# Comandos de Git.

1. git - - version
2. git help
3. git help “comando”
4. git config - -global user.name “Nombre”
5. git config - -global user.email “correo”
6. git config - -global color.ui true
7. git config - -global -e
8. git init
9. git status
10. git add
11. git reset “nombre del archivo”: eliminar seguimiento del archivo
12. git commit -m “nombre commit”
13. git checkout - - “nombre del archivo”: reconstruye el proyecto al estado del último commit
14. git checkout - - / - -m: te informa de lo que fue borrado
15. git branch
16. git branch -m master main: cambiar el nombre de una rama
17. git config - -global init.defaultBranch main: por defecto, la rama principal del nuevo proyecto se llame main
18. git commit -am “nombre commit”: confirma cambios y nuevo mensaje
19. git lg
20. git status - -short
21. git config - -global alias.s “status - -short”
22. git diff: comparar las modificaciones sin escenario
23. git diff - -staged: comparar las modificaciones con escenario
24. git commit - -amend -m “nuevo texto del commit”: actualizar mensaje de un commit
25. git reset - -soft HEAD^: borrar el ultimo commit
26. git reset - -soft “numero commit”
27. git reset - -mixed “numero commit”
28. git reset - -hard “numero commit”:
29. git reflog: historial de todos los commits realizados
30. git mv archivo1md archivo2.md: cambiar nombre de archivo1
31. git rm archivo2.md: borrar archivo
32. git branch “nombre rama”
33. git checkout “nombre rama”
34. git checkout master
35. git merge “nombre rama”
36. git branch -d “nombre rama”: borrar rama
37. git branch -d “nombre rama” -f: forzar borrado
38. git checkout -b “nombre rama”: crear y moverse a ella
39. git merge - -abort: deshacer merge realizado
40. git remote add origin “enlace”
41. git push -u origin main
42. git push (después de haber hecho el anterior)
43. git pull

Soft =>

* "elimina" los commits posteriores al commit al que estás haciendo el reset
* conserva los cambios en el stage área
* conserva los cambios que tengas en tus archivos (working directory)

Mixed =>

* "elimina" los commits posteriores al commit al que estás haciendo el reset
* Deshace los cambios en el stage área
* conserva los cambios que tengas en tus archivos (working directory)

Hard=>

* "elimina" los commits posteriores al commit al que estás haciendo el reset
* Deshace los cambios en el stage área
* Deshace los cambios que tengas en tus archivos (working directory)

# Teoria GitHub

1 introducción

Git es un software de control de versiones diseñado por Linus Torvalds, pensando en la eficiencia, la confiabilidad y compatibilidad del mantenimiento de versiones de aplicaciones cuando estas tienen un gran número de archivos de código fuente. Su propósito es llevar registro de los cambios en archivos de computadora incluyendo coordinar el trabajo que varias personas realizan sobre archivos compartidos en un repositorio de código.

1.1 Ventajas de GIT

1- Facilita el trabajo colaborativo: Distintos programadores pueden estar editando el mismo archivo, o versiones distintas del mismo archivo, y todos los cambios serán reflejados en el documento final.

2- Reduce considerablemente los tiempos de deploy (despliegue) de un proyecto, al subir solamente los cambios (¡no los archivos cambiados, sólo los cambios!), que en Git se conoce como "diff": las diferencias entre la versión local (la que estás trabajando) y la "master" que está en el servidor central.

3- Permite regresar a versiones anteriores de forma sencilla y muy rápida. En caso de haber realizado cambios negativos en un proyecto en producción, volver a la última versión estable es un simple comando, que retrocede a su estado previo todos los cambios realizados en la última modificación. Esto puede hacerse hacia cualquier versión del proyecto, sin importar la cantidad o calidad de los cambios posteriores.

4- Permite generar flujos de trabajo que facilitan el desarrollo y mantenimiento de proyectos de gran tamaño.

5- El ecosistema Git es increíble, y agrega un montón de herramientas a nuestra disposición para facilitarnos el trabajo, de forma robusta, rápida y profesional. A través de los “hooks” de Git, los distintos servicios pueden detectar cambios en el historial de versiones y realizar acciones automáticas (como actualizar los archivos en el servidor o ejecutar una suite de tests y enviarnos su resultado), ¡dejándonos tiempo libre para cosas más productivas!

6- Las "branches" o ramas, permiten trabajar con una base de código paralela al proyecto en sí, donde podemos corregir bugs o desarrollar nuevas características para el producto sin afectar el "master", pero manteniendo todas las ventajas de usar un sistema de control de versiones. Una vez que estamos contentos con nuestro "branch", podemos combinarlo con el "master" o, en lenguaje Git, hacer un "merge".

7- Empezar a trabajar desde otro entorno es tan fácil como "clonar" el proyecto a tu nuevo entorno, trabajar sobre los archivos que se quieran, y subir los cambios al "master" o a una "branch".

8- Sistema de etiquetas, para etiquetar las distintas versiones del proyecto. Esto es un marcador a una versión específica del proyecto, sólo que en lugar de tener distintos backups de versiones anteriores, apuntamos a distintas versiones dentro de la misma base de código.

Un repositorio de Git es un almacenamiento virtual de nuestro proyecto. Permite guardar versiones del código a las que se puede acceder cuando se necesite.

18 ramas uniones y conflictos

Una rama de un proyecto es una copia en la que empezamos a trabajar y a realizar modificaciones y commits. Desde el punto de vista de git podemos ver cada rama como un camino independiente de commits.

Hasta ahora hemos estado trabajando con sólo una rama que era la rama master que en un momento dado vimos como renombrar a main por ser más políticamente correcto. Crear una nueva rama se puede ver visualmente como una bifurcación en el camino actual. En un momento dado puede que queramos que una rama secundaria se integre en la principal a esto se le llama unión o merge:

18.1 Tipos de merge

• Fast-forward: este tipo de unión se realiza cuando no ha habido cambios en la rama principal y por lo tanto se pueden unir sin problema las dos ramas simplemente agregando los commits de la rama secundaria a la principal

• Uniones automáticas: este tipo de unión se realiza hay cambios en la rama principal pero que no entran en conflicto con los cambios realizados en la rama secundaria, por lo cual git puede realizar la unión de forma automática

• Manuales: en este caso hay modificaciones en la rama principal y en la secundaria que entran en conflicto, por lo que git no puede decidir por sí mismo como hacer la unión y lo que hace es preguntarnos como realizar la unión y a continuación realizar un commit que se denomina Merge Commit

19.1 Control de acceso al repositorio remoto.

Git no se encarga de controlar el acceso al repositorio remoto, es decir de controlar que usuario tiene permiso para modificar ciertos archivos y cuáles no. De esto se encargan otras herramientas. Tenemos dos tipos de herramientas para controlar el acceso al repositorio remoto:

Hosted services: en este caso usamos los servicios que nos proporciona alguna plataforma.

Servicios gestionados por nosotros: utilizamos una herramienta desde nuestro equipo la cual nos permite configurar todo por nosotros mismos.

19.2 GitHub

GitHub es una plataforma de desarrollo colaborativo de software para alojar proyectos. GitHub es una plataforma muy popular porque su versión gratuita nos proporciona muchas características como:

• Repositorios ilimitados

• Páginas HTML, CSS y JS ilimitadas

• Push, Pull, Clones ilimitados

• Issues, Wikis, estadísticas ilimitadas

• Organizaciones ilimitadas

• Desde 2019 permite crear repositorios privados además de públicos.