

## Contenido

1	Introducción .....	4
1.1	Ventajas de GIT.....	4
2	Instalaciones recomendadas: .....	5
2.1	Extensiones de VSCode .....	5
2.2	Tema que estoy usando en VSCode .....	5
3	Configuración .....	5
3.1	Primeros comandos.....	6
3.2	Configurando .....	6
3.2.1	Establecer nombre de usuario .....	6
3.2.2	Configurar la dirección de correo del usuario .....	6
3.2.3	Activar colores de la interfaz .....	6
3.2.4	Ver las configuraciones .....	6
4	Creando el primer repositorio .....	6
5	¿Qué hace git por nosotros hasta el momento? .....	9
6	Cambiar el nombre de la rama Master a Main.....	10
6.1	Cambiar el nombre de una rama .....	11
6.2	Configurar que por defecto la rama principal de un nuevo proyecto se llame main.....	11
7	Archivo Readme.md y comando log .....	11
7.1	Ver los commits realizados.....	12
8	Adds y commits con visual studio code .....	13
9	Diferentes formas de agregar archivos al escenario. ....	15
9.1	Añadir archivos sueltos .....	15
9.2	Añadir archivos usando el comodín * .....	15
9.3	Carpetas vacías.....	16

9.4	Archivo .gitkeep.....	17
9.5	Añadir una carpeta y todo su contenido.....	17
10	Creando alias para nuestros comandos.....	18
11	Cambios en los archivos.....	18
11.1	Comparar cambios en archivos con visual studio code.....	20
12	Actualizar mensajes de commits y deshacer commits .....	21
12.1	Actualizar el mensaje de un commit .....	21
12.2	Borrar un commit .....	21
13	Preparando un repositorio para viajes en el tiempo .....	21
14	Viajes en el tiempo, resets y reflogs .....	24
15	Cambiar el nombre y eliminar archivos con git .....	27
16	Cambiar el nombre y eliminar archivos fuera de git .....	28
16.1	Renombrar .....	28
16.2	Borrar archivos.....	31
17	Ignorar archivos que no deseamos.....	31
18	Ramas uniones y conflictos.....	33
18.1	Tipos de merge .....	33
18.2	Merge Fast-forward.....	34
18.3	Merge unión automática .....	40
18.4	Merges manuales (con conflictos).....	44
19	Github, Git Remote, Push y Pull.....	48
19.1	Control de acceso al repositorio remoto.....	49
19.2	GitHub.....	50
19.3	Creación de una cuenta en github.....	51
19.4	Push a GitHub .....	52

20 Comando pull .....59

# UNIDAD 3: GIT

## 1 Introducción

Git es un software de control de versiones diseñado por Linus Torvalds, pensando en la eficiencia, la confiabilidad y compatibilidad del mantenimiento de versiones de aplicaciones cuando estas tienen un gran número de archivos de código fuente. Su propósito es llevar registro de los cambios en archivos de computadora incluyendo coordinar el trabajo que varias personas realizan sobre archivos compartidos en un repositorio de código.

### 1.1 Ventajas de GIT

- 1- Facilita el trabajo colaborativo: Distintos programadores pueden estar editando el mismo archivo, o versiones distintas del mismo archivo, y todos los cambios serán reflejados en el documento final.
- 2- Reduce considerablemente los tiempos de deploy (despliegue) de un proyecto, al subir solamente los cambios (no los archivos cambiados, sólo los cambios!), que en Git se conoce como "diff": las diferencias entre la versión local (la que estás trabajando) y la "master" que está en el servidor central.
- 3- Permite regresar a versiones anteriores de forma sencilla y muy rápida. En caso de haber realizado cambios negativos en un proyecto en producción, volver a la última versión estable es un simple comando, que retrocede a su estado previo todos los cambios realizados en la última modificación. Esto puede hacerse hacia cualquier versión del proyecto, sin importar la cantidad o calidad de los cambios posteriores.
- 4- Permite generar flujos de trabajo que facilitan el desarrollo y mantenimiento de proyectos de gran tamaño.
- 5- El ecosistema Git es increíble, y agrega un montón de herramientas a nuestra disposición para facilitarnos el trabajo, de forma robusta, rápida y profesional.

A través de los "hooks" de Git, los distintos servicios pueden detectar cambios en el historial de versiones y realizar acciones automáticas (como actualizar los archivos en el servidor o ejecutar una suite de tests y enviarnos su resultado), dejándonos tiempo libre para cosas más productivas!

6- Las "branches" o ramas, permiten trabajar con una base de código paralela al proyecto en sí, donde podemos corregir bugs o desarrollar nuevas características para el producto sin afectar el "master", pero manteniendo todas las ventajas de usar un sistema de control de versiones. Una vez que estamos contentos con nuestro "branch", podemos combinarlo con el "master" o, en lenguaje Git, hacer un "merge".

7- Empezar a trabajar desde otro entorno es tan fácil como "clonar" el proyecto a tu nuevo entorno, trabajar sobre los archivos que se quieran, y subir los cambios al "master" o a una "branch".

8- Sistema de etiquetas, para etiquetar las distintas versiones del proyecto. Esto es un marcador a una versión específica del proyecto, sólo que en lugar de tener distintos backups de versiones anteriores, apuntamos a distintas versiones dentro de la misma base de código.

## 2 Instalaciones recomendadas:

1. [VSCode - Visual Studio Code](#)
2. [Google Chrome](#)
3. [Git](#)

### 2.1 Extensiones de VSCode

[Activitus Bar](#)

### 2.2 Tema que estoy usando en VSCode

- [Tokio Night](#)
- [Iconos](#)

## 3 Configuración

## 3.1 Primeros comandos

```
git --version
```

Nos muestra la versión de git

```
git help
```

Muestra ayuda sobre los comandos

```
git help "comando"
```

Muestra ayuda sobre un comando concreto. Ejemplo:

```
git help commit
```

## 3.2 Configurando

### 3.2.1 Establecer nombre de usuario

```
git config --global user.name "Álvaro Sarrión"
```

### 3.2.2 Configurar la dirección de correo del usuario

```
git config --global user.email alvarosf@educastur.org
```

Se usa para anotar que usuario realizó cada acción

### 3.2.3 Activar colores de la interfaz

```
git config --global color.ui true
```

### 3.2.4 Ver las configuraciones

```
git config --global -e
```

## 4 Creando el primer repositorio

Un repositorio de Git es un almacenamiento virtual de nuestro proyecto. Permite guardar versiones del código a las que se puede acceder cuando se necesite.

Creamos una carpeta para contener nuestro repositorio y extraemos dentro algunos archivos por ejemplo el contenido del archivo comprimido "01-bases.zip" que contiene una página web.

Desde la línea de comandos nos situamos en la carpeta que será nuestro repositorio:

```
C:\Users\Alvaro>cd D:\Git\01-bases
```

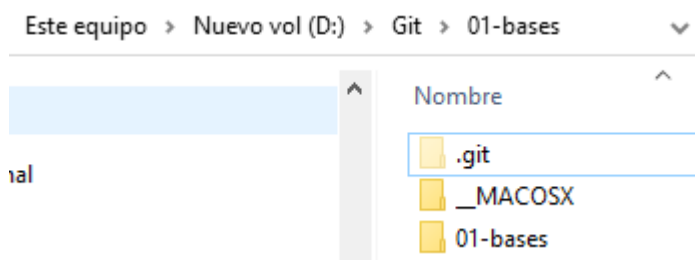
En este caso la carpeta que será nuestro repositorio es 01-bases que es la que contiene la página web.

A continuación iniciamos el repositorio con el comando:

```
git init
```

```
D:\Git\01-bases>git init  
Initialized empty Git repository in D:/Git/01-bases/.git/
```

Dentro de la carpeta donde hemos creado el repositorio se crea la carpeta oculta .git :



Este directorio no se debe borrar porque es el directorio de trabajo de git.

Vamos a ver el estado de nuestro repositorio:

```
git status
```

Vemos que estamos trabajando en la rama master que es la rama principal

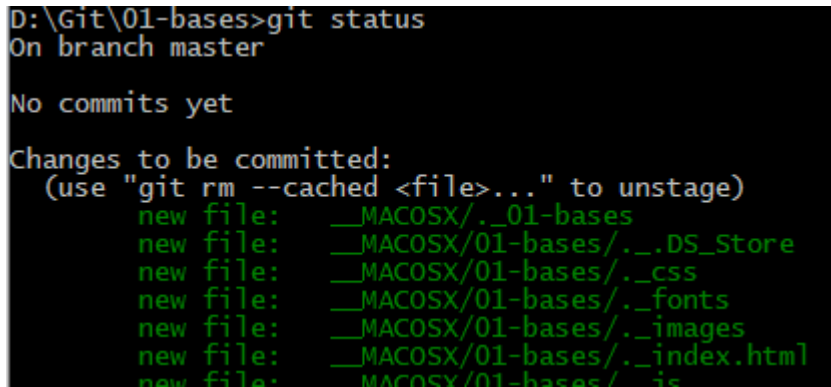
```
D:\Git\01-bases>git status  
On branch master  
  
No commits yet  
  
Untracked files:  
  (use "git add <file>..." to include in what will be committed)  
    01-bases/  
    _MACOSX/  
  
nothing added to commit but untracked files present (use "git add" to track)  
D:\Git\01-bases>
```

Además git nos indica que no estamos haciendo seguimiento de algunos archivos.

Vamos a indicar a git que queremos hacer seguimiento de un archivo:

```
git add __MACOSX
```

Si vemos otra vez el estado con git status:



```
D:\Git\01-bases>git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   __MACOSX/._01-bases
    new file:   __MACOSX/01-bases/._.DS_Store
    new file:   __MACOSX/01-bases/._css
    new file:   __MACOSX/01-bases/._fonts
    new file:   __MACOSX/01-bases/._images
    new file:   __MACOSX/01-bases/._index.html
    new file:   __MACOSX/01-bases/._is
```

Vemos que ahora se nos muestran muchos archivos de los que hacemos seguimiento (los que están en color verde) ya que lo que hemos añadido en vez de un fichero suelto es un directorio y se han añadido todos los archivos dentro de él.

Si queremos añadir todo el contenido de la carpeta actual podemos usar el comando:

```
git add .
```

El punto indica el directorio actual.

De la misma forma que podemos añadir seguimiento de archivos podemos eliminarlo con el comando:

```
git reset <nombre_archivo>
```

Cuando tengamos seleccionados los archivos de los que queremos hacer seguimiento procedemos a realizar una instantánea, es decir guardar el estado actual de los archivos. Para ello usamos el comando:

```
git commit -m "Primer commit"
```

El texto entre comillas sirve para identificar la instantánea tomada.

Si usamos git status después de hacer el commit:



```
D:\Git\01-bases>git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    __MACOSX/

nothing added to commit but untracked files present (use "git add" to track)
```

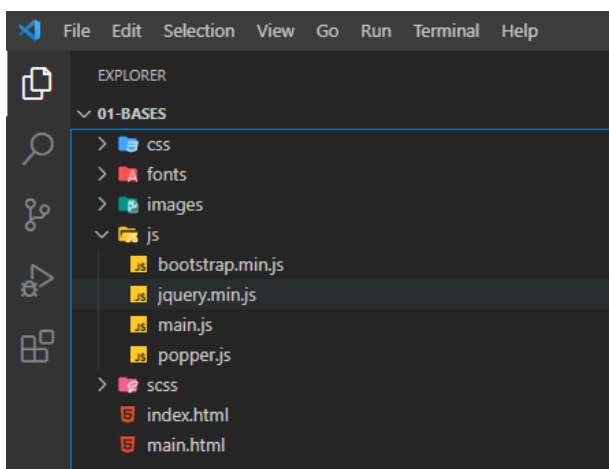
Nos indica que todo lo que habíamos configurado para hacer el seguimiento está guardado y no hay cambios que necesiten hacer commit ("nothing added to commit")

```
git checkout -- .
```

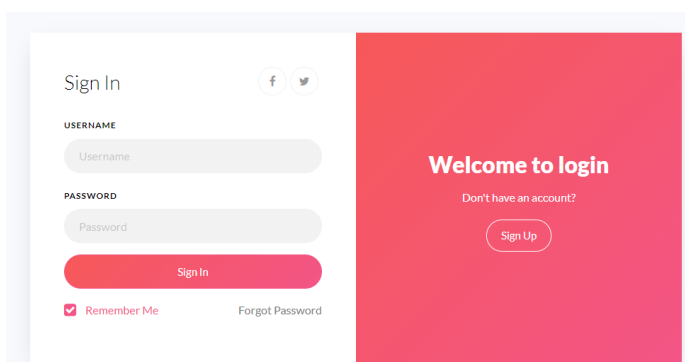
Reconstruye el proyecto al estado del último commit.

## 5 ¿Qué hace git por nosotros hasta el momento?

Arrastramos la carpeta 01-Bases al editor Visual Studio Code. Lo cual hace que se muestre toda la estructura de ficheros que contiene:



Si visualizamos el archivo index.html en el navegador vemos una página web:



Si modificamos el archivo index.html y borramos parte del contenido, puede llegar a no verse la página web, o verse de forma incorrecta como en la siguiente imagen:

asfd asfdasd fasdf a sdf asd  
asfdasdfas fdsaf

## Welcome to login

Don't have an account?

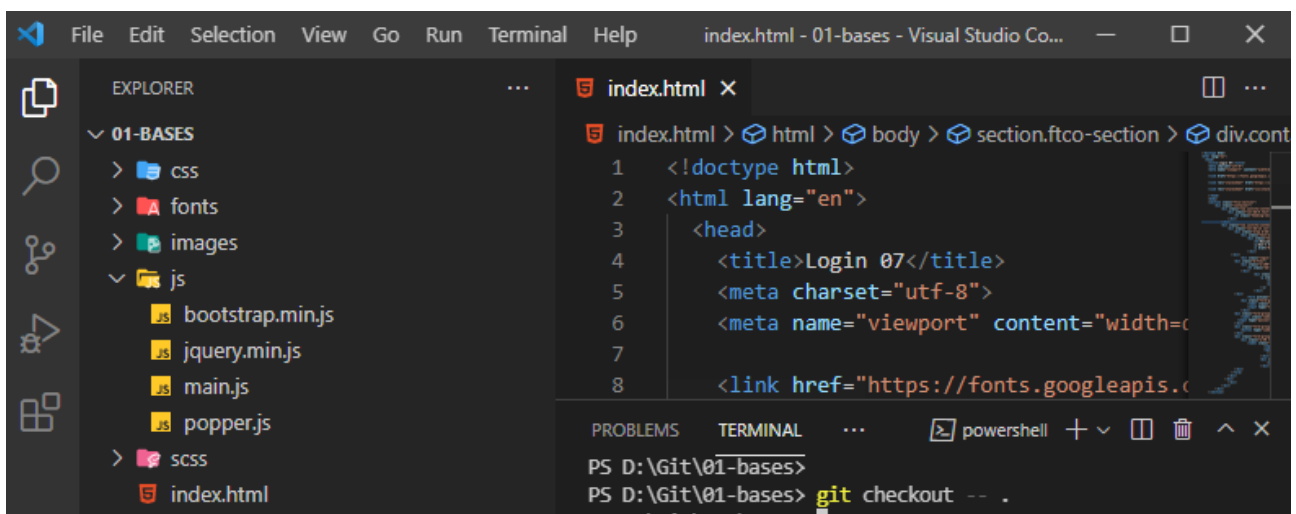
[Sign Up](#)

### Sign In

Username   
Password   
  
Remember Me ☒  
[Forgot Password](#)

Podemos arreglar las modificaciones de forma que la web se vuelva a ver correctamente:

```
git checkout -- .
```



## 6 Cambiar el nombre de la rama Master a Main

En general, una **rama de desarrollo** ("Git Branch") es una bifurcación del estado del código que crea un nuevo camino para la evolución del mismo.

El comando git branch nos indica en que rama estamos trabajando:

```
git branch
```

```
D:\Git\01-bases>git branch
* master
```

## 6.1 Cambiar el nombre de una rama

Para cambiar el nombre de una rama usamos el siguiente comando:

```
git branch -m master main
```

El -m indica que se va a cambiar de nombre a una rama.

A continuación ponemos el nombre de la rama a renombrar (en este caso master).

Finalmente indicamos el nuevo nombre (en este caso main).

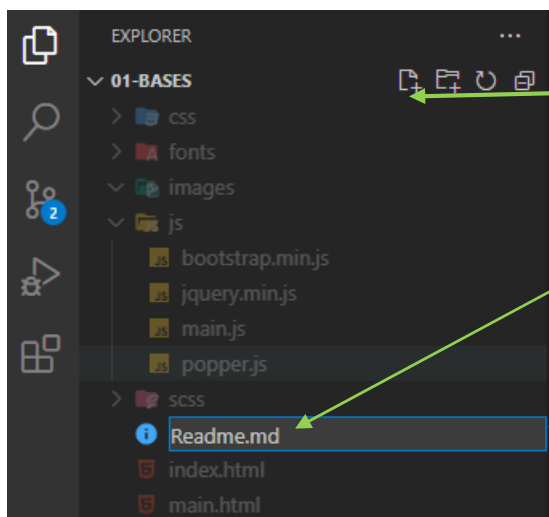
```
D:\Git\01-bases>git branch -m master main
D:\Git\01-bases>git branch
* main
D:\Git\01-bases>
```

## 6.2 Configurar que por defecto la rama principal de un nuevo proyecto se llame main

```
git config --global init.defaultBranch main
```

## 7 Archivo Readme.md y comando log

Vamos a crear un archivo nuevo en nuestro proyecto, al que llamaremos Readme.md:

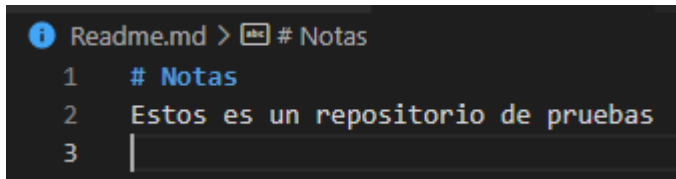


Pulsamos en el botón de nuevo archivo:

Y escribimos el nombre en el recuadro que aparece:

Un archivo README contiene información acerca de otros archivos en un directorio. Es una forma de documentación de software. En Github es habitual agregar un archivo README a un repositorio para comunicar información importante sobre el proyecto.

Escribimos algo de contenido en el archivo:



```
Readme.md > # Notas
1  # Notas
2  Estos es un repositorio de pruebas
3  |
```

Le damos seguimiento al archivo:

```
git add Readme.md
```

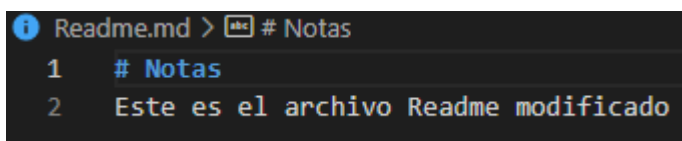
Hacemos commit:

```
git commit -m "Readme añadido"
```

Borramos el archivo desde el visual estudio y como hemos hecho commit previamente podemos recuperarlo con:

```
git checkout -- m
```

Modificamos el archivo Readme:



```
Readme.md > # Notas
1  # Notas
2  Este es el archivo Readme modificado
```

Hacemos un segundo commit con una versión nueva del comando checkout:

```
git commit -am "Readme modificado"
```

Este comando con `-a` sólo funciona si ya le estamos dando seguimiento al archivo pero si estuviera marcado con la U de "untracked" es decir sin seguimiento no funcionaria.

## 7.1 Ver los commits realizados

Vamos a ver los commits que tenemos hechos hasta el momento, para ello vamos a utilizar el siguiente comando:

```
git log
```

Vemos los tres commits que hemos hecho hasta ahora. La etiqueta head indica la última versión

```
PS D:\Git\01-bases> git log
commit 60280ed41e09e2bec6161e4c45d65e3bf2a9f129 (HEAD -> master)
Author: Alvaro <alvarosf@educastur.org>
Date:   Wed Dec 1 16:50:21 2021 +0100

    Readme modificado

commit 206e3b4a753b794d7a21749817ba5338005111aa
Author: Alvaro <alvarosf@educastur.org>
Date:   Wed Dec 1 16:49:35 2021 +0100

    Readme añadido

commit 6b6aaaeb184a2ba8c66c60cb80efd5021b4a0bd8
Author: Alvaro <alvarosf@educastur.org>
Date:   Wed Dec 1 16:48:07 2021 +0100

    Primer commit
```

del repositorio:

Identificador (hash) del commit

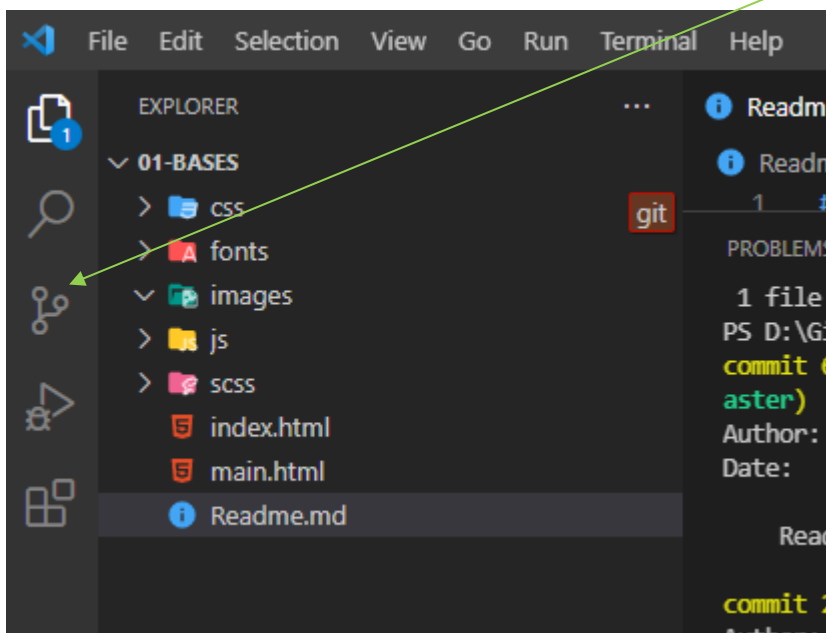
Autor del commit

Fecha y hora del commit

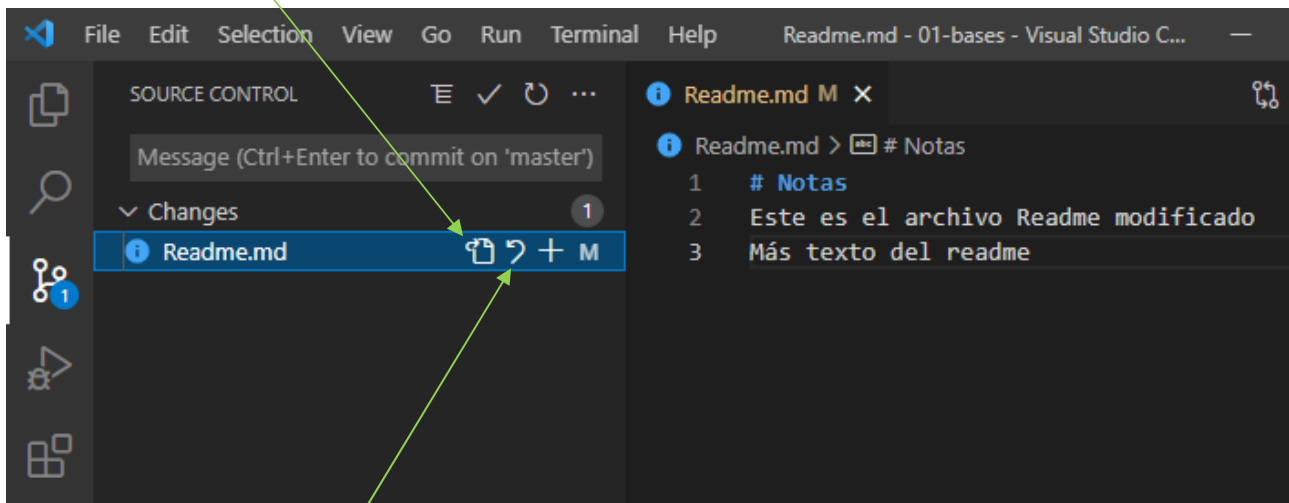
Texto identificativo del commit

## 8 Adds y commits con visual studio code

En Visual studio code tenemos un apartado para realizar operaciones de git:

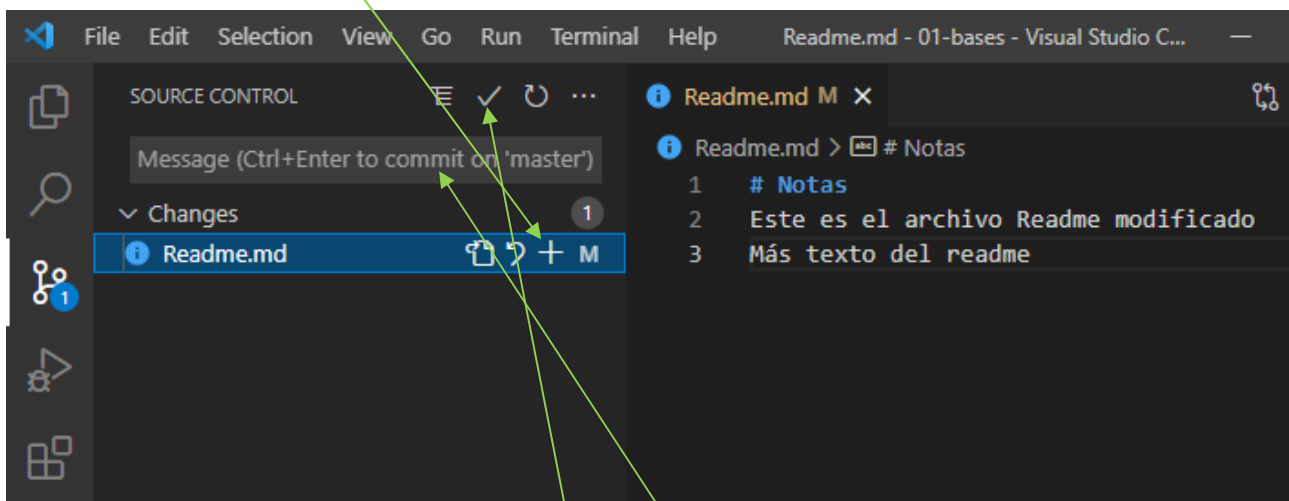


Con este icono abrimos el archivo:



Con este otro deshacemos los cambios realizados en el archivo:

Para añadir los cambios al escenario usamos este botón:



Y para hacer un commit escribimos el mensaje en este campo. Y a continuación utilizamos la combinación Ctrl+Enter o pulsamos en el botón de commit.

## 9 Diferentes formas de agregar archivos al escenario.

Cerramos en Visual Studio Code nuestro proyecto actual que era el de la carpeta 01-bases. Para ello vamos al menú File -> Close Folder o pulsamos la combinación Ctrl + K y luego F.

Descomprimos el archivo 02-bases.zip y copiamos la carpeta 02-bases a la carpeta donde habíamos copiado la carpeta 01-bases.

Arrastramos esta carpeta a visual studio code como hicimos con 01-bases

Nos situamos en la carpeta 02-bases e inicializamos el repositorio:

```
git init
```

Vamos a ver cómo hacer commits de grupos de archivos en vez de commit de todos los archivos como hicimos en apartados anteriores.

### 9.1 Añadir archivos sueltos

```
git add index.html main.html
```

Podemos añadir archivos sueltos simplemente escribiendo sus nombres separados por espacios.

### 9.2 Añadir archivos usando el comodín \*

Podemos añadir todos los archivos html de la siguiente forma:

```
git add *.html
```

A continuación hacemos commit de estos archivos

```
git commit -m "archivos html añadidos"
```

Si ahora intentamos añadir todos los archivos .js

```
git add *.js
```

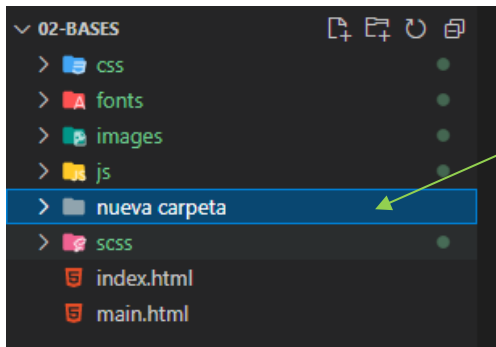
Esto no va a funcionar porque en el directorio actual no hay archivos .js , los archivos .js están en la carpeta js por lo que vamos a tener que indicarlo a la hora de ejecutar el comando:

```
git add js/*.js
```

(**Nota:** Parece que en la versión actual si ponemos `add *.js` si que busca en las carpetas a partir de la carpeta actual y añade todos los archivos .js)

## 9.3 Carpetas vacías

Git no hace seguimiento a las carpetas vacías. Vamos a crear una carpeta nueva para verlo.

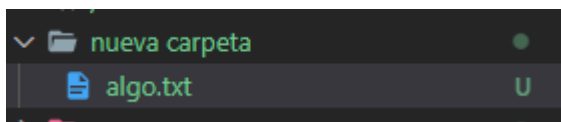


Si hacemos ahora `git status`:

```
PS D:\Git\02-bases> git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    .DS_Store
    css/
    fonts/
    images/
    js/.DS_Store
    scss/
```

Vemos que git ni la muestra en la lista de archivos de los que no se hace seguimiento.

Si queremos que git la tenga en cuenta tenemos que crear un archivo dentro:



Y si ahora hacemos `git status`, vemos que ahora si aparece la nueva carpeta:

```
PS D:\Git\02-bases> git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    .DS_Store
    css/
    fonts/
    images/
    js/.DS_Store
    nueva carpeta/
    scss/
```

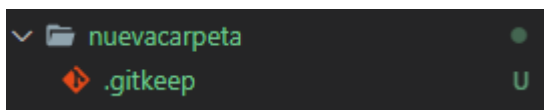


## 9.4 Archivo .gitkeep

Cuando queramos añadir al menos un archivo a una carpeta vacía para que git la tenga en cuenta, en vez de crear cualquier archivo existe un archivo con un nombre especial que podemos crear: el archivo .gitkeep

Este archivo está especialmente pensado para realizar esta función y ocupa un espacio muy pequeño.

(**Nota:** Renombramos “nueva carpeta” como “nuevacarpeta”)



Vemos además que visual studio code le pone un icono específico al archivo con este nombre.

Añadimos este archivo con el siguiente comando:

```
git add nuevaCarpeta/*.gitkeep
```

En algunas versiones este comando puesto así podría fallar y podríamos usar esta otra opción:

```
git add nuevaCarpeta/.gitkeep
```

Hacemos commit:

```
git commit -m ".gitkeep añadido"
```

Si borramos la carpeta y restauramos con:

```
git checkout -- .
```

Vemos que se restaura tanto el archivo .gitkeep como la carpeta que lo contenía.

## 9.5 Añadir una carpeta y todo su contenido

Si queremos añadir por ejemplo todo el contenido de la carpeta css (tanto archivos como directorios que contiene) podemos hacerlo con el siguiente comando:

```
git add css\
```

Hacemos commit:

```
git commit -m "Estilos agregados"
```

## 10 Creando alias para nuestros comandos

Para mostrar el estado de los archivos del repositorio con una descripción corta podemos usar el siguiente comando:

```
git status--short
```

Podemos crear un alias para este comando de forma que podamos ejecutarlo escribiendo menos texto:

```
git config --global alias.s "status --short"
```

Como vemos para crear un alias usamos la configuración global y ponemos alias. y después del punto ponemos el texto que servirá de alias. A continuación escribimos entre comillas el comando que queremos abreviar sin poner git delante.

## 11 Cambios en los archivos

Creamos un nuevo repositorio que se llame 03-instalaciones y lo abrimos en visual studio.

Hacemos el **git init** para inicializar el repositorio y creamos un archivo que se llame instalaciones.md con el siguiente contenido:

```
# Pasos para instalar
Seguir estos pasos:

...

npm install
...
```

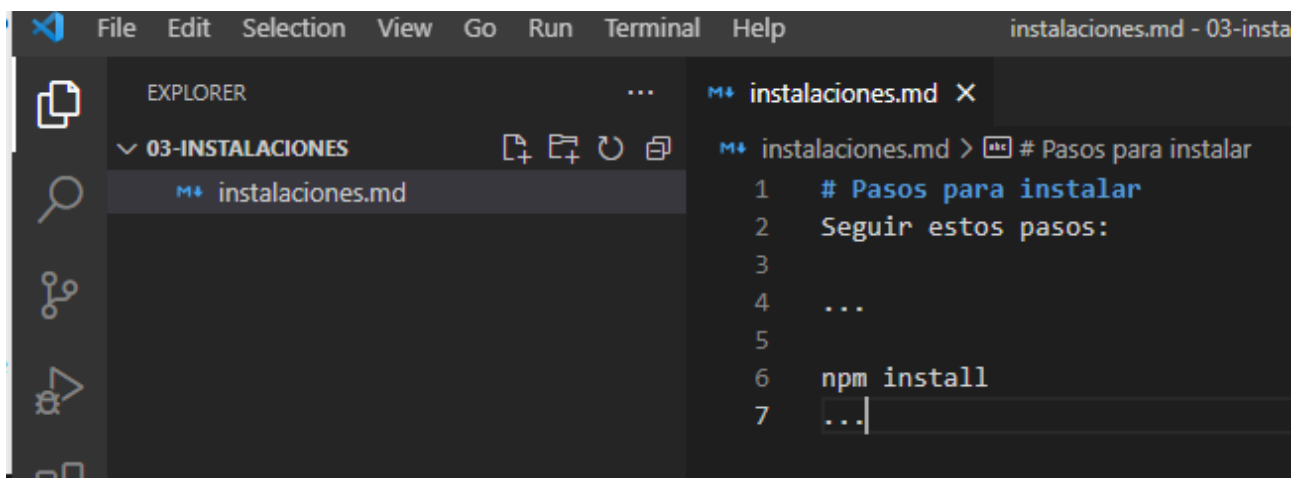
Lo agregamos al stage:

```
git add .
```

Y hacemos commit:

```
git commit -m "Instalaciones agregado"
```

Deberíamos ver algo así en nuestro visual studio code:



Modificamos el archivo instalaciones y lo dejamos así:

```
# Pasos para instalar
Seguir estos pasos:

...

yarn install
...
```

(Hemos modificado una línea y añadido 3 líneas vacías debajo de la segunda línea con ...)

Guardamos los cambios pero no hacemos add ni commit.

Vamos a comparar las modificaciones realizadas en este archivo, para ellos usamos el siguiente comando:

```
git diff
```

La versión a del archivo instalaciones nos dice que tiene menos cosas que la versión b del mismo archivo

```
PS D:\Git\03-instalaciones> git diff
diff --git a/instalaciones.md b/instalaciones.md
index 4920948..d7266ac 100644
--- a/instalaciones.md
+++ b/instalaciones.md
@@ -3,5 +3,7 @@ Seguir estos pasos:

...

-npm install
...
\ No newline at end of file
+yarn install
+.
```

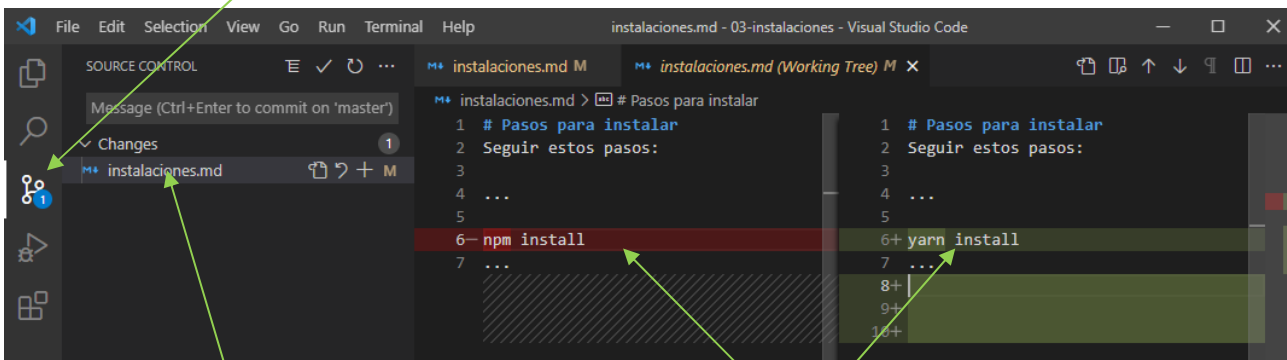
Las líneas en color rojo y precedidas por un - nos indican cosas la primera versión no tiene respecto al segundo y las cosas en verde precedidas por un + nos indican cosas que la segunda versión tiene y la primera no.

El comando git diff nos compara las modificaciones de los archivos que no están en el escenario. Si queremos usar el comando con un archivo que hemos añadido al escenario usaremos lo siguiente:

```
git diff --staged
```

## 11.1 Comparar cambios en archivos con visual studio code

Desde la opción de source control de visual studio podemos comparar las modificaciones realizadas en un archivo de formas más visual:



Hacemos click en el archivo del cual queremos ver las diferencias

## 12 Actualizar mensajes de commits y deshacer commits

### 12.1 Actualizar el mensaje de un commit

Para cambiar el mensaje del último commit realizado usaremos el siguiente comando:

```
git commit --amend -m "Nuevo texto para el commit"
```

### 12.2 Borrar un commit

Para borrar el último commit realizado usaremos el siguiente comando:

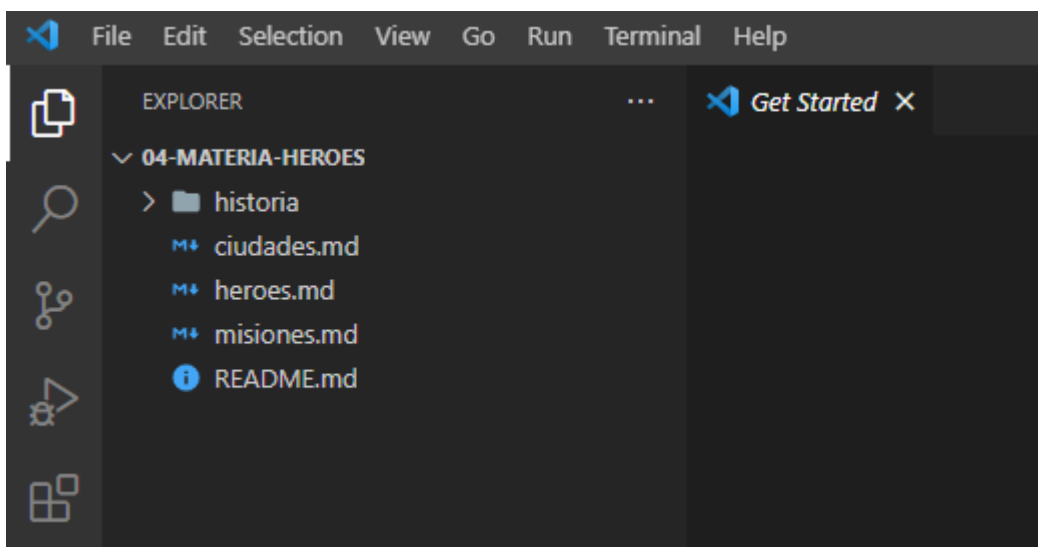
```
git reset --soft HEAD^
```

Esto eliminará el commit y nos dejará el repositorio como estaba antes de hacer dicho commit. Es decir si teníamos archivos modificados volverán a aparecer como modificados. El mantener los cambios se debe al parámetro --soft.

## 13 Preparando un repositorio para viajes en el tiempo

Descomprimos el archivo Materia-Heroes y colocamos la carpeta Materia-Heroes en la carpeta donde estemos creando nuestros repositorios.

Renombramos la carpeta como: 04-Materia-Heroes y arrastramos la carpeta a visual estudio code. Deberíamos tener algo así:



Vamos a ir ejecutando los siguientes comandos:

---

Inicializamos el repositorio:

```
git init
```

---

Añadimos el archivo README.md

```
git add README.md
```

Hacemos el commit de este archivo:

```
git commit -m "README.md agregado"
```

---

Añadimos el archivo misiones.md

```
git add misiones.md
```

Hacemos el commit de este archivo:

```
git commit -m "misiones.md agregado"
```

---

Añadimos el archivo heroes.md

```
git add heroes.md
```

Hacemos el commit de este archivo:

```
git commit -m "heroes.md agregado"
```

---

Añadimos el archivo ciudades.md

```
git add ciudades.md
```

Hacemos el commit de este archivo:

```
git commit -m "ciudades.md agregado"
```

---

Agregamos la carpeta historia

```
git add historia/
```

Hacemos el commit de los archivos agregados:

```
git commit -m "carpeta historia agregada"
```

---

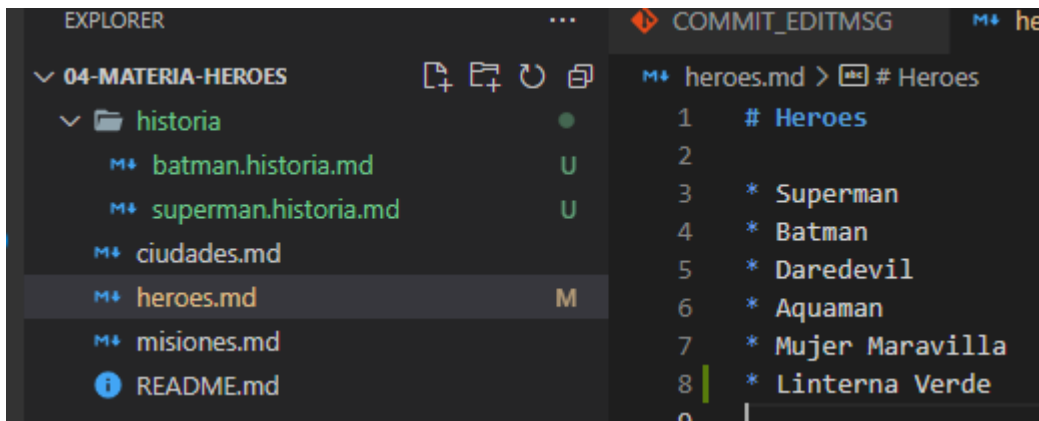
Cambiamos el texto del último commit:

```
git commit --amend
```

Al ejecutar el comando sin poner al final `-m` “nuevo texto para el commit” se abrirá el editor y podremos cambiar en él el texto del commit:

```
.git > COMMIT_EDITMSG
1  Agregada la hitoria de batman y superman
2
3  # Please enter the commit message for your changes. Lines starting
4  # with '#' will be ignored, and an empty message aborts the commit.
5  #
6  # Date:      Thu Dec 16 12:06:24 2021 +0100
7  #
```

Editamos el archivo `héroes.md` y agregamos al héroe Linterna Verde:



Hacemos un nuevo commit:

```
git commit -am "Agregamos el héroe Linterna Verde"
```

Ejecutamos el comando `git lg` para ver un listado de los commits que hemos realizado:

```
PS D:\Git\04-Materia-Heroes> git lg
* ac3cf1d - (2 minutes ago) Agregamos el héroe Linterna Verde - Alvaro (HEAD -> master)
* 80de664 - (23 minutes ago) Agregada la hitoria de batman y superman - Alvaro
* 9dff538 - (24 minutes ago) heroes.md agregado - Alvaro
* 8bc807e - (25 minutes ago) Misiones agregado - Alvaro
* 421469d - (27 minutes ago) README agregado - Alvaro
```

## 14 Viajes en el tiempo, resets y reflogs

Vamos a añadir el héroe Robin en el archivo heroes.md:

En vez de hacer un nuevo commit para añadir este cambio queremos que haya un solo commit con la agregación de Robin y Linterna Verde.

Hacemos un `git lg`:

```
PS D:\Git\04-Materia-Heroes> git lg
* ac3cf1d - (2 minutes ago) Agregamos el héroe Linterna Verde - Alvaro (HEAD -> master)
* 80de664 - (23 minutes ago) Agregada la hitoria de batman y superman - Alvaro
* 9dff538 - (24 minutes ago) heroes.md agregado - Alvaro
* 8bc807e - (25 minutes ago) Misiones agregado - Alvaro
* 421469d - (27 minutes ago) README agregado - Alvaro
```

Vamos a regresar a un commit anterior usando como referencia el código hash del commit al que queremos volver:

```
git reset --soft 80de664
```

Hacemos un nuevo commit:

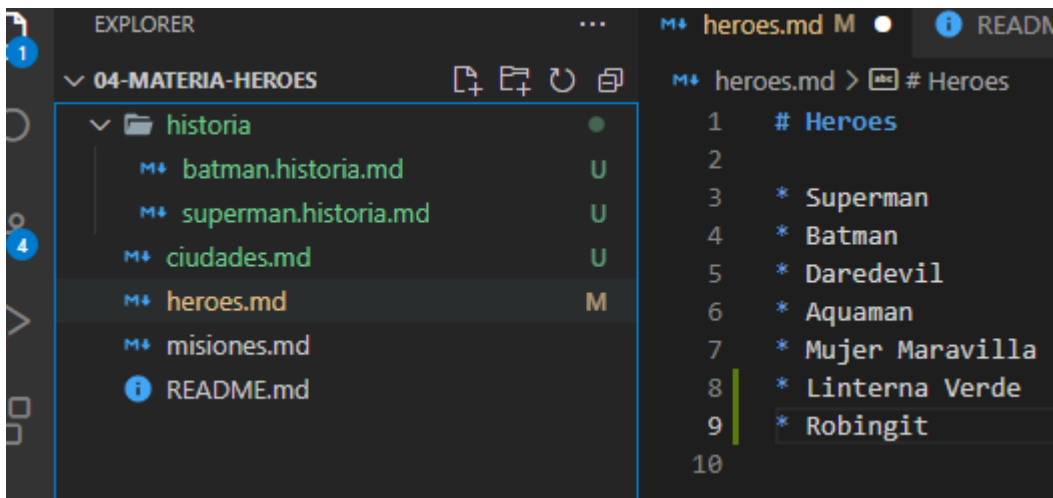
```
git commit -am "Agregamos a los héroes Linterna Verde y Robin"
```

Ahora decidimos que los últimos cambios realizados no están bien y que queremos volver al punto en el que agregamos el archivo de heroes. Y además queremos que los archivos de los que se hubiera hecho commit se queden fuera del stage si es que los habíamos subido. Para ello hacemos un `git reset --mixed`:

```
git reset --mixed 9dff538
```

Si no ponemos una opción del modo de reset (`--soft` `--mixed` etc) el tipo de reset que se hace es el `--mixed` ya que es el modo por defecto.

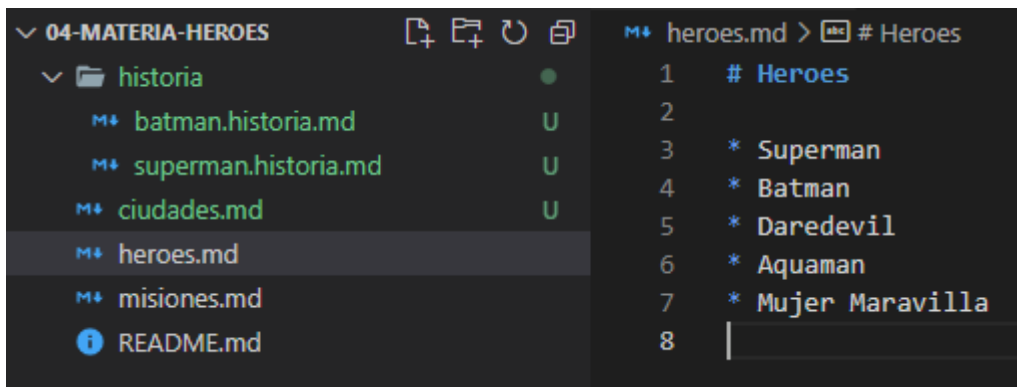




Vemos que los archivos han pasado a estar con la U de untracked es decir sin seguimiento pero que los últimos cambios realizados en heroes.md se han mantenido.

Si queremos hacer un reset que borre los cambios usaremos la opción --hard:

```
git reset --hard 9dff538
```



Vemos que el archivo heroes.md ha vuelto al estado del commit al que hemos regresado.

Hacemos un reset hard al commit donde agregamos las misiones:

```
git reset --hard 8bc807e
```

Si hacemos un nuevo git lg:

```
PS D:\Git\04-Materia-Heroes> git lg
* 8bc807e - (69 minutes ago) Misiones agregado - Alvaro (HEAD -> master)
* 421469d - (71 minutes ago) README agregado - Alvaro
```

Vemos que sólo nos quedan dos commits. Si ahora nos damos cuenta de que estaba todo bien y que queremos volver a un commit de los que borramos, todavía es posible ya que git guarda un historial:

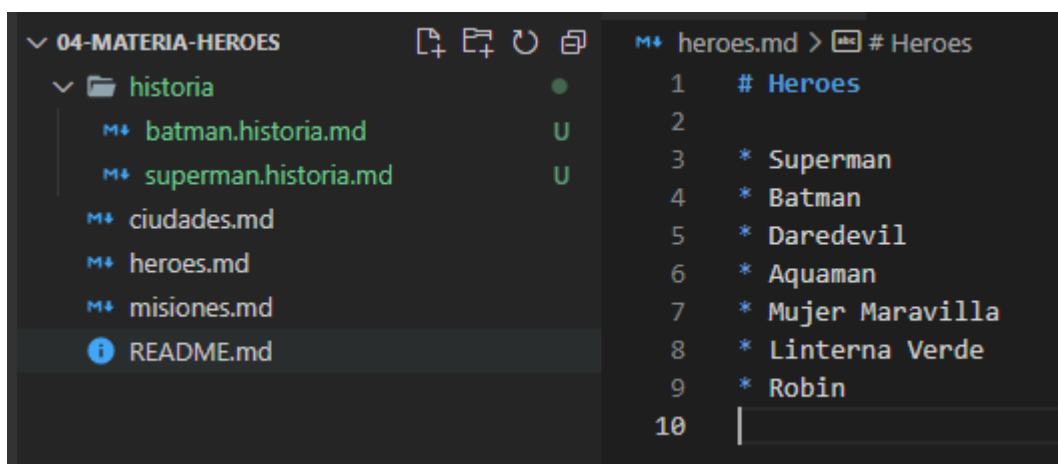
```
git reflog
```

```
PS D:\Git\04-Materia-Heroes> git reflog
8bc807e (HEAD -> master) HEAD@{0}: reset: moving to 8bc807e
9dff538 HEAD@{1}: reset: moving to 9dff538
9dff538 HEAD@{2}: reset: moving to 9dff538
010e1cc HEAD@{3}: commit: Agregamos a los héroes Linterna Verde y Robin
80de664 HEAD@{4}: reset: moving to 80de664
ac3cf1d HEAD@{5}: commit: Agregamos el héroe Linterna Verde
80de664 HEAD@{6}: commit (amend): Agregada la hitoria de batman y superman
cded8f6 HEAD@{7}: commit (amend): Agregada la hitoria de batman y superman
b0d3712 HEAD@{8}: commit: ciudades.md agregado
9dff538 HEAD@{9}: commit: heroes.md agregado
8bc807e (HEAD -> master) HEAD@{10}: commit: Misiones agregado
421469d HEAD@{11}: commit (initial): README agregado
```

Como podemos ver en este historial se guardan tanto los commit como los reset que se fueron realizando. Esto nos permite ver los hash de los commit borrados y si por ejemplo queremos volver al commit en el que agregamos a los héroes Linterna Verde y Robin no tenemos más que ejecutar el siguiente comando:

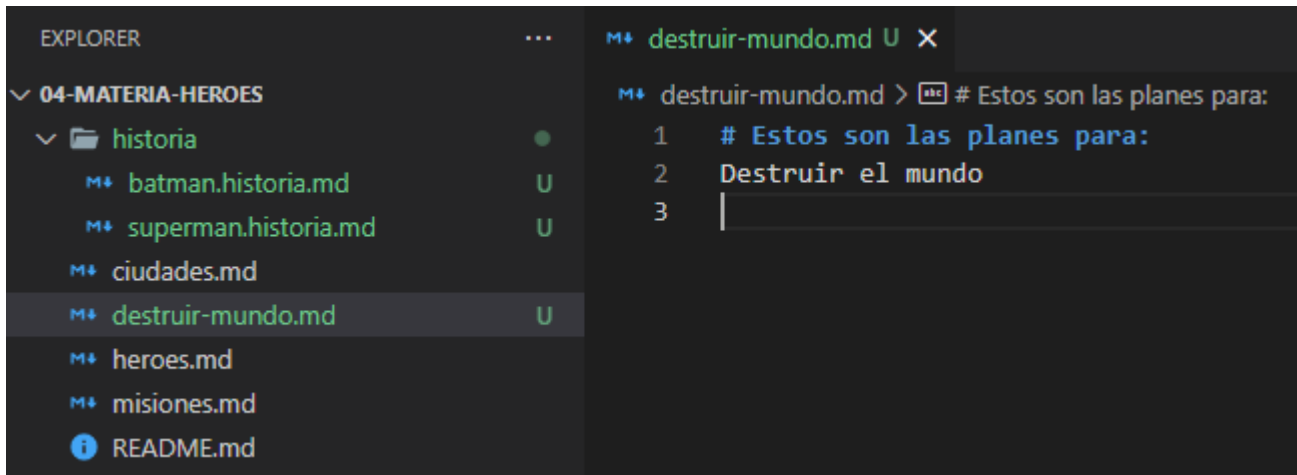
```
git reset --hard 010e1cc
```

Vemos que todo vuelve a ese punto:



## 15 Cambiar el nombre y eliminar archivos con git

Creamos mediante visual studio un nuevo archivo llamado “destruir-mundo.md”:



Guardamos los cambios del archivo.

Lo subimos al escenario:

```
git add .
```

Hacemos commit:

```
git commit -m "Destruir-mundo añadido"
```

Vamos a cambiar el nombre a este archivo:

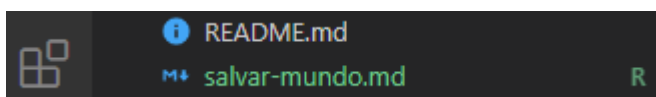
```
git mv destruir-mundo.md salvar-mundo.md
```

Aunque el comando `git mv` es para mover un archivo al moverlos a la misma ruta con diferente nombre lo que hacemos es renombrarlo.

Al renombrarlo vemos que si usamos el comando `git s` se muestra el archivo con una R que indica que fue renombrado:

```
PS D:\Git\04-Materia-Heroes> git s
R  destruir-mundo.md -> salvar-mundo.md
```

Visual studio también muestra una R de renombrado:



El archivo renombrado está además ya subido al stage listo para hacer commit:

```
git commit -m "Destruir-mundo renombrado"
```

Podemos también borrar un archivo con un comando de git:

```
git rm salvar-mundo.md
```

Después de ejecutar este comando el archivo queda marcado con la letra D de borrado:

```
D  salvar-mundo.md
PS D:\Git\04-Materia-Heroes> git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        deleted:    salvar-mundo.md
```

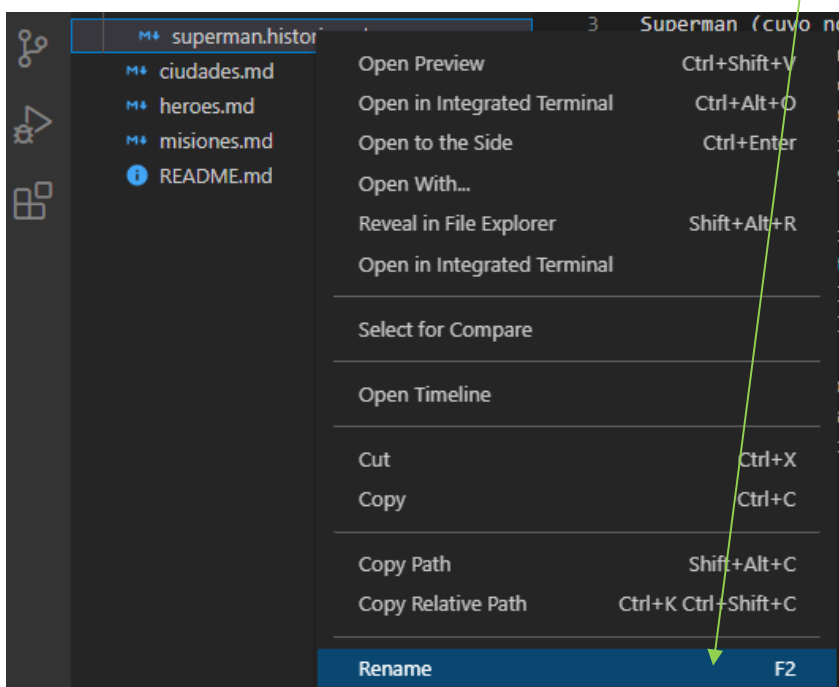
Para confirmar el borrado hacemos commit:

```
git commit -m "Salvar-mundo borrado"
```

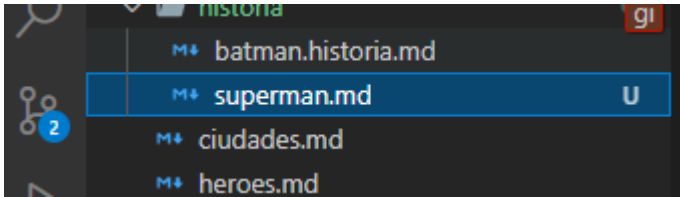
## 16 Cambiar el nombre y eliminar archivos fuera de git

### 16.1 Renombrar

Vamos a renombrar el archivo "superman.historia.md" desde la interfaz de visual studio. Pulsamos botón derecho sobre el archivo y elegimos la opción Rename:



Lo renombramos como "superman.md"



Vemos que el archivo renombrado queda marcado con la U de untracked (sin seguimiento) en vez de con la R de rename como sucedía en el apartado anterior al renombrar un archivo con el comando git.

Si hacemos un `git status` o `git s` vemos lo siguiente:

```
PS D:\Git\04-Materia-Heroes> git s
D historia/superman.historia.md
?? historia/superman.md
```

Para git es como si hubiéramos borrado el archivo original que como vemos queda marcado con una D, y el archivo renombrado lo marca con dos interrogaciones considerando que es un archivo nuevo.

Si ahora subimos los cambios al stage:

```
git add .
```

Si ahora hacemos `git s`

```
PS D:\Git\04-Materia-Heroes> git s
R historia/superman.historia.md -> historia/superman.md
```

Vemos que ahora el sólo aparece el archivo renombrado marcado con la R de renomend (renombrado). Al hacer `git add .` git ha analizado los archivos y se ha dado cuenta de que son el mismo y se ha producido un renombrado.

Hacemos commit de los cambios:

```
git commit -m "Historia de superman renombrada"
```

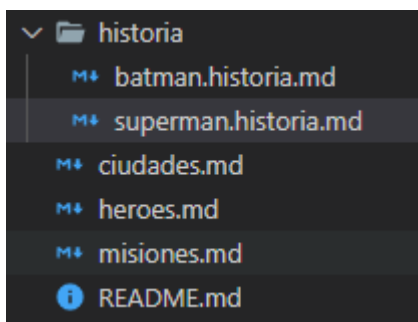
Hacemos un `git lg` para ver nuestro commits:

```
PS D:\Git\04-Materia-Heroes> git lg
* 3411dd0 - (89 seconds ago) Historia de superman renombrada - Alvaro (HEAD -> master)
* 817d605 - (18 hours ago) Salvar-mundo borrado - Alvaro
* 89a7327 - (19 hours ago) Destruir-mundo renombrado - Alvaro
* 4f0e2ec - (3 days ago) Destruir-mundo añadido - Alvaro
* 010e1cc - (2 weeks ago) Agregamos a los héroes Linterna Verde y Robin - Alvaro
* 80de664 - (2 weeks ago) Agregada la hitoria de batman y superman - Alvaro
* 9dff538 - (2 weeks ago) heroes.md agregado - Alvaro
* 8bc807e - (2 weeks ago) Misiones agregado - Alvaro
* 421469d - (2 weeks ago) README agregado - Alvaro
```

Vamos a movernos al commit “Salvar-mundo borrado”:

```
git reset --hard 817d605
```

Observamos que el archivo de historia de superman vuelve a tener su nombre original:



Si queremos deshacer el reset y volver al punto anterior hacemos:

```
git reflog
```

```
PS D:\Git\04-Materia-Heroes> git reflog
817d605 (HEAD -> master) HEAD@{0}: reset: moving to 817d605
3411dd0 HEAD@{1}: commit: Historia de superman renombrada
817d605 (HEAD -> master) HEAD@{2}: commit: Salvar-mundo borrado
89a7327 HEAD@{3}: commit: Destruir-mundo renombrado
4f0e2ec HEAD@{4}: commit: Destruir-mundo añadido
010e1cc HEAD@{5}: reset: moving to 010e1cc
8bc807e HEAD@{6}: reset: moving to 8bc807e
9dff538 HEAD@{7}: reset: moving to 9dff538
9dff538 HEAD@{8}: reset: moving to 9dff538
010e1cc HEAD@{9}: commit: Agregamos a los héroes Linterna Verde y Robin
80de664 HEAD@{10}: reset: moving to 80de664
ac3cf1d HEAD@{11}: commit: Agregamos el héroe Linterna Verde
```

El comando reflog nos permite ver el commit “Historia de superman renombrada” que no veríamos con git log:

```
PS D:\Git\04-Materia-Heroes> git reset --hard 3411dd0
HEAD is now at 3411dd0 Historia de superman renombrada
```

## 16.2 Borrar archivos

Borramos el archivo `batman.historia.md` desde visual studio pulsando botón derecho sobre él y eligiendo la opción Delete.

Si ejecutamos `git s` tras borrarlo:

```
PS D:\Git\04-Materia-Heroes> git s
D historia/batman.historia.md
```

Vemos que el archivo aparece marcado con la D de deleted (borrado) pero no está subido al escenario como sucedía cuando borrábamos con un comando git.

Lo subimos al escenario:

```
git add .
```

Hacemos commit:

```
git commit -m "Historia de batman borrada"
```

## 17 Ignorar archivos que no deseamos

Vamos a crear algunos carpetas y archivos de prueba para luego configurar que git no les de seguimiento.

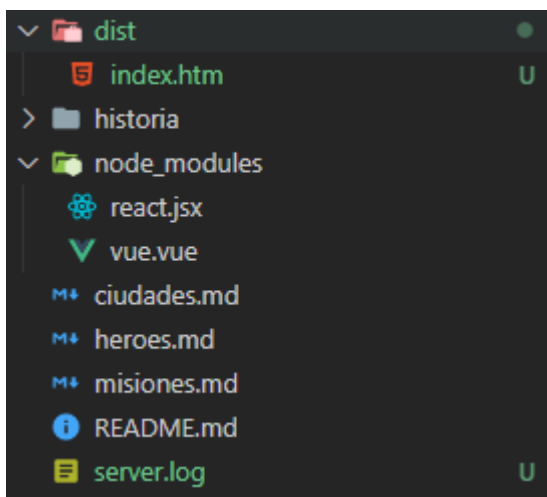
Creamos las carpetas:

- `dist`
- `node_module`

Y los siguientes archivos:

- `server.log`
- `dist/index.html`
- `node_modules/react.jsx`
- `node_modules/vue.vue`

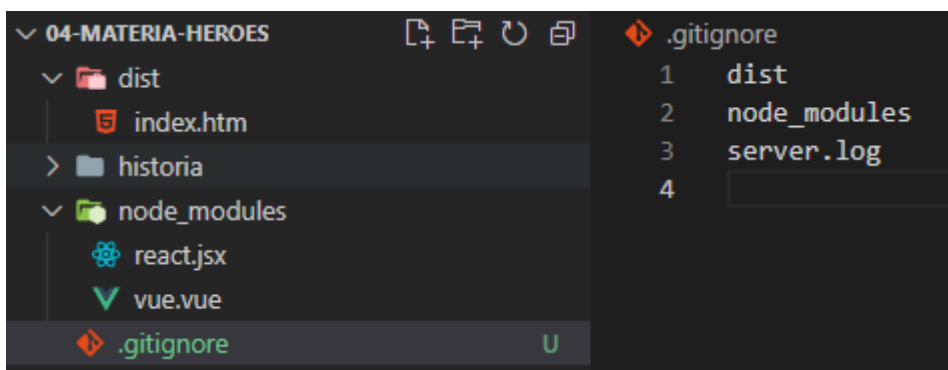
En los archivos podemos poner el texto que queramos o dejarlos vacíos.



Si hacemos `git s` vemos que git nos indica que los nuevos archivos no tienen seguimiento:

```
PS D:\Git\04-Materia-Heroes> git s
?? dist/
?? node_modules/
?? server.log
```

Ahora lo que queremos es configurar que git ignore estos archivos porque no queremos darles seguimiento. Para ello creamos en la carpeta raíz del proyecto un archivo llamado **.gitignore** y dentro de este archivo escribiremos los nombres de archivos y directorios que queremos que git ignore. Podemos usar comodines para indicar los archivos:



Si después de escribir este archivo y guardar los cambios hacemos nuevamente `git s`:

```
PS D:\Git\04-Materia-Heroes> git s
?? .gitignore
```

Vemos que efectivamente git está ignorando todos los archivos de las dos carpetas nuevas y el archivo `server.log`, sólo aparece como archivo sin seguimiento el archivo que hemos creado **.gitignore**



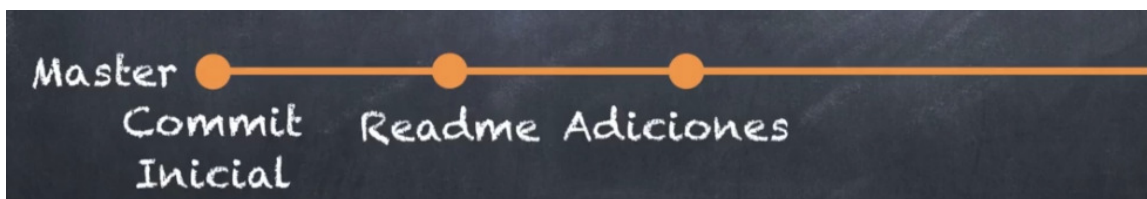
A este archivo si le vamos a dar seguimiento ya que queremos que se guarden las configuraciones que hemos hecho:

```
git add .  
git commit -m "Archivo .gitignore creado"
```

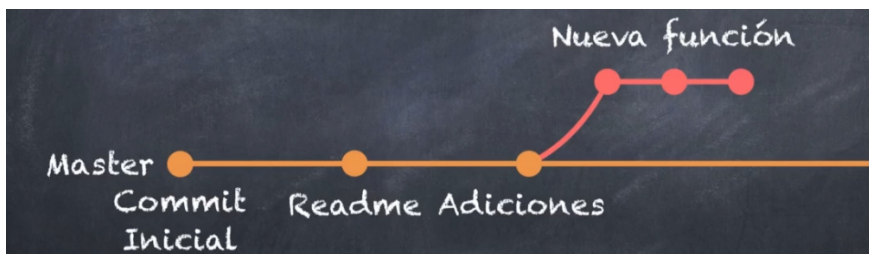
## 18 Ramas uniones y conflictos

Una rama de un proyecto es una copia en la que empezamos a trabajar y a realizar modificaciones y commits. Desde el punto de vista de git podemos ver cada rama como un camino independiente de commits.

Hasta ahora hemos estado trabajando con sólo una rama que era la rama **master** que en un momento dado vimos como renombrar a **main** por ser más políticamente correcto:



Crear una nueva rama se puede ver visualmente como una bifurcación en el camino actual:

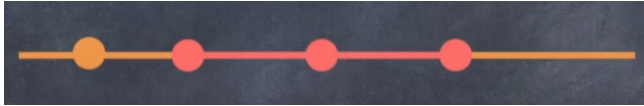


En un momento dado puede que queramos que una rama secundaria se integre en la principal a esto se le llama **unión** o **merge**:



### 18.1 Tipos de merge

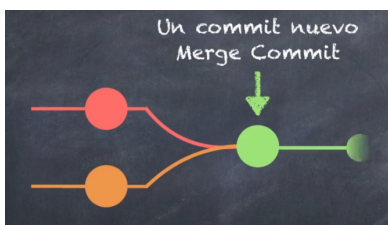
- **Fast-forward:** este tipo de unión se realiza cuando no ha habido cambios en la rama principal y por lo tanto se pueden unir sin problema las dos ramas simplemente agregando los commits de la rama secundaria a la principal:



- **Uniones automáticas:** este tipo de unión se realiza hay cambios en la rama principal pero que no entran en conflicto con los cambios realizados en la rama secundaria, por lo cual git puede realizar la unión de forma automática:

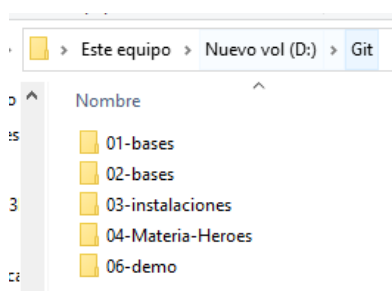


- **Manuales:** en este caso hay modificaciones en la rama principal y en la secundaria que entran en conflicto, por lo que git no puede decidir por sí mismo como hacer la unión y lo que hace es preguntarnos como realizar la unión y a continuación realizar un commit que se denomina **Merge Commit**

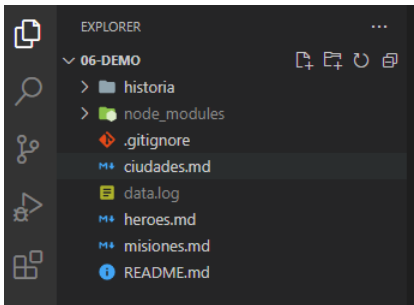


## 18.2 Merge Fast-forward

Vamos a descomprimir el archivo de recursos 06-demo.zip en la carpeta de git donde estamos subiendo nuestros repositorios. Al final tendremos que tener una nueva carpeta que se llamará 06-demo:



Cerramos la carpeta que tuviéramos abierta en visual studio code y arrastramos esta nueva carpeta 06-demo:



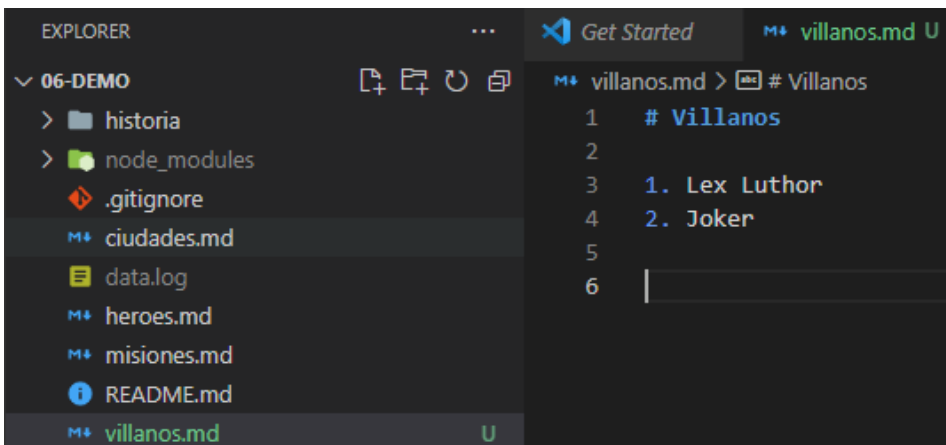
Este repositorio ya tiene la carpeta oculta .git es decir es un repositorio en el que ya se ha inicializado git y además tiene ya una serie de commits realizados y además tiene el archivo .gitignore también creado.

Vamos a ver los commits que tiene el repositorio en este momento:

```
git lg
```

```
PS D:\Git\06-demo> git lg
* 62c8a10 - (4 years, 7 months ago) Agregando el gitignore - Strider (HEAD -> master)
* ac0d374 - (4 years, 7 months ago) Borramos la historia de batman - Strider
* b4c748c - (4 years, 7 months ago) Cambiamos el nombre de la historia de superman - Strider
* d877f01 - (4 years, 7 months ago) Borrando archivo salvar mundo - Strider
* c9ee153 - (4 years, 7 months ago) Renombrando archivo a salvar-mundo - Strider
* fa3cd3a - (4 years, 7 months ago) Creando el archivo destruir el mundo - Strider
* 4e889d4 - (4 years, 7 months ago) Agregamos a Linterna verde y a Robin - Strider
* 345d7de - (4 years, 7 months ago) Editamos el readme.md - Strider
* 860c6c2 - (4 years, 7 months ago) Agregamos las historias de los heroes - Strider
* bc1a1e5 - (4 years, 7 months ago) Agregamos las ciudades - Strider
* 6b8f60d - (4 years, 7 months ago) Agregamos los heroes - Strider
* da24862 - (4 years, 7 months ago) Agregamos las misiones - Strider
* 88a423d - (4 years, 7 months ago) Se agrego el archivo readme - Strider
```

Supongamos que estamos pensando en añadir al proyecto un apartado de villanos, aunque no estamos seguros de si al final esa parte se quedará en el proyecto o no. De momento vamos a crear un archivo llamados villanos.md con el texto que se ve en la imagen de más abajo:



Vamos a crear una nueva rama que se llame rama-villanos:

```
git branch rama-villanos
```

Ahora para ver las ramas existentes y en cual estamos ahora mismo ejecutamos el comando:

```
git branch
```

```
PS D:\Git\06-demo> git branch
* master
  rama-villanos
```

El asterisco y el color verde nos indica que estamos en la rama master.

Para cambiar a la nueva rama escribimos:

```
git checkout rama-villanos
```

```
PS D:\Git\06-demo> git checkout rama-villanos
Switched to branch 'rama-villanos'
```

Como vemos un mensaje nos indica que nos hemos movido a la rama rama-villanos.

Si hacemos `git lg` vemos lo siguiente:

```
* 62c8a10 - (4 years, 7 months ago) Agregando el gitignore - Strider (HEAD -> rama-villanos, master)
* ac0d374 - (4 years, 7 months ago) Borramos la historia de batman - Strider
* b4c748c - (4 years, 7 months ago) Cambiamos el nombre de la historia de superman - Strider
```

Vemos que la cabecera apunta a la rama rama-villanos y que la rama master está en el mismo punto.

Agregamos el archivo al stage y hacemos commit:

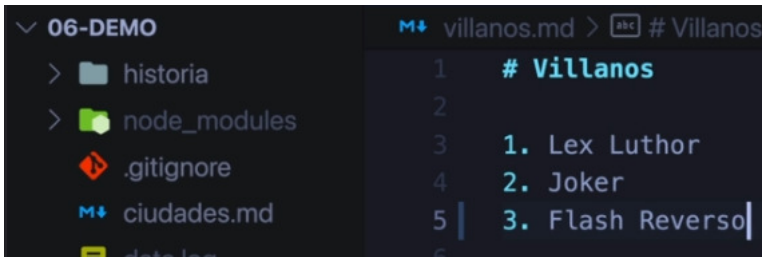
```
git add .
git commit -m "Villanos.md agregado"
```

Si hacemos nuevamente `git lg`:

```
PS D:\Git\06-demo> git lg
* e1672bb - (41 seconds ago) Villanos.md agregado - Alvaro (HEAD -> rama-villanos)
* 62c8a10 - (4 years, 7 months ago) Agregando el gitignore - Strider (master)
* ac0d374 - (4 years, 7 months ago) Borramos la historia de batman - Strider
* b4c748c - (4 years, 7 months ago) Cambiamos el nombre de la historia de superman - Strider
```

Vemos que el Head apunta a la rama rama-villanos y que la rama master se ha quedado un commit atrás.

Añadimos un nuevo villano al archivo de villanos:



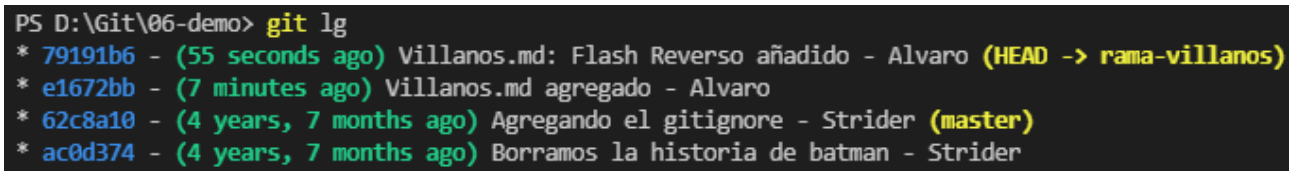
```
06-DEMO
├── historia
├── node_modules
├── .gitignore
├── ciudades.md
└── data.log
```

```
villanos.md > # Villanos
1 # Villanos
2
3 1. Lex Luthor
4 2. Joker
5 3. Flash Reverso
```

Hacemos un nuevo commit con este cambio usando la forma que sube el archivo al escenario y hace el commit a la vez:

```
git commit -am "Villanos.md: Flash Reverso añadido"
```

Hacemos nuevamente `git lg`:

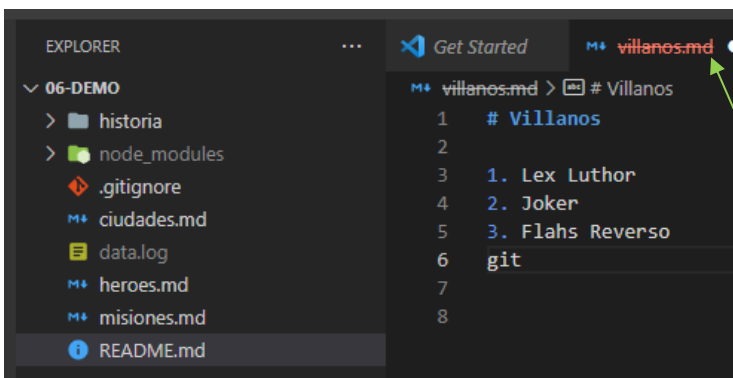


```
PS D:\Git\06-demo> git lg
* 79191b6 - (55 seconds ago) Villanos.md: Flash Reverso añadido - Alvaro (HEAD -> rama-villanos)
* e1672bb - (7 minutes ago) Villanos.md agregado - Alvaro
* 62c8a10 - (4 years, 7 months ago) Agregando el gitignore - Strider (master)
* ac0d374 - (4 years, 7 months ago) Borrarnos la historia de batman - Strider
```

Vemos que la rama master se encuentra ya dos commits por detrás de la rama-villanos.

Si ahora cambiamos a la rama master:

```
git checkout master
```



```
EXPLORER
├── 06-DEMO
│   ├── historia
│   ├── node_modules
│   ├── .gitignore
│   ├── ciudades.md
│   ├── data.log
│   ├── heroes.md
│   ├── misiones.md
│   └── README.md
└── villanos.md
```

```
villanos.md > # Villanos
1 # Villanos
2
3 1. Lex Luthor
4 2. Joker
5 3. Flahs Reverso
6 git
7
8
```

Vemos que el archivo villanos.md desaparece ya que en la rama master no está, y además visual studio code nos indica que la versión que teníamos abierta de villanos.md corresponde a un archivo borrado y por eso nos lo muestra en color rojo y tachado.

Si vuelvo a la rama rama-villanos el archivo volverá a aparecer:

```
git checkout rama-villanos
```

Ahora decidimos que los cambios implementados en esta funcionalidad de villanos son correctos y por lo tanto queremos integrarlos en la rama principal que es la rama master. Al hacer la unión de ramas hay que tener en cuenta que rama tenemos seleccionada, ya que la rama seleccionada recibirá los cambios de la otra rama.

Si hacemos `git branch`:

```
PS D:\Git\06-demo> git branch
master
* rama-villanos
```

Vemos que ahora mismo está seleccionada la rama rama-villanos y de hacer la unión ahora los cambios de la rama master pasarían a la rama-villanos y lo que queremos es agregar los cambios de la rama rama-villanos en la rama master. Por lo tanto cambiamos de rama:

`git checkout master`

Hacemos la unión o merge:

`git merge rama-villanos`

```
PS D:\Git\06-demo> git merge rama-villanos
Updating 62c8a10..79191b6
Fast-forward
 villanos.md | 7 ++++++
 1 file changed, 7 insertions(+)
 create mode 100644 villanos.md
```

Vemos que se ha realizado un merge fast-forward que es la mejor opción porque implica que no ha habido conflictos y git ha podido hacer la unión sin ningún tipo de problemas.

También nos indica que al hacer el merge se incluyó el archivo villanos.md y 7 modificaciones.

Si hacemos una vez más `git lg`:

```
PS D:\Git\06-demo> git lg
* 79191b6 - (29 minutes ago) Villanos.md: Flash Reverso añadido - Alvaro (HEAD -> master, rama-villanos)
* e1672bb - (35 minutes ago) Villanos.md agregado - Alvaro
* 62c8a10 - (4 years, 7 months ago) Agregando el gitignore - Strider
```

Vemos que después del merge las dos ramas están en el mismo punto

Como hemos terminado el desarrollo que estábamos haciendo en la rama rama-villanos y ya hemos unido su funcionalidad con la rama master, no necesitamos ya la rama rama-villanos. Vamos por lo tanto a borrar esta rama:

```
git branch -d rama-villanos
```

```
PS D:\Git\06-demo> git branch -d rama-villanos  
Deleted branch rama-villanos (was 79191b6).
```

Si hubiera cambios en una rama que queremos borrar que no han sido unidos a otra rama, git nos avisará por si estamos borrando la rama por error. En ese caso podríamos forzar el borrado de la rama añadiendo el parámetro `-f`:

```
git branch -d rama-villanos -f
```

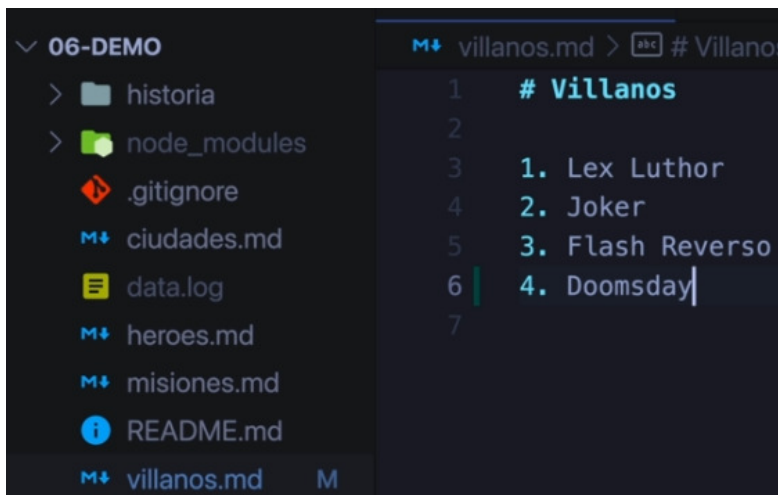
## 18.3 Merge unión automática

Vamos a trabajar en unas modificaciones sobre el archivo de villanos y para ello vamos a crear una nueva rama. En el apartado anterior vimos como crear una rama y posteriormente movernos a ella usando por tanto dos comandos distintos. Vamos a ver ahora como crear una rama y movernos a la misma con un solo comando:

```
git checkout -b rama-villanos
```

```
PS D:\Git\06-demo> git checkout -b rama-villanos  
Switched to a new branch 'rama-villanos'
```

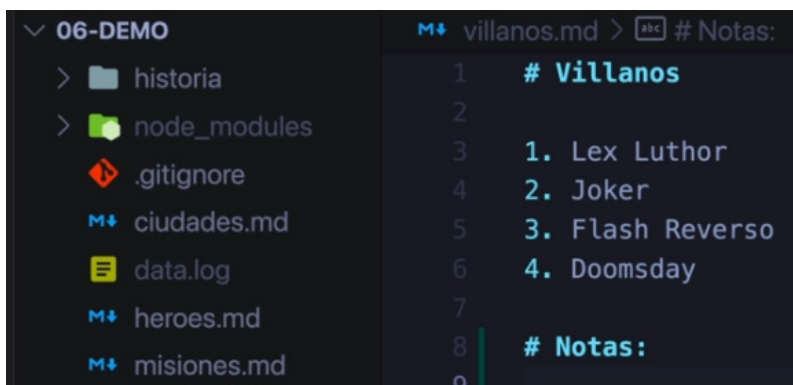
A continuación modificamos el archivo villanos.md y guardamos los cambios:



Hacemos commit de estos cambios:

```
git commit -am "Villanos.md: Agregamos a Doomsday"
```

Hacemos otra modificación en el archivo villanos.md:



Y realizamos un nuevo commit:



```
git commit -am "Villanos.md: Notas agregadas"
```

Hacemos git lg:

```
PS D:\Git\06-demo> git lg
* 4a31144 - (3 seconds ago) Villanos.md: Notas agregadas - Alvaro (HEAD -> rama-villanos)
* 462527f - (2 minutes ago) Villanos.md: Agregamos a Doomsday - Alvaro
* 79191b6 - (89 minutes ago) Villanos.md: Flash Reverso añadido - Alvaro (master)
* e1672bb - (2 hours ago) Villanos.md agregado - Alvaro
```

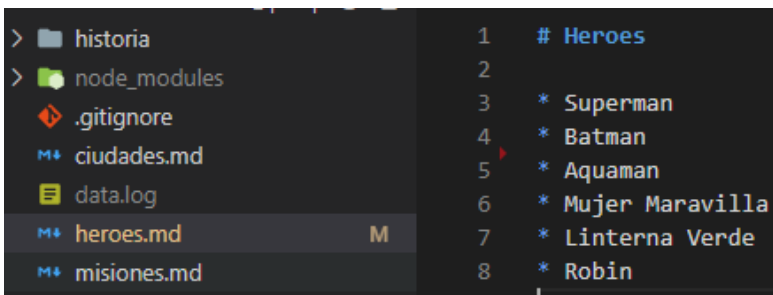
Vemos que la rama master está dos commits por detrás de rama-villanos.

Ahora nos piden que hagamos un cambio en la rama master ya que hay un error en el archivo de heroes ya que que Daredevil es un héroe de Marvel y queremos tener sólo heroes del universo DC en ese archivo.

Nos cambiamos a la rama master:

```
git checkout master
```

Borramos Daredevil del archivo de heroes y guardamos los cambios:



```
> historia
> node_modules
> .gitignore
M+ ciudades.md
E data.log
M+ heroes.md M
M+ misiones.md
```

```
1 # Heroes
2
3 * Superman
4 * Batman
5 * Aquaman
6 * Mujer Maravilla
7 * Linterna Verde
8 * Robin
```

Hacemos commit de este cambio:

```
git commit -am "Heroes.md: Daredevil borrado"
```

Si hacemos `git lg`:

```
PS D:\Git\06-demo> git lg
* 99966c1 - (4 minutes ago) Heroes.md: Daredevil borrado - Alvaro (HEAD -> master)
| * 4a31144 - (10 minutes ago) Villanos.md: Notas agregadas - Alvaro (rama-villanos)
| * 462527f - (12 minutes ago) Villanos.md: Agregamos a Doomsday - Alvaro
|/
* 79191b6 - (2 hours ago) Villanos.md: Flash Reverso añadido - Alvaro
* e1672bb - (2 hours ago) Villanos.md agregado - Alvaro
* 62c8a10 - (4 years, 7 months ago) Agregando el gitignore - Strider
* ac0d374 - (4 years, 7 months ago) Borrarnos la historia de batman - Strider
```

Vemos que la cabecera está en la rama master y que la rama rama-villanos está un commit por detrás. Además visualmente vemos que hay dos ramas.

Vamos a realizar la unión de estas dos ramas:

Queremos unir los cambios de rama-villanos a la rama master por lo que comprobamos si estamos en ella con `git branch`:

```
PS D:\Git\06-demo> git branch
* master
  rama-villanos
```

En esta ocasión ya nos encontrábamos en la rama master ya que estuvimos trabando en ella modificando el archivo de heroes.

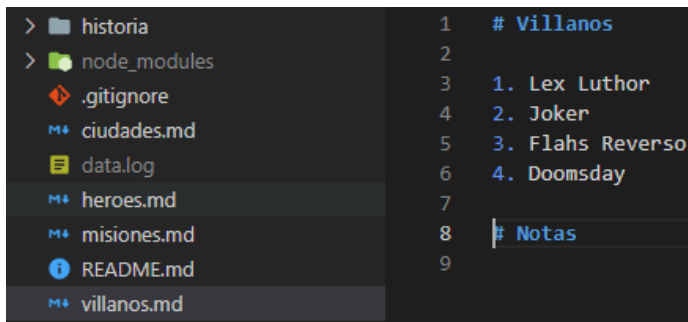
Hacemos el merge:

```
git merge rama-villanos
```

```
PS D:\Git\06-demo> git merge rama-villanos
Merge made by the 'recursive' strategy.
 villanos.md | 3 ++-
 1 file changed, 2 insertions(+), 1 deletion(-)
```

Vemos que git indica que ha realizado la unión usando la estrategia 'recursive' que es a la que estamos llamando en este apartado unión automática, en la cual ha habido cambios en la rama master pero no hay conflictos y git ha podido hacer la unión sin preguntarnos nada.

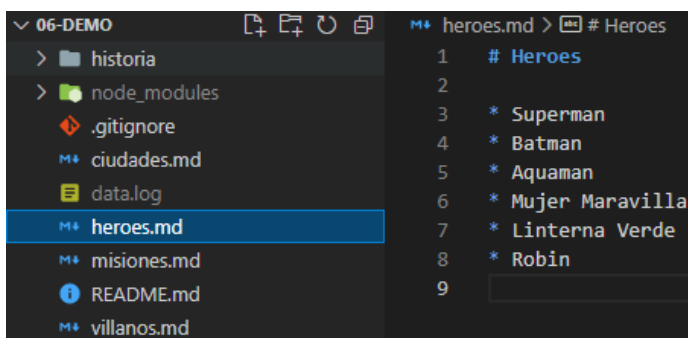
Después de hacer la unión podemos ver que tenemos el archivo de villanos con los 4 villanos y el apartado de notas:



```
> historia
> node_modules
.gitignore
ciudades.md
data.log
heroes.md
misiones.md
README.md
villanos.md

1 # Villanos
2
3 1. Lex Luthor
4 2. Joker
5 3. Flahs Reverse
6 4. Doomsday
7
8 # Notas
9
```

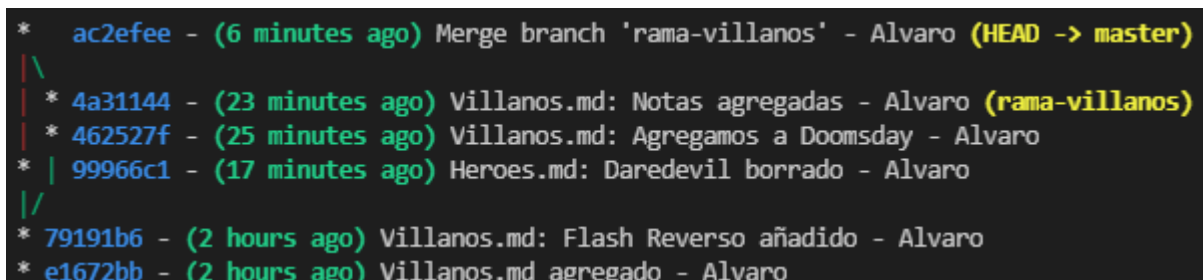
Y que en el archivo héroes no tenemos a Daredevil:



```
06-DEMO
> historia
> node_modules
.gitignore
ciudades.md
data.log
heroes.md
misiones.md
README.md
villanos.md

heroes.md > # Heroes
1 # Heroes
2
3 * Superman
4 * Batman
5 * Aquaman
6 * Mujer Maravilla
7 * Linterna Verde
8 * Robin
9
```

Si hacemos una vez más `git lg`:



```
* ac2efee - (6 minutes ago) Merge branch 'rama-villanos' - Alvaro (HEAD -> master)
| \
| * 4a31144 - (23 minutes ago) Villanos.md: Notas agregadas - Alvaro (rama-villanos)
| * 462527f - (25 minutes ago) Villanos.md: Agregamos a Doomsday - Alvaro
| * | 99966c1 - (17 minutes ago) Heroes.md: Daredevil borrado - Alvaro
| /
|
| * 79191b6 - (2 hours ago) Villanos.md: Flash Reverseo añadido - Alvaro
| * e1672bb - (2 hours ago) Villanos.md agregado - Alvaro
```

Vemos los dos caminos de commits y como se unieron en el último commit realizado al llevar a cabo la unión de las dos ramas.

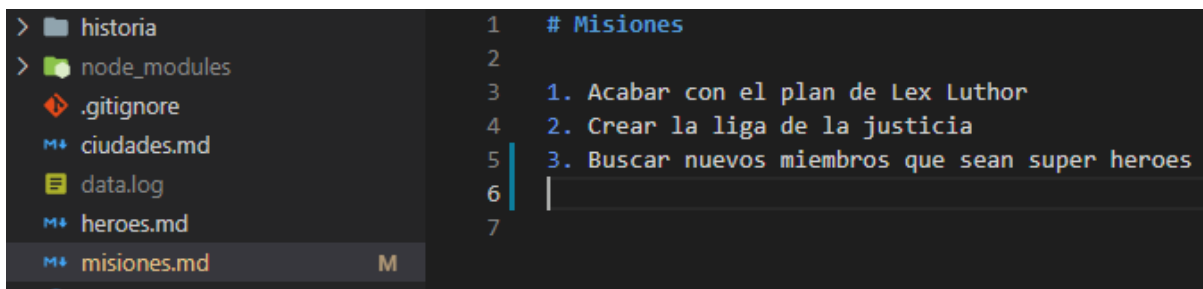
## 18.4 Merges manuales (con conflictos)

Vamos a crear una nueva rama que se va a llamar rama-conflicto:

```
git checkout -b rama-conflicto
```

```
PS D:\Git\06-demo> git checkout -b rama-conflicto
Switched to a new branch 'rama-conflicto'
```

Vamos a modificar el archivo misiones.md y guardar los cambios:



```
> historia
> node_modules
> .gitignore
M+ ciudades.md
data.log
M+ heroes.md
M+ misiones.md M
1 # Misiones
2
3 1. Acabar con el plan de Lex Luthor
4 2. Crear la liga de la justicia
5 3. Buscar nuevos miembros que sean super heroes
6
7
```

Comprobamos que estamos en la rama rama-conflicto:

```
PS D:\Git\06-demo> git branch
master
* rama-conflicto
rama-villanos
```

Si nos aparece la rama-villanos del apartado anterior como en la imagen la borramos ya que para este apartado no la necesitamos:

```
PS D:\Git\06-demo> git branch -d rama-villanos
Deleted branch rama-villanos (was 4a31144).
```

Hacemos commit de los cambios realizados en el archivo misiones.md:

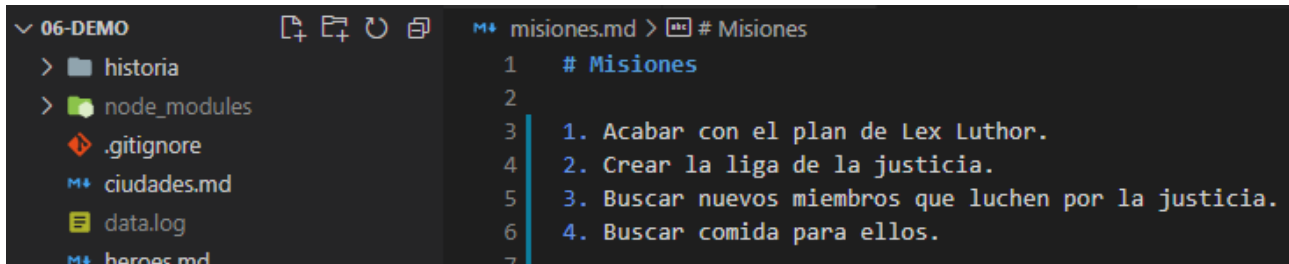
```
git commit -am "Misiones.md actualizado"
```

Miramos el estado actual:

```
PS D:\Git\06-demo> git lg
* 716bed8 - (47 seconds ago) Misiones.md actualizado - Alvaro (HEAD -> rama-conflicto)
* ac2efee - (3 hours ago) Merge branch 'rama-villanos' - Alvaro (master)
| \
| * 4a31144 - (3 hours ago) Villanos.md: Notas agregadas - Alvaro
| * 462527f - (3 hours ago) Villanos.md: Agregamos a Doomsday - Alvaro
* | 99966c1 - (3 hours ago) Heroes.md: Daredevil borrado - Alvaro
| /
* 79191b6 - (4 hours ago) Villanos.md: Flash Reverso añadido - Alvaro
```

Vamos a regresar a la rama master y realizar cambios en el mismo archivo villanos.md:

```
PS D:\Git\06-demo> git checkout master  
Switched to branch 'master'
```



(Nota: se ha cambiado la línea 3 y añadido la línea 4, además **se han añadido puntos en todas las líneas**)

Hacemos commit de estos cambios:

```
git commit -am "Misiones.md: Actualizado en la rama master"
```

Comprobamos el estado actual:

```
PS D:\Git\06-demo> git lg  
* b1ea060 - (85 seconds ago) Misiones.md: Actualizado en la rama master - Alvaro (HEAD -> master)  
| * 716bed8 - (10 minutes ago) Misiones.md actualizado - Alvaro (rama-conflicto)  
|/  
* ac2efee - (3 hours ago) Merge branch 'rama-villanos' - Alvaro  
|\  
| * 4a31144 - (3 hours ago) Villanos.md: Notas agregadas - Alvaro  
| * 462527f - (3 hours ago) Villanos.md: Agregamos a Doomsday - Alvaro  
* | 99966c1 - (3 hours ago) Heroes.md: Daredevil borrado - Alvaro  
|/  
* 79191b6 - (5 hours ago) Villanos.md: Flash Reverso añadido - Alvaro  
* e1672bb - (5 hours ago) Villanos.md agregado - Alvaro
```

Ahora mismo tenemos dos ramas que tienen el mismo archivo modificado, con modificaciones diferentes en cada una de las ramas.

Vamos a realizar la unión o merge. Nos aseguramos que estamos en la rama master ya que queremos traer los cambios de la rama rama-conflicto a la rama master:

```
PS D:\Git\06-demo> git branch  
* master  
rama-conflicto
```

Realizamos el merge:

```
git merge rama-conflicto
```

```
PS D:\Git\06-demo> git merge rama-conflicto
Auto-merging misiones.md
CONFLICT (content): Merge conflict in misiones.md
Automatic merge failed; fix conflicts and then commit the result.
```

Git nos muestra un error. Nos dice que ha intentado hacer el merge automático (Auto-merging) pero que no es posible realizarlo porque hay conflictos que hay que resolver de manera manual.

Si abrimos el archivo misiones.md veremos algo así (si no nos aparece algo parecido puede deberse a que lo teníamos abierto con anterioridad, en ese caso debemos cerrarlo y volverlo a abrirlo):

```
1  # Misiones
2
3  Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
4  <<<<<< HEAD (Current Change)
5  1. Acabar con el plan de Lex Luthor.
6  2. Crear la liga de la justicia.
7  3. Buscar nuevos miembros que luchen por la justicia.
8  4. Buscar comida para ellos.
9  =====
10 1. Acabar con el plan de Lex Luthor
11 2. Crear la liga de la justicia
12 3. Buscar nuevos miembros que sean super heroes
13 >>>>>> rama-conflicto (Incoming Change)
```

Vemos las dos versiones del archivo. En color verde nos indica que es la versión actual ya que es la de la rama en la que estamos. En color azul nos marca la versión de la rama rama-conflicto y nos dice que es la versión entrante (incoming).

La forma de resolver el conflicto de forma manual es comparar visualmente ambas versiones e ir editando el archivo hasta dejar la versión que queramos. Por ejemplo podemos dejar una mezcla de las dos versiones con las 4 líneas pero quitando los puntos:

```
> historia
> node_modules
> .gitignore
> ciudades.md
> data.log
> heroes.md
> misiones.md

1  # Misiones
2
3  1. Acabar con el plan de Lex Luthor
4  2. Crear la liga de la justicia
5  3. Buscar nuevos miembros que luchen por la justicia
6  4. Buscar comida para ellos.
7
8
```

Guardamos los cambios.

Si ahora hacemos `git s`:

```
PS D:\Git\06-demo> git s
UU misiones.md
```

Vemos que git nos marca el conflicto en la unión de las dos versiones del fichero con dos U en color rojo.

Si lo vemos con la versión `git status`:

```
PS D:\Git\06-demo> git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)
    both modified:   misiones.md

no changes added to commit (use "git add" and/or "git commit -a")
```

Vemos que git nos da más información y nos dice que tenemos un conflicto que tenemos que resolver y hacer luego commit. También nos dice que podemos deshacer el merge usando `git merge --abort`

Como ya hemos arreglado el conflicto de forma manual editando el fichero con las dos versiones sólo nos falta hacer el commit:

`git commit -am "Unión con rama-conflicto"`

```
PS D:\Git\06-demo> git commit -am "Unión con rama-conflicto"
[master 037c571] Unión con rama-conflicto
```

Si hacemos `git s` nuevamente vemos que ya se resolvió el conflicto:

```
PS D:\Git\06-demo> git s
PS D:\Git\06-demo> 
```

Y si miramos el estado actual:

```
PS D:\Git\06-demo> git lg
* 037c571 - (4 minutes ago) Unión con rama-conflicto - Alvaro (HEAD -> master)
| \
| * 716bed8 - (37 minutes ago) Misiones.md actualizado - Alvaro (rama-conflicto)
* | b1ea060 - (29 minutes ago) Misiones.md: Actualizado en la rama master - Alvaro
| /
* ac2efee - (3 hours ago) Merge branch 'rama-villanos' - Alvaro
| \
| * 4a31144 - (4 hours ago) Villanos.md: Notas agregadas - Alvaro
| * 462527f - (4 hours ago) Villanos.md: Agregamos a Doomsday - Alvaro
* | 99966c1 - (3 hours ago) Heroes.md: Daredevil borrado - Alvaro
```

Vemos que ya tenemos el commit en el que hemos unido la rama master con la rama rama-conflicto.

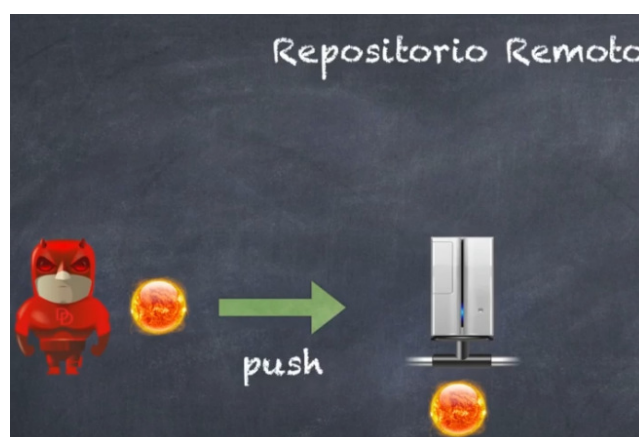
Como ya no necesitamos la rama rama-conflicto la borramos:

```
git branch -d rama-conflicto
```

```
PS D:\Git\06-demo> git branch -d rama-conflicto
Deleted branch rama-conflicto (was 716bed8).
```

## 19 Github, Git Remote, Push y Pull

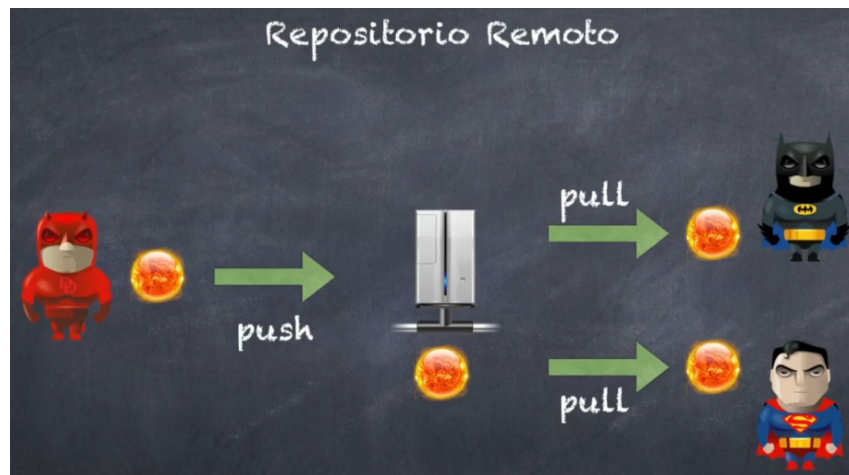
Si por alguna razón se nos estropea el disco duro donde tenemos nuestro repositorio, por mucho que tengamos muchos commits perderemos toda la información. Por eso es recomendable tener una copia en un repositorio remoto, es decir en un repositorio que no sea nuestro equipo.



Para subir una copia de nuestro repositorio a un repositorio remoto se usa el **comando push**.



Por otra parte si varias personas están trabajando sobre el mismo repositorio y alguien ha utilizado el **comando push** para subir cambios, las demás personas podrán actualizar su repositorio local con estos cambios utilizando el **comando pull**:



## 19.1 Control de acceso al repositorio remoto.

Git no se encarga de controlar el acceso al repositorio remoto, es decir de controlar que usuario tiene permiso para modificar ciertos archivos y cuáles no. De esto se encargan otras herramientas.

Tenemos dos tipos de herramientas para controlar el acceso al repositorio remoto:

**Hosted services:** en este caso usamos los servicios que nos proporciona alguna plataforma.

**Servicios gestionados por nosotros:** utilizamos una herramienta desde nuestro equipo la cual nos permite configurar todo por nosotros mismos.



## 19.2 GitHub

GitHub es una plataforma de desarrollo colaborativo de software para alojar proyectos.

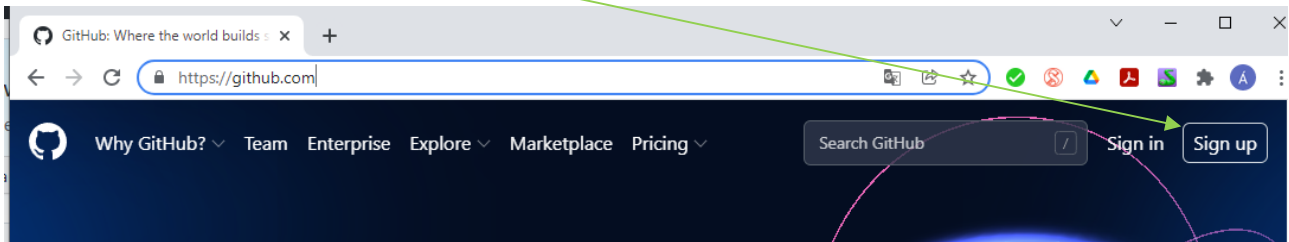
GitHub es una plataforma muy popular porque su versión gratuita nos proporciona muchas características como:

- Repositorios ilimitados
- Páginas HTML, CSS y JS ilimitadas
- Push, Pull, Clones ilimitados
- Issues, Wikis, estadísticas ilimitadas
- Organizaciones ilimitadas
- Desde 2019 permite crear repositorios privados además de públicos.

## 19.3 Creación de una cuenta en github

Para crear una cuenta iremos a la web de GitHub: <https://github.com/>

Y pulsaremos en el **enlace Sign Up**:



A continuación tendremos que rellenar un formulario con datos introduciendo entre otros un correo electrónico al que nos mandarán datos para verificar la cuenta:

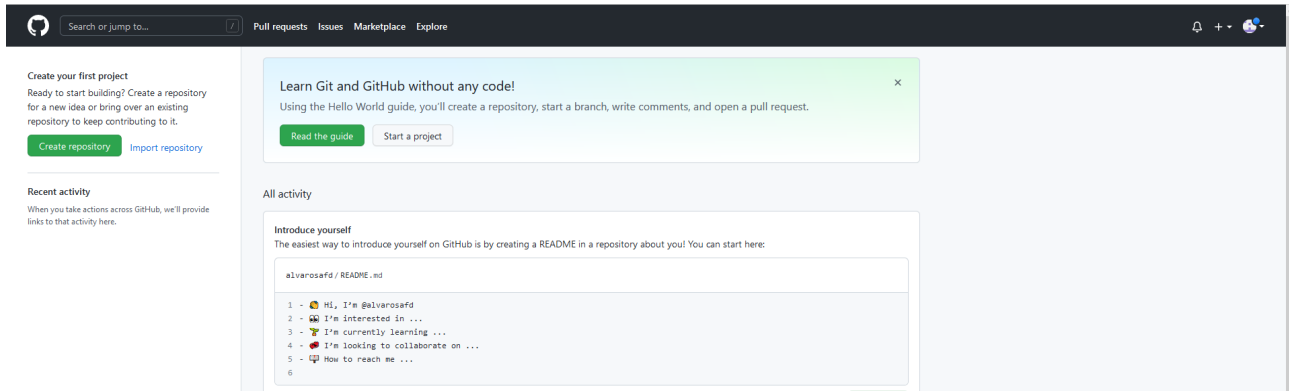
A screenshot of the GitHub sign-up form, which has a dark background with light-colored text. The form contains the following fields and prompts: 'Welcome to GitHub! Let's begin the adventure', 'Enter your email' with a green checkmark and the email 'klerith@yahoo.com', 'Create a password' with a green checkmark and a masked password '.....', 'Enter a username' with a green checkmark and the username 'KlerithX2', and a checkbox prompt 'Would you like to receive product updates and announcements via email? Type "y" for yes or "n" for no'. At the bottom right is a green 'Continue' button.

Después de rellenar el formulario y completar un cáctup tendremos que verificar la cuenta introduciendo el código recibido en el correo:

A screenshot of the GitHub account verification screen. It has a dark background with light-colored text. The text reads: 'You're almost done!', 'We sent a launch code to klerith@yahoo.com', and '→ Enter code'. Below the text are six empty input boxes for the verification code. At the bottom, there is a link that says 'Didn't get your email? Resend the code or update your email address.'

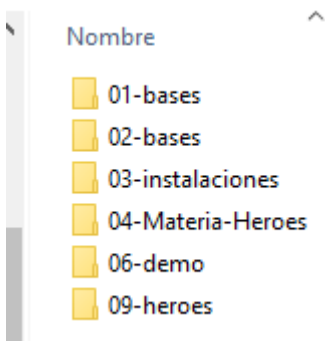
En algún punto del proceso nos permitirá elegir si queremos una cuenta de estudiante lo cual nos proporcionará acceso a funciones avanzadas de forma gratuita. Para ello será necesario adjuntar mediante foto un documento acreditativo de ser estudiante. Este documento os lo habrá facilitado el tutor con anterioridad.

Una vez esté la cuenta creada y estemos logueados veremos algo así:

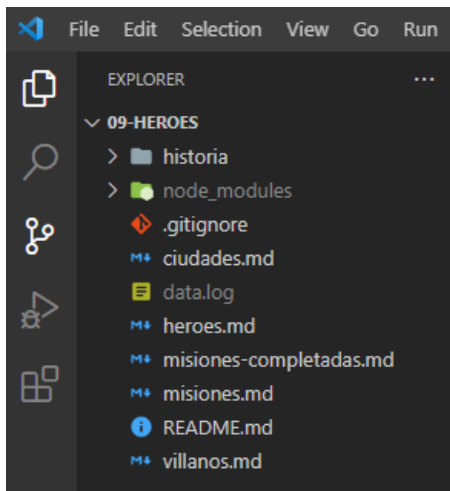


## 19.4 Push a GitHub

Vamos a coger el archivo de recursos 09-heroes.zip y descomprimirlo. Cogemos la carpeta 09-heroes y la copiamos a la carpeta que estamos usando para guardar nuestros repositorios. Nos debería quedar algo así:



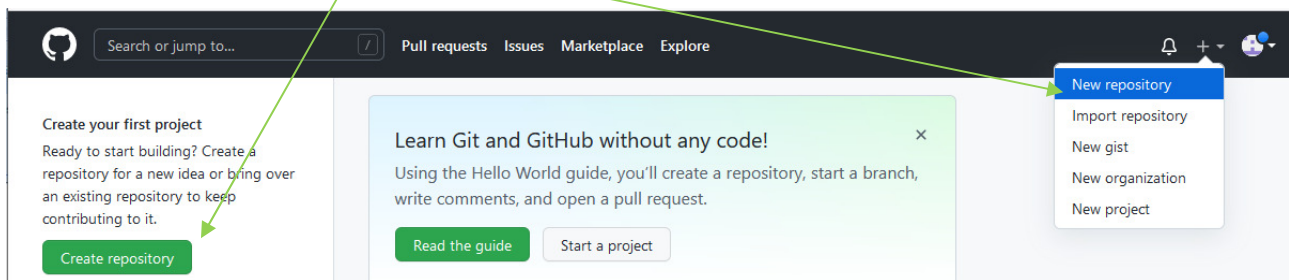
Arrastramos la carpeta 09-heroes a visual studio code (cerramos previamente la carpeta que tuviéramos abierta si es que teníamos alguna):



Este repositorio ya tiene git iniciado es decir la carpeta 09-heroes ya venía con el directorio .git y commits realizados, podemos comprobarlo haciendo `git lg`:

```
PS D:\Git\09-heroes> git lg
* c3e22b9 - (4 years, 7 months ago) Agregamos a Deadshot - Strider (HEAD -> master, tag: v2.0.0)
* 69e85f8 - (4 years, 7 months ago) Actualizaciones al readme - Strider
* 093e6ef - (4 years, 7 months ago) Update: Misiones completadas - Strider
* f4523dd - (4 years, 7 months ago) Agregamos el archivo de las misiones completadas - Strider
* 158ba9e - (4 years, 7 months ago) Se agrego a la liga: Volcán Negro - Strider
* 300c014 - (4 years, 7 months ago) Misiones nuevas agregadas - Strider
* acea380 - (4 years, 7 months ago) Actualización de las misiones - Strider
* 31efae8 - (4 years, 7 months ago) Retomando el trabajo que guarde en el stash - Strider
* 8239e26 - (4 years, 7 months ago) Cambios de emergencia en el README - Strider
* 7dd23f1 - (4 years, 7 months ago) Agregamos más misiones - Strider
* 5b0c87f - (4 years, 7 months ago) Modificaciones en el README de emergencia - Strider
* 5440fe5 - (4 years, 7 months ago) Resolviendo conflictos - Strider (tag: v1.0.0)
| \
| * 52c9666 - (4 years, 7 months ago) Modificamos las misiones - Strider
* | b936625 - (4 years, 7 months ago) Actualizamos las misiones MASTER - Strider
| /
* a4a9834 - (4 years, 7 months ago) Merge branch 'rama-villano' - Strider
| \
| * b3e02bc - (4 years, 7 months ago) Agregamos a doomsday - Strider
* | 4c24baa - (4 years, 7 months ago) Borre a los heroes que no son de DC - Strider
| /
* ad66a33 - (4 years, 7 months ago) Agregamos al flash reverso - Strider
* df6ed62 - (4 years, 7 months ago) Agregando villanos - Strider
```

Vamos a ir a GitHub y crear un repositorio:



## Create a new repository

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

Owner \*

alvarosafd

Repository name \*

/ liga-justicia

Great repository names are short and memorable. Need inspiration? How about [fictional-octo-carnival](#)?

Description (optional)

Este es el repositorio de mis planes con la liga de la justicia

☐ 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:

Skip this step if you're importing an existing repository.

☐ Add a README file

This is where you can write a long description for your project. [Learn more.](#)

☐ Add .gitignore

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

☐ Choose a license

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

Create repository

Le damos un nombre

que debe ser único  
dentro de nuestra  
cuenta.

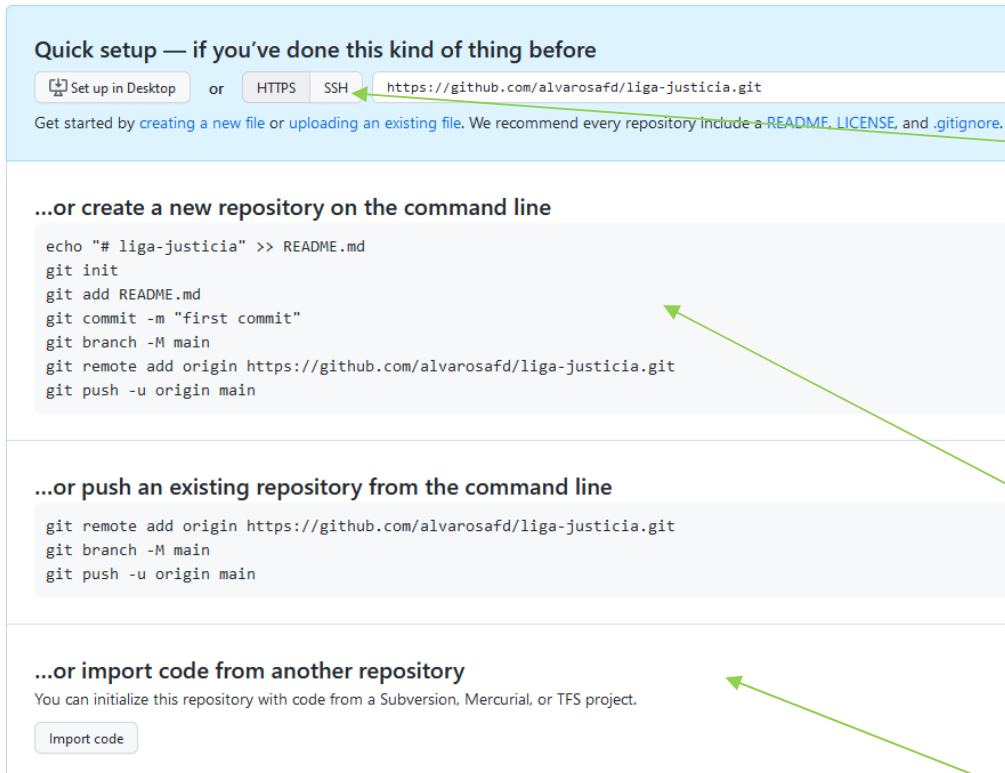
Rellenamos la  
descripción del  
repositorio

Elegimos si lo  
queremos que sea  
público o privado

Podemos marcar que se creen ciertos archivos como un archivo README o el archivo .gitignore, en nuestro caso estos dos archivos ya existen en nuestro repositorio local que queremos subir a GitHub, así que no marcamos la opción.

Si elegimos que el repositorio sea público cualquiera podrá verlo y hacer una copia, pero no modificarlo salvo que nosotros lo permitamos.

Una vez le damos al botón de crear el repositorio GitHub nos muestra opciones para rellenarlo con archivos:



The screenshot shows the GitHub 'Quick setup' page. It has three main sections: 'Quick setup — if you've done this kind of thing before', '...or create a new repository on the command line', and '...or push an existing repository from the command line'. The first section has buttons for 'Set up in Desktop', 'HTTPS', and 'SSH', with a text input field containing 'https://github.com/alvarosafd/liga-justicia.git'. The second section shows a list of git commands to create a new repository. The third section shows git commands to push an existing repository. There are green arrows pointing from text on the right to specific parts of the screenshot: one points to the 'SSH' button, another points to the git commands in the second section, and a third points to the 'Import code' button in the third section.

Aquí podemos elegir si queremos usar HTTPS o SSH

Esta opción nos muestra los comandos para crear un nuevo repositorio local y subirlo a GitHub

...or import code from another repository  
You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

Por último también tenemos la opción de rellenar nuestro repositorio cogiendo código de otras fuentes.

Vamos a usar la segunda opción, la de subir un repositorio ya existente a GitHub. Copiaremos los tres comando que nos proporciona GitHub y los ejecutaremos desde nuestro equipo:

Añadimos a nuestro repositorio local un repositorio remoto, **add origin** está añadiendo el repositorio remoto y le está dando el nombre local de **origin** este es un nombre estándar que se suele dar al primer repositorio remoto que añadimos pero podríamos ponerle otro.

```
git remote add origin https://github.com/alvarosafd/liga-justicia.git
```

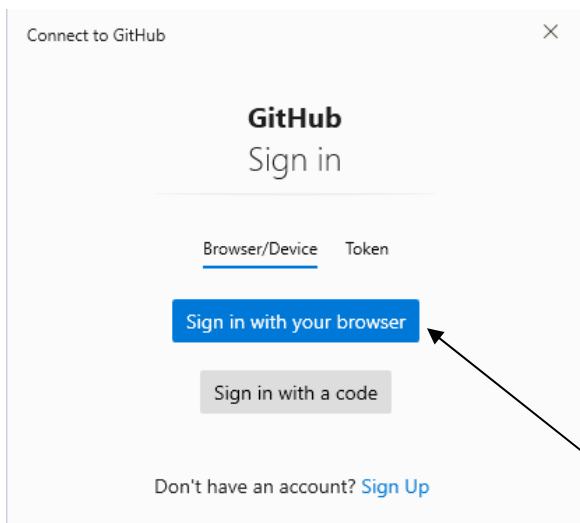
A continuación cambiamos el nombre de la rama principal para que se llame main

```
git branch -M main
```

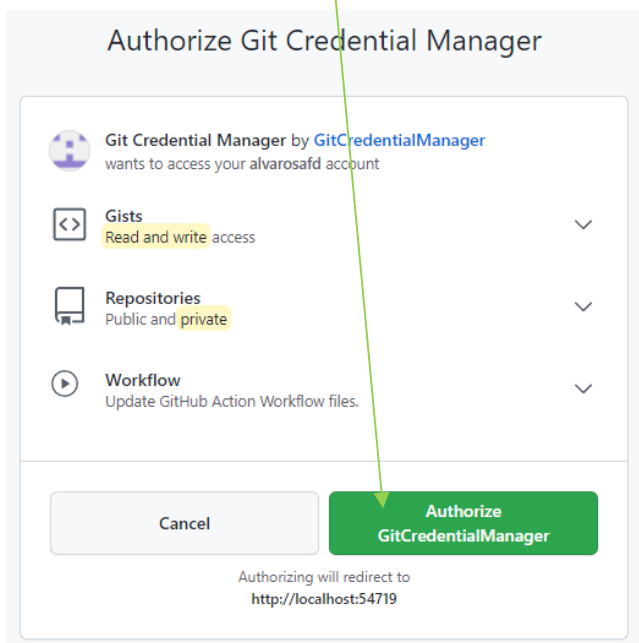
Hacemos el push del repositorio (es decir lo subimos a GitHub), la opción -u hace que a partir de ahora no tengamos que indicar el nombre del repositorio al que vamos a hacer el push ya que estableceremos que será por defecto el que indiquemos en este comando. En este caso es el repositorio origin

```
git push -u origin main
```

Al ejecutar el último comando nos pedirá que nos identifiquemos en GitHub:

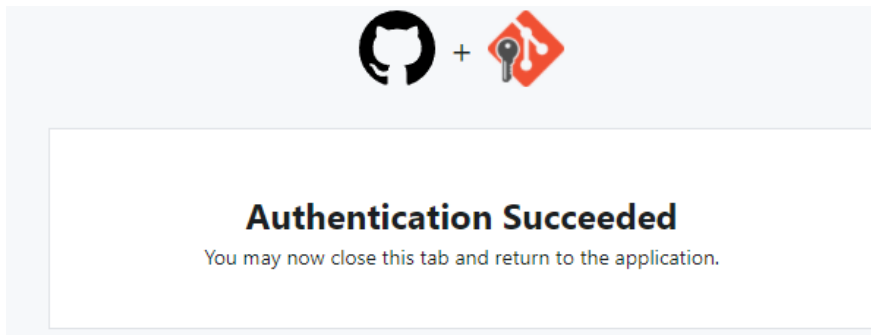


Una vez que nos identifiquemos por ejemplo con la opción de usar el navegador, nos pedirá que demos permisos de acceso:



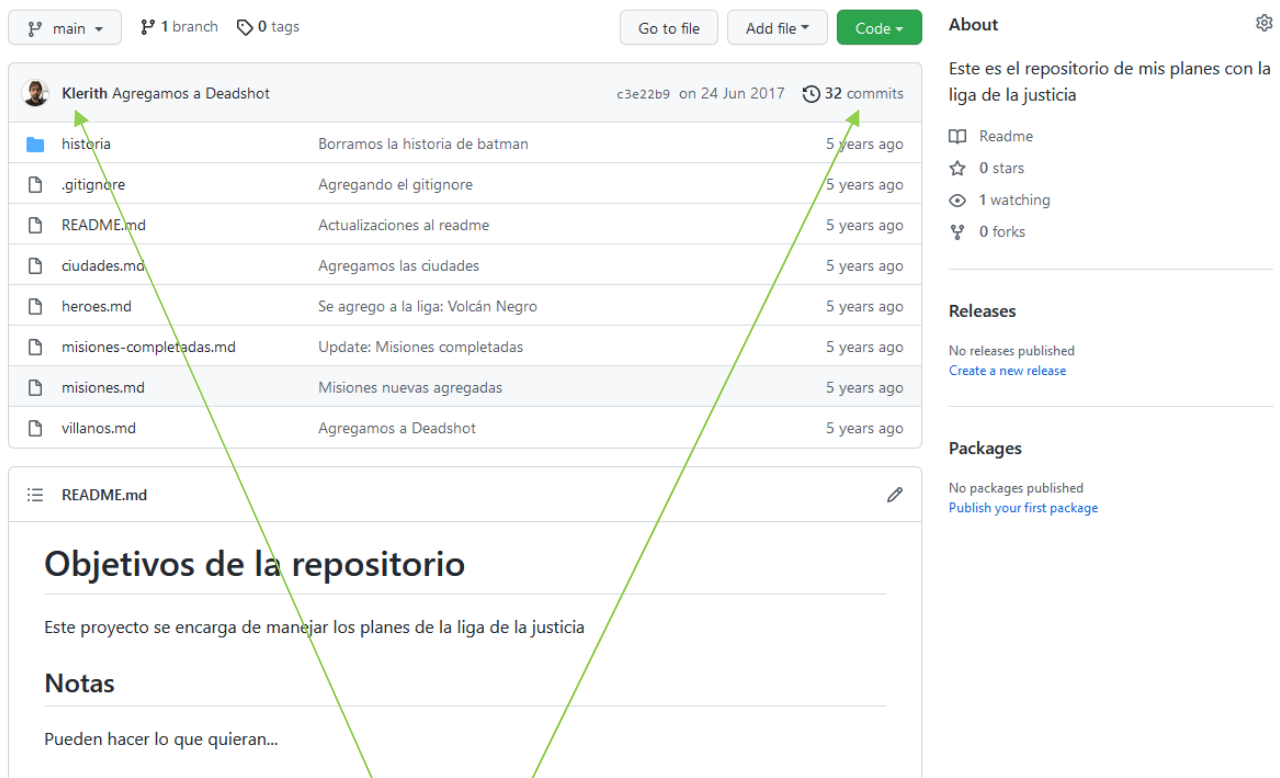


Una vez que le demos al botón de autorizar nos mostrará el siguiente mensaje si todo ha ido bien:



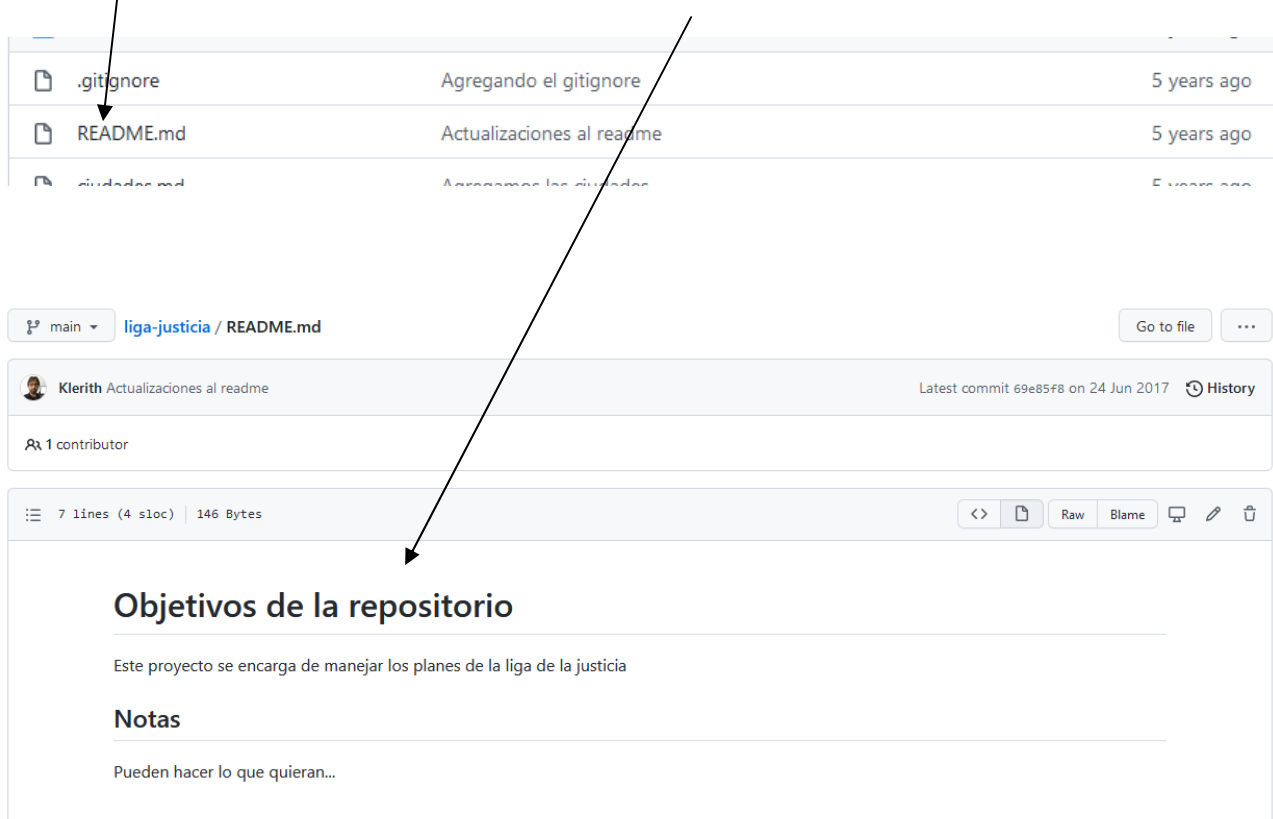
Nos indica que se realizó con éxito la autenticación y que podemos cerrar esta pestaña.

Si ahora vamos a ver nuestro repositorio en GitHub veremos que se ha subido el contenido que teníamos en nuestro repositorio local:

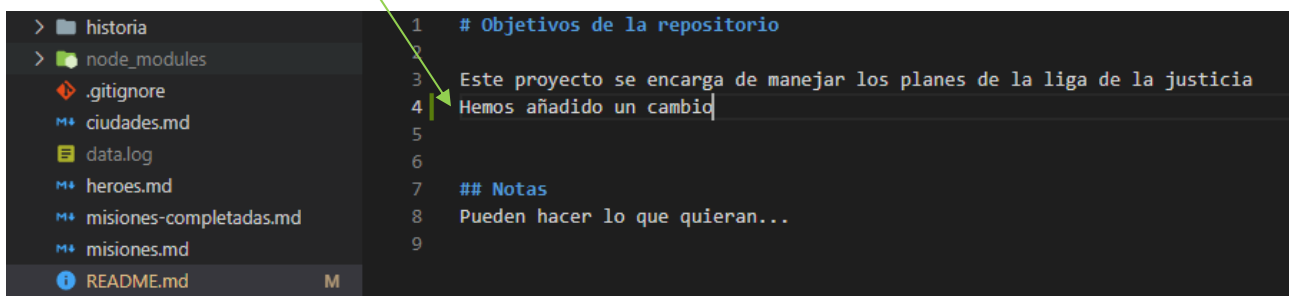


Como vemos git identifica al usuario creador de los archivos del repositorio y además nos indica que este repositorio tiene 32 commits.

Si pinchamos en cualquier archivo podemos ver el contenido:



Vamos a hacer un cambio en el repositorio local y subirlo a GitHub. Editamos el archivo README.md



Hacemos commit de este cambio:

```
git commit -am "README.md moficado"
```

Y subimos el cambio a GitHub

```
git push
```

Como anteriormente usamos `git push -u origin main` dejamos establecido el repositorio remoto y por eso ahora no nos hace falta especificarlo de nuevo.

Si abrimos de nuevo en GitHub el fichero README.md:

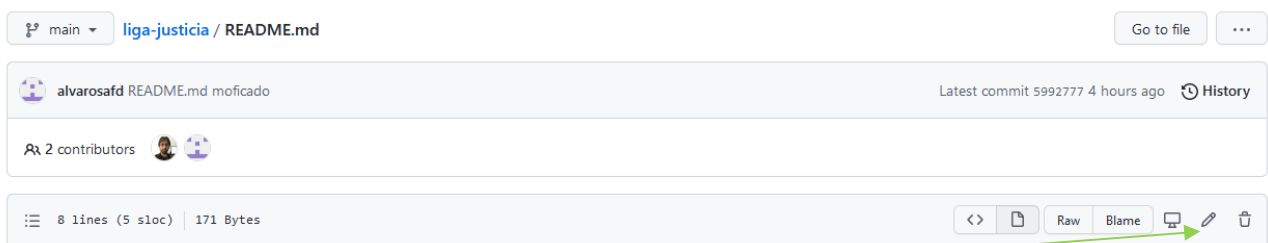


Vemos que en efecto el archivo se actualizó.

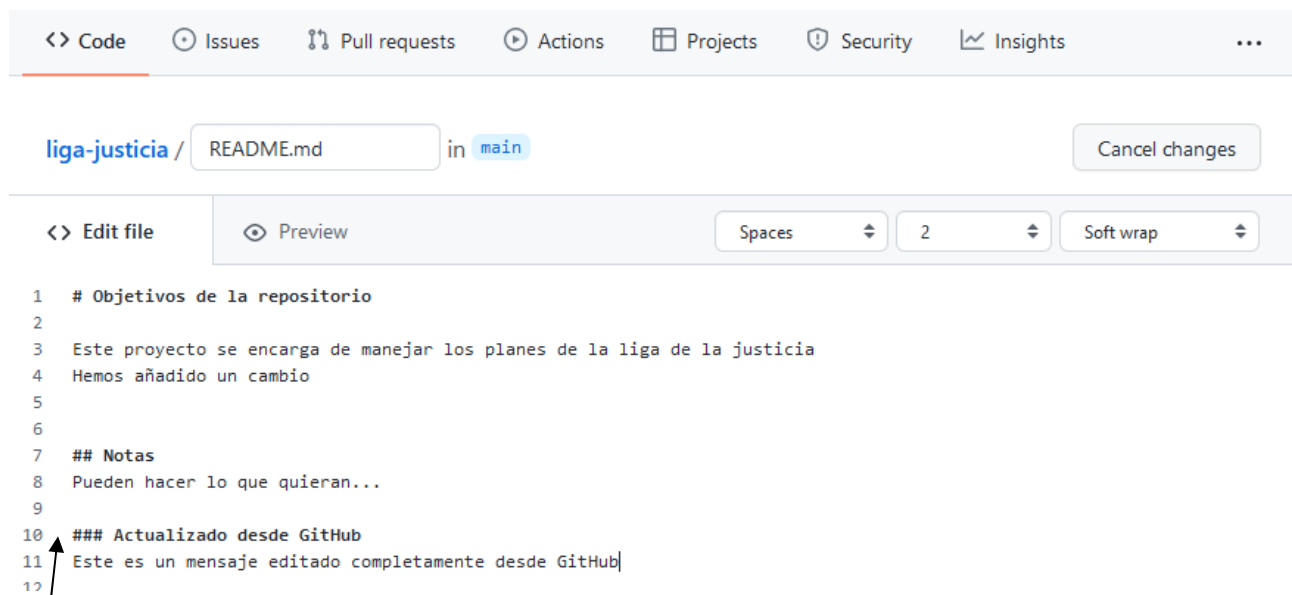
## 20 Comando pull

Desde GitHub podemos editar los archivos que se encuentran en el repositorio remoto. Para actualizar estos cambios en nuestro repositorio local usaremos el comando pull.

Vamos a modificar desde GitHub el archivo README.md



Al pulsar el botón de editar pasaremos al modo de edición:



The screenshot shows the GitHub web interface for editing a file. The file is named 'README.md' and is in the 'main' branch. The content of the file is as follows:

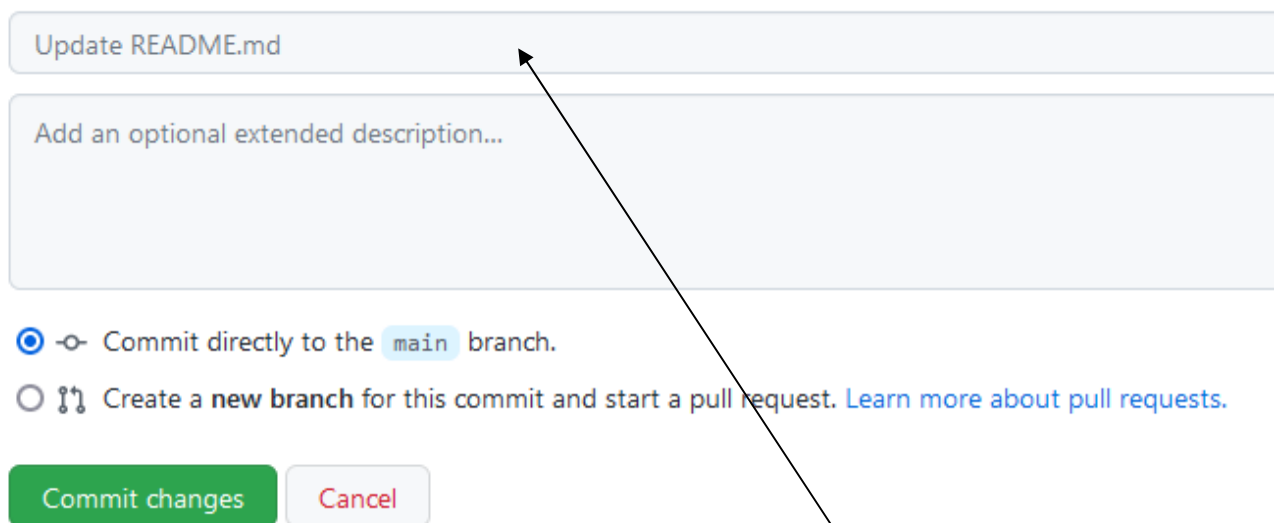
```
1 # Objetivos de la repositorio
2
3 Este proyecto se encarga de manejar los planes de la liga de la justicia
4 Hemos añadido un cambio
5
6
7 ## Notas
8 Pueden hacer lo que quieran...
9
10 ### Actualizado desde GitHub
11 Este es un mensaje editado completamente desde GitHub
12
```

An arrow points to line 10, which contains the text '### Actualizado desde GitHub'.

### Realizamos los cambios

Para guardar los cambios nos desplazamos hacia abajo en la página web:

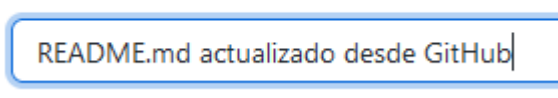
### Commit changes



The screenshot shows the 'Commit changes' form in GitHub. The form has a text input field with the default message 'Update README.md'. Below it is a larger text area for an optional extended description. There are two radio buttons: one selected for 'Commit directly to the main branch' and another for 'Create a new branch for this commit and start a pull request'. At the bottom are 'Commit changes' and 'Cancel' buttons. An arrow points from the text 'mensaje' in the paragraph below to the text input field.

Para guardar los cambios GitHub crea un nuevo commit con el mensaje que establezcamos. En este caso si no ponemos nada el mensaje será por defecto "Update README.md". Vamos a poner "README.md actualizado desde GitHub":

### Commit changes



The screenshot shows the 'Commit changes' form with the custom commit message 'README.md actualizado desde GitHub' entered in the text input field.

A continuación pulsamos el botón de Commit Changes:

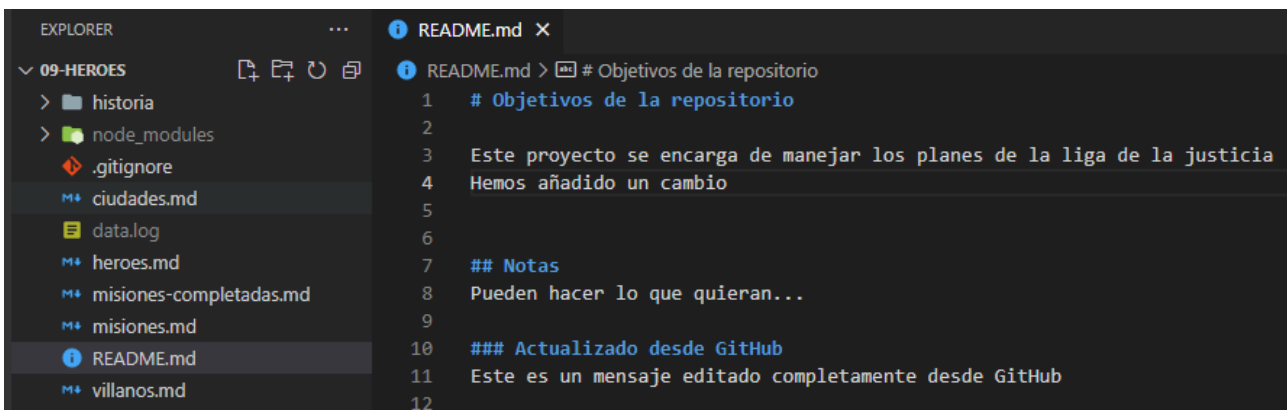


Para traer este cambio realizado en GitHub a nuestro repositorio local ejecutamos el siguiente comando desde nuestro repositorio local:

`git pull`

```
PS D:\Git\09-heroes> git pull
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 2), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 745 bytes | 16.00 KiB/s, done.
From https://github.com/alvarosafd/liga-justicia
 5992777..5f32173  main      -> origin/main
Updating 5992777..5f32173
Fast-forward
 README.md | 3 +++
 1 file changed, 3 insertions(+)
```

Si ahora editamos el archivo desde visual studio code veremos que se han añadido los cambios realizados desde GitHub:



Y si hacemos `git lg` veremos que se ha añadido el commit realizado desde GitHub:

```
PS D:\Git\09-heroes> git lg
* 5f32173 - (15 minutes ago) README.md actualizado desde GitHub - alvarosafd (HEAD -> main, origin/main)
* 5992777 - (4 hours ago) README.md modificado - Alvaro
* c3e22b9 - (4 years, 7 months ago) Agregamos a Deadshot - Strider (tag: v2.0.0)
* 69e85f8 - (4 years, 7 months ago) Actualizaciones al readme - Strider
* 093e6ef - (4 years, 7 months ago) Update: Misiones completadas - Strider
```