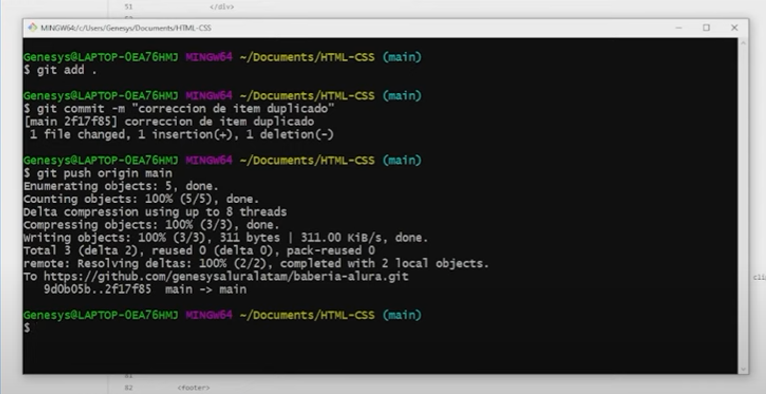
**GITHUB**

Para realizar un cambio en github se tiene que modificar primero en el EDITOR DE TEXTO (localmente), luego se tiene que agregar el comando “add”, después “commit” y agregar lo que se haya hecho de actualización o corrección al código de la forma mas explicita posible. Luego se agrega el comando “push origin main” para que se complete la actualización en github.

**VCS** (Sistema de Control de Versiones)

Nos deja organizar el trabajo en equipo, manteniendo los cambios en los archivos en un servidor específico para eso.

Git nos permite almacenar los cambios realizados en cada archivo en un servidor propio para eso. Toda la gestión de cambios la realiza Git y solo debemos preocuparnos por crear un código que funcione, y no por quién lo cambió antes.

Permite almacenamiento y acceso a un historial de cambios.

Interfaz de usuario gráfica

Descripción generada automáticamente Cada cambio que realices se registra en un historial y se puede ver y restaurar en cualquier momento.

Comandos:

“git -- versión” sirve para ver si se instaló bien git y que versión tenemos

“dir” sirve para buscar directorios, es un comando de linux

“ls” funciona para ver directorios también, es un comando de Linux

**Instalación en Linux**

Instalar Git en Linux es muy sencillo y en algunas distribuciones ni siquiera es necesario, ya que ya está instalado. Si este no es el caso con tu distribución, mira [aqui](https://git-scm.com/download/linux" \t "_blank) el comando requerido para instalarlo:

<https://git-scm.com/download/linux>

**Instalación en macOS**

Una captura de pantalla de un celular

Descripción generada automáticamente con confianza mediaLa instalación en macOS también es muy sencilla. Simplemente sigue las instrucciones de este link: <https://git-scm.com/download/mac>.

Para cambiar a otro disco duro solo se pone “cd /d” (la letra del disco duro) y apartir de ahí “ls” para ver el contenido de las carpetas y de “cd” para ingresar a las carpetas.

Texto

Descripción generada automáticamente

Ya que tenemos ubicada la carpeta utilizamos el comando “git init” para crear el repositorio de git. Una vez creado el repositorio se podrán ver las modificaciones o actualizaciones que realicen los demás usuarios. “git init” inicializa un repositorio en el directorio donde se ejecuta el comando. Desde este comando, Git podrá administrar los cambios realizados en los archivos.

Texto

Descripción generada automáticamente“git status” sirve para verificar que archivos están siendo modificados y monitoreados. En este caso no está siendo monitoreado ningún archivo (todos están en color rojo).

Texto

Descripción generada automáticamente“git add .” **agrega todos los archivos localizados en la carpeta para que sean monitoreados** por git**. Si quisiéramos que solo agregara un archivo** tendríamos que poner “git add <y el nombre del archivo>”. **Si quisiéramos que github dejara de monitorear algún** Texto

Descripción generada automáticamente**archivo** se tendría que poner “git rm --cached <y el nombre del archivo>”.

Cuando se requiere guardar el repositorio se usa el comando: git commit -m "aquí se agrega un mensaje breve de lo que se realizo".

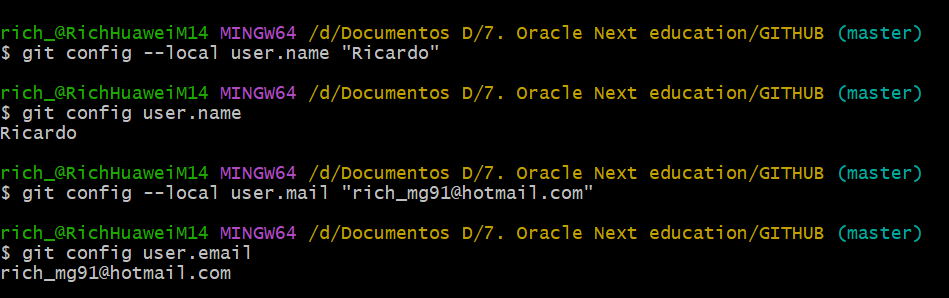
Antes de cualquier interacción con git, es necesario informar quién eres para que almacene correctamente los datos del autor de cada uno de los cambios en el código.

Se utilizan para agregar el nombre y mail del usuario que realiza la modificación:

git config --global user.email [you@example.com](mailto:you@example.com)

**git config --global user.name "Your Name"** significa que las configuraciones seran para todos los proyectos que haremos en nuestra computadora

las configuraciones **git config --local user.name "Your Name"** significan que las configuraciones serán solo para este repositorio y este proyecto.

**git config –local user.name** muestra el nombre del autor y lo mismo con el email

Al ejecutar el comando **git status,** recibimos información que puede no ser tan clara, especialmente cuando nos encontramos con términos como HEAD, working tree, index, etc.

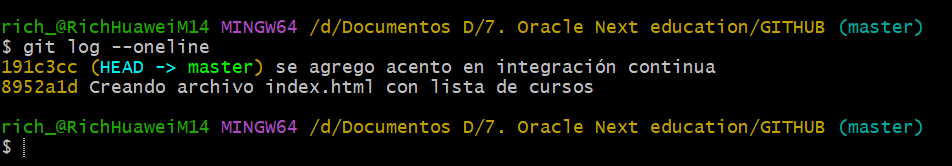
Solo para aclarar un poco, ya que entenderemos mejor cómo funciona Git durante el curso, aquí hay algunas definiciones interesantes:

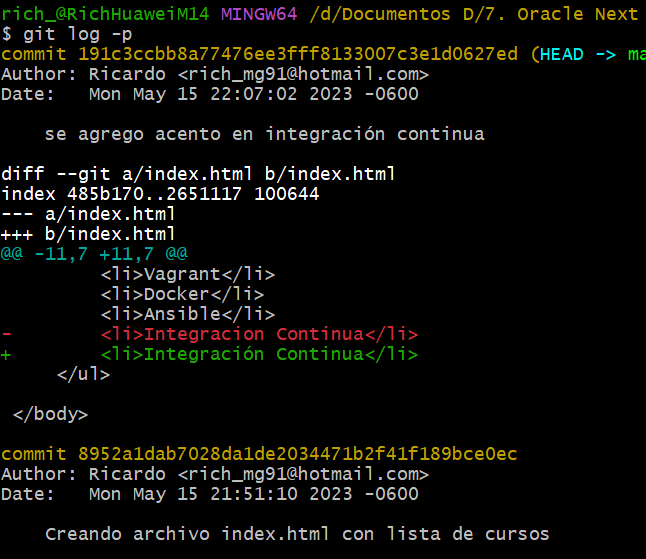
* HEAD: Estado actual de nuestro código, es decir, donde nos colocó Git
* Working tree: Lugar donde los archivos realmente están siendo almacenados
* index: Lugar donde Git almacena lo que será *commiteado*, es decir, la ubicación entre el *working tree* y el repositorio de Git en sí.

<https://git-scm.com/book/es/v2/Fundamentos-de-Git-Guardando-cambios-en-el-Repositorio>

**Una captura de pantalla de un celular

Descripción generada automáticamente con confianza media“git log”** muestra el historial de modificaciones o actualizaciones y quien lo realizo, también nos muestra un bash (un número serial extenso al lado de commit) para cada una de ellas.

** git log --oneline** muestra información mas resumida de las modificaciones realizadas.

**git log -p** muestra información detallada de los cambios que se realizaron, información extensa de los cambios o actualizaciones que se hicieron por el autor.

Para salir de esta pantalla se presiona “q” al final de todo el código.

**Texto

Descripción generada automáticamente**

Formas alternativas de mostrar los cambios o modificaciones realizados y por quien fueron hechos.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamentePara que git ignore una carpeta o archivo debemos crear un archivo dentro de la carpeta en nuestro EDD llamado **.gitignore**

Texto

Descripción generada automáticamenteDespues se tiene que agregar con **git add .gitignore** en el bash y luego se tiene que poner **git commit -m “el mensaje”** para dar la descripción de lo que se realizó en ese add.

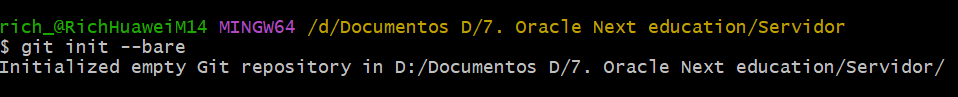
Nunca se debe realizar un commit (agregar un cambio y su descripción) de un código que no funcione.

Cuando se corrige un bug o en cada alteración significativa se debe realizar un commit

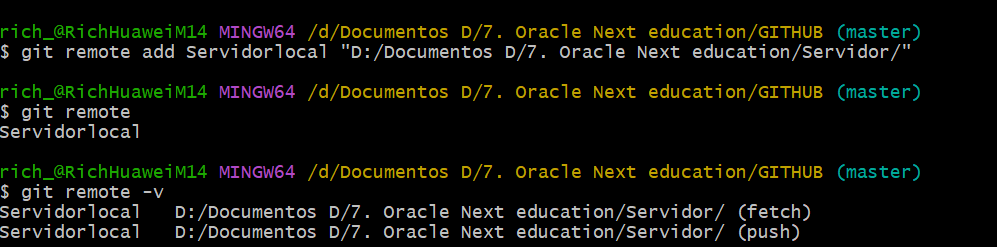
Deberíamos generar un *commit* siempre que nuestra base de código esté en un estado que nos gustaría recordar. Nunca deberíamos tener *commits* de código que no funcionen, pero tampoco es interesante dejar el *commit* solo al final de una función.

Texto

Descripción generada automáticamentePara crear una carpeta desde git se pone el comando **“mkdir**” y el **“nombre de la carpeta”,** para **volver atrás de una carpeta** se pone **“cd ..”.** Para volver 2 carpetas atrás se pone **“cd ../..”** y así sucesivamente.



**“git init --bare”** sirve para controlar y guardar solo las modificaciones, indica que solo contiene las modificaciones de los archivos esa carpeta(Servidor).  Con este comando creamos un repositorio que no tendrá la *working tree*, o sea, no contendrá una copia de nuestros archivos. Como el repositorio servirá solo como servidor, para que otros miembros del equipo sincronicen sus trabajos, disminuimos espacio de almacenamiento de esta forma.

Para agregar un servidor remoto a nuestro servidor local se usa el comando: **“git remote add” “seguido del nombre del servidor” y “la ruta de la carpeta”** (es importante remarcar que cuando hay espacios en el nombre de la carpeta se debe de agregar comillas “en/la/ruta” para que lo tome como un solo parámetro y funcione).

**“git remote”** muestra los servidores locales creados.

**“git remote -v”** muestra de donde se envían y donde se obtienen los datos (fetch lugar donde se obtienen los datos/push lugar donde se envían los datos).

Texto

Descripción generada automáticamentePara copiar el contenido de un servidor a otra carpeta

Se utiliza “**git clone /y/la/ruta/del repositorio” un espacio “nombre de la carpeta”** para crear una nueva carpeta.

Texto

Descripción generada automáticamentepara enviar los datos de un lugar a otro se pone **“git push \*nombredelservidor\* master”**

De esta forma enviamos las modificaciones de nuestra *Branch* master (hablaremos más sobre branches dentro de poco) hacia el repositorio remoto. Basta con substituir [repositorio] por el nombre que dimos al repositorio al agregarlo. Ahora, para traer los datos que están en el repositorio remoto, podemos utilizar git pull [repositorio] master.

**Texto

Descripción generada automáticamente**

git pone por defecto el nombre origin al agregar un repositorio, para cambiar ese nombre se pone el comando **“git remote rename origin \*nombre nuevo\*”**

**Texto

Descripción generada automáticamente“git pull \*nombreservidor\* master”** envia los archivos del servidor local a otra carpeta

Texto

Descripción generada automáticamentelo siguiente que se tiene que hacer para que los demás usuarios vean los cambios realizados en el proyecto. Compartir las modificaciones.

Texto

Descripción generada automáticamente(La carpeta “GITHUB” es un usuario). Lo que se tiene que hacer para que otro usuario vea los cambios realizados es ingresar a su carpeta y poner el comando **“git pull \*nombre del servidor\* master”**

Texto

Descripción generada automáticamente

**“git push [repositorio] master”**

De esta forma enviamos las modificaciones de nuestra *Branch* master hacia el repositorio remoto. Basta con substituir [repositorio] por el nombre que dimos al repositorio al agregarlo. Ahora, para traer los datos que están en el repositorio remoto, podemos utilizar git pull [repositorio] master.

GITHUB.com crear repositorios git en internet

Con GitHub, podemos tener repositorios remotos públicos y privados gratuitos para almacenar y compartir el código de nuestros proyectos.

**Creación de token de seguridad en GitHub**

https://www.aluracursos.com/blog/exigendia-autenticacion-por-token

este cambio en el uso de tokens para autenticación es muy interesante, ya que puede ofrecer un mayor control y seguridad. Puede realizar todo el proceso mencionado anteriormente tantas veces como sea necesario, así como configurar los permisos y el tiempo de expiración de cada token generado de la manera que consideres mejor.

Texto

Descripción generada automáticamenteEn caso de fallas de seguridad, por ejemplo, la persona estará limitada a los permisos otorgados al token y su cuenta de GitHub seguirá a salvo. Además, si esto sucede, elimina tu token yendo a GitHub, luego a Settings, Developer settings y Personal access tokens, el mismo procedimiento para generar el token, pero haz clic en Delete:

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamenteen github.com se crea un repositorio en la esquina superior derecha, con el signo de +.

Texto

Descripción generada automáticamenteLuego se agregan esos comandos generados por github.com en la terminal bash de git de nuestro Sistema Operativo.

**Acceso Remoto a servidores con SSH**

En esta publicación vimos cómo podemos acceder a servidores Linux con Windows, Linux o Android. Además, vimos que este tipo de acceso requiere ciertos cuidados, especialmente cuando los servidores están en producción.

<https://www.aluracursos.com/blog/ssh-acceso-remoto-a-servidores>

Texto

Descripción generada automáticamenteLas Branches se usan para tener alguna forma de separar nuestras ramas de desarrollo de forma que podamos saber exactamente dónde estamos modificando y no interfiera en el código compartido.

**“git Branch [NombreDelNuevoBranch]”** agrega un nuevo Branch

**“git branch”** muestra los branchs existentes

**“git checkout [Nombre del Branch al que queremos cambiar]”** cambia de una Branch a otra

**Texto

Descripción generada automáticamente“git branch -b [nombre del nuevo branch]”** crea una nueva rama o Branch y cambia automáticamente a ella sin tener que agregar el comando “git checkout” para cambiar de Branch.

**“git add .”** agrega todos los archivos al repositorio, ya no se tiene que poner el nombre del archivo

Las *branches* ("ramas") se utilizan para desarrollar funcionalidades aisladas entre sí. La *branch* master es la *branch* "predeterminada" cuando creas un repositorio.

Es interesante separar el desarrollo de funcionalidades en diferentes *branches*, para que los cambios en el código de una no influyan en el funcionamiento de otra.

**Como llevar datos de una Branch a otra**, de una branch a master, en la lista master siempre va el trabajo que ya esté terminado y sin errores.

**Texto

Descripción generada automáticamente“git merge \*nombre de la Branch rama**\*” lleva las modificaciones de una Branch a otra, por ejemplo, en este caso de titulo a master.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

De esta forma colocaremos el HEAD en la *Branch* master, o sea, haremos que nuestro código esté en el estado que lo dejamos con el último commit en master. Después, uniremos el trabajo de la *Branch* titulo con la *Branch* actual (master).

Texto

Descripción generada automáticamentedespués de poner el comando de arriba se aparece esta pantalla, la cual puede ser editada, para salir se utiliza el comando **“:x”** y enter.

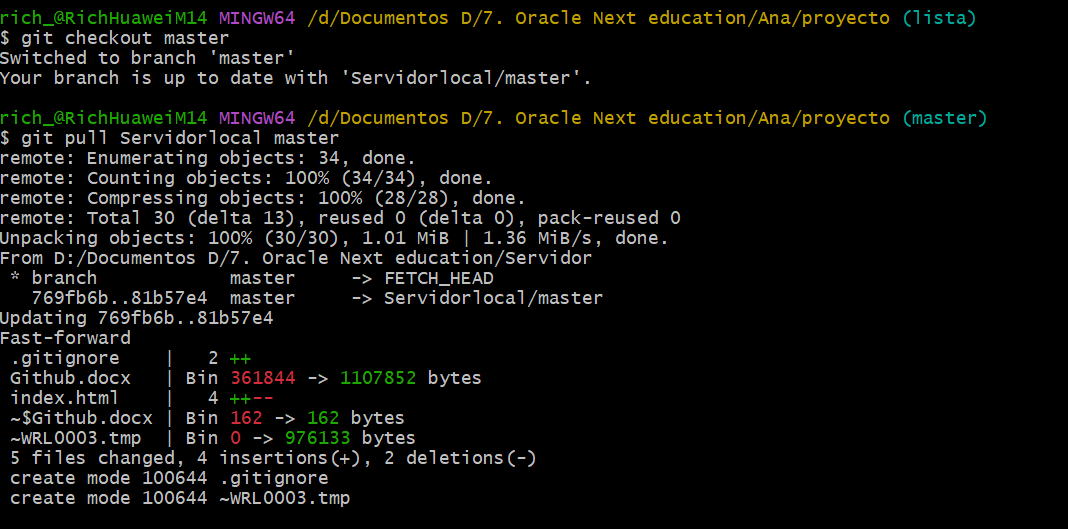
**“Git rebase \*nombre de la branch\*”** junta todos los commits de una branch en una sola línea para no generar commits extra cuando agregamos el comando git merge.

**Una captura de pantalla de un celular con texto e imagen

Descripción generada automáticamente con confianza media“git log --graph”** muestra los commits de una forma mas organizada.

Texto

Descripción generada automáticamenteUna vez corregidos los datos se suben al Servidorlocal con push



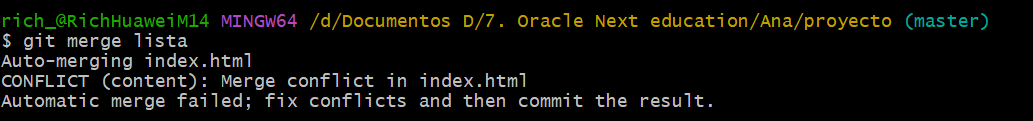
Ahora desde otro usuario se pasa a la Branch master y se hace un git pull para obtener todas las modificaciones que realizo el usuario principal “GITHUB”

“Ctrl + c” sirve para parar la línea por si nos equivocamos y así escribirla de nuevo corregida.

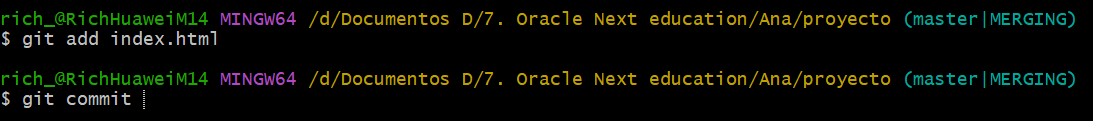
Ya sabemos cómo traer el trabajo de otra *Branch* y unir con la *Branch* actual. Conocemos dos formas de hacer esto: merge y rebase. ¿cuál es la diferencia entre los comandos rebase y merge?

El**merge** junta los trabajos y genera un merge commit. El **rebase** aplica los *commits* de otra Branch en la *Branch* actual.

Con esto evitamos los *commits* de *merge*. Hay una larga discusión sobre lo que es “mejor”: rebase o merge; Estudie, busque, y genere sus propias conclusiones. Acá tienen un artículo (de miles de otros) que hablan sobre el asunto: <https://medium.com/datadriveninvestor/git-rebase-vs-merge-cc5199edd77c>.

Texto

Descripción generada automáticamenteesto aparece en MVS (el EDD) son las líneas que se tienen que corregir o aceptar para que se pueda hacer el merge sin problemas.

despues se agrega el index.html ya corregido para que el bash sepa que ya fue solucionado el problema, y despues un git commit para que proceda el merge, después se envía al servidorlocal con un git push.