COMANDOS DE LINUX CLASE

Comando	Descripción
$ cat nombre_archivo.txt	Muestra el contenido del archivo nombre_archivo.txt en terminal.
$ cd	Se mueve a el directorio home donde están los archivos del usuario.
$ cd -	Se mueve al último directorio visitado.
$ cd ..	Volver a la carpeta anterior.
$ cd /	Se mueve a el directorio de la raíz del disco.
$ cd /c	Se mueve a el disco C.
$ cd ~	Se mueve a el directorio del usuario en home ~ Es una variable que tiene el nombre del directorio de usuario en home.
$ cd C:/Platzi/'Curso De Introducción A La Terminal'	Se mueve a el directorio de la ruta especificada en la terminal.
$ cd nombre_directorio	Se mueve a el directorio nombre_directorio si está dentro del directorio que estamos actualmente en la terminal.
$ clear	Limpiar consola, otra opción (Ctrl + L).
$ code nombre_archivo.txt	Abre el archivo nombre_archivo.txt para editar con Visual Studio Code (Si está instalado).
$ comando --help	Muestra una serie de especificaciones de como función el comando especificado Ejemplo rm --help
$ history	Muestra el historial de todos los comandos que se han usado en una lista, si quieres volver a usar un comando escribes el singo de admiración ! seguido del número comando lista Ejemplo $!72.
$ ls	Ver los archivos que hay dentro de la carpeta que estamos en la terminal.
$ ls -a	Ver los archivos incluido los ocultos dentro de la carpeta.
$ ls -al	Ver los archivos en forma de lista incluido los ocultos dentro de la carpeta con propiedades y permisos.
$ ls -l	Ver los archivos en forma de lista dentro de la carpeta con propiedades y permisos.
$ mkdir nombre_directorio	Crear un directorio con nombre nombre_directorio dentro del directorio que está actualmente la terminal.
$ pwd	Muestra el directorio actual donde estas parado en la terminal.
$ rm nombre_archivo.txt	Borra el archivo especificado "nombre_archivo.txt"
$ rm -rf nombre_directorio/	Obliga a borrar el directorio especificado "nombre_directorio"
$ rmdir nombre_directorio/	Borra el directorio especificado "nombre_directorio"
$ touch archivo.txt	Crear un archivo vacío con el nombre y la extensión especificada "archivo.txt"
$ vim nombre_archivo.txt	Abre el archivo nombre_archivo.txt (en caso de no existir lo crea) con el editor de archivos Vim Dentro de Vim ejecutar comandos escribiendo dos puntos (:) • :i Insertar texto para modificar el archivo, otra opción (Shift +i). • :w Guarda los cambios realizados al archivo. • :q Salir del editor Vim. • :wq Guardar cambios y salir de Vim, otra opción (Esc + Shift + Z + Z). • :q! Salir sin guardar en Vim.

Notas:

- En Windows no diferencia entre MAYUSCULAS y minúsculas
- Si escribes cd espacio y las primeras letras de la carpeta donde deseas ingresar y la tecla TAB te autocompleta el nombre de esta

COMANDOS GIT

Notas:

- **Working Directory**: Son los archivos Untracked O Unstaged, que No están dentro de Git o esta desactualizado ya que no ha sido afectado por el comando *git add* y su existencia o sus últimas actualizaciones solo están en Disco Duro.
- **Staging Area: Son los archivos Staged, que SI están dentro de Git han sido afectados por el comando *git add* tienen cambios pendientes, pero aun no han sido guardados en el repositorio falta ejecutar el comando *git commit*.**
- **Git Repository**: Son los archivos Tracked, que Si están dentro de Git, no tienen cambios pendientes y sus ultimas actualizaciones han sido guardadas con el comando *git commit*.
- **Repositorio Remoto:** Son los archivos que se encuentran en un repositorio remoto y que pueden ver y trabajar todos los miembros del equipo estos servidores pueden ser GitHub, GitLab, BitBucket, entre otros.

Alias	
Comando	**Descripcion**
$ alias nombre_comando = "comando"	Configura un comando (comando) para la terminal que se ejecuta con su alias (nombre_comando) Ejemplo $ nombre_comando
$ alias arbolito = "git log –all –graph –decorate ---oneline"	Establece el alias arbolito para el comando de Git git log –all –graph –decorate ---oneline Y escribiendo la palabra arbolito ejecuta el comando.
$ git config --global alias.nombre_comando "comando_git"	Configura un comando (comando_git) global de Git que se ejecuta con su alias (nombre_comando) Ejemplo $ git nombre_comando
$ git config --global alias.stats "shortlog -sn --all --no-merges"	Configura el comando shortlog -sn --all --no-merges para que se ejecute con el alias stats Ejemplo $ git stats
$ git config --global alias.superlog "log --graph --abbrev-commit --decorate --date=relative --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"	Establece un alias al comando que se configura para mostrar de forma ordenada y con colores los logs con el nombre superlog. Ejemplo $ git superlog

Add	
Comando	**Descripción**
$ git add .	Agrega todos los archivos del **Working Directory** al **Staging Area.**
$ git add nombre_archivo.txt	Agrega un archivo o carpeta del **Working Directory** al **Staging Area.**

Blame	
Comando	**Descripción**
$ git blame -c nombre_archivo.html	Muestra en un archivo nombre_archivo.html quien hizo cada línea especificando la fecha y hora con tabulaciones entre cada atributo.
$ git blame nombre_archivo.html	Muestra en un archivo nombre_archivo.html quien hizo cada línea especificando la fecha y hora.
$ git blame nombre_archivo.html -Lnumero_inicio,numero_fin -c	Muestra en un archivo nombre_archivo.html quien hizo cada línea especificando la fecha y hora con tabulaciones entre cada atributo entre el numero de la línea de inicio numero_inicio y la línea de número final numero_fin. Ejemplo $ git blame nombre_archivo.html -L35,53 -c

Branch	
Comando	**Descripción**
$ git branch -a	Muestra las ramas que existen en nuestro **Git Repository** y en el **Repositorio Remoto (GitHub).** • **Las ramas en blanco son las que están en nuestro Git Repository** • **La que está en verde con un * es la rama actual (con la rama en la que nos encontramos posicionados en el HEAD).** • **Las ramas que están en rojo son ras ramas del Repositorio Remoto (GitHub).**
$ git branch -d nombre_rama	Elimina la rama nombre_rama. Con -D se fuerza el borrado.
$ git branch -l	Lista todas las ramas que existen.
$ git branch -m nombre_rama rama_nueva	Permite renombrar una rama nombre_rama en rama_nueva
$ git branch nombre_rama	Crea una nueva rama con el nombre nombre_rama basado en la rama actual (con la rama en la que nos encontramos posicionados en el HEAD).
$ git branch -r	Muestra las ramas que existen en el **Repositorio Remoto (GitHub).**
$ git show-branch	Muestra cuales son las ramas que existen y cuál es su historia.
$ git show-branch --all	Muestra todas las ramas que existen y cuál es su historia.
$ git stash branch nombre_rama	Crea una rama nombre_rama a partir del Stash guardado y te posiciona en ella.

Checkout	
Comando	**Descripción**
$ git checkout -b nombre_rama	Crea una nueva rama nombre_rama y se posiciona en ella.
$ git checkout hash nombre_archivo.txt	Permite volver el archivo nombre_archivo.txt a un estado de acuerdo a un Commit identificado por su HASH.
$ git checkout master nombre_archivo.txt	Devuelve el archivo nombre_archivo.txt a la versión master del archivo la cual es la última versión que se realizó Commit.
$ git checkout nombre_rama	Permite moverse entre ramas a hacia la rama nombre_rama (posicionar el HEAD en esta rama).

Cherry-Pick *(Mala Práctica)*	
Comando	**Descripción**
$ git cherry-pick hash	Trae un Commit viejo de otra rama identificado por su HASH a la rama actual (con la rama en la que nos encontramos posicionados en el HEAD).

Clean	
Comando	**Descripción**
$ git clean --dry-run	Hace una simulación de los archivos duplicados que se va a borrar sin borrarlo. Muestra los archivos que se van a borrar
$ git clean -f	Borrar los archivos duplicados dentro de un proyecto (Solo archivos NO carpetas, tampoco tiene en cuenta los archivos especificados en el .gitignore*)*

Clone	
Comando	**Descripción**
$ git clone HTTPS/SSH	Con la dirección del **Repositorio Remoto (GitHub).** ya se HTTPS o SSH(Para esta opción debes configurar las llaves pública y privada para el repositorio) se trae una copia del master a nuestro **Working Directory** de los archivos tal cual y crea una base de datos con todos los cambios históricos en el **Git Repository** y deja **Staging Area** quieto y listo para trabajar.

Commit	
Comando	**Descripción**
$ git commit -am	Agrega los archivos al **Git Repository** desde el **Working Directory**. (Solo funciona si los archivos tienen historia dentro del proyecto, si son archivos nuevos hay que hacerles un *git add*)
$ git commit --amend	En caso de tener un error en el último Commit realizado, se puede rectificar y agregar los cambios con este comando (Los cambios deben estar agregados en el **Staging Area** con *git add*)
$ git commit --amend -m "mensaje"	En caso de tener un error en el último Commit realizado, se puede rectificar y agregar los cambios con este comando (Los cambios deben estar agregados en el **Staging Area** con *git add*) y modifica el mensaje del Commit original.
$ git commit -m "mensaje"	Agrega los archivos del **Staging Area** al **Git Repository** con una descripción mensaje.

Config	
Comando	**Descripción**
$ git config	Muestra todas las configuraciones que trae Git.
$ git config --global color.ui true	Habilita true colores en la UI.
$ git config --global core.editor "'C:\Program Files\Sublime Text 3\sublime_text.exe' --wait"	Cambia el editor de preferencia Vim por sublime text (si está instalado en esa ruta).
$ git config --global user.email "correo@gmail.com"	Establece un correo electrónico correo@gmail.com.
$ git config --global user.name "Nombre_Usuario"	Establece un nombre de usuario Nombre_Usuario.
$ git config --list	Muestra las configuraciones por defecto que trae el Git, además se puede ver cuales faltan.
$ git config –list –show-origin	Muestra donde están las configuraciones guardadas.
$ git config --global alias.nombre_comando "comando_git"	Configura un comando (comando_git) global de Git que se ejecuta con su alias (nombre_comando) Ejemplo $ git nombre_comando
$ git config --global alias.stats "shortlog -sn --all --no-merges"	Configura el comando shortlog -sn --all --no-merges para que se ejecute con el alias stats Ejemplo $ git stats
$ git config --global alias.superlog "log --graph --abbrev-commit --decorate --date=relative --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"	Establece un alias al comando que se configura para mostrar de forma ordenada y con colores los logs con el nombre superlog. Ejemplo $ git superlog

Diff	
Comando	**Descripción**
$ git diff hash_1 hash_2	Muestra los cambios que se han realizados entre el Commit 1 identificado por su hash_1 y el Commit 2 identificado por su hash_2, especificados por su HASH.
$ git diff hash_antiguo	Muestra los cambios que se han realizados entre el estado actual y un Commit identificado por su HASH hash_antiguo.

Fetch	
Comando	**Descripción**
$ git fetch	Descarga cambios del **Repositorio Remoto (GitHub)** en el **Git Repository** pero no los copia en el **Working Directory.**
$ git fetch nombre_rama	Descarga cambios de **Repositorio Remoto (GitHub)** y crea una rama llamada nombre_rama, que luego se debe fusionar con la rama master de nuestro entorno local.

Gitk	
Comando	**Descripción**
$ gitk	Muestra en un programa de forma grafica la historia de las ramas.

Grep	
Comando	**Descripción**
$ git grep “palabra”	Busca dentro del proyecto donde se ha usado la palabra y muestra el archivo.
$ git grep -c “palabra”	Busca dentro del proyecto donde se ha usado la palabra y muestra el archivo y cuantas veces se a usado dentro de este.
$ git grep -n “palabra”	Busca dentro del proyecto donde se ha usado la palabra y muestra el archivo y el número de línea donde se encuentra.
$ git grep -n “palabra”	Busca dentro del proyecto donde se ha usado la palabra y muestra el archivo y el número de línea donde se encuentra.

Help	
Comando	**Descripción**
$ git nombre_comando --help	Muestra cómo funciona un comando nombre_comando y la descripción de este.

Init	
Comando	**Descripción**
$ git init nombre_repositorio	Crea un repositorio con el nombre nombre_repositorio, el cual es opcional, en caso de colocarlo se creará dicho directorio, en caso de no hacerlo simplemente se creará el repositorio partiendo de la carpeta actual.

Llaves SSH	
Comando	**Descripción**
$ ssh-keygen -t rsa -b 4096 C “correo@gmail.com”	Crear una llave pública y privada para que se conecte automáticamente Git y Github y no pida confirmación a cada rato.
$ cat ~/.ssh/id_rsa.pub	Se copia el contenido de la llave creada.
$ eval $(ssh-agent -s)	Enciende el servidor de llaves este funcionando. Da como resultado: *Servidor ssh esta corriendo* → *Agent* *pid* → *Identificador del proceso* *Para SO proceso corriendo* → *numero*
$ ssh-add ~/.ssh/id_rsa	Agregar la llave privada al sistema desde la ruta donde esta guardada ~ Es una variable que tiene el nombre de la carpeta home.

Log	
Comando	**Descripción**
$ git log	Muestra el historial de los registros (Commits) del proyecto con sus respectivos autores, hora específica y descripciones (en caso de tenerlo).
$ git log --all	Muestra todo el historial de los registros (Commits) del proyecto con sus respectivos autores, hora específica y descripciones (en caso de tenerlo).
$ git log --all --graph --decorate --oneline	Muestra todo el historial de los registros (Commits) de forma gráfica y ordenada.
$ git log nombre_archivo.txt	Muestra el historial de los registros (Commits) del archivo nombre_archivo.txt con sus respectivos autores, hora específica y descripciones (en caso de tenerlo).
$ git log -S "palabra"	Busca dentro de los Commits donde se ha usado la palabra y los muestra.
$ git log --stat	Muestra los cambios específicos de cada archivo de acuerdo a los Commits
$ git reflog	Muestra TODA la historia del proyecto con su HEAD
$ git shortlog	Muestra todos los Commits hechos por cada uno de los Integrantes del proyecto.
$ git shortlog -sn	Muestra el número de Commits hechos por cada uno de los Integrantes del proyecto.
$ git shortlog -sn --all	Muestra el número de Commits hechos por cada uno de los Integrantes del proyecto, incluyendo los Commits borrados.
$ git shortlog -sn --all --no-merges	Muestra el número de Commits hechos por cada uno de los Integrantes del proyecto, incluyendo los Commits borrados, pero sin incluir los Merges.
$ alias arbolito = "git log –all –graph –decorate ---oneline"	$ alias arbolito = "git log –all –graph –decorate ---oneline"
Establece el alias arbolito para el comando de Git git log –all –graph –decorate ---oneline	Establece el alias arbolito para el comando de Git git log –all –graph –decorate ---oneline
Y escribiendo la palabra arbolito ejecuta el comando.	Y escribiendo la palabra arbolito ejecuta el comando.

Merge	
Comando	**Descripción**
$ git merge	Funciona los archivos del **Git Repository** con los que están en mi **Working Directory**.
$ git merge nombre_rama	Fusiona una rama nombre_rama con la actual (con la rama en la que nos encontramos posicionados en el HEAD).
$ git merge rama2 --allow-unrelated-histories	Obliga a fusionar la rama rama2 con la actual (con la rama en la que nos encontramos posicionados en el HEAD).

Pull	
Comando	**Descripción**
$ git pull	Descarga cambios del **Repositorio Remoto (GitHub)** en el **Git Repository** y los funciona los archivos del **Git Repository** con los que están en mi **Working Directory**.
$ git pull origin nombre_rama	Descarga cambios de **Repositorio Remoto (GitHub)** de la rama nombre_rama y los fusiona automáticamente la rama actual (con la rama en la que nos encontramos posicionados en el HEAD) de nuestro entorno local **Git Repository**.
$ git pull origin nombre_rama --allow-unrelated-histories	Obliga a fusionar la rama de origin nombre_rama del **Repositorio Remoto (GitHub)** con la actual (con la rama en la que nos encontramos posicionados en el HEAD), que está en el entorno local **Git Repxository.**

Push	
Comando	**Descripción**
$ git push	Envía los Commits a él **Repositorio Remoto (GitHub).**
$ git push origin :refs/tags/nombre_tag	Elimina un tag nombre_tag de él **Repositorio Remoto (GitHub).**
$ git push origin nombre_rama	Enviar la rama nombre_rama a el **Repositorio Remoto (GitHub).**
$ git push origin nombre_rama --tags	Enviar los tags de la rama nombre_rama a él **Repositorio Remoto (GitHub).**
$ git push origin --tags	Envía los tags a él **Repositorio Remoto (GitHub).**

Rebase *(Mala Práctica)*	
Comando	**Descripción**
$ git rebase -i master	Reescribe los Commit en la rama master con los de la actual (con la rama en la que nos encontramos posicionados en el HEAD), agregando según el orden que se agregó en el commit de forma interactiva
$ git rebase master	Reescribe los Commit en la rama master con los de la actual (con la rama en la que nos encontramos posicionados en el HEAD), agregando según el orden que se agregó en el commit

Reflog	
Comando	**Descripción**
$ git reflog	Muestra TODA la historia del proyecto con su HEAD

Remote	
Comando	**Descripción**
$ git remote add origin url	Conecta un repositorio en la url y da un origen de **Repositorio Remoto (GitHub)** a nuestro equipo local en el **Git Repository**.
$ git remote remove origin	Elimina una conexión con **Repositorio Remoto (GitHub).**
$ git remote set-url origin url	Cambia la Url del **Repositorio Remoto (GitHub)** Origin
$ git remote -v	Lista las conexiones existentes. Donde hacer Fetch y Push

Reset	
Comando	**Descripción**
$ git reset --hard	Si uno está parado en el **Staging Area** elimina los últimos cambios y archivos agregados.
$ git reset --hard Hash/Head	Partiendo de un Commit identificado por su HASH o su HEAD, elimina todos los Commits futuros a él y no mantiene los cambios ni en el **Staging Area** ni en el **Wolking Directory** de los Commits eliminados.
$ git reset HEAD nombre_archivo.txt	Quita del **Staging Area** el archivo nombre_archivo.txt y lo deja en nuestro **Working Directory.**
$ git reset --mixed Hash/Head	Partiendo de un Commit identificado por su HASH o su HEAD, elimina todos los Commits futuros a él y mantiene todos esos cambios (registrados en los Commits eliminados) en el **Working Directory**.
$ git reset --soft Hash/Head	Partiendo de un Commit identificado por su HASH o su HEAD, elimina todos los Commits futuros a él y mantiene todos esos cambios (registrados en los Commits eliminados) en el **Staging Area**.

Rm	
Comando	**Descripción**
$ git rm --cached nombre_archivo.txt	Elimina un archivo o carpeta nombre_archivo.txt, del **Staging Area** y lo deja en el **Working Directory**.
$ git rm --force nombre_archivo.txt	Elimina un archivo o carpeta nombre_archivo.txt, del **Staging Area** y del **Working Directory**.

Shortlog	
Comando	**Descripción**
$ git shortlog	Muestra todos los Commits hechos por cada uno de los Integrantes del proyecto.
$ git shortlog -sn	Muestra el número de Commits hechos por cada uno de los Integrantes del proyecto.
$ git shortlog -sn --all	Muestra el número de Commits hechos por cada uno de los Integrantes del proyecto, incluyendo los Commits borrados.
$ git shortlog -sn --all --no-merges	Muestra el número de Commits hechos por cada uno de los Integrantes del proyecto, incluyendo los Commits borrados, pero sin incluir los Merges.

Show	
Comando	**Descripción**
$ git show	Muestra todos los cambios históricos realizados y muestra donde está el HEAD o l lugar donde estoy trabajando.
$ git show-branch	Muestra cuales son las ramas que existen y cuál es su historia.
$ git show-branch --all	Muestra todas las ramas que existen y cuál es su historia.
$ git show-ref --tags	Lista los tags existentes con los Commits respectivos.
$ git config –list –show-origin	Muestra donde están las configuraciones guardadas.

Stash	
Comando	**Descripción**
$ git stash	Guarda el Status de forma temporal.
$ git stash apply	Aplica el último Stash creado.
$ git stash apply stash@{#}	Aplica el Stash seleccionado por su ID stash@{#}.
$ git stash branch nombre_rama	Crea una rama nombre_rama a partir del Stash guardado y te posiciona en ella.
$ git stash drop	Elimina el ultimo Stash.
$ git stash drop stash@{#}	Elimina un Stash por medio de su ID stash@{#}.
$ git stash list	Lista los Stashes.
$ git stash pop	Abre el stash que se tenía.

Status	
Comando	**Descripción**
$ git status	Muestra el estado de los archivos o directorios y en que rama se encuentra. • **Los archivos que salen en rojo:** Se encuentran en el **Working Directory**. • **Los archivos que salen en verde:** Se encuentran en el **Staging Area**.

Tag	
Comando	**Descripción**
$ git tag -a nombre_tag -m “mensaje” hash	Registra un tag nombre_tag un mensaje y el HASH del Commit donde se va a agregar en tag.
$ git tag -d nombre_tag	Elimina un tag nombre_tag de **Git Repository**.
$ git tag -f -a nombre_tag -m "mensaje" hash	Permite renombrar un tag v0.1 y un mensaje y para el HASH.
$ git tag -l	Lista los tags existentes.
$ git push origin :refs/tags/nombre_tag	Elimina un tag nombre_tag de él **Repositorio Remoto (GitHub)**.
$ git push origin nombre_rama --tags	Enviar los tags de la rama nombre_rama a él **Repositorio Remoto (GitHub)**.
$ git push origin --tags	Envía los tags a él **Repositorio Remoto (GitHub)**.
$ git show-ref --tags	Lista los tags existentes con los Commits respectivos.

- **Working Directory**: Son los archivos Untracked O Unstaged, que No están dentro de Git o esta desactualizado ya que no ha sido afectado por el comando *git add* y su existencia o sus últimas actualizaciones solo están en Disco Duro.
- **Staging Area: Son los archivos Staged, que SI están dentro de Git han sido afectados por el comando *git add* tienen cambios pendientes, pero aun no han sido guardados en el repositorio falta ejecutar el comando *git commit*.**
- **Git Repository**: Son los archivos Tracked, que Si están dentro de Git, no tienen cambios pendientes y sus últimas actualizaciones han sido guardadas con el comando *git commit*.
- **Repositorio Remoto:** Son los archivos que se encuentran en un repositorio remoto y que pueden ver y trabajar todos los miembros del equipo estos servidores pueden ser GitHub, GitLab, BitBucket, entre otros.