02. Comandos Git

Comandos Git Hub Cheat sheet
https://training.github.com/downloads/es_ES/github-git-cheat-sheet.pdf

Aquí tienes una lista de algunos de los comandos más importantes en Git junto con una breve explicación de cada uno:

1. **git init:**
   - Inicializa un nuevo repositorio Git en el directorio actual. Crea un subdirectorio oculto llamado ".git" que almacena la configuración y el historial del repositorio.
2. **git clone [URL]:**
   - Clona un repositorio Git existente desde una URL remota y crea una copia local del repositorio en tu computadora.
3. **git add [archivo]:**
   - Agrega un archivo específico al área de preparación (staging area) para ser incluido en el próximo commit.
4. **git add . o git add -A:**
   - Agrega todos los archivos modificados y no rastreados al área de preparación para ser incluidos en el próximo commit.
5. **git commit -m "mensaje":**
   - Confirma los cambios en el repositorio junto con un mensaje descriptivo que documenta los cambios realizados en ese commit.
6. **git status:**
   - Muestra el estado actual del repositorio, incluyendo los archivos modificados, los archivos preparados para commit y los archivos no rastreados.
7. **git log:**
   - Muestra un registro de todos los commits en el repositorio, incluyendo detalles como el autor, la fecha y la descripción de cada commit.
8. **git pull:**
   - Recupera los cambios desde el repositorio remoto y los fusiona con la rama local actual.
9. **git push:**
   - Envía los commits locales al repositorio remoto. Si se utiliza `git push origin rama`, se enviarán los commits de la rama local a la rama remota llamada "rama".
10. **git branch:**

- Muestra una lista de todas las ramas en el repositorio. La rama actual se resalta.

11. **git checkout [rama]:**

- Cambia de una rama a otra. Puedes usar este comando para cambiar entre diferentes ramas en el repositorio.

12. **git merge [rama]:**

- Fusiona los cambios de una rama en la rama actual. Se utiliza para combinar el trabajo de una rama en otra.

13. **git rebase [rama]:**

- Reorganiza los commits de la rama actual sobre la base de la rama especificada. Crea una línea de tiempo lineal y ordenada.

14. **git remote -v:**

- Muestra una lista de los repositorios remotos vinculados y sus URLs. Es útil para verificar las conexiones remotas.

15. **git fetch:**

- Recupera información sobre los cambios en el repositorio remoto sin fusionarlos en la rama local. Se utiliza para actualizar tu conocimiento sobre el estado del repositorio remoto.

16. **git tag [nombre_del_tag]:**

- Crea un "tag" en un commit específico para marcar puntos importantes en la historia del repositorio, como versiones de lanzamiento.

17. **git diff:**

- Muestra las diferencias entre los cambios realizados en el directorio de trabajo y el área de preparación (staging area).

18. **git reset [archivo]:**

- Permite deshacer los cambios en el área de preparación y el directorio de trabajo para un archivo específico.

19. **git remote add [nombre] [URL]:**

- Agrega un nuevo repositorio remoto con un nombre específico y la URL correspondiente. Esto facilita las operaciones de push y pull hacia ese repositorio remoto.

20. **git checkout -b [nombre_rama]:**

- Crea una nueva rama y cambia a ella inmediatamente. Es una forma abreviada de los comandos `git branch [nombre_rama]` y `git checkout [nombre_rama]`.

21. **git stash:**

- Guarda temporalmente los cambios no comprometidos en una pila de cambios ("stash"). Esto permite cambiar de rama o realizar otras acciones sin comprometer los cambios actuales.

22. **git fetch --prune:**

- Recupera información sobre los cambios en el repositorio remoto y elimina automáticamente las referencias locales a ramas eliminadas en el repositorio remoto.

23. **git remote show [nombre_repositorio]:**

- Muestra información detallada sobre un repositorio remoto, incluyendo la URL del repositorio, las ramas rastreadas y más.

24. **git log --oneline:**

- Muestra un registro de commits en una sola línea, lo que facilita una vista más concisa de la historia del repositorio.

25. **git branch -d [nombre_rama]:**

- Borra una rama específica. Utiliza `-d` para una eliminación segura que evita la pérdida de commits no fusionados. Puedes usar `-D` para una eliminación forzada.

26. **git config:**

- Permite configurar opciones de Git, como el nombre de usuario, la dirección de correo electrónico, alias de comandos y más. Es esencial para personalizar la configuración de Git.

27. **git remote remove [nombre_repositorio]:**

- Elimina un repositorio remoto de la lista de repositorios remotos vinculados. Esto es útil si ya no deseas rastrear ese repositorio.

Comandos `git reset --soft` y `git reset --hard` en Git:

`git reset --soft`:

- Este comando se utiliza para deshacer commits y restablecer el estado del repositorio al commit especificado, pero manteniendo los cambios en el área de preparación (staging area) y el directorio de trabajo. Esto significa que los cambios de los commits que deshaces se vuelven a agregar al área de preparación, y puedes realizar nuevos commits con esos cambios.

Uso de `git reset --soft`:

```
git reset --soft HEAD~3
```

En este ejemplo, `git reset --soft HEAD~3` desharía los últimos 3 commits, pero mantendría los cambios de esos commits en el área de preparación y el directorio de trabajo. Puedes realizar nuevos commits con esos cambios o realizar ajustes antes de confirmarlos.

`git reset --hard`:

- Este comando se utiliza para deshacer commits y restablecer el estado del repositorio al commit especificado, eliminando completamente los cambios en el área de preparación (staging area) y el directorio de trabajo. Ten en cuenta que esta acción es irreversible y se pierden todos los cambios realizados en los commits deshechos.

Uso de `git reset --hard`:

```
git reset --hard HEAD~3
```

En este ejemplo, `git reset --hard HEAD~3` desharía los últimos 3 commits y eliminaría por completo los cambios en el área de preparación y el directorio de trabajo. Esto se utiliza cuando deseas eliminar permanentemente cambios no deseados.

Ten mucho cuidado al usar `git reset --hard`, ya que los cambios eliminados no se pueden recuperar fácilmente. Asegúrate de estar seguro de que deseas eliminar los cambios antes de ejecutar este comando.

Ambos comandos son útiles para gestionar la historia de commits en tu repositorio, pero debes utilizarlos con precaución, especialmente `git reset --hard`, debido a su naturaleza irreversible.