***Rebase en GIT***

Un ***rebase***consiste en incorporar el contenido de una rama de desarrollo en la rama *master, o viceversa,* de manera “silenciosa”; es decir, hacer como que no pasó nada nunca ni que tuvimos que crear una rama de desarrollo nueva para, posteriormente, incorporarle sus cambios en rama *master* (si éste fuera el caso). Para ser más precisos, la rama realmente no se borraría (la que exportaría sus cambios guardados)*;* sin embargo, el historial de *commits* de esa rama, por ejemplo la rama de desarrollo, no se exportarían a *master* al hacer un *rebase* en master de la rama de desarrollo en cuestión; es decir, al incorporar en *master* los cambios guardados o *commits* de la rama de desarrollo; tal que: ***git rebase “rama\_de\_desarrollo”.*** Este último comando lo que hace es traernos los avances registrados en la *“rama\_de\_desarrollo”* a *master;* pero, sin traernos su historial de *commits* (el historial de *commits* de la rama de desarrollo en cuestión)*.*

Una **buena** **práctica** para antes llevar a cabo la realización de un *rebase* a una rama de desarrollo (desde *master),* es hacer el *rebase* de *master,* primeramente, en la rama de desarrollo en sí: esto se hace para asegurarnos de que nos mantenemos actualizados de la información que contiene, parcialmente, la rama *master (y no perdernos nada de ella)* más allá de los propios cambios guardados que ya tenga la rama de desarrollo; sí, precisamente, los cambios que deseamos exportar, con un *rebase,* a *master* posteriormente.

Ahora si por ejemplo desea eliminar, de forma definitiva, cualquier registro de la existencia de una rama de desarrollo, que le ha pasado su contenido a *master;* usted, además de no tener un registro de los *commits* de la rama de desarrollo, al ser exportados sus cambios, en *master*; puede, adicionalmente, eliminar de forma definitiva la rama de desarrollo en cuestión, tal que así: ***git branch -D “rama\_de\_desarrollo”.***

*Entonces, ya con esto, ahora sí, es como si nunca hubiera pasado absolutamente... nada más allá de la manipulación sobre una única rama: la rama master... Eso es un* ***rebase:*** *hacer fusiones entre ramas con la posibilidad de no dejar rastro o evidencia alguna; incluso, ni de la rama auxiliar que luego pasó su contenido a master,... si así eso desea.*

Se recomienda sólo hacer *rebases* en GIT, no se sugiere ejecutar este tipo de dinámicas en repositorios remotos.

*Pregunta fuera de contexto: ¿Qué hace* ***git******stast*** *y para qué sirve?*

***Git clean***

*Git clean* es un comando que nos permite eliminar todo aquello que por error ha sido creado; o bien, porque eso que fue creado, no nos interesa a lo sumo que haga parte de nuestro proyecto... *aunque incluso haya sido consecuencia de algún proceso ligado al mismo*; entonces, mientras esté en nuestro directorio de trabajo o proyecto inicializado con *Git*, el archivo en cuestión, podría ser eliminado con el comando ***git clean:*** es como la manera de ejecutar la acción de *eliminar* algodesde los comandos propios de Git en la terminal.

Hago mención a “archivo en cuestión” porque este comando no puede eliminar directorios, sólo archivos; y, puntualmente, archivos que aún no han sido *trackeados (porque realmente no te interesa hacerles un seguimiento; antes, desearías eliminarlos).*

Entonces, este comando para que corra *(borrar los archivos sin seguimiento o no agregados aún con “add”),* el comando como tal se debe ejecutar de la siguiente forma:***git clean --dry-run;*** entonces, luego de ejecutado el anterior comando, *Git* quiere estar plenamente seguro de que usted realmente sí desea eliminar de forma permanente esos archivos *no trackeados* en cuestión; para confirmar su decisión, debe ejecutar este posterior comando: ***git clean -f***

*Excepciones o limitaciones del comando* ***Git clean:***

**1.** *Git clean* no puede eliminar los archivos incorporados en *gitignore;* pues, por la misma lógica que está desarrollada detrás de la función *gitignore.* Cuando agregamos un archivo en la lista de *gitignore,* le estamos diciendo a *Git* que ignore completamente dichos archivos; es decir, estamos limitando a *git* de tal manera que no pueda ejecutar ninguna acción sobre ellos (al menos que uno manualmente los manipule, pero *git* como tal y sus comandos no pueden ejecutar ninguna acción sobre ellos).

**2.** *Git clean* tampoco puede eliminar carpetas; entonces, no se sorprenda si en ocasiones se encuentra que hay archivos que no pueden ser eliminados por *git clean* al enterarse que, esos archivos, están dentro de una carpeta.

***Git cherry pick***

Pueda que en ocasiones usted desee importar, en su rama de trabajo actual (convengamos que es rama *master),* un *commit* de otra rama de desarrollo; pero, atención, un *commit* en particular, y no precisamente la última copia de seguridad registrada en dicha rama de desarrollo (es decir, el último commit), no. *Cómo se haría esto?* Es decir, si deseo traerme, a mi rama *master,* cualquier *commit* de otra rama de desarrollo sin importar en qué orden fue éste registrado en todo el historico de ellos (en todo el historico de *commits*),... Qué comando de *git* debería poner en práctica? Pues bien, el comando que debería usar sería:   
***git cherry-pick*** + ***el identificador del commit, de la otra rama, de mi interés (el que deseo importar);*** esto, claro, ya estando situados en la rama de trabajo donde deseamos importar un *commit* de otra rama; es decir, para nuestro estudio, ya estando situados en la rama *master.* Y listo, luego de traer ese *commit*, como ya lo trajo en sí mismo, no es necesario *guardarlo* (guardar ese cambio), el cambio ya estaría guardado en rama actual; es decir, ya no es necesario crear otro *commit;* pues, ya se importó el de su interés y se guarda de forma automatica en la rama que lo ha importado, la rama de trabajo actual *master*.

Ahora bien, algo que no comentamos: en principio, como nos interesa traernos el *commit* de otra rama; se sugiere que primero, entonces, se ubique en dicha rama y ejecute el comando: *git log --oneline* para ubicar el commit de su interés y que desea exportar desde esa rama a la rama *master;* en concreto,tomar los avances puntuales de un commit en especifico, de una rama, y llevarlos a otra (es ésta la función de *git cherry pick*).

Luego posteriormente, si desea tomar todos los cambios guardados de la otra rama (importar más precisamente el último *commit* de ella), simplemente haga un *merge* y listo; eventualmente esto le generará un *conflicto,* simplemente acepte el *merge* así.El conflicto realmente lo genera esa misma línea de código, o esas mismas líneas de código, que han sido registradas en ambas ramas al haber ejecutado un *git cherry pick* en su rama de trabajo.