***GIT con amend***

Con este comando podemos corregir los cambios ya guardados en la última copia de seguridad de nuestro entorno local; es decir, corregir o hacer cambios puntuales sobre nuestro último *commit* (sobreescribir información, con intención de modificar parcial o completamente todo el contenido, sobre él)*;* sirve en los casos especiales donde, por *error,* hemos guardado un *commit* que; en principio, debía agregarsele más información o, en su defecto, agregarle otro tipo de información.

Entonces, para superponer unos nuevos cambios requeridos o deseados sobre el reciente *commit* ya guardado*,* ante todo, *no se asuste.* Haga sus respectivos cambios con toda normalidad, posteriormente les hace un seguimiento con *git add*; y, finalmente, ejecuta el comando: ***git commit --amend***.

*Git commit amend* lo que hará es superponer la nueva información que está cargando sobre la información inmediatamente anterior que ha sido guardada en el último *commit*; pero, ojo, manteniendo ese mismo identificador de *commit*; luego, ya queda a disposición propia, definir si desea cambiar el mensaje referencial con el que inicialmente se guardó esa última copia de seguridad (commit).

***Git Reset***

El comando que nos permite viajar entre commits es el siguiente: ***“git reset + identificador”***. Por ejemplo, si resulta que actualmente estamos trabajando con la última copia de seguridad guardada: “48c2e46”, y queremos irnos de nuevo a la copia de seguridad anterior: “2f15ade”, debemos ejecutar el siguiente comando, tal que así: “git reset 2f15ade”. Y listo! Apenas presionen *enter* en pantalla se verá reflejado, de nuevo, sólamente lo que teníamos registrado en el identificador “2f15ade” y no en el identificador 48c2e46 **(anulando así todas las copias de seguridad situadas por encima del identificador “2f15ade”); es decir, se verá ahora el archivo “2f15ade”, lo que equivale a decir: antes de los cambios del archivo “48c2e46”.** Esto nos debe hacer reflexionar muy a fondo cada vez que queramos proceder de esta manera, pues estamos ante un comando destructivo, **especialmente cuando estamos trabajando de forma compartida nuestro repositorio. Por qué? En el fondo el programa que está haciendo?** (y es en esto en lo que tiene que tener cuidado), **en el fondo el comando “reset” lo que está haciendo realmente es eliminar, de forma temporal, todos los “commits” que habían sido creados justo después del commit “2f15ade”.**

*Git reset*, por defecto, lo que hace es, en resumidas cuentas, eliminar las copias de seguridad (“commits”) y, además, situarte en una copia de seguridad “vieja”. (la que has seleccionado y ha removido todas las copias de seguridad posteriores a ella, ese “commit” elegido con “git reset”): esto sin dejar rastro alguno en el “staging area” (area temporal); entonces, si por ejemplo, desea crear una nueva copia de seguridad a partir de la versión que venía trabajando en esta “vieja” seleccionada; debe primero hacerles seguimiento (con git add) a los cambios que ha hecho; y, posteriormente, ejecutar ahora así el comando “git commit” para guardar una nueva copia de seguridad inalterable.

Sin embargo, git reset, también hace uso de funciones más especificas, se compone por el uso de *elementos* propios como: “hard” y “soft”. Se ejecutan así: ***“git reset --hard”*** y ***“git reset --soft”***. Veamos.

**“hard”:** elimina el “commit” y, además, elimina de forma totalitaria todos los cambios que se habían hecho en él; es decir, no es que deje los cambios por fuera del stage (o area temporal) como sí lo hace por defecto “git reset” (pero al menos con *git reset* tiene los cambios ahí aún, sólo que le toca hacerle seguimientos con “git add” si desea recuperarlos total o parcialmente); en cambio, con “git --hard reset” simplemente los cambios se quedan por fuera de todo (ya no existen), se quedan por fuera de la inicialización propia del git, de su repositorio y, en suma, de todo su entorno local. Ahora, **“soft”** es mucho más suave en su tratamiento efectivamente: también elimina el “commit”; pero, esta vez, los cambios no sólo se quedan en la inicialización del repositorio (como lo hace “git reset” por defecto); sino que también mantienen su seguimiento; es decir, los cambios del “commit” recientemente eliminado se mantienen dentro del staging (area temporal), no los borra. Y, no le toca, por ejemplo, ejecutar el comando “git add” porque los cambios ya están ahí (esperando por ser subidos nuevamente en una nueva copia de seguridad inalterable, en un nuevo *commit* por ejemplo). En resumidas cuentas, “soft” sólo deshace las operaciones del “git commit”, pero sin eliminar las operaciones del “git add”.

***Git con Grep y Log***

***git grep*** nos permite buscar una palabra en especifica que tenga algún archivo *de texto plano o para código* en todo nuestro repositorio local ya inicializado con Git; para ser más precisos, lo que hace ***grep*** es buscar en todos los archivos, según una rama y un directorio de trabajo donde esté situado, una palabra que usted le haya pasado; tal que así, por ejemplo: ***git grep “palabra”****.* Entonces, *git grep* nos señala la dirección o destino de todos los archivos que contengan esa palabra en cuestión, en forma de lista.

Ahora bien, este comando puede ser más preciso en su motor de busqueda si le especificamos más parametros. Por ejemplo, si deseamos que, además de decirnos las rutas de los archivos que contienen cierta *palabra,* queremos que nos señale en qué linea, de la totalidad de los archivos, tomando uno a uno de ellos, fue usada... podemos ejecutar el mismo comando de la siguiente forma: ***git grep -n “palabra”.***

Adicionalmente, si deseamos saber cuántas veces usamosla *palabra* en cuestión, para cada uno de los archivos, ejecutamos el comando: ***git grep -c “palabra”.***

***pd. Es una buena práctica que las palabras vayan entre comillas ciertamente.***

Ahora, si deseo por ejemplo buscar la palabra, no en los archivos de texto plano; sino, en el historial de *commits:* más puntualmente en sus *mensajes de referencia*, tendría que correr el siguiente comando: ***git log -S “palabra”***. Este comando no solamente se fija de los mensajes de referencia de los *commits;* sino, también, de los nombres de las ramas: en general, se fija de cualquier cosa que dentro del historial de *commits* mencione a la palabra en cuestión.