1. Disco Duro
2. Memoria Ram
3. Repositorio

git init (Iniciar repositorio)

git status(Saber estado)

git config -l (ver configuración de git)

git config - -global <nombre confi(user.name ejemplo)>

git log (Ver Cambios detalladamante) - -stat(muestra cambios en cada archivo)

git log - -all - -graph - -decorate - -oneline(Ver cambios mejor)

alias <aliasName>=”comando que tendrá ese alias”

git show (Mostrar ultimo cambio)

git commit -m “Mensaje” (Subir a repositorio)

Si no agrego un mensaje: (Guardar y Salir de “Vim”)

* Para escribir el mensaje es Esc + i
* Para salir es Esc + Shift + z + z

(Guardar y Salir de “Nano”)

* Guardar Ctrl + x, preguntara si queremos guardar le damos y
* Para salir después de guardar en Enter

rm (Borra 1)

git rm - - cached (Borra 2,3)

git rm - -force (Borra 1,2,3)

git diff n°Commit1(Version Vieja/Original) + n°Commit2(Version Nueva)(Recomendable) (Compara Cambios)(También puedo ver cambios entre staging y cambios que no están en el staging)

git reset - -hard(Deshace todo, absolutamente todo. Toda la información de los commits y del área de staging se elimina del historial.)

git reset - -soft(Borra el historial y los registros de Git de commits anteriores, pero guarda los cambios en Staging para aplicar las últimas actualizaciones a un nuevo commit.)(Borra 3)

git reset - -mixed (Borra todo, exactamente todo. Toda la información de los commits y del área de staging se elimina del historial.)(Borra 2,3)

git reset - -HEAD (Contrario al add)

git checkout n°Commit(volver al commit indicado guardándolos en un Branch temporal(se borra si se sale))

git swich -c <Nombre nueva rama> (Guardar los cambios hechos del checkout(en la rama temporal) para poder hacerle merge a la rama principal)

## Ramas

git Branch <newName> (Crear Rama)

git Branch -d <rama> (Borrar Rama)

git Branch -m <oldName> <newName>

git checkout <rama> (cambiar entre ramas)

**Merge**

git merge <rama de donde viene>(Unir Ramas), (si hay conflictos es buena la comunicación con el equipo)

git merge –abort (Cancelar Merge)

## Llaves Publicas y privadas

Llave publica: por medio de esta se envia algo al usuario que te dio la llave para que el mismo la decifre con su llave privada que solo funciona para el

Crear Llave

Ssh-keygen -t rsa -b 4096 -C “correo”

eval $(ssh-agent -s)(Verificar que hay agente corriendo)

ssh-add ~/.ssh/id\_rsa.pub (Añadir llave privada a la pc)

Traer repositorio/Repositorios de GitHub

Git remote add <nombre repositorio(origin normalmente)> <url github> (nuevo Origen remoto)

Git remote set-url <nombreRepo> <url nuevo> (Modificar origen remoto ya existente)

git clone <link de git hub>(Clonar repositorio ajeno)

HACER GIT PUSH SOLO CUANDO YA ESTE LISTO PARA ENVIAR PARA NO FORZARLO

git push <remoto> master(subir a repositorio remoto)

git pull <remoto> master(traer al repositorio mio y hacer merge al mismo tiempo)

## Git Tag

Git tag -a <versión ejemplo:v1 o nombre del tag> -m “mensaje” <n°Commit> (Agregar Tag)

Git show-ref - -tags (Visualizar Tags)

Git tag(visualizar tags)

Git push origin - -tags(Subir tags)

Git tag -d <nameTag>(Eliminar Tag de git bash)

Git push origin :refs/tags/<nameTag> (Borrar de git-hub)

Al clonar un repositorio ajeno me tienen que agregar a colaboradores con username preferiblemente

## Pull Requests (Lo hace el DevOps)

Estado Intermedio antes de enviar un merge, es una característica de github(sitio web) no de git

Creando una rama aparte, se hacen cambios, se envían al origin , github indica si se puede hacer un pull request(merge) a la rama main usando “compare and pull request”. No ejecuta el merge de por si

Se puede asignar ese pull a alguien mas, decirle que otro que lo revise, colocarle etiquetas, milenstons que son como objetivos

Para colaborar a proyectos ajenos (Codigo abierto por ejemplo, nada mas publicos)

Fork al proyecto, git clone, hacer el push