Contenido

[Introducción a GIT 2](#_Toc120790554)

[Que es? 2](#_Toc120790555)

[Primeros pasos 3](#_Toc120790556)

[Primer Proyecto de Git 3](#_Toc120790557)

[GIT INIT 4](#_Toc120790558)

[GIT STATUS 4](#_Toc120790559)

[GIT ADD 5](#_Toc120790560)

[GIT COMMIT 5](#_Toc120790561)

[GIT LOG 7](#_Toc120790562)

[GIT CHECKOUT 8](#_Toc120790563)

[GIT PUSH 18](#_Toc120790564)

# Introducción a GIT

By código facilito -> <https://www.youtube.com/watch?v=zH3I1DZNovk&list=PL9xYXqvLX2kMUrXTvDY6GI2hgacfy0rId>

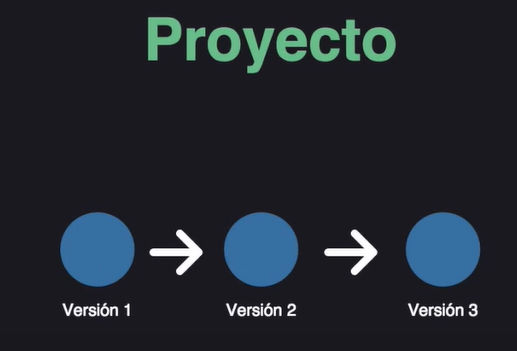
Nota: Se utilizo lo que es el proyecto de la carpeta Git/GardenProject de mi computadora, donde trabaje desde el Windows con el git bash para la realización de este documento.

## Que es?

Es un sistema de control de versiones creado por Linus Torvalds.

Este sistema de control de versiones nos registra los cambios realizados en un archivo o conjunto de archivos.

Ejemplo, tenemos un proyecto con varios archivos, el cual seria la versión 1 de nuestro proyecto y con forma más cambios o archivos entren se ira incrementando las versiones.



Git monitorea todo lo que hacemos en nuestro proyecto va ver nuestras líneas de código.

Y con git podemos ver si tenemos algún error, es decir, si en la versión 3 tenemos un error, podemos regresarnos a la versión 2 para poder entender que es lo que salió mal o en donde tenemos que realizar el cambio.

Otra ventaja es la organización, con los commits.

Git trabajo con 3 estados:

1. **Working directory:** Aquí es donde editamos y trabajamos con nuestro proyecto, cuando abrimos el editor de código y realizamos un nuevo archivo.
2. **Staging área:** aquí es donde escogmos que archivos están listos para pasar al tercer estado, al igual que decidimos que archivos no están listos por el momento. Ejemplo, tenemos un archivo donde aun no esta listo y otro que si, entonces hacemos que el archivo listo pase al siguiente estado.
3. Repository: Aquí es el registro de todo nuestro proyecto. Esto se registra en algo llamado “commit”.



# Primeros pasos

Instalar git esta tanto para Windows, Linux y Mac. En Windows tenemos el git bash que es una terminal cmd donde podemos ejecutar comandos de git.

Ahora se van a realizar unas configuraciones básicas; Con el comando:

1. git config - -global user.name “Miriam-Janet”

2. git config - -global user.email “mjgs290919980@gmail.com”

Para visualizer colocamos:

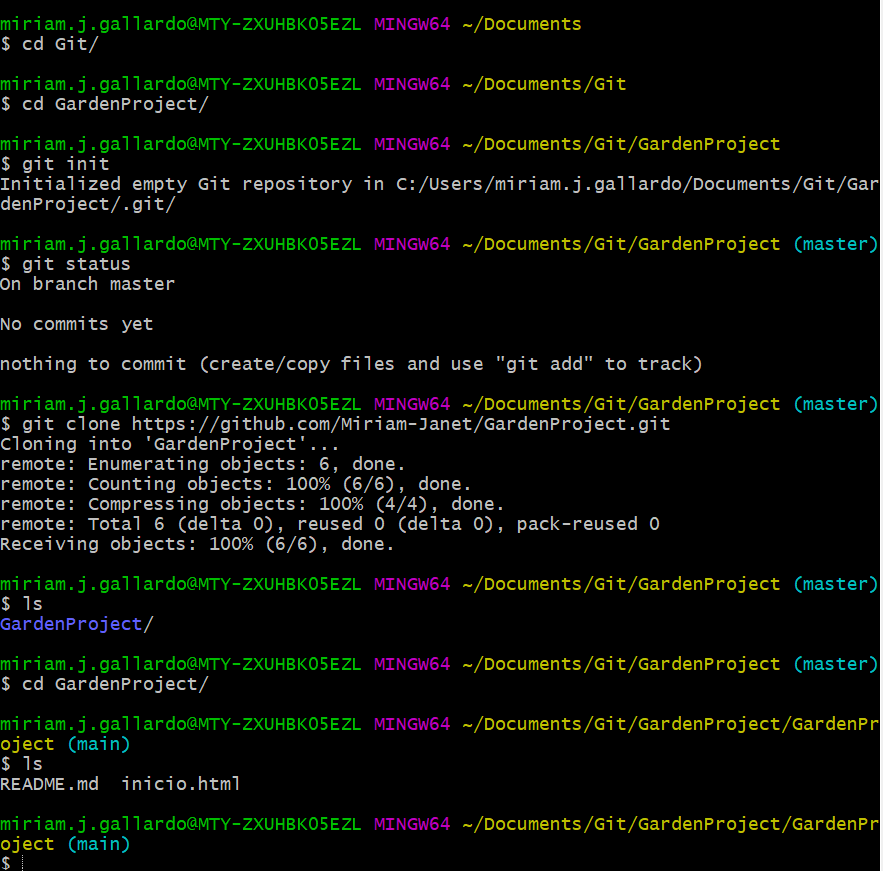
3. git config --global --list

Ahora configuramos los colores, esto es opcional:

4. git config - -global color.ul true

# Primer Proyecto de Git

Primeramente, abrimos lo que es git bash, ya sea de Windows, si es de Linux desde su terminal; ya en el git bash, se debe de ingresar a lo que en nuestra carpeta el proyecto.



## GIT INIT

Ya dentro de nuestra carpeta donde estaremos en el proyecto se va a colocar el siguiente comando:

1. git init

Este comando marcara el inicio de nuestro proyecto, aquí le decimos a Git que empiece a monitorear todos nuestros cambios. Siempre utilizarlo al comenzar un proyecto de git

## GIT STATUS

Ahora seguido de colocar ese comando tenemos el siguiente que es:

1. git status

Este nos va a mostrar el estado de nuestro proyecto. Ya sea que se requiera de agregar los archivos o cuales ya han sido agregados.

## GIT ADD

Ahora seguido de este si tenemos archivos por agregar al repositorio se va a colocar lo que es:

1. git add

Con este comando decidimos que archivos están listos para agregarse. Ejemplo: git add index.txt

Y para agregar todo lo que tenemos colocamos

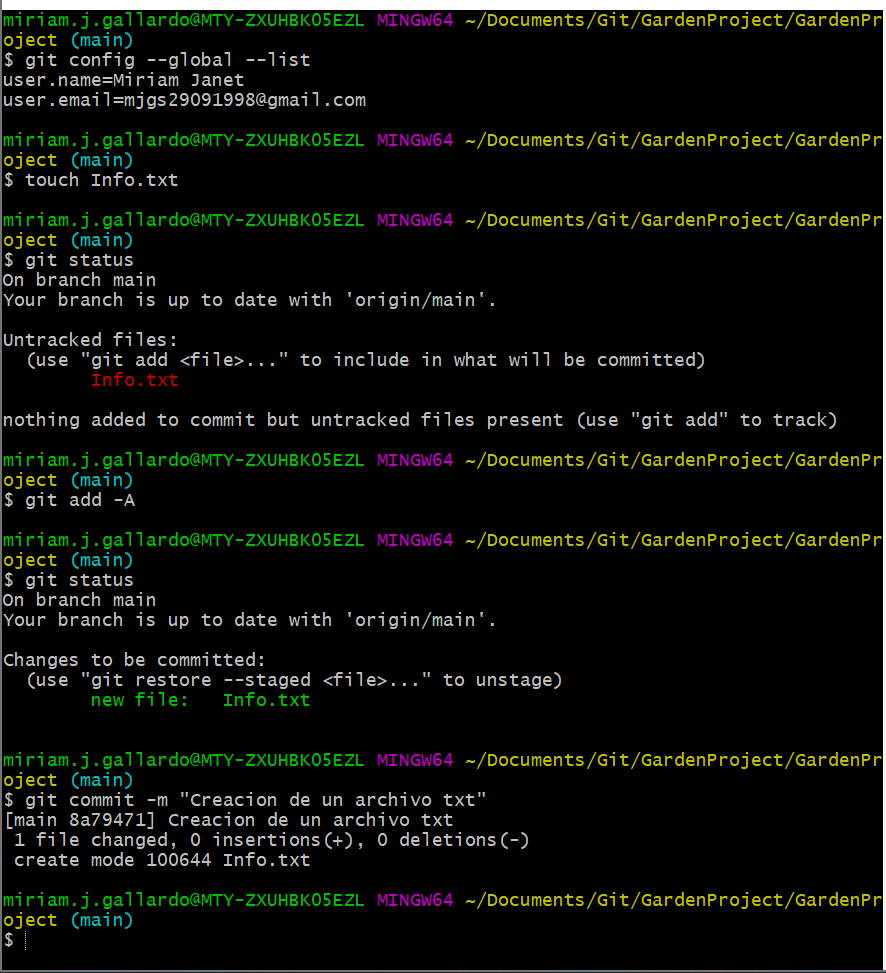
* git add -A

## GIT COMMIT

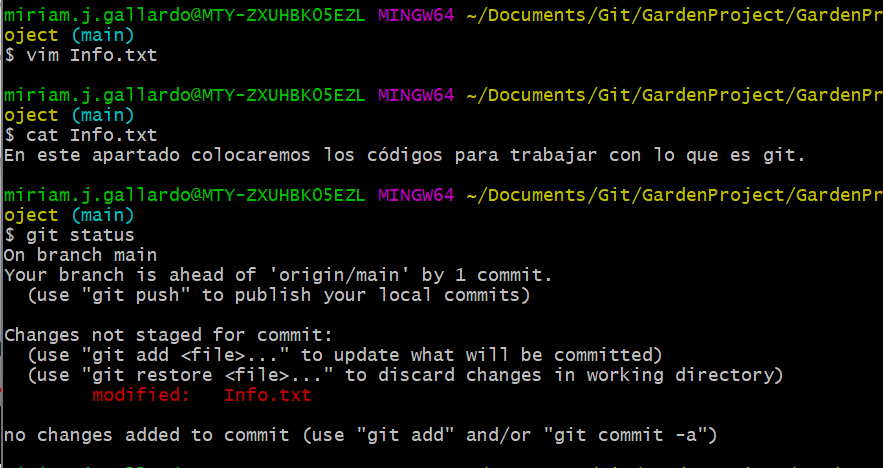
Ahora ya que se agrego lo que se coloca es el siguiente comando para poder identificarlo:

1. git commit -m “Mensaje”

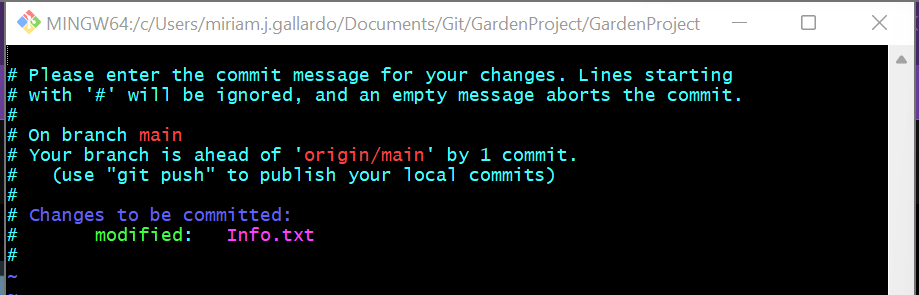
lo que va a hacer este comando es guardar los cambios con un mensaje para identificarlos. En los mensajes debemos colocar algo muy especifico



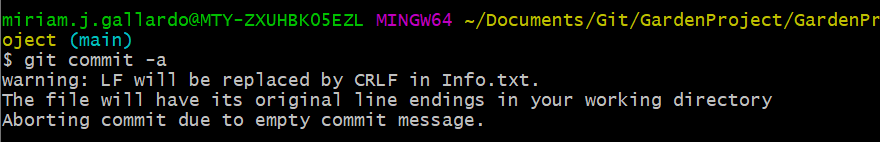
Ahora si modificamos algún archivo entonces git nos dira que se han modificado nuevas cosas y nos la va a mostrar en “ git status ”.



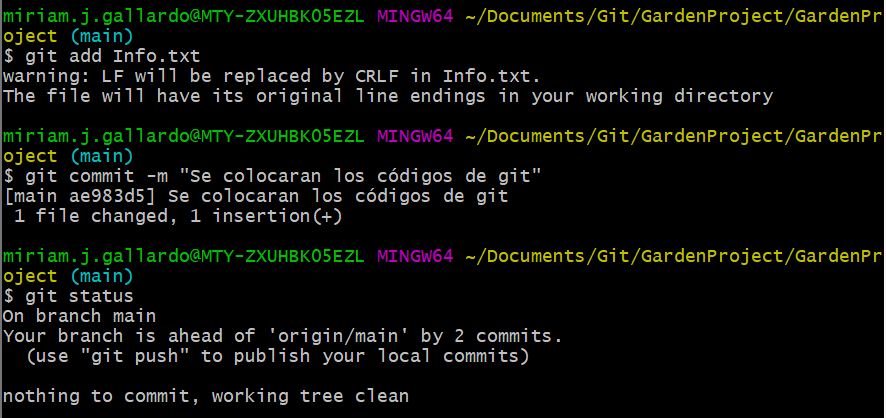
Como vemos se modifico un archivo y ahora lo que vamos a hacer es utilizar los códigos de “ git add “ y “ git commit -a “ este ultimo es una forma de guardar sin mensaje, pero si se coloca lo que nos muestra es:



Para salir usamos “esc” o “:wq”.

Y nos dice que: 

Ahora como ya colocamos “ git commit -a “ y se queremos colocar lo que es “ git add “ entonces nos dirá que ya se realizó, pero debemos de colocar lo que es “ git commit -m “mensaje aquí” para que se pueda guardar y al ver “ git status “ nos muestra que se puede usar el siguiente código que es “ git push”



Nota: con “clear” se limpia el cmd.

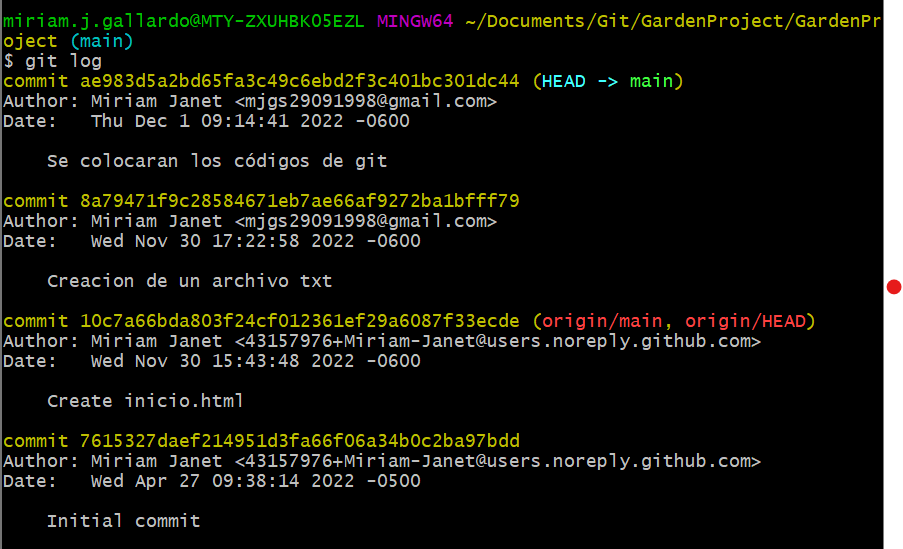
Nota: al momento de colocar un mensaje en

## GIT LOG

Pero si queremos ver que son los commit o para que son? Entonces para eso es por lo que se utiliza:

1. git log

Nos da una lista de todos nuestros commits con su respectiva información. El autor, la fecha y nos muestra el mensaje con su código sha para verlos

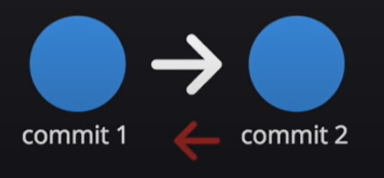


## GIT CHECKOUT

Como se dijo anteriormente, tenemos que con los commits podemos regresar de una versión a la versión pasada de ese commit, y para eso se utiliza lo que es el:

1. git checkout

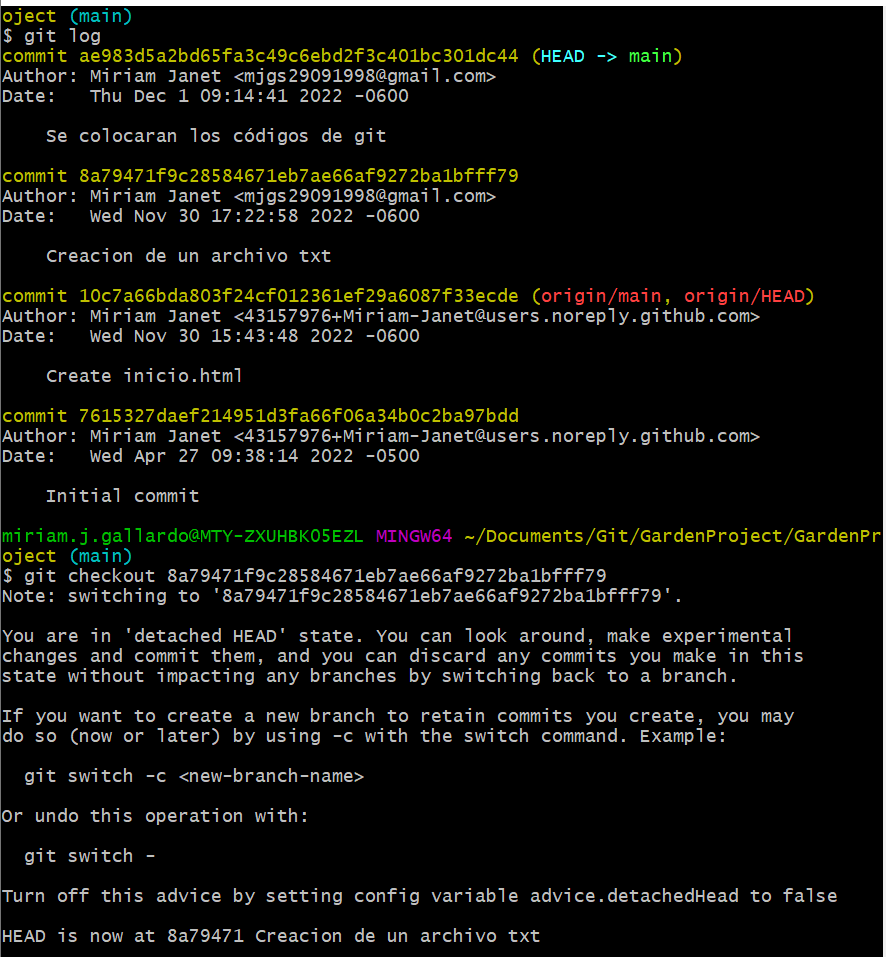
Con este comando viajamos a través de nuestros commits o nuestras ramas.



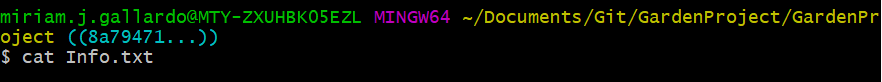
Y para realizar eso, colocamos lo que es:

1. git log
2. copiamos el código sha del commit al cual vamos a viajar o retomar
3. git checkout “pegamos el commit”

y al darle enter, lo que pasa es que nos regresa al archivo que teníamos antes de hacerle los cambios.

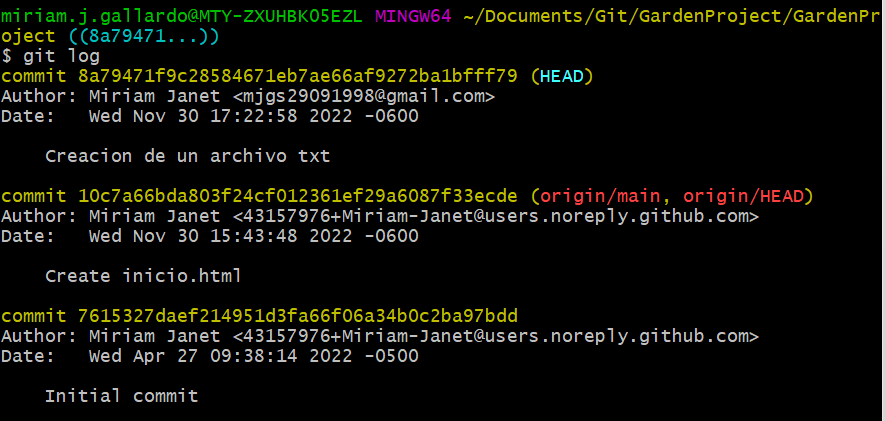


Y cuando vemos en el archivo, vemos que lo que se escribió dentro de el ya no esta por que al principio solo se creo un archivo, pero ahora como le dimos en “git checkout” lo que paso es que nos regresó al primer commit.



Esto es muy útil para poder detectar los errores o simplemente hacerlo por la curiosidad de saber como era nuestro proyecto antes.

Y cuando le damos nuevamente a “git log” vemos que ya no nos muestra nuestro commit donde modificamos el archivo.

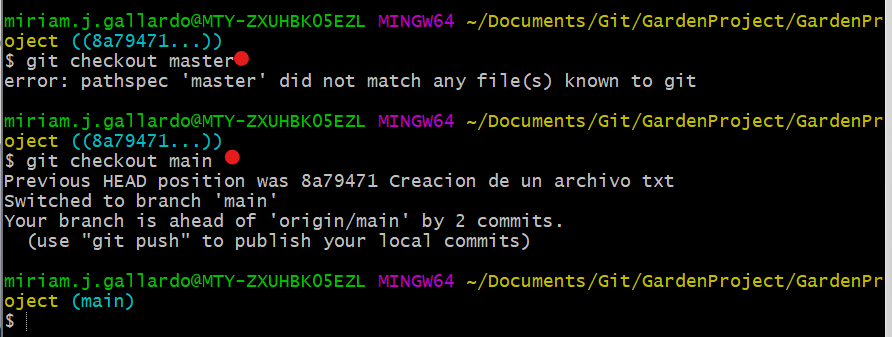


Ahora si nos queremos regresar al último commit que realizamos, lo que se va a colocar es:

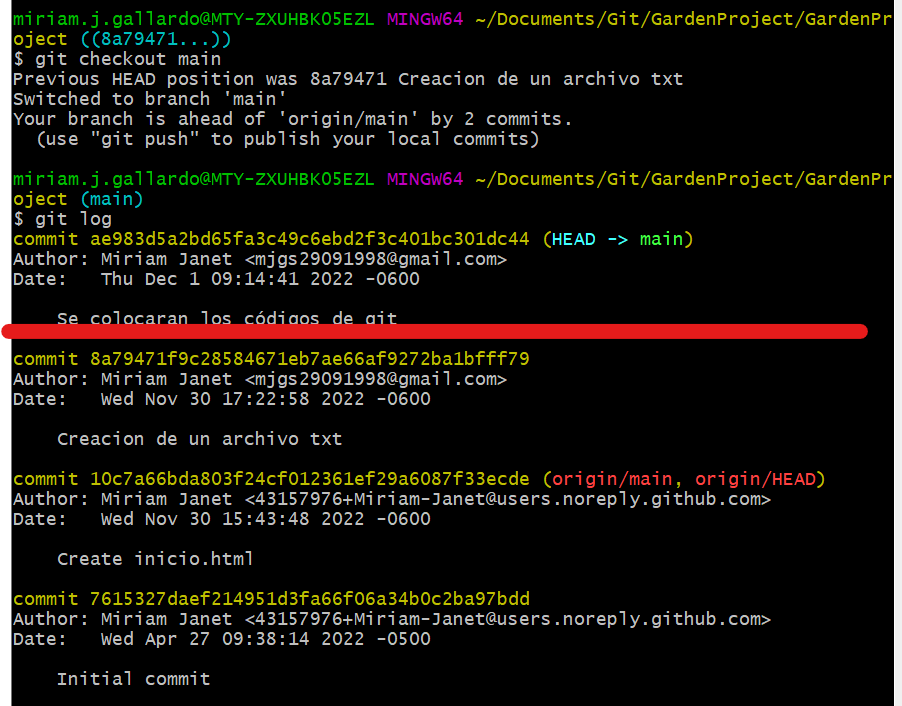
1. git checkout master -> git checkout main

Este nos regresa nuevamente al archivo que hemos modificado.

Mucho ojo en que nombre de Branch estamos para que se coloque, ya que si colocamos mal, nos puede mostrar un error.



Y como vemos ya regresamos la ultima versión del commit.

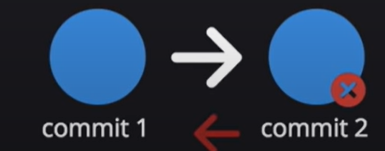


Y como vemos Git guarda cada cambio de lo que hacemos y nos permite ver toda esta clase de resultados.

Ahora mucho ojo con este comando que es el:

1. git reset

Es similar a checkout a diferencia que este ELIMINA los commits.



El cual este tiene 3 tipos de git reset

1. git reset --soft

Es el git reset más simple y que no toca nuestro “Working Area” (No se mete con nuestro código). No le pasara nada al código

1. git reset --mixed

Este borra el “staging Area”, sin tocar el “Working Area”. Casi no se usa mucho, no es muy recomendado usarlo.

1. git reset --hard

Este borra absolutamente todo lo que hay en el commit. Este es el más peligroso por que ya cuando se borra no hay vuelta atrás.

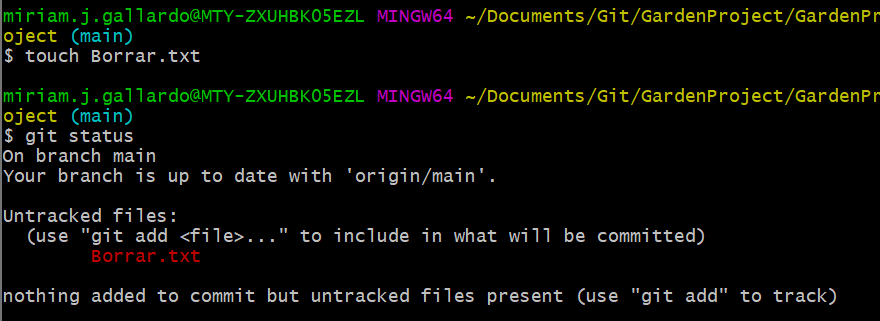
Ahora antes de utilizar este git reset; si queremos tener a la mano todos los commits vamos a colocar lo que es:

* git log > commits.txt

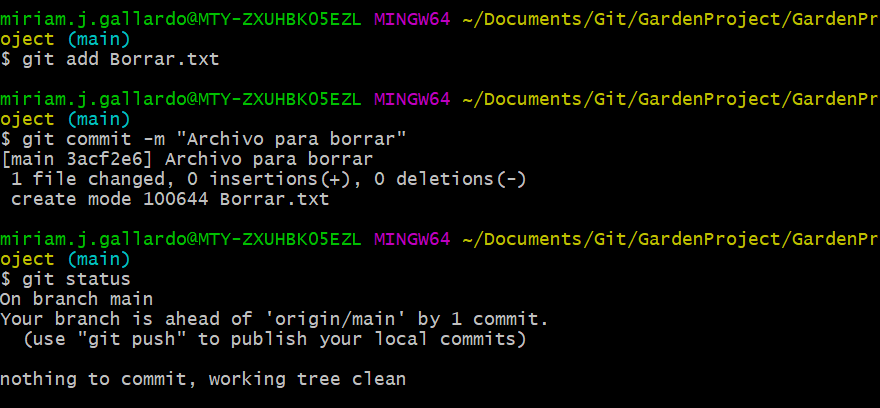
y dentro de este nos mostrara todos los commits realizados en la hoja txt.

**Retomando**.

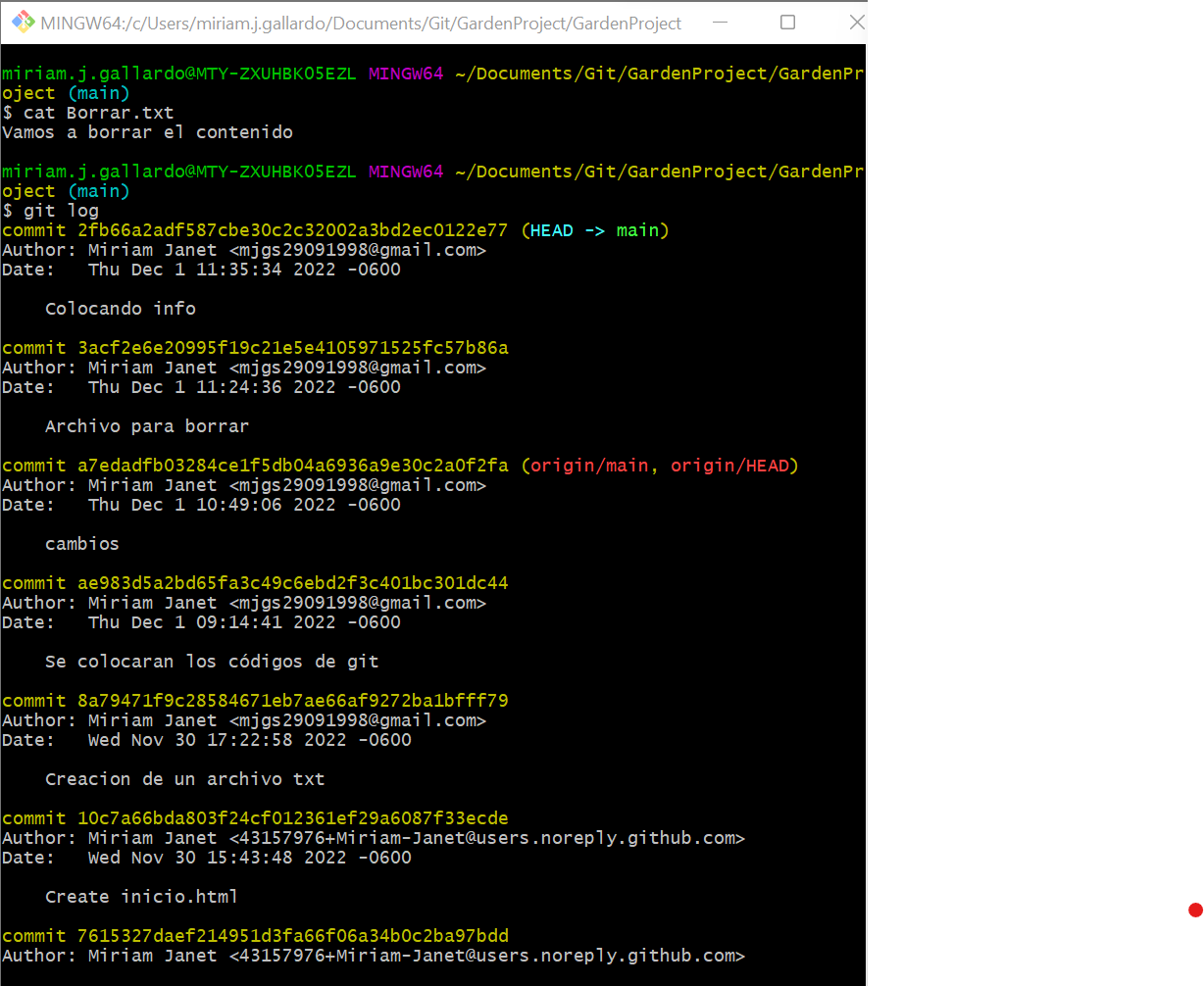
Ahora voy a crear un archivo y vere su status.

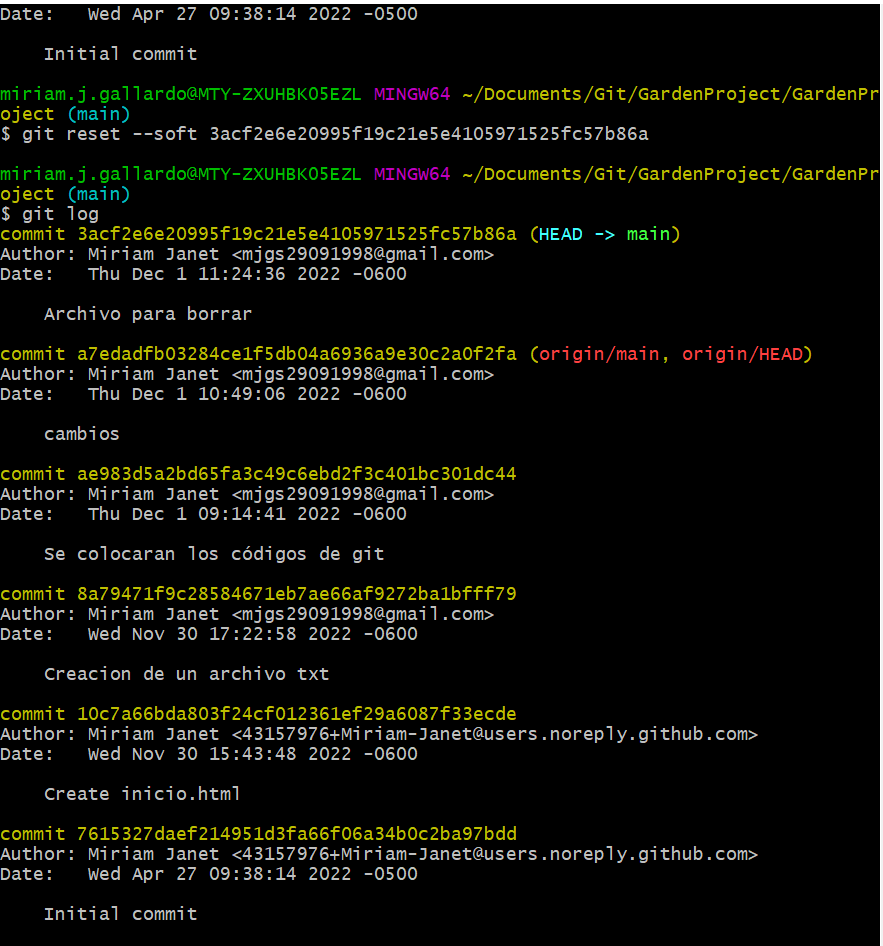


Ahora debemos de agregar el archivo y darle en commit.

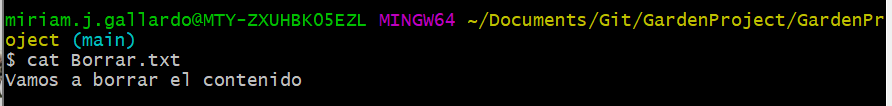


Ahora colocamos “ git log “ para ver el último commit y usamos lo que es el “git reset --soft “pegamos el número del commit”





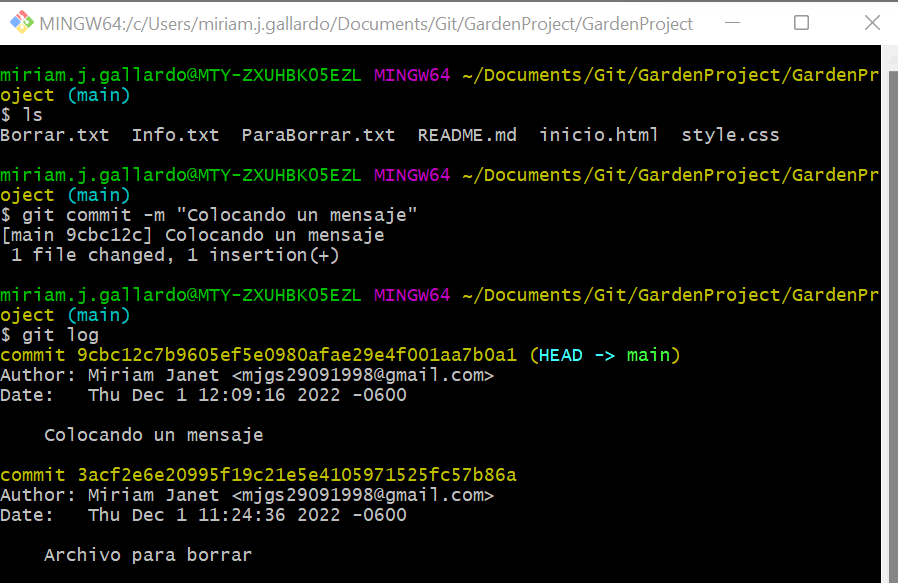
Y como vemos se borro el commit, pero no lo que pusimos en nuestro archivo.



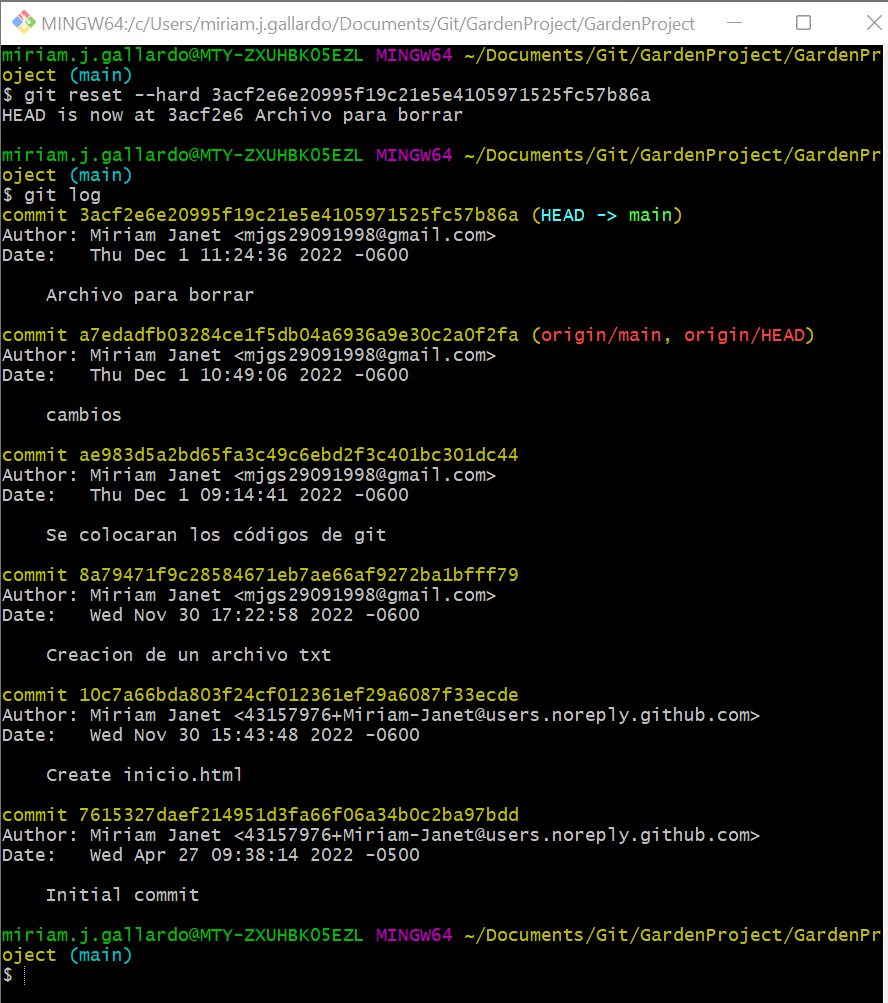
Es algo bueno si queremos borrar solo el commit que no queramos, pero sin que nos afecte en nuestro código.

Ahora continuamos con el “git reset hard ”

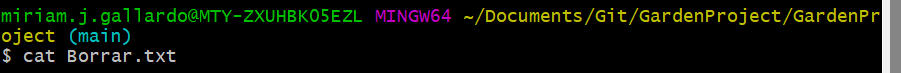
Hacemos un commit, para que podamos hacer este ejemplo:



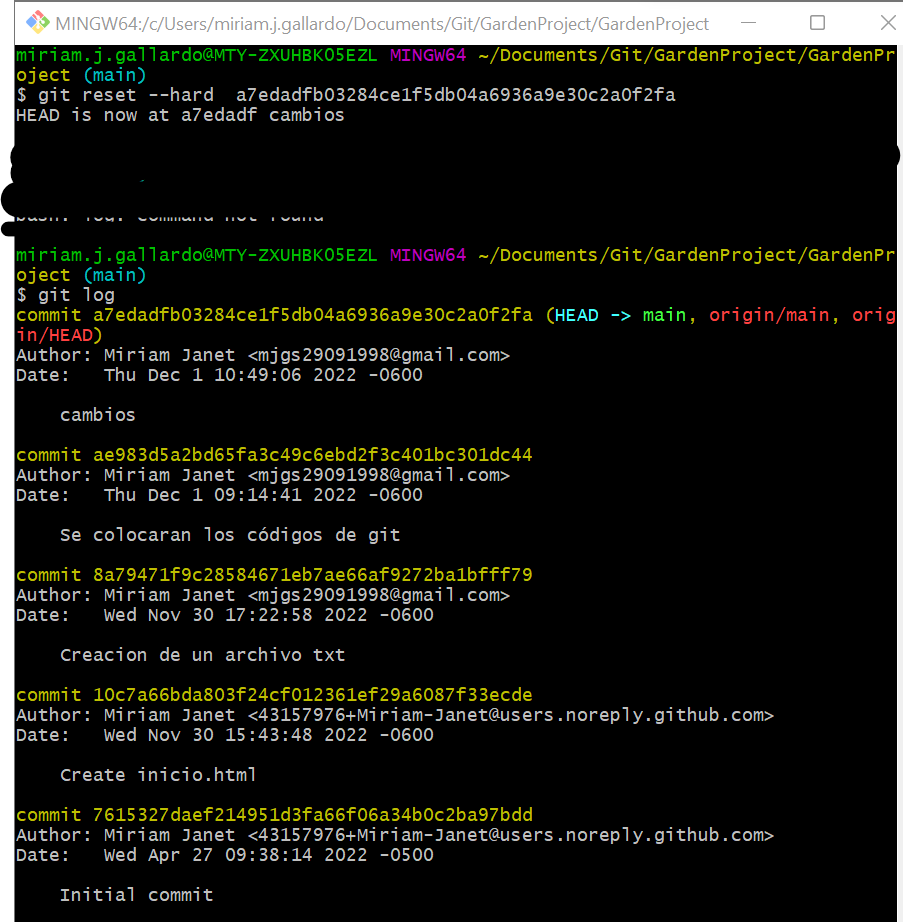
Ahora llegaremos a borrar lo que es hasta el punto donde creamos el contenido que venia en el archivo Borrar.txt, para borrarlo definitivamente.

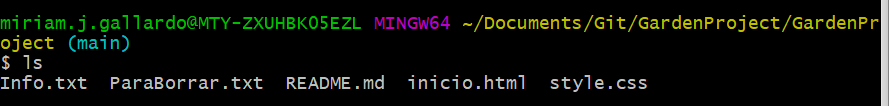


Y vemos



Aquí realice otro reset hard , donde ahora si borramos el archivo Borrar.txt. Hard borra todo, no importa si hay cambios ni nada, pero BORRA TODO.





NOTA: PARA SALIR SE UTILIZA “q”

Seguido tenemos lo que es:

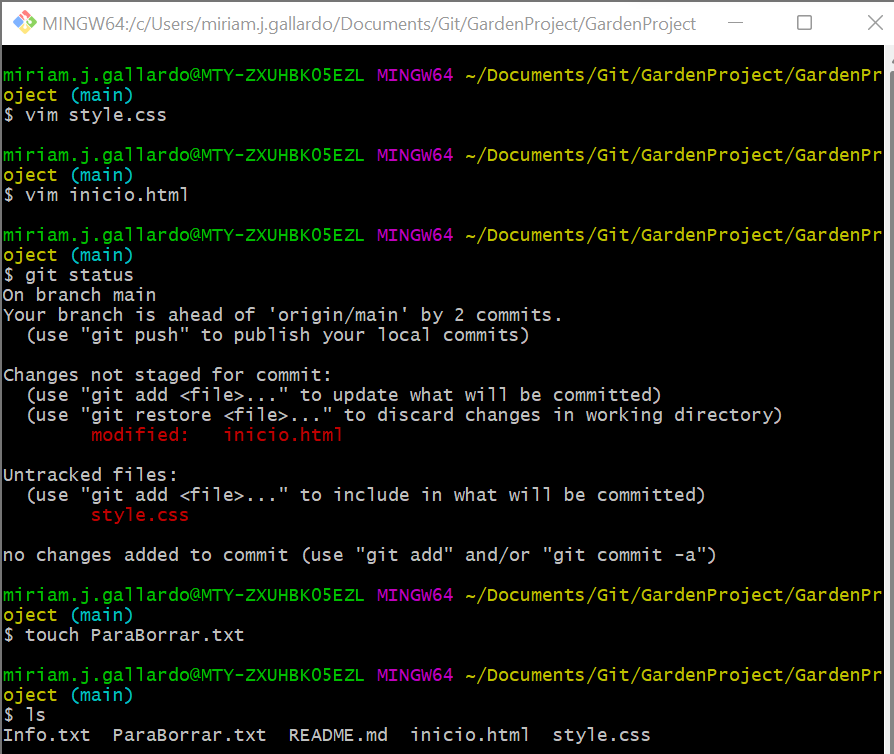
1. git help

Este comando nos ayuda a saber como funciona git o alguno de sus comandos. Y nos da una documentación de cada cosa.

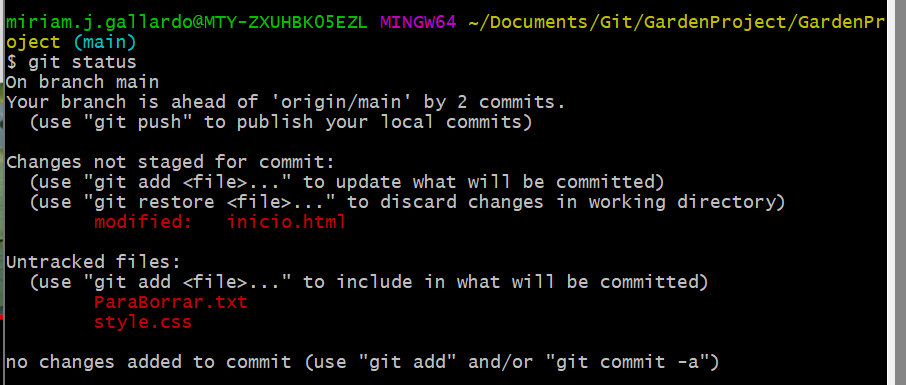
--------------------------------------------------------------------------------------------------------------------------------------

## GIT PUSH

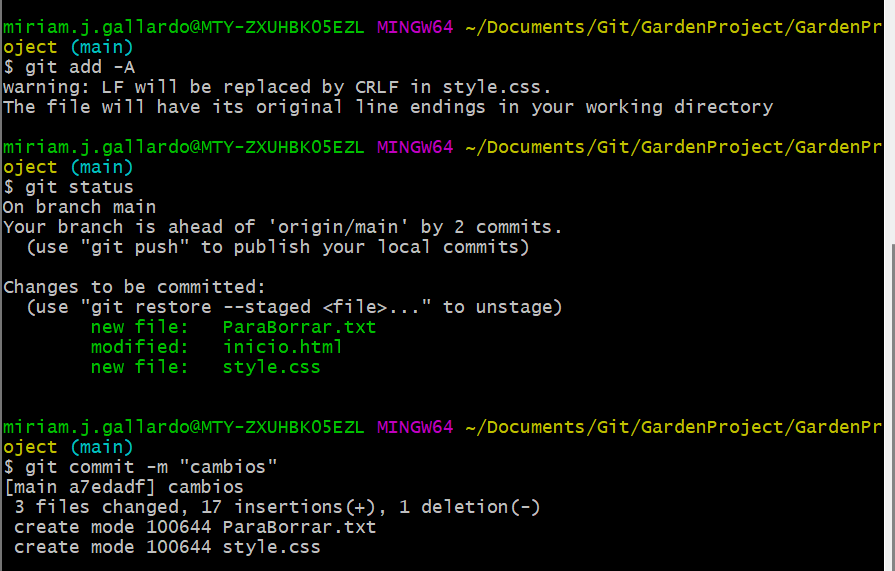
Voy a crear y modificar archivos del proyecto para que se puedan utilizar todos los comandos que se realizaron.



Ahora ya que tenemos lo que queremos, le damos en git status para que nos muestre que archivos se modificaron y cuales se crearon:



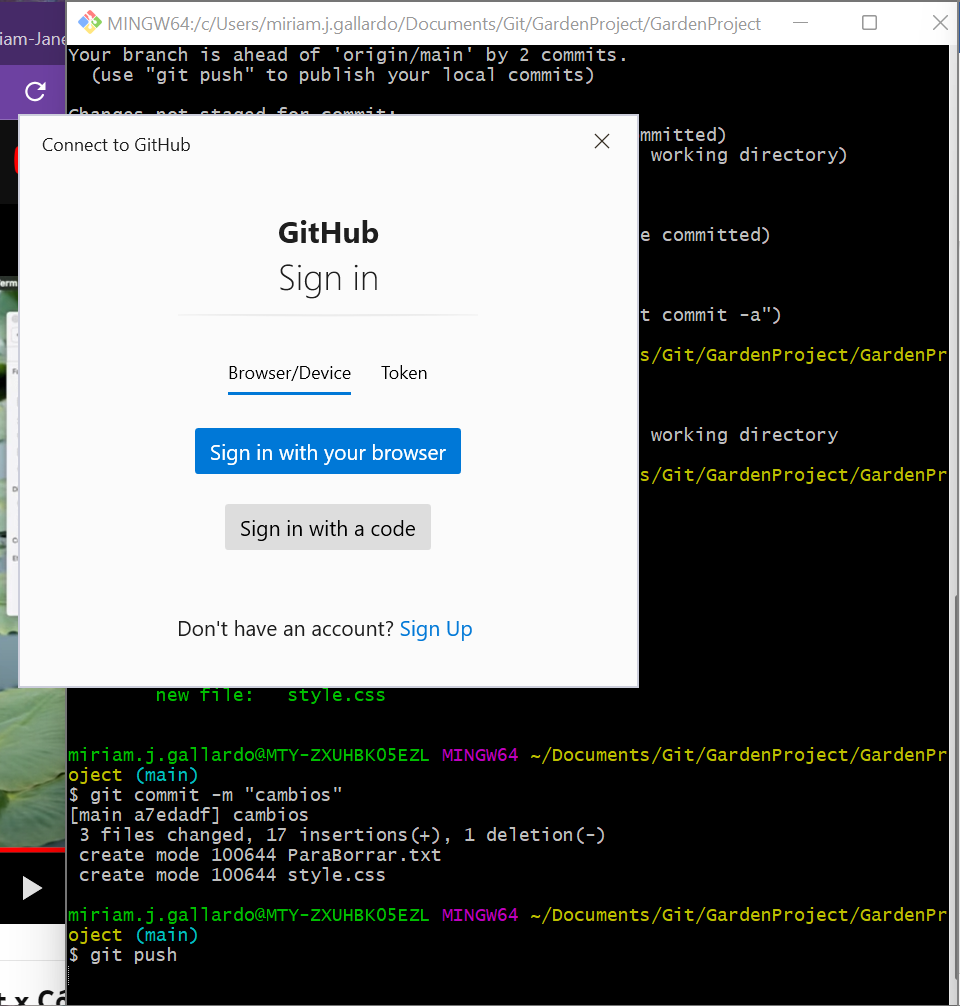
Ahora vamos a darle en git commit – a para agregar todos. También podemos colocar mensaje si queremos, dependiendo de lo que hagamos.



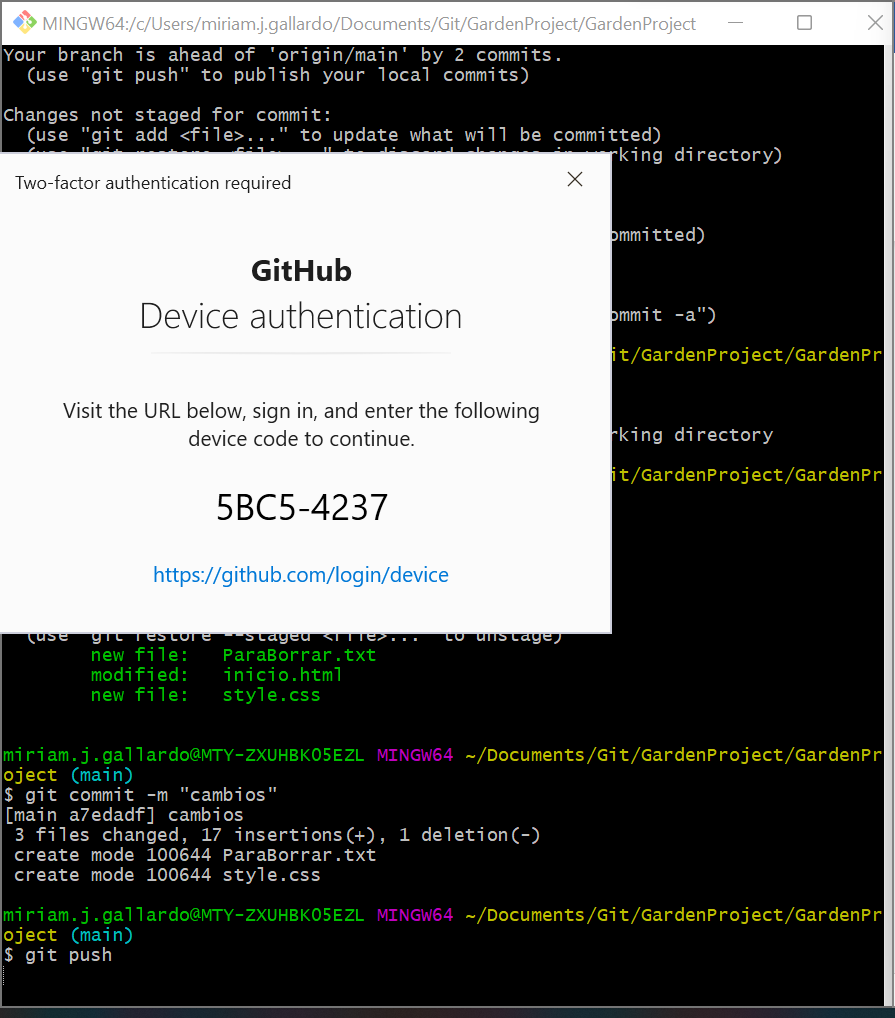
Veremos lo que es el

1. git push

Ahora cuando le damos nos muestra:



Si le damos en “Sing in with a code” nos da un código de autentificación.

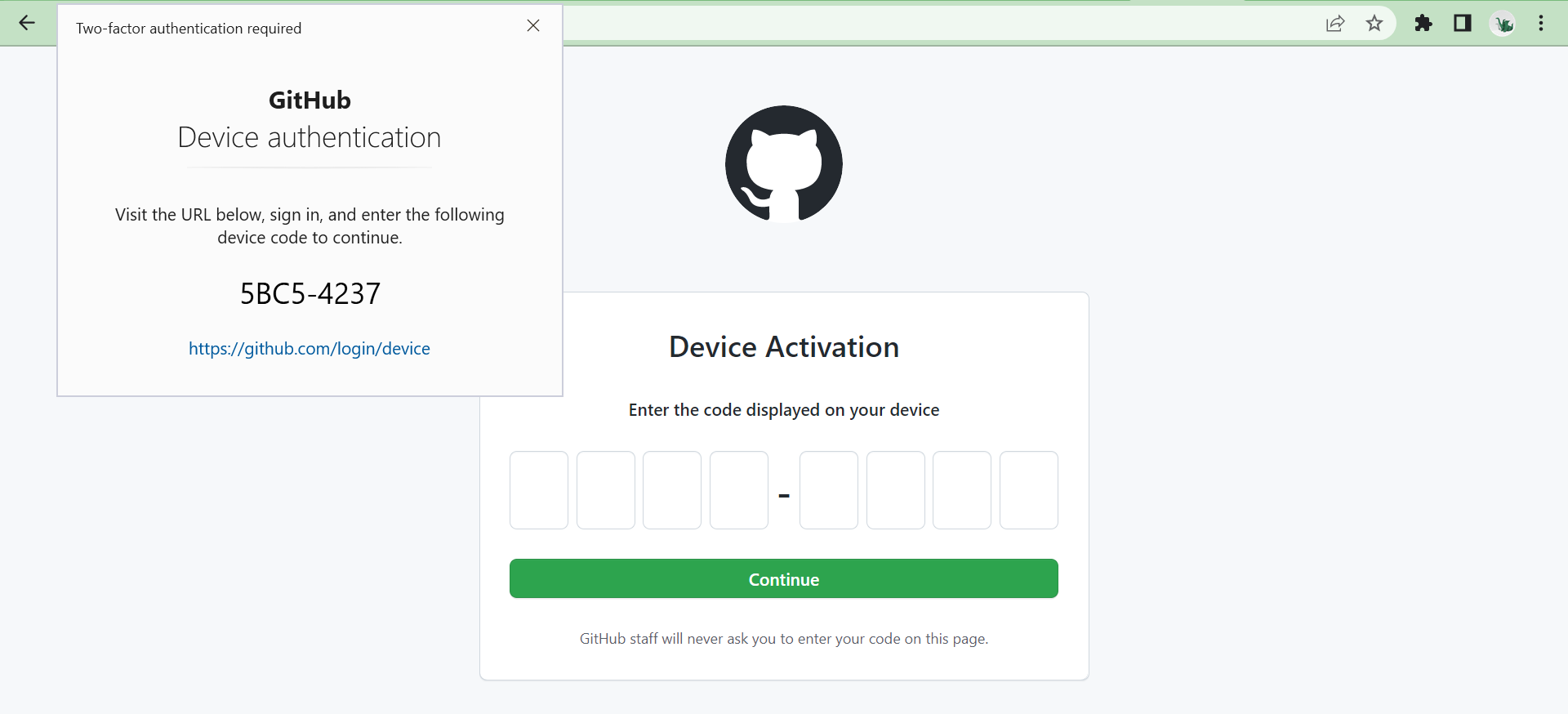


Le damos en la dirección y nos pedirá nuestro usuario y contraseña de Github:

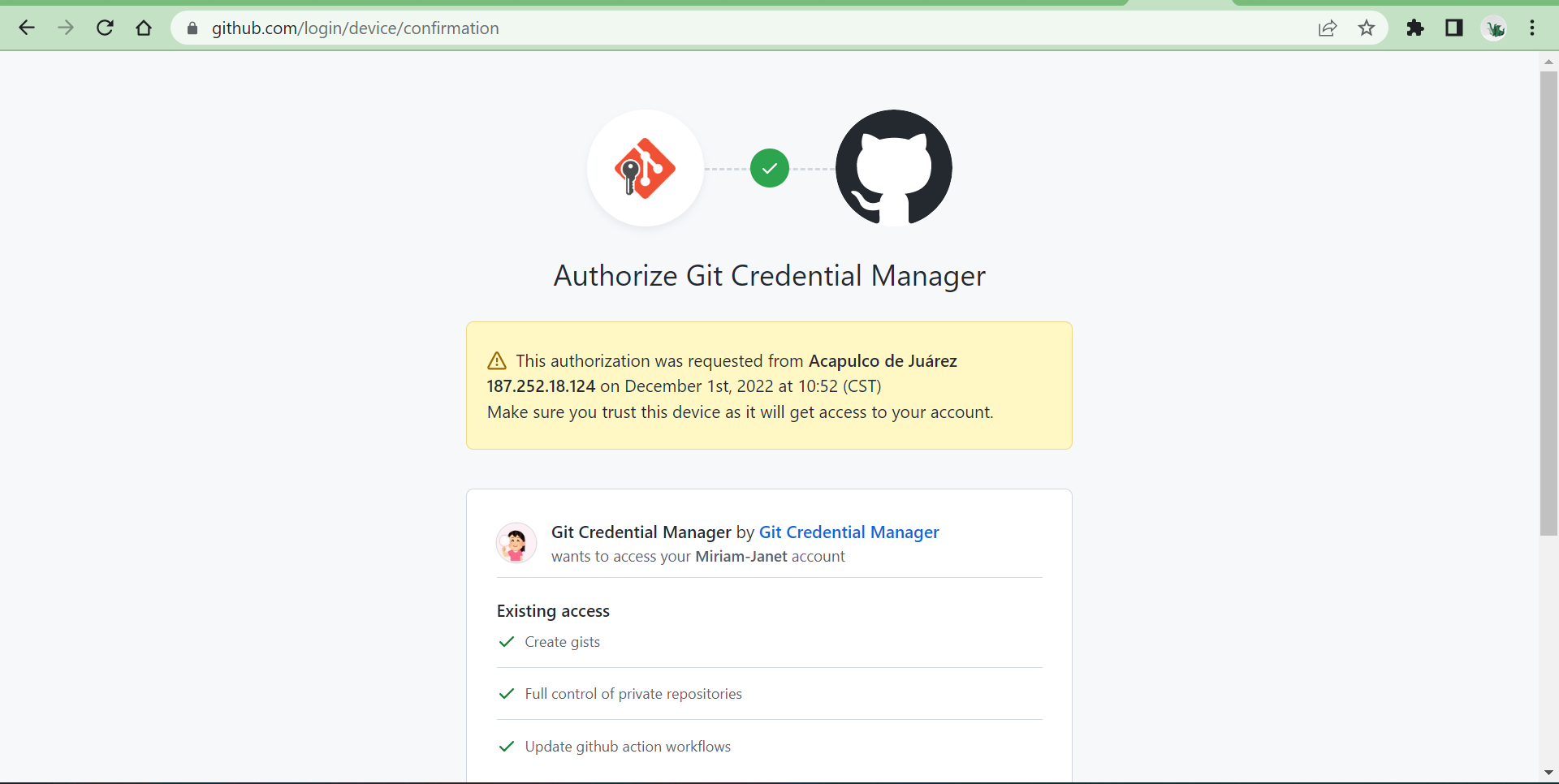
User or email: \*\*\*\*\*\*\*\*\*\*@gmail.com

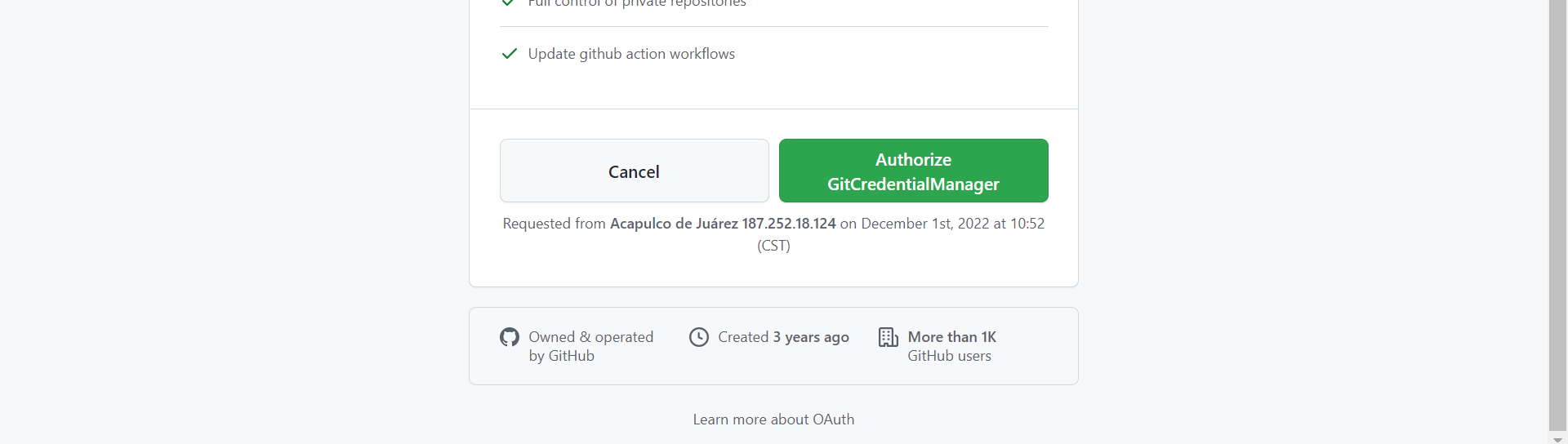
Pass: \*\*\*\*\*\*\*\*\*\*\*\*

Seguido colocamos lo que es el código que nos dieron:

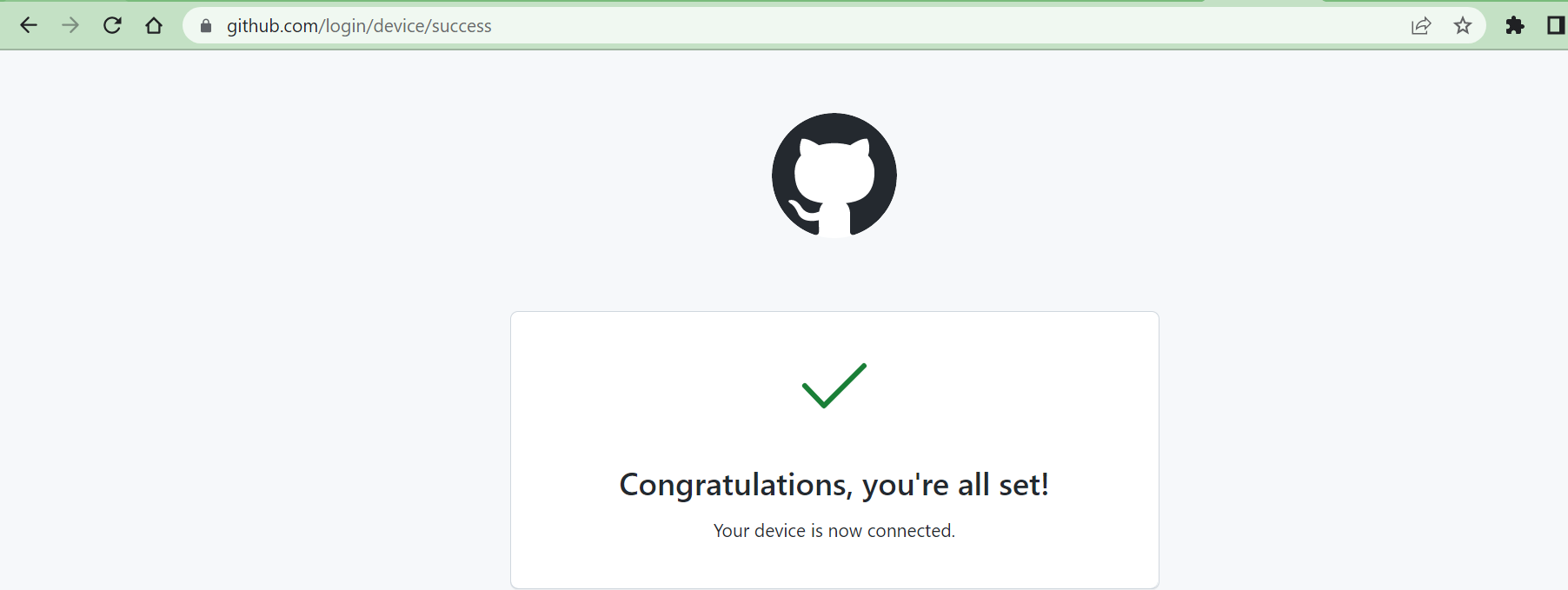


Ya colocado el código, nos muestra lo que es:

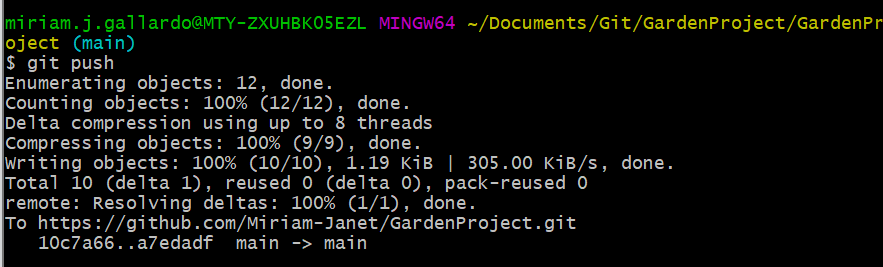




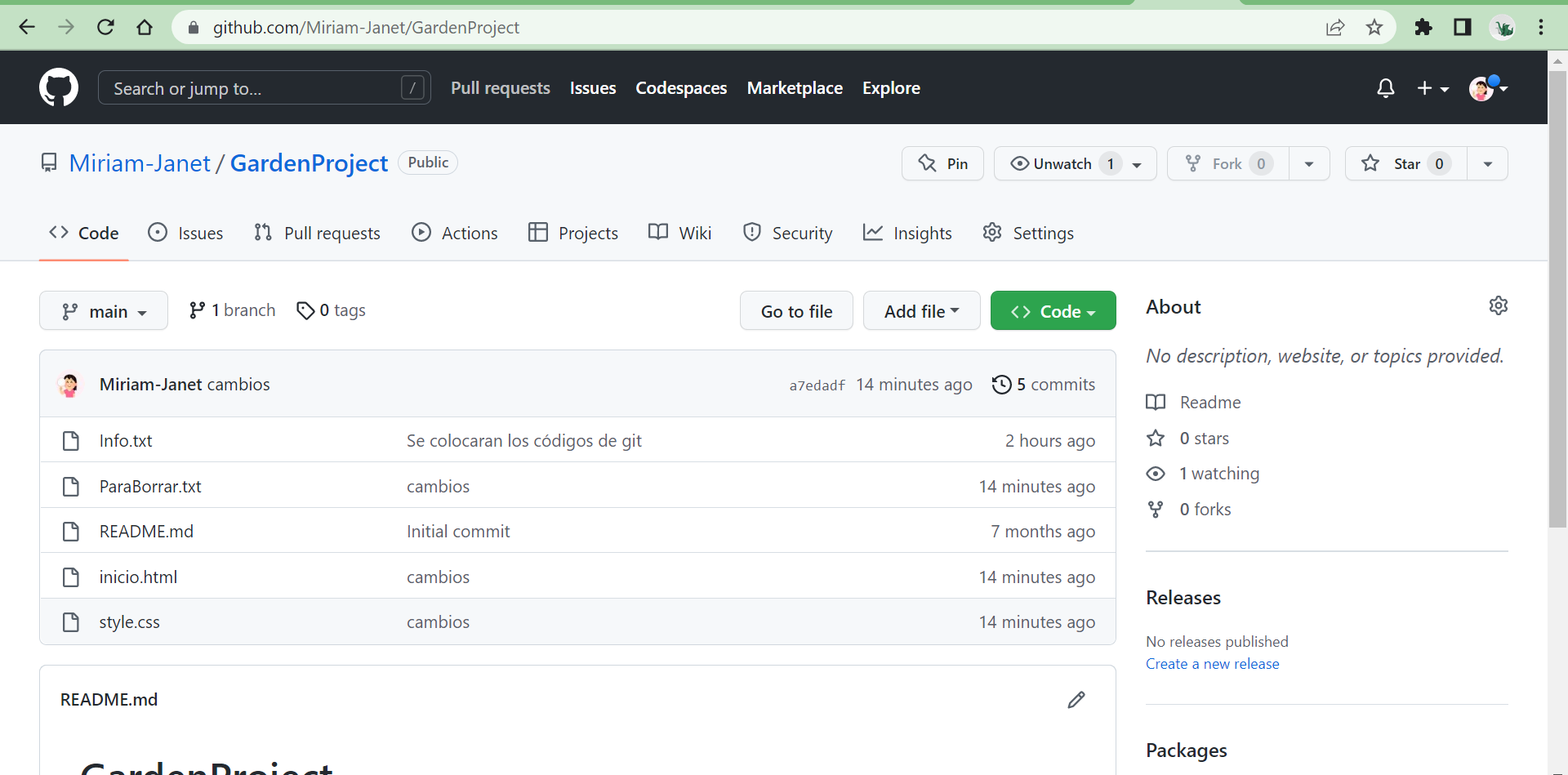
Ya que autorizamos, nos aparecerá:



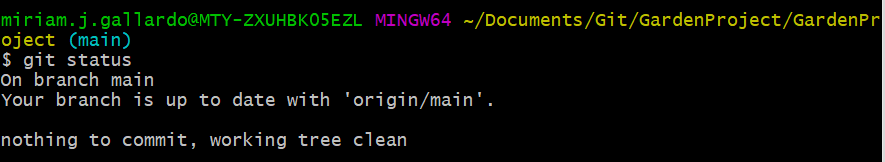
Y en nuestro git bash nos muestra que si se puedo conectar a lo que es nuestro repositorio en Github.



Y al entrar a lo que es nuestro repositorio en Github podemos visualizar que se han realizado todos los cambios y agregado los archivos correspondientes que se crearon.



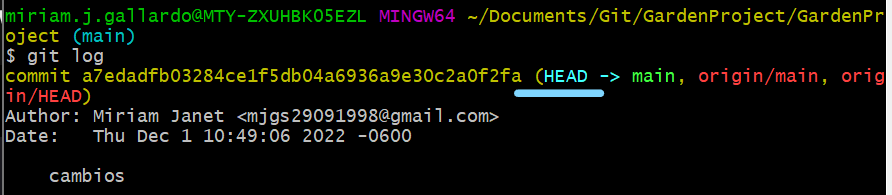
Y ahora en el git bash, cuando se coloca el comando “ git status “ ya podemos denotar que no tenemos más commits realizados.



# Ramas y funciones

## HEAD

El concepto “head” es en el commit donde nos encontramos en este momento.



## RAMAS

Es una línea de tiempo en nuestro proyecto, que nos sirven para arreglar errores, experimentar, hacer grandes cambios, etc.

Cualquier cosa que no afecte a nuestro proyecto actual.

## Rama master

Es en donde comenzamos a trabajar, es la rama principal y estable de nuestro proyecto. Cuando hacemos git init es la rama que se genera por default y es la rama donde hemos generado los commits o el commits donde estamos hasta ahora, estos estarán en la rama master, es la rama mas estable a menos que cambiemos de rama.

Para ver las ramas:

1. git branch

Este nos permite visualizar las ramas

Como crear una rama?, para ello se debe de colocar:

1. git branch rama

