

# **Visual Studio Code y Git**

## Índice

1.- Sistemas de control de versiones.....	3
2.- Git.....	4
3.- Crear cuenta en Github.....	5
4.- Git con Visual Studio Code.....	8

# 1.- Sistemas de control de versiones

Un control de versiones es un sistema que registra los cambios realizados en un archivo o conjunto de archivos a lo largo del tiempo, de modo que puedas recuperar versiones específicas más adelante.

Si eres diseñador gráfico o de web y quieres mantener cada versión de una imagen o diseño (es algo que sin duda vas a querer), usar un sistema de control de versiones es una decisión muy acertada. Dicho sistema te permite regresar a versiones anteriores de tus archivos, regresar a una versión anterior del proyecto completo, comparar cambios a lo largo del tiempo, ver quién modificó por última vez algo que pueda estar causando problemas, ver quién introdujo un problema y cuándo, y mucho más. Usar un sistema de control de versiones también significa generalmente que si arruinas o pierdes archivos, será posible recuperarlos fácilmente.

Podemos encontrarnos con tres tipos de sistemas de control de versiones:

- **Sistema de control de versiones locales.** Un método de control de versiones, usado por muchas personas, es copiar los archivos a otro directorio (quizás indicando la fecha y hora en que lo hicieron, si son ingeniosos). Este método es muy común porque es muy sencillo, pero también es tremendamente propenso a errores. Es fácil olvidar en qué directorio te encuentras y guardar accidentalmente en el archivo equivocado o sobrescribir archivos que no querías.

Para afrontar este problema los programadores desarrollaron hace tiempo sistemas de control de versiones locales que contenían una simple base de datos, en la que se llevaba el registro de todos los cambios realizados a los archivos.

- **Sistemas de control de versiones centralizados.** El siguiente gran problema con el que se encuentran las personas es que necesitan colaborar con desarrolladores en otros sistemas. Los sistemas de Control de Versiones Centralizados fueron desarrollados para solucionar este problema. Estos sistemas, como CVS, Subversion y Perforce, tienen un único servidor que contiene todos los archivos versionados y varios clientes que descargan los archivos desde ese lugar central.

Esta configuración ofrece muchas ventajas, especialmente frente a sistemas de control de versiones locales. Por ejemplo, todas las personas saben hasta cierto punto en qué están trabajando los otros colaboradores del proyecto. Los administradores tienen control detallado sobre qué puede hacer cada usuario, y es mucho más fácil administrar un sistema de control de versiones centralizado que tener que lidiar con bases de datos locales en cada cliente.

Sin embargo, esta configuración también tiene serias desventajas. La más obvia es el punto único de fallo que representa el servidor centralizado. Si ese servidor se cae durante una hora, entonces durante esa hora nadie podrá colaborar o guardar cambios en archivos en los que hayan estado trabajando. Si el disco duro en el que se encuentra la base de datos central se corrompe, y no se han realizado copias de seguridad adecuadamente, se perderá toda la información del proyecto, con excepción de las copias instantáneas que las personas tengan en sus máquinas locales. Los sistemas de control de versiones locales sufren de este mismo problema: Cuando tienes toda la historia del proyecto en un mismo lugar, te arriesgas a perderlo todo.

- **Sistemas de control de versiones distribuido.** Los sistemas de Control de Versiones Distribuidos ofrecen soluciones para los problemas que han sido mencionados. En los sistemas de control de versiones distribuidos, los clientes no solo descargan la última copia instantánea de los archivos del servidor, sino que se replica completamente el repositorio. De esta manera, si un servidor deja de funcionar y estos sistemas estaban colaborando a través de él, cualquiera de los repositorios disponibles en los clientes puede ser copiado al servidor con el fin de restaurarlo. Cada clon es realmente una copia completa de todos los datos.

Además, muchos de estos sistemas se encargan de manejar numerosos repositorios remotos con los cuales pueden trabajar, de tal forma que puedes colaborar simultáneamente con diferentes grupos de personas en distintas maneras dentro del mismo proyecto. Esto permite establecer varios flujos de trabajo que no son posibles en sistemas centralizados, como pueden ser los modelos jerárquicos.

Un ejemplo de sistemas de control de versiones distribuido es el Git.

## 2.- Git

Git es un sistema de control de versiones desarrollado en 2005 por Linus Thorvalds, el creador de Linux, y publicado bajo la licencia de software libre GPLv2 de GNU. La particularidad de esta herramienta es que, aunque guarda un **repositorio central** para cada proyecto, todos los participantes descargan una **copia local** del mismo en su propio dispositivo. Cada una de estas copias constituye una copia completa de todo el contenido del repositorio, por lo que no es necesario estar conectado a la red para trabajar. Además, estos archivos sirven como copia de seguridad en caso de que el repositorio principal falle o resulte dañado. Los **cambios** en los archivos pueden **intercambiarse con todos los demás participantes del proyecto** en cualquier momento y, si corresponde, añadirse al repositorio.

Todas estas razones hacen que en programación sea muy útil el uso de un sistema de control de versiones como pueda ser git.

Git tiene tres estados principales en los que se pueden encontrar tus archivos: confirmado (committed), modificado (modified), y preparado (staged):

- **Modificado (modified).** Significa que has modificado el archivo pero todavía no lo has confirmado a tu base de datos.
- **Preparado (staged).** Significa que has marcado un archivo modificado en su versión actual para que vaya en tu próxima confirmación.
- **Confirmado (committed).** Significa que los datos están almacenados de manera segura en tu base de datos local.

Esto lleva a que la forma de trabajar de forma básica en Git es algo como:

1. Modificas una serie de archivos en tu directorio de trabajo.
2. Preparas los archivos, añadiéndolos a tu área de preparación.
3. Confirmas los cambios, lo que toma los archivos tal y como están en el área de preparación y almacena esa copia instantánea de manera permanente en tu directorio de Git.

Si una versión concreta de un archivo está en el directorio de Git, se considera confirmada (committed). Si ha sufrido cambios desde que se obtuvo del repositorio, pero ha sido añadida al área de preparación, está preparada (staged). Y si ha sufrido cambios desde que se obtuvo del repositorio, pero no se ha preparado, está modificada (modified).

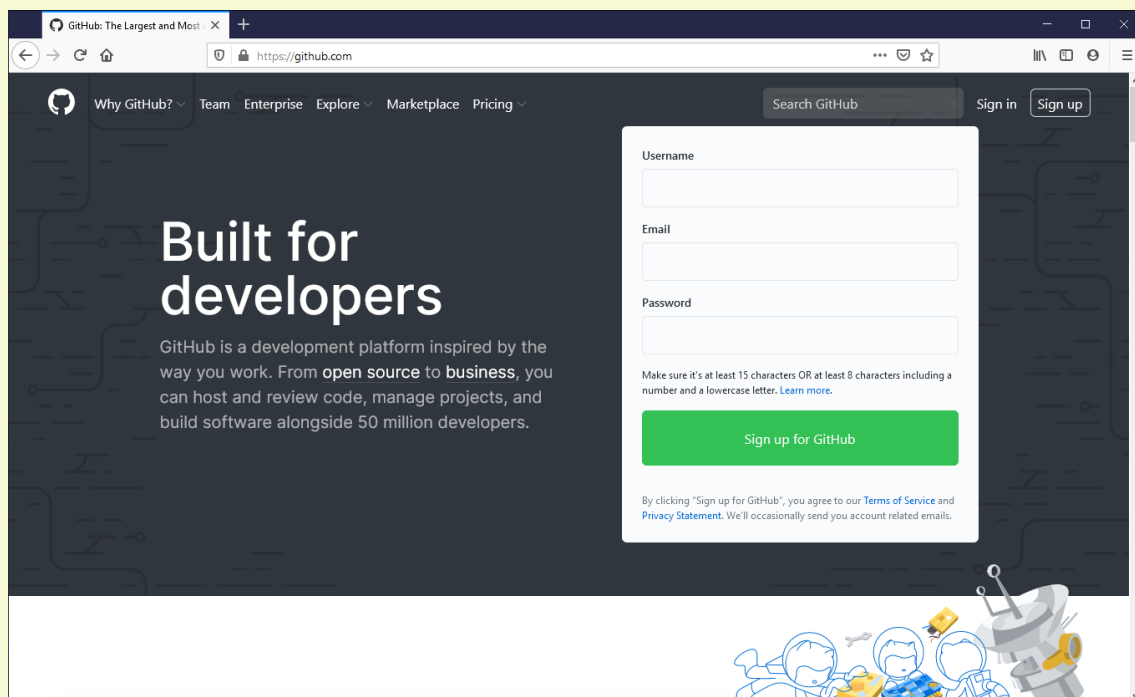
Para trabajar con Git de forma distribuida tendremos que emplear un servidor que almacene nuestros ficheros (el repositorio). Los repositorios pueden ser de dos tipos:

- **Públicos**, accesibles para todo el mundo, y que son útiles para compartir código libre.
- **Privados**, accesibles solo para determinados usuarios a los que hay que proporcionarle acceso personalmente.

Existen diversos proveedores que nos permiten crear repositorios. La mayoría de ellos permiten crear repositorios públicos gratuitamente, pero algunos también proporcionan repositorios privados, pero limitan el número de usuarios que pueden acceder al repositorio privado. Uno de esos proveedores es Github.

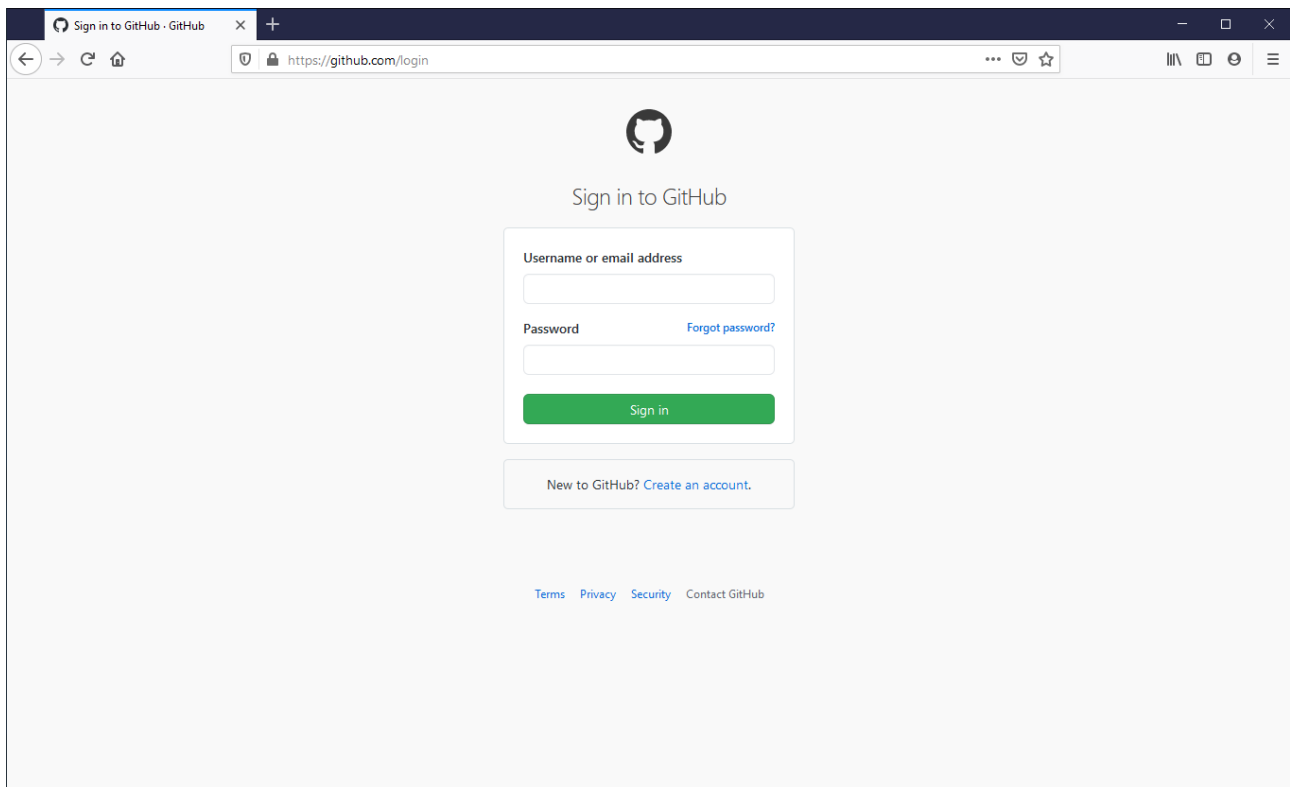
### 3.- Crear cuenta en Github

Para crear una cuenta en github visitaremos a la página web [www.github.com](https://github.com)

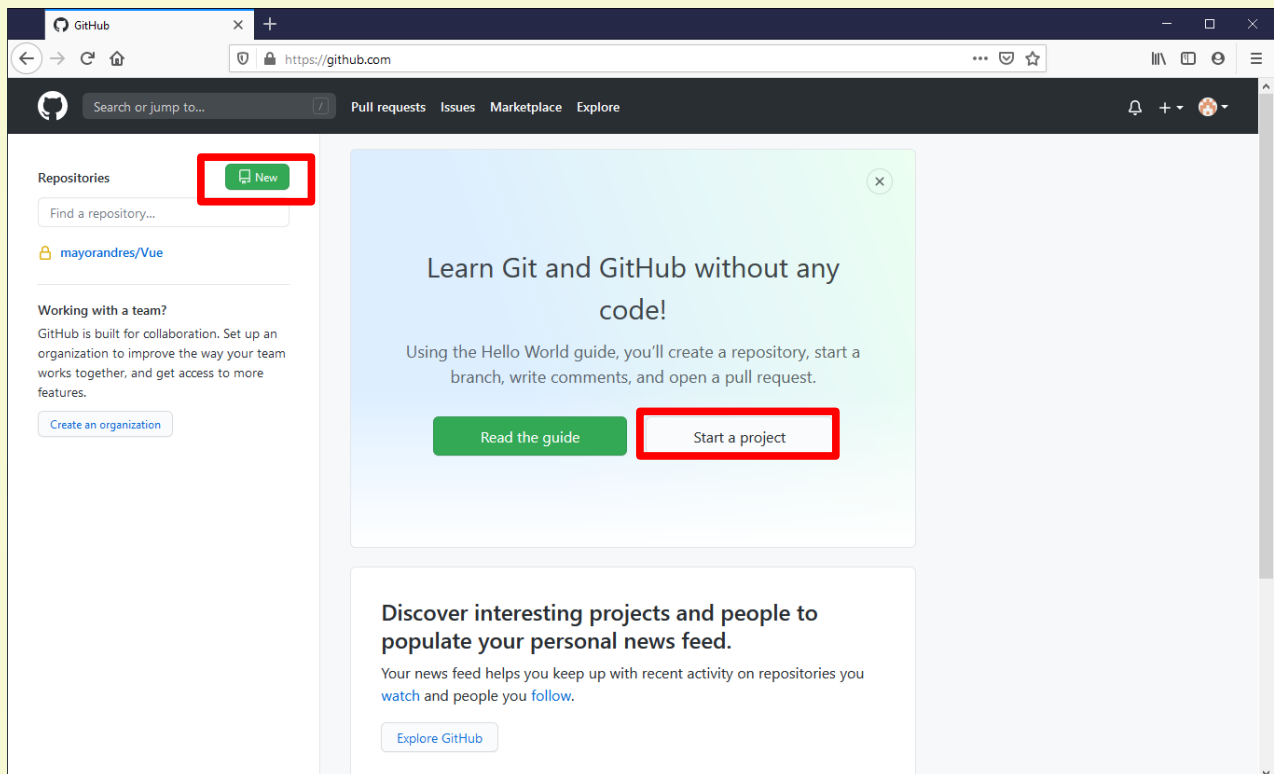


y completaremos los datos del usuario, correo electrónico y contraseña. De esta forma se nos enviará un correo electrónico que hayamos indicado para confirmar la creación de la cuenta.

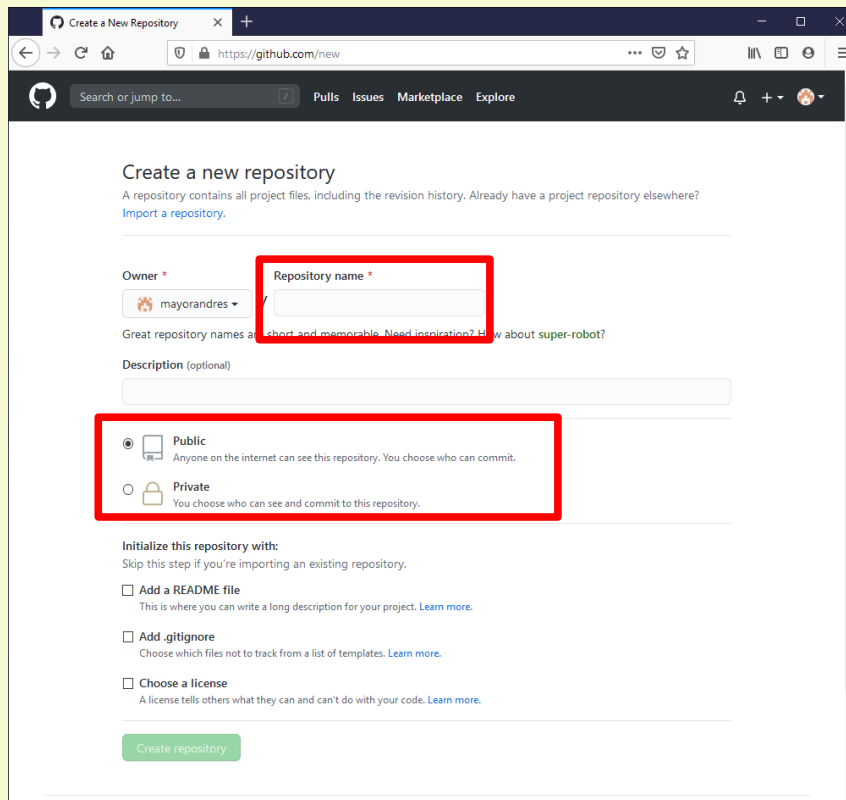
Una vez confirmada la creación de la cuenta, ya podemos ir de nuevo a [www.github.com](https://github.com) y pulsando en el enlace “Sign in” pasar a validarnos.



Después de validarnos accederemos a nuestra cuenta:

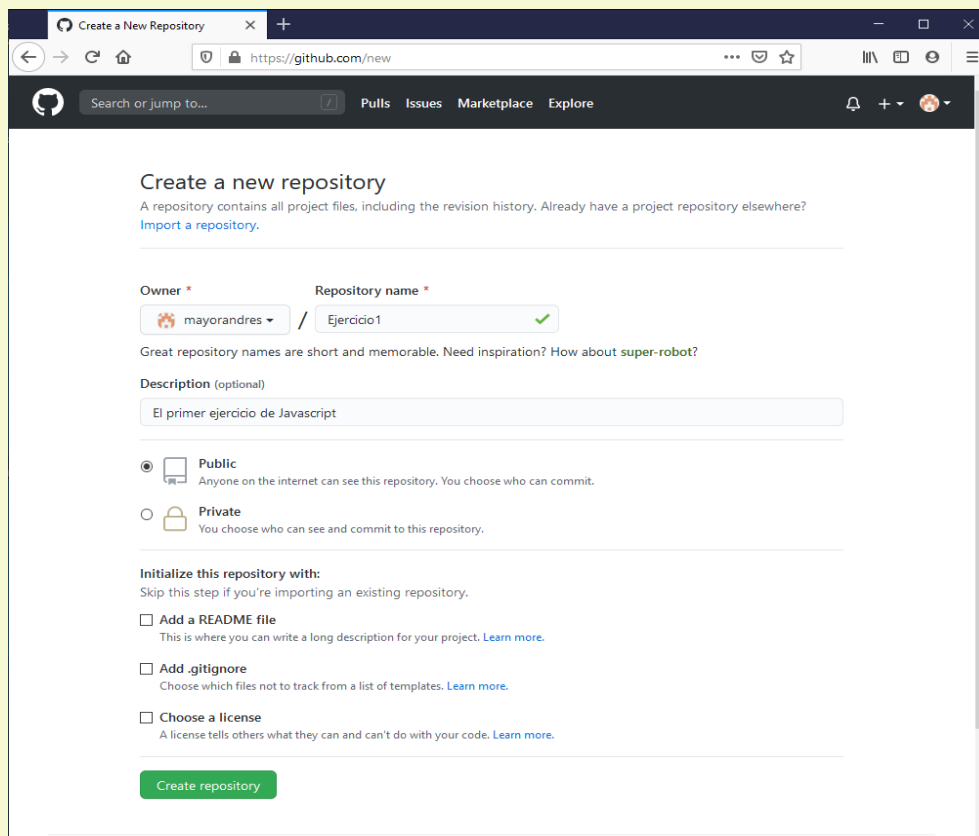


A través del botón “New” o “Start a project” podremos crear nuestro repositorio:



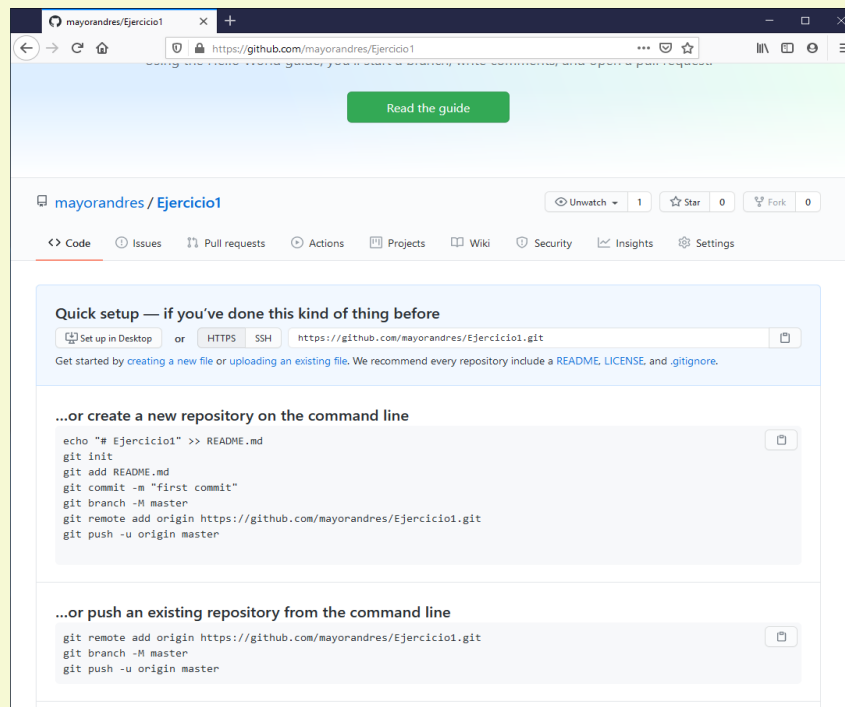
The screenshot shows the GitHub 'Create a new repository' page. The 'Repository name' field is highlighted with a red box. Below it, the 'Public' and 'Private' radio button options are also highlighted with a red box. The 'Public' option is selected. The 'Create repository' button is visible at the bottom.

Pondremos el nombre al repositorio (Repository name) y elegimos la visibilidad del proyecto (Publico o Privado). Ya podemos pulsar sobre el botón “Create repository” para crear el repositorio.



The screenshot shows the GitHub 'Create a new repository' page with the repository name 'Ejercicio1' entered in the 'Repository name' field. The 'Public' option is selected for visibility. The description 'El primer ejercicio de Javascript' is entered in the 'Description (optional)' field. The 'Create repository' button is visible at the bottom.

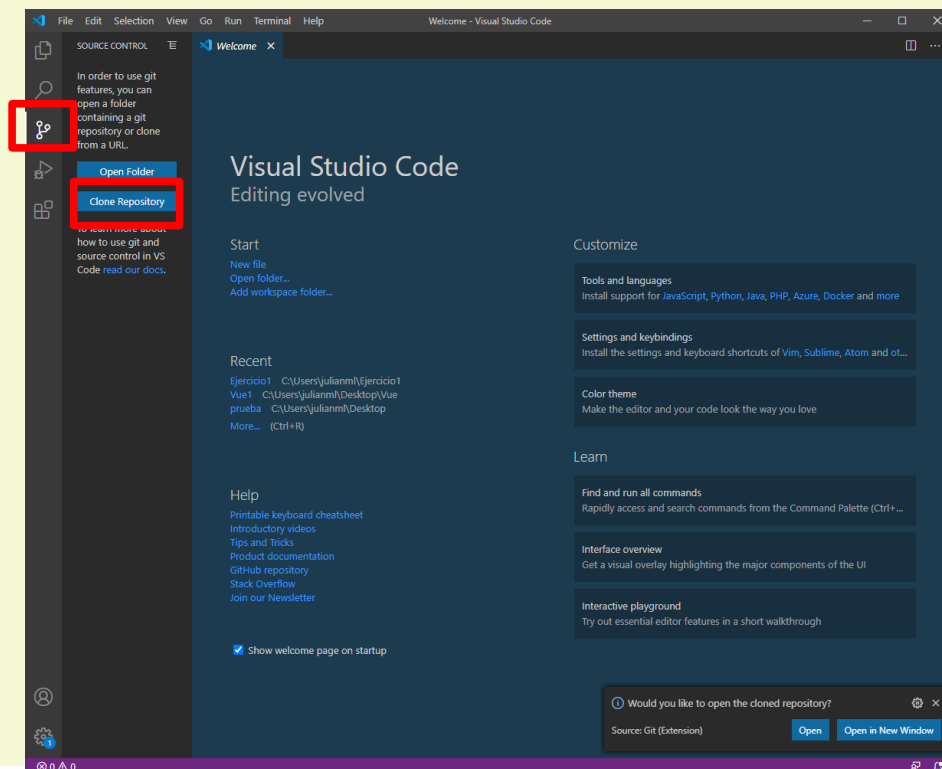
En ese momento tendremos el repositorio vacío y nos proporcionará instrucciones para, mediante comandos, como configurar localmente el repositorio local para poder subirlo al repositorio remoto, pero nosotros lo haremos de forma más sencilla. Usaremos el Visual Studio Code a partir de ahora.



## 4.- Git con Visual Studio Code

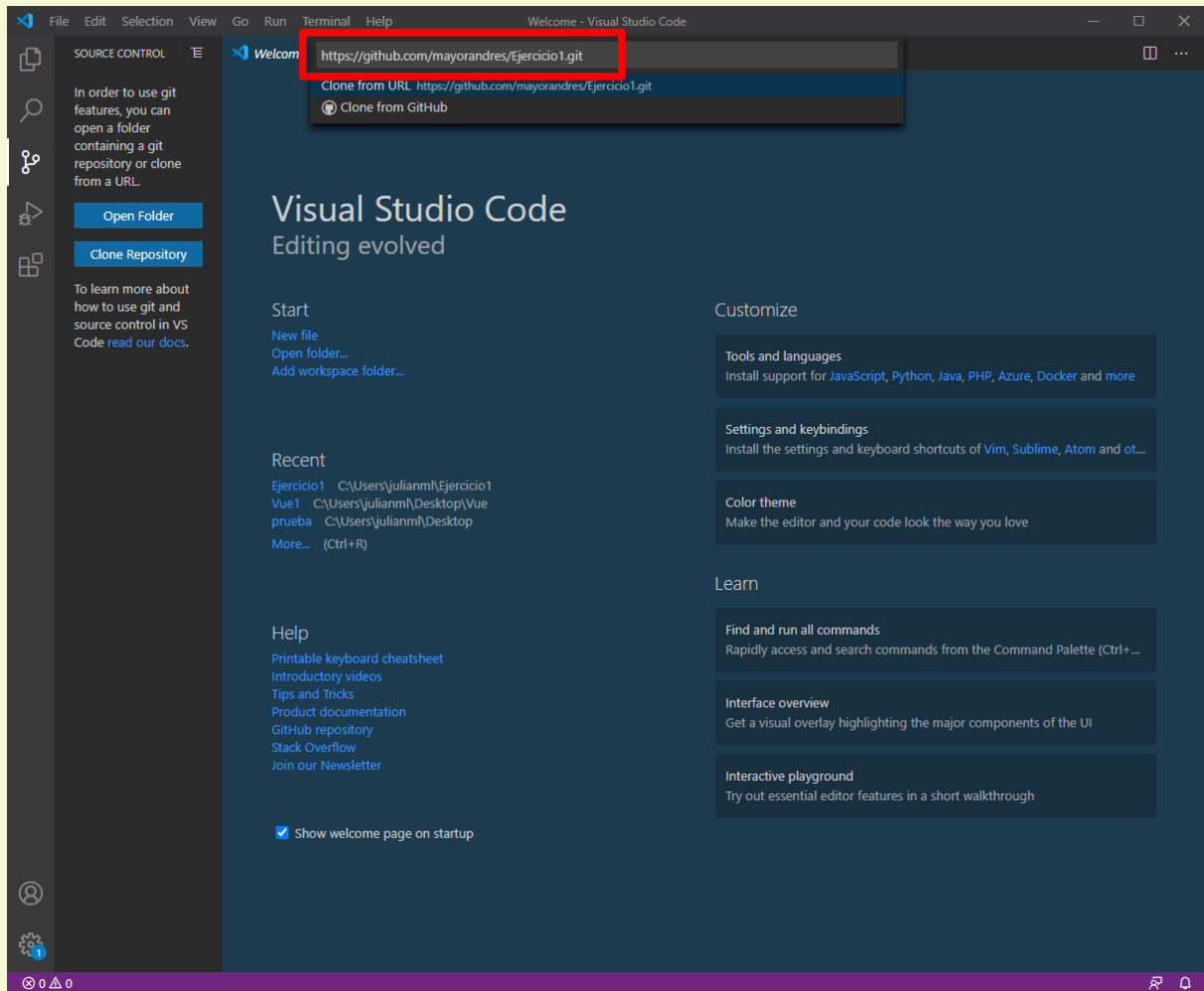
Emplear el Visual Studio Code con el sistema de control de versiones git es mucho más sencillo de hacer si partimos de que ya tenemos el repositorio creado en un proveedor como Github.

Para ello solo tenemos que abrir el Visual Studio y acceder al menú lateral que se encarga del sistema de control de versiones.

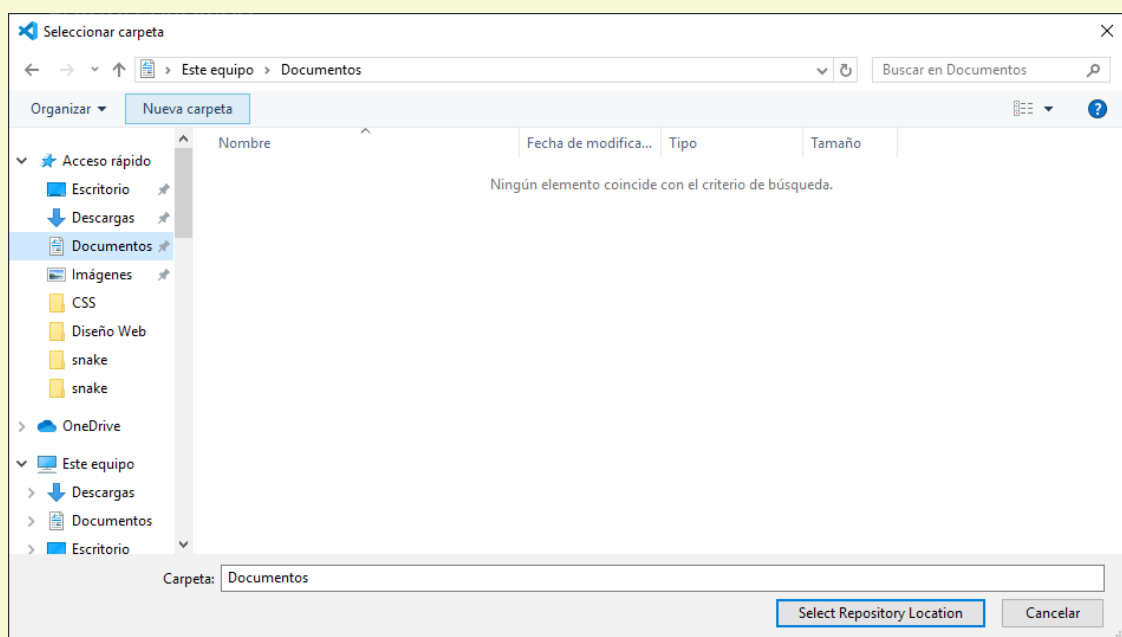


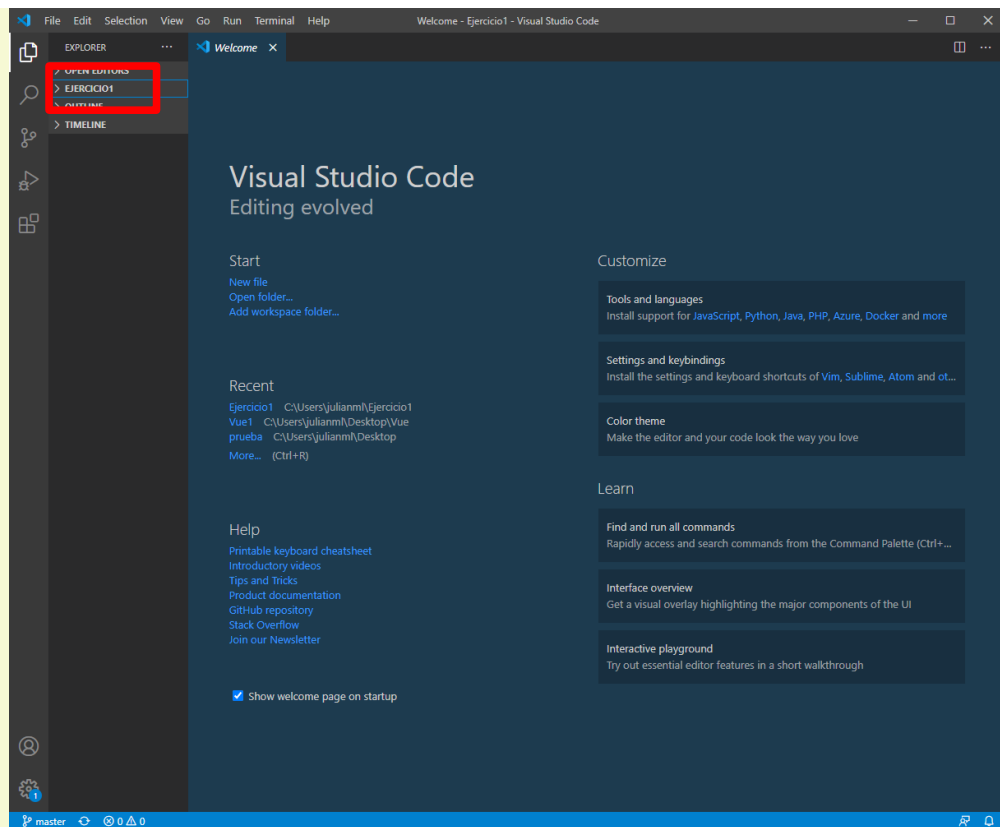


Pulsaremos sobre el botón “Clone Repository”

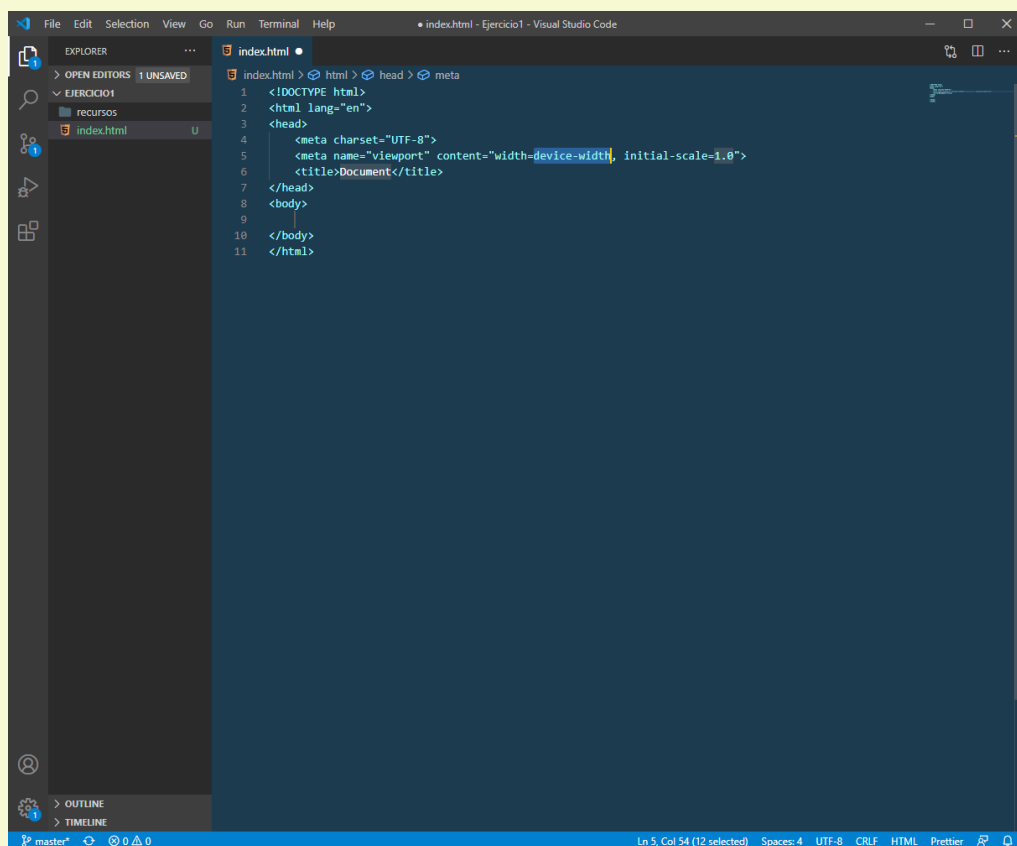


En ese momento tendremos que indicarle la dirección del repositorio remoto de github e indicarle en qué carpeta queremos copiarlo



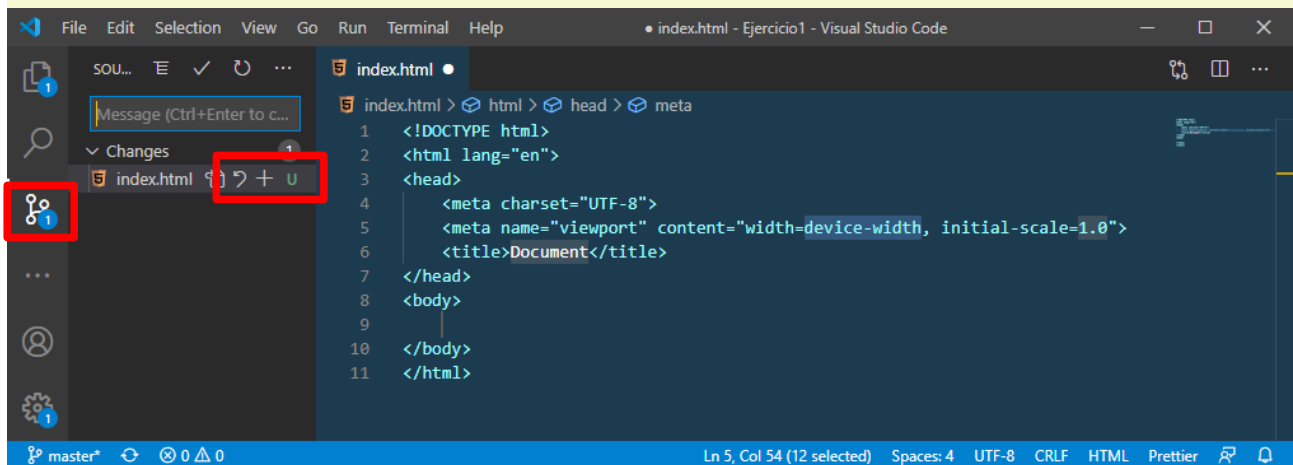


Y ya tendremos el repositorio remoto clonado a nuestro equipo, aunque estará vacío. Podemos crear los ficheros para nuestro proyecto (o copiarlos a la carpeta del repositorio local)



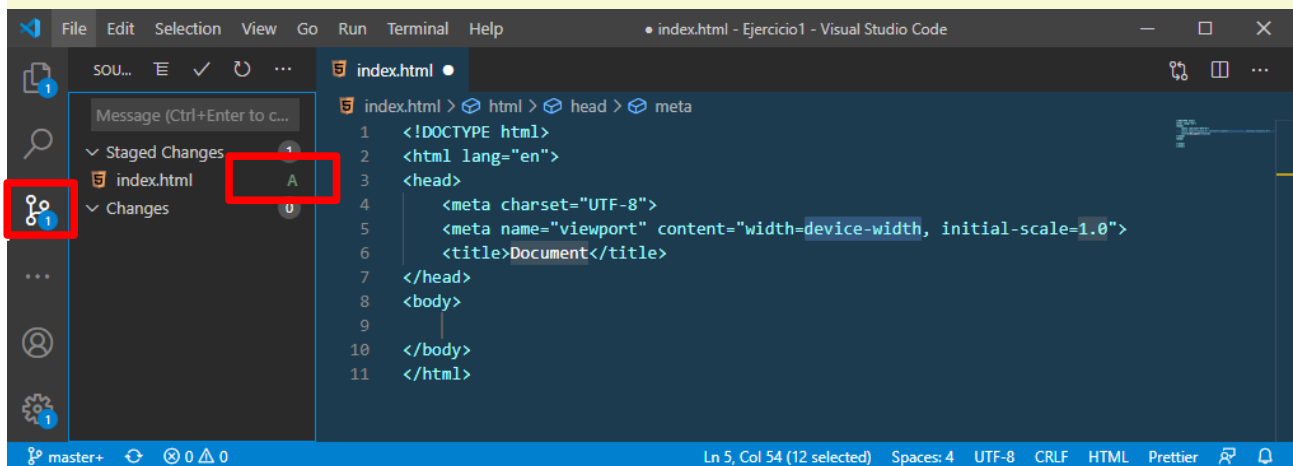
Una vez creados o copiados a la carpeta y guardado los cambios, se puede ver que en la opción de menú que se encuentra en el lateral izquierdo ya notifica cambios en el proyecto. Es decir, estamos

en el estado modificado (modified). Ahora tenemos que indicar al repositorio que empiece a hacer seguimiento de los ficheros que se han modificado para pasar el estado preparado (staged) y para ello pulsaremos sobre el símbolo +



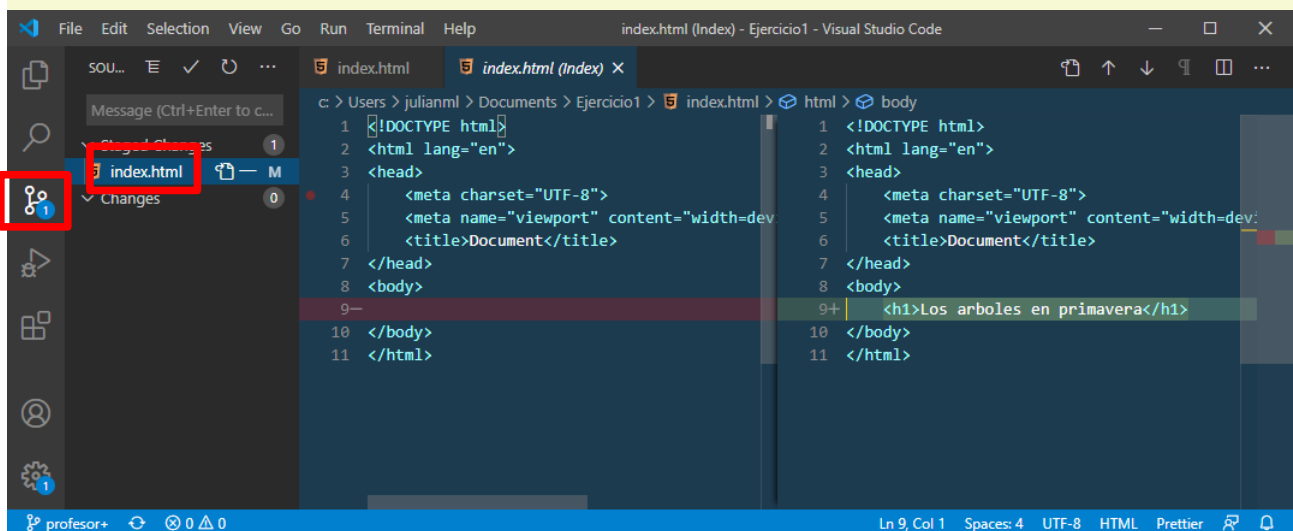
```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>Document</title>
7 </head>
8 <body>
9
10 </body>
11 </html>
```

Y veremos como la letra que hay al lado del fichero para de ser U a ser A.



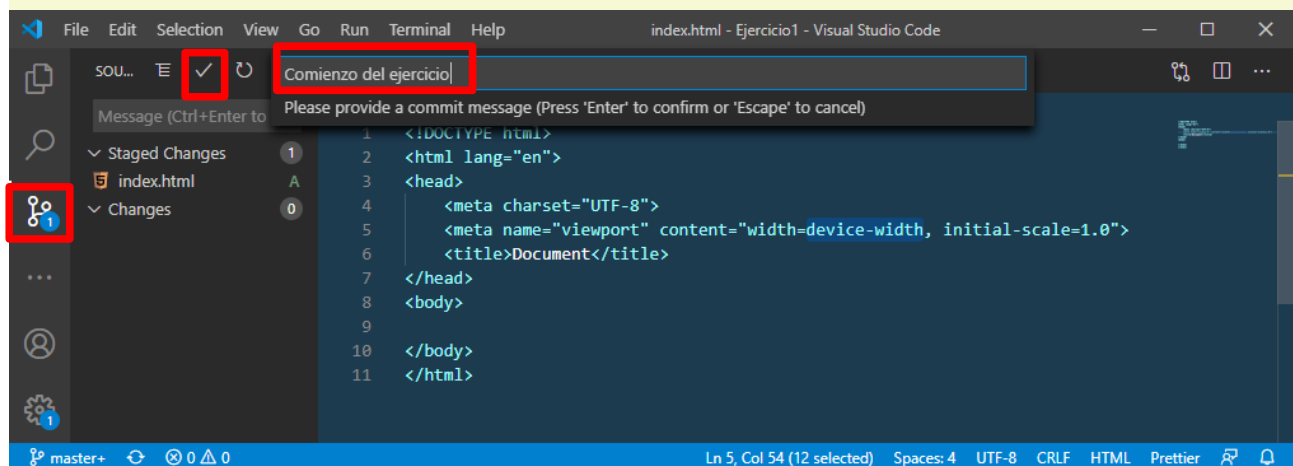
```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>Document</title>
7 </head>
8 <body>
9
10 </body>
11 </html>
```

Si realizamos nuevos cambios, podemos observar los cambios respecto al estado inicial pulsando sobre el fichero (en este caso index.html) y se nos mostrará la diferencia entre ambas versiones

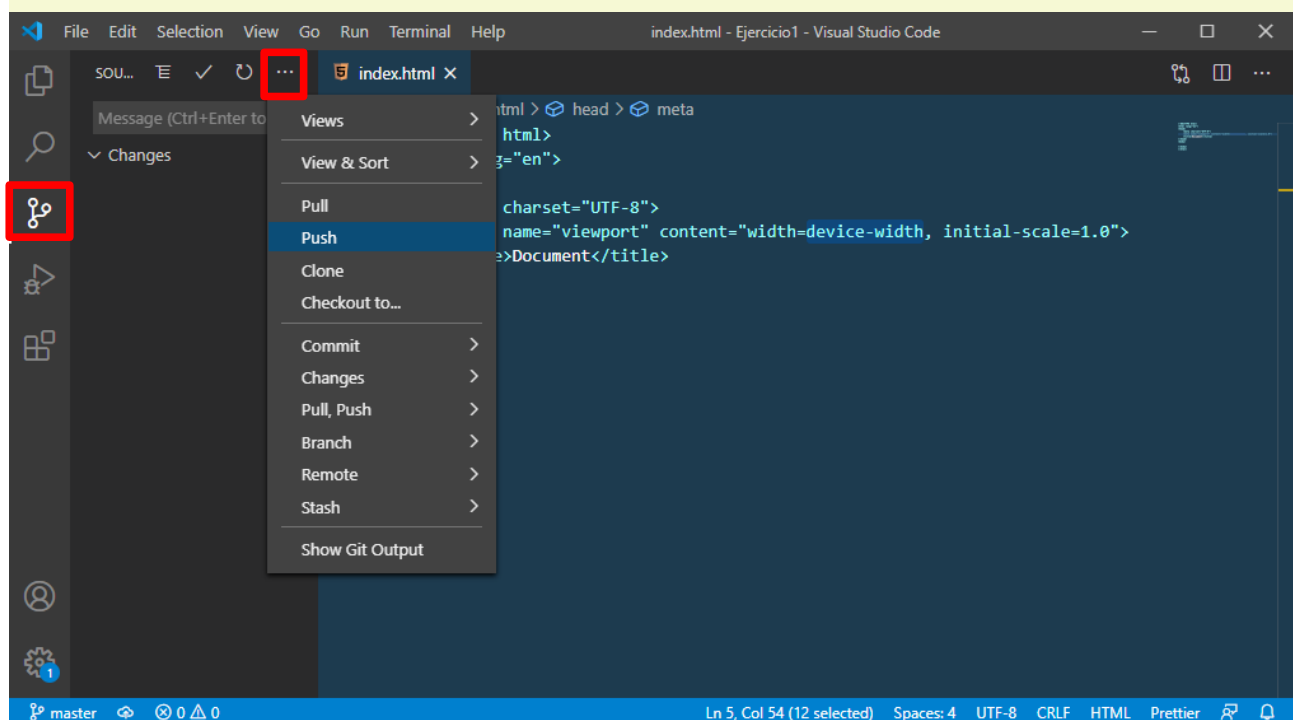


```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>Document</title>
7 </head>
8 <body>
9
10 </body>
11 </html>
```

Una vez creamos que los ficheros del proyecto ya están lo suficientemente bien podemos confirmarlos (pasar al modo confirmado o committed) indicando una referencia para el estado del proyecto confirmado.

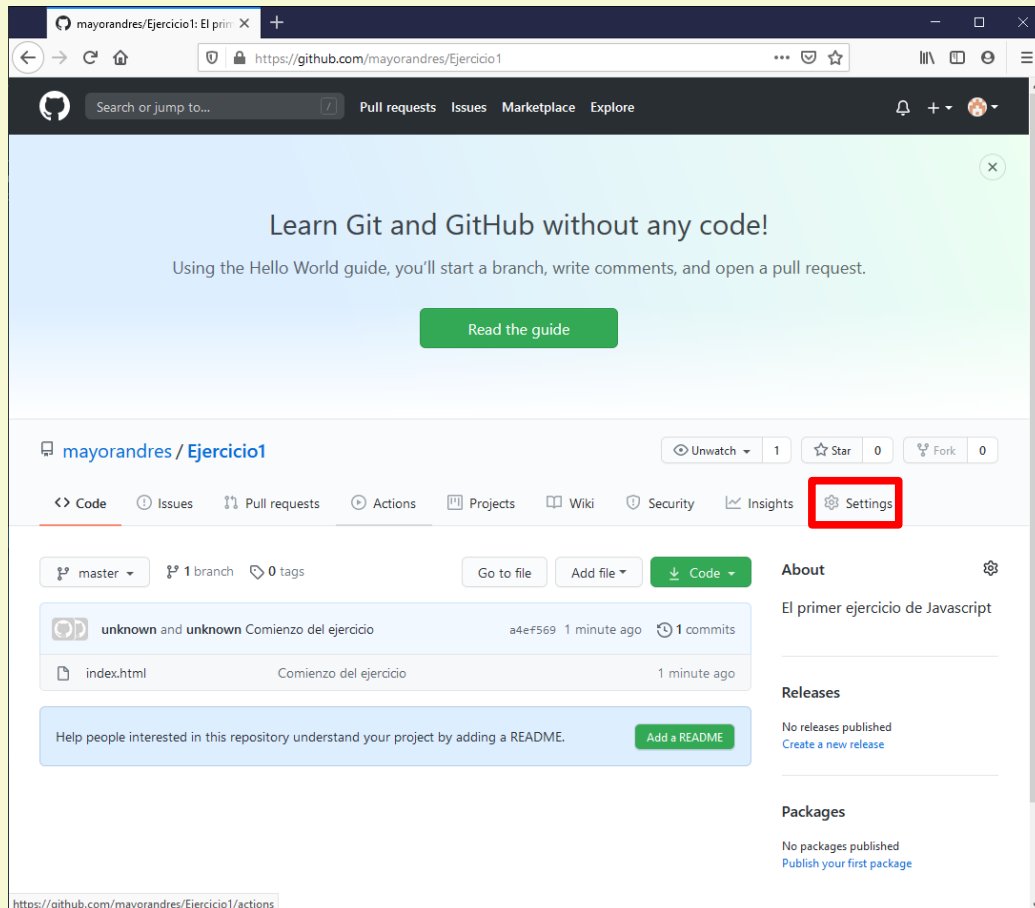


Ya solo nos queda subirlo a github. Para ello accedemos al menú ... del control de versiones y ejecutamos la orden Push

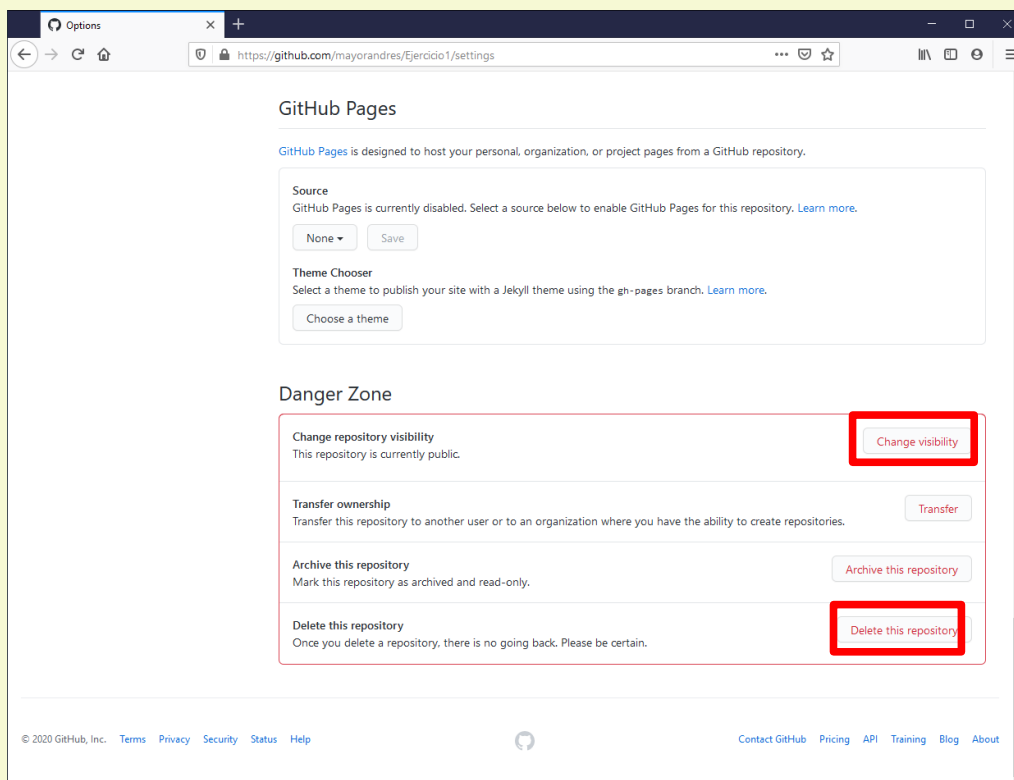


Resumiendo: añadimos/creamos/modificamos contenido en la carpeta del repositorio local y los guardamos, indicamos que esos contenidos añadidos/creados/modificados están preparados, confirmamos que queremos guardarlos de forma segura en el repositorio local con un identificador y, finalmente, lo subimos.

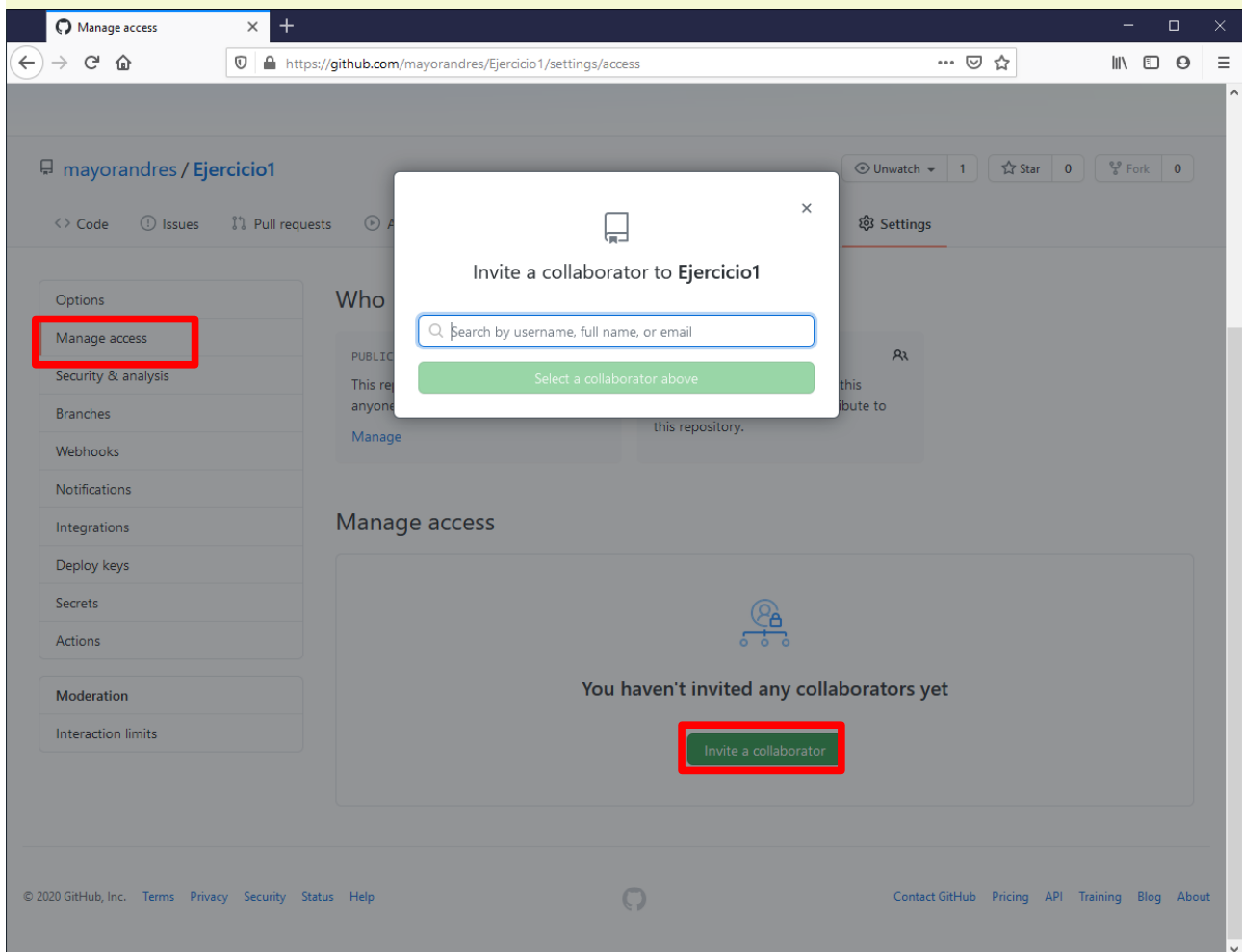
Si visitamos github veremos que nuestro proyecto ya ha sido subido



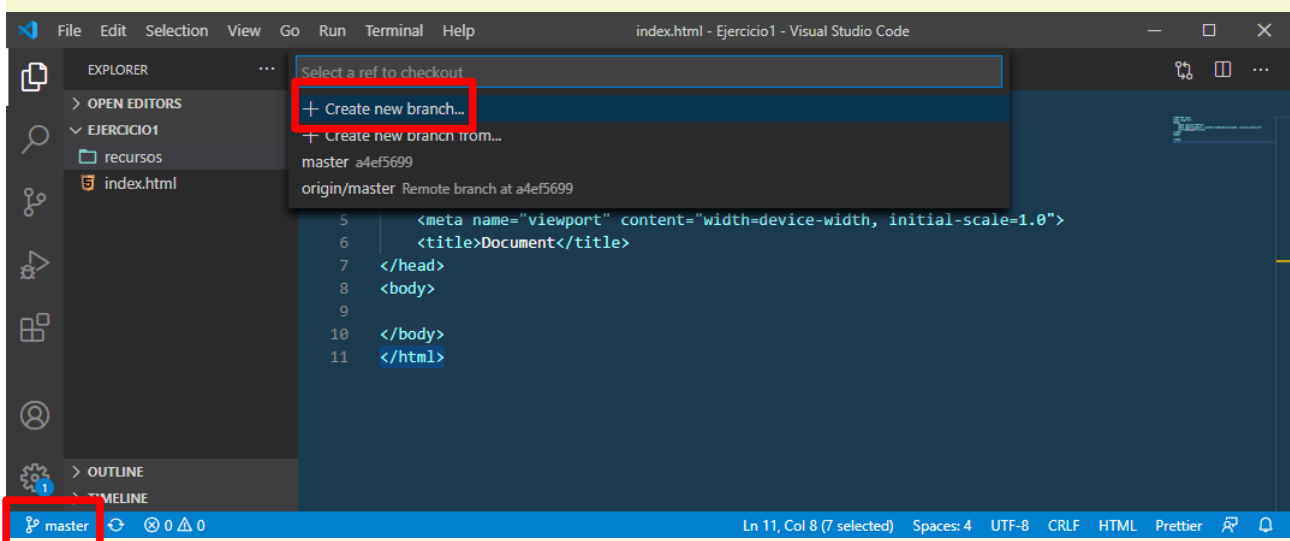
En caso de que queramos cambiar la visibilidad o borrar el proyecto podemos acceder al apartado “Settings” del proyecto correspondiente y al final de todas las opciones tendremos una serie de botones para realizar dichas tareas, pidiéndonos confirmación del cambio/borrado.



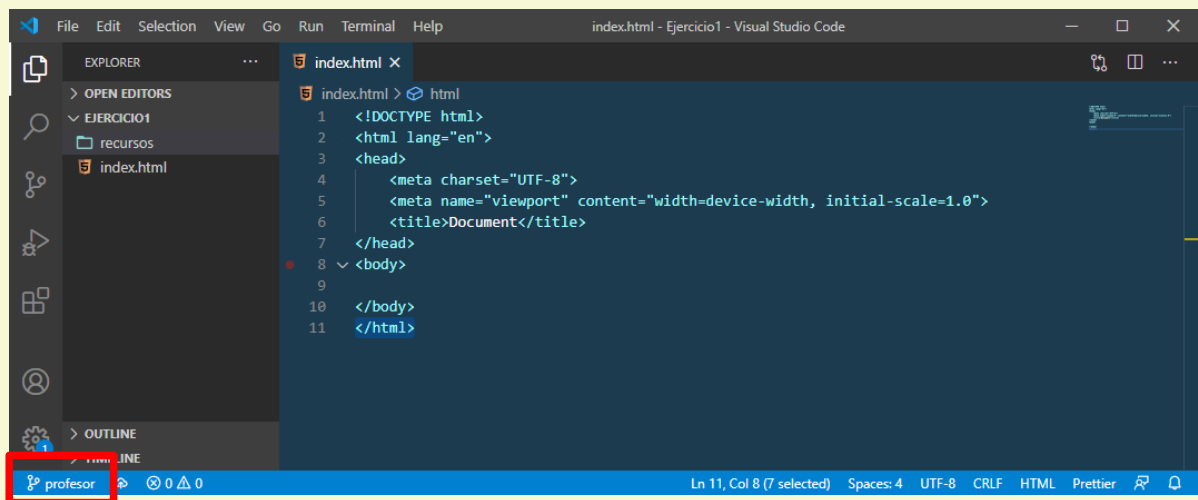
Si el proyecto es privado y queremos que otra/s persona/s puedan acceder tenemos que proporcionar acceso a esas personas enviándoles una invitación a participar en el proyecto. Eso lo realizamos a través de la opción “Manage Access” del menú “Settings” del proyecto e invitando al colaborador introduciendo su dirección de correo, quien recibirá un correo y deberá confirmar su colaboración.



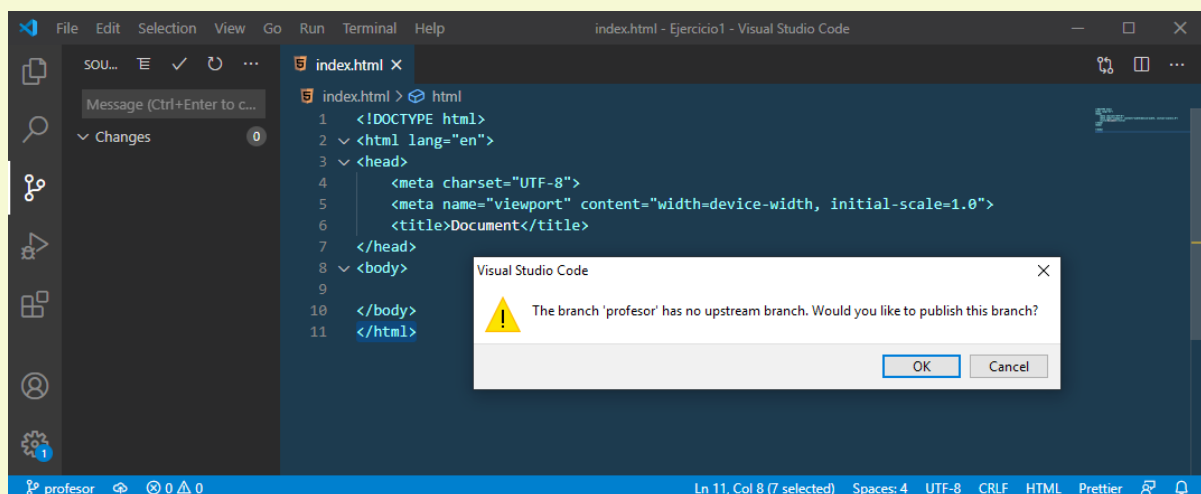
Empleando git podemos crear ramas de desarrollo de nuestro proyecto para incorporar nuevas funcionalidades al mismo.



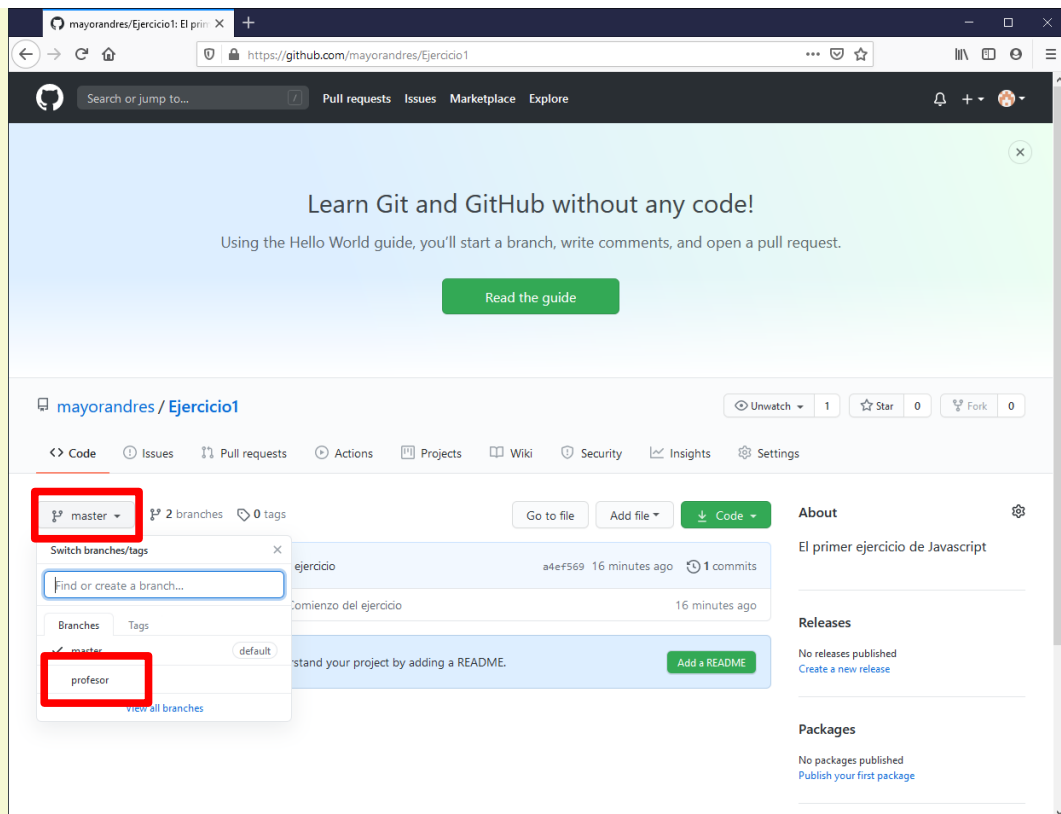
Y después de poner el nombre “profesor” a la rama quedaría



Al hacer un Push, nos avisa que no hay dicha rama en github y si deseamos crearla



Visitando github, ya podemos ver las dos ramas de trabajo



Si tenemos abierto un proyecto clonado de una cuenta de github de otra persona y lo queremos subir a nuestro repositorio, primero creamos un proyecto vacío en nuestra cuenta de github, y después, cambiamos la dirección remota a la que queremos subir el proyecto. Para ello accedemos a la consola de comandos de Visual Studio Code, pulsando Ctrl+Ñ, y cambiamos a dónde subir el proyecto ejecutando en la consola algo similar a:

```
git remote set-url https://github.com/nombre_cuenta_github/nombre_proyecto.git
```

,de forma que al hacer un Push nos subiría todo a nuestro repositorio.

