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.

En el video no hice mucho hincapié en esto, pero para que puedas hacer esto en tu computadora, si estás comenzando a usar git ahora, simplemente escribe los siguientes comandos (estando en la carpeta del repositorio de git):

git config --local user.name "Tu nombre aquí"

git config --local user.email [Tu@email.aqui](mailto:Tu@email.aqui)

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>.

Créame, aunque parezca confuso ahora, ¡durante el curso todo tendrá mucho más sentido! :-D

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

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>.

En esta aula aprendimos:

* Que un commit es la forma de guardar un estado o versión de nuestro código;
* Cómo agregar archivos para *commitear* con git add;
* Cómo *commitear* archivos usando el comando git commit;
* Cómo verificar el historial de *commits*, a través de git log y algunas de sus opciones:
  + git log --oneline
  + git log -p
  + git log --pretty="parámetros de formato"
* Cómo hacer que Git no monitoree archivos, a través de **.gitignore**
* Que no debemos realizar *commits*, es decir, guardar un estado, de nuestra aplicación que no esté funcionando.

En esta aula aprendimos:

* Que una *branch* (o rama) es una línea de *commits* separada, y que se puede usar para desarrollar funcionalidades independientes;
* Que con *branches* separadas, podemos evitar que el código de una funcionalidad interfiera con otra;
* Cómo traer el trabajo realizado en una *branch* a otra *branch*, como la master, usando el comando git merge;
* Que git merge genera un nuevo *commit*, informando que hubo una mezcla entre dos *branches*;
* Cómo traer *commits* de una *branch* a otra con git rebase
* Que git rebase no genera un *commit* de merge, lo que simplifica nuestro *log*;
* Cómo presenta Git los conflictos;
* Cómo resolver conflictos y conservar solo los cambios deseados con Git.