

22 de abril de 2020

David Martin Vergues

Curos Git y GitHub

UDEMY course

# Tema 1

## ¿Cómo funciona GIT?

Básicamente con git conseguimos tener un proyecto distribuido. A ese proyecto de le denomina repositorio. Distribuido significa que cada integrante del proyecto tiene una copia de ese repositorio en local.

Git establece una línea del tiempo desde que se crea el repo hasta el día de hoy. Durante esa línea del tiempo vamos tomando fotografías de como se encuentra el proyecto, los llamados commits, de tal manera que siempre podremos volver a alguno de esos momentos.

## Primeros pasos

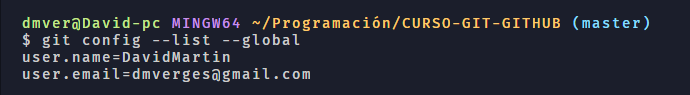
Lo primero que debemos hacer es guardar un nombre de usuario y un mail en nuestra máquina para presentarnos quienes somos en GIT, estas credenciales serán las usadas en todos los repositorios que tengamos en local. Podemos optar por tener un nombre y un mail específicos por repositorio. Todo depende si añadimos al comando la palabra global.

Usamos: config global





Para saber si se ha guardado bien hacemos



Podemos ir directamente al archivo q almacena esta información en .gitconfig y alterarla(no recomendable)





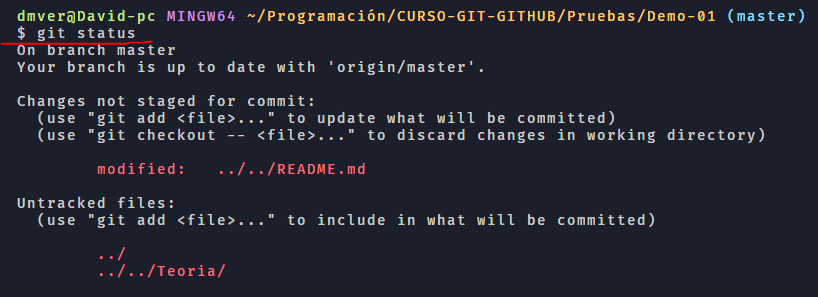
Usamos: init status add commit .gitignore

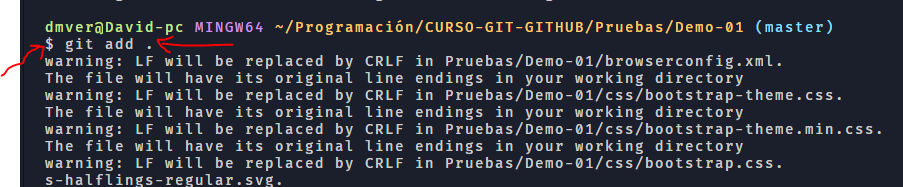
Cuando hacemos un init en nuestro repositorio local se crea la carpeta git donde estará todo lo necesario para guardar nuestros cambios.



Luego creamos nuestros archivos

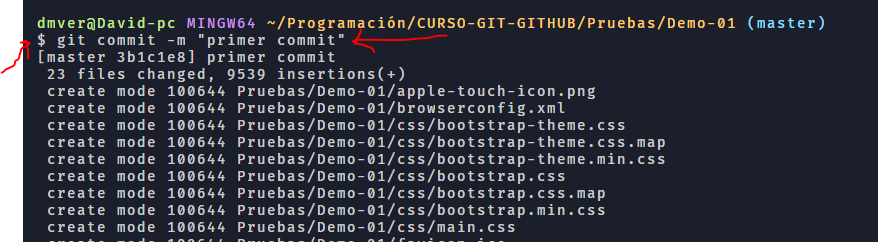
Cuando queremos ver el estado de git usamos git status y si hay cambios que hay que incluir usamos git add.





Al importar un repositorio hecho en Linux en un entorno Windows el final de las líneas en diferente, en Linux usan LF (line feed) y en Windows CR (carriage return).

Para tomar la foto del proyecto actual usamos commit -m “mensaje”



Con git init inicializamos el repositorio. Creamos los archivos necesarios. Con git add . ó nombre del archivo añadimos al stage esos archivos, después hacemos un commit que toma una foto de todos los archivos que hay colocados en el stage (escenario) para que git pueda registrar como se encuentran.

Si queremos descartar algún tipo de archivo, para q no sea seguido por git creamos un archivo .gitignore con la ruta a los archivos que no queremos controlar.

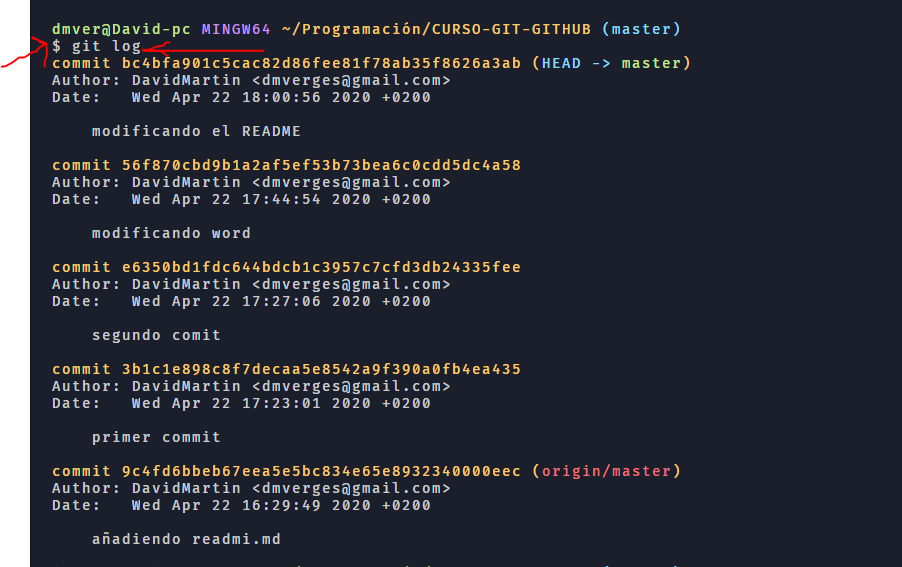


Usamos: checkout log

Imaginemos que por error modificamos un archivo o eliminamos su contenido. Con el comando checkout recuperamos el proyecto como estaba en el último commit. Esto también sirve para recuperar carpetas borradas

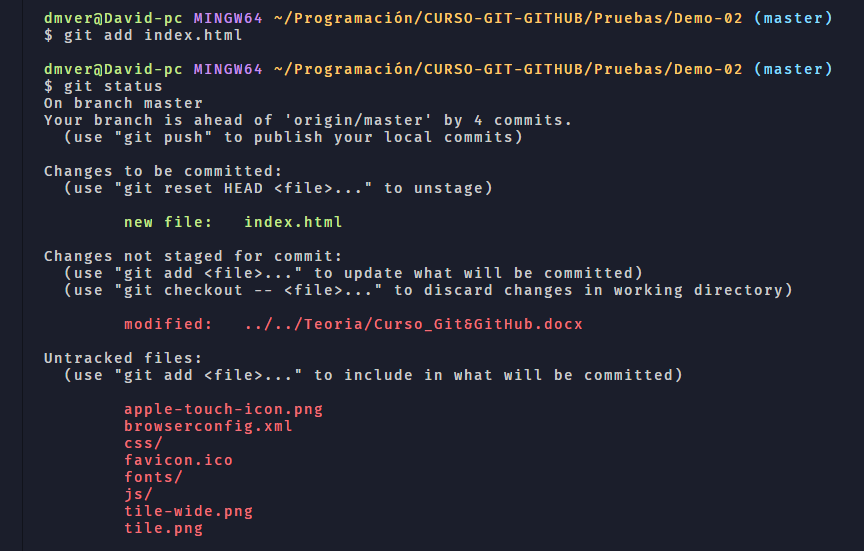


A medida que vamos haciendo commits se va creando un registro con toda la información. Para acceder a este registro usamos log



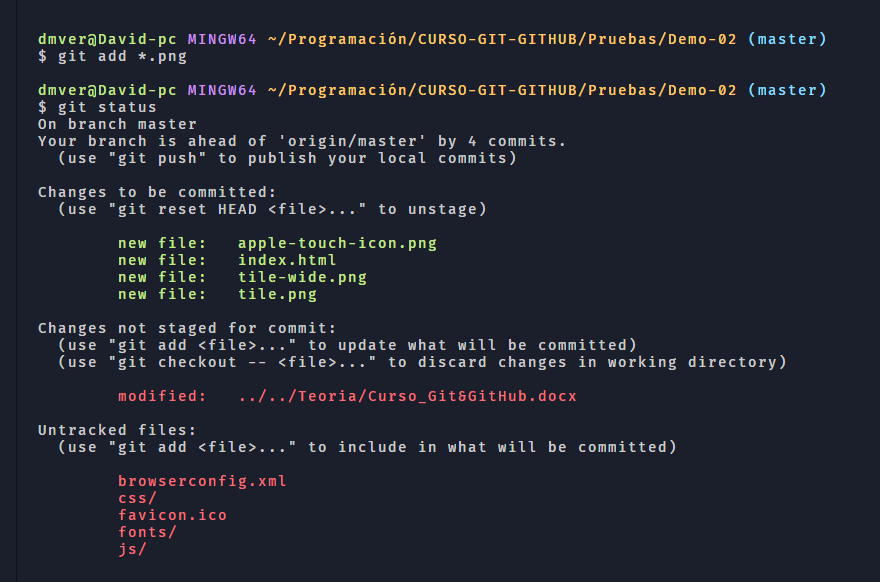
Lo correcto sería hacer commits por archivos, es decir si tenemos varios archivos modificados incluir cada uno de los archivos por separado y hacer el commit. Por jemplo hago el commit solo del index.

Usamos: add \*.png add css/ git add -A reset

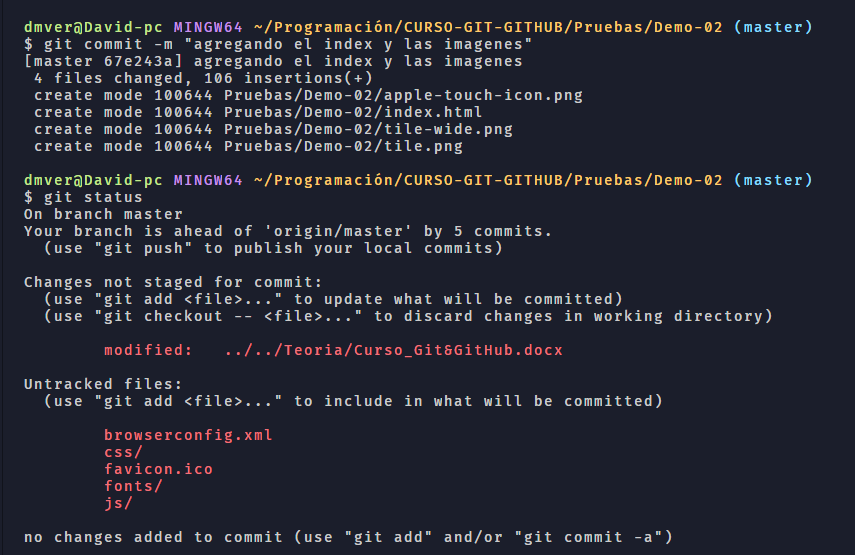


En la imagen vemos como index está en el stage y el resto de archivos en rojo todavía no.

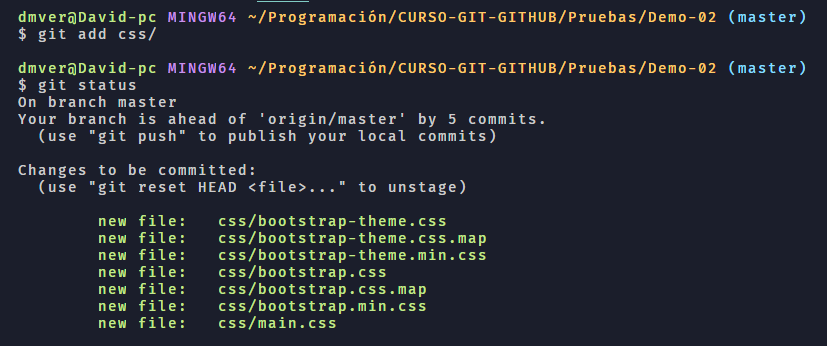
Para añadir todos los archivos de un mismo tipo git add \*.png



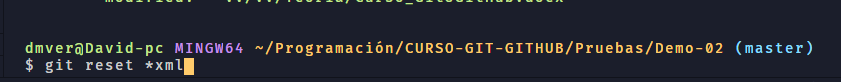
Hacemos el commit indicando que trackeamos el index y las imágenes.

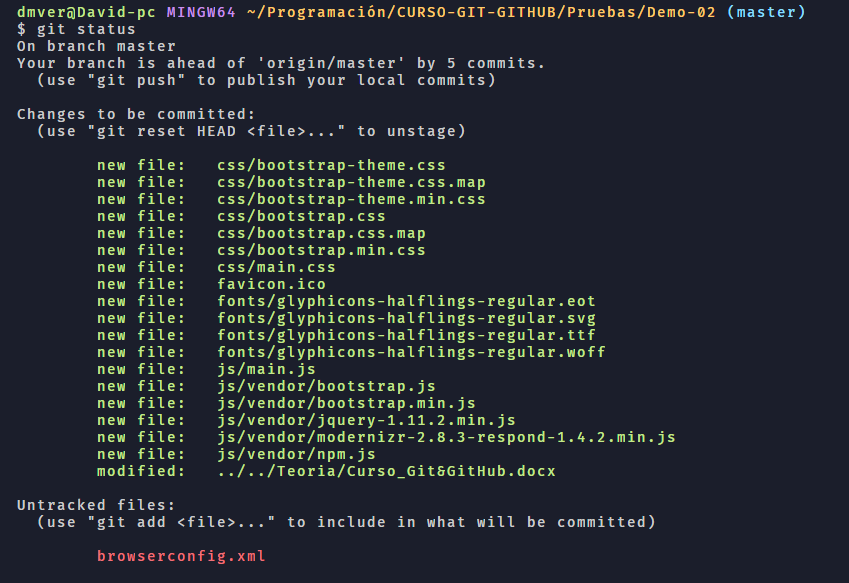


Agregar toda una carpeta el stage



Con git add -A añadimos todos los archivos con cambios al stage pero si queremos descartar uno antes del commit hacemos un reset





Y lo podemos añadir a parte. Si hacemos un log veremos todo el regustro



### Tipos de add

Git add “\*.txt” => añade al stage los archivos txt modificados en todo el proyecto

Git add \*.txt => añade al stage los archivos txt modificados sólo en el directorio actual

Git add .

Git add - -all los tres agregan todos los archivos modificados en todo el proyecto

Git add -A

Git add <file1.txt, file2.js> => agrega una lista de archivos.

Git add carpeta/\*.pdf => agrega todos los archivos de un tipo dentro de una carpeta

Git add carpeta/ => agrega todos los archivos dentro de una carpeta

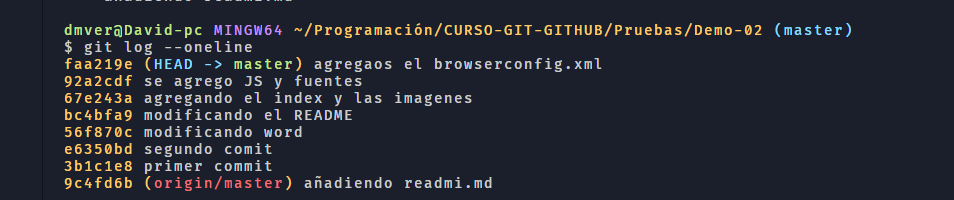
### Como revisar el log

Usamos: log --oneline --decorate --all –graph status -s -b

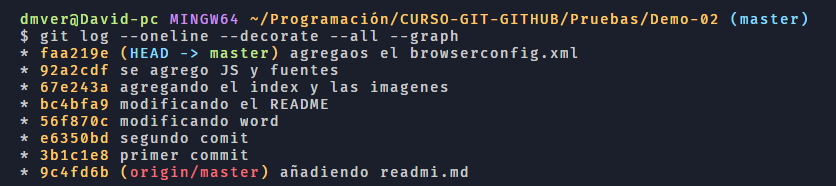
Cuando hacemos un log vemos el HEAD que nos indica el último commit de la rama en la que estamos, en este caso master



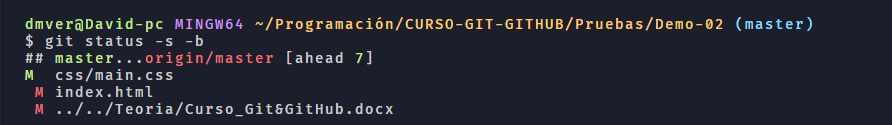
Oneline



Todos estos parámetros me servirán para ver mejor el log cuando tengamos ramas y merges…



status -s (silence) -b (brach) reducir la info del status



En M rojo modificados pero fuera del stage y M verde modificado pero dentro del stage