Laboratorio git stash

Laboratorio #1 básico

Paso 1: Crea un nuevo directorio para el laboratorio

```
mkdir git-lab
cd git-lab
```

Paso 2: Inicializa un repositorio Git

```
git init
```

Esto inicializa un nuevo repositorio Git en el directorio actual.

Paso 3: Crea un archivo y confírmalo en el repositorio

```
echo "Versión inicial del archivo" > archivo.txt
git add archivo.txt
git commit -m "Agregar versión inicial de archivo.txt"
```

Aquí has creado un archivo llamado `archivo.txt` y has hecho un commit con un mensaje.

Paso 4: Haz algunos cambios no confirmados en el archivo

```
echo "Cambios en progreso, aún no listos para confirmar" >>
archivo.txt
```

Ahora, si ejecutas `git status`, verás que hay cambios sin confirmar.

```
git status
```

Debería mostrar algo como:

```
Changes not staged for commit:
  modified:   archivo.txt
```

Paso 5: Usa `git stash` para guardar los cambios no confirmados

Si necesitas guardar temporalmente esos cambios para trabajar en otra cosa, puedes usar `git stash`.

```
git stash
```

Esto guarda los cambios en una especie de "cajón temporal" y restaura tu repositorio al último commit limpio. Si ejecutas `git status` ahora, deberías ver que no hay cambios pendientes:

```
git status
```

El resultado debería ser:

```
nothing to commit, working tree clean
```

Paso 6: Recupera los cambios guardados con `git stash apply`

Si ya terminaste lo que necesitabas y quieres recuperar los cambios guardados, usa el comando `git stash apply`:

```
git stash apply
```

Esto aplicará los cambios que guardaste en el "stash" nuevamente a tu área de trabajo.

Paso 7: Confirma los cambios

Finalmente, puedes confirmar los cambios como lo harías normalmente.

```
git add archivo.txt
git commit -m "Aplicar cambios guardados con git stash"
```

Paso 8: Limpia el stash

El comando `git stash apply` no elimina el stash, solo aplica los cambios. Si quieres eliminar el stash, puedes usar:

```
git stash drop
```

Si quieres aplicar y eliminar el stash en un solo paso, puedes usar:

```
git stash pop
```

Resumen de comandos

- `git stash`: Guarda los cambios no confirmados.

- `git stash apply`: Recupera los cambios guardados.
- `git stash pop`: Recupera y elimina los cambios guardados.
- `git stash list`: Muestra los stash guardados.

Laboratorio #2 intermedio

Paso 1: Crea el directorio y el repositorio

```
mkdir git-lab-complejo
cd git-lab-complejo
git init
```

Paso 2: Crea la rama principal y agrega un archivo

Primero, agrega un archivo básico en la rama `main` y confírmalo.

```
echo "Versión inicial en main" > main.txt
git add main.txt
git commit -m "Agregar archivo inicial en main"
```

Paso 3: Crea una nueva rama para una característica (feature)

Ahora, imagina que estás comenzando a trabajar en una nueva característica en una rama separada.

```
git checkout -b feature/nueva-caracteristica
echo "Nueva característica en desarrollo" > caracteristica.
txt
git add caracteristica.txt
git commit -m "Comenzar nueva característica"
```

Hasta ahora, has iniciado un nuevo trabajo en la rama `feature/nueva-caracteristica`. Ahora sigamos adelante.

Paso 4: Realiza cambios en la nueva característica (sin confirmar)

Trabaja en tu característica y haz algunos cambios que aún no estás listo para confirmar.

```
echo "Progreso no terminado en la nueva característica" >>
caracteristica.txt
```

Paso 5: Recibes una solicitud urgente para corregir un bug en la rama principal

Ahora, mientras estás trabajando en la rama `feature/nueva-caracteristica`, recibes la solicitud de corregir un bug urgente en la rama `main`. Necesitas cambiar de rama inmediatamente, pero **tienes cambios no confirmados** en tu archivo `caracteristica.txt` que no puedes perder.

Si intentas cambiar a la rama `main` sin hacer algo con esos cambios no confirmados, Git te impedirá hacerlo:

```
git checkout main
```

Verás algo como:

```
error: Your local changes to the following files would be o
verwritten by checkout:
        caracteristica.txt
Please commit your changes or stash them before you switch
branches.
```

Paso 6: Usa `git stash` para guardar tus cambios

Para poder cambiar de rama sin perder tu trabajo, usa `git stash` para guardar temporalmente los cambios que hiciste en `caracteristica.txt`.

```
git stash
```

Paso 7: Cambia a la rama `main` y corrige el bug

Ahora que tus cambios están guardados, puedes cambiar a la rama `main` sin problemas.

```
git checkout main
```

Imagina que el bug está en el archivo `main.txt`. Haz la corrección del bug:

```
echo "Corrección de bug urgente" >> main.txt
git add main.txt
git commit -m "Corrección de bug urgente en main"
```

Paso 8: Vuelve a la rama de tu característica

Una vez que hayas corregido el bug en la rama `main`, puedes volver a la rama de tu característica:

```
git checkout feature/nueva-caracteristica
```

Paso 9: Recupera tus cambios con `git stash apply`

Los cambios que guardaste anteriormente siguen en el "stash". Para continuar donde lo dejaste, aplica el stash:

```
git stash apply
```

Esto recupera tus cambios no confirmados en `caracteristica.txt`. Ahora puedes seguir trabajando en la característica y, eventualmente, confirmar los cambios cuando estén listos:

```
git add caracteristica.txt
git commit -m "Continuar con el desarrollo de la nueva cara
cterística"
```

Paso 10: Limpia el stash

Si ya aplicaste tus cambios, puedes eliminar el stash para limpiar el historial:

```
git stash drop
```

Resumen de flujo de trabajo:

1. **Inicio**: Trabajas en una nueva característica en una rama separada.
2. **Cambios no confirmados**: Haces algunos cambios no confirmados en la característica.
3. **Interrupción**: Recibes una solicitud urgente para corregir un bug en la rama principal.
4. **Usa** `git stash`: Guardas tus cambios no confirmados.
5. **Cambias de rama**: Corregir el bug en la rama `main`.
6. **Regreso**: Vuelves a tu rama de trabajo y recuperas los cambios usando `git stash apply`.
7. **Continúas trabajando**: Confirmas los cambios de tu característica.

Este flujo de trabajo muestra claramente cómo `git stash` es esencial para gestionar interrupciones sin perder el progreso en tu desarrollo.

Este laboratorio te obliga a usar `git stash` para poder corregir el bug urgente en otra rama sin perder los cambios no confirmados en la característica en la que estás trabajando.