Actividad 04 – Comandos GIT

Contenido

[1 Consultar cambios en los archivos 2](#_Toc151585945)

[1.1 Comparar cambios en archivos con visual studio code 4](#_Toc151585946)

[2 Actualizar mensajes de commits y deshacer commits 5](#_Toc151585947)

[2.1 Actualizar el mensaje de un commit 5](#_Toc151585948)

[2.2 Borrar un commit 5](#_Toc151585949)

[3 Resets y reflogs 6](#_Toc151585950)

[4 Ignorar archivos que no deseamos 8](#_Toc151585951)

**A continuación, completa este manual con tus capturas tras la ejecución de los comandos, sustituyendo cuando sea necesario las imágenes por las tuyas.**

# Consultar cambios en los archivos

Creamos un nuevo repositorio en el directorio GIT de C:/Users/tu\_usuario/Git 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

...

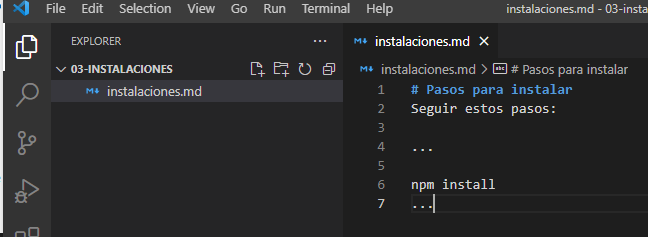
Lo agregamos al stage:

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

...

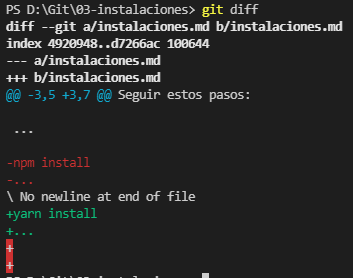
(Hemos modificado una línea y añadido 3 líneas vacías debajo de la segunda línea con …)

**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



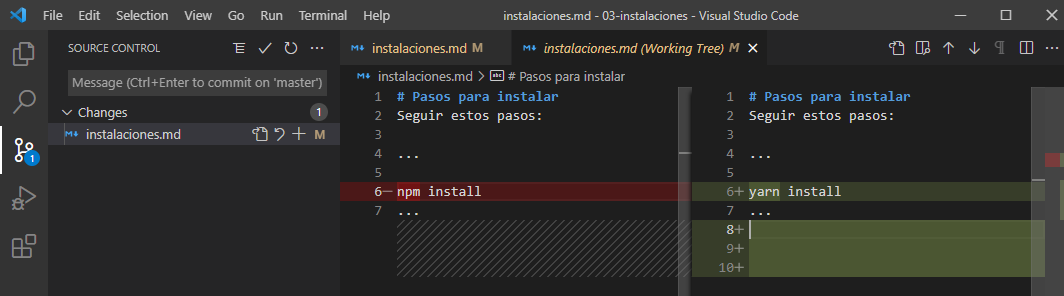
Las líneas en color rojo y precedidas por un - nos indican cosas la primera versión no tiene respecto al segundo y las cosas en verde precedidas por un + nos indican cosas que la segunda versión tiene y la primera no.

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

## 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

# Actualizar mensajes de commits y deshacer commits

## 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”

## Borrar un commit

Modificamos el archivo instalaciones.md (escribimos cualquier cosa) y hacemos commit:

git commit -m “Segundo commit”

Para borrar el último commit realizado usaremos el siguiente comando (debe haber al menos dos commits realizados ya que no se puede borrar un commit si sólo hay uno):

Dependiendo de la versión Git puede ser alguno de estos 2 comandos

git reset --soft HEAD^

git reset --hard HEAD~1

Ejecutar git log para comprobar si ha desaparecido el ultimo commit

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.

Si en vez de --soft usamos --hard no se mantendrán los cambios realizados antes del último commit, es decir se volverá al estado del commit anterior al borrado tal cual estaba el estado de los ficheros en ese commit.

git reset --hard HEAD^

git reset --hard HEAD~1

Ejecutar git log para comprobar si ha desaparecido el ultimo commit

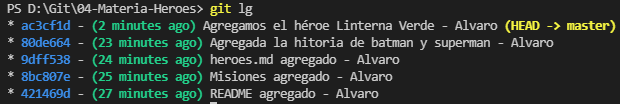
# Resets y reflogs

**Los resets se utilizan para volver a commits anteriores.**

Vamos a **añadir el héroe Robin o cualquier otro que no exista** en el archivo heroes.md:

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

Hacemos un git lg . O bien git log dependiendo de la versión Git.



Vamos a regresar a un commit anterior usando como referencia el código hash del commit al que queremos volver. Nos lleva al comit indicado y **además nos deja los ficheros modificados en el stage**

git reset --soft 80de664

Comprueba con git log que el ultimo commit ha cambiado

Comprueba con git status si los ficheros modificados están en el stage

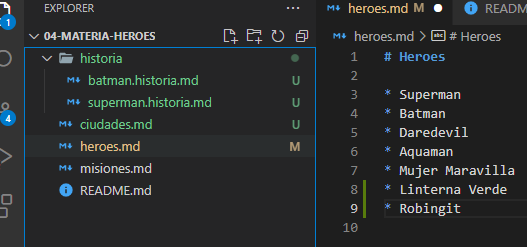
Hacemos un nuevo commit:

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 NO se queden en el stage** haremos un git reset --mixed:

git reset --mixed 9dff538

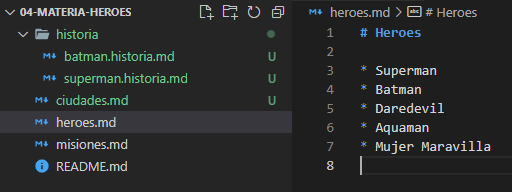
Si no ponemos una opción del modo de reset (--soft --mixed etc) el tipo de reset que se hace es el --mixed ya que es el modo por defecto.



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 todos 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:

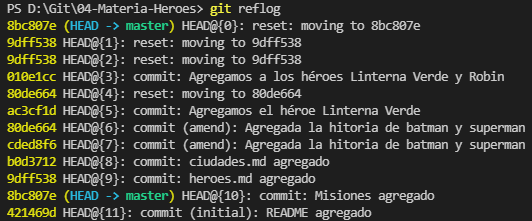
git reset --hard 8bc807e

Si hacemos un nuevo git lg:



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 de resets y commits realizados:**

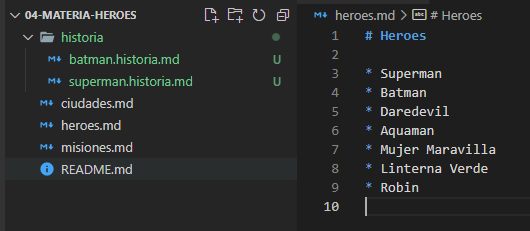
git reflog



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:



# Ignorar archivos que no deseamos

Si queremos crear carpetas y archivos a los que queremos configurar que git no les de seguimiento existe un fichero especial llamado **.gitignore donde registramos los ficheros y carpetas que no queremos que git tenga en cuenta.**

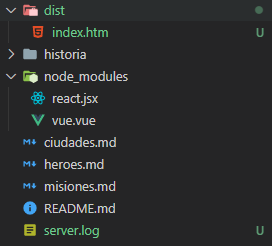
Por ejemplo creamos estas carpetas:

* dist

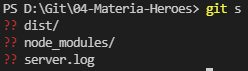
Y los siguientes archivos:

* server.log
* dist/index.html

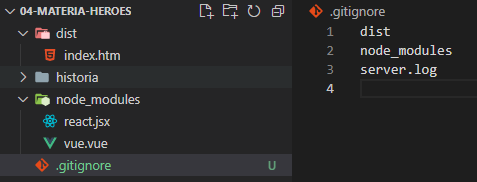
En los archivos podemos poner el texto que queramos o dejarlos vacíos.



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



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:



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”