# GIT

Realiza todos los pasos que se sugieren en el documento y entrega en Aules un PDF con todas las capturas de pantalla que se solicitan, en total 12.

La guía para seguir los diferentes apartados está <a href="https://www.youtube.com/watch?v=3GymExBkKjE">aquí</a>.

## CONFIGURACIÓN DE GIT

El primer paso de todos es introducir un nombre y un email, puesto que todas las acciones que hagamos en el repositorio tienen que tener un autor.

```git

git config --global user.name "user\_name"

git config --global user.email "user\_email"

```

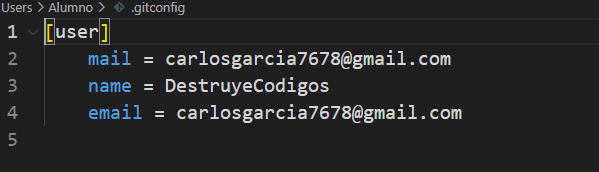
Puedes ver los valores de configuración con:

```

git config --list

```

1. Haz una captura de pantalla del archivo .gitconfig, donde aparezca los datos del usuario de Git en tu equipo.



## CREAR UN REPOSITORIO

Crea un repositorio con un archivo \*\*"fichero\_1.php"\*\*, el cual estará vacío cuando iniciemos el repositorio.

```

git init

```

Vemos en la consola de Git Bash que estamos situados en la rama Master.

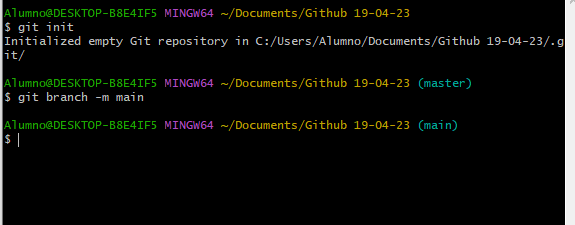
Renombramos la rama master por la rama main.

```

git branch -m main

```

2. Captura de pantalla del nombre de la consola de Git con el nombre de la rama cambiado.



## GUARDAR CAMBIOS

Para comprobar el estado de nuestro repositorio.

```

git status

```

Haz un commit llamado "Mi primer commit". Este commit será una versión de nuestro repositorio con únicamente el fichero \*\*"fichero\_1.php"\*\* vacío.

Ahora crea otro archivo \*\*"fichero\_2.php"\*\* y haz un commit con el siguiente mensaje "Este es mi segundo commit".

## GIT CHECKOUT

Vamos a volver a una versión anterior de nuestro repositorio.

```

git checkout nombre\_fichero

```

Entre otras funcionalidades, permite descartar los cambios locales que se han hecho en un fichero y volver a la última versión (commit) que había del repositorio.

Para probarlo puedes añadir un texto al \*\*fichero\_1.php\*\* y solo guardar los cambios en local. De esta manera, si hacemos:

```

git checkout fichero\_1.php

```

Podemos volver a la versión que teníamos en el momento que habíamos hecho el commit.

## GIT LOG

Vamos a modificar el archivo \*\*fichero\_1.php\*\* y hacemos un tercer commit.

Podemos ver todo el historial de commits de diferentes formas:

```

git log

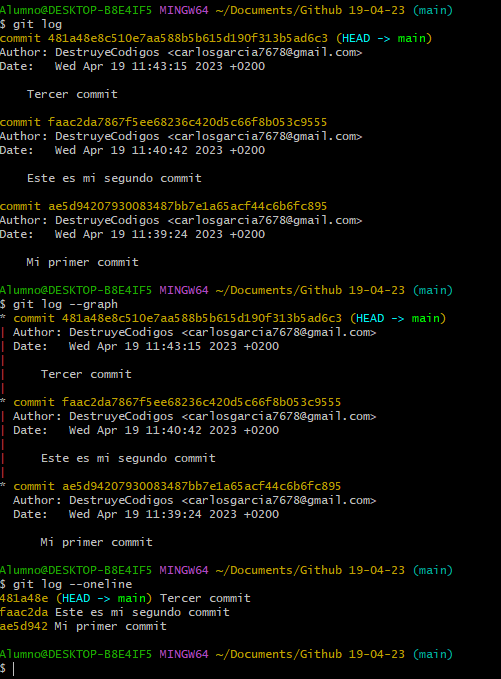
git log --graph

git log --oneline

```

De esta forma tenemos el hash y el lugar donde está el HEAD, entre otros datos.

3. Captura de pantalla del log hasta el momento, es decir los tres commits.



## GIT ALIAS

Vamos a crear un alias llamado \*\*"git tree"\*\* para que cuando queramos ejecutar el comando:

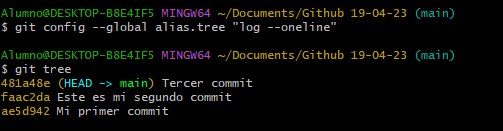
```

git log --oneline

```

podamos únicamente escribir \*\*git tree\*\*.

4. Adjunta una screenshot de la consola ejecutando el comando git tree.



## FICHERO .GITIGNORE

Crea un fichero que será aquel que ignoremos en los commits, por ejemplo \*\*"fichero\_3.php"\*\*. Después crea el fichero \*\*.gitignore\*\* a través de la consola.

```

touch .gitignore

```

Añadimos el fichero que no queramos incluir en los commits dentro del archivo creado:

```

\*\*/nombre\_fichero

```

5. Añade una captura haciendo un git status del repositorio, en el que aparezca el fichero .gitignore pero no el fichero\_3.php.

Por último, haz un commit en el que incluyas el fichero .gitignore.



## GIT DIFF

Haz cambios en el fichero \*\*fichero\_1.php\*\* y sin hacer un commit comprueba con el comando git diff que puedes ver cual han sido los cambios que has hecho en el fichero. Puedes añadir el siguiente código:

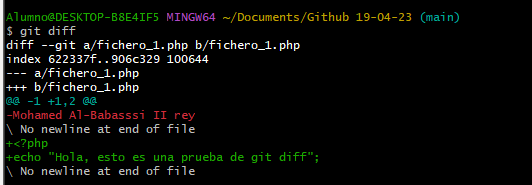
```php

<?php

echo "Hola, esto es una prueba de git diff";

```

6. Inserta una captura con las diferencias que aparecen en ambos ficheros.



Guarda los cambios en local y después vuelve a la versión del último commit con:

```

git checkout

```

Puedes hacer checkout de un fichero concreto o de todos en los que se hayan registrado cambios.

```

git checkout nombre\_fichero

git checkout .

```

Comprueba que has vuelto al último commit con el comando git status, que indicará que no hay ningún cambio para subir al apartado de stage.

## DESPLAZAMIENTO EN UNA RAMA

Ahora que hemos hecho varios commit, vuelve al principio de la rama:

```

git checkout <hash>

```

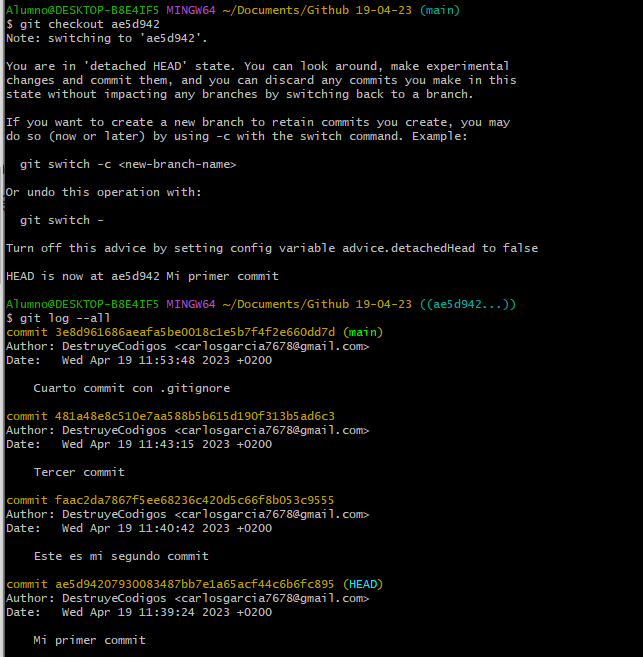
Ahora tienes el HEAD en el primer commit, pero los que has hecho posteriormente siguen estando ahí. Puedes consultarlos con el siguiente comando:

```

git log --all

```

7. Adjunta una captura una captura de todo el historial de commits.



Ahora vuelve a situar el HEAD en el último commit con:

```

git checkout <hash>

```

Fíjate que todavía no estamos en la rama main, por tanto, vuelve a la rama main con:

```

git checkout main

```

## GIT RESET HARD Y REFLOG

GIT RESET nos permitirá situarnos en un commit concreto de nuestra rama.

Ahora vamos a volver al commit en el que añadimos el fichero \*\*.gitignore\*\* y vamos a borrar los que hicimos posteriormente.

```

git reset --hard <hash>

```

Comprueba que no existen los demás commits con:

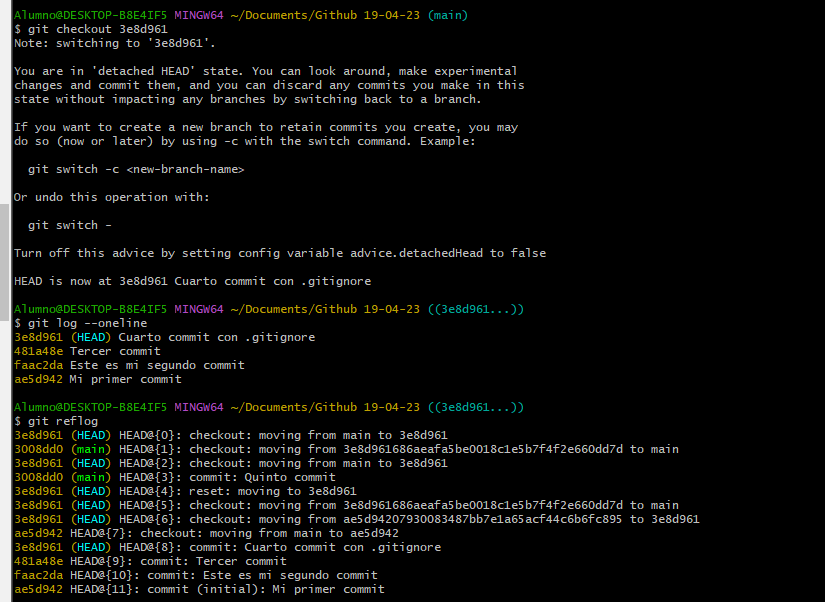
```

git log --oneline

git log --all

```

8. Adjunta la captura de pantalla de uno de estos dos logs.



Sin embargo, hay un lugar donde se quedan almacenados todos esos commits:

```

git reflog

```

Aquí encontramos un historial completo de interacciones con nuestro repositorio.

Ahora vuelve a situar el \*\*HEAD\*\* en el último commit que habíamos hecho, volviendo a hacer otro reset.

## GIT TAG

Vamos a utilizar los tags para etiquetar commits, concretamente aquellos que sean puntos relevantes dentro de mi repositorio.

Añade el tag \*\*clase\_1\*\* y después haz un commit para comprobar que el tag se queda guardado dentro del tercer commit.

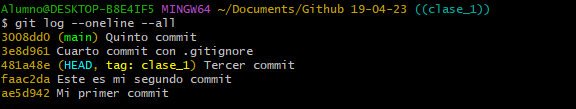
Vuelve el \*\*HEAD\*\* al commit en el que insertamos el tag con:

```

git checkout tags/clase\_1

```

9. Añade una captura de pantalla con el \*\*HEAD\*\* en el tag pero que también aparezca el commit posterior.



La ventaja es que podemos trabajar de forma más intuitiva y no únicamente con el hash del commit.

## GIT BRANCH Y SWITCH

Vamos a crear una nueva rama. Por ejemplo, estamos trabajando en un proyecto y queremos desarrollar una nueva funcionalidad para hacer un login.

Crea la rama y comprueba que se ha creado con:

```

git branch

```

La rama activa se marca con un \*

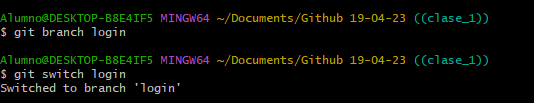
Ahora cambiamos a la rama \*\*login\*\*:

```

git switch login

```

10. Adjunta una captura con la consola en la nueva rama.



Ahora creamos el fichero \*\*login.php\*\* y hacemos un commit con el siguiente mensaje \*\*"commit del login"\*\*.

Volvemos a la rama \*\*main\*\* y comprobamos que no está el fichero \*\*login.php\*\*.

Dentro de la rama \*\*main\*\* y comprueba que cuando cambias a la rama \*\*login\*\* no se encuentran los cambios.

## GIT MERGE

Vamos a trabajar en la unión de ambas ramas. Para ello, cambiamos el \*\*HEAD\*\* a la rama login para traer los cambios que hemos hecho en la main a la rama login.

```

git merge main

```

Puedes dejar el mensaje que se introduce por defecto.

Comprueba que se han traido los cambios que has hecho en el último commit de la rama main a la rama login.

## RESOLUCIÓN DE CONFLICTOS

Hasta ahora no había ningún problema porque hemos trabajado en ficheros distintos en cada rama, pero ahora vamos a crear en cada rama un cambio en la misma línea, para que GIT nos informe de un conflicto. Para ello tendrás que hacer un commit en cada rama con un cambio en la misma línea. Cuando hagas el merge, es decir, cuando quieras llevar los cambios de la rama main a la rama login, git nos informará de un conflicto. Dentro de VSCode podremos ver las dos opciones con las que contamos.

Seleccionamos que queremos mantener los cambios que hemos hecho desde la rama \*\*main\*\*. Después únicamente debemos hacer un commit para guardar la correccción de ese conflicto.

## GIT STASH

Realiza algún cambio en culquier fichero de la rama \*\*login\*\*, intenta moverte a la rama \*\*main\*\* y verás como no permite cambiar a otro rama si todavía no has ido hecho un commit de esos cambios.

Si no queremos hacer un commit porque el código todavía no es correcto o no es totalmente funcional, podemos utilizar el siguiente comando:

```

git stash

```

De esta forma no se guardarán de manera definitiva, pero podremos recuperarlos.

Puedes comprobar los stash creados con:

```

git stash list

```

Comprueba que ahora si puedes moverte a la rama main.

11. Añade una captura de pantalla con la lista en la que aparezca el stash.

Vuelve a la rama login y verás que no están guardados los últimos cambios que hiciste. Si quieres recuperarlos y continuar por donde te habías quedado debes utilizar:

```

git stash pop

```

Por último haz un commit en el que incluyas estos cambios.

## REINTEGRACIÓN DE RAMAS

Ahora vamos a integrar la rama de login dentro de la rama \*\*main\*\*, es decir, un merge de la rama login a la \*\*main\*\*. Comprueba que los commits hechos en la rama \*\*login\*\* se han integrado en la rama \*\*main\*\*.

## ELIMINACIÓN DE RAMAS

Por último vamos a eliminar la rama de \*\*login\*\* que no será necesaria, puesto que todos los cambios que había hecho en la rama de \*\*login\*\* ya están en la rama \*\*main\*\*.

```

git branch -d login

```

Comprueba que ya no está la rama:

```

git branch

```

12. Adjunta una captura de pantalla con las ramas eliminadas.





git branch

## PROPUESTA

Vuelve al primer commit con un reset hard y después vuelve al último commit en el que se había integrado la rama login dentro de la master.