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.

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 los archivos que se eliminarían proporcionando una vista previa de los cambios potenciales.
- `-d` - Elimina también directorios sin seguimiento. Esta opción es útil para limpiar completamente el espacio de trabajo.
- `-x` - Ignora los archivos que están listados en el archivo `.gitignore` . Esta opción es útil cuando no se desea conversar algunos archivos no rastreados que están excluidos en el archivo `.gitignore` .
- `-i` - Modo interactivo, solicita la confirmación antes de eliminar cada archivo. Esta opción brinda un mayor control, ya que permite revisar y aprobar cada eliminación de forma individual.

Es fundamental tener precaución al utilizar el comando `git clean -f` , ya que eliminará permanentemente los archivos no rastreados.

Cherry-Pick

El comando `cherry-pick` permite aplicar selectivamente uno o más ***commits*** de una rama especifica a otra. Esto es útil cuando deseas llevar cambios específicos de una rama a otra sin traer consigo todo el historial de esa rama. Este comando es fácil de ejecutar, pues solo se necesita conocer el hash del ***commit***, es decir, el identificador único del commit a aplicar.

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

Algunas opciones utilizadas con el comando `git cherry-pick` son:

- `-i` - Modo interactivo, te permite revisar los cambios de cada commit antes de ser aplicados a la rama destino.
- `origin/rama-remota` - Es posible aplicar commits de una rama remota a una rama destino local. Esto es útil para traer cambios específicos sin fusionar toda la rama remota.

- `-- continue` - Pueden existen conflictos durante el proceso del comando. En estos casos, se deben resolver manualmente los conflictos antes de completar el `cherry-pick`.
- `--abort` - En el caso de que se desee abortar el proceso, esta opción permite revertir los cambios realizados hasta el momento y devuelve el repositorio al estado anterior al inicio del `cherry-pick`.