COMANDOS ESENCIALES GIT

# COMIENZO. ESTOS TRES PASOS SON IMPORTANTES:

PRIMERO:

**git config --global user.name "Fulanito Setanito"**

**git config --global user.email "<fulanito@taltal.com>"**

* Configura el nombre y el correo de la persona para que sea más fácil identificar los cambios que hacemos cada uno.

SEGUNDO:

**git init**

* Inicializar repositorio vacío y en local (crea una carpeta ".git", oculta se supone, en el directorio en el que nos encontremos con la terminal y a partir de dicha carpeta tendremos creado nuestro repositorio y podremos crear otras carpetas (mkdir), archivos (touch), etc.). NOTA: Tenemos que estar en el directorio/carpeta en la que queramos crear nuestro repositorio y tener todo el proyecto con el que trabajemos y, una vez ahí, ejecutamos el "git init".

TERCERO:

**git clone https://github.com/EGC-G2-Trabajo-1718/RedesSociales.git**

* Clona todo el contenido del repositorio remoto para tener una copia en local.

CUARTO: Una vez tengamos ya nuestro repositorio local creado e inicializado, tendremos que crear los tres tipos de ramas con las que trabajaremos con git. Se explicará en que consiste cada una más adelante. La rama MASTER ya viene creada por defecto, con lo que solo tendríamos que crear DEVELOP y HOTFIX

**git branch develop**

* Crea la rama que usaremos para llevar a cabo todo el código. Esta es ÚNICA, al igual que MASTER.

**git branch hotfix-x**

* Crea una rama para llevar a cabo la resolución de un fallo donde “x” es el número de issue del github sin almohadilla. Ejemplo: hotfix-13. **ESTA RAMA, COMO ES OBVIO, SOLO SE CREA CUANDO SE LLEVA CABO LA RESOLUCIÓN DE UN FALLO DE IMPLEMENTACIÓN QUE HAYAMOS DETECTADO POSTERIORMENTE.**

# AÑADIR NUEVOS CAMBIOS AL REPOSITORIO LOCAL Y SUBIRLO AL REPOSITORIO REMOTO

PRIMERO: ANTES DE HACER NINGUN ADD O MODIFICAR ALGO, HAY QUE HACER PULL SIEMPRE QUE QUERAMOS HACER CUALQUIER CAMBIO, PARA REDUCIR CONFLICTOS

**git pull https://github.com/EGC-G2-Trabajo-1718/RedesSociales.git <rama>**

* Este comando trae a nuestro repositorio local que creamos antes, todo lo que haya subido en nuestro repositorio remoto. En "rama" tiene que ir el nombre de la rama de la que queremos obtener los archivos: master, develop o hotfix. Se explica cada rama a continuación:
  + **master:** **es la rama más importante**, en ella ira todo el código limpio de errores y listo para las entregas. Desde esta rama nos traeremos cada entrega para seguir desarrollando hasta la siguiente. Ejemplo: Estamos en la milestone 2, la acabamos, la entregamos al Benavides, para la milestone 3 tenemos que tener lo de la 2 más un incremento. Pues nos traemos lo que teníamos hasta la milestone 2 y seguimos trabajando desde ese proyecto.
  + **develop:** es la rama donde llevaremos a cabo toda la codificación de features (funcionalidades) y todo lo que tenga que ver con el proyecto en sí. Salvo la corrección de errores.
  + **hotfix:** esta rama solo la usaremos para la corrección de algún fallo que hayamos detectado, separando el código que estemos intentando arreglar con el resto de funcionalidades del proyecto. Una vez solucionado el fallo, se coge el código corregido y se subirá a develop. NUNCA A MASTER (master solo se usa para la entrega final y para obtener todo el código de cada entrega y seguir trabajando para incluir nuevas funcionalidades).

**El flujo de trabajo de las ramas es el siguiente: MASTER <--> DEVELOP <--> HOTFIX**

SEGUNDO: Una vez hayamos hecho el pull y tengamos todo del repositorio remoto, tendremos que hacer un add de los archivos que queramos subir del siguiente modo:

**git checkout develop**

* Cambiamos a la rama "develop", que es la rama donde programamos, a no ser que estemos corrigiendo un error, cuyo caso la rama a la que cambiaremos seria “hotfix”.

**git add archivo.php**

* Añade archivo por archivo a la pila de commit.

**git add .**

* Añade todos los archivos que hayamos creado/modificado en nuestro repositorio local, a la pila de commit.

El comando "add" no es hacer un commit aun, solo los añade a una pila para la posterior confirmación, esta pila se llama "STAGIN INDEX". La forma de confirmar esos cambios definitivamente es mediante un commit. No obstante, antes del commit, es bueno que veamos los cambios que hemos hecho dentro del "stagin index" de esta forma:

**git status**

* Obtenemos por colores las diferentes modificaciones que hemos hecho a nuestros archivos en local. Bien si hemos creado nuevos archivos, los hemos modificado o incluso borrado.

Posteriormente, una vez identificamos los cambios y de qué tipo son cada uno, ya podemos confirmarlos todos mediante un commit:

**git checkout develop**

* Cambiamos a la rama "develop", que es la rama donde programamos, a no ser que estemos corrigiendo un error, cuyo caso la rama a la que cambiaremos seria “hotfix”. Esto es solo para asegurarnos que estamos en esa rama y no en “master”, por ejemplo.

También podemos ver que ramas tenemos en el repositorio y en cual nos situamos (la que tiene asterisco) con el comando "**git branch**".

**ANTES DE LLEVAR A CABO NINGUN COMMIT, NI SIQUIERA EMPEZAR A PROGRAMAR FUNCIONALIDADES ESPECIFICAS, TENEMOS QUE CREAR UN ISSUE EN GITHUB POR CADA FUNCIONALIDAD A IMPLEMENTAR**:

Crear widget para los últimos tweets de twitter, integración con RSS, corrección de fallo de autenticación en Facebook...ETC

TERCERO: Llevaremos a cabo el commit:

**git commit**

* Se nos abrirá un editor de texto donde tenemos que detallar el commit que queremos llevar a cabo. Este commit, al estar situados en la rama "develop", se subirá solo ahí, que es lo que queremos al fin y al cabo.

Si ya tenemos la issue creada en github, podemos llevar a cabo el commit y el formato de cualquier commit que hagamos es el siguiente:

*Un ejemplo*

**Título del commit (Muy breve):** Fix: redirección errónea tras emitir voto.

<ESPACIO EN BLANCO>

**Cuerpo del commit:** Después de que el usuario emitiese su voto este era redireccionado a una URL no existente. Ahora el usuario es redireccionado al panel principal.

<ESPACIO EN BLANCO>

**Pie del commit:** Closes #<número de la incidencia en GitHub>

CUARTO: Subimos todos los cambios.

Ya que tenemos la confirmación de todos nuestros cambios en nuestro repositorio local (solo tenemos los cambios cada uno en su pc), falta subirlo al repositorio remoto, para que el resto de miembros del equipo podamos hacer un pull y traernos esos cambios.

Para subir los cambios al repositorio remoto basta con este comando:

**git push https://github.com/EGC-G2-Trabajo-1718/RedesSociales.git <rama>**

* Sube todos los cambios a la rama del repositorio remoto. Es lo mismo que un "pull", solo que pull obtiene todos los archivos del repositorio remoto, y "push" los sube.

# ELIMINAR ARCHIVOS DEL STAGIN INDEX (HEMOS HECHO EL COMMIT A UN ARCHIVO QUE NO ERA EL QUE QUERIAMOS SUBIR)

**git reset HEAD archivo.php**

* Elimina cada archivo que le digamos del stagin index (pila de commit), que lo elimine del stagin index significa simplemente que lo descarta y no sube ese archivo modificado, NO LO BORRA DEL SISTEMA.

# “TRACKEAR” ARCHIVOS MEDIANTE EL STAGIN INDEX (NO LE HACEMOS SEGUIMIENTO (AÑADIR A STAGIN INDEX) A UN ARCHIVO QUE SI QUEREMOS QUE SEA “TRACKEADO”)

**git checkout -- archivo.php**

* “Trackea” los archivos que le digamos para que se añada a la pila de commit (stagin index).

# EXAMINANDO EL REGISTRO DE COMMITS (LOG)

**git log**

* Muestra un histórico de todos los commits que se han llevado a cabo en el repositorio.

**git log --oneline**

* Muestra un histórico de todos los commits que se han llevado a cabo en el repositorio en una sola línea de forma mucho más resumida y fácil de ver.

**git log --oneline --decorate**

* Muestra un histórico de todos los commits que se han llevado a cabo en el repositorio en una sola línea de forma mucho más resumida y fácil de ver, y además, muestra con que ramas de destino se han creado esos commits.

# ACTUALIZAR EL CONTENIDO DE UNA RAMA CON EL DE OTRA (MERGE)

Pongamos la situación en la que tenemos que hacer una entrega (milestone) definitiva, es decir, tenemos que llevar todo nuestro código de la rama Develop a la rama Master.

PRIMERO: Cambiamos a la rama Master, que es donde queremos juntar todo:

**git checkout master**

SEGUNDO: Unimos todo lo que tenga Master con todo lo que tiene Develop actualmente:

**git merge develop**

# SOLUCIONAR CONFLICTOS EN LA MODIFICACION DE LA MISMA LINEA EN ARCHIVOS (CONFLICTO GIT: BOTH MODIFIED)

Pongamos la situación de que Fulanito está trabajando con este código PHP y añade como nueva línea la línea 2:

|  |
| --- |
| **archivo.php**  1. for ($i = 1; $i <= 10; $i++) {  **2. if $i%2==0:** 3.     echo $i;  4. } |

Este código lo único que hace es recorrer una lista de números del 1 al 10 e imprimir por pantalla solo los números pares. Fulanito coge este archivo le hace el “add”, lo “commitea” y le hace el push.

Mientras tanto, Setanito, que también tenía ese archivo ya descargado del repositorio **pero sin esa línea 2 nueva**, coge el mismo archivo y en vez de añadirle: “if $i%2==0”, le añade que imprima los impares: “if $i%2!=0”. Al igual que Fulanito, Setanito hace el “add”, “commitea” ese archivo y le hace push.

Posteriormente Setanito hace merge a la rama en la que está trabajando también Fulanito.

En el momento del **merge**, Git va a reportar un CONFLICTO de doble modificación “both modified”. Esto se produce porque Git tiene registrado que se ha cambiado la línea 2 de distinta forma en 2 commits diferentes (imprimir pares e imprimir impares). Con lo que si abrimos ese archivo.php tendríamos lo siguiente:

|  |
| --- |
| **archivo.php**  1. for ($i = 1; $i <= 10; $i++) {  **2. <<<<<<<<<<<< HEAD**  3. if $i%2==0:  **4. ============**  5. if $i%2!=0:  **6. >>>>>>>>>>>> develop** 7.     echo $i;  8. } |

Como se puede comprobar, en las líneas 2 y 6 se ve claramente de que rama procede el cambio original (2), y la rama de la que se quiere obtener el código modificado (6).

Este conflicto se resuelve de forma sencilla. Borramos las anotaciones generadas por git, las de las líneas 2, 4 y 6, llegan a un consenso Fulanito y Setanito de implementar una de las dos líneas. Quedando así el archivo, ya que Fulanito ha visto con buenos ojos el cambio de su compañero.

|  |
| --- |
| **archivo.php**  1. for ($i = 1; $i <= 10; $i++) {  **2. if $i%2!=0:** 3.     echo $i;  4. } |

Una vez tenemos el archivo con el conflicto ya resuelto, lo guardamos y le hacemos un “add”.

Automáticamente, git detecta que se ha resuelto el conflicto y **no es necesario crear un nuevo commit para la resolución del conflicto,** basta con el “add”.