**TEMA 3**

Contenido

[INSTALACIÓN DE ACTIVITUS BAR 3](#_Toc95881343)

[INSTALACIÓN DE TOKYO NIGHT 3](#_Toc95881344)

[INSTALACIÓN DE ICONOS 4](#_Toc95881345)

[INSTALACIÓN DE TERMINAL 5](#_Toc95881346)

[INSTALACIÓN DE GIT 6](#_Toc95881347)

[COMANDOS DE PRUEBA EN GIT 9](#_Toc95881348)

[Git –versión 9](#_Toc95881349)

[Git help 9](#_Toc95881350)

[Git help commit 10](#_Toc95881351)

[Git config –global user.name “” 10](#_Toc95881352)

[Git config –global user.email “” 10](#_Toc95881353)

[Git config –global color.ui true 10](#_Toc95881354)

[Git config –global –e 10](#_Toc95881355)

[Git init 10](#_Toc95881356)

[Git status 11](#_Toc95881357)

[Git add index.html 11](#_Toc95881358)

[Git add . 11](#_Toc95881359)

[Git commit –m/ --message “” 11](#_Toc95881360)

[Git branch 11](#_Toc95881361)

[CARPETA 01 BASES 11](#_Toc95881362)

[Cambiar nombre de la raiz 13](#_Toc95881363)

[ARCHIVO README 14](#_Toc95881364)

[CONTROL SOURCE 15](#_Toc95881365)

[SUBIR ARCHIVOS DE FORMA ALTERNATIVA A GIT 16](#_Toc95881366)

[CREANDO ALIAS PARA LOS COMANDOS 21](#_Toc95881367)

[CAMBIOS EN LOS ARCHIVOS 21](#_Toc95881368)

[ACTUALIZAR MENSAJES DEL COMMIT 24](#_Toc95881369)

[BORRAR UN COMMIT 24](#_Toc95881370)

[CREAR UN REPOSITORIO PARA VIAJES EN EL TIEMPO 25](#_Toc95881371)

[VIAJES EN EL TIEMPO, RESETS Y REFLOGS 29](#_Toc95881372)

[CAMBIAR NOMBRE Y ELIMINAR ARCHIVOS CON GIT 32](#_Toc95881373)

[CAMBIAR EL NOMBRE Y ELIMINAR ARCHIVOS FUERA DEL GIT 34](#_Toc95881374)

[RENOMBRAR 34](#_Toc95881375)

[BORRAR ARCHIVOS 37](#_Toc95881376)

[IGNORAR ARCHIVOS 38](#_Toc95881377)

[RAMAS 39](#_Toc95881378)

[MERGE FAST-FORWARD 39](#_Toc95881379)

[MERGE UNIÓN AUTOMÁTICA 44](#_Toc95881380)

[MERGE MANUALES (CON CONFLICTOS) 48](#_Toc95881381)

[GIT HUB 51](#_Toc95881382)

[PUSH 52](#_Toc95881383)

[PULL 59](#_Toc95881384)

[EJERCICIOS PRÁCTICOS 61](#_Toc95881385)

[EJERCICIO 1 61](#_Toc95881386)

[EJERCICIO 2 61](#_Toc95881387)

[EJERCICIO 3 61](#_Toc95881388)

[EJERCICIO 4 62](#_Toc95881389)

[EJERCICIO 5 62](#_Toc95881390)

[EJERCICIO 6 62](#_Toc95881391)

[EJERCICIO 7 62](#_Toc95881392)

[EJERCICIO 8 62](#_Toc95881393)

[EJERCICIO 9 63](#_Toc95881394)

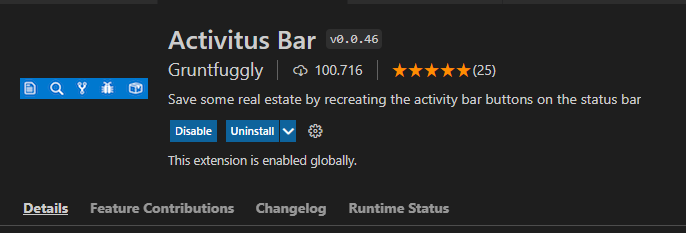
[EJERCICIO 10 63](#_Toc95881395)

[EJERCICIO 11 64](#_Toc95881396)

[EJERCICIO 12 64](#_Toc95881397)

# INSTALACIÓN DE ACTIVITUS BAR

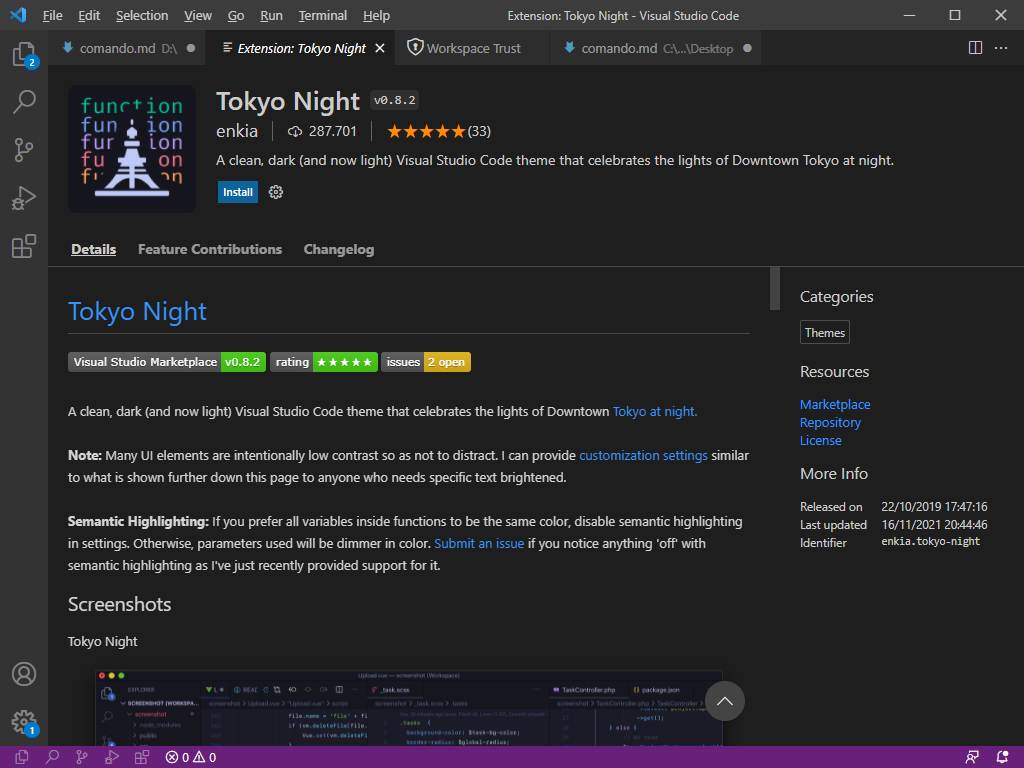
Vamos a instalar el activitus bar en el Visual Studio Code. Para hacerlo, abrimos el enlace en el documento, y le damos a Install.



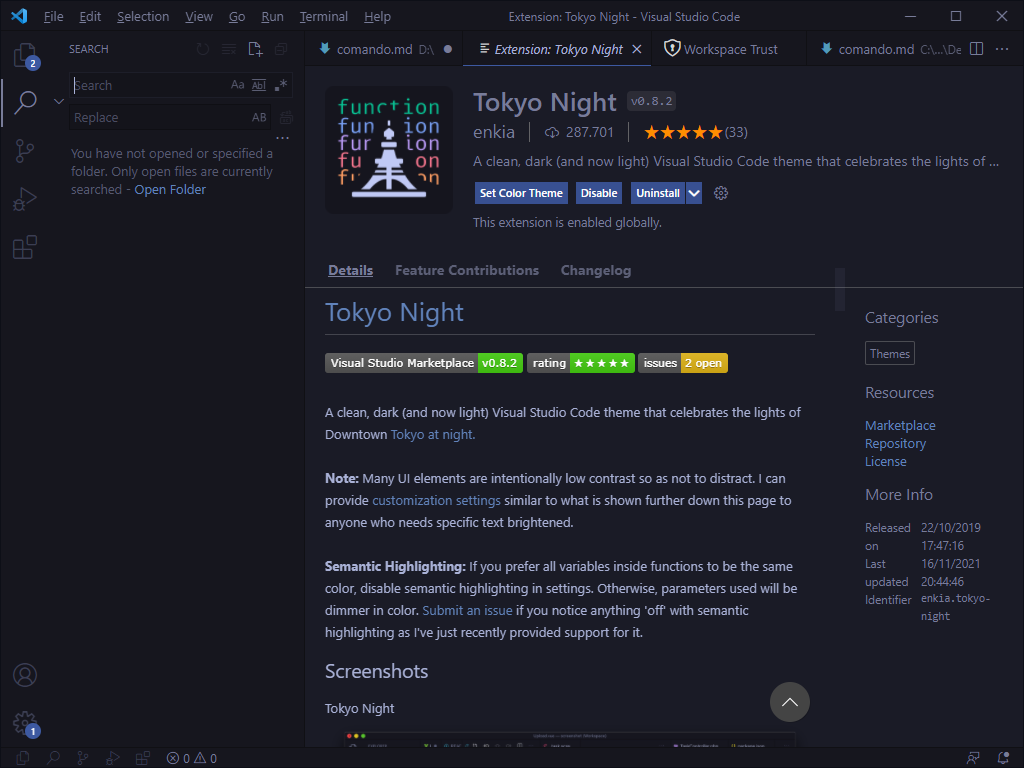
Como podemos comprobar, ya está instalado.

# INSTALACIÓN DE TOKYO NIGHT

Ahora instalaremos Tokyo Night. Le damos al enlace que aparece en el documento del tema, y le damos a Install.



Le damos, y nos instala automáticamente.

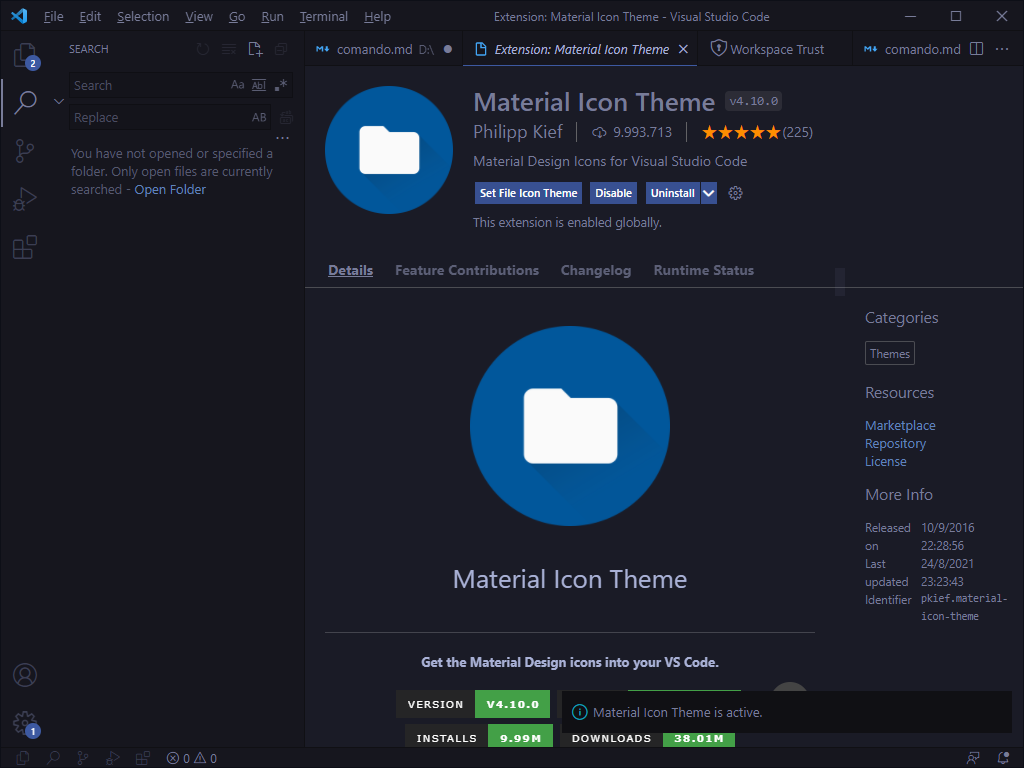


# INSTALACIÓN DE ICONOS

Vamos a instalar los Iconos. Seleccionamos el enlace de extensión de iconos en el documento del tema. Nos abrirá una página externa, y le damos a Install. Nos llevará a la extensión de Visual Studio Code, y le damos a Install.

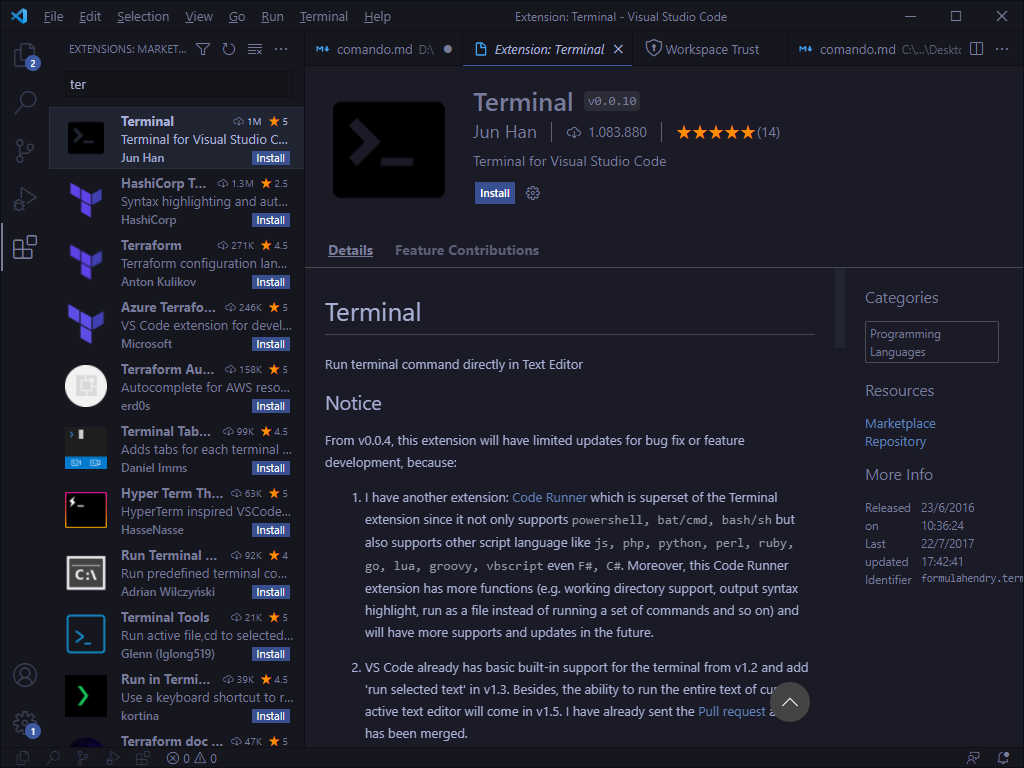


Ahora ya está instalado.



# INSTALACIÓN DE TERMINAL

Ahora, vamos a instalar la extensión de Terminal. En el apartado de la izquierda, le damos al dibujo de los cubos juntándose. Al darle a ese botón, nos aparece todas las extensiones de Visual Studio Code. Buscamos Terminal. Le damos a Install.

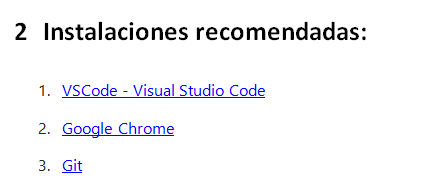


Ahora, ya está instalado.



# INSTALACIÓN DE GIT

Ahora vamos a instalr Git. Seleccionamos su enlace en el documento del tema para entrar en la página.



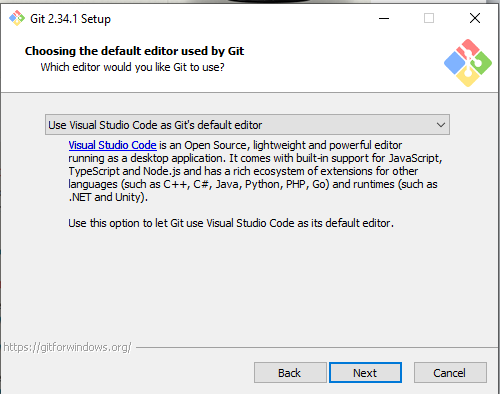
Al darle, al enlace, nos lleva a la página oficial de Git. Seleccionamos la instalación para Windows.



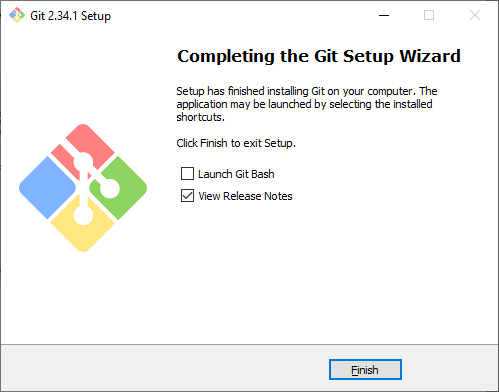
Seleccionamos la versión de 64 bits de Windows Setup.



Ahora, vamos a iniciar el instalador. Las opciones que aparezcan en la instalación las dejaremos por defecto, menos la opción del editor. Seleccionamos el Visual Studio Code.



Tras seleccionar todas las opciones, le damos a Install, y ya lo tendremos instalado.



# COMANDOS DE PRUEBA EN GIT

## Git –versión

Comprueba la versión actual del git.

Git help

## Git help

Nos muestra ayuda sobre los comandos

## Git help commit

Nos muestra ayuda sobre algún comando en concreto

## Git config –global user.name “”

Nos permite configurar el nombre del usuario

## Git config –global user.email “”

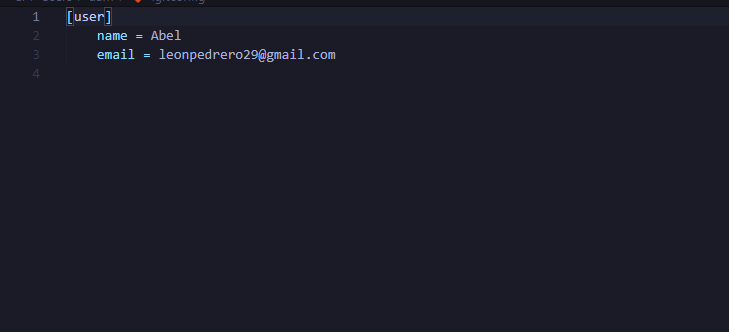
Nos permite configurar el nombre del email

## Git config –global color.ui true

Nos permite activar los colores de la interfaz

## Git config –global –e

Nos permite visualizar las configuraciones hechas hasta ahora.



## Git init

Crea el repositorio local.



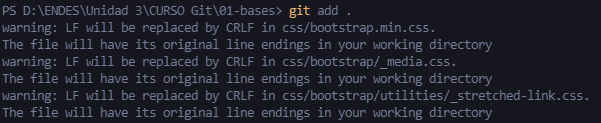
## Git status

Indica el estado del repositorio.

## Git add index.html

Añade el archivo index.html al repositorio

## Git add .



## Git commit –m/ --message “”

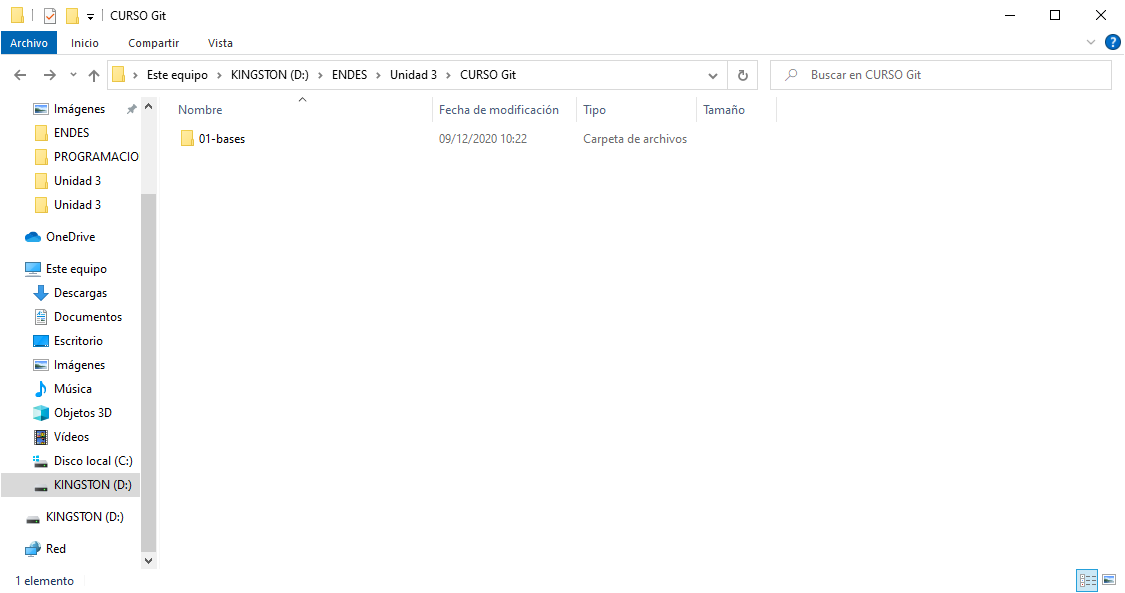
Nos permite hacer un commit y poner un mensaje para identificar a ese commit.

## Git branch

Sirve para saber en qué rama estamos

# CARPETA 01 BASES

Descargamos la carpeta 01 Bases de Teams, y lo ponemos en una carpeta cualquiera.



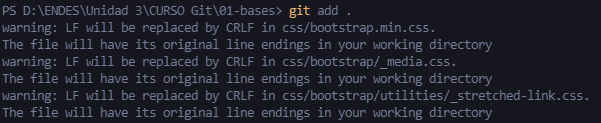
Ahora abrimos el git cmd, y nos ubicamos donde esté la carpeta 01 Bases.



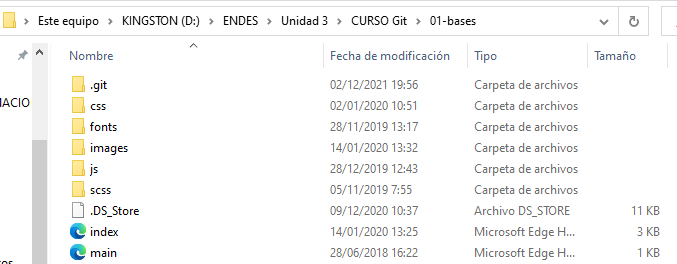
Ahora creamos el repositorio. Ponemos el comando git init.



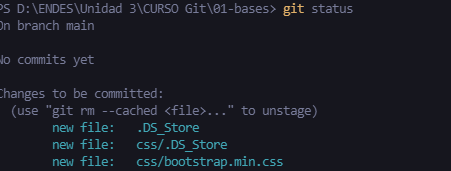
Ahora el repositorio estaría creado. Ponemos el comando add. Para que añada todos los archivos necesarios.



Volvemos a la carpeta, y comprobamos que se creó una carpeta oculta llamada .git



Ponemos el comando git status para actualizar la carpeta.



## Cambiar nombre de la raiz

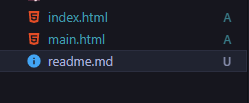
Ahora, vamos a cambiar el nombre de la rama. Ponemos el comando git branch –m master nombre (en nuestro caso, ponemos main)



Ahora borramos la carpeta .git y repetimos el proceso.

# ARCHIVO README

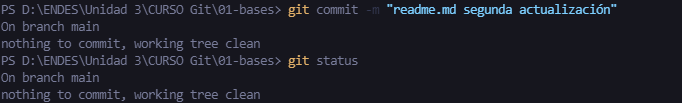
En Visual Studio Code, creamos un nuevo archivo llamado readme.md



Añadimos el archivo al stage con el comando add



Cambiamos un poco el mensaje del readme y le hacemos otro commit.

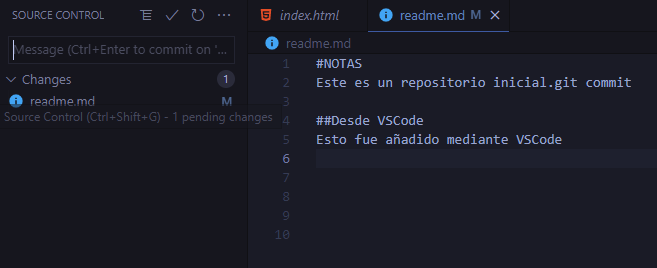


Si ponemos el comando git log, nos mostrará todos los cambios hechos hasta ahora. Aparecerá los commit hechos, y quén los hizo, y cuando.

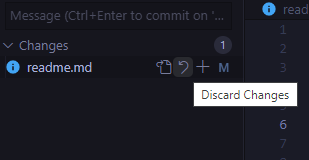


# CONTROL SOURCE

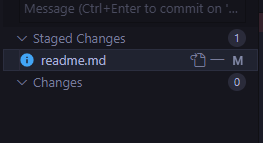
Cambiamos un poco el archivo y lo guardamos. Comprobamos el control source y vemos que el archivo readme aparece



Si ponemos el ratón sobre el archivo, haremos que el archivo descarte los cambios hechos anteriormente.



Si le damos al botón mas, haremos que se suba al escenario.



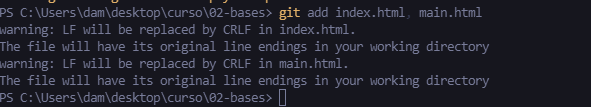
Si le damos al Control y luego al enter, haremos un commit del archivo.

# SUBIR ARCHIVOS DE FORMA ALTERNATIVA A GIT

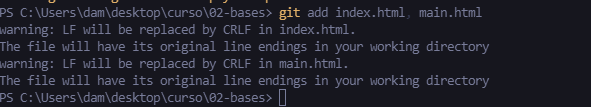
Primero descargamos la carpeta 02 bases. Inicicalizamos dentro el repositorio git.



Puedes subir 2 archivos al escenario a la vez. Para hacerlo se pone el comando git add nombre del archivo, nombre del archivo…



Para bajarlos, se hace lo mismo, salvo que se usa el comando git reset.

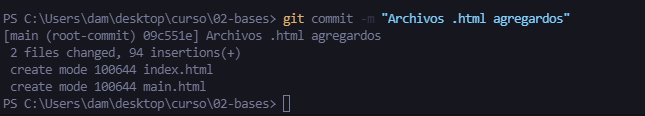


Ahora vamos a añadir todos los archivos html. Ponemos el comando git add, y ponemos un asterisco seguido de .html.

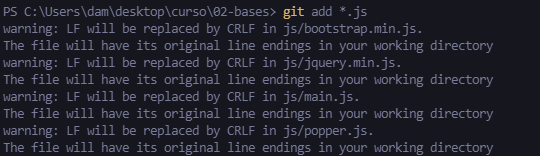
Texto

Descripción generada automáticamente

Vamos a hacer un commit con todo lo que hemos hecho hasta ahora. Ponemos el comando git commit –m. El –m sirve para que nos salte un mensaje a la hora de hacer el commit.



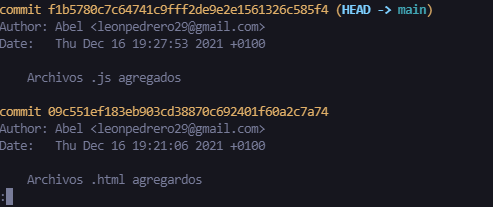
Ahora vamos a añadir todos los archivos .js



Y volvemos a hacer el commit.



Comprobamos todo lo hecho hasta ahora con el comando git log.

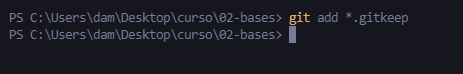


Creamos un archivo llamado gitkeep. Este archivo nos permite tener en cuenta las carpetas vacías a la hora de hacer commit.

Interfaz de usuario gráfica

Descripción generada automáticamente

Añadimos el archivo gitkeep al escenario.



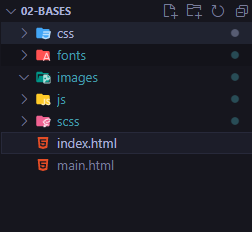
También podemos hacerlo de esta forma. Si lo ponemos así, sólo subirá ese archivo que haya en la carpeta que nosotros le indiquemos como ruta.



Se le hace un commit al .gitkeep.



Ahora borramos la carpeta uploads.



Ahora con el comando git checkout, restauramos la carpeta uploads.

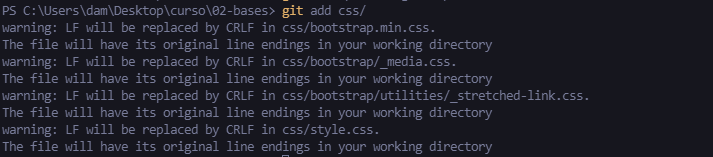


Y la carpeta uploads ha sido restaurada.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Vamos a añadir al escenario los css. Ponemos el comando git add y luego ponemos css/ para que suba la carpeta entera.



Le hacemos un commit.

Texto

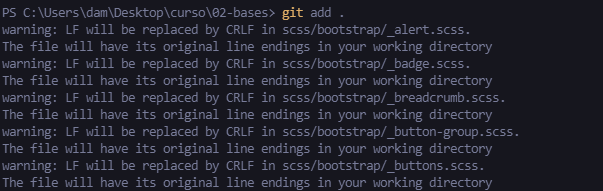
Descripción generada automáticamente

Con git status comprobamos lo que nos queda por subir al escenario.

Texto

Descripción generada automáticamente

Para subir todo lo que queda en la carpeta, ponemos el comando git add ..



Le hacemos otor commit y el cambio estaría guardado.



# CREANDO ALIAS PARA LOS COMANDOS

Para crear un alias a un comando, se pone el comando git config –global alias.(nombre del alias) (comando).



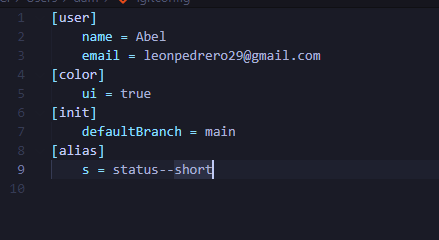
Para que funcione bien, tenemos que poner el comando entre comillas para que lo guarde.



Ahora vamos a comprobar los ajustes en los alias. Ponemos el comando git config –global –e.



Se puede comprobar todos los cambios hechos en los alias. También se pueden modificar los alias desde el archivo.





# CAMBIOS EN LOS ARCHIVOS

Creamos una carpeta llamada 03-Instalaciones y ponemos el comando git init.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Dentro de la carpeta creamos un archivo llamado instalaciones.md. Ponemos lo que hay en la siguiente captura:

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

y guardamos el archivo.

Lo añadimos al escenario con el comando git add.

Texto

Descripción generada automáticamente

Y le hacemos un commit.

Texto

Descripción generada automáticamente

Luego cambiamos el contenido del archivo.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Ponemos el comando git diff. Nos mostrará las diferencias entre las versiones del mismo archivo.

Texto

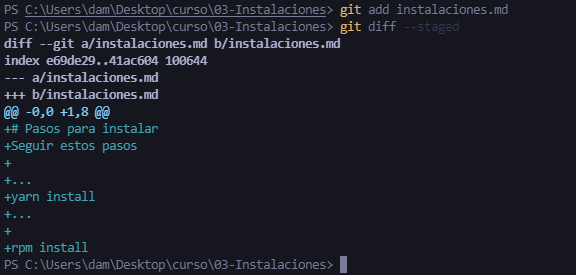
Descripción generada automáticamente

Ahora, volvemos a modificar el archivo.

Interfaz de usuario gráfica, Texto

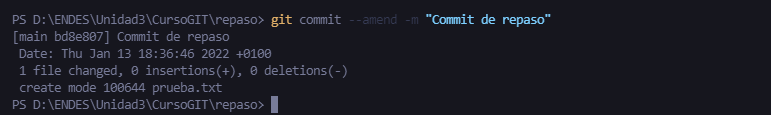
Descripción generada automáticamente

Lo guardamos y luego, lo subimos al stage. Luego lo vamos a comparar con el comando git diff –staged. Esto comparará la versión actual, con la versión que hay en el escenario.



# ACTUALIZAR MENSAJES DEL COMMIT

Es posible cambiar el mensaje del último commit que realizamos. Para ello, hay que poner el siguiente comando: git commit –amend –m “Mensaje”



# BORRAR UN COMMIT

Ahora vamos a eliminar un commit. Para hacerlo, vamos a insertar el siguiente comando: git reset –soft HEAD

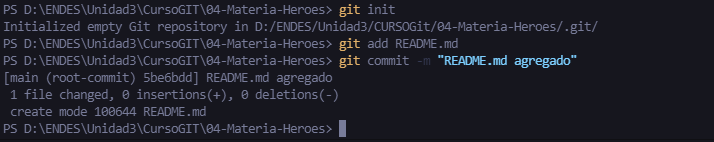


# CREAR UN REPOSITORIO PARA VIAJES EN EL TIEMPO

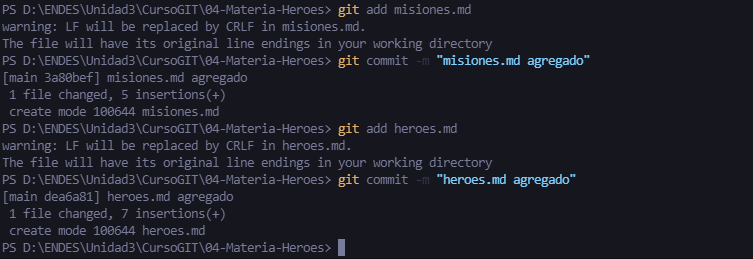
Para realizar este apartado, primero descargamos la carpeta comprimida Materia-Heroes. Sacamos la carpeta Materia-Heroes, y la metemos en la carpeta del curso de GIT, y la renombramos como 04-Materia-Heroes.



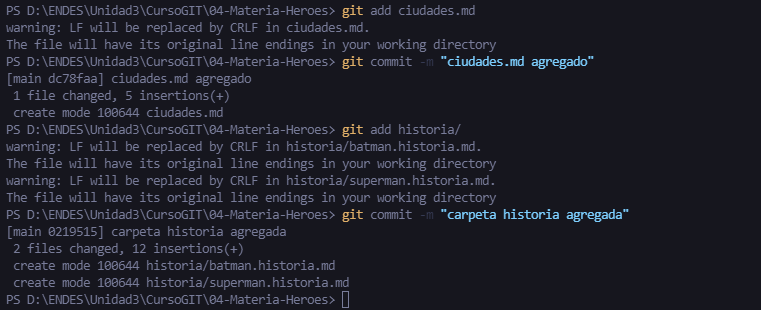
Para empezar, haremos lo siguiente: inicializamos el repositorio git en la carpeta, luego añadimos al stage el archivo README.md, y le hacemos un commit con el mensaje README.md agregado.



Ahora añadimos al stage el archivo misiones.md, y le hacemos un commit con el mensaje misiones.md agregado. Luego, repetimos este proceso con héroes.md. Lo agregamos al stage, y le hacemos un commit con el mensaje héroes.md agregado.,



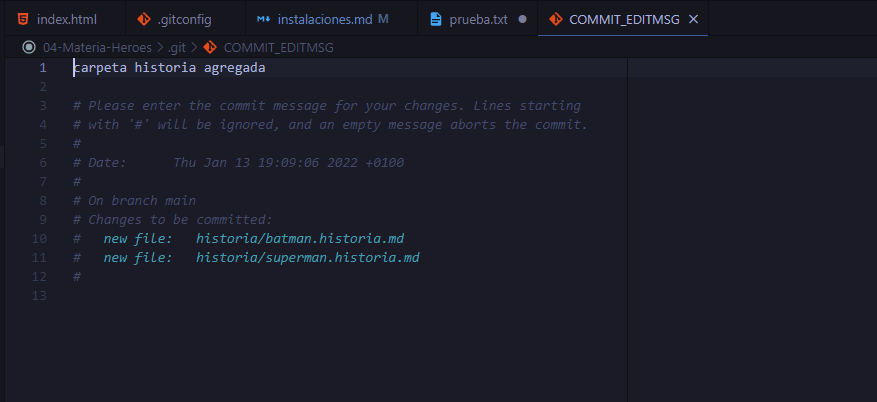
Ahora ñadimos el archivo ciudades.md al stage, y le hacemos commit con el mensaje ciudades.md agregado. Después, añadimos la carpeta historia al stage, y le hacemos un commit con el mensaje carpeta historia agregada.



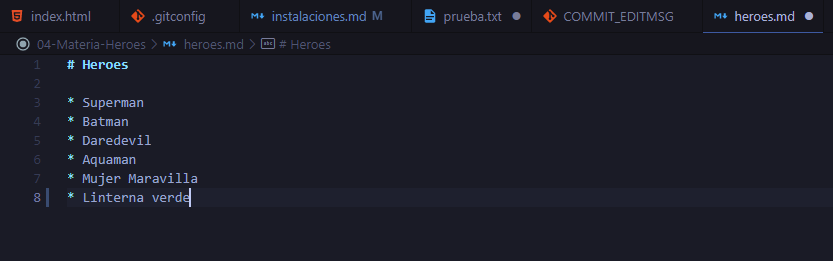
Vamos a cambiar el commit. Ponemos el comando git commit --amend



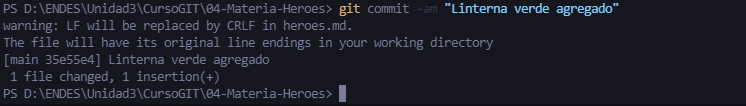
Al hacerlo, se nos abrirá un archivo para editar el texto del commit.



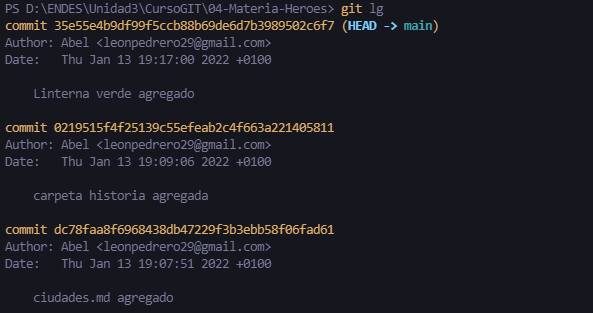
Ahora, en el archivo héroes.md, agregamos a Linterna verde.

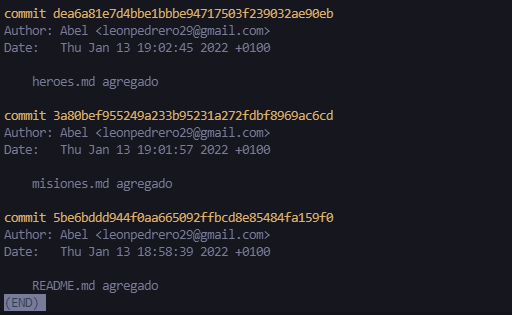


Guardamos los cambios en el archivo, y le hacemos un commit con el mensaje Linterna verde agregado.



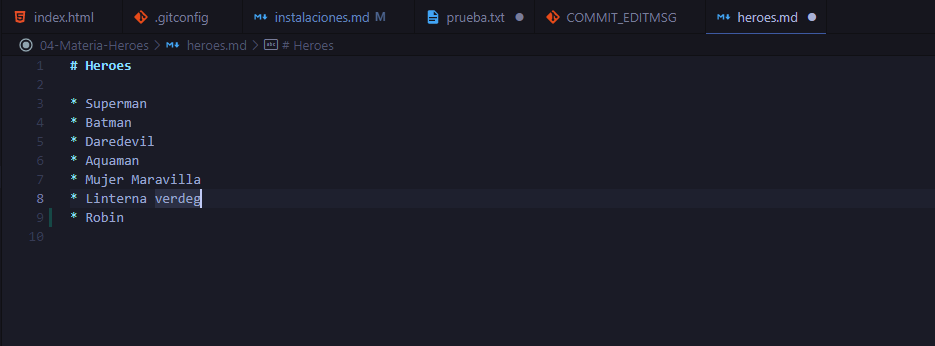
Ahora usamos el comando git lg para comprobar todos los commit que hemos hecho hasta ahora.



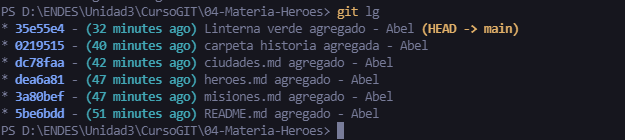


# VIAJES EN EL TIEMPO, RESETS Y REFLOGS

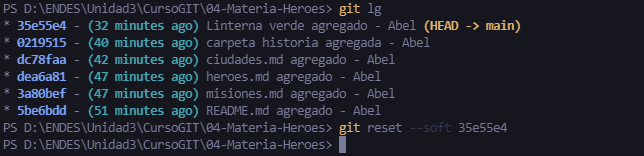
Empezamos añadiendo a Robin en el archivo héroes.md



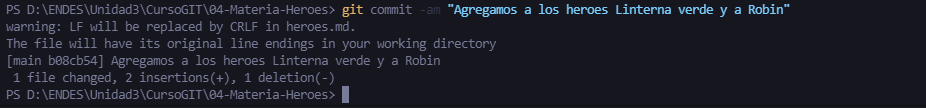
Ahora, le vamos a hacer un commit, pero queremos que nuestro commit nos indique quetanto Linterna verde como Robin hayan sido agregados. Hacemos un git lg para que nos muestre el commit de Linterna verde.



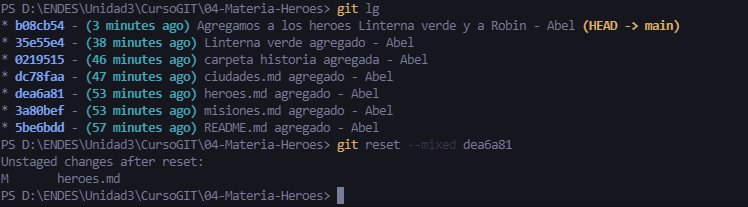
Copiamos su clave, y luego ejecutamos el siguiente comando: git reset –soft código del commit.



Hacemos un commit con el mensaje Agregamos a los héroes Linterna verde y a Robin.



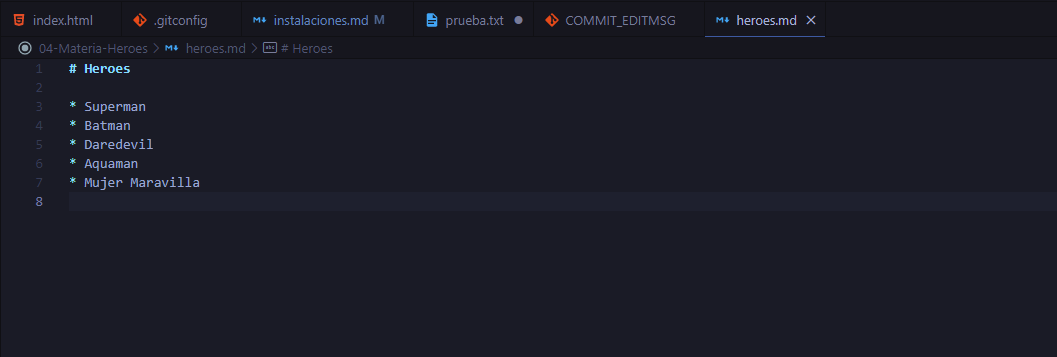
Ahora, vamos a volver al momento antes de agregar el archivo héroes.md, y vamos a eliminar los archivos del stage del commit que hubiésemos hecho de este archivo. Para ello, ponemos el comando git reset –mixed código del commit.



Ahora vamos a borrar los cambios hechos en el archivo de héroes. Ponemos el comando git reset –hard código del commit.



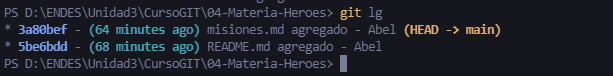
Como podemos ver, el archivo de héroes ha vuelto al estado en el que estaba.



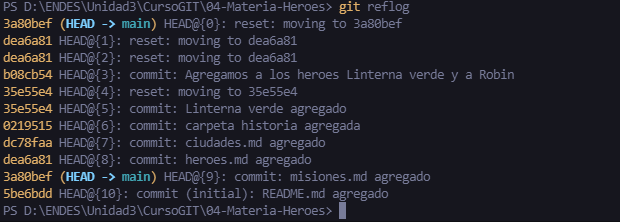
Ahora, hacemos otro reset hard, pero esta vez, desde el commit en el que agregamos el archivo misiones.md.



Usamos el comando git lg, y comprobamos que solo quedan dos commit.



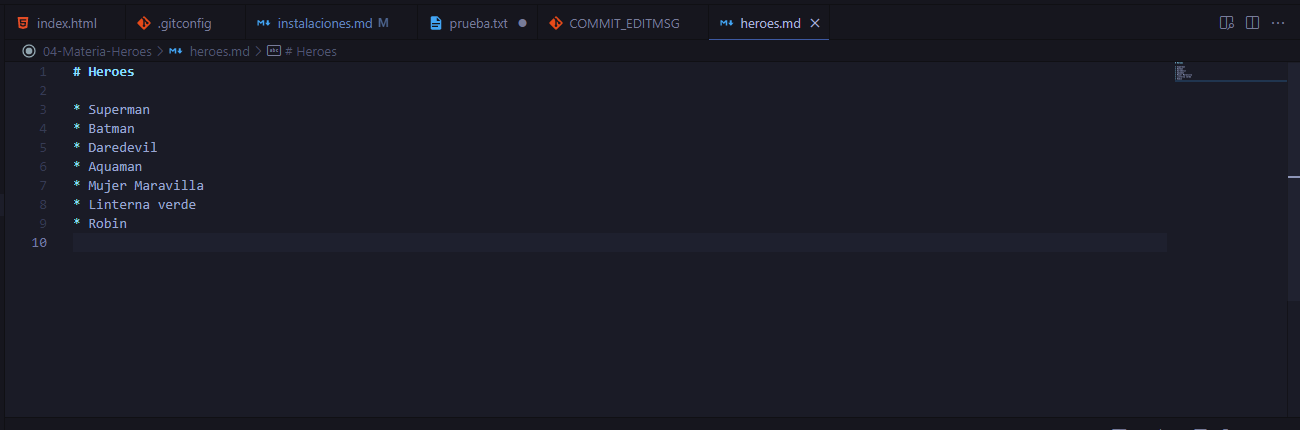
Sin embargo, si usamos el comado git reflog, nos aparece todos los commit, incluso los que ya habíamos eliminado.



Y si queremos volver al momento en el que agregamos a Linterna verde y Robin, hacemos reset –hard y luego, el código del commit.



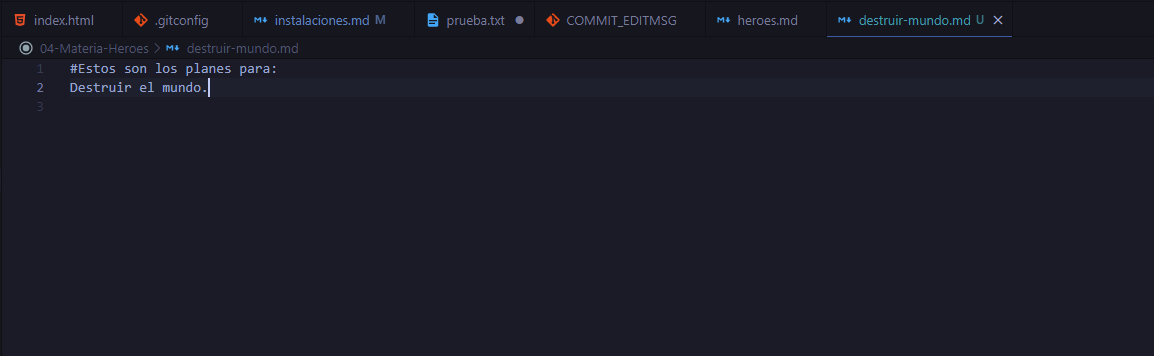
Todo volvió a como estaba desde ese momento.



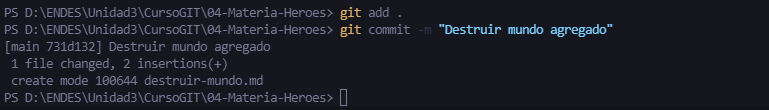


# CAMBIAR NOMBRE Y ELIMINAR ARCHIVOS CON GIT

Primero, creamos un archivo llamado destruir-mundo.md, y escribimos lo siguiente.



Ahora, lo añadimos al stage y le hacemos un commit con el mensaje Destruir mundo agregado.



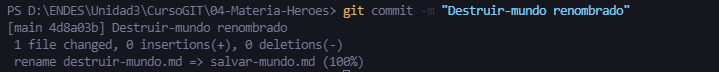
Vamos a cambiar el nombre del archivo. Para hacerlo, vamos a usar el comando mv. Lo ponemos de la siguiente forma: git mv nombre-actual nuevo-nombre.



Si usamos el comando git s, nos indica que el archivo destruir-mundo ha sido renombrado.



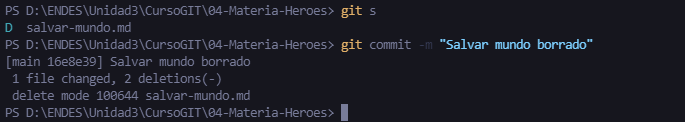
Le hacemos un commit con el mensaje Destruir-mundo renombrado.



Vamos a borrar el archivo salvar-mundo.md. Para hacerlo, vamos a usar el comando rm. Se pone así: git rm nombre-archivo



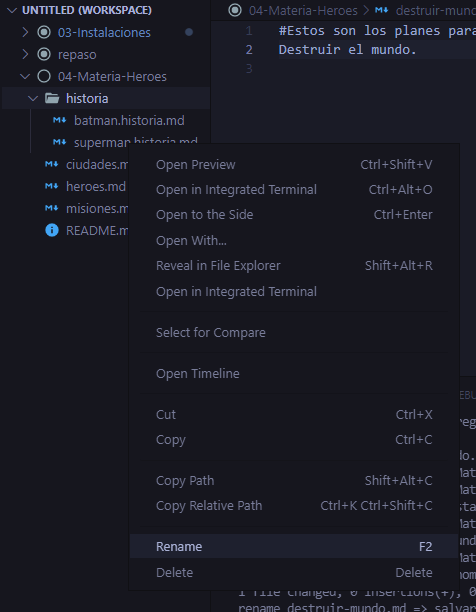
Después de borrarlo, si hacemos un git status (o git s), vemos que el archivo salvar-mundo está marcado con una d. Eso significa que está borrado. Hacemos un commit con el mensaje Salvar mundo borrado.



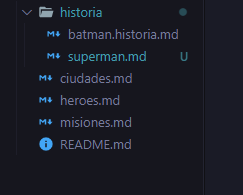
# CAMBIAR EL NOMBRE Y ELIMINAR ARCHIVOS FUERA DEL GIT

## RENOMBRAR

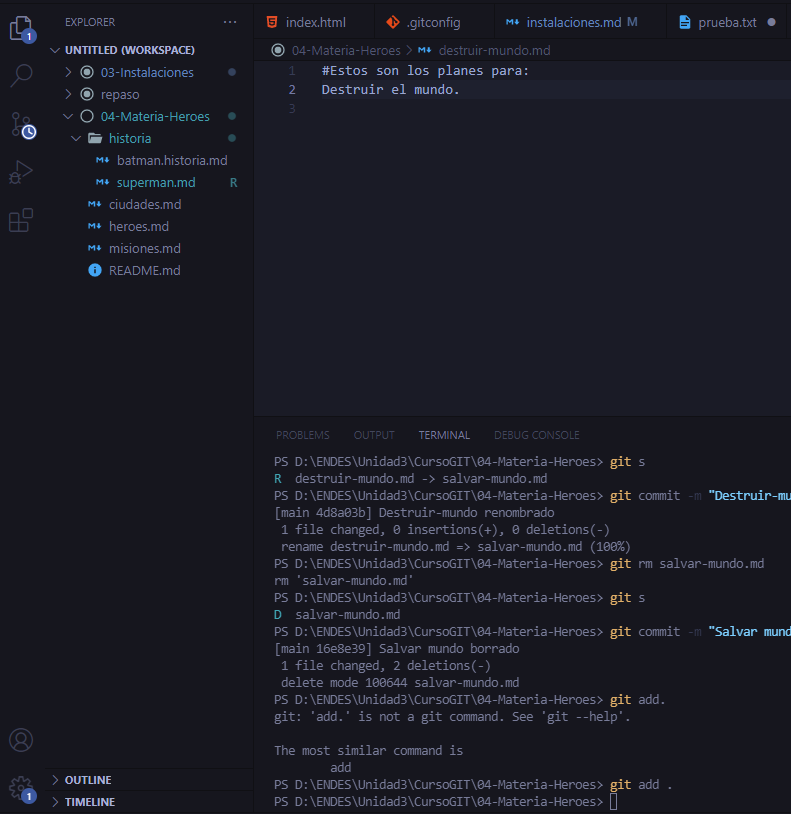
En la carpeta historia, seleccionamos el archivo de Superman, y con el botón derecho, le damos a Rename.



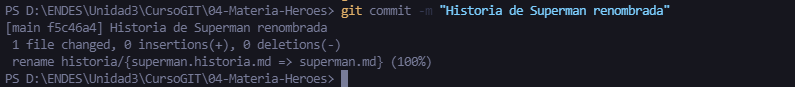
Lo renombramos como Superman.md



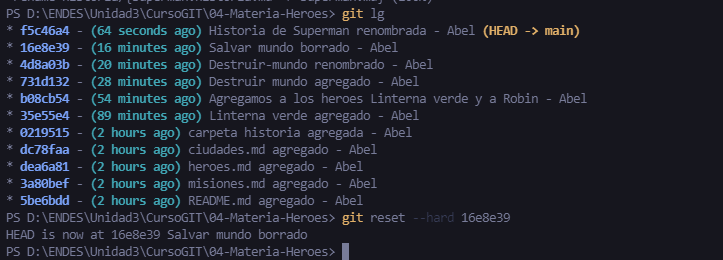
Le hacemos un git add. Al hacerlo, se nos queda marcado con una r. Significa que ha sido renombrado.

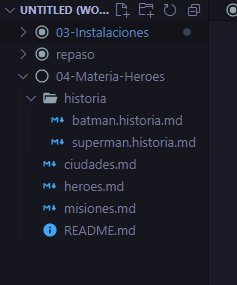


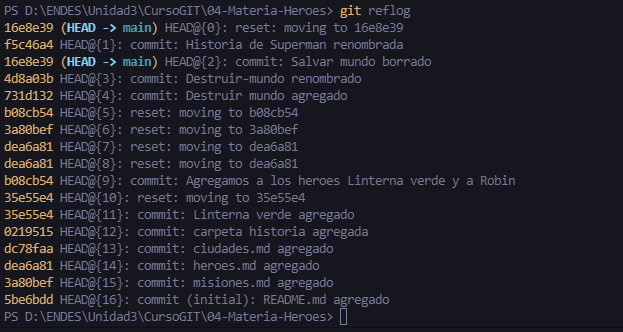
Hacemos un commit con el mensaje Historia de Superman renombrada.



Usamos el comado git lg, y luego hacemos un reset –hard al commit de Salvar-mundo borrado. Si lo hacemos el nombre del archivo de Superman, vuelve a como estaba antes.

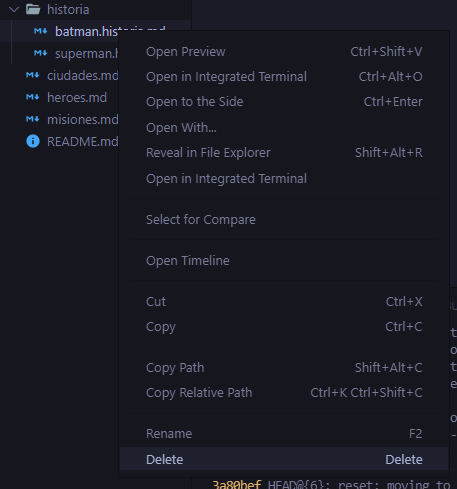




Si queremos deshacer el reset, podemos usar el comando git reflog para volver al commit del renombrado de Superman. 

## BORRAR ARCHIVOS

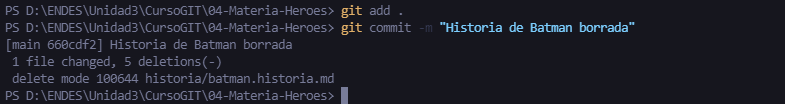
Con el botón derecho, borramos el archivo de Batman.



Si hacemos un git s, nos indicará el archivo con una d. Significa que está borrado.



Lo subimos al escenario, y luego hacemos commit con el mensaje Historia de Batman borrada.



# IGNORAR ARCHIVOS

Empezamos creando las carpetas dist, node\_modules y el archivo server.log en la carpeta de héroes. Luego, dentro de la carpeta dist, creamos el archivo index.html, y dentro de la carpeta node\_modules, creamos los archivos vue.vue, y react.jsx.

Pantalla de computadora con letras

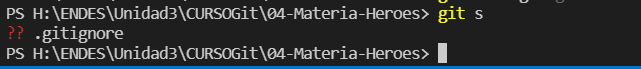
Descripción generada automáticamente con confianza media

Para poder ignorar estos archivos (no hacerles seguimiento), creamos un archivo llamado .gitignore, y dentro de ese archivo escribimos los nombres de todos los archivos y carpetas que queramos que ignoren.

Captura de pantalla de computadora

Descripción generada automáticamente

Si hacemos git s, nos mostrará que no puede hacerles seguimiento, porque los está ignorando.



# RAMAS

## MERGE FAST-FORWARD

Para empezar este apartado, primero descomprimimos la carpeta comprimida en Teams 06-Demos, y la metemos en nuestra carpeta de Curso Git. Luego, la abrimos en Visual Studio.

Imagen de la pantalla de un celular con letras

Descripción generada automáticamente con confianza media

Esta carpeta ya tiene un repositorio git inicializado, y también tiene commits creados. Vamos a hacer un git lg para comprobar todos los commit hechos en esta carpeta.

Texto

Descripción generada automáticamente

Creamos un archivo llamado villanos.md.

Captura de pantalla de computadora

Descripción generada automáticamente

Vamos a crear una nueva rama. Para hacerlo, usaremos el comando git Branch, y se usará de esta forma: git Branch nombre\_rama



Ahora, con el comando git Branch, comprobaremos en la rama que estamos. Las ramas que estén marcadas en verde y con un asterisco significan que son las ramas en las que estamos ahora (en mi caso, estoy en la rama master).

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Con el comando git checkout, cambiamos la rama en la que queremos estar.

Interfaz de usuario gráfica

Descripción generada automáticamente con confianza media

Si hacemos git lg, vemos que está apuntando a la rama villanos, en vez de a la master.

Texto

Descripción generada automáticamente

Añadimos el archivo villanos al stage, y hacemos un commit con el mensaje: villanos agregados.

Texto

Descripción generada automáticamente

Ahora, añadimos a Flash Reverso al documento. Luego, hacemos un commit con el mensaje Villanos.md Flash Reverso añadido.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Si hacemos git lg, la rama master queda dos commit por detrás de la rama villanos.

Texto

Descripción generada automáticamente

Cambiamos a la rama master. Si lo hacemos, la pestaña de villanos quedará tachada. Eso significa que el archivo villanos no se encuentra en la rama master. Esto se deshará si volvemos a la rama villanos.

Interfaz de usuario gráfica, Texto

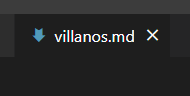
Descripción generada automáticamente

Nombre de la empresa

Descripción generada automáticamente con confianza media

Volvemos a cambiar a la rama villanos, y el archivo ya no estará tachado.



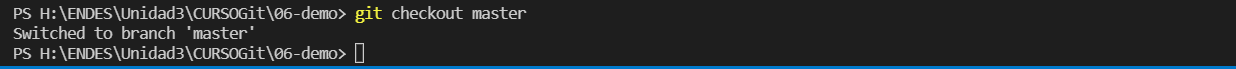


Ahora, vamos a unir las ramas. Primero, comprobamos en que rama estamos con el comando git Branch.

Texto

Descripción generada automáticamente

Ahora mismo, estamos en la rama villanos. Para hacer la unión, vamos a cambiarnos a la rama master.



Vamos a unir las ramas. Usaremos el comando git merge para poder unirlas.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente con confianza media

Ahora ambas ramas se han unido en una sola. Si hacemos git lg, podemos ver que las dos ramas se han unido después del merge.

Texto

Descripción generada automáticamente

Como ambas ramas ahora están unidas, vamos a borrar la rama villanos.



## MERGE UNIÓN AUTOMÁTICA

Vamos a empezar creando y moviéndonos a una rama nueva, y modificaremos el archivo de vllanos. Usamos el comando git checkout -b rama-villanos.



Añadimos a Doomsday al archivo, y hacemos un commit con el mensaje: VillanoDoomsday agregado.

Texto

Descripción generada automáticamente

Añadimos # Notas al archivo, y hacemos otro commit con el mensaje: Notas agregadas.

Texto

Descripción generada automáticamente

Ahora, hacemos un git lg. Si lo hacemos, comprobaremos que la rama master está unos dos commit por detrás de la rama villanos.

Texto

Descripción generada automáticamente

Nos vamos a la rama master con el comando git checkout, y nos vamos al archivo héroes. Vamos a borrar a Daredevil del archivo (porque él es un héroe de Marvel, y solo queremos héroes de dc).

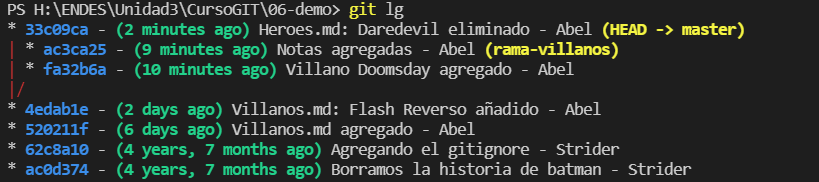


Borramos el nombre de Daredevil del archivo, y hacemos un commit con el mensaje: Heroes.md Daredevil eliminado.

Texto

Descripción generada automáticamente

Hacemos git lg. Podemos comprobar que la rama de villanos está un commit por detrás, además, se pueden ver 2 ramas diferentes.



Ahora vamos a unir las ramas. Primero hacemos git Branch para comprobar en que rama estamos. Luego usamos el comando git merge rama-villanos para unir la rama master con la rama de villanos.

Texto

Descripción generada automáticamente

Ahora, con las ramas unidas, vamos al archivo villanos. Deberían estar los 4 villanos que pusimos. Luego, si vamos al archivo héroes, deberían estar todos los héroes, excepto Daredevil.

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

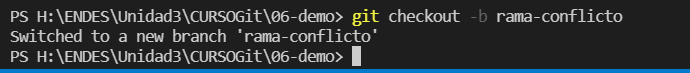
Si hacemos un git lg, vemos que ambas ramas se han unido.

Texto

Descripción generada automáticamente

## MERGE MANUALES (CON CONFLICTOS)

Empezamos el apartado creando una nueva rama llamada rama-conflictos.



Modificamos el archivo misiones, y lo guardamos.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Ahora, hacemos git Branch para comprobar la rama en la que estamos. Podemos ver que la rama-villanos todavía sigue existiendo. Como no la vamos a usar, la vamos a borrar.

Texto

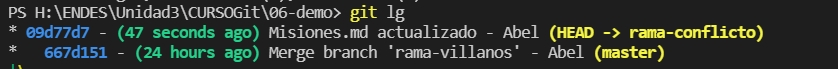
Descripción generada automáticamente

Hacemos un commit con el siguiente mensaje: Misiones.md actualizado.

Texto

Descripción generada automáticamente

Ahora hacemos un git lg para echar un vistazo al estado actual.



Volvemos a la rama master, y modificamos el archivo misiones añadiendo más frases y puntos.

Texto

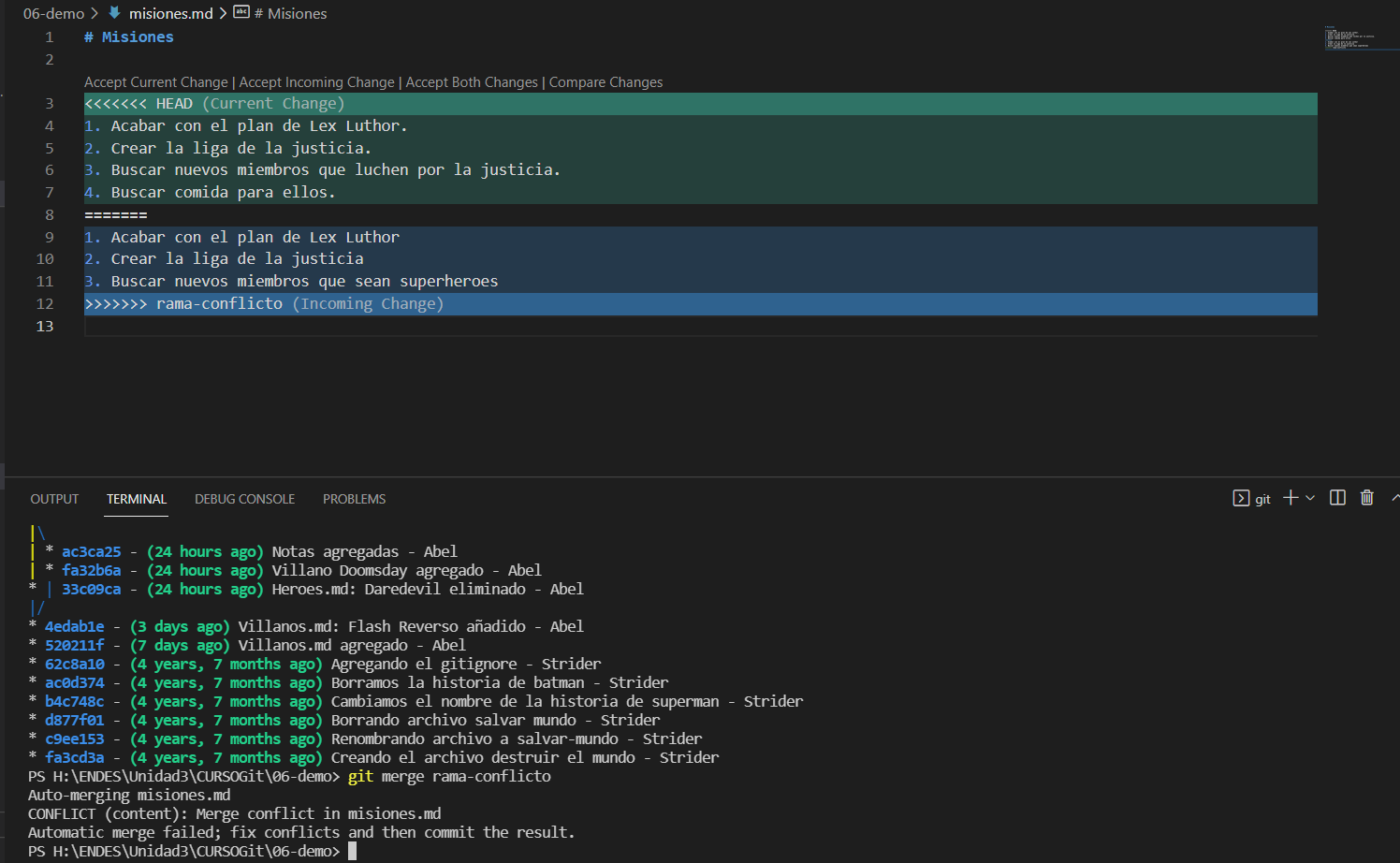
Descripción generada automáticamente

Hacemos un comit con el mensaje: Misiones.md actualizado en master. Luego, comprobamos su estado usando git lg.

Texto

Descripción generada automáticamente

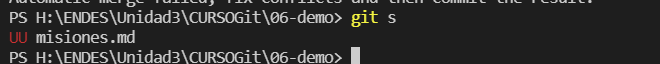
Ahora, vamos a unir las ramas. Al hacerlo, nos salta un error. Se debe a que el mismo archivo tiene modificaciones diferentes entre sí. Nos aparecerán los archivos de las dos ramas marcads en colores diferentes. La verde es la versión con la que trabajamos (master), y la azul, es la versión con la que se intenta fusionar (conflicto).



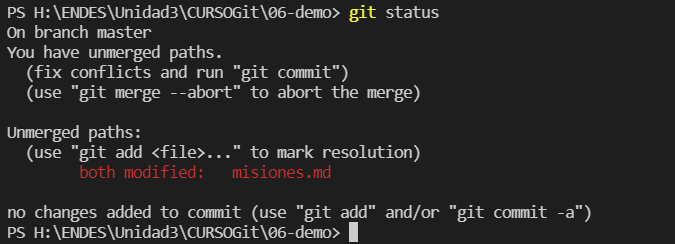
Vamos a igualar lo que pone en ambos archivos. Ambos tendrán escrito lo mismo que en la versión verde, pero se le quitarán los puntos.



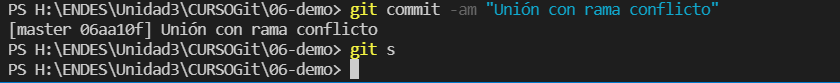
Guardamos los cambios, y hacemos un git s. Vemos que el archivo misiones está marcado con una UU. Significa que está en conflicto.



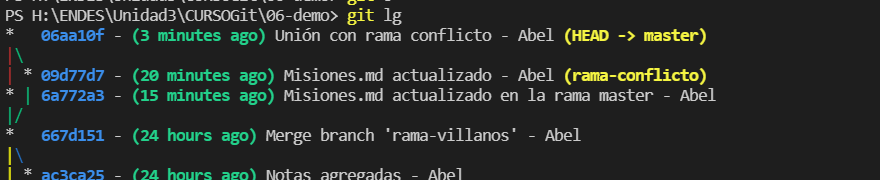
Si hacemos un git status, nos dará mas detalles del problema. Nos dirá que tenemos que resolver el conflicto, y después hacer un commit.



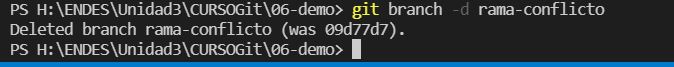
Hacemos un commit con el mensaje: Unión con rama conflicto. Luego, hacemos un git s para comprobar que no haya más conflictos.



Hacemos un git lg para comprobar que el commit se ha realizado con éxito.

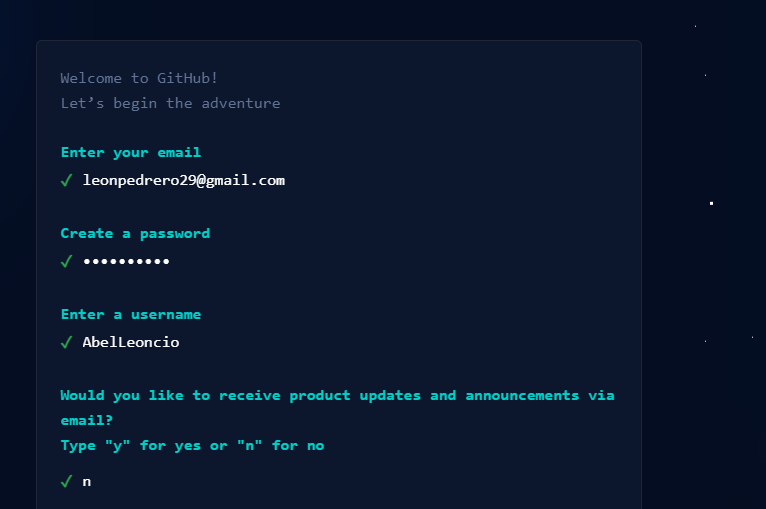


Para acabar, borramos la rama conflicto.



# GIT HUB

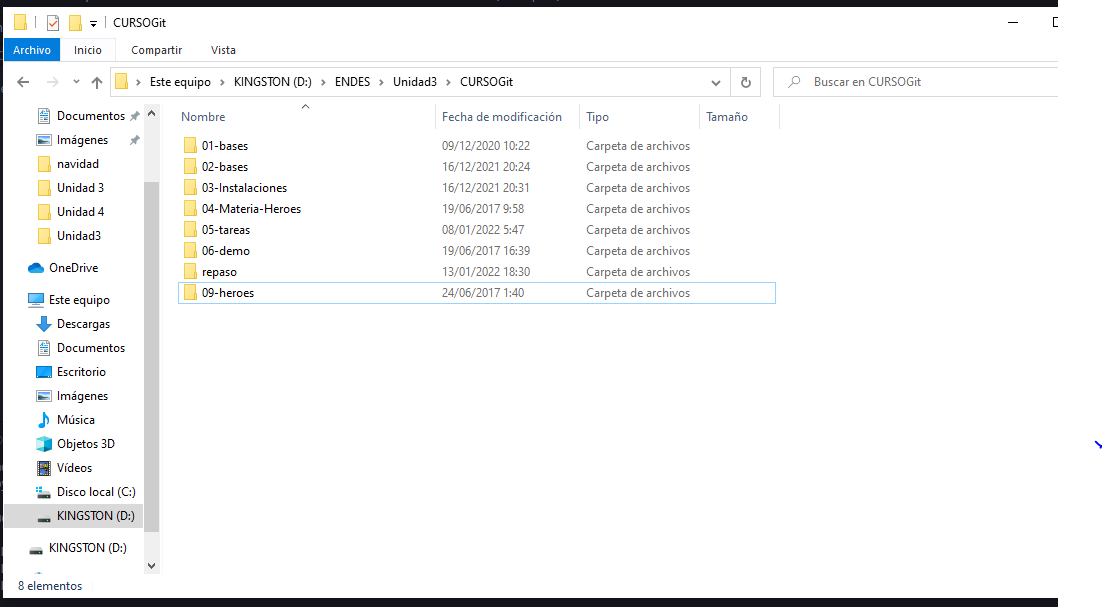
Creamos una cuenta en Git Hub para poder guardar nuestros repositorios.



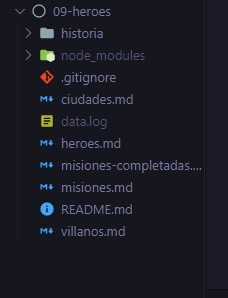
Ponemos nuestro email, una contraseña para la cuenta, y un nombre de usuario para registrarse (y también una autenticación para ver que no eres un robot).

## PUSH

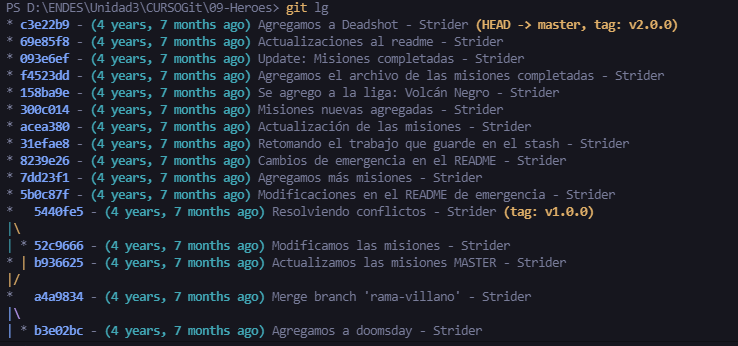
Descargamos y descomprimimos la carpeta de 09-Heroes, y la ponemos en nuestra carpeta de curso GIT.



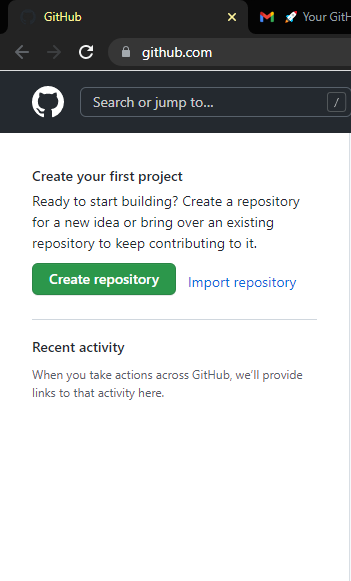
Añadimos la carpeta de héroes al Visual Studio.



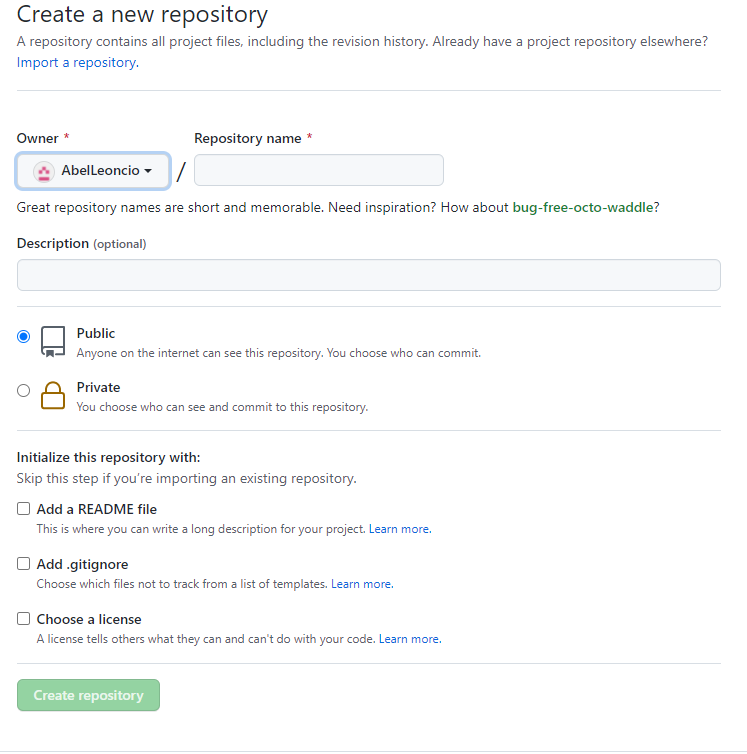
Si hacemos un git lg, vemos que esta carpeta ya tiene un repositorio git inicializado, y tiene commits hechos.



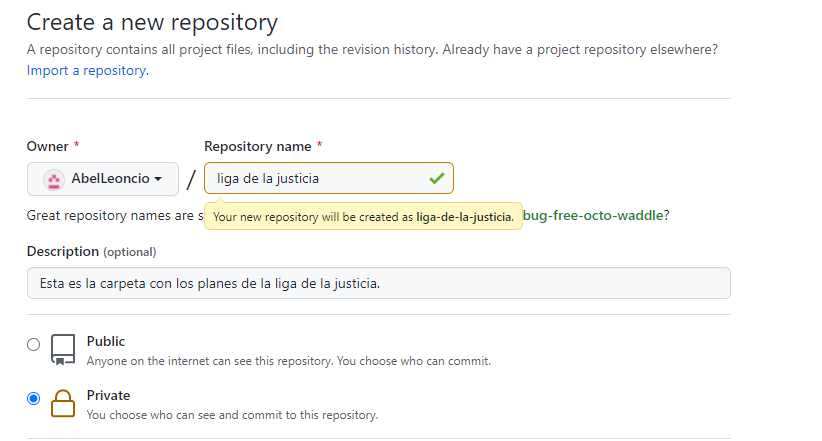
Ahora en Git Hub, le damos a Create a repository.



Al darle, nos lleva a un menú para poder añadir archivos en nuestro repositorio (en nuestro caso, no añadimos nada).



Le ponemos un nombre, y una descripción. También podemos elegir si queremos que sea público o privado. Le llamamos Liga de la justicia, y le ponemos una descripción, y seleccionamos que sea privado.



Después, la página nos crea unos archivos para el repositorio a nuestra elección.



Vamos a seleccionar la segunda opción. Esa opción nos permite subir un repositorio ya existente en Git Hub. Para ello usaremos tres comandos.



Primero añadiremos un repositorio remoto. Usamos el comando git remote add origin dirección\_del\_enlace.



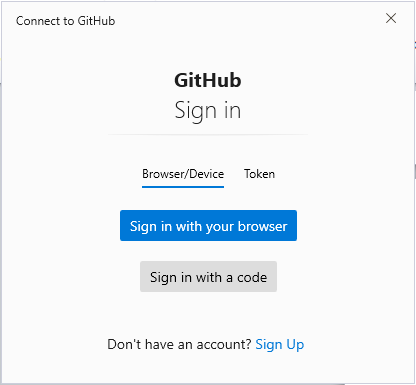
Ahora, cambiamos de rama a la main.



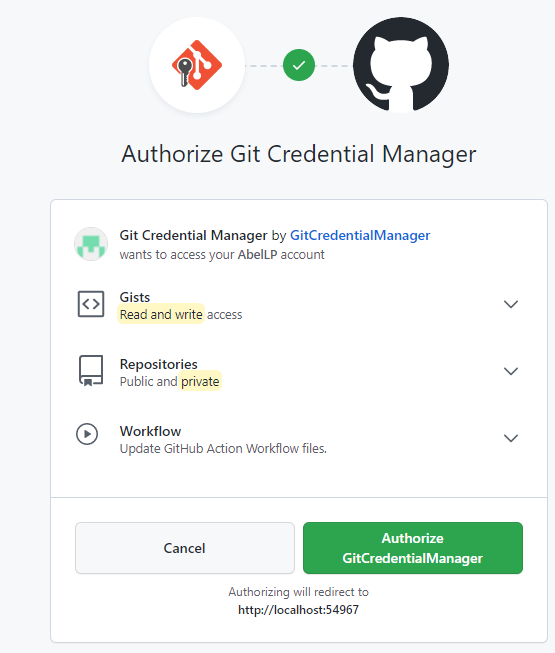
Luego usamos el comando git push –u origin main, para subirlo a Git Hub. Si le ponemos un –u, a partir de ahora, cada vez que hagamos un push, usará esta carpeta por defecto.



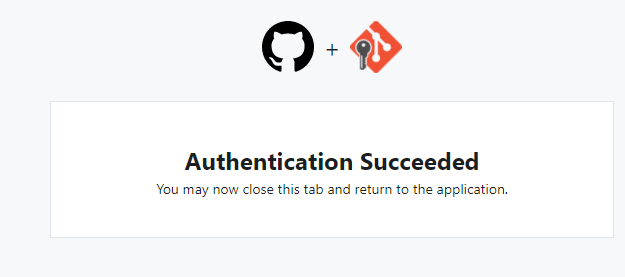
Después de insertar el comando, nos abrirá una página para abrir sesión en Git Hub.



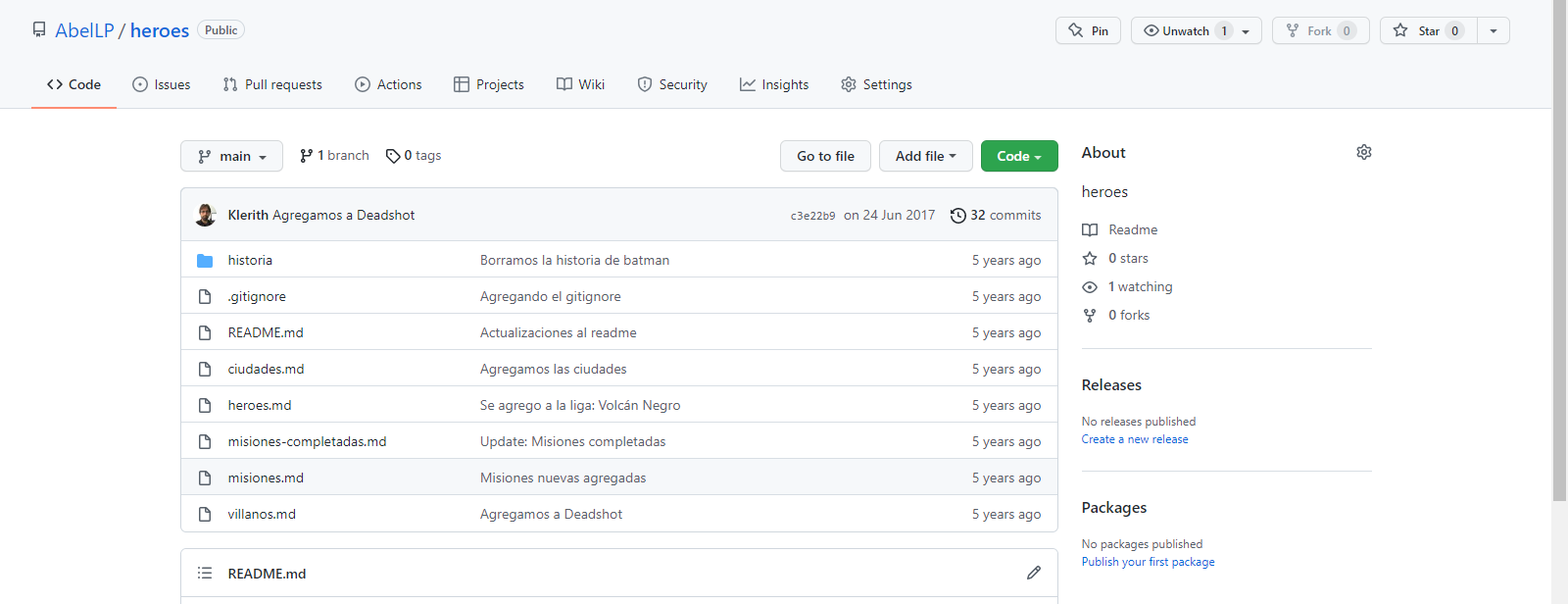
Si seleccionamos la primera vez (la opción de abrir sesión con el navegador), nos abrirá una página para poder autorizar las credenciales al Visual Studio.



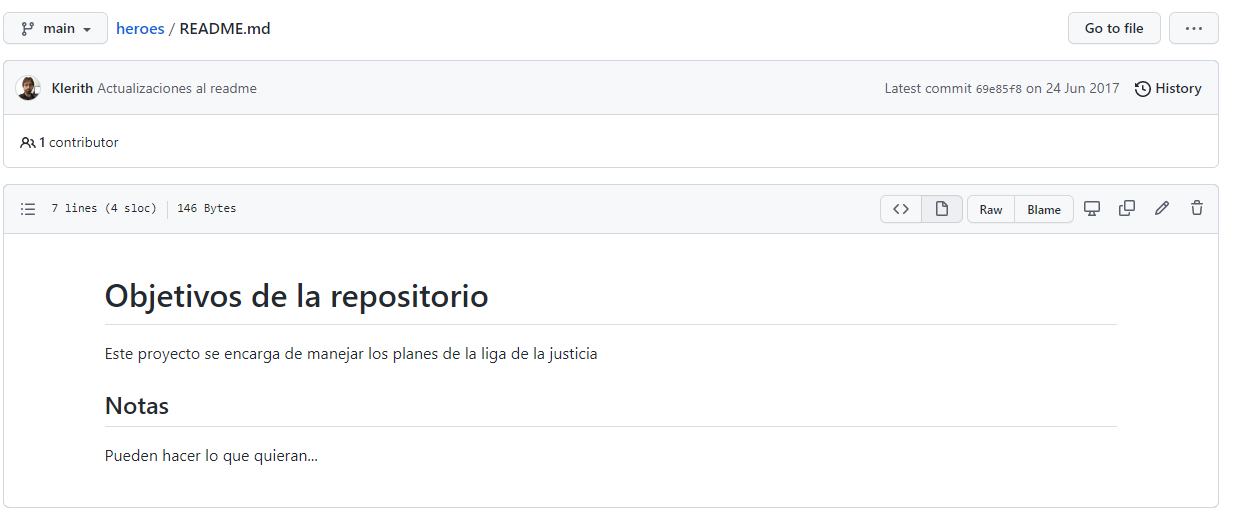
Nos pedirá la contraseña de la cuenta. Cuando la metamos, la autorización se habrá completado.



Tras autentificarte, nuestro repositorio estará subido a Git Hub.



Si pinchamos, por ejemplo en el archivo readme, nos aparece lo que hay escrito.



Ahora, hacemos una modificación al archivo readme.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Le hacemos un commit con el mensaje:Readme.md modificado. Luego hacemos un git push. Como antes, hicimos un git push -u main origin, nos deja por defecto el repositorio al que apuntamos antes.

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Si vamos a github, vemos que se han hecho los cambios en el redadme.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

# PULL

Empezamos este apartado entrando a la carpeta Heroes en github. Una vez allí, iremos al archivo readme.md y seleccionamos el dibujo del lápiz.

Patrón de fondo

Descripción generada automáticamente

Si le damos, nos llevará a una hoja de edición para editar el archivo. Ponemos una línea editada.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Abajo, en la misma página, se puede hacer commit en la página y le podemos poner el mensaje que queramos. Nosotros pondremos README.md actualizado desde GitHub.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Ya nos aparece el mensaje en el archivo.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

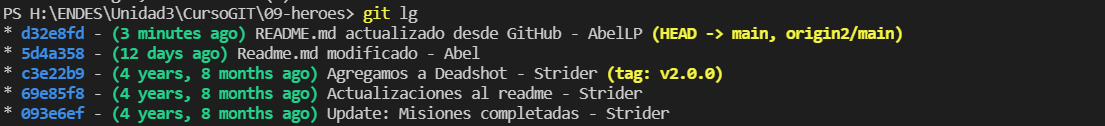
Descripción generada automáticamente

Ahora, usamos un comando llamado git pull. Este comando guarda todos los cambios en el repositorio hechos en GitHub. Cuando lo usamos, el documento pasa a tener lo mismo que el documento de GitHub.

Texto

Descripción generada automáticamente

Si hacemos un git lg, vemos que el commit hecho en GitHub se ha añadido.



# EJERCICIOS PRÁCTICOS

## EJERCICIO 1

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

## EJERCICIO 2



## EJERCICIO 3

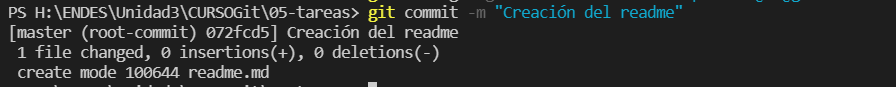
Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

## EJERCICIO 4

Texto

Descripción generada automáticamente



## EJERCICIO 5

Imagen de la pantalla de un celular con letras

Descripción generada automáticamente con confianza baja

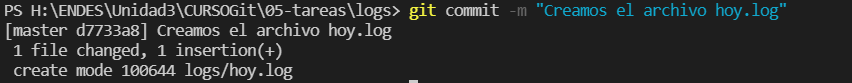
## EJERCICIO 6



Interfaz de usuario gráfica, Patrón de fondo

Descripción generada automáticamente

## EJERCICIO 7



## EJERCICIO 8

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Texto

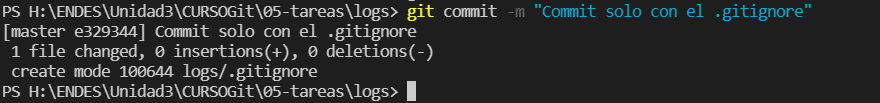
Descripción generada automáticamente

## EJERCICIO 9



Interfaz de usuario gráfica, Texto

Descripción generada automáticamente con confianza media



## EJERCICIO 10

Captura de pantalla de un celular

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

## EJERCICIO 11

Texto

Descripción generada automáticamente

## EJERCICIO 12

Porque nunca le hicimos un commit al historia.log. Al no hacerlo, no puede restáuralo de ningún modo.