 ¿Cómo sabés si primero hay que agarrar las versiones de ellos y luego tenemos que colocar nuestro código encima? ¿Qué parte deberíamos modificar con nuestro código y qué parte deberían quedar en el código de nuestros compañeros? Eso es un problema. Para esto, una posible solución es colocar un servidor específico para enviar nuestras alteraciones, entonces cuando finalizamos una modificación, la enviamos a ese servidor. Y si otra persona quiere mandar una modificación de su código, necesitamos una herramienta que gestione el código y diga: “Esa versión en la que basaste tu código no es la más actualizada. Hay una versión nueva”. Y así, no deja que se envíe directamente ese archivo. Lo que pide es que antes de enviar las modificaciones, esa persona tiene que bajar la versión más nueva, para que ahí pueda modificar solo la parte de código que tuvo cambios.

Y luego sí, pueda enviar ese código al servidor. A eso es a lo que se llama control de versión. Si tenemos diferentes versiones del código, necesitamos un sistema que controle esas versiones y para eso sirve Git. Git no es el único sistema que existe de control de versiones. Existe CVS, SVN, Mercurial.

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

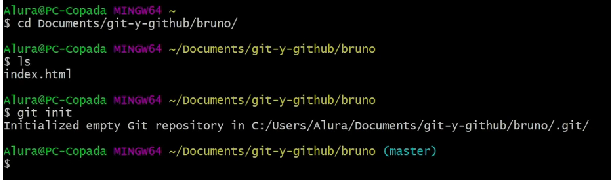
Para saber si está bien instalado, nosotros lo que vamos a hacer es escribir: git, espacio, guión, guión, version, damos enter y allí vemos que nos muestra la versión de Git.

Acá quería mostrarles lo que decía antes de utilizar comandos de Linux dentro de Windows.

Que es, por ejemplo en Windows, normalmente, para ver un directorio y sus carpetas, nosotros colocamos dir, enter y acá nos muestra todos los archivos y documentos y carpetas. En Linux, normalmente se utiliza el comando ls y vean que acá ls también funciona. Entonces nos permite utilizar comandos de Linux dentro de Windows.

si yo hago un ls, acá puedo ver todos los archivos que tengo dentro de esa carpeta.

Si nosotros reescribimos cd vamos a cambiar de carpeta, independientemente de dónde estemos.



En la terminal (o **Git Bash**, en Windows) navega hasta la carpeta recién creada (usa el comando cd para navegar entre carpetas);

En la carpeta del proyecto, ejecuta el comando git init para inicializar un repositorio Git;

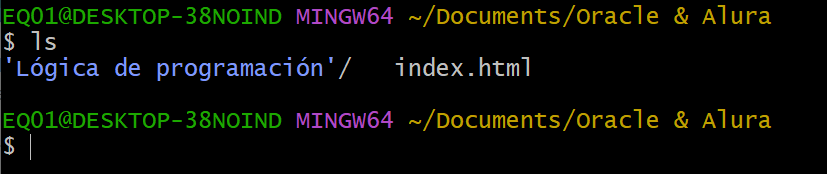
Ejecuta el comando git status para asegurarte de que estás en un repositorio de Git.

En esta aula aprendimos:

* Qué son (y para qué sirven) los sistemas de control de versiones y cómo pueden ayudar a nuestro flujo de desarrollo
  + Nos ayudan a mantener un historial de cambios;
  + Nos ayudan a tener control sobre cada cambio en el código;
  + Nos ayudan a que un cambio de una persona no influya en el cambio realizado por otra;
  + Etc.
* Qué es Git y cómo instalarlo
* Que con el comando git init pudimos crear un repositorio Git;
* Cómo analizar el estado de nuestro repositorio usando el comando git status.

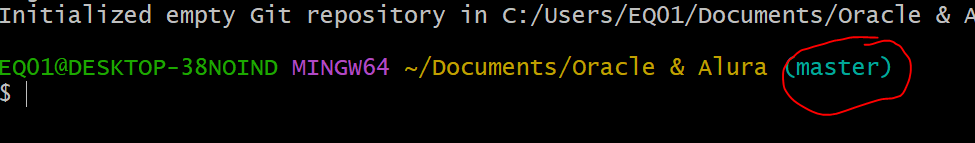
luego de que haces cd Documents/ y que ls te muestre solo esas dos carpetas pon **pwd**. El pwd te dice cual es la ruta en que te encuentras y luego desde tu explorador de archivos sigue la ruta. Así podrás aasegurate que estás en la carpeta Documents que tiene tus archivos o no.

Una forma mas sencilla de hacerlo es colocarte sobre tu proyecto, hacer clic derecho y elegir la opcion -GIT BASH HERE- y te lo abre directamente, por lo menos a mi me resulto ya que no pude hacerlo como lo indica el instructor en el video, luego le das LS:

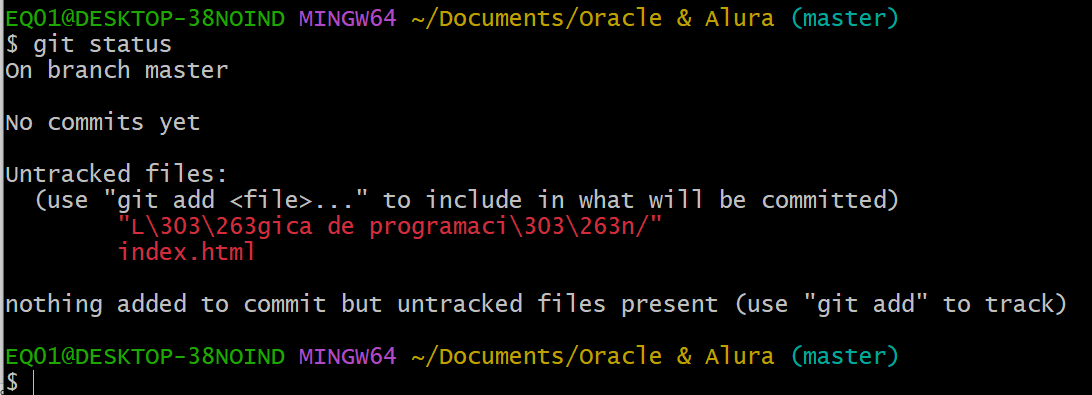


 Esta carpeta va a ser mi repositorio donde quiero que Git cree ese repositorio. Para eso voy a utilizar el comando git init, damos enter y acá lo que hace este git init es decir: "Acá hay un repositorio". Con esto ya tenemos un repositorio inicializado.

atención que cuando hice git init se agregó automáticamente una palabra luego del directorio, que esto de aquí significa que es una branch. Está en la branch master.



Un comando que nos sirve para saber los cambios realizados y estado en el que están los archivos dentro del repositorio es git status, damos enter y acá vemos que comienza diciendo que está en la rama master, eso lo vamos a ver después. Dice que no hay commits aún.



Y después dice: “archivos que no están siendo monitoreados”. En este caso dicen que aún no están registrando los cambios en el archivo index.html. ¿Cómo hacemos para decirle a Git que comience a rastrear un archivo? Lo que tenemos que hacer es ese Git add y el nombre del archivo.

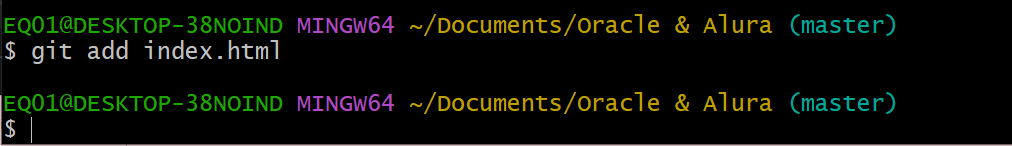
ntonces todo lo que hay dentro de esta carpeta, a menos que informemos lo contrario, Git va a prestar atención a qué es lo que ocurra ahí adentro y analizar en el futuro y si fuera necesario, va a guardar esas informaciones. Genial. Entonces Git ya ve nuestros archivos, pero como dice acá, aún no está monitoreándolos.

¿Cómo hacemos que Git vea una carpeta en particular como un repositorio y observe los cambios en sus archivos? A través del comando git init.

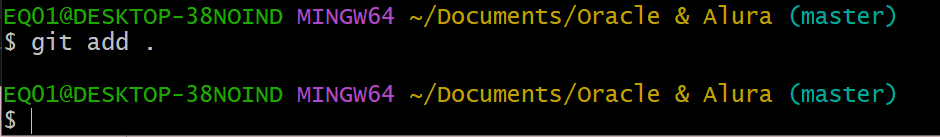
¡Opción correcta! git init inicializa un repositorio en el directorio donde se ejecuta el comando. Desde este comando, Git podrá administrar los cambios realizados en los archivos.

si nosotros colocamos Git status y enter ahí lo que vemos es que Git ya reconoce nuestra carpeta como repositorio pero está viendo que nuestros archivos aún no son trackeados o los cambios que hacemos a nuestros archivos no están siendo monitoreados por Git.

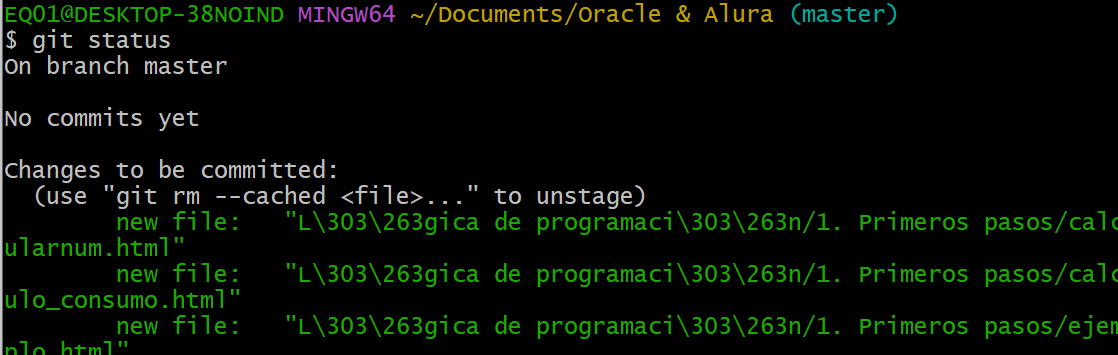
Entonces lo primero que necesitamos hacer para guardar una modificación de un archivo en Git, es hacer que Git monitoree los cambios de ese archivo. Como no está siendo monitoreado ese archivo que es aquí el index.html, necesitamos usar el comando git add, espacio, y el nombre del archivo, index.html en este caso.



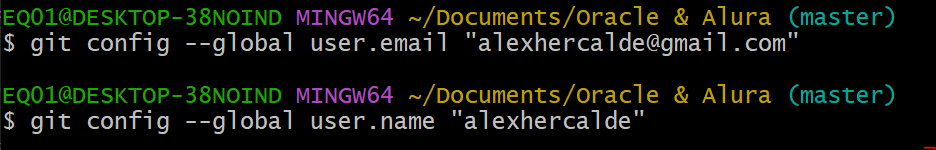
Pero, ¿y si tuviera varios archivos? ¿Tengo que ir poniendo cada uno de los nombres de mis archivos? No. Con colocar git add punto, lo que hacemos es agregar todos los archivos que están dentro de esa carpeta y hacemos que Git monitoree todo esos archivos. En este caso voy a usar ese comando git add punto, presiono enter y cuando doy enter parece como si nada ocurrió.



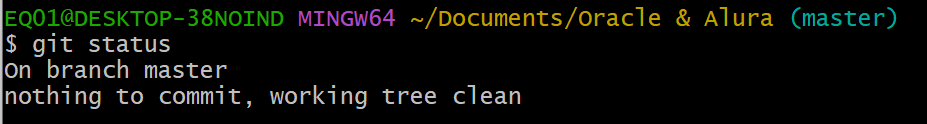
Pero si por ejemplo yo escribo git status, doy enter, acá vemos que ahora aparece en verde nuestro archivo. Esto significa que ahora Git ya está monitoreando ese archivo. Miren el mensaje. Cambios a ser commiteados o enviados o guardados. ¿Y si no queremos monitorear más ese archivo, este index.html? Lo que podemos hacer es usar este comando de acá: “git rm --cached" espacio, el nombre del archivo.



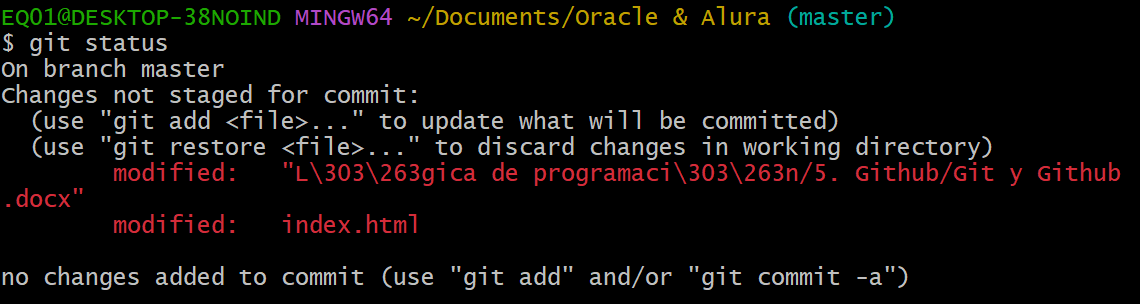
Entonces podemos crear un checkpoint o un punto dentro de la historia de ese archivo para decir: "Acá ocurrió alguna cosa en este archivo, alguna modificación." Y este punto se llama commit. En ese commit vamos a colocar nuestro archivo y un mensaje indicando lo que se modificó en ese archivo.



Si ejecutamos git status nos dice que estamos en la branch master, eso no importa y que no hay nada para commitear. Dice que no hay nada para commitear pero no significa que no existan commits ahí dentro. Git ya sabe que hay un repositorio de Git, sabe que existe commits y no tengo ninguna modificación.

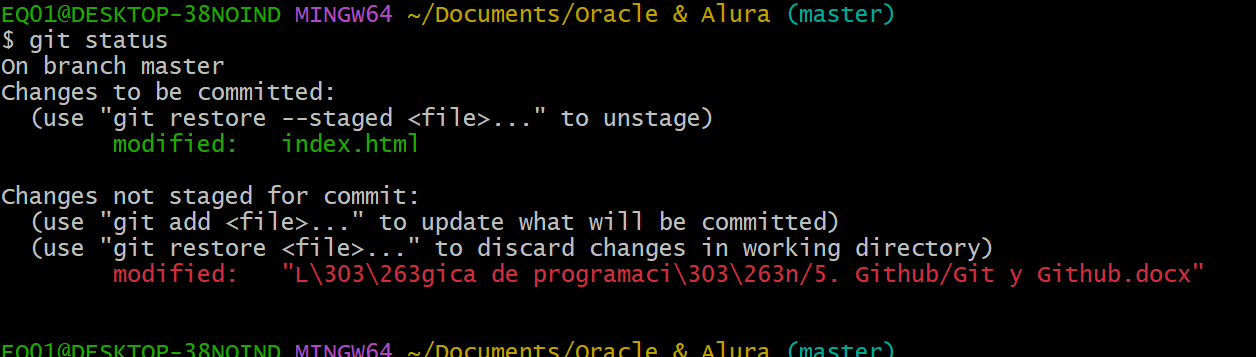


Habiendo guardado nuestro archivo, nosotros lo que vamos a hacer ahora es colocar git status para ver qué es lo que está viendo Git. Colocamos eso y vemos acá que nos indica que fue modificado un archivo.



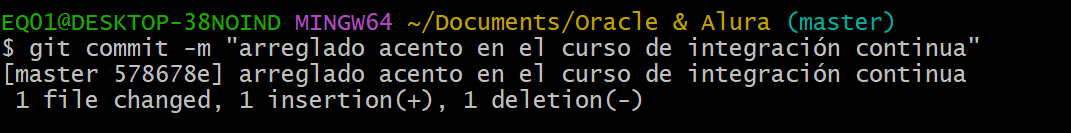
Entonces, para agregar esa modificación hacemos git add, espacio y el nombre del archivo. O pueden colocar punto si solamente tienen este archivo también dentro de su carpeta o si quieren agregar todos los archivos que existan ahí adentro.

Si damos git status de nuevo nos dice que fue agregado ese index.html y que está listo para ser commiteado.



Ahora ponemos git commit, espacio, -m, espacio, abro comillas, "Arreglado acento en el curso de integración continua". Bien. Pueden colocar acentos y símbolos dentro de ese mensaje.

Damos enter y bien, ya hemos hecho nuestro commit, nos dice que un archivo fue modificado y se agregó una línea y se eliminó una línea. Esa línea que se agregó y esa línea que se modificó es la misma línea. Git lo toma como que se eliminó una línea y agregamos una nueva encima.



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.

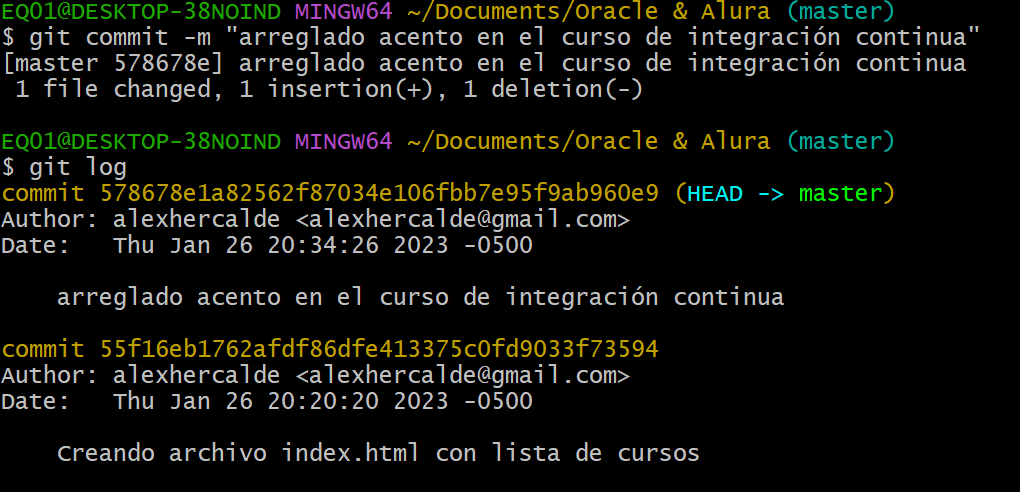
Además de eso, los posibles estados de nuestros archivos son explicados con detalle en este link: <https://git-scm.com/book/es/v2/Fundamentos-de-Git-Guardando-cambios-en-el-Repositorio>.

El histórico de modificaciones se consulta a través de git log.

Rápidamente, primero vemos un hash del commit. ¿Qué es un hash? Un hash es una identificación única. Un ID para ese commit.

No existen dos commits con el mismo hash. Con esto podemos hacer algunas manipulaciones que vamos a ver más adelante. Segundo, branch. Acá vemos dos palabras.

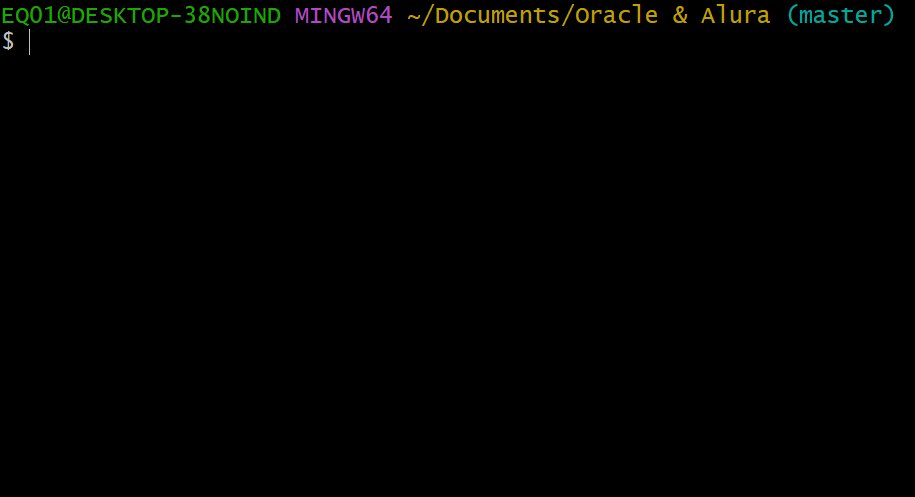
Primero vemos la palabra HEAD. HEAD nos indica dónde estamos parados en este momento en el proyecto. Vemos también, eso de aquí, el HEAD no es tan importante ahora, en este momento. Master es la branch o la rama.



Luego tenemos el autor, que acá aparece mi nombre y mi email. Luego tenemos la fecha en que fue hecho ese commit y luego el mensaje que hemos colocado.

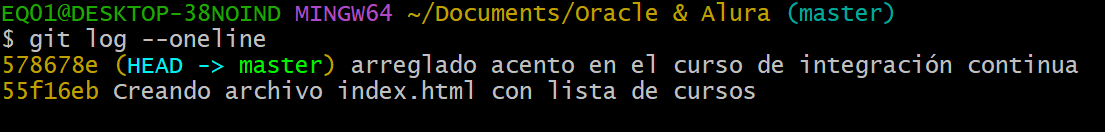
¿Cómo hacer para configurar mi email y mi nombre? Lo que tenemos que hacer es colocar git config y acá existen dos parámetros para colocar. Puede ser --local, que esto significa que las configuraciones que vamos a crear van a ser solo para este repositorio en este proyecto y existe la configuración --global, que esto es para todos los proyectos que vamos a hacer de ahora en adelante en nuestra computadora.

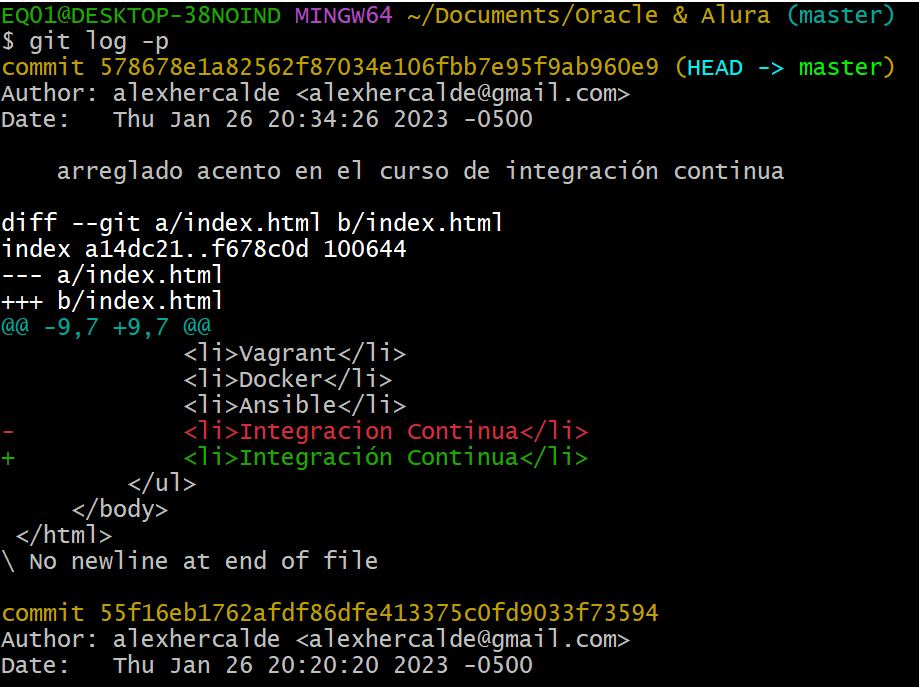
Voy a hacer un clear aquí, para limpiar la pantalla para que no tenga tanta información.



¿existe otra forma o formato para ver ese git log? Sí, existen infinidad de formas de ver esas informaciones. Una muy común es visualizar todos los commits donde cada uno de ellos ocupen solo una línea.

vamos a utilizar el comando git, espacio, lógica, espacio, --oneline. Una línea en inglés. Con esto vemos que primero al principio tenemos un resumen del hash, los primeros caracteres del hash, tenemos en cuál branch fue ejecutado y tenemos acá la información que nosotros colocamos en el mensaje, en el commit.



¿Pero si en vez de ver menos, quiero ver más? ¿Qué tal si quiero ver las modificaciones de los archivos? Lo podemos hacer usando git log, espacio, -p. Enter y acá vemos, este fue nuestro primer commit, nos dice todas las informaciones que ya habíamos visto anteriormente, pero además de eso nos hace lo que se llama un div que nos va diciendo cuáles fueron las modificaciones que fueron realizadas en ese commit.

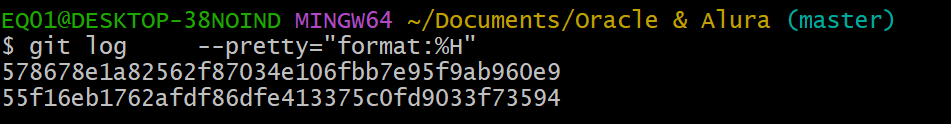
En este caso, el último commit lo que nos dice fue que se quitó esa línea y se agregó esa línea con acento. Si damos acá enter para continuar hacia abajo vemos que el commit anterior a ese último es este que vemos que esta todo en verde. Todo en verde significa que se agregaron todas esas líneas.

simplemente ese archivo fue el primer archivo que colocamos dentro de nuestro Git, entonces todas las líneas son nuevas para Git. Para salir de esa parte del git log -p pueden hacer :q, así, voy a mostrar acá de nuevo: git log, espacio -p, enter, damos enter, enter, enter, enter y acá podemos hacer dos puntos y la letra Q y salimos de ese editor.

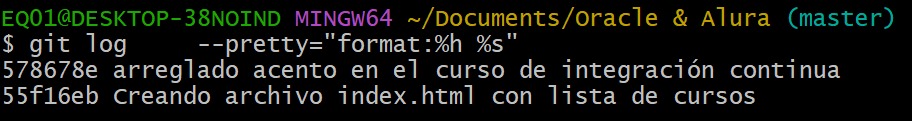
Devhints.io/git-log

podemos colocar un pretty, que es una forma más customizada, más de acuerdo a lo que nosotros queremos mostrar. En este caso vamos a utilizar este pretty para dar un ejemplo: clic derecho + copiar, volvemos a Git Bash, entonces colocamos git, espacio, log, espacio, y ahora clic derecho + pegar.

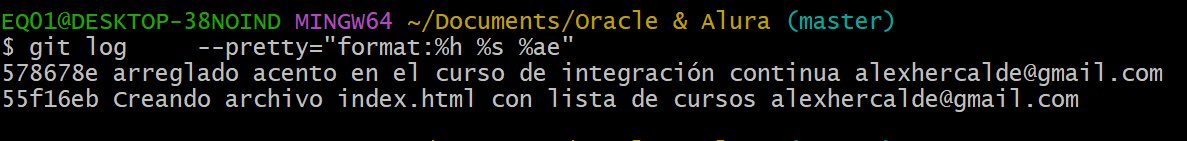
solamente estoy viendo los hash de cada uno de los commits:



Y acá vemos con el %h vemos un commit abreviado, para que no ocupe tanto espacio en la pantalla.Un %s nos pone por ejemplo cuál fue el mensaje. Entonces vamos a hacer eso. Vamos a colocar ese H y ese S. Vamos a dar un clear para limpiar un poco la pantalla, usamos la flechita de arriba del teclado para ir a los comandos anteriores que hemos colocado. Voy a colocar H, espacio. Voy a colocar ahora %s, doy enter y acá vemos el hash abreviado y el mensaje.



Ahora, ¿qué tal si quiero agregar el email? El email de la persona por ejemplo. Bueno, %+A+E. Vamos a usar la flechita para arriba y acá agregamos %ae. Damos enter y acá vemos que luego del mensaje aparece el email.



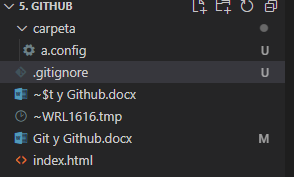
 ¿Qué ocurre si yo estos archivos no los quiero realmente en mi proyecto? Entonces, ¿cómo hago para que Git ignore esos archivos? Eso lo vamos a hacer con Git ignore.

¡Opción correcta! Así todos los commits ejecutados en este repositorio serán atribuidos a la persona con el nombre **Nombre de la persona**. Para más detalles y otras configuraciones posibles (incluso algunas más avanzadas), pueden visitar este link: <https://git-scm.com/book/en/v2/Customizing-Git-Git-Configuration>.

Como vimos en el último video, podemos ver el historial de cambios de nuestro proyecto de manera muy sencilla, usando el comando git log.

Aunque es fácil, este comando es muy poderoso. Ejecuta git log --help y ve algunas de las posibles opciones. Para obtener algunos ejemplos más fáciles de entender, puedes buscar sobre git log o echar un vistazo a este link: <https://devhints.io/git-log>.

Entonces, ¿cómo puedo decir: "Git, está bien que esa carpeta esté ahí pero yo no la quiero monitorear realmente"? Bueno, lo que podemos hacer es crear un archivo especial de Git llamado Git ignore



En este caso, nosotros tenemos una carpeta, entonces lo que vamos a hacer es colocar carpeta/, esa barra acá identifica que son carpetas. Y si yo tuviera un archivo que no quiero colocar, por ejemplo este a.config si yo lo coloco dentro de Bruno por ejemplo, yo no quiero que gitignore tome ese a.config, entonces puedo colocar a.config, doy enter y ahí con eso damos, guardamos, salvamos.

