

*GIT*

**stash, clean, cherry-pick**

# Stash en Git: Guardando cambios temporales

- **¿Qué es un stash?**

En Git, un stash es un mecanismo que permite guardar temporalmente los cambios en tu directorio de trabajo sin hacer un commit.

- **¿Cuándo usar un stash?**

Cuando necesitas cambiar de rama pero tienes cambios no completados.

Para cambiar de contexto y trabajar en una solución diferente temporalmente.

Antes de actualizar con cambios remotos.

- **Comandos clave:**

git stash save "mensaje": Guarda los cambios en un stash con un mensaje descriptivo.

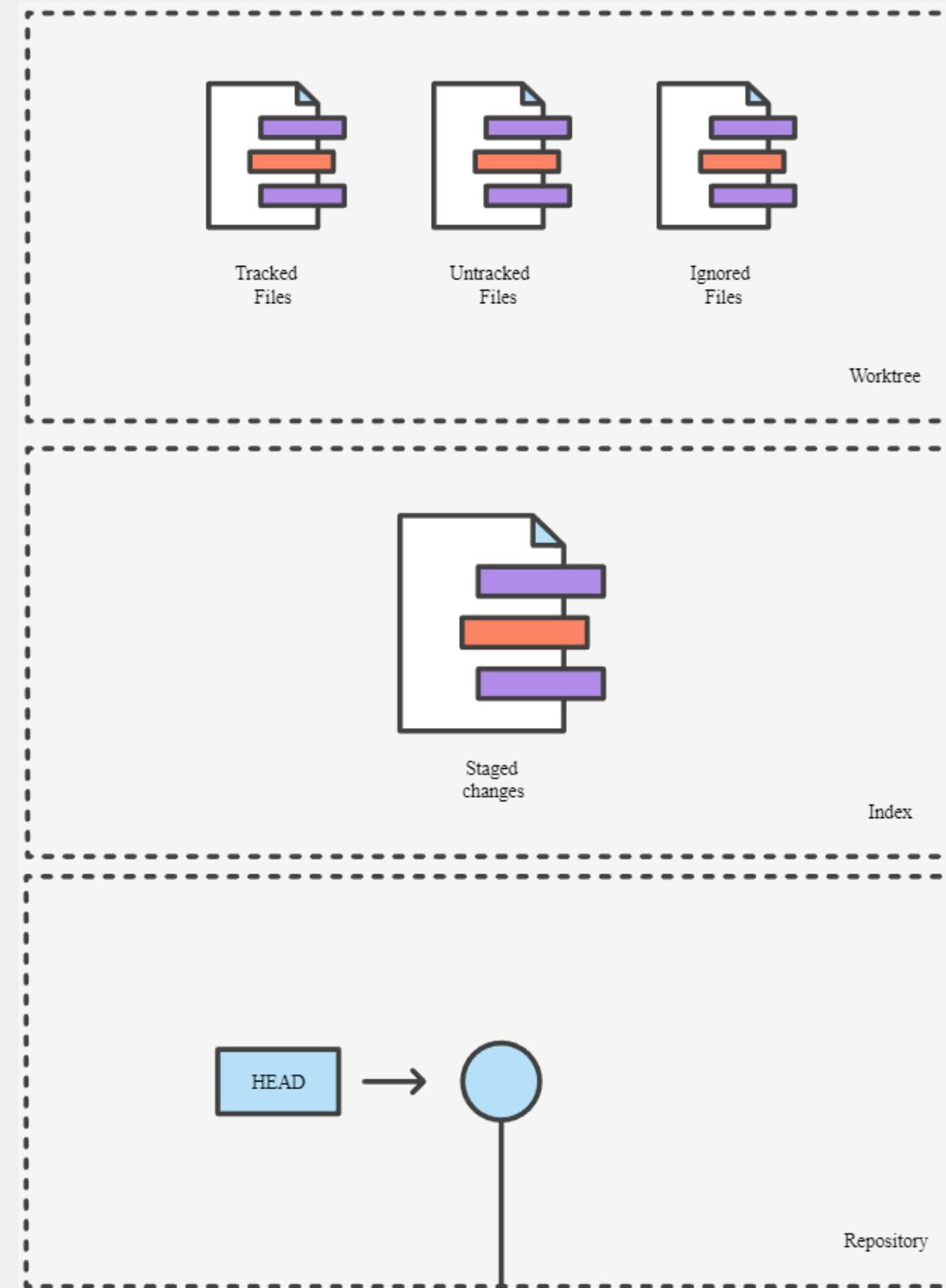
git stash list: Muestra la lista de stashes guardados.

git stash apply stash@{n}: Aplica un stash específico sin eliminarlo.

git stash pop: Aplica el stash más reciente y lo elimina de la lista.

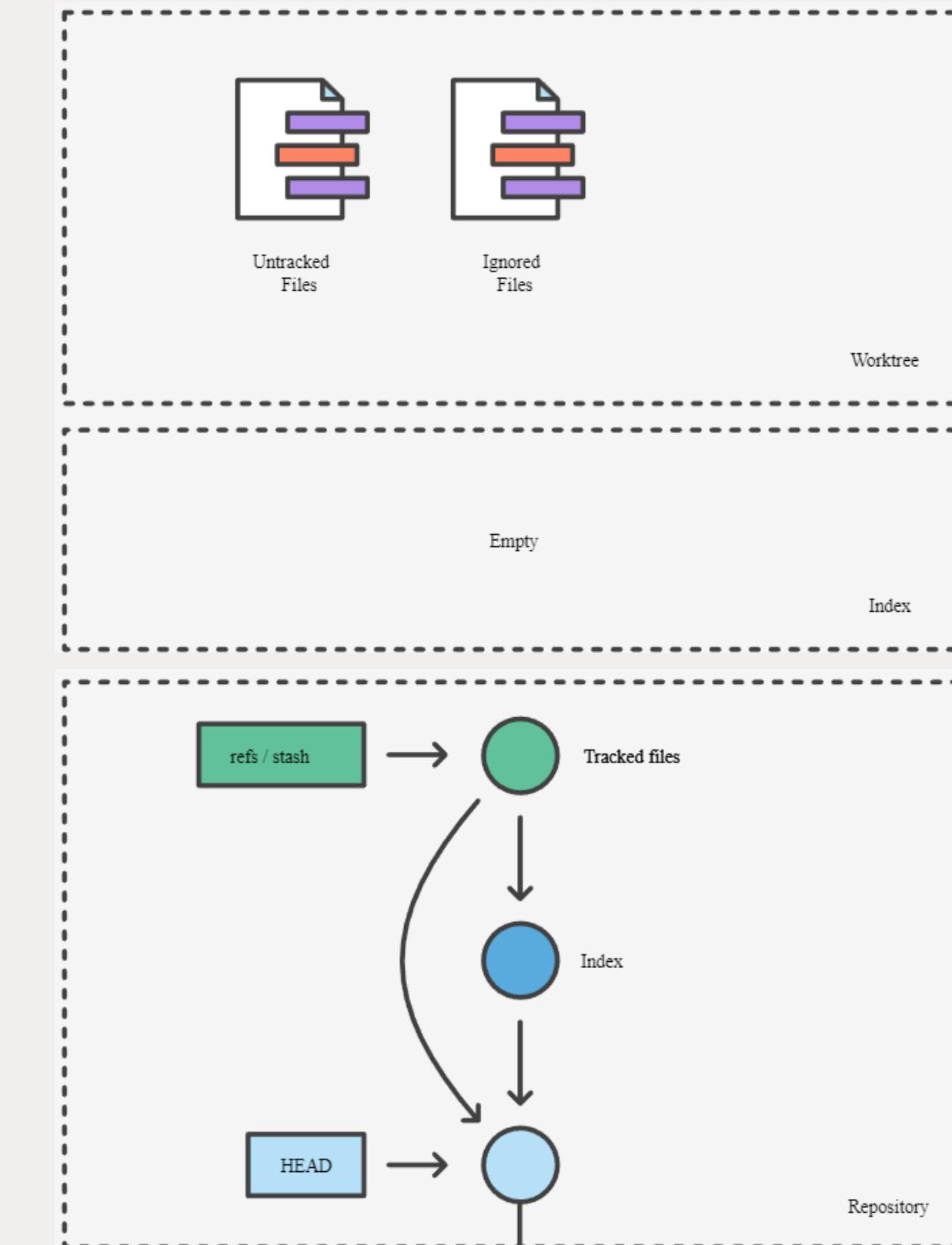
1

Before stashing, your worktree may contain changes to tracked files, untracked files, and ignored files. Some of these changes may also be staged in the index.

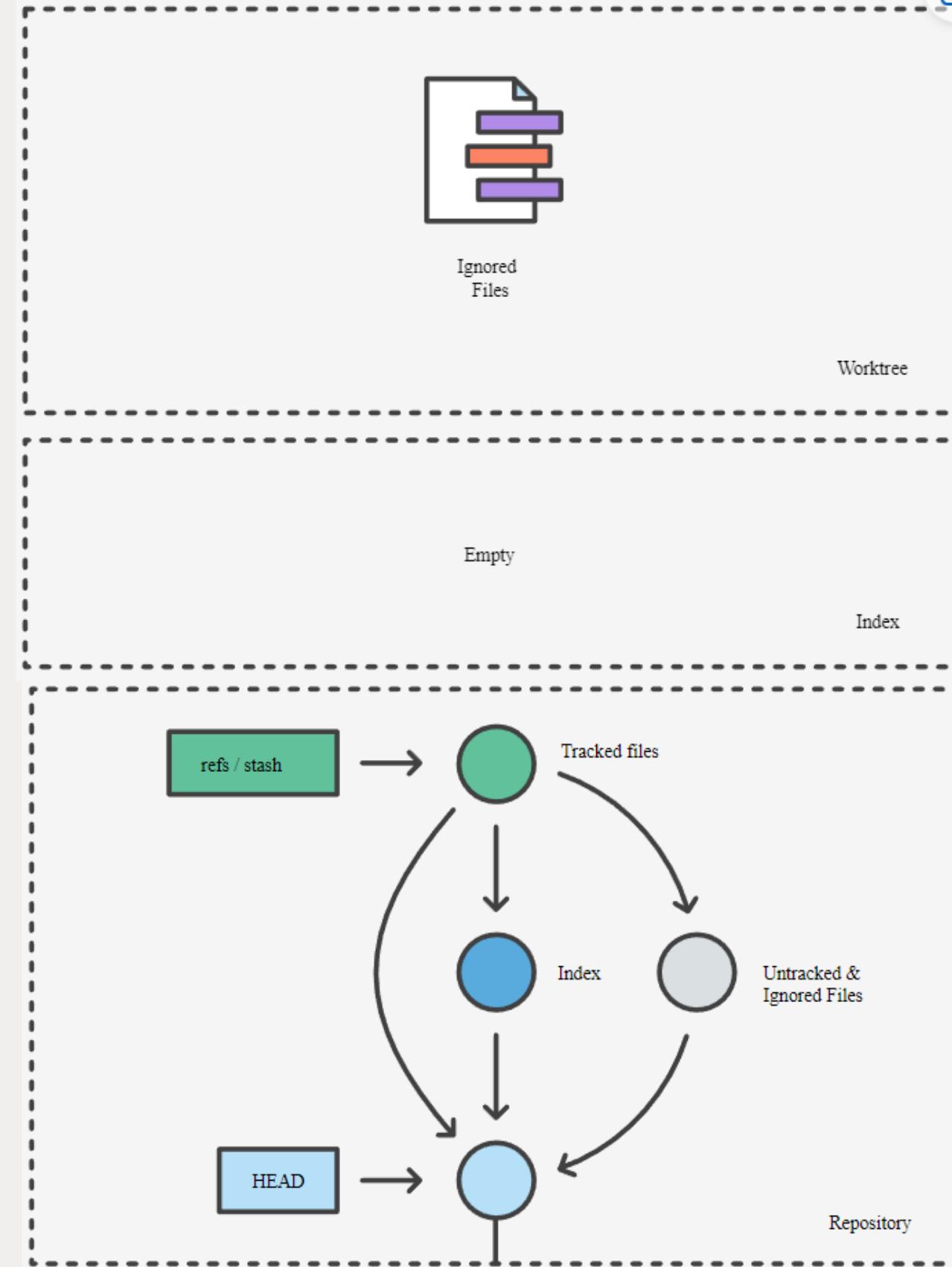


2

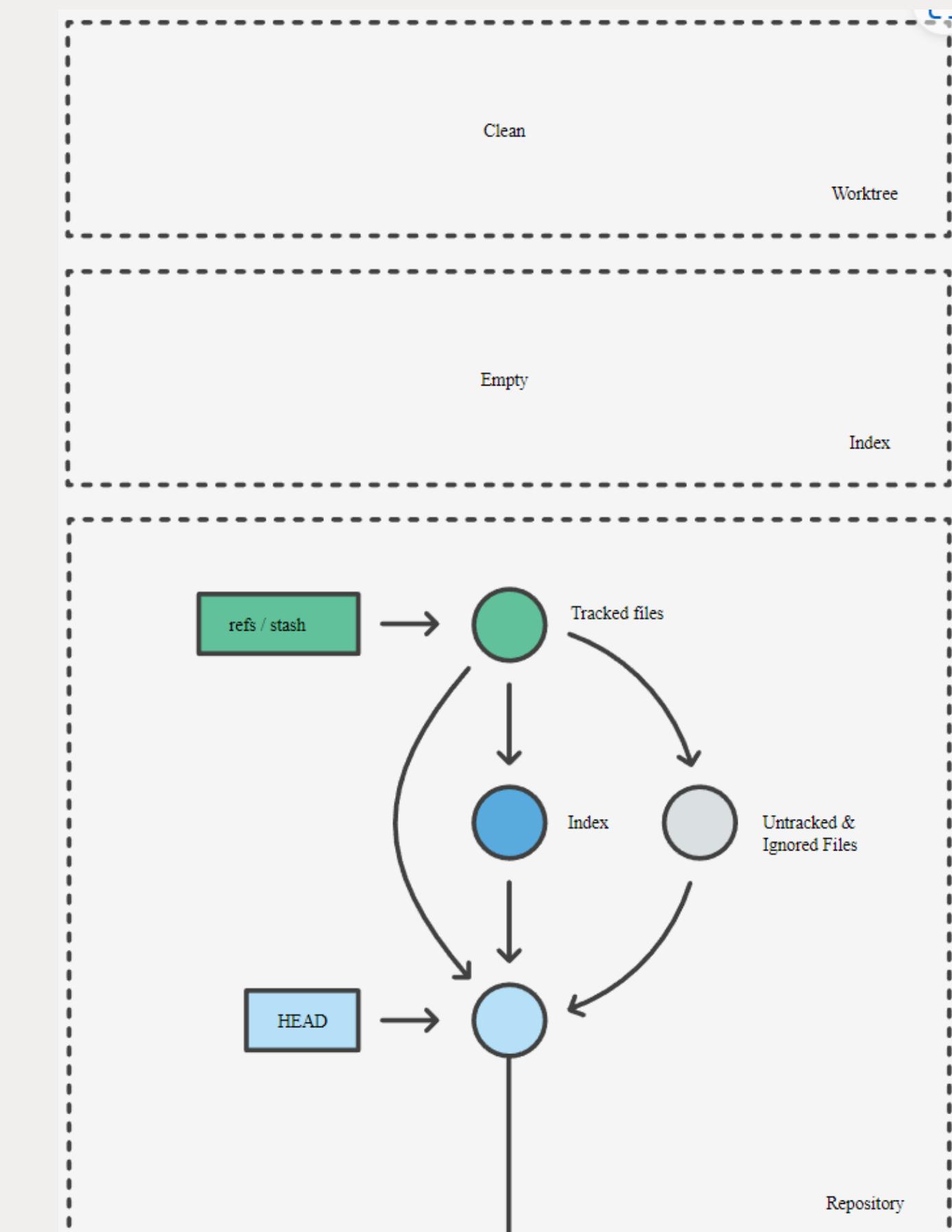
Invoking git stash encodes any changes to tracked files as two new commits in your DAG: one for unstaged changes, and one for changes staged in the index. The special refs/stash ref is updated to point to them.



3 Using the --include-untracked option also encodes any changes to untracked files as an additional commit



4 Using the --all option includes changes to any ignored files alongside changes to untracked files in the same commit.



# Clean en Git: Eliminando archivos no rastreados

- **¿Qué es el comando clean?**

El comando git clean se utiliza para eliminar archivos y directorios no rastreados en el directorio de trabajo.

- **Escenarios comunes para usar clean:**

Cuando deseas deshacerte de archivos generados automáticamente (por ejemplo, archivos de compilación).

Para limpiar archivos que están en el .gitignore.

- **Comandos clave:**

git clean -n: Muestra los archivos y directorios que serán eliminados, pero no los elimina realmente.

git clean -f: Elimina los archivos y directorios no rastreados de forma definitiva.

git clean -df: Elimina directorios no rastreados junto con sus contenidos.

# Cherry-Pick en Git: Aplicando cambios específicos

- **¿Qué es el cherry-pick?**

Cherry-pick es una operación que te permite seleccionar y aplicar cambios específicos de un commit a otra rama.

- **Cuándo usar cherry-pick:**

Cuando solo necesitas algunos cambios de una rama en otra.

Para aplicar correcciones específicas a una rama sin traer todo el historial.

- **Comandos clave:**

**git cherry-pick <hash del commit>**: Aplica los cambios de un commit específico en la rama actual.

**git cherry-pick -x <hash del commit>**: Realiza un cherry-pick y agrega un mensaje con información sobre el commit original.

Para demostrar cómo utilizar git cherry-pick, supongamos que tenemos un repositorio con el siguiente estado de rama:

Usar git cherry-pick es sencillo y se puede ejecutar de la siguiente manera:

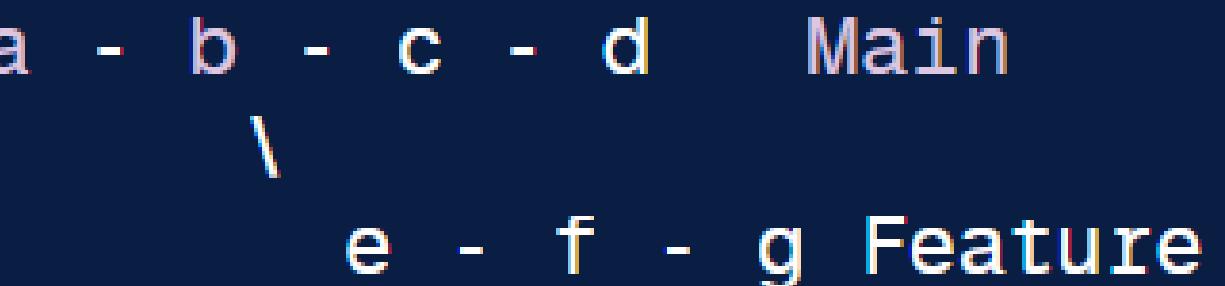
En este ejemplo, commitSha es una referencia de confirmación

Puedes encontrar una referencia de confirmación con el comando git log. En este caso, imaginemos que queremos aplicar la confirmación f a la rama principal. Para ello, primero debemos asegurarnos de que estamos trabajando con la rama principal.

A continuación, ejecutamos cherry-pick con el siguiente comando:

Una vez ejecutado, el historial de Git se verá así:

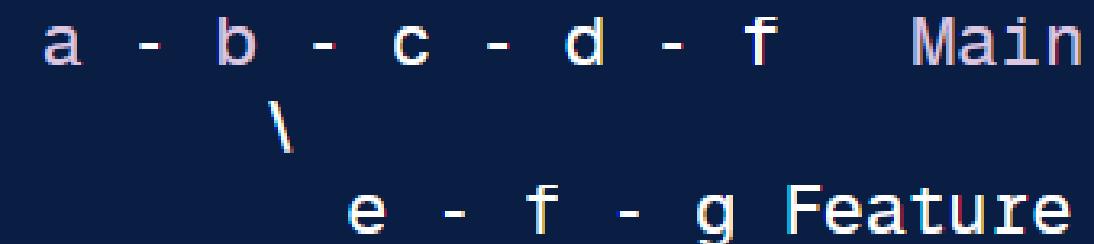
La confirmación f se ha introducido correctamente en la rama principal



```
git cherry-pick commitSha
```

```
git checkout main
```

```
git cherry-pick f
```



# GRACIAS

