



Git para Geeks: control de versiones en proyectos empresariales



## Temas del taller



- Breve reseña histórica.
- Control de versiones.
- Flujo de estados de datos en Git.
- Bases de datos locales.
- Repositorios remotos.
- Proyectos colaborativos.



## Requisitos para realizar el taller

- Instalar Git.
- Crear un usuario en GitHub.
- Crear un directorio de trabajo.



## Historia



- Creado por Linus Torvalds.
- Se utilizó inicialmente para el kernel de Linux.
- Soporta desarrollo no linear.
- Eficiencia con proyectos de gran tamaño.



## Control de versiones



- Crea puntos de restauración de archivos.
- Genera copias de un mismo proyecto.
- Registra los cambios realizados.
- Es colaborativo.



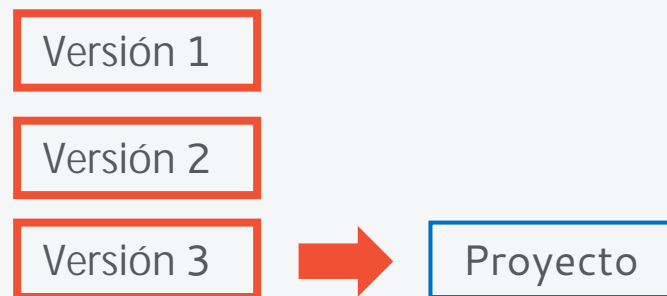
## Tipos de control de versiones



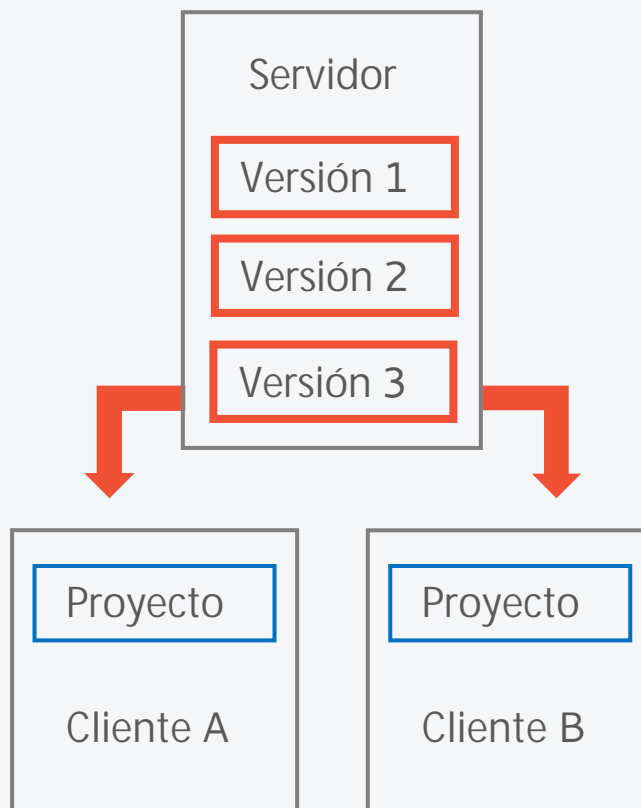
- Control de versiones local.
- Sistemas centralizados de control de versiones.
- Sistema de control de versiones distribuida.



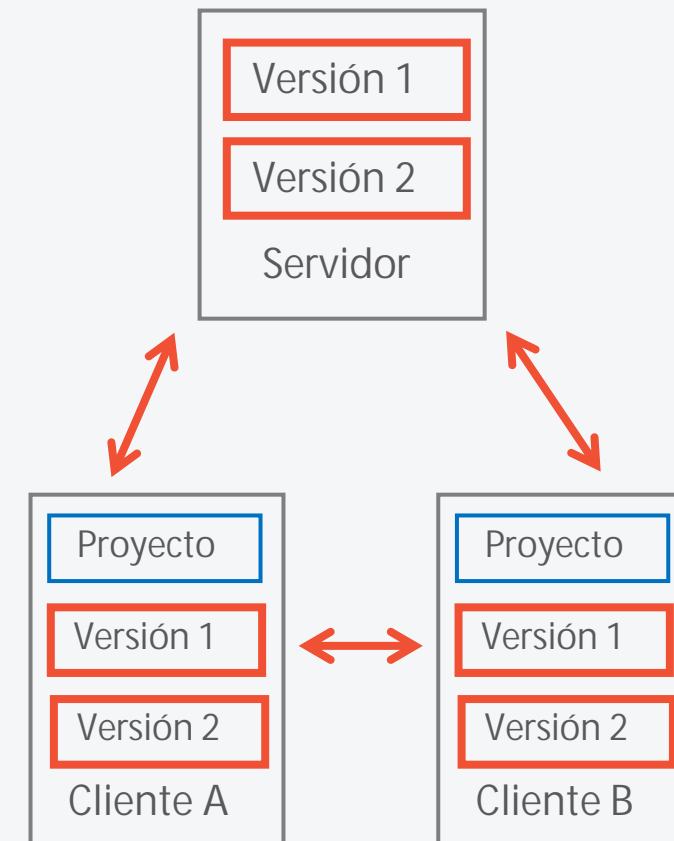
### VCS Local



### Sistema centralizado



### Sistema distribuido





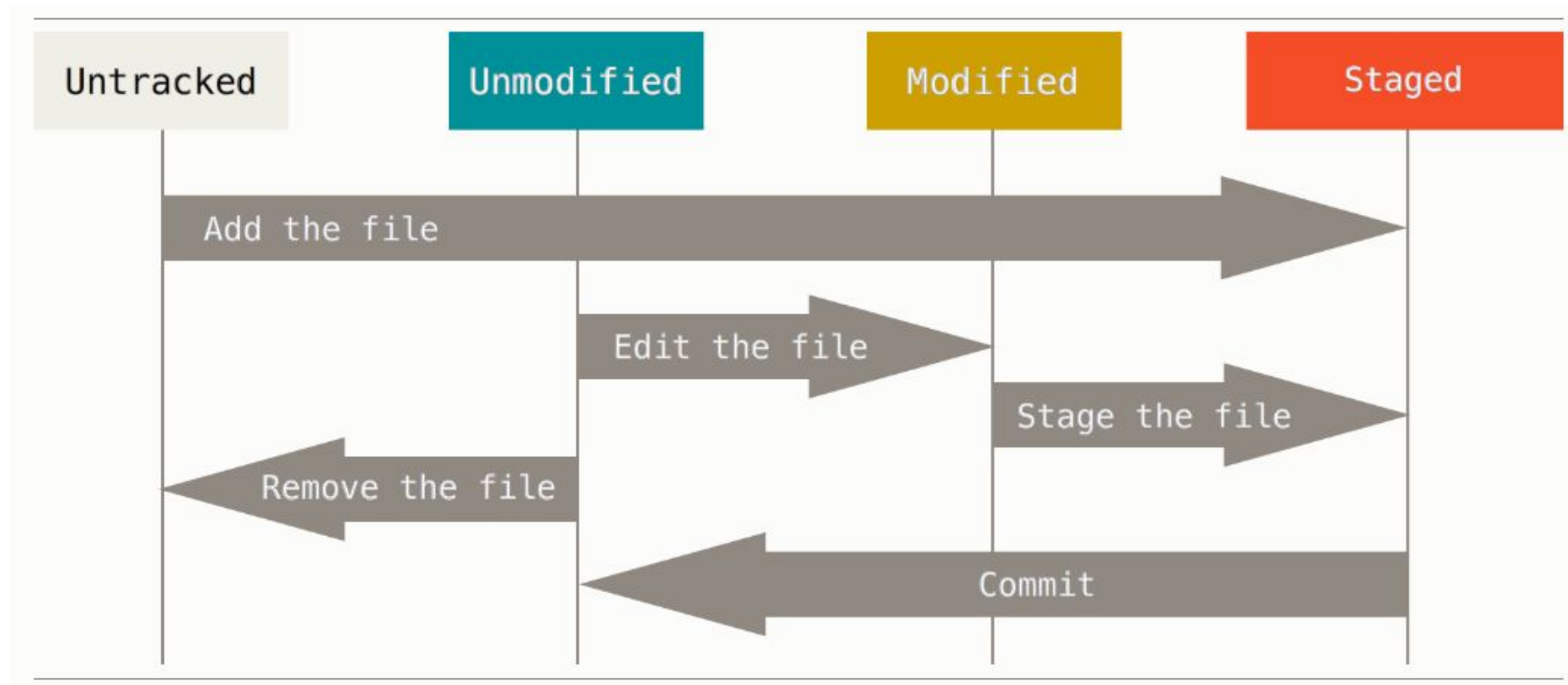
## Flujo de estados de datos en Git

- **Untracked:** archivos sin rastreo.
- **Staged:** archivos listos para commit.
- **Unmodified:** archivos actualizados.
- **Modified:** archivos modificados.





## Estados de datos en Git





## Instalación de Git



- **Fedora:** `yum install git-core`
- **Linux base Debian:** `apt-get install git`
- **Mac:** <http://git-scm.com/download/mac>
- **Windows:** <http://git-scm.com/download/win>




## Configuración inicial



- `git config --global user.name "Nombre de usuario"`
- `git config --global user.email email@example.com`
- `git config --global color.ui true`
- **Ver configuración:** `git config --list`



## Bases de datos locales

- Iniciar Git.  `git init`
- Ver estado.  `git status`
- Agregar archivos a Git.  `git add -A`
- Hacer el primer commit.  `git commit -m "descripción"`
- Ver log.  `git log`
- Remover archivos.  `git rm [nombreArchivo]`
- Retroceder a commit anterior.  `git checkout [codigoSha]`

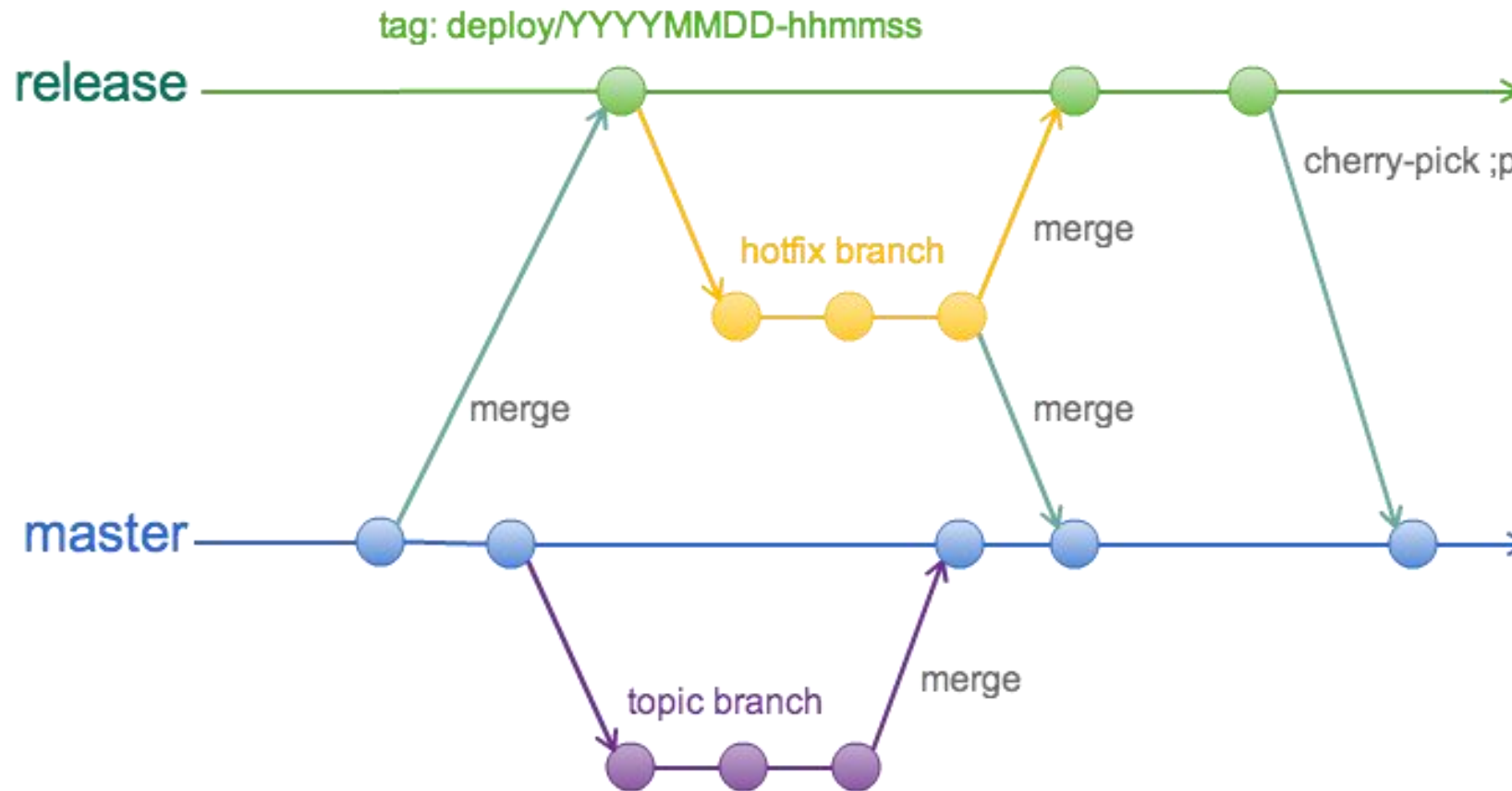


## Git Reset

- `git reset --soft target`  
Regresa a un commit sin cambiar el contenido de los archivos.
- 
- `git reset --hard target`  
Regresa a un commit cambiando el contenido a estado del commit.
- `git reset --keep target`  
Regresa a un commit conservando los cambios *unmodified*.



## Ramas (branch)





## Ramas (branch)

- Crear rama.  `git branch [nombreRama]`
- Ver ramas.  `git branch`
- Moverse a otra rama.  `git checkout [nombreRama]`
- Combinar ramas.  `git merge [nombreRama]`
- Eliminar ramas.  `git branch -D [nombreRama]`



## Repositorios remotos

- GitLab



- BitBucket



- CodePlex



- GitHub







# GitHub







## Repositorios remotos

- Agregar repositorio.  `git remote add origin [url]`
- Agregar proyecto al repositorio.  `git push origin master`
- Descargar proyecto del repositorio.  `git fetch origin`  
`git merge origin/master master`
- Subir ramas al repositorio.  `git push origin [rama]`
- Ver repositorios.  `git remote -v`
- Ver información de repositorios.  `git remote show origin`



## Gitignore

Gitignore es una extensión de archivo que se utiliza para excluir documentos del repositorio.

Dentro del archivo *.gitignore* se declaran los documentos que Git no va a rastrear.  
Ejemplo:

```
# Esto es un comentario
```

```
# Ignorar todos los archivos .php  
*.php
```

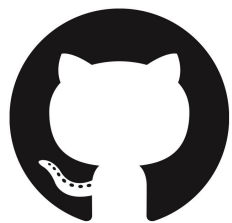
```
# Crear una excepción  
!index.php
```



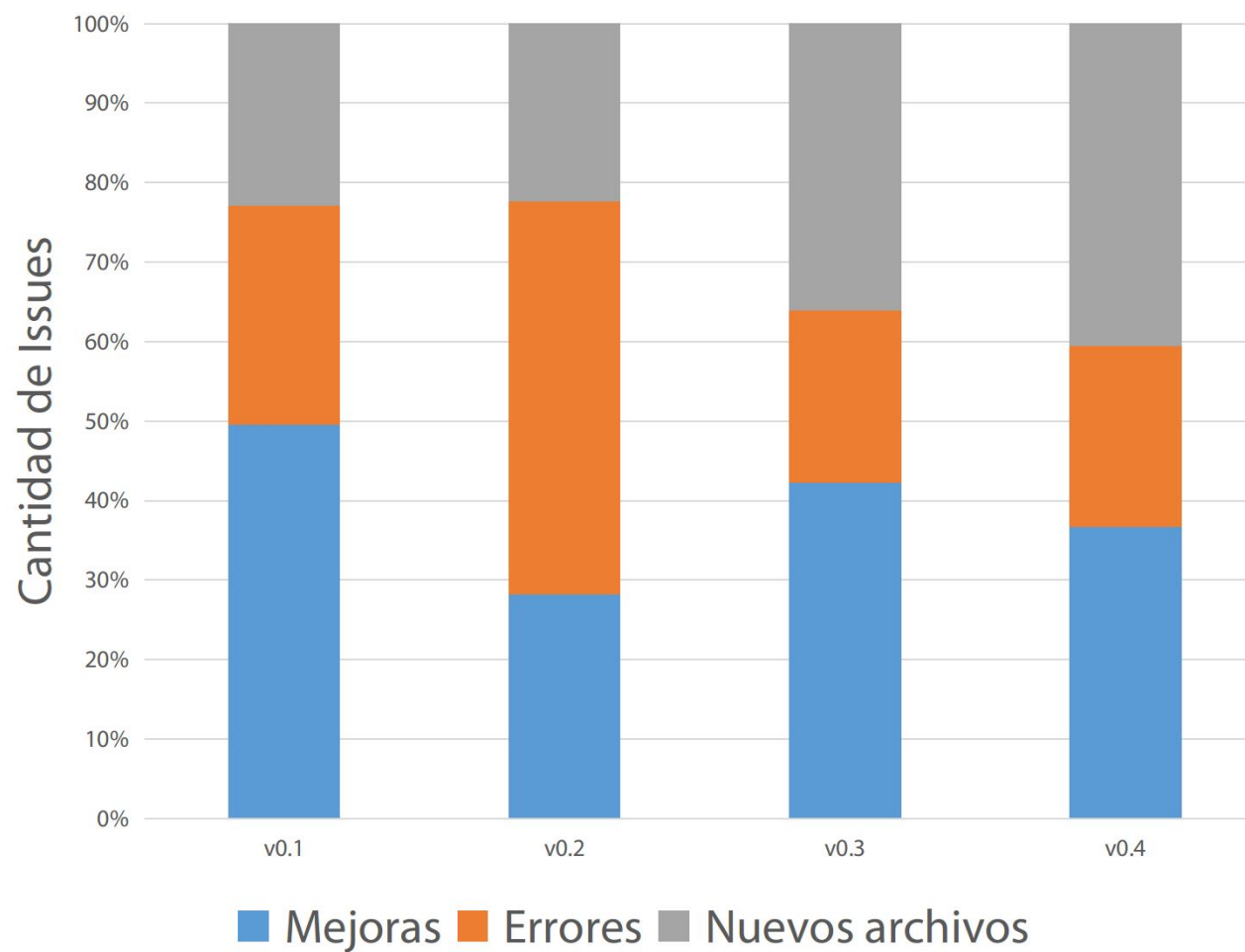
## Milestone & issues

**Issues:** Los issues pueden ser errores, mejoras o cualquier tipo de cambio que requiera el proyecto. Los issues se les asigna a uno o varios contribuyentes como tareas que deben realizar.

**Milestone:** Los milestone marcan un punto en donde el proyecto evoluciona en una nueva versión. Estos agrupan issues con el fin de que, al completarlos se de paso a la nueva versión.



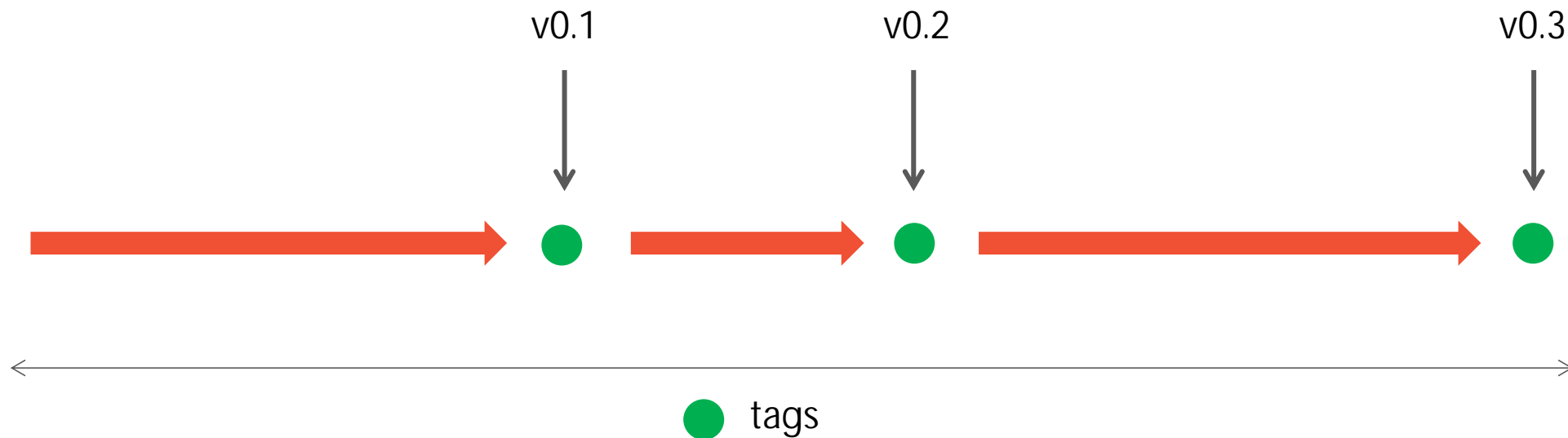
## Milestone & issues





## Etiquetas (tags)

Las etiquetas se utilizan para marcar un commit como una versión del proyecto, Ejemplo; v1.0.2





## Etiquetas (tags)

- Crear un tag en formato corto.      ➡      `git tag v0.1`
- Crear un tag con descripción.      ➡      `git tag -a v0.1 -m "descripción"`
- Ver información de los tag.      ➡      `git tag`
- Subir un tag al repositorio.      ➡      `git push origin v0.1`



## Hooks

Los hooks son scripts que se ejecutan antes o despues de una acción en Git. Como son ordenes de consola se puede ejecutar cualquier comando de sistema, ejemplo:

```
#!/bin/sh  
git push origin master
```



## Hooks / post-commit

Crear un archivo en el directorio hooks/

```
$ touch .git/hooks/post-commit
```

Agregar el script.

```
$ nano .git/hooks/post-commit
```

```
#!/bin/sh
```

```
# acción
```

Dar permisos al directorio.

```
$ chmod 777 -R .git/hooks/
```



