción de hacerlo. Una vez que conectamos el IDE con nuestra cuenta de GIT, se va a crear un repositorio local con la información del proyecto y podremos trabajar sobre el mismo y asociar los cambios realizados con nuestro repositorio en la nube.

PENDIENTE VIDEO 4

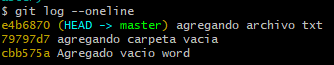
**GIT RESET**

Este comando permite al usuario deshacer cosas ya hechas, este comando tiene dos formas de operar, la forma dura y la forma ligera.

Forma ligera

Ejemplo

Sabemos que con cada COMMIT que creamos se genera un punto de control dentro de GIT, por lo que en el presente ejemplo hemos creado tres COMMIT’S y accedemos al control de los mismos usando el comando GIT LOG –ONELINE



Tal como se ve en la anterior imagen, se nos presentan los tres registros de COMMIT’S que hemos creado y el cabecero se ubica en el último COMMIT registrado.

Si lo que queremos es deshacer algún cambio, contamos las posiciones a partir del head, en este caso queremos deshacer el incluir el archivo de Word.



Lo que hacemos al usar el anterior comando es decirle que queremos deshacer un COMMIT contando a partir del cabecero (Incluido el cabecero) por lo que cuando hagamos nuevamente un COMMIT, vamos a borrar (deshacer) un COMMIT, en este caso se borrara el de cabecero que en nuestro ejemplo lleva por nombre “agregando archivo txt”

Resultado



Tal como se ve, ahora tenemos 3 registros de COMMIT’S incluido el de los cambios hechos usando RESET, pero ya no vemos el COMMIT donde se agregó el archivo TXT, por lo que, efectivamente se borró el archivo en cuestión.

IMPORTANTE

GIT no ha borrado archivos, lo que ha hecho es resetear el HEAD por lo que ahora se encuentra ubicado donde le hemos dicho

Forma pesada



Lo que hemos hecho con el comando es regresar 2 posiciones incluido el cabecero, por lo que ahora nos encontramos ubicados en el momento en el que agregamos el archivo de Word.

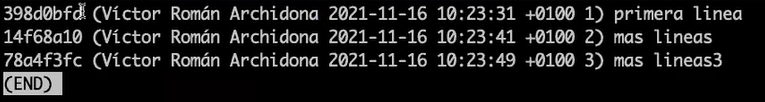
IMPORTANTE: Utilizando este comando, los archivos que habíamos creado anteriormente se han borrado, no solamente hemos reseteado el HEAD, también hemos eliminados todo lo que GIT internamente no conoce hasta el punto donde regresamos.

**GIT BLAME**

Este comando lo que hace es permitir al usuario ver sobre determinado fichero, quien y cuando hizo determinados cambios.



Para usarlo, ingresamos el comando en consola, seguido ponemos el nombre del fichero sobre el cual queremos trabajar y damos ENTER



Tal como se ve, una vez ingresamos el comando, podemos ver sobre que COMMIT se trabajó, que cambios se hicieron y e que fecha se hizo.

**RAMAS DENTRO DE GIT**

GIT permite al usuario crear ramas sobre determinado fichero, lo que permite que trabajemos usando la información del fichero original sin perturbar su código ya que contamos con una copia del mismo, una vez los cambios sean los esperados podemos actualizar esa información sobre nuestra rama principal.

**GIT BRANCH**

Nos permite ver que ramas tenemos creadas en nuestro fichero.



Tal como se ve, hemos hecho uso del comando mencionado y la terminal nos dice que hasta el momento solo contamos con la rama por defecto que nos construye GIT la cual es MASTER

**Crear nuevas ramas**

Si lo que queremos es crear una nueva rama para trabajar sobre la misma, podemos hacerlo usando el comando GIT BRANCH seguido del nombre que queremos darle a la nueva rama



**Cambiar de ramas**

Una vez estemos dentro de nuestra rama y contemos con otras, podemos navegar entre cada una de ellas y para ello hacemos uso del comando CHECKOUT seguido del nombre de la rama a la cual queremos ir



Tal como se ve en la anterior imagen, hemos cambiado nuestra posición, ahora nos encontramos en la rama seleccionada.

IMPORTANTE

Los cambios que se agregan a determinada rama, ya sea la creación de nuevos archivos o los cambios realizados, solo existen sobre la rama sobre la cual estamos trabajando.

**Otra forma de crear ramas**



Lo que hacemos con el anterior comando es crear una nueva rama y asignarle un nombre, de igual forma hacemos el cambio a esa rama, algo que se debe tener en cuenta es que, cuando hacemos uso de este comando, si la rama en cuestión no existe, este comando la crea.

IMPORTANTE: Cuando usamos este comando, la copia se va a hacer a partir de la rama en la que nosotros estemos ubicados, si estamos en MASTER, se va a hacer la copia de la rama MASTER.

**Creando una nueva rama a partir de un COMMIT**

GIT nos permite crear una nueva rama partiendo de determinado COMMIT



Tal como se ve en la imagen, usamos nuestro comando GIT BRANCH, asignamos un nombre y finalmente damos el código del COMMIT desde el cual queremos hacer una rama.

**Ramificación y unificación**

Una vez tenemos nuestra rama creada a partir de otra, suele presentarse la necesidad de unificar los cambios desde una a la otra y por esta razón GIT nos da las herramientas necesarias para hacerlo. Para hacer la unificación GIT nos presenta dos comandos.

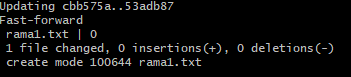
**GIT MERGE**

Para usar este comando, inicialmente debemos ubicarnos en la rama la cual queremos que sea actualizada, una vez ubicados en la rama hacemos uso el comando.



Tal como se ve en la imagen, hemos usado el comando mencionado seguido del nombre de la rama de la cual queremos traer los cambios.

Resultados

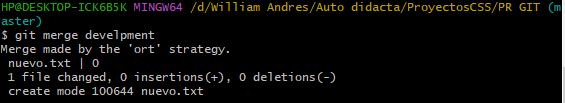


En la imagen podemos ver la confirmación de la actualización realizada sobre nuestra rama MASTER ya que fue a la rama MASTER a la cual trajimos los cambios realizados en nuestra rama DEVELPMENT

Una vez hecho esto, la rama MASTER se encuentra actualizada.

Como se puede ver en la anterior imagen, para hacer el MERGE fue necesario usar una herramienta de GIT llamada FAST-FORWARD, lo que hace esta herramienta es realizar el MERGE de forma lineal, unificando los cambios realizados hasta el lugar desde donde se tomo la rama, esto no presenta mayor problema ya que la rama MASTER no siguió cambiando. Por el contrario, si la rama MASTER sufrió cambios por parte de otros programadores mientras se trabajó en las otras ramas, al momento de hacer MERGE va a ser necesario que se guarden los cambios y se registren mediante un COMMIT

EJEMPLO



Podemos ver que desde nuestra rama MASTER hacemos una actualización y para ello traemos los cambios de la rama DEVELPMENT, se debe resaltar que tanto la rama MASTER como la DEVELPMENT cambiaron antes de hacer el MERGE por lo que GIT nota esto y nos envía a la consola de GIT solicitando registrar el MERGE con un COMMIT.

IMPORTANTE

Dentro del espacio destinado para el COMMIT, escribimos un mensaje representativo del cambio realizado, una vez hecho esto presionamos la tecla ESC y posteriormente escribimos: wq de esta forma se va a registrar nuestro COMMIT

**GIT REBASE**

Este comando nos permite de igual forma hacer un MERGE con la diferencia de que, no se nos va a generar un COMMIT a partir de la fusión de ramas, el sistema va a fusionarse hasta convertirse en un historial plano.



Tal como se ve en la anterior imagen, primero nos ubicamos en la rama la cual queremos que se actualice, posteriormente hacemos uso del comando anteriormente mencionado, seguido del nombre de la rama de la cual traemos los cambios. Como se dijo anteriormente, el hacer uso de este comando, evita la creación de un COMMIT por lo que el historial es plano.

IMPORTANTE

Se recomienda usar REVASE cuando trabajamos desde un repositorio local y usar MERGE cuando lo hacemos desde un repositorio en la nube.

**Borrar todo el historial de COMMIT’S**

GIT nos permite eliminar todo el historial de COMMIT’S y para ello debemos hacer uso del siguiente comando.



Se resalta que el uso del anterior comando se debe usar de forma prudente ya que usarlo implica eliminar todo el historial dentro de nuestro fichero.

**GIT STASH**

Este comando permite al usuario el dejar el proceso dentro de GIT en un estado estacionario.

Si el usuario se encuentra trabajando con GIT sobre un fichero y aún no ha realizado COMMIT para guardar sus cambios finales, pero necesita cambiar a otro proyecto GIT, puede usar el comando GIT STASH lo cual permitirá que los cambios que ha realizado en el GIT inicial no se pierdan, de modo que podrá salir de su repositorio para trabajar en otro sin temor de perder los cambios iniciales, de este modo podrá volver al proyecto inicial cuando quiera.



Una vez realizados los respectivos cambios sobre el segundo proyecto, si queremos regresar al primero, podemos hacerlo usando;



Este comando permite regresar al lugar inicial de trabajo, los cambios del segundo proyecto se guardarán y los del primer proyecto continuarán estacionados hasta que se les haga su respectivo COMMIT.

NOTA: Los comandos STASH son apilables por lo que es posible tener varios al tiempo, para saber cuántos STASH tenemos, hacemos uso del comando **GIT STASH LIST.**

**GIT STASH CLEAR**

Usamos este comando después de utilizar GIT STASH APPLY y lo que hace es permitir al usuario eliminar el estado estacionario.

**Crear GIT STASH personalizado**

Si queremos crear un STASH sobre elementos que han sido modificados y sobre los cuales aún no hemos hecho un COMMIT y darle un nombre a gusto, podemos hacerlo



Como se ve en la anterior imagen, el comando GIT STASH SAVE seguido del nombre que queremos darle, nos permite poner el fichero y sus cambios en estado estacionario.

NOTA: Si no le damos un nombre especifico, el nombre que tendrá será el del último COMMIT realizado sobre el proyecto en cuestión.

**GIT CHERRY PICK**

Este es un comando que permite al usuario agregar los cambios realizados sobre una rama a otra rama, pero a partir de un COMMIT por lo que no estaríamos haciendo un MERGE



Tal como se ve en la anterior imagen, nos posicionamos en la rama donde queremos guardar el cambio y desde ahí hacemos uso del comando mencionado, seguido del código del COMMIT del cual queremos hacer la copia.

NOTA: Usamos este comando cuando no necesariamente queremos hacer un MERGE, sino que simplemente queremos agregar uno o más cambios a partir de determinado COMMIT.

IMPORTANTE

Este comando permite hacer copia de los cambios realizados en varios COMMITS y si lo que queremos es que los cambios de varios COMMIT’S se apliquen, podemos hacerlo.



**Guardar cambios entre dos COMMIT’S (Incluidos el COMMIT inicial y el final)**

****

Con el anterior comando lo que hacemos es, actualizar nuestra rama (Master en nuestro ejemplo) a partir del contenido de lo que tenemos dentro del intervalo de dos COMMIT’S incluyéndolos.

Si lo que queremos es no incluir el inicial y el final también podemos hacerlo



Tal como se ve en el anterior código, hemos hecho lo mismo que anteriormente, dentro del código lo que cambió es la eliminación de un carácter “^” y dentro del proceso, lo que hemos hecho es una copia de los COMMIT’S seleccionados y el intervalo entre ellos sin incluir los extremos especificados.

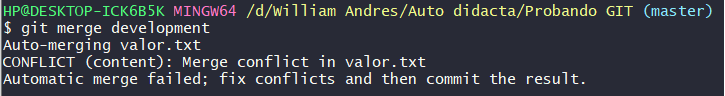
PENDIENTE: Parte final video 4 tema: HOOKS

Se continua VIDEO 5

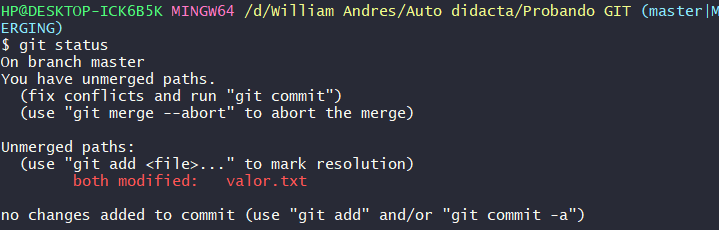
Pendiente VIDEO 5 tema: GITLAB

**Solucionar conflictos**

Puede darse el caso de que estemos trabajando dos o mas personas sobre un mismo fichero y que se hagan cambios diferentes por cada una de las personas involucradas, esto implica que, cuando se requiera hacer MERGE a la rama master, se encuentre un conflicto entre ambas.



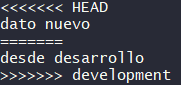
Tal como se ve en la imagen, hemos modificado el mismo archivo desde diferentes ramas, por lo que al momento de querer hacer un MERGE desde nuestra rama MASTER, el sistema nos notifica que tenemos un conflicto. Si estando en este punto hacemos uso del comando GIT STATUS podremos ver donde se presenta el problema



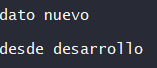
Para poder seleccionar con que cambio nos quedamos, hacemos uso del comando



El cual nos permite ingresar al archivo que presentó el conflicto



En la imagen se puede ver que se presentan dos ramas y los cambios realizados en cada una de las dos ramas, es aquí donde debemos escoger con que cambio queremos quedarnos, de modo que el otro sea borrado

 Borramos lo que no queremos incluir y dejamos lo que si nos interesa

Agregamos un nuevo COMMIT y su respectivo ADD



Una vez hecho esto, el conflicto ya fue solucionado

PENDIENTE VIDEO 6 0:10:00

**Borrar ramas**

GIT permite al usuario eliminar ramas que ya no use y para eso hacemos uso del comando



Si no se ha hecho un MERGE es posible que GIT nos arroje un error al momento de solicitar la eliminación de la rama, por esta razón podemos cambiar la “d” minúscula a una “D” mayúscula y de esta forma forzamos al sistema para que elimine la rama.

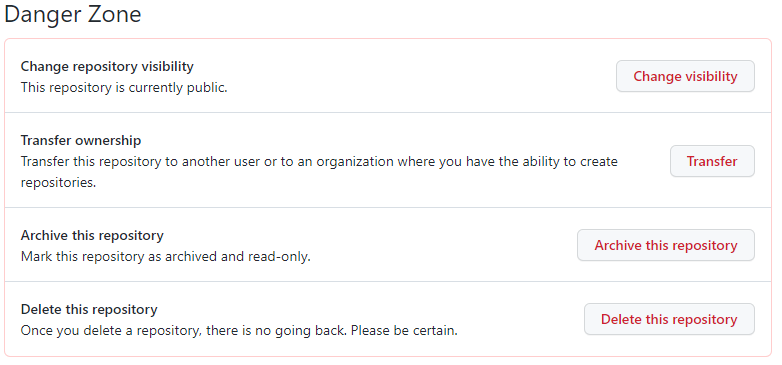


**Eliminar un repositorio**

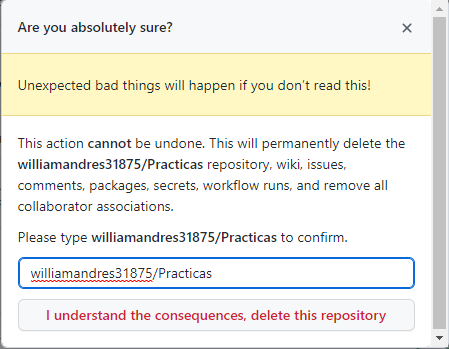
Para eliminar un repositorio, ingresamos al repositorio que queremos eliminar, posteriormente damos CLICK a la opción de SETTINGS



Después vamos a la parte de debajo de la página y seleccionamos la opción de eliminar repositorio.



Una vez seleccionemos la opción, que nos interesa, nos pedirá que ingresemos el nombre del usuario, seguido del nombre del repositorio.



Si ya ingresamos los datos solicitados, presionamos en la confirmación.