Github

# Fundamentos

* Para saber la versión ponemos en consola **git –-version**
* Para la ayuda podemos dar **git –help**
* Para saber mas información sobre un comando en especial es **git help (commit)**
* Para avanzar más rápido dentro de la consola al revisar un documento o algo bastante amplio, apretar **q**
* Para hacer una configuración básica del correo y en general **git config –-global user.name “Cerbero”**
* Para el correo es lo mismo, pero se cambia por email **git config –-global user.email “red.cerberus640@gmail.com”**
* Para saber esta información que acabamos de ingresar debemos de teclear **git config –-global -e**

# Iniciando el proyecto

* Para inicializar un proyecto debemos de ir a la carpeta donde va a estar el repositorio y debemos poner **git init**
* Para saber el status y los archivos que no han sido subidos a ningún lado debemos de poner **git status**
* Para poder ver una versión más corta de status ponemos: **git status –-short**
* Con esta instrucción le decimos a git que queremos que este pendiente de los cambios en TODOS los archivos **git add .** (**git add index.html** para un solo archivo)
* Si por alguna razón, un archivo que no queríamos que estuviera listo lo ingresamos, lo podemos quitar de la lista poniendo **git reset (nombre del archivo a sacar de la lista)**
* Para dar commit y tomar un ‘screenshot’ de como esta el proyecto en este momento deberíamos poner **git commit -m “Primer Commit”** con esto le mandamos un mensaje con el commit.
* Para ver los logs de los commits damos **git log**
* Si en algún momento presentamos un error que nos menciona algo sobre los CRLF es por la interpretación de un carácter, y se quita poniendo este comando **git config core.autocrlf true**
* Para recuperar los cambios hechos hasta la ultima versión del commit por errores en archivos poner **git checkout -- .**
* Por si aparece un error de CRLF, es sobre un carácter y ejecutar **git config core.autocrlf true**
* Para hacer un commit de un tipo especial de archivo, poner **git add \*.png**
* Para agregar todos los archivos de una carpeta poner **git add css/**
* Para dar commit a todos los archivos excepto uno (ejemplo xml) ponemos **git add -A** que agrega todos los archivos modificados y después **git reset \*.xml**
* Agrega todos los txt de TODO el proyecto **git add “\*.txt”**
* Agrega todos los txt en el directorio actual **git add \*.txt**
* Agregamos todos los archivos **git add –all**
* Agrega los archivos que listemos separados por una coma **git add <lista de archivos>**
* Agrega todos los pdfs dentro de la carpeta **git add pdfs/\*.pdf**
* Para hacer un commit directo, sin antes pasar al stage ponemos **git commit -am “<Mensaje mal escrito>”**
* Para ver los logs mas cortos escribimos **git log –oneline**
* Para un detalle mas del log con decoradores escribimos g**it log –online –decorate –all –graph**
* Para manipular el detalle del status y ver el detalle solo de los archivos que podemos subir al stage ponemos **git status -s**
* Para saber solo los archivos y en que rama estamos ponemos **git status -s –b**

Alias

* Para un alias para git status –-short ponemos **git config –-global alias.s “status –-short”**
* Para configurar un alias para trabajar una función en concreto poner **git config –global alias.lg “log –oneline –decorate –all --graph”**
* Para agregar también el alias del status ponemos **git config –global alias.s “status -s -b”**
* Para poder ver el archivo de configuración de los shortcut, podemos ejecutar **git config –global -e**
* Para ver los alias sin abrir un editor de texto ponemos **git config –global –l**
* Para que git mantenga una carpeta vacia, tenemos que crear un archivo llamado .gitkeep
* Para agregar dos shortcuts ponemos 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"
* git config --global alias.s status –short
* git config --global alias.s status -sb

# Sección 3 : líneas de tiempo

* Para saber las diferencias del ultimo commit y los archivos modificados podemos poner **git diff**
* Si hemos puesto en el stage un cambio pero no recordamos que, podemos poner **git diff –staged**, y vemos los archivos que están en stage que tienen cambios
* Para quitar el archivo del stage principal ponemos **git reset HEAD README.md**
* Para revertir los cambios en un archivo en especifico y dejarlo como esta en el commit ponemos **git checkout – README.md**
* Para enmebndar un mensaje de commit mal echo podemos poner **git commit -–amend -m “<Mensaje>”**
* Para poder volver a hacer commit pero a un commit que ya existe (ósea se nos olvido agregar algo y ya habíamos echo el commit) utilizamod la sentencia **git resert –soft HEAD^** (si queremos otro que no sea HEAD que es el ultimo commit agregamos ^), si no funciona el ^, ponemos **~1** y cuando terminamos de hacer los cambios, volvemos a hacer el commit, es muy útil para cuando olvidamos hacer un cambio, ponemos el primer comando, luego revisamos que estén las dos MM una verde y una roja y damos de nuevo un git commit -am “<mensaje>”
* Para agregar un registro más completo en el repositorio (multilinea) ponemos el comando **git commit**
* Para volver a edita el ultimo commit como multilínea que vamos a enmendar ponemos **git commit –amend**

## Viajes en el tiempo resets y reflog

* El primer viaje es poner **git reset –-soft <numero del commit que queremos tener>**
* Para ir a cierto commit y eliminar todos los demás ponemos **git reset –mixed <id de commit al que queremos viajar>** pero recordemos que elimina todo los commits que se hicieron después de este commit, pero los cambios que se hicieron aun están en los archivos, ósea que podemos regresar al pinto inicial solohaciendo un commit
* Para eliminar definitivamente los cambios echos después de cierto commit debemos poner **git reset –hard <Id del commit al que queremos regresar>**
* Para poder ver TODO lo que se ha hecho desde git ponemos **git reflog**
* Y para volver al ultimo cambio que hicimos, seleccionamos el id de operación y hacemos un **git resert –hard <id de el punto que queremos ir del histórico>**
* Para hacer un cambio de nombre de archivo, sin afectar todas las operaciones que se han hecho sobre él, y es muy recomendable, ponemos  **git mv destruir-mundo.txt(antiguo nombre) salvar-mundo.txt(nuevo nombre**)
* Para poder eliminar un archivo debemos poner **git rm salvar-mundo.txt(archivo a eliminar),** podemos regresar el archivo de nuevo a la vida con un git reset --hard
* Al cambiar el nombre a un archivo, lo que reconoce git es que eliminamos el archivo y creamos uno nuevo, para que git reconozca nuestros cambios le damos un update y ponemos **git add -u** y le agregamos un **git add -A** para que reconozca que es un renombramiento **Esto si cambiamos el nombre desde el editor de texto**
* Hacemos el mismo procedimiento del elemento anterior solo que eliminando, eliminamos el archivo luego el git add -u y hacemos un commit **Esto si cambiamos el nombre desde el editor de texto**
* Para ignorar archivos primero creamos un archivo llamado .gitignore, y dentro ponemos lo que queremos que el git ignore, por ejemplo un archivo en especial, solo ponemos el path del archivo. Para ignorar todos los archivos que terminen en log ponemos \*.log y para ignorar todos los los elementos de una carpeta solo la ruta y marcarla como / ej node\_modules/

# Sección 4: Ramas, Uniones, Conflictos y Tags

## Merge: Fast-Forward

* Para saber en qué rama nos encontramos, debemos poner **git branch**
* Para cambiar el nombre de una rama, en especifico el master por el main, hay que escribir el siguiente comando **git branch –m master main**
* Para que este cambio de poner main a master se pueda hacer cada vez que nosotros iniciemos un proyecto nuevo con git init, cambiamos la configuración con el siguiente comando **git config –-global init.defaultBranch main**
* Para crear una nueva rama, ocupamos el comando **git branch <Nombre Rama>**
* Para saber cuantas ramas hay, podemos dar **git Branch** y se marca en verde, en que rama nos encontramos.
* Para que cambiemos de rama ponemos **git checkout rama-villanos**
* También podemos ver en que rama estamos solo viendo en git s
* Para saber diferencias entre dos rama podemos poner **git diff rama-villanos(rama local) master(raam principal)**
* Para unir dos ramas tenemos el comando **git merge <Nombre de la rama>**
* Para eliminar unba rama ejecutamos el comando **git branch -d <Nombre de la rama>**
* Para eliminar una rama de manera forzada agregamos –f al final de la sentencia **git branch -d <Nombre de la rama> -f**

## Merge: Union Automática

* Para crear una rama y automáticamente pasarte a esta ponemos **git checkout -b rama-villano(rama ejemplo)**
* Igual hacemos un **git merge rama-villano**(rama ejemplo), los merge siempre se deben de hacer desde el maester

## Merge: Uniones con conflicto

## Tags

* Para crear un nuevo tag ponemos **git tag <Nombre del tag>**
* Para ver todos los tag que existen solo debemos de poner **git tag**
* Para eliminar un tag podemos poner **git tag -d <Nombre del tag>**
* Para hacer un tag mas compuesto y que se recomienda ponemos **git tag -a v1.0.0 -m “Version1.0.0”**
* Para hacer tags de un cierto commit que no sea el actual, copiamos el id del commit y ponemos **git tag -a v0.1.0 <Id del commit> -m “Version alfa”**
* Para poder ver más detalle de las versiones podemos dar **git show v1.0.0(tag de ejemplo)**

# Git Stash y Git Rebase

* Para temporalmentemostrar mis cambios podemos poner **git stash** (ósea si tenemos una modificación pero aun no uqeremos mostrarla y nos piden un cambio de urgencia, podemos ocultar nuestras modificaciones con este código, luego las podemos volver a retomar una vez que ya hayamos hecho lo urgente)
* Para ver todos los stash que están en este momento ponemos **git stash list**
* Para recuperar los cambios después de resolver la emergencia, ponemos: **git stash pop**  es como si hiciéramos un git stash apply y después un git stash drop, con un mismo solo comando.
* Para eliminar un stash que ya no queremos podemos poner **git stash drop** este comando elimina el ultimo stash en cola(ósea el ultimo que se ingreso)
* Para tomar el ultimo registro del stash y recuperarlo se pone **git stash apply**
* Para retomar un stash en especifico, primero ponemos el **git stash list**, después vemos el id del stash que por lo general viene como **splash@{0}** y este numero lo ponemos en la petición de esta manera **git stash apply stash@{1}**
* Para eliminar un stash en especifico ponemos **git stash drop stash@{1}**
* Guarda todo menos los archivos en el stage o en el escenario **git stash sabe –keep-index**
* Incluye todos los archivos, junto a los que git no le da seguimiento **git stash sabe –include-untracked**
* Para obtener mas información de un stash y lo que se hizo, ponemos **git stash list –stat**
* Para una mayor información que el stash ponemos **git show stash**, este comando por si solo, nos da las líneas modificadas por la ultima entrada modificada del stash, si queremos la información de un stash en especifico ponemos **git show stash@{1}**
* Para poner una breve descripción desde que se hace el stash podemos poner **git stash save “Agregamos a loki en los villanos”**
* Para eliminar todas las entradas del stash ponemos **git stash clear**
* Para actualizar tu repositorio antes de hacer un commit podemos poner en la rama donde nos encontremos **git rebase master**
* Para poder hacer unir varios commit que se hicieron que tienen que ver con un solo tema por ejemplo, ponemos **git rebase -i HEAD~4** (i es de interactivo, ~4 es a los archivos que queremos unir)
* Para unirlos (squash) solo al ultimo pick se le cambia por s de squash
* Después en la siguiente pantalla solo debemos poner un mensaje sobre el commit en general
* Acortamos el mensaje del commit que acabamos de generar o del rebase, por si es más largo o solo se quere modifar la descripción, ponemos **git rebase -i HEAD~1** (para solo modificar el ultimo commit)
* Al hacer el comando anterior, nos va a mostrar un editor con la información del commit, en este punto cambiamos la palabra pick por r de reward , guardamos y cerramos, al hacer esto, nos abrirá otra pantalla donde ahora si vamos a poder editar el nombre del commit que queremos
* Para poder revertir un cambio que se realizo podemos poner **git checkout – misiones.md(archivo de pruebas)**
* Si se han modificado y dado commit a dos archivos y queremos separar las modificaciones de dichos dos archivos damos **git rebase -i HEAD~2** en el editor nos muestra la primera pantalla, al commit que queremos editar le cambiamos el pick por edit o solo e.
* Después ponemos el comando **git reset HEAD^** para revertir el commit sin afectarlo, al terminar de modificar lo que tengamos que modificar damos **git rebase --continue** para que los cambios puedan ser echos

# Sección 6 github, git remote, push & pull

* Para agregar un repositorio a un acceso remoto ponemos **git remote add origin** [**https://github.com/klerith/udemy-heroes.git(url**](https://github.com/klerith/udemy-heroes.git(url) **de ejemplo)**
* Para revisar las fuentes remotas ponemos **git remote -v**
* Esta línea nos ayuda a que la próxima vez que queramos hacer push, no necesitemos especificar la rama que queremos enviar **git push -u origin master**
* Para subir los tags debemos de poner **git push –-tags**
* Para actualizar el repositorio ponemos **git pull**
* Para cambiar manualmente como queremos que nos traiga los datos el pull, o que estrategia, ponemos **git config –-global pull.ff only**
* Para clonar todo un repositorio a nuestras maquinas desde github, debemos de poner el siguiente comando **git clone** [**https://github.com/arrova/udemy-heroes.git(Ejemplo**](https://github.com/arrova/udemy-heroes.git(Ejemplo) **de repositorio**)
* Para ponerle un nombre en especifico a la carpeta en la que queremos que se deposite el clon debemos de poner la petición anterior mas el nombre de la carpeta contenedora ej **git clone** [**https://github.com/arrova/udemy-heroes.git demo-10**](https://github.com/arrova/udemy-heroes.git%20demo-10)
* Para que al momento de tener problemas podamos ver que esta pasando y hacer un rebase ponermos **git config pull.rebase true**  debemos después de corregir el conflicto, hacer un commit y al final tenemos que poner **git rebase –continue**
* Para que estos cambios se hagan de forma permanente debemos de correr **git config –global pull.rebase true**
* Para subir cambios al repositorio damos **git push**
* El git fecth no hace el merge de forma automática y se pone **git fetch**
* Después ponemos el git pull y nos abre un mensaje de success y después ya podemos tener los cambiso que se hicieron

# Sección 8: Github – Avanzando

* Para revisar los cambios que se hacen en un repositorio online que no podemos hacer un pull, o push hacemos un **git remote add upstram y la dirección del repositorio**
* Para descargar el repositorio ponemos **git pull upstream master** y con esto bajamos el repositorio actualizado del que hicimos el forrk
* Para revisar el trabajo de otras personas podemos poner **git fetch**, luego **git Branch -a** e ir a la rama de la persona que queremos ver su trabajo con **git checkout rama-capitan**
* Para subir cambios , nos vamos a la rama master con **git checkout master**, después **git merge rama-capitan** y terminamos con un **git push**
* Para hacer un pull request para que alguien más pueda revisar nuestro código y hacer una discusión sobre lo que se va a subir damos **git push origin rama-silver**
* Para crear un pull más completo ponemos **git pull –all**  que nos trae hasta las ramas ralizadas en el repositorio
* Para ver todas las ramas del proyecto en general ponemos **git branch -a**
* Para eliminar la basura de ramas que vamos dejando, ponemos desde la terminal **git push origin :rama-misiones**
* Si el paso anterior falla, es por que a lo mejor desde el github ya hemos eliminado la rama por lo tanto damos **git remote prune origin**
* Para crear un tag desde consola, es recomendable estar sincronizados con el master y damos el siguiente código **git tag -a v1.0.0 -m "Versión 1 - Lista para producción"**
* Para poner la versión 0.0.1 ponemos **git tag -a v1.0.0.1 <id del commit>**
* Para hacer un mensaje más elaborado solo le quitamos el -m a la sentencia anterior
* Para subir estos tags a github ponemos **git push –tags**
* Para anexar un commit a un Issue en particular ponemos **git commit -am “Agregamos a Nick fixes #4”**  oara cerrar un issue, en vez de poner fixes, poner closes o resolves
* Para obtener los últimos commits pero no actualizar el estado del commit de nuestra rama utilizamos **git fetch**  cuando estemos totalmente seguros de que queremos obtener el ultimo commit del repositorio entonces hacemos git pull
* Para poder pasar sólo un archivo a un estado de tiempo anterior, muy ocupado cuando queremos que solo un archivo de todo un commit sea el que se regrese a una versión anterior ponemos **git checkout <Numero de commit al que queremos regresar> <nombre del archivo en especifico que queremos regresar>** Ejemplo: git checkout 7b18972 miembreos.md
* Para ver el trabajo de otros usuarios ponemos primero **git fetch** luego **git branch –a** y al final vamos a la rama git checkout rama-capitan
* Para liberar podemos seguir los siguientes pasos: **git checkout master** para pasarnos a la rama principal, luego **git merge rama-capitan** y al final hacer **git push** al repositorio remoto
* Para obtener a todas las ramas remotos ponemos **git branch –a**
* Para poder realizar el cambio de ramas sin problemas podemos poner **git checkout <Nombre de la rama>**
* Para poder eliminar o actualizar todas las ramas que se encuentran en el repositorio debemos poner **git remote prune origin**

# Sección 11: Organizaciones y Equipos