Git notes

Crear rama y hacer tracking con rama remota

**Crear una nueva rama local llamada 'A\_Branch' y cambiar a ella**:

bash

git checkout -b A\_Branch

Este comando crea una nueva rama local llamada **'A\_Branch'** y cambia a ella.

**Empujar la nueva rama local al repositorio remoto**:

bash

git push -u origin A\_Branch

El argumento **-u** establece la rama local recién creada para que rastree a la rama remota con el mismo nombre, lo que significa que en futuros git push y git pull, Git sabrá a qué rama remota enviar o desde dónde traer los cambios.

Luego de ejecutar este comando tendremos una salida similar :

Total 0 (delta 0), reused 0 (delta 0), pack-reused 0

remote:

remote: Create a pull request for 'A\_Branch' on GitHub by visiting:

remote: https://github.com/User/gitRepo/pull/new/A\_Branch

remote:

To github.com:User/MyProject.git

\* [new branch] A\_Branch -> A\_Branch

branch 'A\_Branch' set up to track 'origin/A\_Branch'.

**Traer los cambios de la rama remota por encima de la rama local**

Para descartar los cambios locales en tu rama A\_Branch y volver al estado del último commit registrado en la rama remota A\_BranchRemote, puedes seguir estos pasos:

1. Asegúrate de estar en la rama A\_Branch:

bash

git checkout A\_Branch

1. Descarga los cambios más recientes de la rama remota A\_BranchRemote:

bash

git fetch origin A\_BranchRemote

1. Reemplaza los cambios locales con los de la rama remota A\_BranchRemote. Puedes hacerlo de varias maneras, pero una forma segura es hacer un reset duro:

bash

git reset --hard origin/A\_BranchRemote

Con estos pasos, habrás descartado los cambios locales en la rama A\_Branch y habrás actualizado tu rama local con los cambios más recientes de la rama remota A\_BranchRemote. Asegúrate de que no tengas cambios importantes en la rama A\_Branch que desees conservar, ya que el reset duro eliminará todos los cambios locales no confirmados.

**Si es que persisten archivos al hacer un git status**

Si realizaste un reset duro con git reset --hard origin/A\_BranchRemote y aún así tienes archivos no rastreados, es posible que estos archivos no estén bajo control de versiones en tu repositorio remoto y, por lo tanto, no fueron afectados por el reset duro.

Para eliminar los archivos no rastreados, puedes ejecutar el siguiente comando:

bash

git clean -fd

Este comando eliminará todos los archivos no rastreados en tu directorio de trabajo. Asegúrate de revisar los archivos no rastreados antes de ejecutar este comando, ya que eliminará los archivos de forma permanente. Si hay archivos que deseas conservar, muévelos a un directorio seguro antes de ejecutar git clean -fd.

**Subir un Repositorio local a uno remoto (con ssh)**

Para este ejemplo usaremos el repositorio local llamado ‘veterinary’ y el repositorio remoto

<https://github.com/demargin/veterinary>

git@github.com:demargin/veterinary.git

1. Primero crear un repositorio remoto en github
2. Conectar el repositorio Local a Github
   1. git remote add origin <https://github.com/demargin/veterinary>
   2. Verificar con el siguiente comando :
      1. Git remote –v
3. Subir los cambios con “git add .” y con “git commit –m “Mensaje” “
4. Ver las claves instaladas en el ordenador con :
5. ls –al ~/.ssh  
     
   Resultado de ejemplo  
     
   -rw------- 1 devmartin devmartin 419 Jan 24 22:20 id\_ed25519   
   -rw------- 1 devmartin devmartin 419 Mar 11 16:17 id\_ed25519\_devMartin   
   -rw-r--r-- 1 devmartin devmartin 108 Mar 11 16:17 id\_ed25519\_devMartin.pub   
   -rw-r--r-- 1 devmartin devmartin 104 Jan 24 22:20 id\_ed25519.pub

Crear o modificar el archivo ‘config’ de SSH  
nano ~/.ssh/config   
  
Ejemplo:  
  
# Configuración para GitHub con clave personal

Host github.com-personal

HostName github.com

User git

IdentityFile ~/.ssh/id\_ed25519

# Configuración para GitHub con clave de trabajo

Host github.com-trabajo

HostName github.com

User git

IdentityFile ~/.ssh/id\_ed25519\_devMartin

### 5. Asociar la configuracion con la URL del Repositorio: Ejemplo: git remote set-url origin [git@github.com-personal:demargin/veterinary.git](mailto:git@github.com-personal:demargin/veterinary.git) Verificar la configuracion: git remote -v 6. Realizar push al repositorio remoto git push origin main \*\*\*Nota puede que haya que resolver conflictos: **Rebase tus Cambios sobre la Rama Remota**

Si prefieres reescribir la historia de tu rama local sobre la base de la rama remota, puedes usar el rebase. Esto aplicará tus commits locales sobre la rama remota, uno por uno. Ejecuta el siguiente comando:

bash

git pull origin main --rebase

Git realizará un rebase de tu rama local (main) sobre la rama remota (origin/main). Si hay conflictos, Git te pedirá que los resuelvas durante el proceso de rebase.  
  
si aparecen mas conflictos se debe verificar con **git status** que archivos estan causando los conflictos, luego con **git add .**  Agregar los archivos conflictivos y usar **git rebase —continue** , finalmente si ya no existe ningun conflicto, empujar los cambios del repo local hacia la rama remota con  **git push origin main .**

## Configurar mas de una llave SSH

Cuando queremos usar mas de una llave SSH en nuestro ordenador, luego de haberlas generado de una manera similar al siguiente comando:  
  
**❯ ssh-keygen -t ed25519 -C "demargin2409@gmail.com" -f ~/.ssh/id\_gitHubPersonal**

donde id\_gitHubPersonal es el nombre de nuestra llave, asi podemos usar mas de una llave SSH en nuestro ordenador.  
  
Luego debemos crear un archivo de configuracion que administrara las llaves SSH que hemos creado he instalado en nuestro ordenador:  
  
**❯ sudo nano ~/.ssh/config**

Una vez dentro del archivo, vamos a definirlo de la siguiente manera:  
  
 GNU nano 6.2 /home/devmartin/.ssh/config

# Configuración para cuenta personal de GitHub (demargin)

Host github.com-demargin

HostName github.com

User git

IdentityFile ~/.ssh/id\_gitHubPersonal

# Configuración para cuenta de trabajo de GitHub (DevMartinG)

Host github.com-DevMartinG

HostName github.com

User git

IdentityFile ~/.ssh/id\_gitHubTrabajo

# Match para repositorios de la cuenta personal (demargin)

Match Host github.com-demargin

User demargin

# Match para repositorios de la cuenta de trabajo (DevMartinG)

Match Host github.com-DevMartinG

User DevMartinG  
  
Lo guardamos con la combinacion de teclas ‘ctrl + o’, asi habremos configurado en este ejemplo dos llaves SSH en nuestro ordenador. Si los repositorios con los que trabajaremos pertecen a cualquiera de la llaves que hemos configurado y sean propietarios no habra ningun problema al usar un ‘git clone’, sin embargo si el repositorio es de otro propietario pero somos participantes de ese repositorio, lo que podemos hacer es usar la llave correspondiente la cual esta asociada a la cuenta participante de ese repo, por ejmplo:  
  
**❯ git clone git@github.com:Ernesto/crapping.git -c core.sshCommand="ssh -i ~/.ssh/id\_gitHubTrabajo"**

En este caso el repositorio le pertenece al usuario Ernesto y el nombre del repo es ‘crappin.git’, pero si somos participantes de ese repo , entonces le indicamos que llave usaremos, en este caso usamos la llave ‘id\_gitHubTrabajo, este comando asociara esa llave para ser usada en ese repositorio , de esta manera nos identificaremos de manera correcta para poder hacer ‘pull’ y ‘push’ a ese repo.

**Asignar las nuevas llaves Ssh a repositorios Locales**

Otro caso en donde debamos configurar que llave usar es cuando hayamos creado un repositorio en github completamente vacio y queramos agregar un repositorio local a ese repositorio remoto(hacer push a ese repo vacio), para ello podemos usar un comando similar al siguiente:

Una vez dentro de nuestro repositorio local, primero debemos asegurarnos de no tener ningun archivo pendiente a ser agregado omodificaciones pendientes, de tener alguna, entonces deberemos primero correr los comandos ‘git add .’, luego ‘git commit -m “My commit”, una vez resuelto cualquier pendiente recien ejecutaremos el siguiente comando:

**❯ git remote add origin** [**git@github.com-DevMartinG**](mailto:git@github.com-DevMartinG)**:DevMartinG/pruebas2.git**

Con este comando estamos indicando que usaremos la llave SSH asociada a DevMartinG para hacer un push hacia un repositorio remoto vacio o nuevo ‘pruebas2.git’, de esta manera quedara asociada esa llave a ese repo para poder realizar los ‘push’ y los ‘pull’.

**NOTA : Caso especial**

Puede que exista el siguiente caso en el que tenias ya una llave ssh generada, la hayas borrado y esta estabas usandola para unos repositorios locales sincronizados a repositorios remotos , pero al momento de hacer push te bota un error asi:  
  
❯ git push

ERROR: Permission to usuarioDeRepo/RepoRemoto.git denied to DevMartinG. fatal: No se pudo leer del repositorio remoto. Por favor asegúrate de que tengas los permisos de acceso correctos y que el repositorio exista.  
  
Como vez aqui no esta usando la llave correcta, en su lugar esta usando las credenciales de otra llave SSH, para resolver esto, usaremos el siguiente comando para actualizar las credenciales:  
  
→ **git remote set-url origin** [**git@github.com-demargin**](mailto:git@github.com-demargin)**:usuarioDeRepo/RepoRemoto.git**como se ve, el error era causado porque la llave que estaba tomando era para el usuario **DevMartinG** y no para el correcto que en este caso es el usuario **demargin,**  en este caso en particular asi se resolveria el problema, y ya podremos hacer push sin ningun problema.

**ELIMINAR RAMAS LOCALES, SEGUN RAMAS REMOTAS**

### 1. **Eliminación automática de ramas locales que ya no existen en el remoto**

Usa el siguiente comando para eliminar las ramas locales que no tienen un equivalente en el repositorio remoto (es decir, aquellas que ya fueron eliminadas en GitHub):

bash

Copy code

git remote prune origin

Este comando elimina las referencias remotas obsoletas que no existen en el repositorio remoto (origin es el nombre por defecto del remoto, si tu remoto tiene otro nombre, reemplázalo por ese).

Después de eso, puedes usar un script para eliminar las ramas locales que ya no tienen una referencia remota. Ejecuta este comando:

bash

Copy code

git branch --merged | grep -v '\\*' | grep -v 'master' | grep -v 'main' | xargs git branch -d

### Explicación:

* git branch --merged: Lista todas las ramas locales que han sido completamente fusionadas en la rama actual.
* grep -v '\\*': Excluye la rama en la que estás actualmente.
* grep -v 'master' y grep -v 'main': Excluye las ramas principales (puedes ajustar esto dependiendo de las ramas principales de tu proyecto, si no es main o master).
* xargs git branch -d: Elimina las ramas que se listan como fusionadas.

Este comando eliminará de manera automática todas las ramas locales que ya han sido fusionadas con tu rama actual y que no tienen un equivalente remoto.

### 2. **Eliminar ramas locales que no están en el remoto (más específico):**

Si quieres eliminar todas las ramas locales que ya no tienen una contraparte en el remoto, puedes usar el siguiente comando más general:

bash

Copy code

git branch -vv | grep ': gone]' | awk '{print $1}' | xargs git branch -D

### Explicación:

* git branch -vv: Muestra las ramas locales y su información sobre el seguimiento remoto.
* grep ': gone]': Filtra las ramas locales que ya no tienen una rama remota asociada (el estado "gone" indica que la rama remota fue eliminada).
* awk '{print $1}': Extrae el nombre de la rama.
* xargs git branch -D: Elimina las ramas locales que ya no tienen una contraparte en el remoto.

### 3. **Verificación final**

Finalmente, asegúrate de que solo tienes las ramas que deseas (tanto locales como remotas):

* **Ramas locales**:

bash

Copy code

git branch

* **Ramas remotas**:

bash

Copy code

git branch -r