Comandos Avanzados de Git

Los comandos avanzados son herramientas que proporcionan mayor control y flexibilidad en la gestión del proyecto. A continuación, se describirán algunos comandos avanzados.

Pull Request

Un ***pull request*** es una propuesta para combinar un conjunto de cambios de una rama a otra. Esta solicitud permite a los colaboradores conocer sobre posible incorporación de cambios y así debatir y decidir si agregar o no los commits antes de que se fusionen a la rama destino.

En este ***pull resquest*** se puede revisar y analizar las diferencias entre el contenido de la rama de origen y el contenido de la rama destino.

Los pasos más comunes para realizar un `pull request` son:

- Se crea una nueva rama para realizar las modificaciones, se puede utilizar el comando:

```
git checkout -b <nombre-nueva-rama>
```

- Se realizan los `commits` necesarios:

```
git add .
git commit -m "Mensaje del commit"
```

- Se suben los cambios de la rama local al repositorio remoto:

```
git push <repositorio-remoto> <rama>
```

- Se crea el ***pull request*** en el repositorio remoto a a través de la interfaz de usuario.
- Se examinan los cambios y si todo está en orden, el administrador podrá fusionar los cambios.

Fork

Un **fork** o ***bifurcación*** es una función que permite crear una copia exacta de un repositorio, pero a partir del ***fork*** el repositorio clonado será tratado como un repositorio distinto, es decir, existirán dos repositorios independientes. Para realizar un ***fork*** existen dos maneras:

A través de la interfaz del repositorio en línea:

- Tener una cuenta de usuario del repositorio en línea.
- Ir al URL del repositorio a clonar.
- Buscar la opción de `fork`.
- Dar clic y esperar a que se clone en la cuenta del usuario.

A través de línea de comando.

- Ir al URL del repositorio a clonar.
- Conseguir la dirección HTTPS.
- Utilizar el comando `clone` con las credenciales de usuario.

```
git clone https://github.com/<tu-usuario>/<nombre-del-repositorio>.git
```

Rebase

Git rebase es una operación que permite tener un historial de confirmación de una rama de manera más lineal y clara a través de la reorganización o el cambiamiento de la base una rama de `commit` a otra. Esta reorganización es util y se logra visualizar en una línea del tiempo secuencia y elimina las bifurcaciones no necesarias, reduciendo así el crecimiento de los `merge`.

Hay tres formas importantes para realizar el comando `git rebase`:

Rebase estándar

Se realizará un ***rebase*** de la rama actual sobre su punto de origen. Es decir, aplicará los cambios locales sobre el ancestro de la rama actual.

```
git rebase
```

Rebase de una rama sobre otra

Se realizará un ***rebase*** de la rama actual sobre la rama destino. Después del ***rebase*** la rama origen se verá como una extensión lineal de la rama destino.

```
git rebase <rama-base>
```

Rebase interactivo

Se realizará un ***rebase*** que permitirá una mayor flexibilidad para editar, combinar y eliminar `commits` durante el proceso.

```
git rebase -i <rama-base>
```

Stash

El comando `git stash` se utiliza para almacenar temporalmente una captura de los cambios sin enviarlos al repositorio remoto. Este comando separa el directorio de trabajo de área de preparación o del repositorio.

Es útil para cuando se tiene que cambiar de contexto, pero los cambios no están del todo lista para confirmarlos.

Algunos comando importantes sobre el comando `stash` son:

- Guardar los cambios en el stash:

```
git stash save "mensaje-opcional"
```

- Ver los cambios guardados en el stash:

```
git stash list
```

- Borrar un cambio guardado en el stash:

```
git stash drop <nombre-del-stash>
```

- Borrar todos los cambios guardados en el stash:

```
git stash clear
```

- Recuperar un cambio guardado en el stash:

```
git stash apply <nombre-del-stash>
```

- Recuperar un cambio guardado en el stash y eliminarlo:

```
git stash pop <nombre-del-stash>
```

Clean

El comando `git clean` elimina todos los archivos sin seguimiento en el directorio de trabajo, es decir, todos los archivos que aún no se han añadido al repositorio con el comando `add` .

```
git clear -f
```

Algunas opciones utilizadas con el comando `git clear` son:

- `-n` - Modo simulación, muestra que habría sido eliminado.
- `-d` - Elimina también directorios sin seguimiento.
- `-x` - Ignora los archivos que están en `.gitignore`
- `-i` - Modo interactivo, pregunta antes de eliminar cada archivo.

Cherry-Pick

El comando `cherry-pick` permite aplicar selectivamente uno o más ***commits*** de una rama especifica a otra. Este comando es fácil de ejecutar pues solo se necesita conocer el hash del ***commit***.

```
git cherry-pick <hash-commit> <commit-hash>
```