UTILIDADES GIT – GIT COMMANDS

INDICE DOCUMENTO

GIT CONCEPTOS ÚTILES

git	Siempre la orden empieza con **git**[espacio]
STAGING AREA	Es un área intermedia entre tu directorio de trabajo local y el repositorio Git, que permite seleccionar y preparar los cambios a incluir en la próxima confirmación (commit) antes de enviarlos al repositorio.

UTILIDADES GIT

CLONAR REPOSITORIO git clone

clone	git clone	Clonar un Repositorio Remoto
Por Ssh	`git clone git@github.com:Gerardo-Mtnz-Dev/my_linux.git`	
Por Https	`git clone https://github.com/Gerardo-Mtnz-Dev/my_linux.git`	

Así los clona en directorio `~/my_linux`

Para clonar el Repo con otro nombre de Dir, añadir "*folder-name*"

`git clone git@github.com:Gerardo-Mtnz-Dev/my_linux.git gerar`

Así los clona en directorio `~/gerar`

AÑADIR AL REPOSITORIO git add + git commit + git push

`git add`	Agrega archivos al ***Staging Area*** para ser confirmados en el próximo commit.
`git commit -m "mensaje"`	Crea un nuevo commit en el repositorio de Git con los cambios añadidos al área de preparación, Agregando un mensaje descriptivo.
`git push`	Envía los commits locales al repositorio remoto

➔ Estas 3 órdenes se hacen siempre, cambia lo que se añade en `git add`	
`git add .`	Agrega todos los Archivos y Directorios, hayan sido modificados o no.
`git add folder-file-name`	Agrega los Archivos/Directorios *file-name ó folder-name* que se indican
`git add`	Agrega
`git add`	Agrega

ACTUALIZAR LOCAL git pull.

`git pull`	Actualiza el

ACTUALIZAR LOCAL CON LOS CAMBIOS REMOTOS (SIN BORRAR ARCHIVOS LOCALES NO VERSIONADOS):

```
git fetch origin
git reset --hard origin/main
```

`git fetch origin`	Trae los cambios del repositorio remoto
`git reset --hard origin/main`	Apunta tu rama local exactamente al commit remoto descartando todos los cambios locales.

`git pull`	Actualiza el

ACTUALIZAR LOCAL CON LOS CAMBIOS REMOTOS (SIN BORRAR ARCHIVOS LOCALES NO VERSIONADOS):

```
git fetch origin
git reset --hard origin/main
```

`git fetch origin`	Trae los cambios del repositorio remoto
`git reset --hard origin/main`	Apunta tu rama local exactamente al commit remoto descartando todos los cambios locales

ACTUALIZAR LOCAL (SE VUELVA UNA COPIA LIMPIA DEL REMOTO):

```
rm -rf .git

git init
git remote add origin urlrepositorio
git fetch origin
git reset --hard origin/main
```

Esto es útil si el repositorio local está desordenado o tiene archivos que quieres eliminar completamente.

```
rm -rf mi-repositorio
git clone <url>
```

O más sencillo: vuelve a clonar el repositorio desde cero:

ELIMINAR

ELIMINAR SEGUIMIENTO ARCHIVOS DEL STAGING AREA

`git reset head file-name`	**Elimina** el **fichero** *file-name* del directorio actual
`git reset head`	Elimina **todos los ficheros** del directorio actual

ELIMINAR LOCAL Y REMOTO

`git rm file-name`	**Elimina** el **fichero** *file-name* del directorio actual
`git rm -r folder-name`	**Elimina** el **directorio** *folder-name*

ELIMINAR REMOTO / CONSERVAR LOCAL

`git remote remove repo`	elimina la conexión con el repositorio remoto llamado repo
`git remote -v`	*verificar que se ha eliminado*

ELIMINAR REMOTO / CONSERVAR EN LOCAL TRAS IGNORARLOS .gitignore

`rm --cached folder·file`	Si ya se ha subido el archivo/directorio, para quitarlo del remoto pero mantenerlo localmente

ELIMINAR LOCAL / CONSERVAR EN LOCAL

PASOS RECOMENDADOS

1. ELIMINA LA CARPETA LOCALMENTE:

`rm -rf ruta/a/la/carpeta/`	Esto la borra solo de tu disco, sin tocar Git todavía.

2. EVITA QUE GIT INTENTE BORRARLA DEL REMOTO

Git detectará el borrado local como un cambio. Para que no se suba esa eliminación al remoto, haz uno de estos dos enfoques:

2,1 Restaura la carpeta desde el índice de Git (sin volver a copiarla)

Esto hará que Git restaure la carpeta sin que tengas que volver a clonarla:

`git restore ruta/a/la/carpeta/`	deshace la eliminación local, es útil si te equivocaste.

2,2 Ignorar los cambios locales a esa carpeta

Si quieres mantenerla borrada localmente pero que Git no la considere modificada,

`git update-index --assume-unchanged ruta/a/la/carpeta/*`	Git: "Ignorá los cambios en estos archivos.

LIMITACION:

Si alguien actualiza esa carpeta en el remoto, no verás esos cambios hasta que reviertas este "assume unchanged".

1. PARA REVERTIRLO

`git update-index --no-assume-unchanged` *ruta/a/la/carpeta/**	revierte "assume unchanged".

COMANDOS GIT

COMANDOS MÁS USADOS

`git remote -v`	Verificar Repositorio

GITIGNORE [Ignorar Archivos]

REGLAS / PATRONES DE .gitignore

`#` *al principio*	Ignorar las líneas en blanco y aquellas que comiencen con #
`*`	Todo lo que….. (* corresponde a cero o más caracteres)
`*.txt`	cualquier archivo de tipo txt
`*.[oa]`	cualquier archivo que termine en “.o” o “.a”
`*~`	todos los archivos que terminen con una tilde
`/` *al principio*	para evitar recursividad.
`/` *al final*	para especificar un directorio.
`!` *al principio*	El patrón se niega
`[0-9]`	corresponde a cualquier caracter entre ellos (en este caso del 0 al 9).
`[abc]`	corresponde a cualquier caracter dentro de los corchetes (en este caso a, b o c)
`?`	corresponde a un caracter cualquiera;
`a/**/z`	directorios anidados; `a/**/z` coincide con a/z, a/b/z, a/b/c/z, etc.

`*.swo`	# ignora los archivos que genera Bash
`*.swp`	# ignora los archivos que genera Vim

*.vscode	# ignora los archivos que genera VS Code
*.DS_store	# ignora los archivos temporales que genera Files de Linux:
*.out	# Ignorar los ejecutables de compilación de C:
*.o	# Ignorar los archivos de compilación de C:

EJEMPLO DE UN ARCHIVO .gitignore

```
# ignora los archivos terminados en .a
*.a
# pero no lib.a, aun cuando había ignorado los archivos terminados en .a en la línea anterior
!lib.a
# ignora unicamente el archivo TODO de la raiz, no subir/TODO
/TODO
# ignora todos los archivos del directorio build/
build/
# ignora doc/notes.txt, pero no este: doc/server/arch.txt
doc/*.txt
# ignora todos los archivos .txt del directorio doc/
d oc/**/*.txt
#---------------------------------------------
# Archivos temporales que genera Files
*.DS_store
```

MI ARCHIVO .gitignore

```
#---------------------------------------------
# ARCHIVOS DEL REPO LOCAL
#---------------------------------------------
# ignore dir Docs_edit
/linux_docs/docs_edit
# ignore EDITABLE-DOCS
#*.odt
#*.odt#
#*.docx
# ignore ALL .xcf files
*.xcf
#---------------------------------------------
# ARCHIVOS DE SINCRONIZACIÓN
#---------------------------------------------
# Ignore archivos de database
*.ffs_db
#---------------------------------------------
# MIS ARCHIVOS DE CABECERA
#---------------------------------------------
# Ignorar los Archivos de mi Cabecera de Archivo excepto el Md
my_header.*
!my_header.md
#---------------------------------------------
# ARCHIVOS DE COMPILACIÓN DE C
#---------------------------------------------
# Ignorar los Archivos de compilación de C:
*.out
*.o
#---------------------------------------------
# ARCHIVOS QUE GENERA LINUX
#---------------------------------------------
# Ignore all Folder Icon Files
*.directory
```

```
# Ignora los archivos que genera Bash
*.swo
# Archivos que genera Vim
*.swp
# Archivos que genera VSCode
*.vscode
# Archivos temporales que genera Files
*.DS_store
```

CONTINUACIÓN

```
#------------------------------------------------
# ARCHIVOS TEMPORALES DEL SISTEMA
#------------------------------------------------
# ignore ALL files in ANY directory named temp
temp/
# Archivos de respaldo
*.bak

Archivos temporales generales
*.tmp
# Archivos con tilde (~) al final (backup)
*.~*
# Caché del sistema
.cache/
.thumbnails/
.Trash-*/
# Logs del sistema
*.log
# Archivos de sesiones y bloqueos
*.lock
*.pid
      #Fin
```

RAMAS

CREAR / ELIMINAR RAMAS

`git branch` *Branch-Name*	Crear una Rama
`git checkout -b` *Branch-Name*	Atajo para crear y cambiar de rama

`git branch`
ó
`git branch --list = git branch -l`
	comprobar las ramas que tenemos creadas

`git branch -d` *Branch-Name*	Para eliminar una rama en local

`git push origin -delete Branch-Name`
ó
`git push origin :Branch-Name`
	Para eliminar una rama en remoto

`git checkout Branch-Name`	cambiar de rama
`git push -u origin Branch-Name`	Subir una rama al repositorio
`git pull Branch-Name`	Bajarse a local una rama
`git diff [first-branch]...[second-branch]`	Muestra diferencias de contenido entre dos ramas

`git pull –rebase`
ó las 3 órdenes
`git config pull.rebase true`
`git pull`
`git config pull.rebase false`
	Forzar pull en ramas

FUSIONAR RAMAS

1º	`git status`	Asegurarse de que no haya cambios pendientes en repositorio local o en área preparación.
2º	`git checkout main`	Asegurarse de estar en la rama Main (ó master). Solo se puede fusionar desde esta rama
3º	`git merge Branch-Name`	Fusionar la rama creada con la rama main
4º	`git branch -d Branch-Name`	Eliminar una rama en local

5º	`git push origin -- delete Branch-Name`
	ó
	`git push origin :Branch-Name`
		Eliminar una rama en remoto

CONFLICTOS AL FUSIONAR RAMAS

CONFLICTOS	Cuando fusionamos dos ramas (o fusiona una rama local y una remota), a veces puede surgir un conflicto. Por ejemplo, dos desarrolladores, sin saberlo, trabajan en la misma parte de un archivo. Uno de ellos envía sus cambios al repositorio remoto de Github. Cuando los lleve a su repositorio local, obtendrá un conflicto de fusión.

Git tiene una forma de manejar los conflictos, por lo que puede ver ambos conjuntos de cambios y decidir cuál desea conservar.

1· Cuando tenemos un conflicto de fusión, toma nota de los archivos que tienen un conflicto.
2· En tu editor de código, abra un archivo en conflicto y busque estos marcadores de conflicto:

`<<<<<<< HEAD`	Marca el inicio de los cambios.
`=======`	Divide sus cambios de los cambios en la otra rama.
`>>>>>>> branch-name`	Marca el final de los cambios.

3· Después de editar el archivo, podemos usar el comando git add a para preparar el nuevo contenido fusionado
4· El paso final es crear una nueva confirmación con la ayuda del comando git commit. Veamos ahora los comandos de Git que pueden desempeñar un papel importante en la resolución de conflictos.

COMANDO	ACCION
`git log --merge`	Producir la lista de confirmaciones que están causando el conflicto
`git diff`	Identificar las diferencias entre los repositorios o archivos
`git reset --mixed`	Deshacer los cambios en el directorio de trabajo y el área de preparación
`git merge --abort`	Salir del proceso de fusión y volver al estado anterior a que comenzara la fusión
`git reset`	Restablecer los archivos en conflicto a su estado original

TOKEN DE ACCESO PERSONAL

ACCESO CON TOKEN

USO DE UN TOKEN DE ACCESO PERSONAL EN LA LÍNEA DE COMANDOS

Una vez que tenga un token de acceso personal, puede ingresarlo en lugar de su contraseña al realizar operaciones de Git a través de HTTPS.

Por ejemplo, para clonar un repositorio en la línea de comandos, debe ingresar el siguiente git clone comando. Luego, se le solicitará que ingrese su nombre de usuario y contraseña. Cuando se le solicite su contraseña, ingrese su token de acceso personal en lugar de una contraseña.

```
git clone https://github.com/username/repo.git
```

`username`: *Your-Username*

password: *Your-Personal-Access-Token*

Quedaría:

```
git clone https://github.com/username/repo.git
gerardo-mtnz-personal
ghp_1fbb7k7bvf4spkeiojqxnofm0luncc4uleka
```

Los tokens de acceso personal solo se pueden usar para operaciones Git HTTPS. Si su repositorio usa una URL remota SSH, deberá cambiar el control remoto de SSH a HTTPS .

GIT LAB

GIT LAB TOKENS

TOKENS DE ACCESO PERSONAL PARA AUTENTICACIÓN DE DOS FACTORES (2FA).

Puede generar un token de acceso personal para cada aplicación que use que necesite acceso a la API de GitLab. También puede usar tokens de acceso personal para autenticarse en Git a través de HTTP. Son la única contraseña aceptada cuando tiene activada la autenticación de dos factores (2FA).

TOKEN DE FEED

Tu token de feed te autentica cuando tu lector RSS carga un feed RSS personalizado o cuando tu aplicación de calendario carga un calendario personalizado. Es visible en las URL de esos feeds. No se puede utilizar para acceder a ningún otro dato.

```
Glxx-rnxxXXEp-RrvxXXxx8nx29r
```

TOKEN DE CORREO ELECTRÓNICO ENTRANTE

Su token de correo electrónico entrante lo autentica cuando crea un nuevo ticket por correo electrónico y se incluye en sus direcciones de correo electrónico personales específicas del proyecto. No se puede utilizar para acceder a ningún otro dato.

```
glxxt-3xxxxyie5e5jxx27xsx5x4zxxi
```
