

## Contenido

1 - QUE ES GIT Y PARA QUÉ SIRVE.....	2
2 - INSTALACIÓN DE GIT .....	2
3 - CONFIGURACIÓN DE GIT .....	3
4 - USO DE GIT .....	3
5 - INICIALIZAR NUESTRA CARPETA CON GIT .....	4
6 - DECIRLE A GIT QUÉ TIENE QUE IR SIGUIENDO.....	5
7 - COMPROBAR EL ESTADO DEL “STAGE AREA” .....	5
8 - GUARDAR VERSIONES DE MI PROYECTO .....	6
9 - VER TODOS LOS COMMITS QUE LLEVAMOS HECHOS .....	6
10 - VOLVER A UN COMMIT ANTERIOR .....	7
11 - DEVOLVER NUESTRO PROYECTO A UN ESTADO ANTERIOR .....	8
12 - HACER UN “git add” Y UN “git commit” A LA VEZ .....	8
13 - GITHUB.....	8
14 - CREAR REPOSITORIOS EN GITHUB .....	9
15 - AÑADIR ARCHIVOS A NUESTRO REPOSITORIO .....	11
16 - SUBIR ARCHIVOS DESDE NUESTRO PC CON GIT A GITHUB:.....	14
17 - CLONAR REPOSITORIOS DE GITHUB EN NUESTRO PC .....	15
18 - ACTUALIZAR UN PROYECTO DESDE GITHUB A NUESTRO PC.....	15
19 - ESTABLECER VERSIONES DE MI PROYECTO.....	16
20 - RAMAS EN GIT.....	16
21 - CREAR UNA RAMA NUEVA.....	16
22 - VER LAS RAMAS EXISTENTES EN EL PROYECTO .....	17
23 - MOVERNOS A OTRA RAMA DEL PROYECTO.....	17
24 - FUSIONAR 2 RAMAS EN 1 SOLA .....	18
25 - BORRAR UNA RAMA.....	19
26 - GIT Y GITHUB EN VISUAL STUDIO CODE .....	19
27 - RAMAS EN VSC .....	20
28 - COMMITS EN VSC.....	20
29 - FUSIONAR RAMAS.....	21
30 - CLONAR REPOSITORIO DE GITHUB DESDE VSC.....	21
31 - SUBIR LOS CAMBIOS A GITHUB.....	22

32 - TRABAJAR CON RAMAS ONLINE EN GITHUB.....	22
33 - CREAR NUEVOS FICHEROS DESDE GITHUB Y GUARDARLOS.....	23
34 - COMPARAR RAMAS DESDE GITHUB .....	23
35 - FORKS EN GITHUB .....	24

## 1 - QUE ES GIT Y PARA QUÉ SIRVE

Cuando estamos desarrollando algún proyecto en algún lenguaje de programación, o cuando estamos elaborando una carpeta con documentos de cualquier tipo, o cuando estamos almacenando en una carpeta ficheros de cualquier tipo, o en cualquier otro caso que se nos pueda ocurrir, es posible que queramos ir haciendo copias de seguridad en determinados momentos de nuestro proyecto.

Para eso podemos ir guardando localmente en nuestro PC distintas carpetas con las distintas fechas en las que vamos avanzando con nuestro proyecto. De esta manera podemos volver a cualquier punto del desarrollo del proyecto y ver cómo estaban mis ficheros en esos momentos.

El hecho de ir creando distintas carpetas para nuestro proyecto, si son pocas carpetas, puede resultar fácil. Pero si nuestro proyecto se compone de muchos ficheros o si tenemos que realizar muchas copias de seguridad de las distintas versiones de nuestro proyecto, esto puede resultar bastante tedioso y complicado de gestionar.

Para ello, existe una alternativa, que es el programa de Gestión de Versiones llamado “**git**”. Este programa, que se instala en local en nuestro PC, nos permite ir realizando copias de seguridad y gestión de las distintas versiones de nuestro proyecto de una forma relativamente fácil. Digo “relativamente” porque es necesario aprenderse una serie de comandos, y esto al principio no resulta fácil, aunque con el uso constante y la práctica se va asimilando mejor y se va entendiendo el funcionamiento de dicho programa “git”.

## 2 - INSTALACIÓN DE GIT

Cómo acabo de comentar, “git” es un programa de Gestión de Versiones, que se instala en nuestro PC. Si tenemos varios PC, tendremos que instalarlo en cada uno de ellos, o por lo menos en aquellos que queramos llevar una Gestión de Versiones de nuestros proyectos.

Instalación de Git:

1. Descarga el instalador:
  - Windows: Visita el sitio web oficial de Git (<https://git-scm.com/downloads>) y descarga el instalador correspondiente a tu versión de Windows.
  - macOS: Si utilizas Homebrew, puedes instalarlo con el comando “brew install git”. Si prefieres el instalador, también está disponible en el sitio web de Git.
  - Linux: La mayoría de las distribuciones de Linux incluyen Git en sus repositorios. Puedes instalarlo usando el gestor de paquetes de tu distribución (por ejemplo, `sudo apt install git` en Ubuntu/Debian, `sudo yum install git` en Fedora/CentOS).
2. Ejecuta el instalador:
  - Windows: Sigue las instrucciones del asistente de instalación. Por lo general, se recomienda aceptar las opciones predeterminadas.
  - macOS: Si utilizaste Homebrew, la instalación se completará automáticamente. Si usaste el instalador, sigue las instrucciones.
  - Linux: El gestor de paquetes se encargará de la instalación.
3. Verifica la instalación:
  - Ejecuta el siguiente comando desde un terminal: `git --version`
  - Si la instalación ha ido bien, nos mostrará la versión de git instalada.

### 3 - CONFIGURACIÓN DE GIT

Una vez instalado “git” hay que decirle “nuestro nombre” y “nuestro email”. Evidentemente no es obligatorio que esos datos sean verdaderos, pero es bastante recomendable. Para ello tenemos que ejecutar los 2 comandos siguientes desde un terminal:

- `git config --global user.name "Tu Nombre Completo"`
- `git config --global user.email "tu_correo@ejemplo.com"`

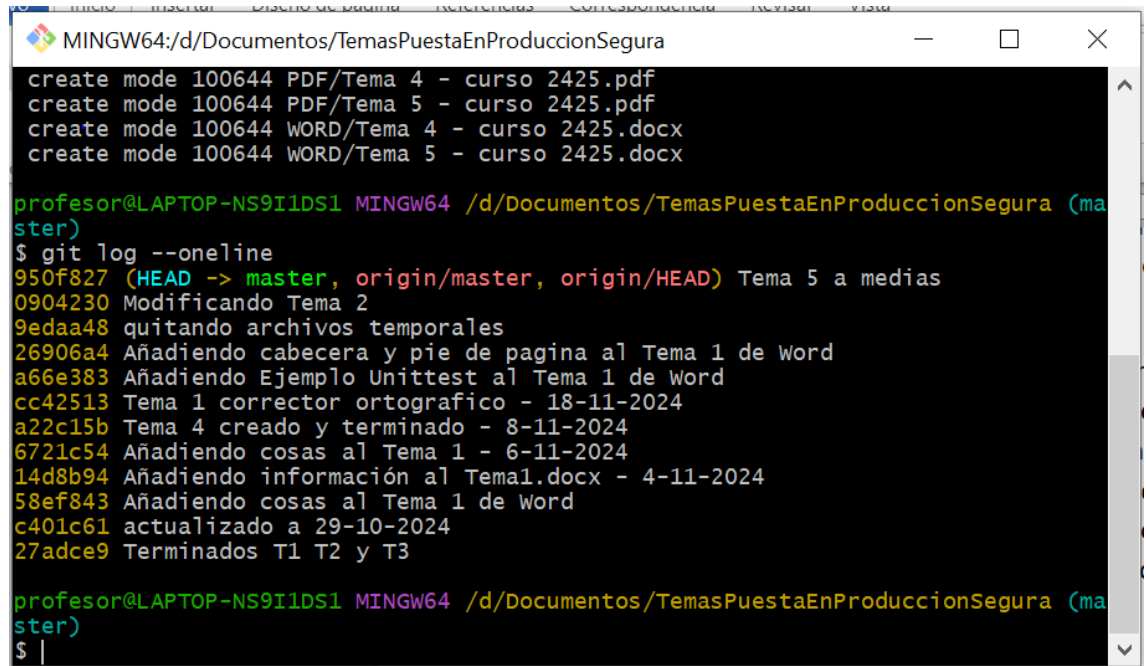
Esto tendremos que hacerlo en cada uno de los PC donde instalemos git, evidentemente

En cuanto a la configuración de “git” esto es todo.

### 4 - USO DE GIT

Una vez instalado y configurado “git”, ya podremos usarlo. Pero ¿cómo se usa?

Lo primero es crearse una carpeta en nuestro equipo, en la cual vamos a ir metiendo los ficheros de nuestro proyecto. O podemos usar una carpeta que ya contenga ficheros.

A screenshot of a terminal window titled 'MINGW64:/d/Documentos/TemasPuestaEnProduccionSegura'. The terminal shows the output of a 'git log --oneline' command. The output lists several commits with their hashes and descriptions. The commits are: '950f827 (HEAD -> master, origin/master, origin/HEAD) Tema 5 a medias', '0904230 Modificando Tema 2', '9edaa48 quitando archivos temporales', '26906a4 Añadiendo cabecera y pie de pagina al Tema 1 de Word', 'a66e383 Añadiendo Ejemplo Unittest al Tema 1 de Word', 'cc42513 Tema 1 corrector ortografico - 18-11-2024', 'a22c15b Tema 4 creado y terminado - 8-11-2024', '6721c54 Añadiendo cosas al Tema 1 - 6-11-2024', '14d8b94 Añadiendo información al Tema1.docx - 4-11-2024', '58ef843 Añadiendo cosas al Tema 1 de Word', 'c401c61 actualizado a 29-10-2024', and '27adce9 Terminados T1 T2 y T3'. The prompt '\$' is visible at the bottom of the terminal.

```
create mode 100644 PDF/Tema 4 - curso 2425.pdf
create mode 100644 PDF/Tema 5 - curso 2425.pdf
create mode 100644 WORD/Tema 4 - curso 2425.docx
create mode 100644 WORD/Tema 5 - curso 2425.docx

profesor@LAPTOP-NS9I1DS1 MINGW64 /d/Documentos/TemasPuestaEnProduccionSegura (ma
ster)
$ git log --oneline
950f827 (HEAD -> master, origin/master, origin/HEAD) Tema 5 a medias
0904230 Modificando Tema 2
9edaa48 quitando archivos temporales
26906a4 Añadiendo cabecera y pie de pagina al Tema 1 de Word
a66e383 Añadiendo Ejemplo Unittest al Tema 1 de Word
cc42513 Tema 1 corrector ortografico - 18-11-2024
a22c15b Tema 4 creado y terminado - 8-11-2024
6721c54 Añadiendo cosas al Tema 1 - 6-11-2024
14d8b94 Añadiendo información al Tema1.docx - 4-11-2024
58ef843 Añadiendo cosas al Tema 1 de Word
c401c61 actualizado a 29-10-2024
27adce9 Terminados T1 T2 y T3

profesor@LAPTOP-NS9I1DS1 MINGW64 /d/Documentos/TemasPuestaEnProduccionSegura (ma
ster)
$ |
```

Una vez que estemos ubicados en esa carpeta, hacemos clic derecho en la ventana de la carpeta, y elegimos la opción “**Open Git Bash Here**”. Esto abrirá un terminal desde el que podremos introducir las ordenes de git. Otra alternativa es abrirnos un terminal de la forma habitual en el Sistema Operativo, y ubicarnos en la carpeta de nuestro proyecto con los comandos correspondientes (por ejemplo el comando “cd”), pero es mucho más cómoda la primera opción.

## 5 - INICIALIZAR NUESTRA CARPETA CON GIT

Una vez que estamos ubicados en esa carpeta desde el terminal, tenemos que decirle a “git” que comience a Gestionar las Versiones en esa carpeta. Para ello ejecutamos la siguiente orden desde el terminal:

- git init

Esto creará en dicha carpeta, una **carpeta oculta llamada “.git”** que es donde “git” va a ir llevando los datos y la información necesaria para Gestionar las Versiones de todo lo que se vaya almacenando en la carpeta de nuestro proyecto.

Este comando **solo hay que ejecutarlo una sola vez**, por cada proyecto que queramos que gestione “git”, puesto que su finalidad es crear dicha carpeta oculta “.git”

Esto es todo lo que hay que hacer para inicializar una carpeta o proyecto en “git”

## 6 - DECIRLE A GIT QUÉ TIENE QUE IR SIGUIENDO

Una vez que le hemos dicho a git que vaya gestionando las versiones en una carpeta determinada, le tenemos que decir qué es lo que tiene que ir controlando dentro de la misma carpeta. Para ello tenemos 2 opciones:

- **“git add .”** : Con este simple comando le indicamos a git que vaya vigilando todos los cambios que se producen en la carpeta actual y en todas sus subcarpetas. Es lo más recomendable
- **“git add fichero1 fichero2 fichero3 carpeta1 carpeta2”**: Pero si solamente queremos ir vigilando los cambios que se van realizando en unos cuantos ficheros y/o carpetas, tenemos que indicárselo de ésta manera.

Este comando se tiene que ejecutar antes de realizar un “commit”, que lo veremos más adelante. A lo largo del desarrollo de un proyecto, se realizarán múltiples commits, y antes de realizarlos hay que realizar su correspondiente “git add”

Cuando hemos ejecutado el comando “git add”, le estamos diciendo a “git” que esos archivos los pase al **“STAGE AREA” o “AREA DE SEGUIMIENTO”**, la cual es una carpeta oculta dentro de la carpeta “.git”. Solamente se vigilarán los cambios de los archivos que hayamos añadido al “Área De Seguimiento”.

Se pueden usar tantos “git add” como queramos antes de realizar un commit

Hasta aquí “git add”

## 7 - COMPROBAR EL ESTADO DEL “STAGE AREA”

Si queremos ver qué es lo que git está siguiendo en cada momento en nuestra carpeta del proyecto tenemos que ejecutar:

- `git status -s`

Esto mostrará el contenido de la carpeta actual y el estado de cada uno de los ficheros o carpetas vigiladas, con las siguientes opciones entre otras:

- **??**: si pone 2 caracteres de interrogación al principio del fichero significa que no esta siendo vigilado por git
- **M**: si pone una letra M, significa que está siendo controlado, pero que se ha modificado
- **A**: si pone una letra A, significa que está siendo controlado por “git”

## 8 - GUARDAR VERSIONES DE MI PROYECTO

El comando “commit” en “git” sirve para ir guardando “instantáneas” de cómo se encuentran los archivos “vigilados” en determinados momentos.

Cuando alguien dice que va a realizar un “commit”, es que va a hacer una copia de seguridad de sus archivos en ese preciso momento, estén como estén dichos archivos.

Para ejecutar un “commit” debemos indicarle una descripción breve de esa “instantánea” de nuestros archivos. Y se hace así:

- **git commit -m “Primer commit de mi proyecto”**

La opción “-m” sirve para añadir una descripción o nombre a nuestro commit.

Una vez hecho esto, se habrá guardado una copia de todos los archivos “vigilados” (los que le hemos indicado con “git add”), en ese mismo momento.

Después de hacer el commit, tenemos que ejecutar otra vez el “git add” correspondiente para seguir o vigilar los cambios en mis archivos. Si queremos que se monitorice o vigile todo lo del directorio actual, tenemos que ejecutar de nuevo “git add .”

Si realizamos un nuevo “commit” sin haber hecho un “git add” antes, no nos guardará nada en esa instantánea.

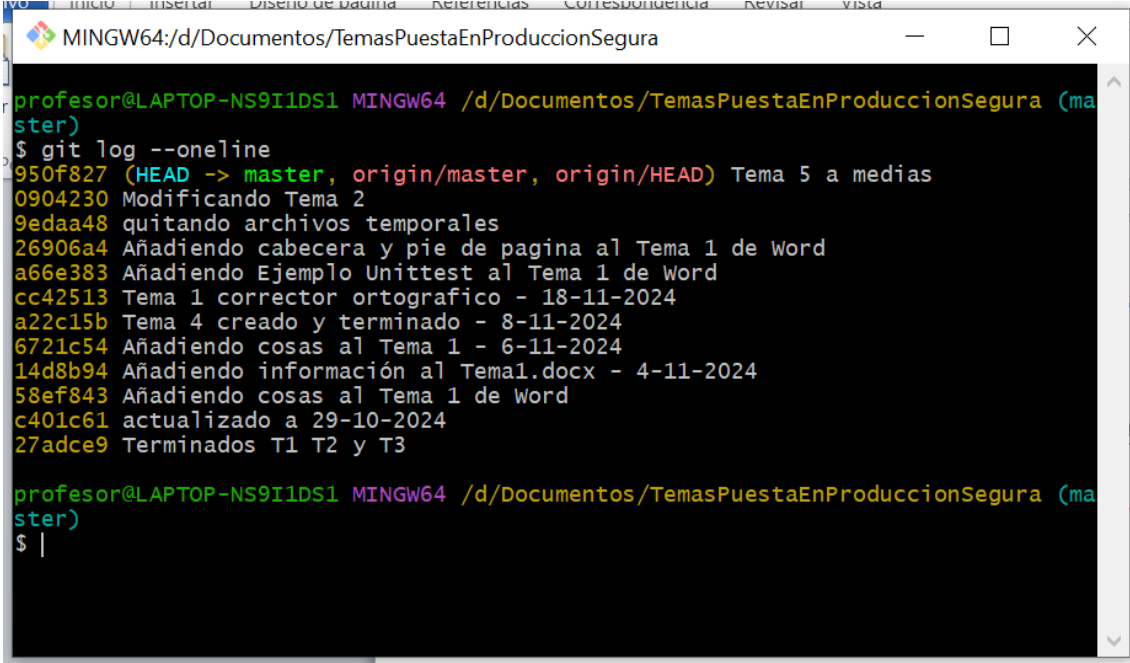
Si después de hacer un “commit” y un “git add”, ejecutamos “git status -s” no aparecerá nada, hasta que modifiquemos algún fichero de los que se están siguiendo. Al modificar o añadir algún fichero, aparecerá una “M” al principio de aquellos archivos modificados cuando hagamos un “git status -s”.

En cada “commit” solo se guardarán aquellos archivos vigilados que han sido modificados. Los no modificados no se guardan en esa “versión” o “instantánea” o “copia de seguridad”.

## 9 - VER TODOS LOS COMMITS QUE LLEVAMOS HECHOS

A medida que vamos realizando “commits” en nuestro proyecto, podemos ver un listado de esos “commits”. Esto lo hacemos con el siguiente comando:

- **git log --oneline**



```
profesor@LAPTOP-NS9I1DS1 MINGW64 /d/Documentos/TemasPuestaEnProduccionSegura (master)
$ git log --oneline
950f827 (HEAD -> master, origin/master, origin/HEAD) Tema 5 a medias
0904230 Modificando Tema 2
9edaa48 quitando archivos temporales
26906a4 Añadiendo cabecera y pie de pagina al Tema 1 de Word
a66e383 Añadiendo Ejemplo Unittest al Tema 1 de Word
cc42513 Tema 1 corrector ortografico - 18-11-2024
a22c15b Tema 4 creado y terminado - 8-11-2024
6721c54 Añadiendo cosas al Tema 1 - 6-11-2024
14d8b94 Añadiendo información al Tema1.docx - 4-11-2024
58ef843 Añadiendo cosas al Tema 1 de Word
c401c61 actualizado a 29-10-2024
27adce9 Terminados T1 T2 y T3

profesor@LAPTOP-NS9I1DS1 MINGW64 /d/Documentos/TemasPuestaEnProduccionSegura (master)
$ |
```

En ese listado podremos ver cada uno de los “commits” realizados (vemos su código de identificación al principio de la línea en amarillo, y en blanco vemos el nombre de cada “commit”). Pero solo nos muestra los commits anteriores al actual en el que estemos situados.

Si queremos ver todos los commits, incluso los posteriores al actual, ejecutamos lo siguiente:

- `git log --all`

## 10 - VOLVER A UN COMMIT ANTERIOR

Si hemos realizado varios “commits” y queremos volver a ver cómo estaba nuestro proyecto anteriormente en algún otro “commit”, podemos volver a éste, ejecutando lo siguiente:

- `git checkout fe3ca73`

El código indicado es el del commit al que queremos volver. Importante: Esto no borra los commits posteriores.

Pero si ahora queremos volver a otro commit posterior a éste, no lo podremos ver con “git log --oneline”, tenemos que ejecutar “git log --all”, copiamos el ID del commit al que queremos avanzar y volvermos a ejecutar un “git checkout”

## 11 - DEVOLVER NUESTRO PROYECTO A UN ESTADO ANTERIOR

Si hemos avanzado con nuestro proyecto, pero de repente nos damos cuenta que queremos volver a un estado anterior de nuestro proyecto, le tenemos que indicar a qué “commit” queremos remontarnos, **teniendo muy en cuenta que los commits posteriores a ese se perderán así como todas sus instantáneas de los archivos modificados.**

Así que esto hay que hacerlo con mucho cuidado y estando totalmente seguros de lo que estamos haciendo.

Habría que ejecutar lo siguiente:

- `git reset --hard 7eb94fc`

Insisto en que todos los commits existentes, posteriores a éste, desaparecerán junto con sus instantáneas.

## 12 - HACER UN “git add” Y UN “git commit” A LA VEZ

Hemos comentado antes que para hacer un “commit”, primero hay que hacer un “git add”, lo cual conlleva ejecutar 2 comandos desde el terminal.

Pero esto se puede hacer de la siguiente manera en un solo comando:

- `git commit -am “Segundo commit de mi proyecto”`

Como se puede observar, la opción “-a” me sirve para realizar un “add” automáticamente sobre toda la carpeta de mi proyecto.

## 13 - GITHUB

“GitHub” es un sitio web pensado para almacenar nuestros proyectos, con la posibilidad de compartirlos con otras personas (repositorio público), o a nivel personal (repositorio privado).

Registrarse en “GitHub” es totalmente gratuito. Una vez registrado te ofrecen un almacenamiento limitado para guardar tus ficheros (unos 500 MB), así como un tráfico de red limitado por cada mes, con lo que no podemos estar constantemente subiendo y bajando archivos, porque nos comemos ese tráfico de red.



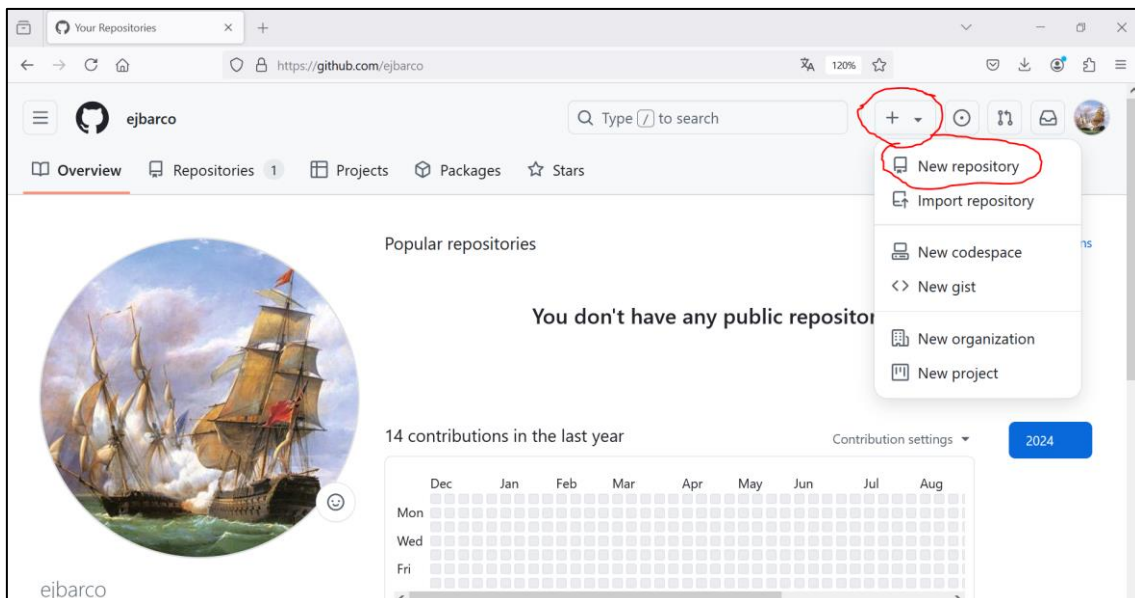
Pero una cosa muy importante: **“git” no es lo mismo que “github”**, aunque sus nombres sean similares.

Una vez que nos hayamos creado la cuenta en “GitHub”, es muy importante **NO OLVIDAR EL NOMBRE DE USUARIO Y LA CONTRASEÑA**. Así que tendremos que ponerlo a buen recaudo para no perder el acceso a nuestra cuenta.

## 14 - CREAR REPOSITORIOS EN GITHUB

Recién creada nuestra cuenta de “GitHub”, evidentemente estará vacía, sin “proyectos” o “repositorios”.

Para crear un nuevo “Repositorio” lo podemos hacer de varias maneras, pero la más sencilla es: botón “+” situado en la parte superior derecha, y clic en “New repository”



Aparecerá lo siguiente:

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Required fields are marked with an asterisk (\*).

Owner \* Repository name \*

ejbarco

Great repository names are short and memorable. Need inspiration? How about [scaling-journey](#) ?

Description (optional)

☒ Public  
Anyone on the internet can see this repository. You choose who can commit.

☐ Private  
You choose who can see and commit to this repository.

Initialize this repository with:

☒ Add a README file  
This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore  
.gitignore template: None

Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

Choose a license

Le damos un nombre a nuestro nuevo repositorio (que no sea muy largo, y si puede ser sin espacios mejor), le damos una Descripción informando de qué trata nuestro repositorio, le decimos si será Público o Privado (para que solamente nosotros podamos acceder a su contenido), y alguna cosa más, y por último pulsar el botón verde “Create Repository”.

Add .gitignore  
.gitignore template: None

Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

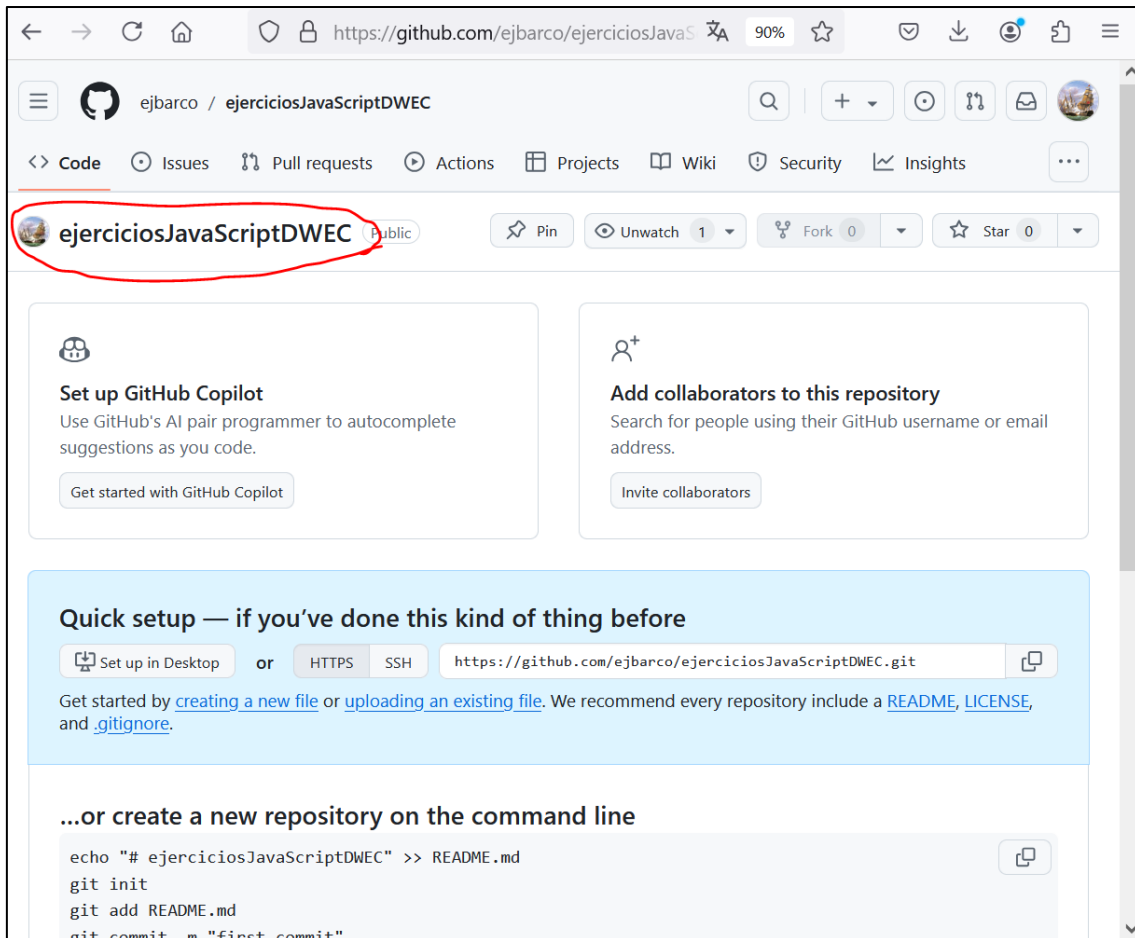
Choose a license  
License: None

A license tells others what they can and can't do with your code. [Learn more about licenses.](#)

*i* You are creating a public repository in your personal account.

Create repository

Y ya tendremos nuestro nuevo repositorio creado:



Si queremos añadir más repositorios lo haremos de la misma manera, teniendo en cuenta que no puede haber nombres repetidos. Y se supone que podemos hacer infinitos repositorios en GitHub.

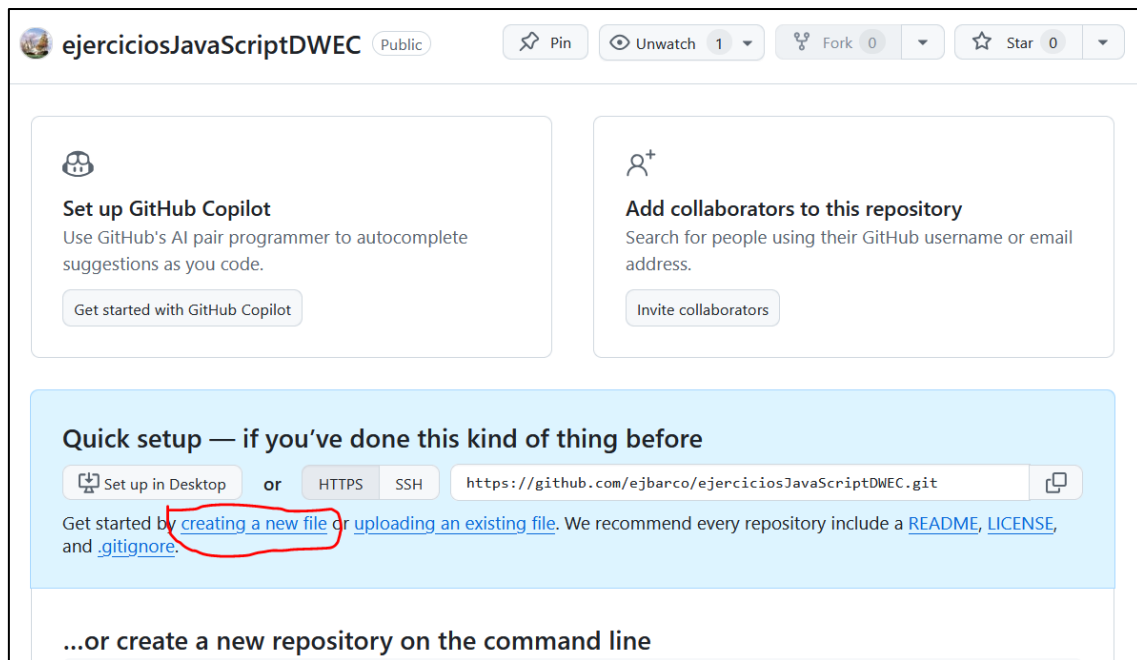
## 15 - AÑADIR ARCHIVOS A NUESTRO REPOSITORIO

Cuando creamos un Repositorio Nuevo, este evidentemente estará vacío. Lo interesante es ir añadiéndole archivos y carpetas. ¿Y cómo se hace eso?

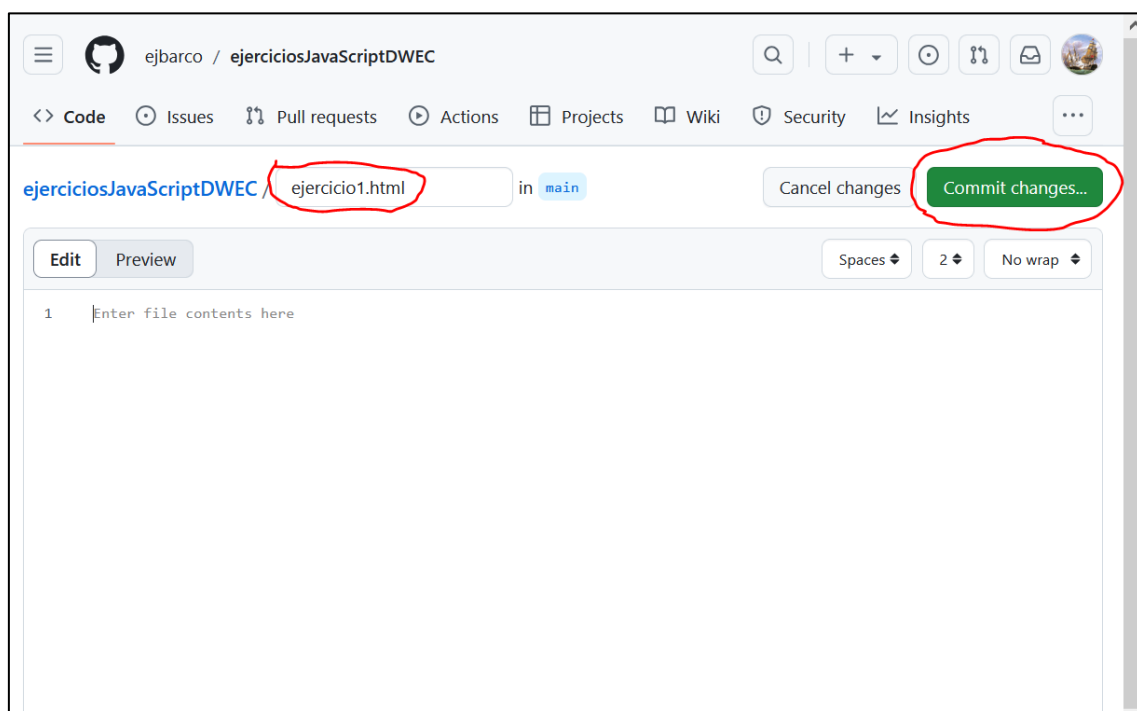
Se puede hacer de varias maneras:

- Directamente desde la web de “GitHub”
- Desde nuestro PC, con “git” instalado

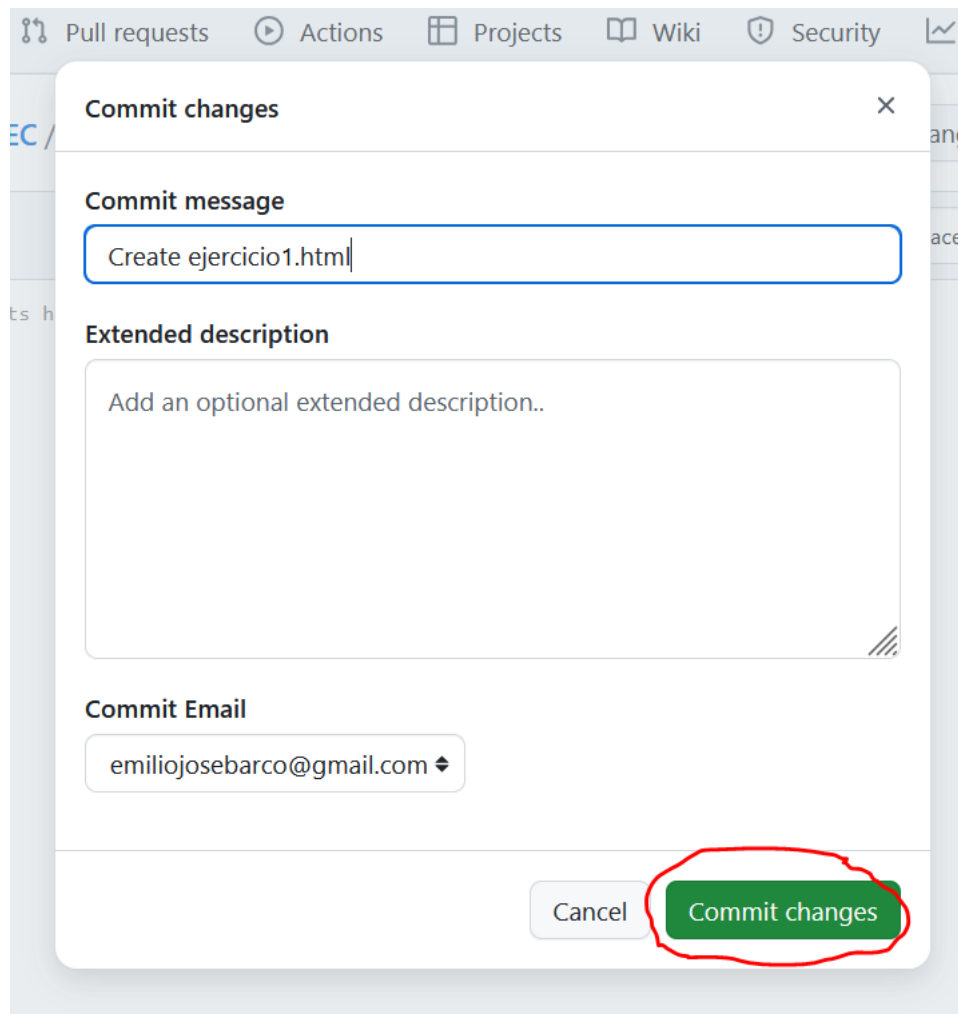
Para **crear archivos directamente desde GitHub**, pulsamos en “creating a new file”:



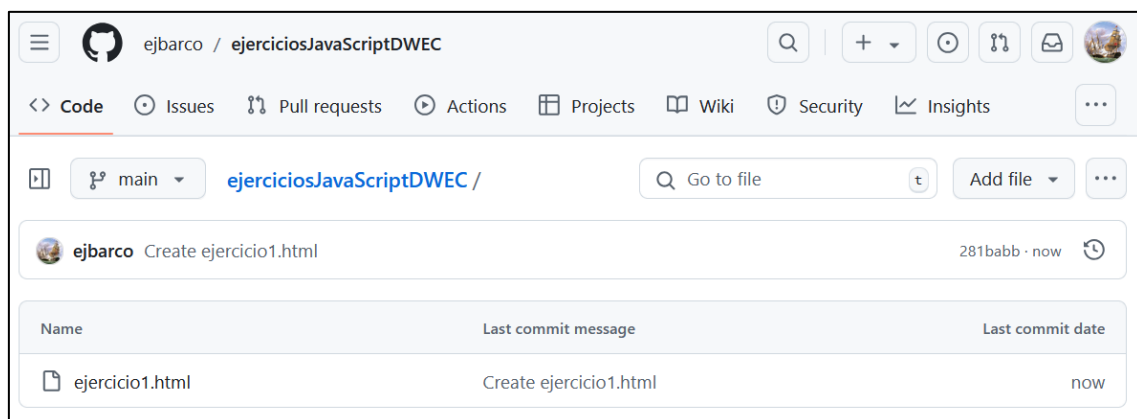
aparece un editor “online” en el que podemos poner el nombre del nuevo archivo y agregar contenido:



para guardar ese archivo pulsaríamos el botón “Commit changes...” lo cual nos crea un nuevo “commit” en “GitHub”. Ese commit nos pedirá un nombre y una descripción antes de realizarlo:



pulsar “Commit changes”, y ya estaría creado nuestro primer archivo del repositorio, de forma online:



Pero, bajo mi punto de vista, es mejor y más eficiente crear esos archivos en local, en tu PC, y luego subirlos a tu repositorio de GitHub.

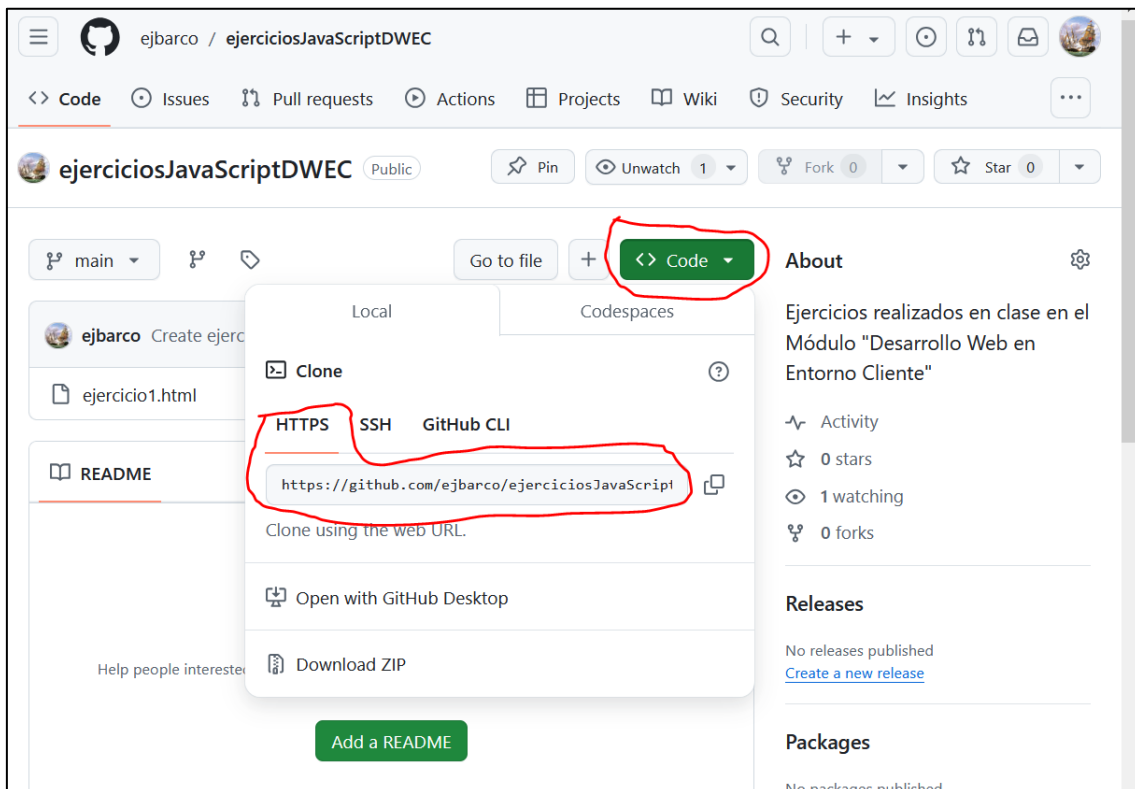
## 16 - SUBIR ARCHIVOS DESDE NUESTRO PC CON GIT A GITHUB:

Ahora volvemos a nuestro PC, en el que ya tenemos instalado y configurado “git”.

Vamos a ejecutar el siguiente comando para “clonar” (hacer una copia exacta de los archivos y carpetas) nuestro repositorio de GitHub a nuestro PC:

- `git clone https://github.com/ejbarco/ejerciciosJavaScriptDWECE.git`

la URL <https://github.com/.....> la copio de la página de GitHub en el botón “Code” -> “HTTPS”:



Esto habrá creado en el directorio actual en el que estuviéramos ubicados (por ejemplo en la carpeta de “Documentos”), una nueva carpeta llamada “ejerciciosJavaScriptDWECE”, con una carpeta oculta llamada “.git” y los demás archivos y/o carpetas que hubieran en el repositorio remoto.

Ahora que tenemos “clonado” el repositorio de GitHub en nuestro PC, podemos ir metiendo archivos en él (archivos nuevos o archivos que ya los tuviéramos creados previamente).

Una vez que hayamos creado los archivos nuevos en esa carpeta, tenemos que hacer un “git add” y un “git commit” en local, y si queremos subir esos cambios a Github tenemos que ejecutar lo siguiente:

- `git push -u origin master`

Esto sincronizará y subirá automáticamente todos esos ficheros y carpetas a nuestro repositorio de GitHub.

Otra alternativa, pero solo para subir archivos nuevos es:

- `git remote add origin https://github.com/ejba...ptDWECEC.git`

Si vamos ahora a GitHub, a nuestro repositorio podremos ver que se han subido los archivos y carpetas que habíamos creado en nuestro PC.

Bajo mi punto de vista ésta es la mejor forma para añadir ficheros a nuestros repositorios de GitHub.

**IMPORTANTE:** la primera vez que subimos a GitHub desde nuestro PC, nos pide que nos autentiquemos en esa plataforma, con nuestro username y nuestra contraseña.

## 17 - CLONAR REPOSITORIOS DE GITHUB EN NUESTRO PC

En el apartado anterior ya hemos visto como clonar un repositorio nuestro a nuestro PC.

Pues igual se pueden clonar Repositorios que sean Públicos de otros usuarios de GitHub, con el comando:

- `git clone https://github.com/... .git`

Teniendo en cuenta que se va a crear una carpeta nueva en el directorio actual en el que estemos situados en el terminal al ejecutar la orden anterior. Se copiará todo lo que haya en dicho repositorio.

La URL de los repositorios se puede obtener igual que hemos visto en el apartado anterior.

## 18 - ACTUALIZAR UN PROYECTO DESDE GITHUB A NUESTRO PC

Si en GITHUB tenemos el proyecto más avanzado y con nuevos commits, que en el PC local no tenemos, podemos descargar esa información, para actualizar en local el proyecto. Para ello, ubicados en el directorio del proyecto, ejecutamos:

- `git pull`

## 19 - ESTABLECER VERSIONES DE MI PROYECTO

Aunque anteriormente dijimos que para hacer versiones era suficiente con hacer un “commit”, realmente para crear una versión se hace de esta manera:

- `git tag 25-10-2024v1 -m “Versión 1 del proyecto”`

Después de “**tag**” ponemos el nombre de la versión, y después de “**-m**” ponemos la descripción de dicha versión.

Una vez creado el nuevo “tag” en local, si queremos subirlo a GITHUB:

- `git push --tags`

De esta manera, en Github, si vamos a la sección “tags” me los mostrará y me dejará descargar el proyecto tanto en “zip” como en “tar.gz”.

## 20 - RAMAS EN GIT

Una rama es una bifurcación de nuestro proyecto a partir de un punto o “commit” específico. Lo podemos usar para que otra u otras personas puedan ayudarnos con nuestro proyecto. No es lo mismo que un FORK.

Por defecto, cuando creamos un proyecto con git, la primera rama se llama “**master**”, pero podemos crear más.

## 21 - CREAR UNA RAMA NUEVA

Para crear una rama nueva desde un “commit” determinado, primero hay que ubicarse en dicho “commit” con “git checkout” y después ejecutar:

- `git branch Rama2`

Podemos poner el nombre que queramos a esa nueva rama, en el ejemplo anterior ha sido “Rama2”.

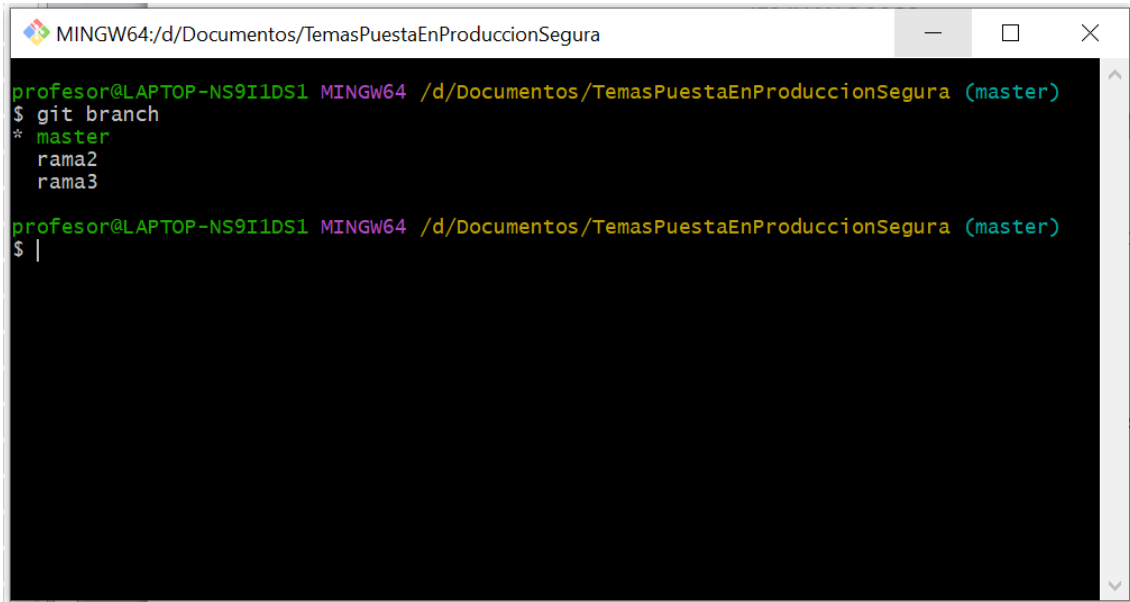
Ahora, cuando ejecutemos un “`git log --oneline`” podremos ver en color verde las ramas que tenemos creadas.



## 22 - VER LAS RAMAS EXISTENTES EN EL PROYECTO

Aunque antes hemos dicho que con un “--oneline” se pueden ver las ramas existentes, hay otro comando con el que podemos ver esas ramas:

- `git branch`



```
MINGW64:/d/Documentos/TemasPuestaEnProduccionSegura
profesor@LAPTOP-NS9I1DS1 MINGW64 /d/Documentos/TemasPuestaEnProduccionSegura (master)
$ git branch
* master
  rama2
  rama3

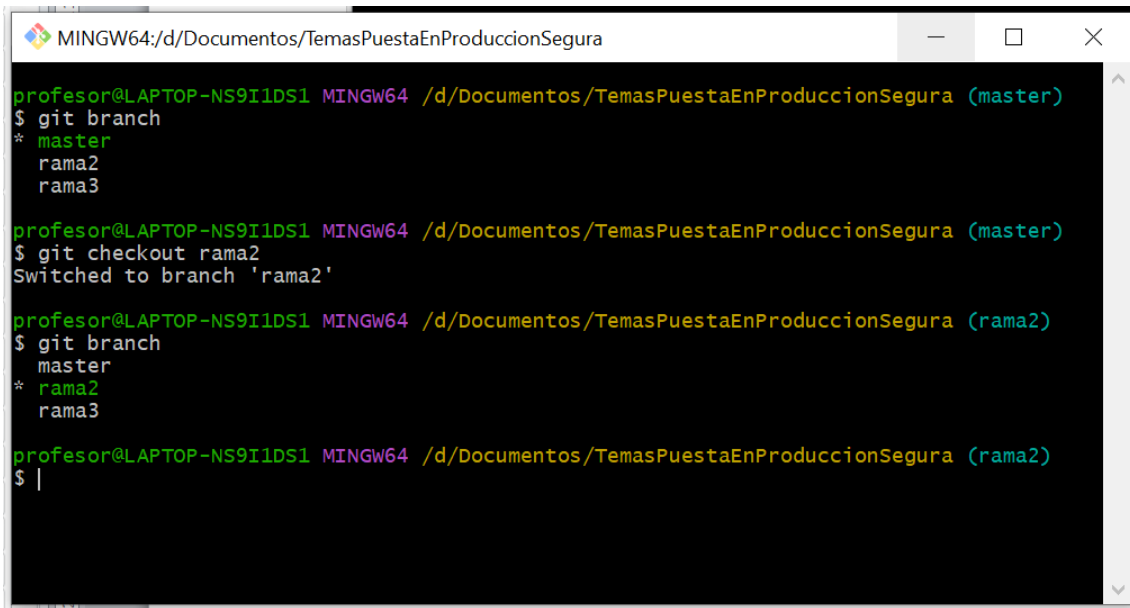
profesor@LAPTOP-NS9I1DS1 MINGW64 /d/Documentos/TemasPuestaEnProduccionSegura (master)
$ |
```

la rama en la que estamos ubicados aparece con un asterisco al principio. En el ejemplo anterior estamos en la rama “**master**”

## 23 - MOVERNOS A OTRA RAMA DEL PROYECTO

Para cambiarnos a otra rama del proyecto ejecutamos:

- `git checkout Rama2`

A screenshot of a terminal window titled 'MINGW64:/d/Documentos/TemasPuestaEnProduccionSegura'. The terminal shows the following commands and output:

```
profesor@LAPTOP-NS9I1DS1 MINGW64 /d/Documentos/TemasPuestaEnProduccionSegura (master)
$ git branch
* master
  rama2
  rama3

profesor@LAPTOP-NS9I1DS1 MINGW64 /d/Documentos/TemasPuestaEnProduccionSegura (master)
$ git checkout rama2
Switched to branch 'rama2'

profesor@LAPTOP-NS9I1DS1 MINGW64 /d/Documentos/TemasPuestaEnProduccionSegura (rama2)
$ git branch
  master
* rama2
  rama3

profesor@LAPTOP-NS9I1DS1 MINGW64 /d/Documentos/TemasPuestaEnProduccionSegura (rama2)
$ |
```

## 24 - FUSIONAR 2 RAMAS EN 1 SOLA

Cuando hemos estado trabajando en paralelo con 2 ramas, llegará el momento (o no) en el que queramos unir o fusionar 2 ramas en una sola. Normalmente se fusiona la rama “master” con otra que hayamos creado.

Los pasos a seguir son los siguientes:

1. Situarnos en la rama principal (master): **git checkout master**
2. Ejecutar el “merge”, que nos fusionará las 2 ramas: **git merge Rama2**

Esto puede generar **conflictos** en los archivos fusionados de ambas ramas, ya que habremos modificado algunos archivos en la rama “master” y los mismos archivos en la otra rama.

Para resolver esos conflictos, cuando abrimos el Editor de Código (por ejemplo Visual Studio Code) podremos ver qué líneas pertenecen a la rama master y cuáles a la otra rama, y tenemos 3 posibilidades:

1. Dejar las 2 modificaciones en el archivo.
2. Elegir solo las modificaciones de la rama master
3. Elegir solo las modificaciones de la otra rama fusionada

Esto lo tendremos que hacer con cada uno de los archivos que tengan conflictos en ambas ramas fusionadas, lo que conlleva tiempo y análisis de lo que se va a dejar en la versión definitiva.

## 25 - BORRAR UNA RAMA

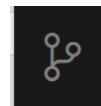
Cuando ya no necesitemos una rama porque la hemos fusionado con otra ejecutamos:

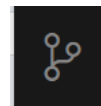
- `git branch -d rama2`

## 26 - GIT Y GITHUB EN VISUAL STUDIO CODE

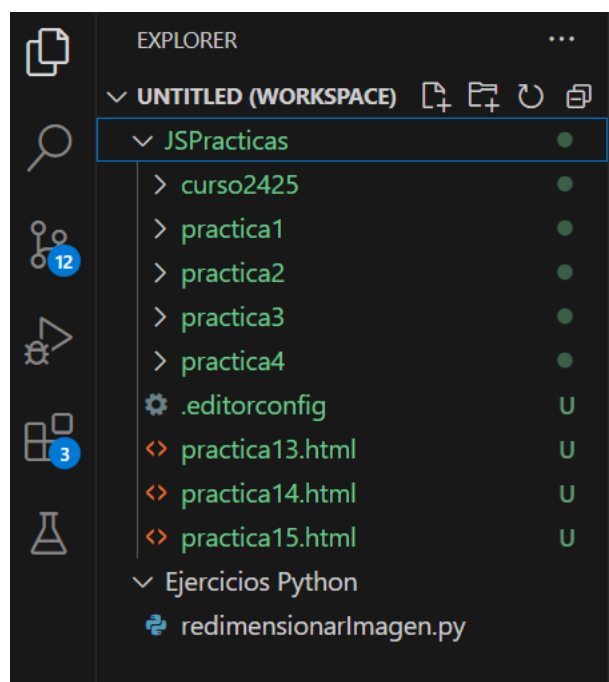
En VSCode, en un terminal podemos ejecutar “`git init`” y comenzará a seguir nuestro proyecto.

En el panel izquierdo, donde se ven las carpetas y archivos, se verán de color verde y aparece a su derecha una “U”, que significa “unfollowed”, es decir, que todavía no los hemos añadido al “Stage Area” porque no hemos hecho todavía un “`git add`”.



En la parte izquierda de Visual Studio Code, hay un icono con la forma . En él nos informa de cuántos cambios tenemos sin guardar en las ramas. Si hacemos clic en ese icono, nos dice cuáles son los archivos que no estamos siguiendo (unfollowed) o que no hemos guardado.

Cuando hagamos un “`git add`” ya desaparecerá la “U” de aquellos archivos, y se convertirá en una “A” para indicar que ya se están siguiendo los cambios en esos archivos.

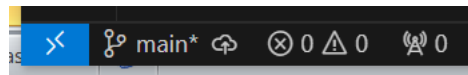


Cuando hagamos un “commit” desde el terminal de VSC, la “A” de esos archivos desaparecerá, así como el color verde de esos archivos.

Al modificar algún archivo y guardarlo con VSC, en el panel izquierdo aparecerá una “M” en ese fichero (modificaciones pendientes de guardar con git). Al hacer otro “commit” las “M” desaparecerán.

## 27 - RAMAS EN VSC


En VSC en la barra de estado, abajo a la izquierda, aparece la rama en la que estamos situados por ejemplo, “master”.

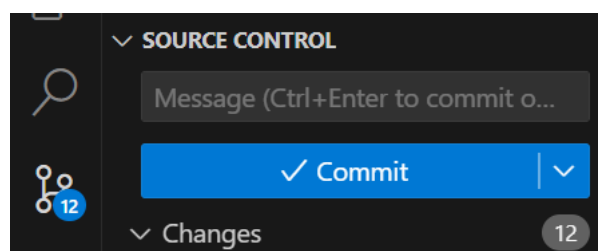


Para crear una nueva rama desde VSC, se puede hacer de varias formas:

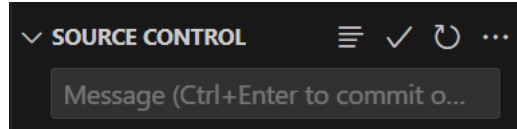
- La primera forma es: Desde la consola con “git branch rama2”. Para ver las ramas existentes “git branch”. Para movernos a una rama nueva “git checkout rama2”.
- La otra forma es hacer clic en la barra de estado donde pone el nombre de la rama actual. En la parte superior aparece un cuadro de texto y una lista desplegable. Escribir el nombre de la nueva rama, pero antes hay que pulsar en “+Create new branch”. Desde este menú también nos podemos cambiar a otra rama existente.

## 28 - COMMITS EN VSC

A parte de poder realizar un commit desde el terminal de VSC, también podemos hacerlo de una forma un poco más cómoda. Pulsando el botón de git , podremos ver un cuadro de texto que pone: `Message (Ctrl+Enter to commit on 'master')`. Si ponemos un texto en él, y hacemos “Ctrl+Enter”, estaremos haciendo un commit, en vez de hacerlo por la consola o terminal.



Justo encima del cuadro para hacer commits, aparecen unos botones:



: para hacer commits




: para hacer refresh



: tiene un menú con muchas herramientas, como “pull”, “push”, “clone”, “commit”, “branch”, etc...

## 29 - FUSIONAR RAMAS

Desde  → “branch” → “merge branch” se despliega arriba todas las ramas disponibles, y hacemos clic en la que queramos fusionar con la rama actual.

Si al hacer un “Merge” ha habido algún conflicto, lo veremos con colores en el área de trabajo de VSC. Justo encima de los conflictos salen 4 opciones:

- **Accept Current Change:** mantener los cambios actuales
- **Accept Incoming Change:** mantener los otros cambios
- **Accept Both changes:** mantener ambos cambios en el archivo
- **Compare changes:** se divide la ventana en vertical para comparar los cambios en paralelo. Una vez que hayas tomado una decisión, cerramos esa pestaña con la ventana partida y podremos decidir qué hacer.

Si en la barra de estado, al lado del nombre de la rama actual aparece un signo de exclamación “!” significa que se ha detectado un conflicto que no se ha solucionado todavía.


## 30 - CLONAR REPOSITORIO DE GITHUB DESDE VSC

Si tenemos un repositorio en Github y queremos traerlo o clonarlo desde VSC:

View → Command Palette → Escribir “Git Clone” y “Enter”. Nos pide la URL del repositorio. Se la escribimos, “Enter”, y ya habrá importado ese repositorio a la carpeta que le digamos.

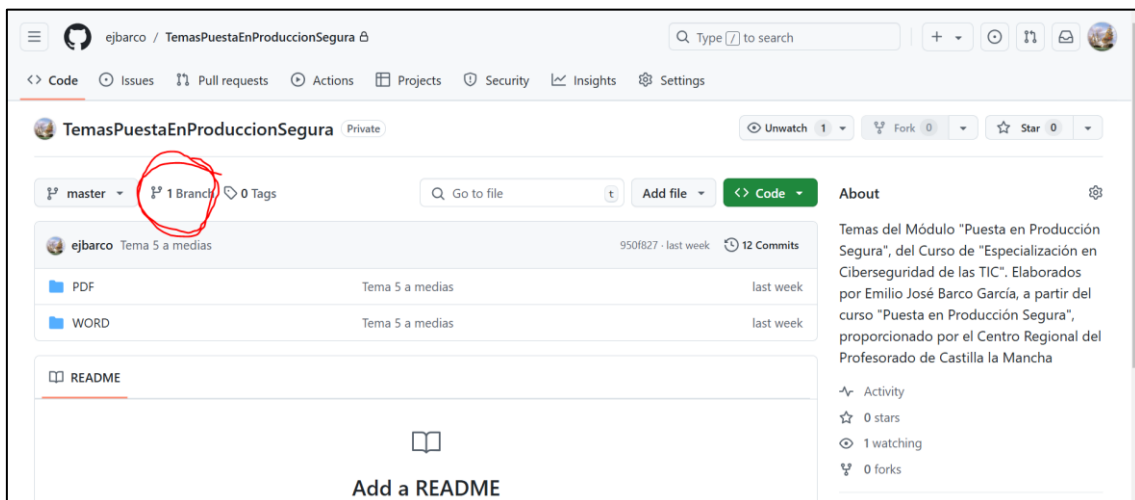
## 31 - SUBIR LOS CAMBIOS A GITHUB

Si hacemos cambios en local de ese repositorio y queremos subirlos a GitHub, primero hacemos un commit, y luego: View → Push. Esto subirá todos los cambios a GitHub.

Si hacemos algún commit desde GitHub, y vamos a VSC, en ese repositorio, en la barra de estado aparecerá un icono  para sincronizar los cambios. Si añadimos cosas nuevas en local y pulsamos ese botón, subirá a GitHub esos cambios automáticamente (como un “push”).

## 32 - TRABAJAR CON RAMAS ONLINE EN GITHUB

En GitHub, en el repositorio que estemos en la parte superior, al lado del nombre de la rama, nos pondrá cuantas ramas hay en el repositorio

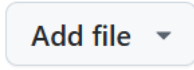


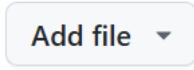
Al hacer clic en ese botón, veremos las ramas que hay y cuando se actualizaron.

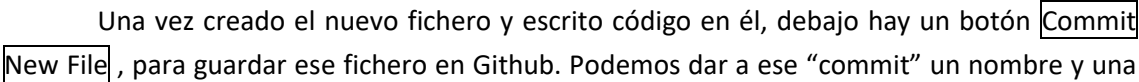
En la pantalla principal del proyecto o repositorio, arriba a la izquierda está el botón del nombre de la rama actual. Si hacemos clic en él se despliega la lista de ramas.

Y hay un cuadro de búsqueda por nombre. Este cuadro también me sirve para crear nuevas ramas. Al crear una nueva rama, automáticamente nos mueve a ella (checkout).

## 33 - CREAR NUEVOS FICHEROS DESDE GITHUB Y GUARDARLOS



En GitHub hay un botón llamado  desde el que podemos crear nuevos ficheros directamente en Web. Desde ahí podemos “crear nuevo archivo” o “subir archivos”.

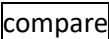
Una vez creado el nuevo fichero y escrito código en él, debajo hay un botón , para guardar ese fichero en Github. Podemos dar a ese “commit” un nombre y una descripción antes de guardarlo.

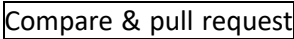
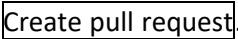
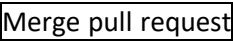


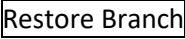
Cada vez que editamos un archivo directamente en Github, si lo queremos guardar, tenemos que hacer el “commit” como en el párrafo anterior.


## 34 - COMPARAR RAMAS DESDE GITHUB

Estando en GitHub, en una rama diferente a la “master”, se nos avisará de que esta rama tiene 3 o 4 o 5 ... commits más que la rama “master” con un mensaje como este:



Hay un botón  para comparar los cambios o diferencias entre la rama actual y la “master”. Las líneas diferentes nos las muestra con un color azul de fondo.

En las ramas que no son la master hay un botón llamado  que sirve para pedir permiso al administrador del proyecto para hacer un merge de esa rama con la master. Antes del “merge” escribimos un comentario de porqué queremos hacer el merge, luego . Al pulsarlo, Github analiza las 2 ramas, y nos dice si hay conflictos o no. Por último hay un botón  para hacer el merge con . La rama desaparecerá si no la necesitamos, con el botón  y si lo pulsamos se convierte en  para recuperar la rama borrada.

Cuando hayamos hecho todos esos cambios, podemos ir a VSC, en local, pulsar el botón  en la barra de estado, y se sincronizará con el repositorio GitHub, aunque me da la advertencia de que eso hará cambios en local con un “push and pull”.

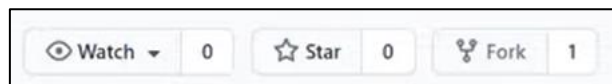
## 35 - FORKS EN GITHUB

Los FORK son ramificaciones de un proyecto que se utilizan para colaborar. No es lo mismo que una rama o branch. Es como hacer una clonación de un proyecto, pero con ventajas. El fork se crea en GITHUB.

Cuando un programador externo que está colaborando con su Fork, hace algún commit, puede mandar un “Pull Request” al dueño del repositorio original. Eso es una sugerencia para que el dueño vea lo nuevo que se ha hecho en el proyecto y pueda aceptar los cambios si lo ve bien.

Si el dueño del proyecto no ve bien lo que se ha hecho en el “Pull Request” lo puede rechazar, pero si le gusta, puede aceptar los cambios y hacer un MERGE con el proyecto del colaborador.


Si queremos hacer un fork de algún repositorio de Github que no sea nuestro, en la parte derecha de la página de ese repositorio en Github, aparecen 3 botones:



Al pulsar el botón “Fork” se realiza automáticamente un fork de ese repositorio a nuestra cuenta de GitHub.

Si vamos ahora a nuestra cuenta de GitHub ahí tendremos el “fork” hecho, y nos dice desde donde le hemos hecho fork.

Si trabajamos en el fork, hacemos cambios en él y añadimos cosas nuevas, y luego hacemos commit y lo subimos a nuestra cuenta de GH, desde ahí podemos solicitar un “Pull

Request” en el botón . Esto nos llevará a otra página en la que pulsamos un botón verde **New PullRequest**. Ahora pasa a otra página en la que me dice que está comparando cambios con el repositorio original, y si me dice “Able to merge” es que no hay muchos conflictos y puedo hacer el “PullRequest”. Me mostrará las modificaciones: Lo que había antes se va a poner en rojo y lo modificado por nosotros se pone en verde. Si queremos confirmar y enviar el PullRequest, le damos al botón verde “Create Pull Request”, se abre una nueva página en la que tenemos que poner una descripción breve de lo que se ha añadido o modificado, y debajo podemos dar una descripción más extendida de las modificaciones. Esto se convertirá en un “chat” entre el dueño del repositorio original y nosotros.

Cuando el dueño del Proyecto entre en GitHub, verá arriba una solicitud de PullRequest del colaborador. Al hacer clic en él, entraremos al chat con esa persona.

Podemos ver un botón **Files Changed (1)** que nos muestra aquellos archivos cambiados y cuántos “commits” se han hecho en ese fork.



Al entrar en [Files changed](#) puedo ver los cambios hechos y si le quiero contestar para comentar algo, le puedo dar al botón [Review changes](#) y le comento cualquier cosa que quiera. Ahí tengo 3 posibilidades (Hay que elegir una):

- Comment : contestar algo sin aprobar el PullRequest
- Approve : aprobar el PullRequest y Merge
- Request changes : pedirle cambios al colaborador

Al pulsar el botón [Submit Review](#) mandaremos el comentario, o haremos el merge o le mandaremos la petición de cambios lo último es pulsar [Merge pull Request](#) si hemos aprobado el PullRequest, y por último [Confirm Merge](#).

