**GIT**

**--**: Palabra completa

**-**: Abreviatura

F1 en el VSCode es para abrir la ayuda

**git –version**: Versión de git

**git help**: Ayuda

**git config –global user.name “Fernando Herrera”** -> Establece el nombre de usuario

**git config –global user.email “pepe@es.es”** -> Establece el correo que se utiliza

**git config –global –e ->** Muestra y permite modificar el usuario e email. Para grabar cambios y salir es: :wq! (escribir y salir. Para editar se pulsa a y para salir de la edición ESC)

**git init** -> Inicializa un repositorio

**git add** . -> Añade los archivos al stage

**git commit –m “Primer commit”** -> Crea el snapshot y realiza el commit, creando la copia y pudiendo volver en el tiempo. Si se pone el –am, hará el add y el commit (solo funciona si se ha hecho antes el git add, es decir, se ha hecho el seguimiento).

**git status** -> Para conocer el estado de los cambios en una rama.

**git checkout** -- . -> Volver al commit anterior. Muy parecido a  **git reset --hard**

**git reset <nombredelarchivo>** -> eliminar un archivo antes de hacer el commit.

Soft =>

* "elimina" los commits posteriores al commit al que estás haciendo el reset
* conserva los cambios en el stage area
* conserva los cambios que tengas en tus archivos (working directory)

Mixed =>

* "elimina" los commits posteriores al commit al que estás haciendo el reset
* Deshace los cambios en el stage area
* conserva los cambios que tengas en tus archivos (working directory)

Hard=>

* "elimina" los commits posteriores al commit al que estás haciendo el reset
* Deshace los cambios en el stage area
* Deshace los cambios que tengas en tus archivos (working directory)

Se puede usar **git revert** que es parecido pero difiere en que puedes deshacer un commit concreto

**git config core.autocrlf true** -> Para arreglar el warning de CRLF.

**git branch** -> Indica la rama en la que nos encontramos.

**git branch –m master main:** Se cambia el nombre de la rama master a main.

**git config –global init.defaultBranch main ->** para que siempre que se cree un repositorio el raiz sea main o no master

**git config –global alias.s “status –short”** -> Se crea un alias s para el comando git status –short. Se podrá usar: git s. Ej:

log --graph --abbrev-commit --decorate --format=format:'%C(bold blue)%h%C(reset) - %C(bold green)(%ar)%C(reset) %C(white)%s%C(reset) %C(dim white)- %an%C(reset)%C(bold yellow)%d%C(reset)' --all

**git diff –staged** -> Para saber los cambios con el staged.

**git commit --amend -m "Instalaciones actualizadas 2"** -> Cambiar el nombre de un commit.

**git reset --soft HEAD^** -> Sirve para mover al commit anterior… Si pones un ^2 te moverá al commit antes del anterior.

**git reflog** -> Para volver a cualquier estado por el que hubiéramos pasado. Con este comando recuperamos todos los hash

**git mv destruir-mundo.md salvar-mundo.md** -> Renombrar el fichero

**git rm salvar-mundo.md** -> Eliminar el fichero

**git branch <nombre-rama> ->** Crea una rama con ese nombre. Para moverse a una rama get checkout <nombre-rama>. También se podría crear una rama y moverse con **git checkout -b <nombre-rama>**

**git branch –a** nos realiza el listado de todas las ramas, incluidas las remotas.

**git merge <nombre-rama>** -> Mezclará la rama dónde nos encontramos con la rama indicada. Se puede dar varios casos:

* + **Fast-foward**: No hay cambios en la principal y los cambios en la rama pueden mergearse con la rama master.
  + **Recursive (automático)**: Hay cambios en varios archivos pero no hay conflictos.
  + **Conflicto**: Hay que seleccionar el cambio deseado ya que se ha cambiado el mismo fichero.

Para borrar la rama:

**git branch -d rama-villanos –f** (el menos f es para forzar el borrado si hubiera cambiamos no comiteados)

**git tag <nombre-tag>:** Para crear tag de una determinada release. Se borran con el atributo –d. Lo más común es crear una versión con la siguiente notación: **git tag -a v1.0.0 [<hash del commit>] -m "Versión 1.0.0 lista"** (el hash del commit es opcional y se pone si queremos poner una versión a un commit que tengamos en la historia). Para subir los tags es: git push --tags

**git stash ->** Permite llevar tu trabajo a un área, de manera que se pueda volver al commit anterior y hacer los cambios que sean necesarios. Para recuperar los cambios del git stash, se usa git stash pop. El git stash clear borra todo el contenido del stash.

Por otro lado, git stash apply 'stash@{1}' permite volver a un stash concreto. Con git stash clear borramos todos los stash

**git rebase** -> Es otra opción para hacer un merge entre ramas pero, en este caso, recopila uno a uno todos los merges de una rama y los aplica en la otra.

**git rebase -i HEAD~4** -> Nos permite unificar dos commit en uno solo. Entra en un formato edición dónde tendremos que pulsar a para modificar y pondremos squash en la rama deseada y que queremos unificar con la anterior

Para modificar los nombres de los commit, se puede usar el git rebase anterior, marcar con r los commit a modificar, y el editor nos pedirá ir modificando uno por uno.

Para editar, entramos de nuevo en el modo interactivo, pulsamos a y marcamos con e los commit a modificar. Una vez hecho esto, podemos volver al commit anterior con **git reset HEAD^.** En ese momento ya se pueden hacer los commit deseados y, por último, pararemos la edición poniendo git rebase –continue.

**PUSH Y PULL**

Son comandos para subir a la nube o para bajar un proyecto de la nube. Hay varios hosted services, como BitBucked y GitHub. GitHub es una plataforma de desarrollo colaborativo de software para alojar proyectos.

Para agregar una referencia remota se usa el comando:

**git remote add origin <ruta del proyecto en github>**

Para saber los orígenes:

**git remote –v**

**git push –u origin master ->** Dónde origin es el nombre del repositorio, master es la rama a enviar y –u sirve para que la próxima vez que se haga un push, no sea necesario indicar la rama. Si hacemos el push de una rama es:

**git push --set-upstream origin rama-villanos**

Para subir los tags hay que poner: **git push –tags**

Para obtener los cambios del repositorio remoto, hay que hacer:

**git pull**

**git config –global pull.ff only** -> Los pull solo van a ser fast foward. En caso de conflicto, se avisa.

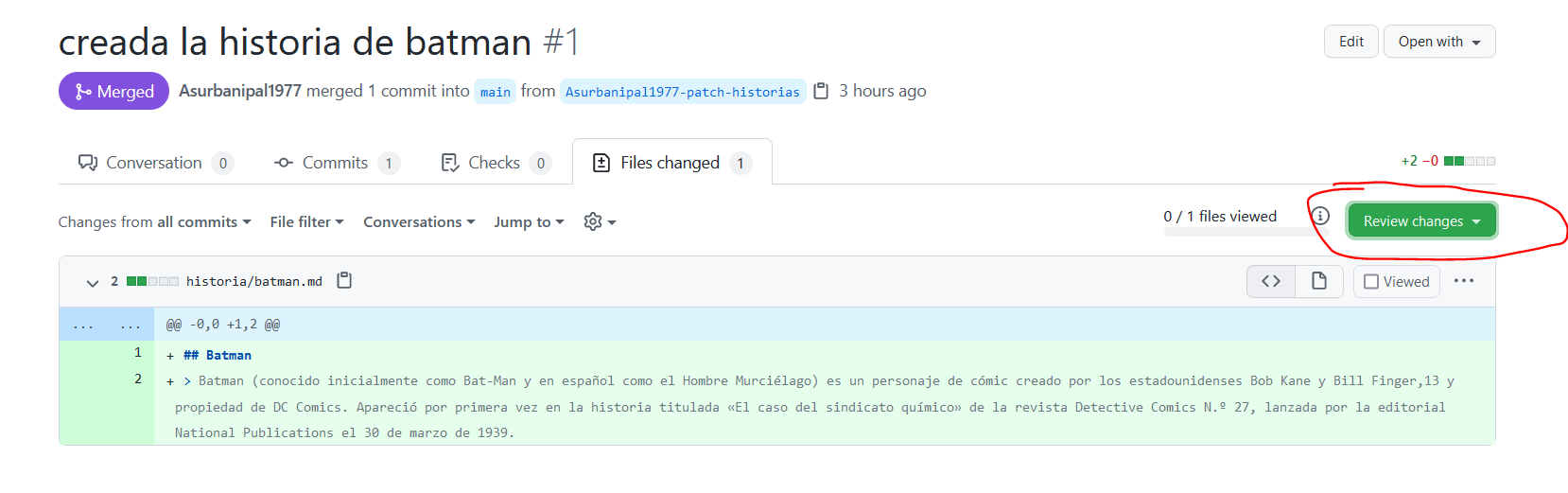
**git config pull.rebase true ->** Establece localmente para el repositorio en el que nos encontramos una estrategia de pull.rebase

**git clone <ruta del repositorio remoto>:** Para clonar en local un repositorio remoto

**git pull request ->** Se solicita al propietario del repositorio remoto que se intente unificar mi rama con el main

**git fetch ->** Permite decir al git local si hay cambios en el repositorio remoto. En diferencia con git pull, no se trae los archivos.

Cuando tenemos un pull request en nuestro repositorio remoto podemos rechazar los cambios si vamos a “Files changed” y después al botón “Review Changes”



**FORK**

Permite clonar un repositorio remoto en un lugar dónde nosotros tengamos acceso.

Para traernos los datos de un repositorio dónde tengamos acceso de lectura pero no de escirtura, podemos definir otro repositorio remoto:

**git remote add upstream <dirección de repositorio>**

Para hacer el pull se hace: **git pull upstream**

**BORRADO DE RAMAS INNECESARIAS**

**git remote prune origin** -> Permite limpiar las ramas de local que ya no están en remoto

Si se elimina y se quiere recuperar, si se marcó con un tag, se puede usar:

**git checkout v1.0.0**

**git checkout –b rama-kitkat**

**git push --set-upstream origin rama-kitkat**