



### UTILIDADES GIT



### △ OTROS DOCUMENTOS DE LINUX

KDE DOCS	LINUX DOCS	GIT DOCS	
UTILIDADES LINUX · kde	6ASH - 6ASIC COMANDS	GIT - GUÍA RÁPIDA	
UTILIDADES APPS ·kde	6ASH COMANDS - Extended	LIBRO PRO·GIT - Indice	
	CARACTERES ASCII	GIT - UTILIDADES	



#### A INDICE DOCUMENTO

CONCEPTOS ÚTILES			COMANDO	S-OPCIONE	S 1s	XX
COMANDOS + USADOS	COMANDOS FILE / DIR	OTROS COMANDOS				
TERMMINAL / KONSOLA						

## COMANDOS GIT



# 🕵 GIT CLONE, ADD, COMMIT, PUSH, PULL, ETC ...

clone	git clone	Clonar un Repositorio	
Por Ssh	git clone git@gi	thub.com:Gerardo-Mtnz-Dev/my_linux.git	
Por Https	git clone https:	//github.com/Gerardo-Mtnz-Dev/my_linux.git	
Así los clona e	en directorio /my_linux	Para clonar el Repo con otro nombre de Dir, añadir	"FolderName"
git clone g	git@github.com:Gera	rdo-Mtnz-Dev/my_linux.git <b>gerar</b>	~/gerar/my_lynux

COMANDO	ACCION	
git add .	Preparar para añadir Todo	
Git commit -m "xxxx"		









# 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.



### EJEMPLO DE UN ARCHIVO .GITIGNORE:

- # ignora los archivos que genera Bash
- \*.SWO
- # ignora los archivos que genera Vim









```
*.SWp
# ignora los archivos que genera VS Code
*.vscode
# ignora los archivos temporales que genera Files de Linux:
*.DS store
# Ignorar los ejecutables de compilación de C:
*.out
# Ignorar los archivos de compilación de C:
*.0
#fin
  #Ejemplos
  # ignora los archivos terminados en .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 subdir/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
#fin
```

## RAMAS



### CREAR / ELIMINAR RAMAS

git branch Branch-Name	Crear una Rama
git checkout -b Branch-Name	Atajo para crear y cambiar de rama
<pre>git branch ó git branchlist = git branch -1</pre>	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
<pre>git diff [first-branch][second-branch]</pre>	Muestra diferencias de contenido entre dos ramas
git pull -rebase ólas3órdenes git config pull.rebase true git pull git config pull.rebase false	Forzar pull en ramas

## S 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
3° git merge Branch-Name	Fusionar la rama creada con la rama main
4° git branch -d Branch-Name	Eliminar una rama en local
git push origin delete Branch-Name  5° ó git push origin :Branch-Name	Eliminar una rama en remoto

## S 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:

<	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 logmerge	Producir la lista de confirmaciones que están causando el conflicto
git diff	Identificar las diferencias entre los repositorios o archivos
git resetmixed	Deshacer los cambios en el directorio de trabajo y el área de preparación
git mergeabort	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

## CLAVE SSH



#### GENERAR UNA NUEVA CLAVE SSH

### ssh-keygen

Generating public/private rsa key pair.
Enter file in which to save the key (/{usuario}/.ssh/id\_rsa):

Presionamos ENTER y nos guardara las claves en el directorio .ssh/ dentro de la carpeta home de nuestro usuario.

Si previamente ya teniamos generadas unas claves veremos lo siguiente:

/home/{usuario}/.ssh/id\_rsa already exists.
Overwrite (y/n)?

Si sobreescribimos las claves no podremos usar las que ya teniamos generadas.









Despues deberiamos ver el siguiente mensaje:

```
Enter passphrase (enter for no passphrase):
```

Si tenemos una passphrase añadiremos una capa mas de seguridad para evitar accesos no autorizados. Despues de añadir la passphrase veremos algo similar a esto:

Ahora si vamos al directorio

### /home/{usuario}/.ssh/

veremos lo siguiente:

```
id_rsa.pub → Clave pública
id rsa → Clave privada
```

## S COPIAR CLAVE PÚBLICA A OTRO EQUIPO Y CONECTARSE DESDE ÉL

Web Medium, Com

## S GENERAR CLAVE SSH PARA GITHUB

Generamos las claves con el mail de la cuenta de GitHub:

```
ssh-keygen -t ed25519 -c "dev.g.....@gmail.com"
enter passphrase (empty for no passphrase): [type a passphrase]
```

(enter para no crear una frase de seguridad)

```
enter same passphrase again: [type passphrase again]
```

(enter para no crear una frase de seguridad)







DOC:

https://

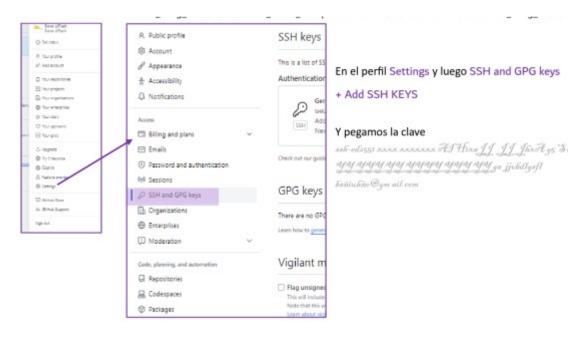


Copiamos la clave pública en la cuenta de GitHub, con un cat simplemente para verla y poder copiarla

### cat ~/.ssh/id ed25519.pub

Y luego en la Cta de GitHub:

### Perfil/settings/SSH and GPG Keys





### **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 clonecomando. 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.

