<https://git-scm.com/doc>

* **Git --version** : Para validar la versión que tenemos instalada de git.
* **(--)** Significa que luego va una palabra completa y **(-)** que viene una abreviatura.
* **Git Help** : para ver la ayuda de Git.
* ***Git help + (un comando):*** nos aparecerá la info de ese comando.
* ***git config --global user.name "Dominic Minaya":*** Se configura de manera local el usuario que se supone que está trabajando en la pc y haciendo cambios local.
* ***git config --global user.email “”*** : Se configura el correo de la persona de manera global.
* ***git config --global -e:*** Para ver la config actual y poder actualizar.
* ***Git init:*** Inicializa el repositorio.
* ***Git Status:*** Da info de los commit, la rama en la que nos encontramos y otras informaciones relacionadas con el status del proyecto.
* ***Git add (name file):*** Indica que un archivo se debe tomar en cuenta al momento de hacer el commit.
* ***Git add .*** : Indica que todos los archivos que aun no han sido trackeados hay que tomarlo en cuenta al momento del commit.
* ***Git reset (name file) :*** Indica que ese archivo debe ser sacado de lo que se tomará en cuenta para el commit.
* ***Git commit -***m”Message” : Es como hacer una fotografía de lo que hay en el proyecto en ese momento el **-m** es para indicar el mensaje del commit.
* ***git checkout -- . :*** Esto retorna el proyecto a como está en el último commit.
* ***Git Branch :*** Nos dice en que rama estamos trabajando.
* ***Git Branch -m master main***: Esto renombra el nombre de una rama, en este caso Master ahora se llama Main.
* ***Git config –global init.defaultBranch main*** : De manera goblal cambiamos la rama por defecto.
* ***Git commit*** -am”Message” : Esto se encarga de hacer el add y el commit al mismo tiempo, pero OJO solo funciona si ya el archivo está previamente agregado al stage y se le esté dando seguimiento.
* ***Git log***: Veremos los commit que hemo hecho y su hash que es único, además el autor y la fecha y el mensaje del commit.
* ***git add \*.html:*** Agrega todos los que estén asociados a esa extensión, pero solo van a entrar lo que estén en el root principal.
* ***Git add js/\*.***js: Agrega todos los asociados a (js) que estén en ese directorio.
* ***Git status - –***short : De manera mas corta muestra el status del repositorio.
* **git config --global alias.s status - –short :** para crear un alias y así podemos reducir la cantidad de lo que escribimos **(alias.s) s:** representa el nuevo alias y luego escribimos de que será el alias **(status - –short)** en este caso de esto.
* **git log --oneline :** aparece los commit de manera corta.
* **git config --global alias.lg "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" :** Un alias que recomienda Fernando para los logs para verlo corto y el tiempo.
* **git diff** : Agarrar y ver lo que se ha modificado.
* **git commit --amend -m "Instalaciones actualizadas”:** Esto permite actualizar el mensaje del commit . OJO solo lo hará al último commit.
* **git reset --soft HEAD^ :** Esto permite sacar el ultimo commit para así poder modificarlo. OJO

**git reset --soft HEAD^(n)** , podemos agregar 2,3,4 etc y lo que hará es buscar el numero antes del último para así sacarlo y poder modificar el commit, el –soft es para que no destruya nada.

* **git commit --amend :** Esto entra a un archivo que muestra los detalles del último commit y podemos modificar ahí mismo el nombre del commit.
* **git reset --soft f1063f3 :** Si le ponemos el hash también funciona pero debemos tener pendiente que el va a sacar a todos los commit a partir del hash que colocamos.
* **git reset --mixed 745a9d3**:Si le ponemos el hash también funciona pero debemos tener pendiente que el va a sacar a todos los commit a partir del hash que colocamos.
* **Git reset** --hard **745a9d3 :** :Si le ponemos el hash también funciona pero debemos tener pendiente que el va a sacar a todos los commit a partir del hash que colocamos,Pero debemos tener cuidado porque este destruye aunque se puede recuperar pero destruye en teoría el commit y si volvemos a hacer lo mismo y está destruido lo restaura y de esa forma restaura al tiempo **OJO** no es lo más recomendable.
* **Git reflog**: Vemos el historial en orden cronologico de todos los commit hasta lo que hemos sacado del stage.
* **git mv destruir-mundo.md salvar-mundo.md** : (mv) de mover pero como lo dejaremos en el mismo archivo lo podemos usar para cambiar el nombre.
* **git rm .\salvar-mundo.md :** Esto es para eliminar un archivo, pero para que el cambio se haga definitivo debemos hacer un commit.
* **Git reset --hard:** Si lo ponemos sin hash el lo que hace es que retorna al último commit parecido al git checkout --.
* **git branch** rama-vtriderillanos **:** Crear una nueva rama.
* **git checkout** rama-villanos: Para cambiar de rama
* **git merge** rama-villanos: De esa forma traemos lo de la rama (rama-villanos) a la rama donde estamos ubicado y trae los datos de la que le indicamos a donde estamos ubicados.
* **git branch -d** rama-villanos : Para borrar una rama
* **git branch -d** rama-villanos **-f**: Si git reclama porque hay cambios pendientes, pero aun así la queremos borrar le agregamos -f para forzar el borrado.
* **git checkout -b rama-villanos:** Crea una rama e inmediatamente nos mueve a ella**.**
* **git tag super-realese : *Para crear un tag*.** Es un mensaje o etiqueta que **hace** referencia a un commit y a todo el estado que se encontraba el proyecto en ese momento.
* **git tag :** Para ver todos los tags**.**
* **git tag -d super-realese :** Para borrar todos los tags.
* **git tag -a v1.0.0 -m "Version 1.0.0 lista”:** Para agregar también un tag, el **-a** el nombre del tag y **-m** el mensaje del tag.
* **git tag -a v1.0.0 4e809d4 -m "Version Alpha de nuestra app”:** Si queremos agregarlo a un commit en especifico
* **git show v0.1.0:** Ver lo relacionado al tag.
* **git stash :** Si no hemos hecho commit el va a guardar lo que hemos hecho hasta el momento para dejarlo como estaba y luego podemos retornarlo para seguir donde nos quedamos.
* **git stash list:** Ver la lista de los stash.
* **git stash pop :** Para retornar el stash y borra el stash , solo trae el primero y mueve la lista .
* **git stash clear:** Limpia todo los stash.
* **git stash save"Agregamos a loki en villanos" :** Guardarlo con nombre el stash**.**
* **git rebase master:** debemos estar en la otra rama y lo que hace es que mueve los commit de la otra rama que no se le ha hecho merge a un espacio temporal, para luego alinearlo en la misma línea con la master.
* **git rebase -i HEAD~4 :** para poder hacer rebase y unir o eliminar o ect el (**HEAD~4)** significa que tomará los últimos 4 commits y ya luego podemos hacer lo que entendamos. Si lo que queremos es unir donde ponemos la **s** debemos dejar la anterior la **p** porque el lo hace con el commit anterior.

**Git Hub**

**Save user and pass Windows**

[**https://docs.github.com/en/get-started/getting-started-with-git/caching-your-github-credentials-in-git#platform-windows**](https://docs.github.com/en/get-started/getting-started-with-git/caching-your-github-credentials-in-git#platform-windows)

**Nota:**

* Los commit deben ser muy relacionados con ciertas cosas que modifiquemos en el proyecto.
* Si creamos un folder y esta vacio git los ignora, pero si le agregamos dentro un archivo automáticamente lo reconoce pero si queremos que reconozca ese folder vacio le ponemos adentro un archivo que se llame **(.gitkeep)**.
* Aunque el git diff es útil es mejor hacerlo por el editor así uno ve exactamente línea por linea y puede comparar mejor es bueno tener en cuenta las líneas para saber lo nuevo.
* **.gitIgnore** para evitar que tome en cuenta ciertos archivos o folders. .gitIgnore es un archivo que va en la raíz del proyecto.
* **Cuando en el merge hace fast-forward significa que git logró agregar todo y no hubo conflictos.**
* **(HEAD)** Es el ultimo commit que se encuentra nuestra rama.
* **(Tag)** es un mensaje o etiqueta que **hace** referencia al último commit y a todo el estado que se encontraba el proyecto en ese momento. Es ideal hacerla cuando la app está en punto maduro y útil. Lo podemos asemejar a versiones de la app .
* **V1.0.0 :** El **1** ahí usualmente va el numero de la versión, el segundo **0** usualmente significa que se le agregó alguna funcionalidad y el tercer **0** la resolución de un bug que se encuentre en esa versión**.**
* **Stash:** Es muy útil cuando tenemos que subir a prod y ya estábamos haciendo cambios y para una manera rápida se guarda lo que teníamos y lo deja como el ultimo commit, pero no es recomendable llenar el repositorio de stash, el stash es como una caja donde guardamos ciertos cambios y después podemos ir a recuperarlo.
* **(q) =** Por ejemplo en git log para volver a ejecutar comandos.
* **Rebase** : Lo mueve a un área temporal y luego lo retomamos, sirve para **ordenar commits, corregir mensajes, unir commits y separar commits.** Si son commits muy viejo o ya hicimos push lo recomendable es dejarlo así.