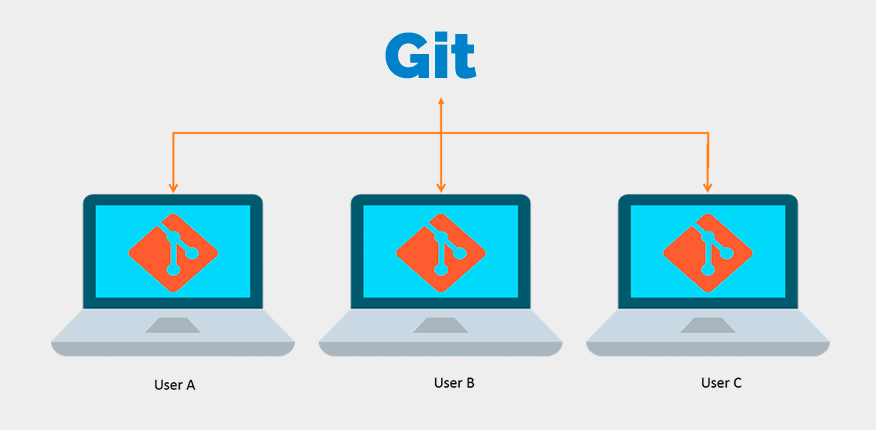
****

**1.Git init**

Crea un repositorio

**2. Git status**

Ver el estado del repositorio

**3. Git add <filename>**

Añade un archivo al staging area

**4.git commit**

Registra el cambio en el repositorio de lo que esté en el staging area y produce un hash para ese cambio

**5.git log**

Muestra los commit registrados en el working directory, con precisión

**6.git commit - - amend**

Para añadir un último cambio de emergencia al último commit realizado sin necesidad de generar un objeto commit más innecesariamente.

**7.git checkout - - <filename>**

Para quitar cambios que aún no están en el staging area

**8.git reset head <filename>**

Le digo a git que me devuelva a mi archivo en la situación que estaba en el último commit , es una forma de sacar el archivo de la staging área.

**9.git log - - online**

Forma de ver el historial de commits de forma más pequeña

**10.git reset <hashcode>**

Destruye los commits antes del commit que le especifique es decir, quiero volver hasta x punto destruyendo los anteriores

**11.git reset - -hard <hascode>**

Quita el commit y quita los cambios del staging area

**12. Git reset - -soft <hascode>**

**Destruye el commit pero deja los cambios en el staging area**

**13.git restore - -staged <file>**

**Saca cambios que están ya en el staging area**

**14.git diff**

**Muestra los cambios que han habido en el código**

**15. Git diff HEAD~n**

Muestra la diferencia del commit actual y el n commits anteriores a él, no solo son números enteros no negativos.

**16.git revert HEAD o <hashcode>**

Revierte el HEAD o puede un commit en específico indicado por su hashcode, esto crea un objeto commit, por cada llamada

**17. Git revert - -no-commit HEAD~n**

Revierte del commit head n commits atrás se hacen todas las reversiones necesarias, creando un único objeto commit por todas las que se necesiten.

**19.git revert - -abort**

Sale de la reversión sin ejecutar nada.

**20.git revert - -continue**

Una vez finalizado el proceso, lo finalizamos con este comando

**21.git branch**

Muestra las ramas

**22.git branch <branchname>**

Crea una nueva rama con el nombre que le especifiquemos

**23.git checkout <branchname>**

se hace un cambio de rama

**24.git branch -m <oldname> <newname>**

Cambia el nombre de una rama

**25.git branch -d <branchname>**

Elimina una rama especificada por el nombre

**26.git checkout -b <branchname>**

Crea una nueva rama y se cambia a ella en un solo paso

**27.git branch -a**

ver todas las ramas, locales y remotas

**28.git log - -oneline - -all**

Podemos ver todos los commits de todas las ramas en una línea

**29.git log - -oneline - -all - -graph**

Muestra los commits de todas las ramas con un grafo que visualmente muestra el progreso, de las ramas.

**30.git merge <branch> ← <otherbranch>**

Une una rama hacia la otra esto generalmente se hace desde la rama a la que quiero aplicar los cambios, generalmente esta rama es la rama master.

**El merge puede generar conflictos, lo ideal es que elija con cuál código del conflicto en particular me quedo y problema resuelto.**

**31.git config - -global alias.aliasname ‘<command>’**

Este comando puede ser bastante útil, ya que varios comandos en git pueden ser lagos y lo que hacemos con esto es asignarle un alias más pequeño a un comando en particular

**32.git config - -global - -get-regexp alias**

Ver los alias que hemos creado

**33.git merge - -abort**

Aborta el merge

**34.git tag ‘tagname’**

Asigna un nombre de etiqueta para el último commit que se ejecutó

**35.git checkout ‘tagname’**

Puedo ver el proyecto en el momento de esa versión

**36.git tag**

Veo todas las tags que he creado

**38.git tag -d ‘tagname’**

Eliminar una etiqueta en específico

**39.git tag ‘tagname’ <hashcode>**

Asignó una etiqueta para cualquier commit especificando su hash

**40.git tag -l “v1.\*.0”**

Lista todas las versiones que tienen coincidencia con lo especificado

**41.git tag ‘tagname’ -a <hashcode>**

Esto hace un tag anotado, el cual tiene un mensaje más serio.

**42.git show ‘tagname’**

Se utiliza para ver los detalles de esa versión especialmente si es tag anotado

**43.git stash**

Guarda en una pila aquello que esté en el staging area, pero que deseo dejar en ‘’stand by’ ni decertarndolo ni commiteandolo de manera, que puedo seguir trabajando con el working directory limpio sin desertar los cambios que ya había hecho

**44.git stash pop**

Saca lo que hay en la pila

**45.git stash save “message”**

Sirve para que a la hora que creemos un stash dejarle un mensaje para nosotros como programadores recordar lo que en ese momento estábamos haciend

**46.git stash list**

Nos muestra todos los stash que tengamos en la pila

**47.git log - -oneline - -decorate - -all - -graph –since= year-month-day**

Sirve para ver el historial de commits a partir de una fecha

**Ejemplo:** git lodag - -since= 2020-09-30

**48.git remote add origin url**

Le adjuntamos un remoto a nuestro working directory local

**49.git remote**

Muestra el nombre del remoto ‘origin’

**50.git remote -v**

Sirve para verificar y mostrar al remoto en el que estamos conectados

**51.git push origin master**

Sirve para enviar código al remoto

**52.git merge origin/master**

Lo mismo que git pull

**53.git fetch origin**

Hacemos la pregunta si han habido cambios en el repositorio remoto, es una forma más segura de hacer los cambios a mi repositorio local

**54.git commit .-m “mensaje”**

Una forma de hacer poner el commit más rápido

**55.git pull - -rebase origin master**

Es la mejor manera y debemos usarla para que el código que estamos trayendo no tenga conflicto con el código que ya tenemos en nuestra máquina local.

**56.git rebase master**

Desde una rama ‘feature’ hacemos un rebase con el master de manera que aunque hayamos hecho las cosas en una rama que no es la master, hagamos que se ordene el los commits de manera lineal y mantener limpieza del historial, lo que quedaría ahora sería ir a la rama master y hacer git merge feature, esto hace un cambio en los hashcode de los commits.

**NOTA: ningún comando destructivo como reset o rebase deben de ser ejecutados con código que se haya pusheado ya que esto causaría problemas fuertes, ÚNICAMENTE DE MANERA LOCAL.**

**57.git rebase -i HEAD~n ‘squash’**

Sirve para poner varios commits locales comprimidos en uno solo para no tener demasiados objetos commits en a lo mejor pequeños cambios, y por lo tanto tener un historial más limpio.

58. git grep “palabra” (-c 2 da contexto incluyendo las 2 lineas anteriores y posteriores a la palabra)

Va a buscar en el repositorio todas las ocurrencias donde aparece la palabra. Esto incluye una búsqueda en todos los archivos del repositorio.