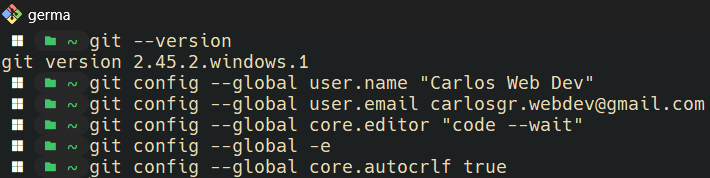
GIT BASH

1. **Configuración:**

Una vez instalado y abierto Git Bash, utilizamos estos comandos para configurarlo 

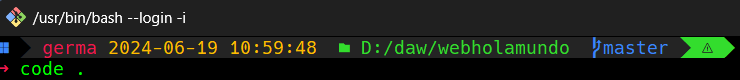
* **git –version** nos dirá si tenemos correctamente instalado Git, nos aparecerá así nuestra versión
* **git config –global** sirve para que la configuración que vamos a agregar se efectue de manera global y no por proyecto, a continuación ponemos el nombre de usuario con user.name entre “estas dos comillas dobles”, las utlizamos para poner espacios. Lo siguiente es configurarlo con nuestro correo electrónico, sin necesidad de comillas.
* **git config –global core.editor** **“code --wait”** 🡪 Es para que la terminal se quede esperando hasta que cerremos nuestro editor de texto, que se abrirá con el siguiente comando:
* **git config –global –e** 🡪 Se nos abrirá nuestro editor de texto predeterminado “VSC” al presionar ENTER nos saldrá la configuración en VSCode. Nuestro nombre, el email, la rama con la que vamos a trabajar inicialmente y el comando que se usa para abrir nuestro editor. Si volvemos a la terminal nos dirá que está esperando a que se cierre el editor y al cerrarlo podremos seguir poniendo comandos.
* **core.autocrlf** 🡪 Supongamos que existen 2 desarrolladores trabajando sobre un repositorio, uno con Windows y otro con Mac o Linux. En Windows, cuando hagamos un salto de línea, nos va a agregar dos caracteres especiales para poder agregar ese salto de línea, se utilizan en este caso los caracteres especiales de ***Carriage Return*** o ***CR*** y también ***Line* *Feed*** o ***LF***. En el caso de Linux y Mac, solo se agrega LF. El desarrollador que esté trabajando en Windows, cuando quiera subir código, va a tener que eliminar necesariamente el carácter especial CR y si por el contrario quiere descargar contenido del repositorio tendrá que usar LF. Para eso, tenemos que cambiar la configuración de GIT y para eso la propiedad de ***core.autocrlf***tiene que tener el valor de true para que se haga de forma automática. **En Mac y Linux el valor de *core.autocrlf* tiene que ser input y no true.** Así dejamos configurado como GIT debe tratar los saltos de línea.
* **git config –h 🡪 Es la sección de ayuda para la configuración.**

1. **Comandos de GIT:**

* **ls** 🡪 lista todas las carpetas que tenemos en el directorio en el que nos encontremos.
* **pwd** 🡪 nos indica en qué directorio o carpeta nos encontramos. Directorio es como se llaman en Linux y Mac lo que en Windows se llama “Carpeta”
* **cd** 🡪 quiere decir “change directory” y podemos agregar un nombre al que nos queramos dirigir. Si tenemos un directorio llamado **workspace** por ejemplo y pulsamos las teclas “**wo” + tab”**, nos autocompletará el nombre y pulsando **enter**, se cambiará de directorio. Si tenemos más de una opción, nos saldrá todas las opciones que tenemos, para seguir autocompletando, escribimos el siguiente carácter que nos interesa. Si queremos cambiar de unidad basta con poner **cd /nombre\_de\_unidad/**
* **cd ..** 🡪 Nos volverá a un directorio anterior en la jerarquía, hasta llegar al directorio padre si lo pulsamos repetidamente.
* **mkdir** 🡪 crea un directorio nuevo, es igual que crearlo de forma gráfica, pero con comandos. Escribimos **mkdir miweb** por ejemplo y verificamos con **ls** que aparece y **cd miweb** para entrar a ese directorio.
* **Git init 🡪** Inicializa un repositorio en la carpeta en la que estemos. Nos mostrará que lo ha creado en la ruta que sea con un archivo .git. Ese archivo está oculto al llevar un punto, si lo listamos no nos aparecerá excepto que ejecutemos el comando **ls –a** o **ls –all**
* **Ctrl+l 🡪** Limpia toda la terminal.
* **En el directorio “.git” ls –a:** Aparecerán todos los archivos que se utilizan para poder gestionar nuestros proyectos, aquí se van a almacenar todos los elementos de nuestro código.

1. **Flujo de trabajo en GIT**

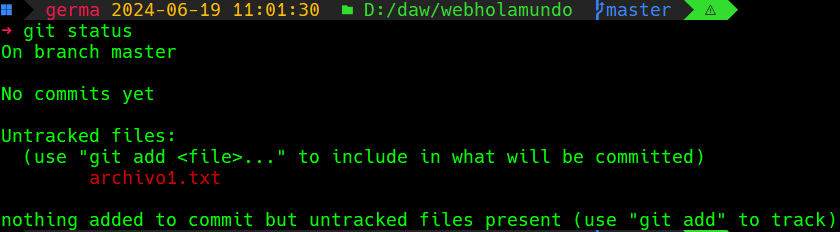
* Existen 4 etapas:
  + **Computer**
  + **Stage**
  + **Commit**
  + **Server**
* Lo primero que vamos a ver es que trabajando en computer, que es nuestro ordenador, vamos a poder modificar los archivos como queramos. Es la etapa de trabajo normal y corriente, el escribir el código. Para subirlo a un repositorio son los siguientes pasos. Aquí, solo seleccionamos los archivos que queremos pasar a la etapa stage.
* En stage se pasan los archivos desde el ordenador con el comando ***Git add*** seleccionando los archivos del ordenador y pasándolos al stage. Cuando los pasamos, lo que estamos haciendo es poder verificar todos los cambios que hacemos y querremos pasar al repositorio. Esta es una etapa intermedia para indicar los cambios que hemos hecho y de los que queremos tener constancia antes de subirlos al repositorio. No necesariamente quiere decir que se vayan a ver en el repositorio y no pasan todos, sino solo los que seleccionemos. Una vez que hayamos revisado esta etapa, podemos quitar elementos de ella, pero suponiendo que tenemos todos los que queremos pasar al repositorio, utilizamos el comando ***Git commit***.
* Cuando ejecutamos el comando ***git commit*** pasamos los archivos a los que les hayamos implantado ese comando al estado ***commit***, una vez que se hayan seleccionado y pasado los cambios aquí, viene una última etapa, que es pasarlos a un servidor, Github o cualquier otro servicio en la nube.
* Para elminiar los archivos restantes, no basta con borrarlos del ordenador, hay que pasarlos de nuevo al stage con ***git add*** aunque suene raro, pero es importante para mantener el registro de lo que estamos haciendo.
  1. **El flujo de trabajo en la terminal:**

****

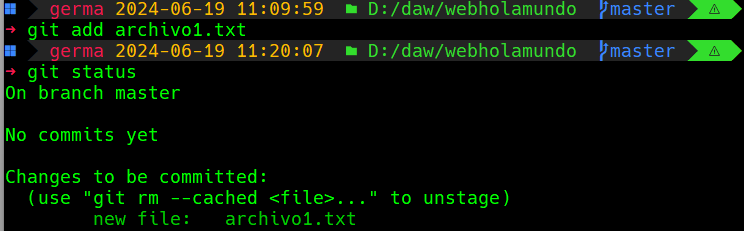
* Estando seguros de que estamos en la carpeta que queremos abrir, con el comando “***code . + enter”*** abriremos nuestra carpeta en el editor de texto.
* Creamos un archivo y lo guardamos:



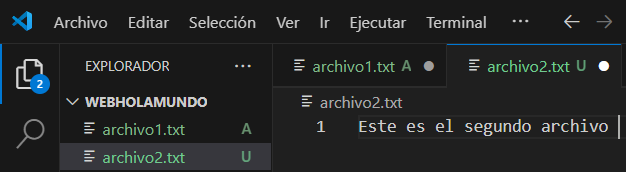
* De vuelta en la terminal podemos escribir ***git status*** para que nos muestre el estado actual de nuestro repositorio



* En este caso nos está indicando que todavía no existe ningún ***commit*** y que hay ***untracked files***, que son archivos que GIT no está siguiendo aún. Por defecto, no sigue todos los archivos que originemos en el proyecto, tenemos que seleccionarlos nosotros con ***git add + nombre del archivo y extensión*** (por ejemplo “archivo1.txt” o si queremos agregar todos los archivos del mismo tipo que se encuentren en la carpeta: \*.txt o podemos utilizar simplemente un punto, para que añada TODOS los archivos que haya, independientemente de su extensión, pero es una mala práctica, porque podríamos incluir archivos binarios enormes, como fotos o lo que sea, lo suyo es poner solamente el archivo que queramos subir).
* En el siguiente caso escribimos ***git add*** y ***git status*** para ver en qué estado se encuentra el archivo.



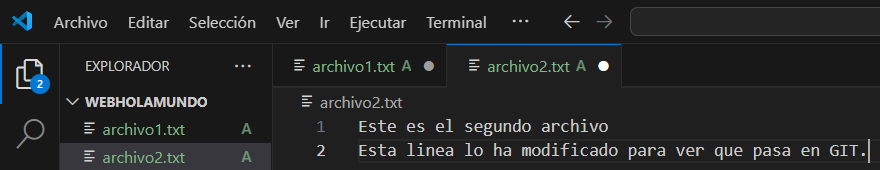
* Nos indica que seguimos en la rama ***master*** y que no tenemos ningún commit pero que hay cambios listos para ser comprometidos. De momento vamos a crear un segundo archivo para ver algunas cosas más:



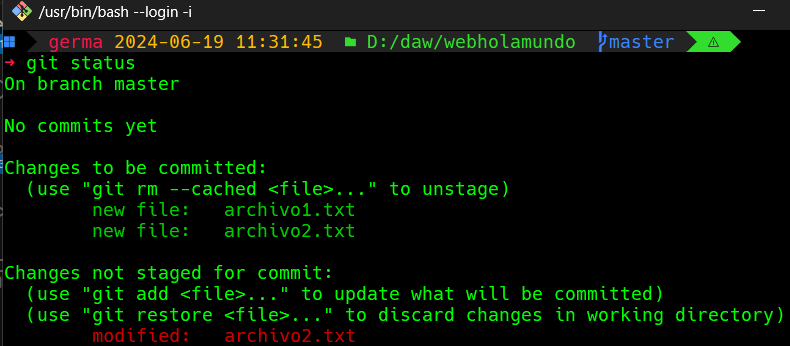
* Al volver a GIT y pasar el comando ***git status*** vemos cómo el primer archivo está en ***stage*** pero el dos no, así que vamos a agregarlo con ***git add archivo2.txt***



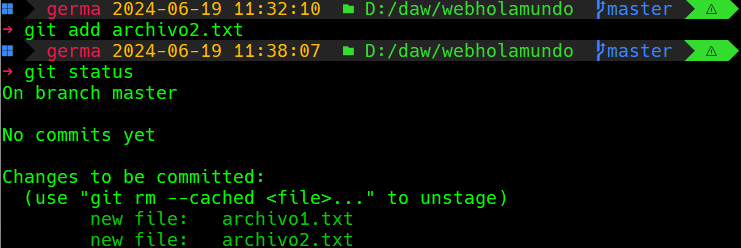
* Ahora hemos añadido el ***archivo2.txt*** al ***stage*** pero vamos a ver qué pasa si hacemos algún cambio en el ***archivo2.txt*** en VSC



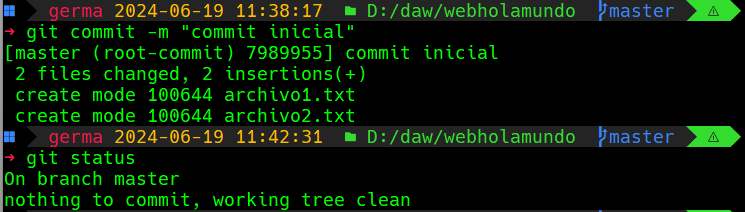
* Escribimos ***git status*** nuevamente y nos dice que están agregados pero que también el ***archivo2.txt*** está modificado y cuyos cambios no se encuentran en una etapa de ***stage***. Se refiere a los del ***archivo2.txt***. Los cambios que se encuentran en la etapa de ***stage*** son los que contenía cuando solo teníamos una línea de texto, es decir, la segunda no se ha incluido. Así que si le hacemos un commit, o lo comprometemos, se va a quedar como el original, solamente con la primera línea y si queremos reemplazar ese cambio, tenemos que volver a agregar el ***archivo2.txt.***



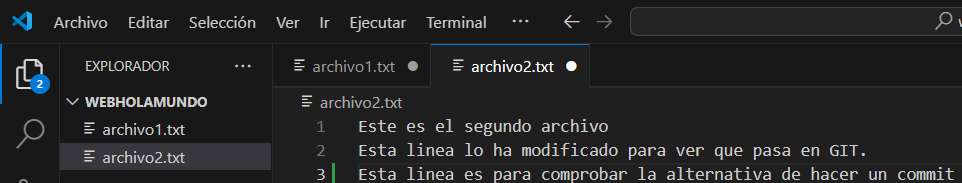
* Agregamos el ***archivo2.txt*** y volvemos a escribir ***git status*** y podemos ver el estado completo de la etapa ***stage.*** Ojo, lo que estamos pasando ¡no son los mismos archivos que antes, sino las modificaciones que les hemos hecho!

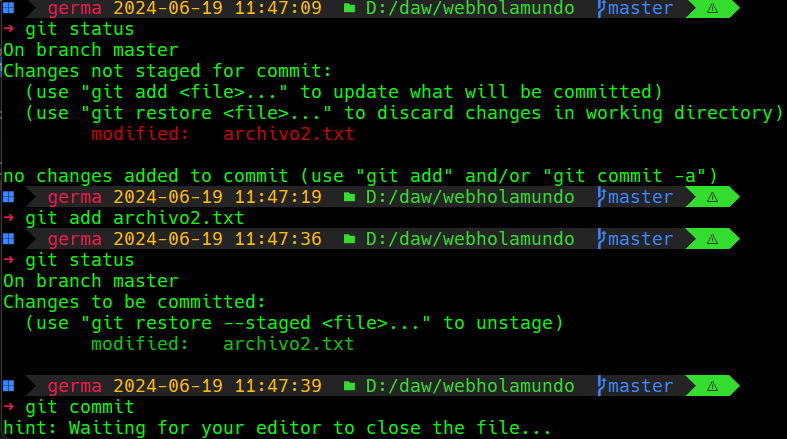


* Ahora ya podemos comprometer o hacer commit los archivos. Existen dos maneras, utilizar el comando: ***git commit –m “mensaje con sentido indicando las modificaciones que hemos hecho, en este caso commit inicial”***

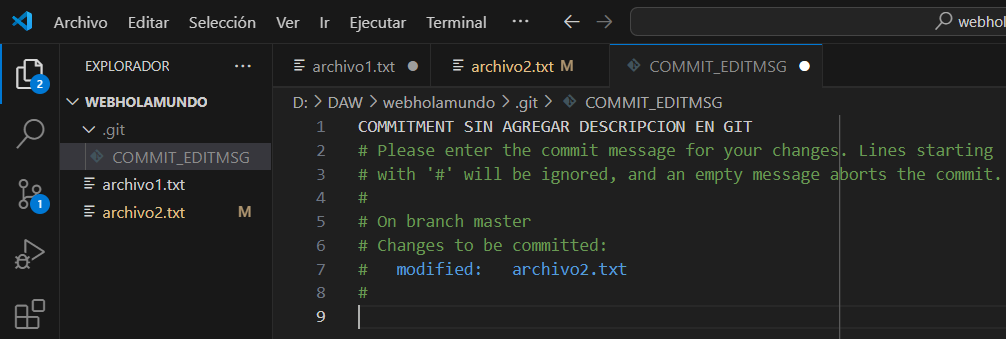


* Una vez que los hemos comprometido, podemos ver el ***status*** y vemos que no hay nada en la ***rama master***. La forma alternativa es volver al editor de texto y en el archivo 2 por ejemplo, agregamos una línea. Volvemos a GIT, ponemos el ***git status*** y vemos que el archivo ha sido modificado, así que vamos a agregarlo de nuevo y volvemos a realizar un ***git commit***, pero sin agregarle ninguna descripción.

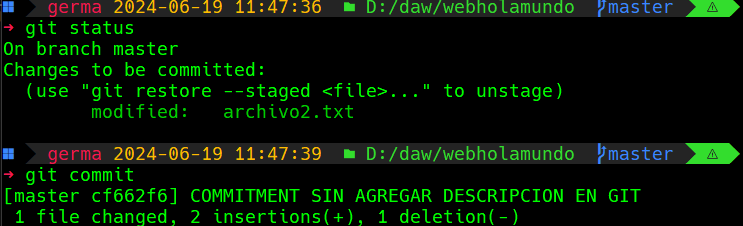




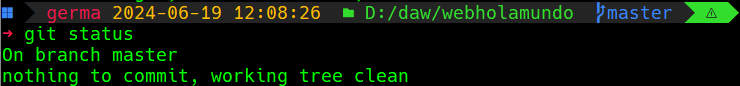
* En este caso al hacer el ***git commit*** sin añadirle una descripción, pide que se abra el VSC. Una vez abierto nos indica cual es el archivo que ha sido modificado y arriba se agrega el dato de modificación arriba. Debemos agregar el mensaje, guardarlo y cerrarlo.



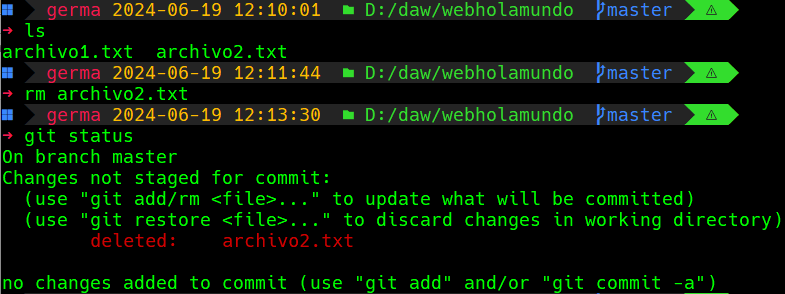
* Una vez guardado y cerrado en GIT el status aparece sin añadir ningún comando de la siguiente manera:



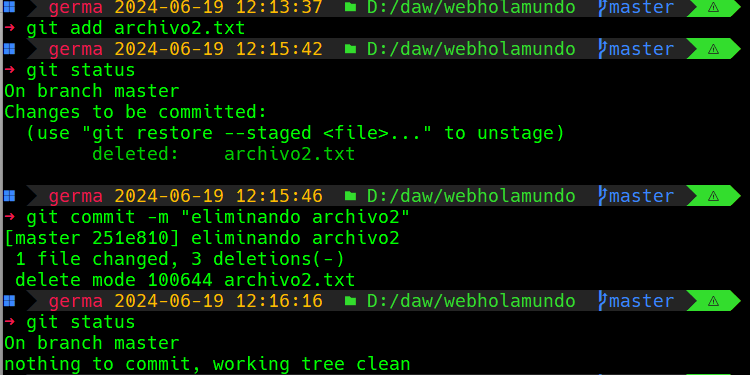
* Si ejecutamos un git status, vemos que ya no tenemos nada a lo que hacerle commit



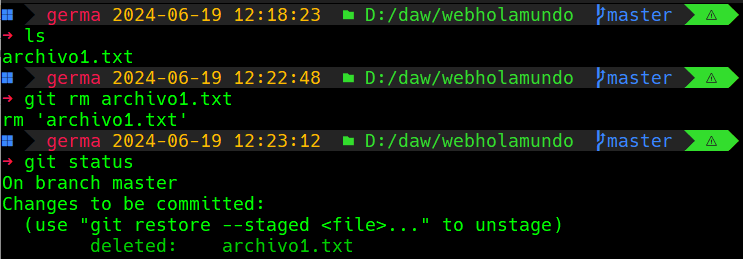
* Ahora vamos a eliminar archivos, primero listamos para ver los archivos existentes y con el comando ***rm archivo2.txt*** eliminamos el ***archivo2.txt.*** y al escribir ***git status*** vemos cómo ahora nos pone ***deleted: archivo2.txt*** pero vemos que estos cambios no están en la etapa de ***stage*** asi que tenemos que agregarlos



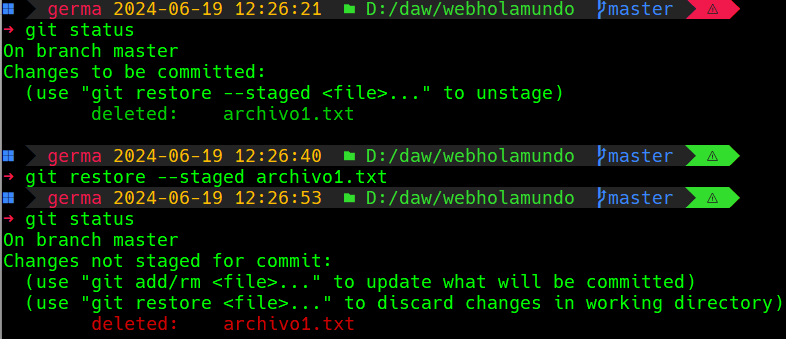
* En la siguiente captura, se ve que se ha agregado el ***archivo2.txt*** al ***stage*** para ser eliminado, pasándole un ***git status*** para ver en qué estado se encontraba y confirmando que se quiere eliminar con ***git commit.*** Es decir, el mismo proceso que se utiliza para agregar un cambio, se utiliza para eliminar un archivo, el único cambio es agregarlo o eliminarlo, pero el caso es el mismo. Se encuentra en nuestro ordenador y de ahí lo ponemos en el ***stage*** y desde ese limbo que es el ***stage*** podemos agregarle información y hacerle un ***commit*** con el código nuevo o eliminarlo con el comando ***rm***, pero siempre eso se hace desde el ***stage***.



* Sin embargo, GIT tiene un comando para eliminar directamente el archivo que se desee. Se va a eliminar el ***archivo1.txt*** utilizando ***ls*** para listar y ver que solo existe ese archivo, el comando ***git rm archivo1.txt*** y ***git status*** para ver cómo se ha eliminado del repositorio.



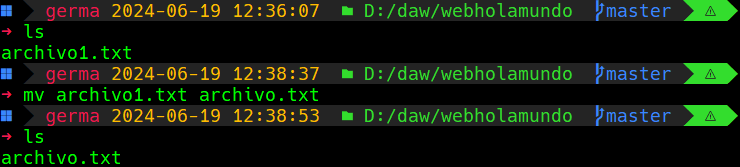
* Sin embargo, esto aun no es definitivo, como se puede ver, porque aun no ha sido comprometido y podemos restaurarlo con el comando que dice: ***git restore –staged archivo1.txt***, pero al escribir ***git status*** vemos que no se encuentra en la etapa de ***stage.*** ¿Pero qué pasa si queremos descartar los cambios? Si ponemos ***ls*** vemos que el archivo se ha eliminado, pero quiero restaurarlo. Así que podemos utilizar ***git restore archivo1.txt*** y ***git status*** para ver cómo se ha restaurado.



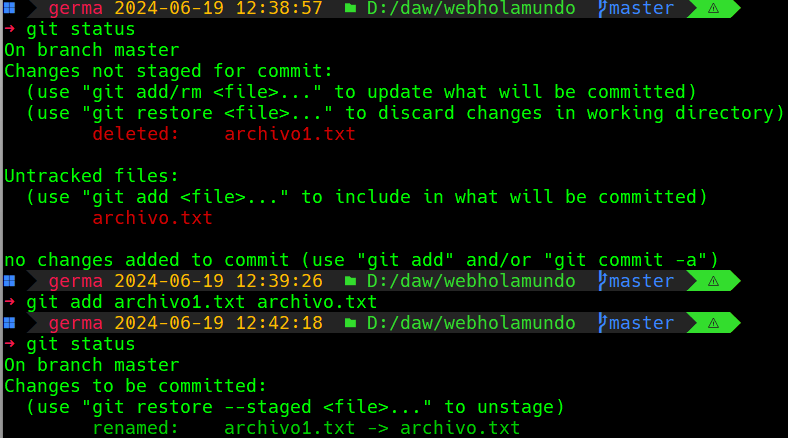
Al listar no aparece nada, hasta que se restaura el archivo1 y recuperarlo hasta una etapa antes de hacer commit.



* Ahora vamos a mover el archivo con el comando ***mv + destino*** esto nos cambiará primero el nombre, al que vamos a llamar “archivo” puesto que solo existe uno.



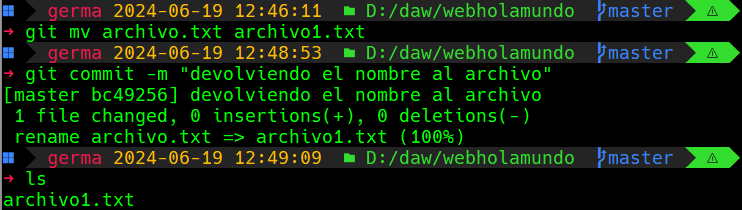
* Al escribir ***git status*** se ven los cambios, el primero es que se ha eliminado el ***archivo1.txt*** y el segundo es que se ha creado un nuevo archivo, el cual GIT no está siguiendo, así que hay que agregar ambos archivos. Para hacer esto se escribe ***git add archivo1.txt archivo.txt*** y tras introducir un ***git statu***s nos aparece la opción de “***renamed”*** lo que quiere decir, que ***archivo1.***txt ha cambiado de nombre a ***archivo.txt***



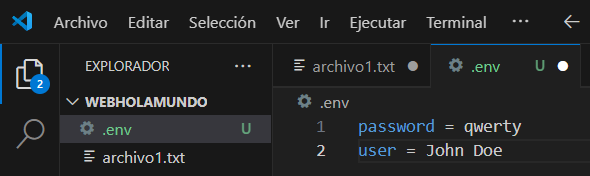
* Ahora se pueden comprometer los cambios con ***git commit –m “renombrando archivo”***



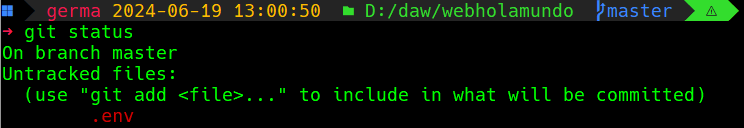
* De la misma manera se puede volver a renombrar el archivo inmediatamente con los comandos ***git mv archivo.txt archivo1.txt*** y ***git commit –m “devolviendo el nombre al archivo”***



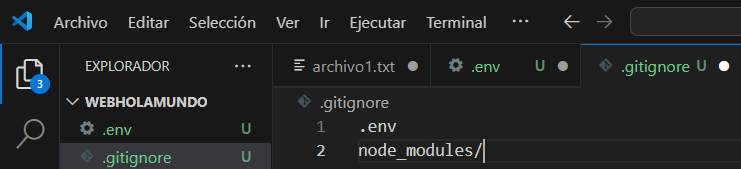
* 1. **Ignorando archivos y directorios**
* Se pueden ignorar archivos para que no se incluyan en nuestro repositorio de GIT. Esto se hace porque a veces vamos a querer tener archivos de configuración específicos para nuestra máquina. Por ejemplo, estamos trabajando con una BD y la tenemos en local y esto es para que podamos estar trabajando con un ambiente de desarrollo. En este caso, los usuarios, contraseñas y demás van a ser diferentes a cualquier otro ordenador y vamos a querer tenerlo almacenado en nuestra máquina, pero no vamos a querer tenerlo obviamente en un repositorio global. Por ejemplo vamos a poner unas variables de configuración en VSC. Creamos un archivo “env” que es un archivo bastante standard para variables de entorno.



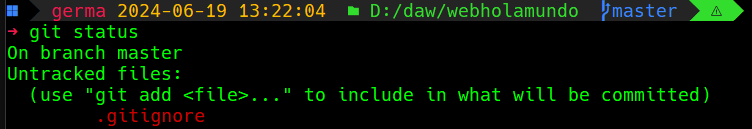
* Supongamos que mi aplicación se va a conectar a una base de datos MySQL utilizando estas variables. Si vamos a la terminal y escribimos ***git status***, veremos que aparece el archivo .***env*** y no queremos llegar a commitearlo.



* Para eso volvemos a VSC y creamos un nuevo archivo, que llamamos “.gitignore” y añadimos los archivos que queremos que sean ignorados, en este caso el archivo ***.env*** e imaginemos que vamos a trabajar con ***node*** y queremos que la carpeta ***modules*** también sea ignorada.



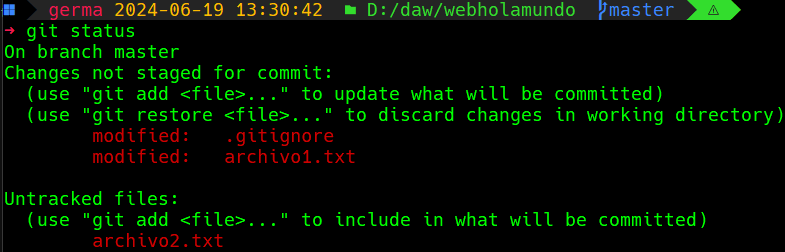
* Volvemos a GIT y escribimos de nuevo ***git status***. Al escribirlo, vemos que no aparece ***.env*** pero si nos aparece ***.gitignore***.



* Ahora vamos a agregar este archivo y comprometerlo inmediatamente agregando “archivo gitignore”



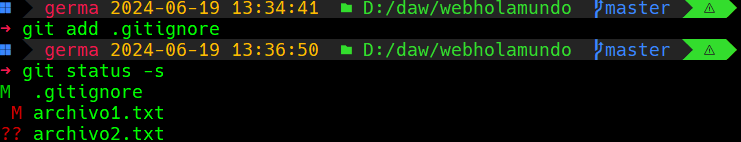
* Aprovechando este momento, vamos a ver un tip bastante útil para ver el estado en GIT, mejor que ***git status***. Este muestra bastante más información de la necesaria, así que vamos a ver una alternativa. Volvemos a VSC y realizamos algunas modificaciones: eliminamos el ***node\_modules/*** que no lo necesitamos y agregamos algo al ***archivo1.txt*** y creamos de nuevo un ***archivo2.txt***
* Limpiamos la terminal y escribimos de nuevo ***git status***.



* Como vemos se da mucha información: en que rama nos encontramos, los cambios que no se encuentran todavía en ***stage***, algunos cambios que todavía no se han agregado para que GIT les pueda hacer seguimiento y algunos comandos de ejemplo que podemos ejecutar en el caso de que queramos llevar a cabo alguna determinada acción. En este caso vamos a suponer que sabemos todo y vamos a escribir el nuevo comando: ***git status –s***



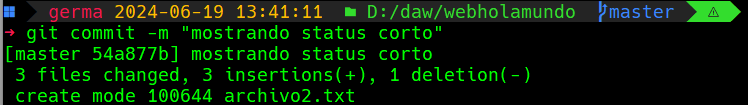
* El primer archivo contiene una “M”, de “modified”, puesto que ha sido modificado y si escribimos git add .gitignore la M cambia de color porque se ha agregado a la etapa de ***staged***.



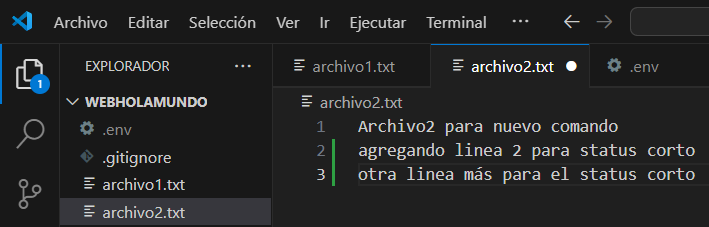
* El ***archivo2.txt*** tiene dos interrogaciones, porque todavía no ha sido agregado para que GIT le pueda hacer seguimiento. Vamos a agregar los archivos 1 y 2:



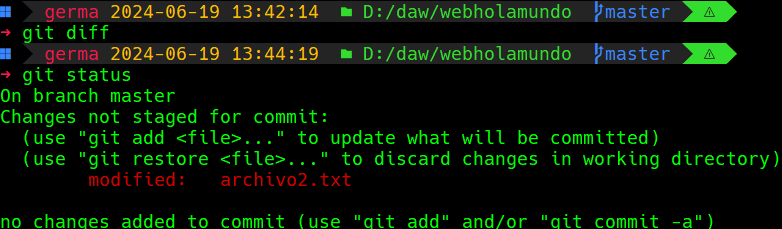
* Ahora aparecen todos en verde, lo que nos indica que están listos para hacer commit



* Vamos a ver cómo podemos ver los cambios que hemos agregado de una forma un poco más visual. Antes de nada en GIT escribimos ***git diff*** y pulsamos ***enter*** y en VSC agregamos algo en el ***archivo2.txt*** por ejemplo y guardamos



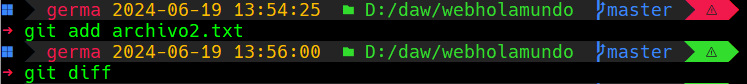
* En GIT escribimos ***git status*** y vemos que el ***archivo2.txt*** está listo para se agregado a la etapa de ***staged***  pero en lugar de eso vamos a escribir ***git diff***



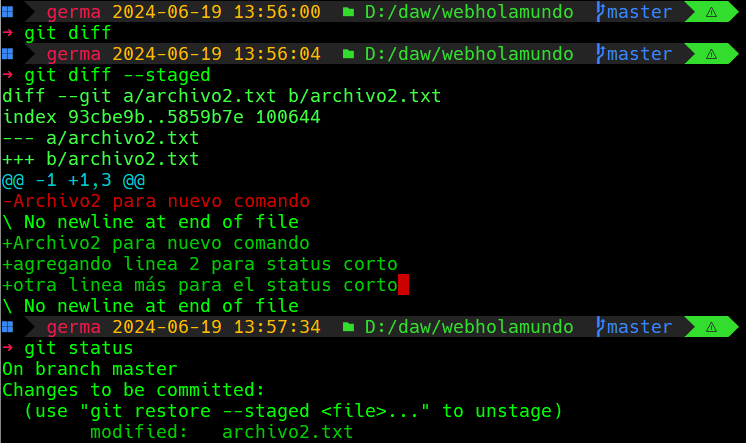
* De esta forma vamos a poder ver todo el texto de lo que representarían los cambios que hemos hecho en VSC:



* Nos está indicando en rojo con el símbolo – que ha eliminado algo y con el + que lo ha vuelto a agregar, esto es porque antes solo teníamos la línea roja y no existía un salto de línea al no pulsar enter. Al agregar el salto de línea, se agrega sobre la misma línea y por eso “agrega” un carácter nuevo. Además, nos indica que está agregando “Archivo2 para nuevo comando, y las otras dos líneas que contienen el símbolo +” Seguido del símbolo @@ nos indica que esto está comenzando desde la línea 1 con la versión anterior de nuestro archivo (-1) y después de esto, lo que ha añadido, también desde la línea 1 (+1) y que se han añadido 3 lineas, las que vienen siendo los cambios. Ahora, podemos agregarlos escribiendo ***git add archivo2.txt*** y si ponemos ***git diff*** vemos que no hay ningún cambio.



* Si agregamos además ***git diff --staged*** y ***git status*** nos va a mostrartodos los cambios que se encuentren en la etapa de ***staged.*** Esto es muy útil ya que si hay cambios vamos a querer ver solo los cambios que nosotros queremos comprometer y no absolutamente todo.



MINUTO 52:32 HISTORIAL