|  |  |
| --- | --- |
| pwd | Nos muestra la ruta de carpetas en la que te encuentras ahora mismo. |
| mkdir | Nos permite crear carpetas (por ejemplo, mkdir Carpeta-Importante). |
| touch | Nos permite crear archivos (por ejemplo, touch archivo.txt). |
| rm | Nos permite borrar un archivo o carpeta (por ejemplo, rm archivo.txt). Mucho cuidado con este comando, puedes borrar todo tu disco duro. |
| cat | Ver el contenido de un archivo (por ejemplo, cat nombre-archivo.txt). |
| ls | Nos permite ver los archivos de la carpeta donde estamos ahora mismo. Podemos usar uno o más argumentos para ver más información sobre estos archivos (los argumentos pueden ser **-- +** el nombre del argumento o **- +** una sola letra o shortcut por cada argumento). |
| ls -a | Muestra todos los archivos, incluso los ocultos. |
| ls -l | Muestra todos los archivos como una lista. |
| cd | Nos permite navegar entre carpetas. |
| cd / | Ir a la ruta principal. |
| cd o cd ~ | Ir a la ruta de tu usuario. |
| cd carpeta/subcarpeta | Navegar a una ruta dentro de la carpeta donde estamos ahora mismo. |
| cd .. | Regresar una carpeta hacia atrás. |
| cd . | Para referirse al directorio en el que te encuentras ahora mismo. |
| history | Ver los últimos comandos que ejecutamos y un número especial con el que podemos repetir su ejecución. |
| explorer . | Abre el explorador de Windows en la carpeta actual desde git bash. |
| ! + numero | Ejecutar algún comando con el número que nos muestra el comando history (por ejemplo, !72). |
| clear | Para limpiar la terminal. También podemos usar los atajos de teclado **Ctrl + L** o **Command +** |
| q | Permite salir de la ventana de lectura en bash. |
| alias nombre\_comando\_de\_alias=”comandos” | Nos permite agregar alias a distintos comandos. |
| git init | Este comando se encargará de dos cosas:   * **Primero**, crear una carpeta **.git**, donde se guardará toda la base de datos con cambios atómicos de nuestro proyecto. * **Segundo**, crear un área que conocemos como **Staging**, que guardará temporalmente nuestros archivos (cuando ejecutemos un comando especial para eso) y nos permitirá, más adelante, guardar estos cambios en el repositorio (también con un comando especial). |
| git status | Nos permite ver el estado de todos nuestros archivos y carpetas. |
| git add | Nos ayuda a mover archivos del **Untracked** o **Unstaged** al estado Staged. Podemos usar **git nombre-del-archivo-o-carpeta** para añadir archivos y carpetas individuales. |
| git add -a | Para mover todos los archivos de nuestro proyecto (tanto **Untrackeds** como **unstageds**). |
| git reset HEAD | Nos ayuda a sacar archivos del estado Staged para devolverlos a su estado anterior. Si los archivos venían de Unstaged, vuelven allí. Y lo mismo se venían de Untracked. |
| git commit | Nos ayuda a mover archivos de Unstaged a Staged. Esta es una ocasión especial, los archivos han sido guardado o actualizados en el repositorio. Git nos pedirá que dejemos un mensaje para recordar los cambios que hicimos y podemos usar el argumento **-m** para escribirlo (**git commit -m "mensaje"**). |
| git commit -am | Agrega a staged y realiza un commit a la vez con su respectivo mensaje.  **Nota**: Sólo se puede hacer esto cuando con los archivos se le habían hecho **git add .** previamente. Con archivos completamente nuevos no funcionará. |
| git rm | Este comando necesita alguno de los siguientes argumentos para poder ejecutarse correctamente:   * **git rm --cached**: Elimina los archivos del área de Staging y del próximo commit, pero los mantiene en nuestro disco duro. * **git rm --force**: Elimina los archivos de Git y del disco duro. Git guarda el registro de la existencia de los archivos, por lo que podremos recuperarlos si es necesario (pero debemos usar comandos más avanzados). |
| git config | Muestra todas las configuraciones de git. |
| git config --list | Muestra las configuraciones por defecto que trae git. |
| git config --list --show-origin | Muestra donde están las configuraciones guardadas. |
| git config --global | Cambia la configuración de todos los usuarios globales. |
| git config --global user.name “nombre usuario” | Cambia la configuración del usuario de git. |
| git config --global user.email “correo” | Agrega un email a la configuración de git. |
| git config --global color.ui true | Habilita colores en la UI. |
| git config --global core.editor “ruta del editor” | Cambia el editor para abrir los archivos. |
| git log nombre\_archivo | Muestra la historia de un archivo y sus ID. |
| git log --oneline | Te muestra el id commit y el título del commit. |
| git log --decorate | Te muestra donde se encuentra el head point en el log. |
| git log --all | Nos muestra todo lo que hemos hecho históricamente. |
| git log --all --graph | Nos dibuja gráficamente las fusiones de las distintas ramas con la master. |
| git log --all –graph --decorate --oneline | Permite dibujar las ramificaciones de las distintas ramas, pero resumido. |
| git log --stat | Nos permite ver los cambios específicos y en que archivos a partir del commit. |
| git log -p | Explica el número de líneas que se cambiaron y te muestra que se cambió en el contenido. |
| git shortlog | Indica que commits ha realizado un usuario, mostrando el usuario y el título de sus commits. |
| git log --pretty=format:"%cn hizo un commit %h el dia %cd" | Muestra mensajes personalizados de los commits. |
| git log -3 | Limitamos el número de commits. |
| git log --after=“2018-1-2” | Localizamos commit por fecha especificada. |
| git log --after=“today” | Muestra los commit que se han hecho en el día actual. |
| git log --after=“2018-1-2” --before=“today” | Para localizar commit entre una fecha y el día actual |
| git log --author=“Name Author” | Commits realizados por autor que cumplan exactamente con el nombre. |
| git log --grep=“INVIE” | Busca los commits que cumplan tal cual está escrito entre las comillas. |
| git log --grep=“INVIE” –i | Busca los commits que cumplan sin importar mayúsculas o minúsculas. |
| git log -S “Por contenido” | Buscar los commits con el contenido dentro del archivo. |
| git log > log.txt | Guardar los logs en un archivo txt. |
| code | Nos abre VS Code si lo tenemos instalado. ¿No funciona? |
| code nombre\_archivo | Nos abre el nombre del archivo especificado con vs code. |
| git show nombre\_archivo | Nos muestra los cambios que han existido sobre un archivo. |
| git push | Envía los commit a él repositorio remoto de GitHub. |
| git commit -m “mensaje” | Agrega los archivos del **Staged Area** al repositorio de git con un mensaje. |
| git diff commitA commitB | Para ver la diferencia entre una versión y otra de un archivo. |
| Esc + Shift + ZZ | Sale del editor de **VIM** y guarda el msj del commit. |
| git checkout + ID\_commit + nombre\_archivo | Nos permite viajar en el tiempo. Podemos volver a cualquier versión anterior de un archivo específico o incluso del proyecto entero. |
| git checkout + master + nombre\_archivo | Nos permite volver a la última versión que se encuentra en la rama master. |
| git reset | No sólo nos permite “volver en el tiempo”, sino que borramos los cambios que hicimos después de este commit. |
| git reset --hard | Borra toda la información que tengamos en el área de staging (y perdiendo todo para siempre.) |
| git reset –soft [cód\_del\_commit] | Mantiene allí los archivos del área de staging para que podamos aplicar nuestros últimos cambios, pero desde un commit anterior. |
| git mv | Mueve o cambia el nombre de un archivo, un directorio o un enlace simbólico. |
| git reset HEAD | Saca archivos del área de Staging. No para borrarlos ni nada de eso, solo para que los últimos cambios de estos archivos no se envíen al último commit. |
|  |  |
|  |  |
| Trabajo remoto | |
| git remote add origin url\_del\_servidor\_remoto | Nos permite agregar un origen remoto de nuestros archivos. |
| git clone url\_del\_servidor\_remoto | Nos permite descargar los archivos de la última versión de la rama principal y todo el historial de cambios en la carpeta .git |
| git push | Luego de hacer **git add** y **git commit** debemos ejecutar este comando para mandar los cambios al servidor remoto. |
| git push origen master | Nos permite enviarle al **origin** la rama **master**. |
| git pull origin master --allow-unrelated-histories | Nos permite fusionar las historias que se encuentran en remoto con las que tengo en local. |
| git fetch | Para traer actualizaciones del servidor remoto y guardarlas en nuestro repositorio local (en caso de que hayan, por supuesto). |
| git merge nombre\_rama | Nos permite combinar/fusionar los últimos cambios del servidor remoto y nuestro directorio de trabajo. |
| git pull | Básicamente, **git fetch** y **git merge** al mismo tiempo. |
| git remote -v | Verifica de la existencia del origen remoto. |
| git config -l | Permite ver los parámetros de configuración y sus valores. |
| git remote set-url origin <ssh-url> | Configura git para conectar con el repositorio remoto a través de SSH. |
| Ramas | |
| git branch | Nos muestra todas las ramas disponibles en el repositorio local. |
| git branch nombre\_rama o git checkout -b nombre\_de\_la\_rama | Crea una rama. |
| git branch -d nombre\_rama | Borra una rama. |
| git checkout nombre\_rama | Nos permite cambiarnos entre las diferentes ramas, incluyendo la master. |
| git push origin nombre-de-la-rama | Publica una rama local al repositorio remoto. |
| git show-branch | Nos muestra las ramas que existen y cuál ha sido su historia. |
| git show-branch --all | Nos muestra lo mismo que **git show-branch**, pero con más detalle. |
| gitk | Permite ver gráficamente nuestro entorno y flujo de trabajo local. |
| git pull origin nombre\_rama | Trae desde el repositorio de internet la rama especificada. |
| Tags y versiones | |
| git tag -a nombre\_del\_tag -m “Mensaje del tag” id\_del\_commit | Crea un nuevo tag y lo asigna a un commit. |
| git tag | Nos muestra todos los tags disponibles en nuestro repositorio local. |
| git show-ref --tags | Nos muestra a que hash o commit está asignado un tag. |
| git tag -d nombre-del-tag | Borra un tag en el repositorio local. |
| git push origin --tags | Publica un tag en el repositorio remoto. |
| git tag -d nombre\_del\_tag | Borra un tag del repositorio local. |
| git push origin :refs/tags/nombre-del-tag. | Borra un tag del repositorio remoto. |
| Reorganización del trabajo | |
| git rebase [nombre\_rama] | Trae todos los cambios confirmados en una rama y lo coloca en otra.  **Nota**: Es una muy mala práctica hacerlo. |
| Guardar cambios en memoria y recuperarlos después | |
| git stash | Guarda el trabajo actual del **Staging** en una lista diseñada para ser temporal llamada **Stash**, para que pueda ser recuperado en el futuro. |
| git stash sabe “mensaje” | Coloca un mensaje para así diferenciarlos en **git stash list**. |
| git stash pop | Recupera los últimos cambios desde el **stash** a tu **staging area** |
| Git stash pop stash@{<num\_stash>} | Aplica los cambios de un stash específico y eliminarlo del stash. |
| git stash apply stash@{<num\_stash>} | Retoma los cambios de una posición específica del Stash, donde el <**num\_stash**> lo obtienes desde el **git stash list** |
| git stash list | Para ver la lista de cambios guardados en Stash. |
| git stash branch <nombre\_de\_la\_rama> | Para crear una rama y aplicar el stash más reciente. |
| git stash branch nombre\_de\_la\_rama stash@{<num\_stash>} | Para crear una rama y aplicar un stash específico. |
| git stash drop | Para eliminar los cambios más recientes dentro del stash (el elemento 0) |
| git stash drop stash@{<num\_stash>} | Sí conocemos el índice del stash podemos borrar uno en particular (mediante **git stash list**). Donde el <**num\_stash**> es el índice del cambio guardado. |
| git stash clear | Elimina todos los elementos del stash. |
| git stash -u | Guardará en el stash los archivos que no estén en el **staging.** |
| limpiar tu proyecto de archivos no deseados | |
| git clean --dry-run | Para saber qué archivos vamos a borrar. Es como una simulación de lo que borrará con **git clean -f**. |
| git clean -f | Para borrar todos los archivos listados (que no son carpetas) |
| Reconstruir commit | |
| git commit --amend | Modifica los cambios al último commit. |
| Sólo en casos de emergencias | |
| git reset HashDelHEAD | Nos devolveremos al estado en que el proyecto funcionaba. |
| git reset --soft HashDelHEAD | Nos mantiene lo que tengamos en staging. |
| git reset --hard HashDelHEAD | Resetea absolutamente **todo** incluyendo lo que tengas en staging. |
| Buscar en archivos y commits | |
| git grep -n [palabra] | Nos buscará en todo el proyecto los archivos en donde está la palabra **color**. |
| git grep -n [palabra] | Nos dirá en qué línea está lo que estamos buscando. |
| git grep -c [palabra] | Nos dirá cuántas veces se repite esa palabra y en qué archivo. |
| git grep -c "<p>" | Nos dirá cuántas veces utilizamos un atributo de HTML. |
| git log -S “palabra” | Trae las palabras parecidas a las que estamos buscando. |
| git log --all --oneline | grep “cabecera” | Trae los commits en donde se encuentra la palabra. |
| Comandos y recursos colaborativos | |
| git shortlog -sn | Muestra cuantos commit han hecho cada miembro del equipo. |
| git shortlog -sn –all | Muestra cuantos commit han hecho cada miembro del equipo hasta los que han sido eliminado. |
| git shortlog -sn --all --no-merge | Muestra cuantos commit han hecho cada miembro quitando los eliminados sin los **merges**. |
| git blame ARCHIVO | Muestra quien hizo cada cosa línea por línea. |
| git [comando] --help | Muestra la ayuda del comando especificado. |
| git blame ARCHIVO -Llinea\_inicial,linea\_final | Muestra quien hizo cada cosa, línea por línea indicándole **desde** que línea **hasta** que línea. |
| git branch -r | Se muestran todas las ramas remotas. |
| git branch -a | Se muestran todas las ramas tanto locales como remotas. |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |