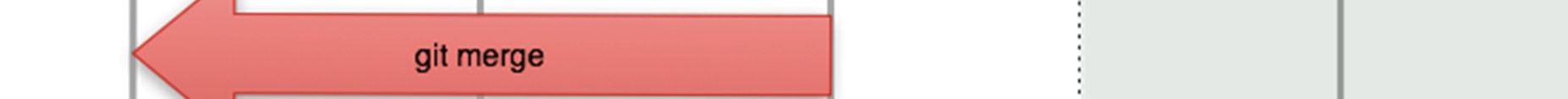
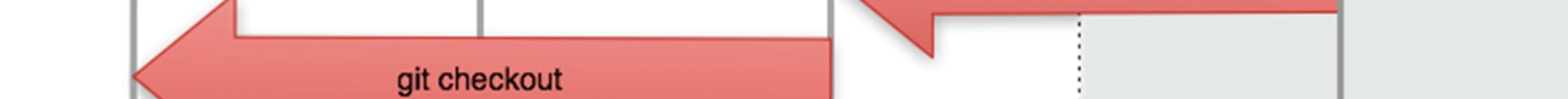
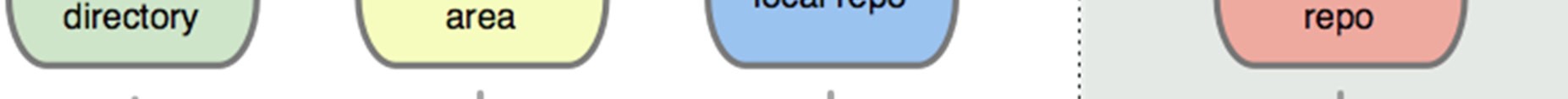
**Fundamentos de GIT**

**Flujo de trabajo de GIT**



En este apartado podrás comenzar a trabajar con git.

# #Comandos básicos

Aprendamos los primeros comandos con git

// Conocer la versión de git instalada git version

// Ayuda sobre los comandos git help

// Iniciar un nuevo repositorio // Crear la carpeta oculta .git git init

// Ver que archivos no han sido registrados

git status

// Agregar todos los archivos para que esté pendiente de los cambios git add .

// Crear commit (fotografía del proyecto en ese momento) git commit -m "primer commit"

// Muestra la lista de commit del mas reciente al más antigüo git log

En resumidas cuentas nosotros realizamos cambios en nuestros archivos, el comando status verificará que archivos han sidos modificados. Cuando deseemos registrar esos cambios tendremos que agregarlos con add . así ya estará listo para poder hacer un commit. El commit realiza la copia de ese instante para poder volver en el tiempo si es que es necesario.

# #Trucos

// Muestra en una línea los commit realizados git log --oneline

// Muestra en una línea los commit realizados pero más elegante git log --oneline --decorate --all --graph // Solo muestra los archivos modificados git status -s

## Diferencias entre -- y -

--decorate hace referencia a una palabra

-s hace referencia al comando o a varios comandos, -sa serían dos comandos diferentes

// Vemos información de la rama maestra

|  |  |
| --- | --- |
| git status -s -b |  |
| git status -sb //Hace lo mismo que el comando anterior | |

|  |
| --- |
|  |

# #Creando alias globales

Los alias nos sirven para crear atajos de comandos, podemos guardar diferentes alias de forma global y quedarán guardados en la configuración de git.

// Guardamos el alias "lg" que ejecutará todo lo que está entre comillas git config --global alias.lg "log --oneline --decorate --all --graph"

// Para ver el archivo config con los alias creados git config --global -e

**Vim** es el editor de código en la línea de comandos

## Salir del modo edición "Vim"

Para salir del modo edición de la líneas de comando precionar :q, en caso de realizar algún cambio sin guardar escribir :qa

:q! también sirve para salir sin guardar

// Modo lectura sin poder modificar git config --global -l

// Realiza el add . y commit más mensaje al mismo tiempo git commit -am "más comandos agregados" // Para editar un commit, como por ej: el mensaje git commit --amend

## Trucos de editor Vim

i para comenzar a editar esc para salir del modo edición wq para guardar y salir q! salir sin guardar cambios

# #Viajes a través de los commit

Vamos a conocer como podemos movernos entre los diferentes commit que tengamos registrados, supongamos que tenemos los siguientes commit:

* f82f457 (HEAD -> master) mas comandos agregados
* f52f3da nuevos comandos en fundamentos.md
* e4ab8af mi primer commit

// Viajamos al commit en específico f52f3da git reset --mixed f52f3da

// Viajamos al commit en específico f52f3da y eliminamos los cambios futuros git reset --hard f52f3da

// Muestra todos los cambios incluso si borramos los commit git reflog

// Viajamos al commit en específico f52f3da y podemos restaurar los archivos git reset --hard f52f3da

Si no hicimos un commit pero aún así queremos revertir los cambios en un archivo específico podríamos utilizar el siguiente comando: git checkout -- nombreArchivo.conExtensión

Si deseamos destruir todos los cambios sin haber realizado un commit podemos utilizar: git reset --hard

# #Renombrar archivos

Puede que queramos renombrar un archivo, es recomendable hacerlo directamente en la línea de comandos para registrar los cambios con git.

// Cambiar nombre git mv nombreOriginal.vue nombreNuevo.vue

Recuerden hacer el commit para registrar los cambios en git.

# #Eliminar archivos

// Cambiar nombre git rm nombreArchivo.vue

También recordar hacer el commit para salvar cambios en git.

# #Ignorando Archivos

Para no hacer seguimiento de carpetas o archivos, debemos crear el siguiente archivo:

 .gitignore

Su estructura de ejemplo sería así:

|  |  |  |
| --- | --- | --- |
| arhivo.js | |  |
| \*.js |  |
| node\_modules | | |

// Ignora el archivo en cuestion

// Ignora todos los arhivos con extensión .js

/ //Ignora toda la carpeta

# #Ramas o branch

Hasta el momento solo hemos trabajado en la rama "master" pero puede que necesitemos crear diferentes ramas para los seguimientos de git.

// Crea una nueva rama git branch nombreRama

// Nos muestra en que rama estamos git branch

// Nos movemos a la nueva rama git checkout nombreRama

Podemos unir la rama master con la nueva, para eso tenemos que estar en la master para ejecutar el siguiente comando:

// Nos movemos a la nueva rama git merge nombreRama // Eliminar una rama git branch -d nombreRama

## Atajos

Podemos utilizar git checkout -b nuevaRama para crear la nuevaRama y viajar a ella.

# #Tags

Con los tags podemos hacer versiones de nuestro proyecto.

// Crear un tags

git tag versionAlpha -m "versión alpha"

// Listar tags git tag // Borrar tags git tag -d nombreTags

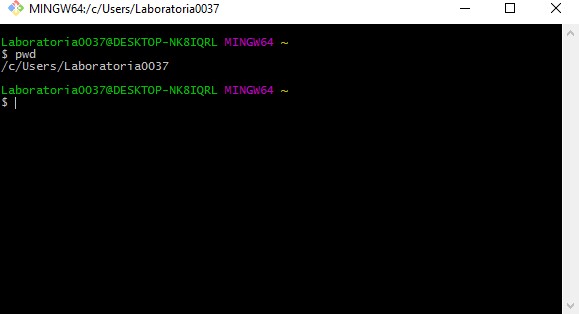
// Hacer una versión en un commit anterior ej: f52f3da git tag -a nombreTag f52f3da -m "version alpha"

// Mostrar información del tag git show nombreTag

# Aprendiendo sobre comandos

Empezaremos con los 6 comandos mas importantes en consola para todo principiante.

1. **pwd :** Para ubicarme en que directorio me encuentro.



**Nota:**

*.. (Punto punto) :* Subir un nivel al directorio moverse a un lugar fuera.

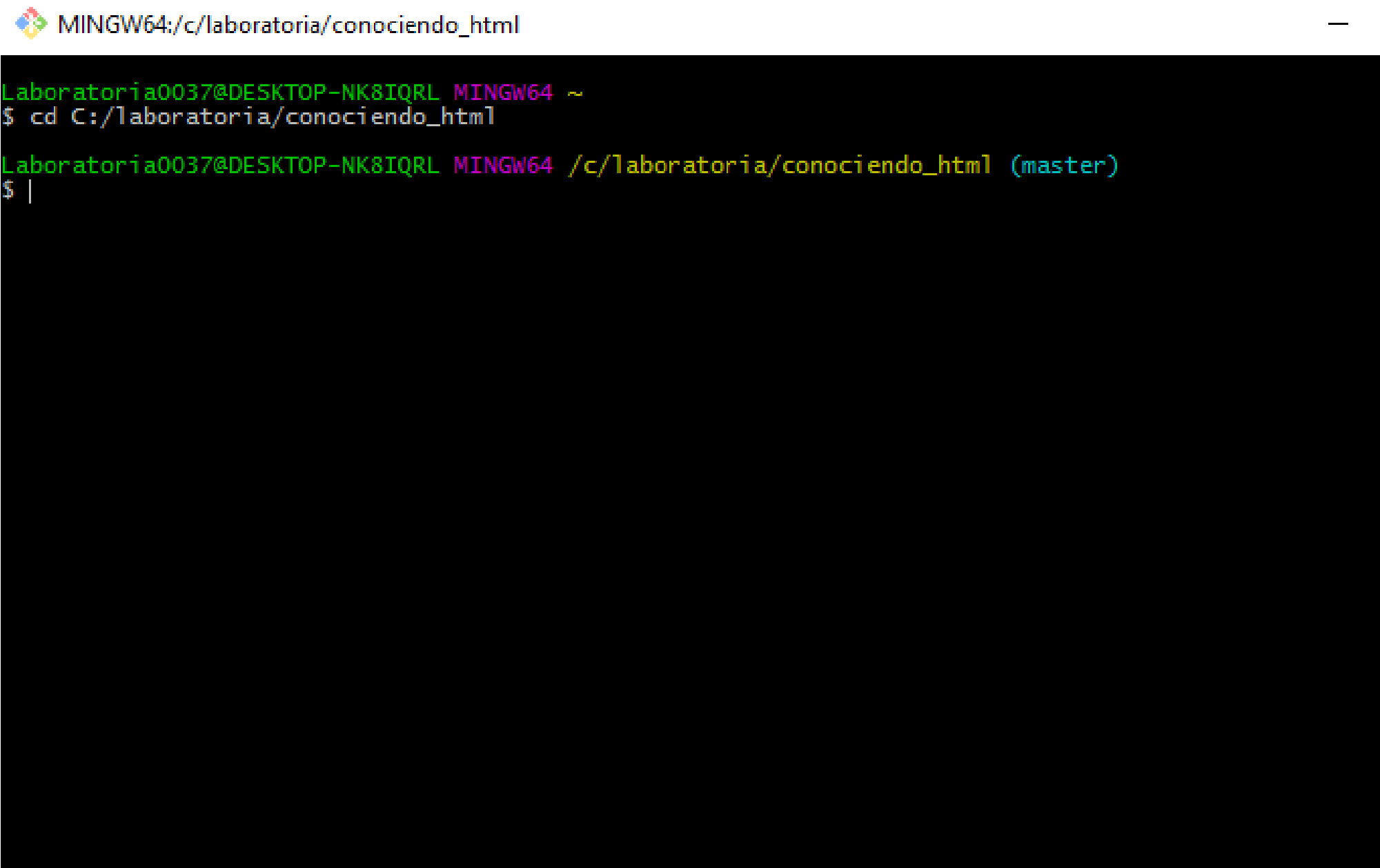
1. **cd :** Para situarse dentro del directorio.

**3**

**. ls :**

Para listar todo lo que está dentro de donde nos

situamos

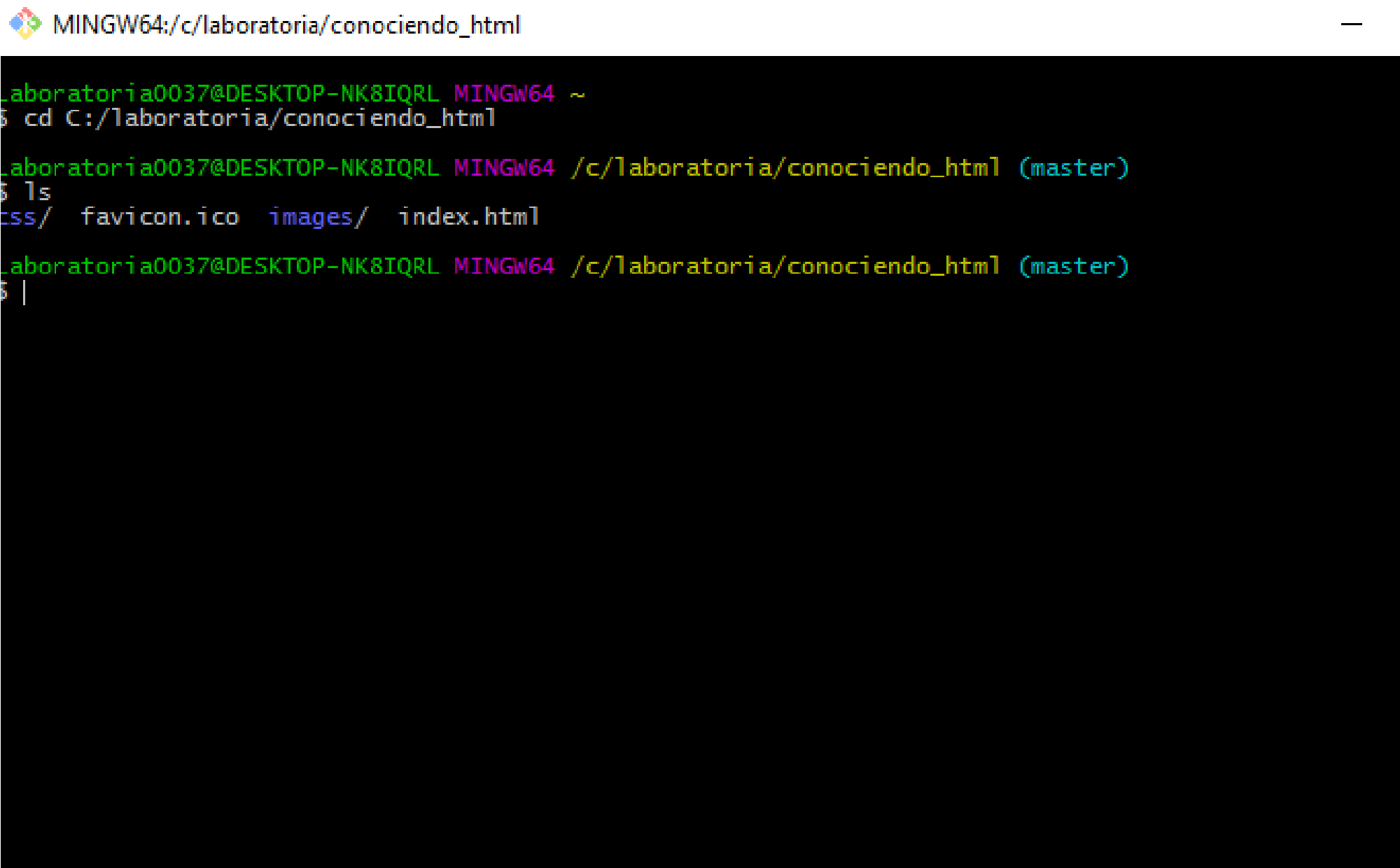


**4**

**. ll :**

Para listar de una forma más grande

Qué entendemos por -rw-r-r — 1



**READ (r)**

**WRITE (w)**

**EXECUTE(x)**

Son permisos para los directorios y archivos.

|  |
| --- |
|  |

# Primer uso de Git Bash

**:**

**. mkdir**

**5**

Para crear directorios.

Resultado:

**6**

**. touch :**

Para crear los ficheros archivos.

Resultado:



**1.git init** : Todo lo que se escriba dentro de la carpeta va a estar dentro del **.git**

Allí va el historial de versiones y está oculto dentro de la carpeta.

→ Solo una vez.

1. **git add** : Lleva el control de los archivos que se agregan luego de escribir ese comando.

**2.1 git add .** : Es para todo agregar pero no es recomendable.

1. **git commit** : Comando que indica que esta lista alguna funcionalidad para que sea una versión del código.
2. **git commit — m “Mi primer commit”**

→ Se repite n veces cada vez se cambia el código.

**Nota :**

* *Se tiene que detallar muy bien lo que se pone en el comentario es una buena práctica para que nuestro equipo colaborativo nos pueda entender y solucionar errores en caso sucediera.*
* *Cuando coinciden las líneas entre una persona y la otra pueden ocurrir conflictos.*

**Push**

Básicamente lo que realiza un push es publicar lo que se encuentra en nuestro servidor local y llevarlo al servidor remoto de Github

El push empuja lo de nuestro servidor local al servidor remoto en github

**Pull**

El pull trae los cambios de nuestro repositorio remoto y los actualiza al repositorio local.

1. **git remote add origin [URL DEL REPOSITORIO EN GITHUB]**
2. **git push origin master :** Es el nombre que se le pone al repositorio remoto al que se conecta.

**¿Qué es un Branch ?**

Es una rama diferente a la rama master que tenemos y se utiliza para trabajar de forma colaborativa y eficaz con esto podremos asignarnos tareas y ordenar mejor nuestro proyecto.

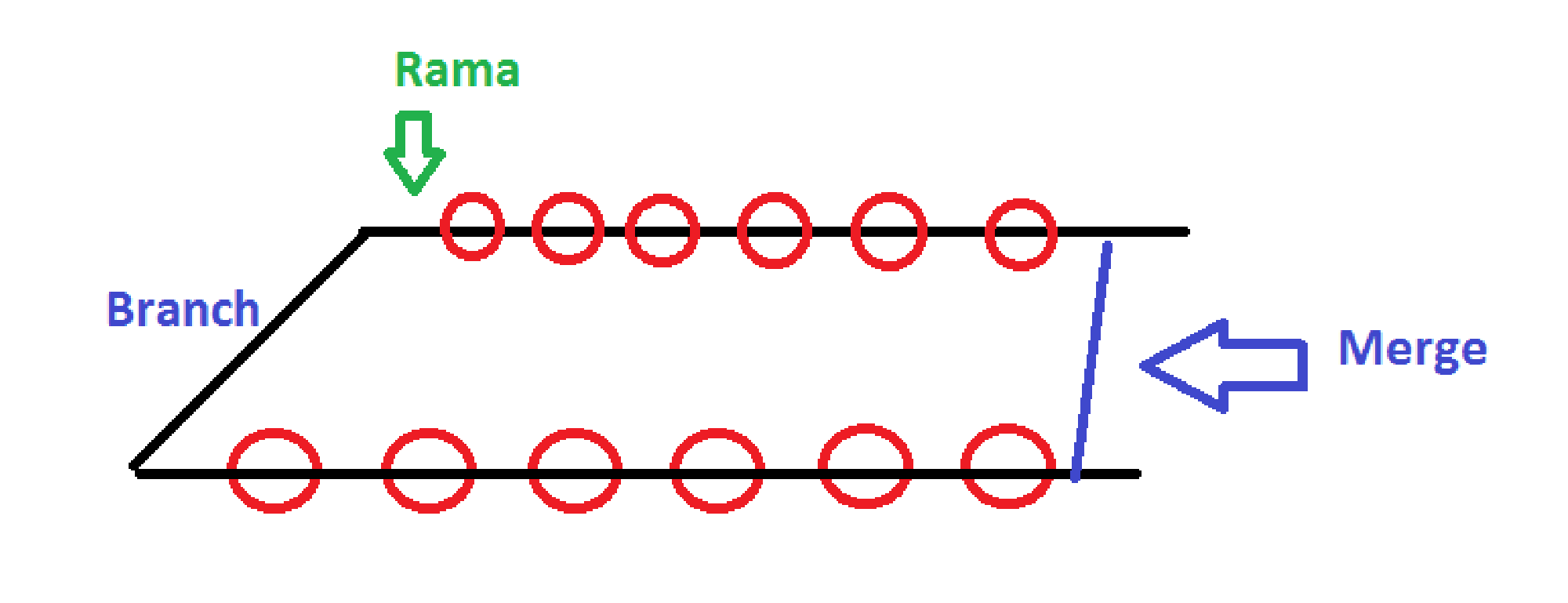
**¿Qué es un Merge?**

Es una fusion entre ramas.

Los conceptos entre Branch y Merge suelen ser dificiles de aprender pero con los ejemplos podrás tener una mejor idea de que es y para que utiliza.

**Utilizando el comando Branch** **git push origin [Nombre de la Rama]**

**Ejemplo de Merge**



1. Situarse en la rama de producción en este caso master: **git checkout master.**

3. Realizar un merge con la rama que se desea incluir : **git**

**merge portada**

5. Publicarlo : **git push origin master**

# Como Publicar mi proyecto en Github

La práctica es mucho mejor que la teoría estos son los primeros pasos para empezar a familiarizarse con la consola de Git y trabajar en proyectos que se alojen en Github.

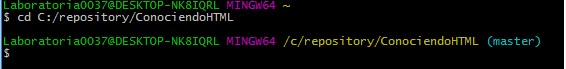
**Pasos:**

1. Situarse en la carpeta donde esta nuestro contenido con cd

2

. Realizar un

**git init**



3. Realizar un add ( **git add .** )

**Realizando mi Primer commit**

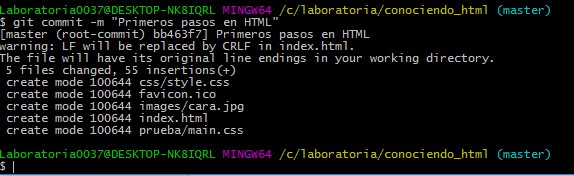
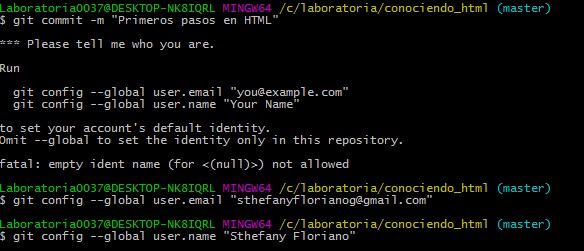
Al escribir el commit cuando lo utilizamos por primera vez nos pide configurar las credenciales 4. Configurando mis credenciales

|  |
| --- |
|  |

3. Utilizar un remote add origin para conectar nuestro repositorio local con el github la url tiene que ser exactamente igual al url que creamos en el github.

2

. git commit — m “Primeros pasos en HTML”



4

. Realizamos un push

Pide credenciales :



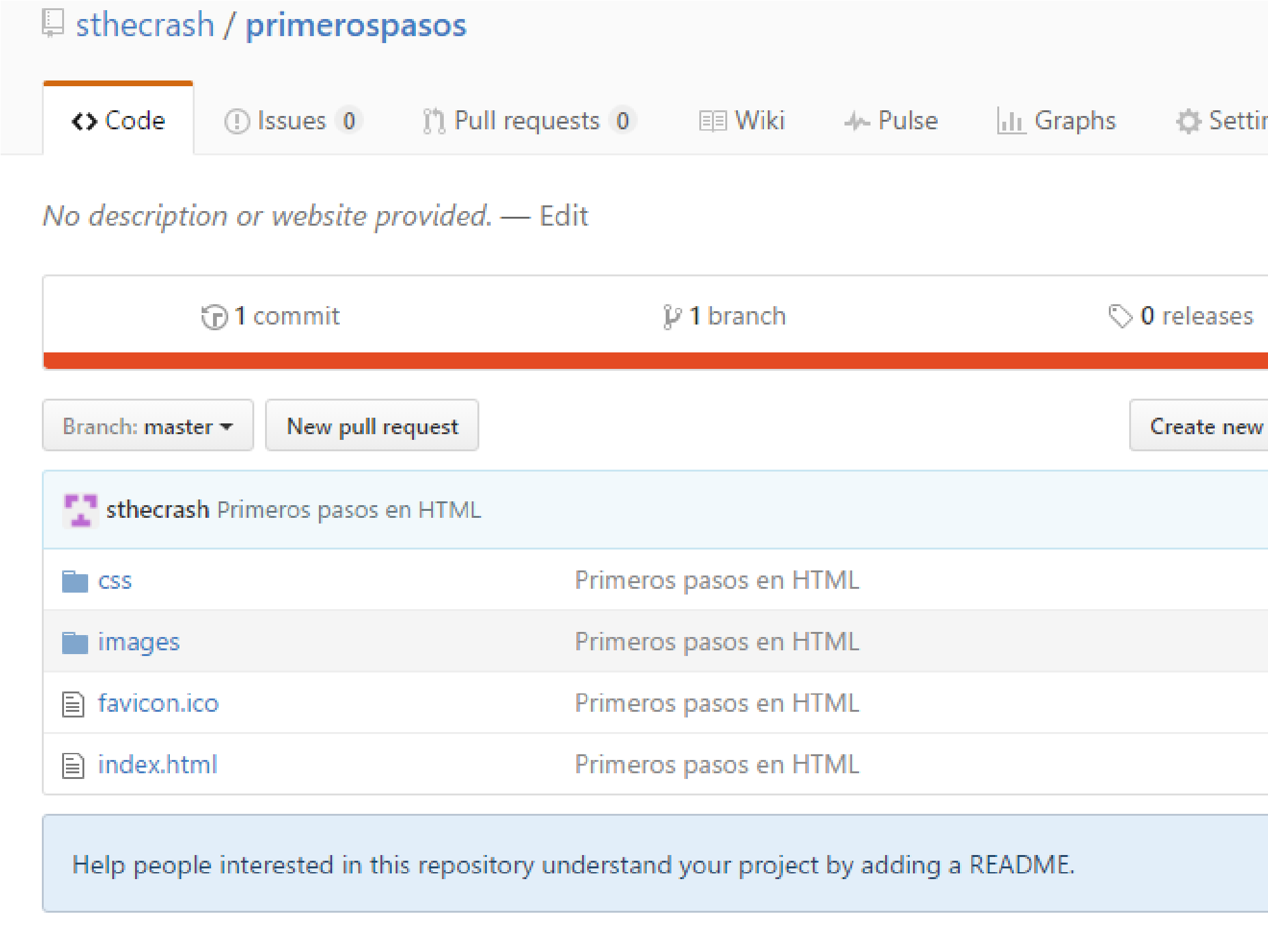
Poner nombre de usuario de github y clave de github

5. Volvemos a hacer el commit

**Listo el Repositorio ha sido publicado en Github.**

Publicando mi proyecto en Github

Pages



1. Situarse en la carpeta donde está el proyecto con cd [Ruta]
2. Para que pueda publicarse tiene que estar dentro de una rama se procede a crearla.



1.

Verificar con

**git status**

Nota : En mi caso no hay cambios pero en caso los hubiera



se usaría un git add

1. Realizar un **git commit -m ‘initial commit’**

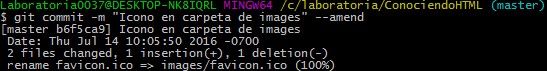


1. Publicarlo **git push origin gh-pages**

**Información adicional**



Si te equivocas al poner el mensaje en el commit se corrige con el — amend



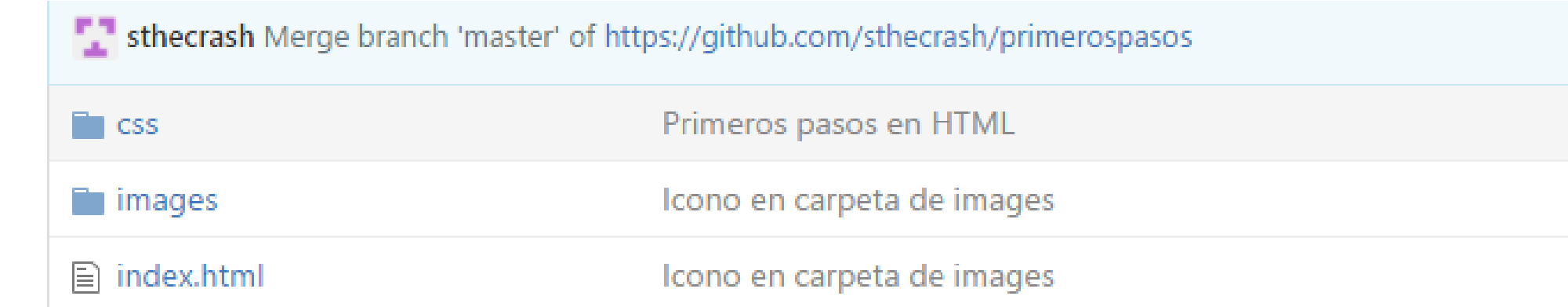
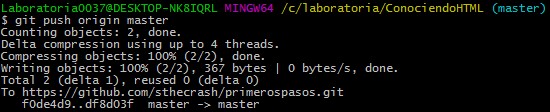
Luego se realiza un pull

Finalmente un push

Y listo corregi el contenido de mi mensaje

Ejemplo : Subiendo Proyecto a

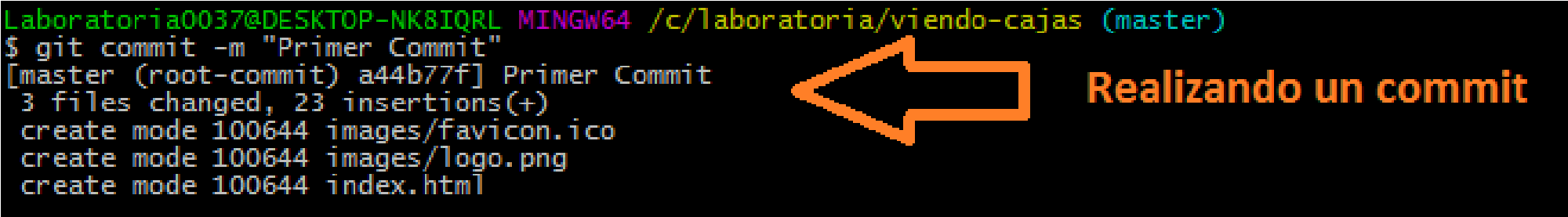
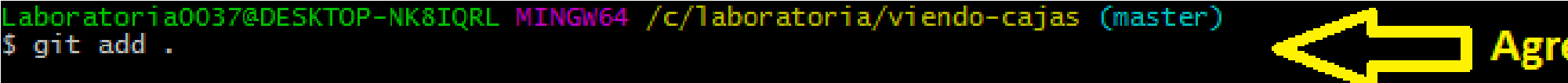
Github



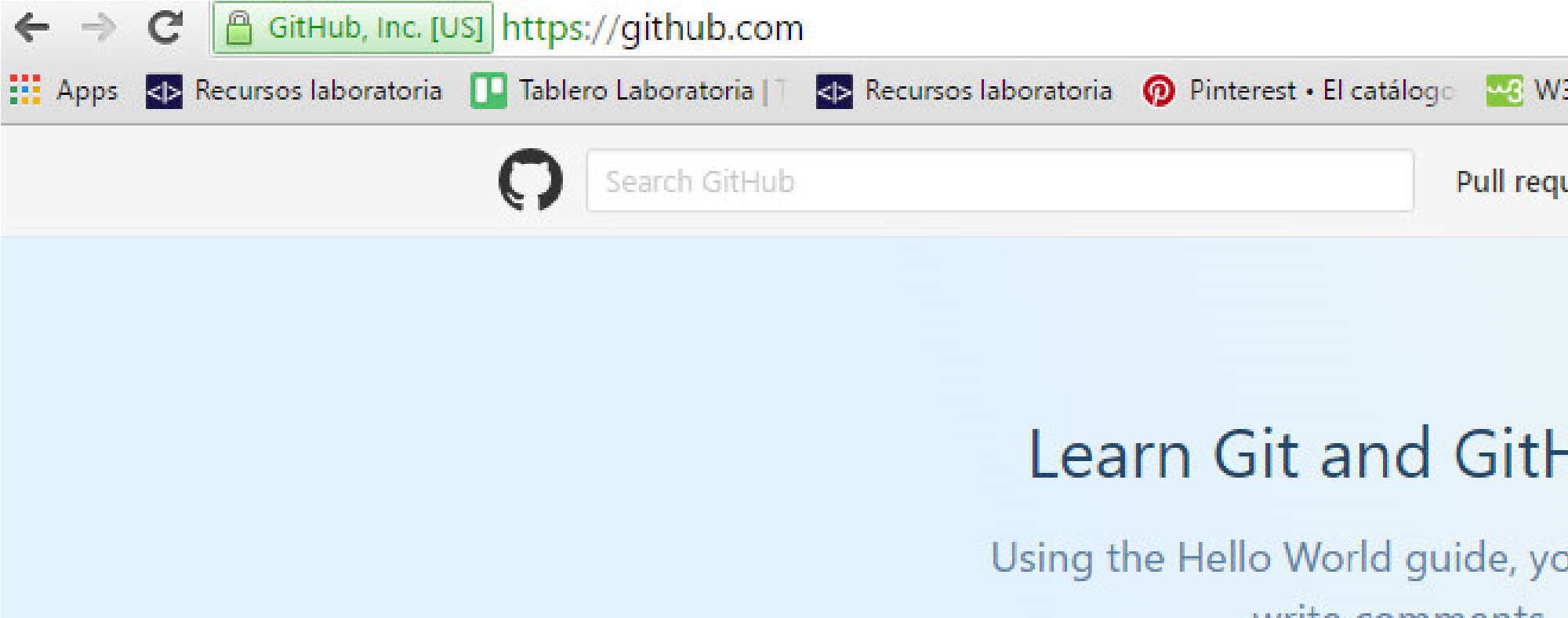
Luego de realizar los primeros pasos y tener las credenciales correctas ya no es necesario realizar los pasos anteriores con este ejemplo aprenderás como Git puede ser más sencillo de lo que imaginas.



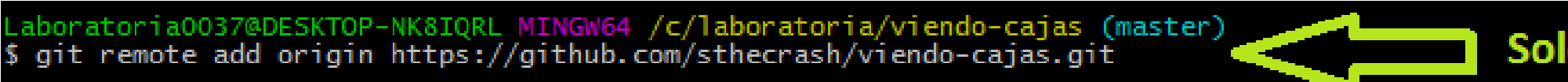
Luego de esto se va a la página de github



(https://github.com) y se crea un nuevo repositorio.

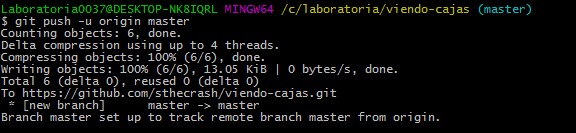


Utilizamos el comando git remote + la url del repositorio que creamos en la página de github.



Ahora realizamos un push para que aparezcan los cambios en el github

# Clonando y Colaborando un repositorio de Github



Si bien vimos como clonar esta vez será diferente por que trabajaremos con ramas(branch) para poder colaborar de forma eficiente en nuestro proyecto.

1. Situarse en el directorio donde traeremos nuestro repositorio en este caso lo pondré en C:/ git clone [URL DEL PROYECTO]

Nota: Se pone la Url de github del proyecto que queremos colaborar.

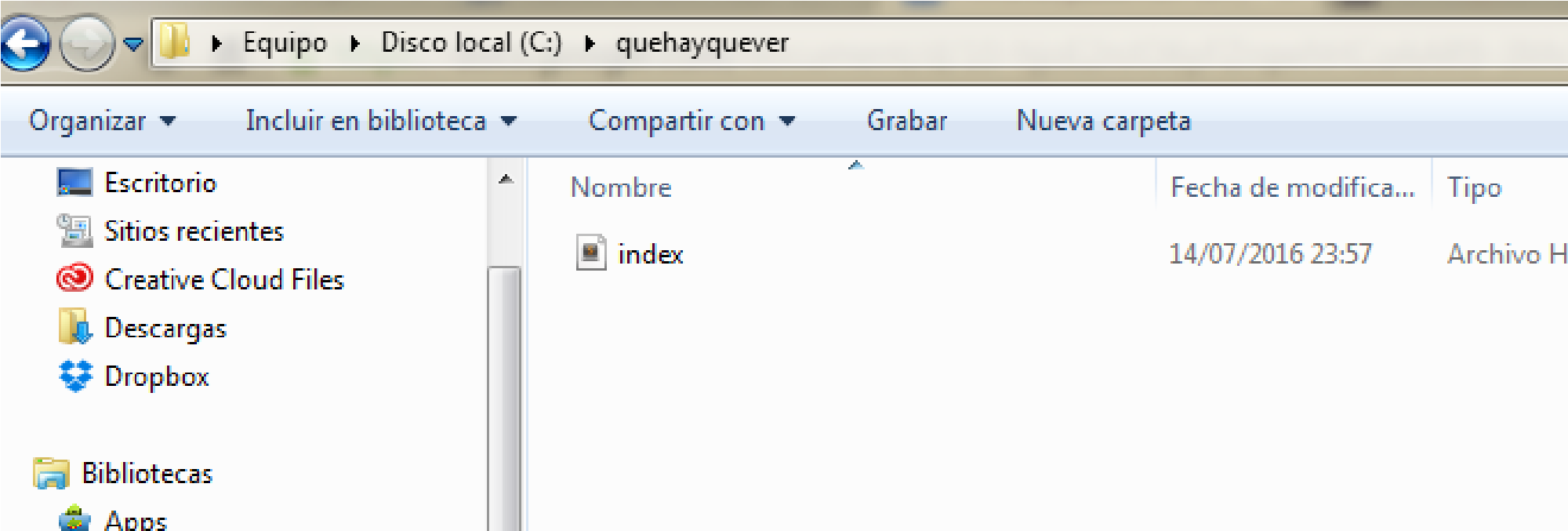
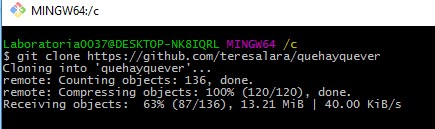
2

.



Resultado :

**Colaborando en Github**



Es importante que para colaborar puedas tener permisos y esto se realiza de la siguiente forma:

|  |
| --- |
|  |

3

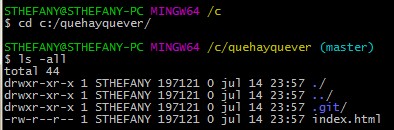
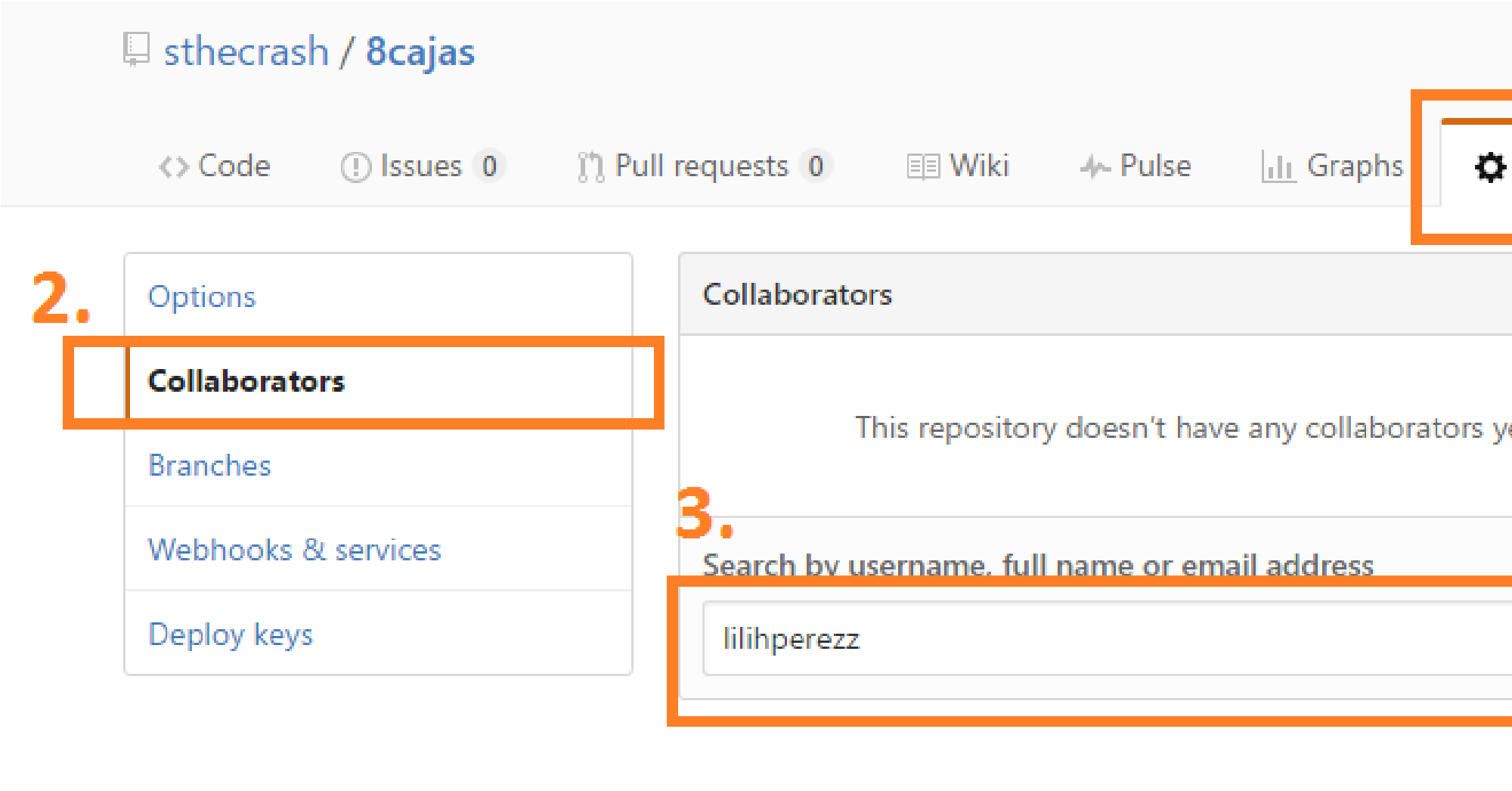
. Nos situamos en la carpeta y ya estamos conectados con la

master

4

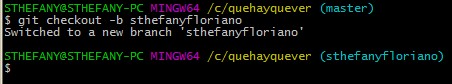
. Creare una rama(branch) para colaborar desde mi

usuario sthecrash



5

. Haré cambios a mi index.html para realizar un commit



6

. Veo los cambios

7

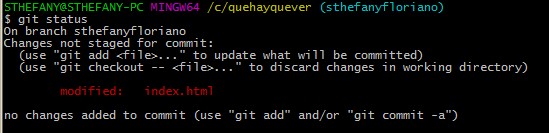
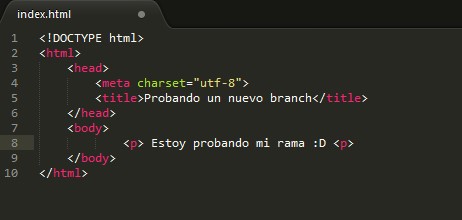
. add

8

. commit

10

. Hacer el push



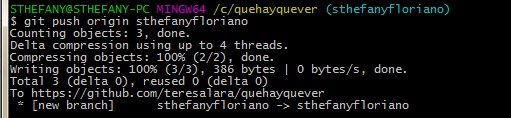
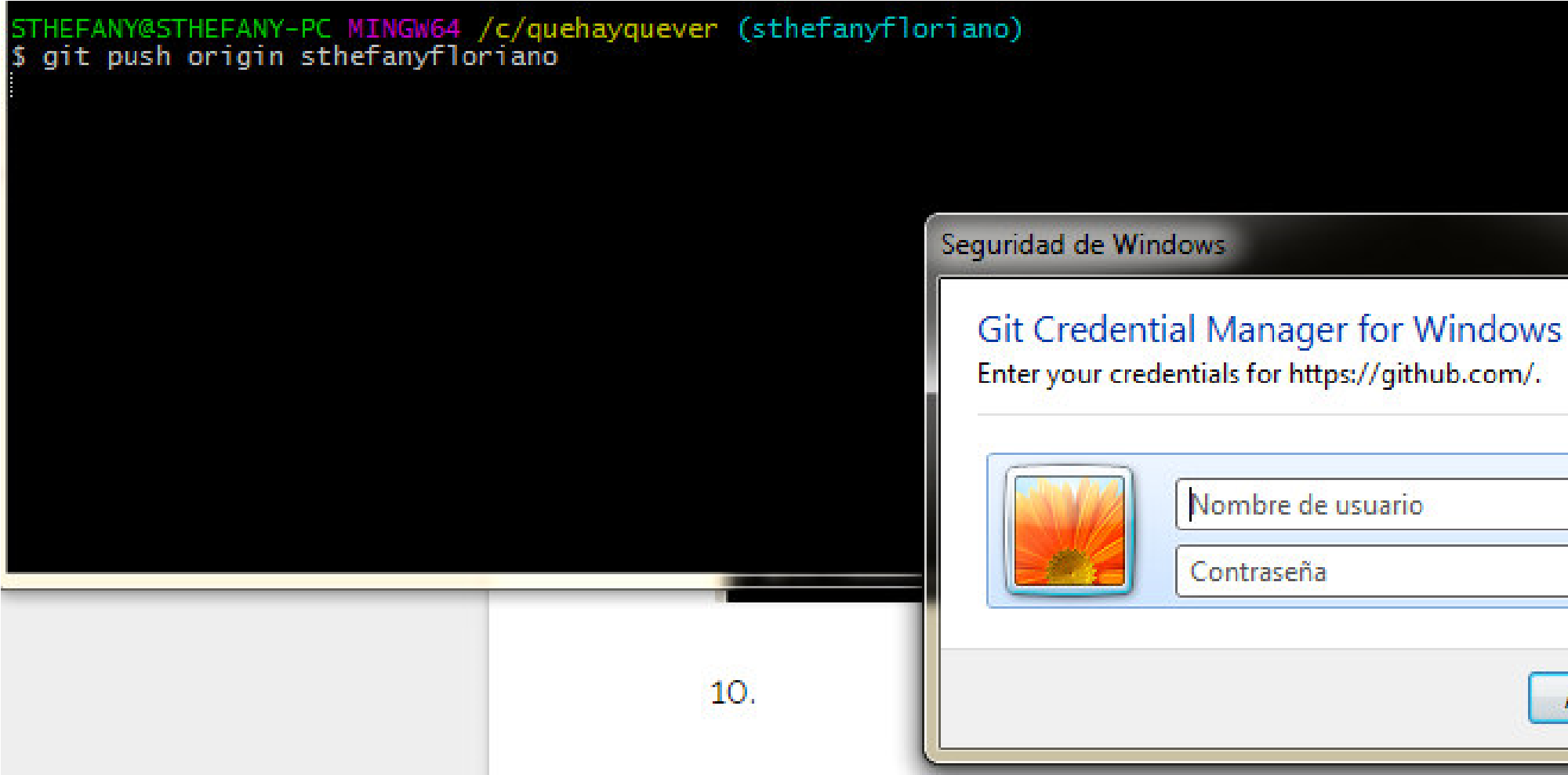
11. llenar con el nombre y contraseña del github

|  |
| --- |
|  |

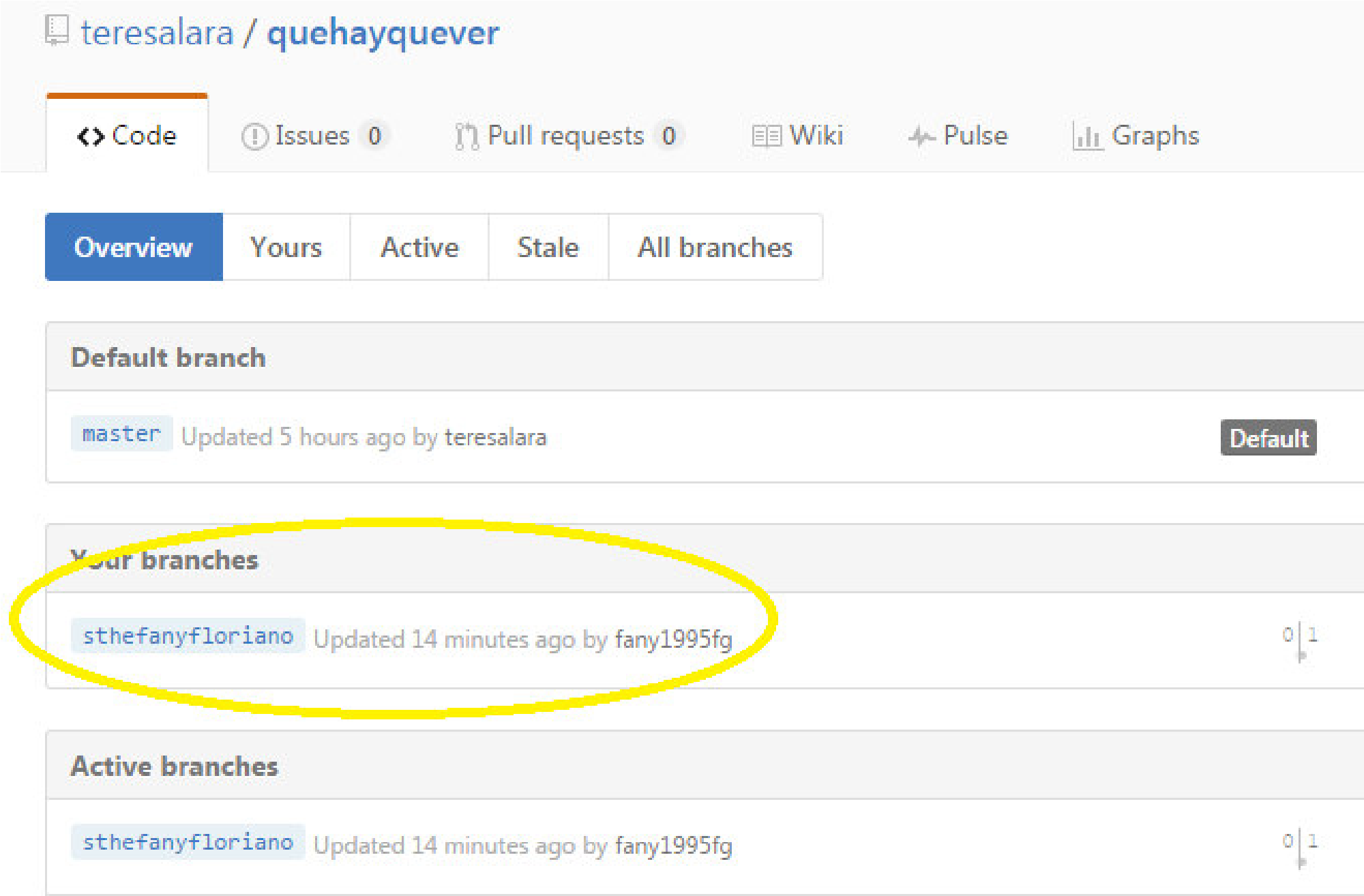
12

. Listo acabo de subir mi rama al github

Resultado :



**LABORATORIO**



1. Crear carpeta de trabajo.
2. Clonar Repositorio desde https://github.com/fsantolaria/labo-git-i210.git
3. Crear un archivo [apellido].txt, [apellido].js
4. Agragar archivos al repositorio
5. Verificar estado de los archivos 6. Hacer consolidar los cambios

7. Verificar estado de los archivos 8. Subir los cambios al repositorio remoto

1. Crear una rama llamada [apellido].
2. Cambiar a la rama [apellido].

11.Crear un archivo [apellido].html

1. Consolidar los cambios
2. Cambiar a la rama master y traer los cambios de la rama apellido.