🔥 Comandos Esenciales

1. Inicializar un Repositorio

- `git init`
  Inicia un nuevo repositorio en el directorio actual.

2. Clonar un Repositorio

- `git clone <url_del_repositorio>`
  Clona un repositorio remoto en tu máquina local.

3. Ver el Estado del Repositorio

- `git status`
  Muestra el estado actual de los archivos (si hay cambios pendientes, archivos no versionados, etc.).

4. Añadir Archivos al Área de Staging (Index)

- `git add <archivo>`
  Añade un archivo específico al área de staging.
- `git add .`
  Añade todos los archivos al área de staging.

5. Hacer un Commit

- `git commit -m "mensaje"`
  Guarda los cambios en el repositorio con un mensaje descriptivo.
- `git commit -am "mensaje"`
  Añade y commitea archivos ya versionados en un solo paso.

🔄 Comandos de Sincronización con Repositorios Remotos

6. Subir Cambios al Repositorio Remoto

- **`git push <remote> <branch>`**
  Sube los commits locales al repositorio remoto (por defecto: `git push origin main`).

7. Descargar Cambios del Repositorio Remoto

- **`git pull <remote> <branch>`**
  Trae los cambios remotos y los combina con la rama local (por defecto: `git pull origin main`).

8. Ver Repositorios Remotos

- **`git remote -v`**
  Lista las URLs de los repositorios remotos.

🛠️ Comandos de Gestión de Ramas (Branches)

9. Crear una Nueva Rama

- `git branch <nombre_rama>`
  Crea una nueva rama sin cambiarte a ella.
- `git checkout -b <nombre_rama>`
  Crea una nueva rama **y** cambia a ella.

10. Cambiar de Rama

- `git checkout <nombre_rama>`
  Cambia a una rama existente.

11. Combinar Ramas (Merge)

- `git merge <rama>`
  Fusiona la rama especificada con la rama actual.

12. Eliminar una Rama

- `git branch -d <nombre_rama>`
  Elimina una rama ya fusionada.
- `git branch -D <nombre_rama>`
  Elimina una rama aunque no esté fusionada.

🕵️ Comandos de Historial y Comparación

13. Ver el Historial de Commits

- `git log`
  Muestra el historial de commits.
- `git log --oneline --graph --decorate --all`
  Muestra el historial de commits de manera simplificada y visual.

14. Ver Diferencias entre Archivos

- `git diff`
  Muestra los cambios entre el área de trabajo y el área de staging.
- `git diff --staged`
  Muestra los cambios entre el área de staging y el último commit.

15. Ver Cambios en un Archivo

- `git show <hash_del_commit>`
  Muestra los cambios de un commit específico.

⏪ Deshacer Cambios

16. Eliminar Archivos del Área de Staging

- `git reset <archivo>`
  Quita el archivo del área de staging, pero mantiene los cambios en el área de trabajo.

17. Deshacer un Commit (Manteniendo Cambios)

- `git reset --soft HEAD~1`
  Deshace el último commit, pero mantiene los archivos en staging.

18. Deshacer un Commit (Perdiendo Cambios)

- `git reset --hard HEAD~1`
  Deshace el último commit y elimina los cambios tanto del área de staging como del área de trabajo.

19. Revertir un Commit (sin Deshacerlo)

- `git revert <hash_del_commit>`
  Crea un nuevo commit que deshace los cambios de un commit anterior sin alterar el historial.

🧹 Comandos de Limpieza

20. Eliminar Archivos No Versionados

- `git clean -f`
  Elimina los archivos no versionados del área de trabajo.
- `git clean -fd`
  Elimina archivos y directorios no versionados.

🌐 Comandos de Colaboración y Manejo de Remotos

21. Agregar un Repositorio Remoto

- `git remote add <nombre> <url>`
  Añade un repositorio remoto con un nombre personalizado.

22. Cambiar la URL de un Repositorio Remoto

- `git remote set-url <nombre> <nueva_url>`
  Cambia la URL de un repositorio remoto.

✨ Comandos de Etiquetas (Tags)

23. Crear una Etiqueta

- `git tag <nombre_etiqueta>`
  Crea una etiqueta ligera.
- `git tag -a <nombre_etiqueta> -m "mensaje"`
  Crea una etiqueta anotada con un mensaje.

24. Subir Etiquetas al Repositorio Remoto

- `git push <remote> <tag>`
  Sube una etiqueta específica al repositorio remoto.
- `git push --tags`
  Sube todas las etiquetas al repositorio remoto.

🚑 Comandos de Emergencia

25. Guardar Cambios Temporalmente

- **`git stash`**
  Guarda temporalmente los cambios no commiteados (útil para hacer un cambio rápido en otra rama).
- **`git stash pop`**
  Restaura los cambios guardados con `git stash`.

26. Rehacer un Merge con Conflictos

- **`git merge --abort`**
  Cancela un merge en proceso y restaura el estado previo.

🔍 Extras

27. Buscar un Cambio por Mensaje

- **`git log --grep="<palabra_clave>"`**
  Busca commits que contengan una palabra o frase específica en su mensaje.

28. Cambiar el Mensaje del Último Commit

- **`git commit --amend -m "nuevo mensaje"`**
  Modifica el mensaje del último commit.

📝 Resumen de Flujos Comunes

Flujo Típico de Trabajo

1. **Clonar o Inicializar Repositorio**
   `git clone <url>` o `git init`
2. **Ver Estado de Cambios**
   `git status`
3. **Añadir Cambios**
   `git add .`
4. **Hacer Commit**
   `git commit -m "mensaje"`
5. **Subir Cambios al Repositorio Remoto**
   `git push origin <rama>`

Manejo de Ramas

1. **Crear Rama**
   `git checkout -b <nueva_rama>`
2. **Trabajar en la Rama**
   Hacer cambios, `git add`, `git commit`.
3. **Fusionar Cambios a `main`**
   `git checkout main`
   `git merge <nueva_rama>`
4. **Eliminar Rama Local**
   `git branch -d <nueva_rama>`

📝 Atajos Útiles

- **`git log --oneline`** → Ver historial resumido.
- **`git diff HEAD`** → Ver diferencias entre el área de trabajo y el último commit.
- **`git blame <archivo>`** → Ver quién modificó cada línea de un archivo.

🚀 Comandos Avanzados

29. Cambiar de Rama y Guardar Trabajo Pendiente

- `git checkout <rama> -m "mensaje"`
  Cambia de rama y automáticamente guarda cualquier cambio sin commitear en un stash.

30. Ver los Cambios entre dos Commits

- `git diff <commit_1> <commit_2>`
  Compara los cambios entre dos commits específicos.

31. Rebase (Reorganizar Commits)

- `git rebase <rama>`
  Reaplica los commits de tu rama actual sobre la rama especificada (ideal para mantener un historial lineal).
- `git rebase -i HEAD~n`
  Rebase interactivo, permite reorganizar, modificar o combinar los últimos n commits.

32. Cambiar la Base de una Rama

- `git rebase --onto <nueva_base> <desde_rama> <rama>`
  Mueve una rama a una nueva base, aplicando los commits desde otra rama.

33. Ignorar Archivos en Git

- `.gitignore`
  Archivo especial que indica qué archivos o carpetas ignorar en Git. Ejemplo de contenido:
  bash

  ```
  # Ignorar archivos binarios

  *.log

  /carpeta_privada/
  ```

🔄 Comandos de Reescritura y Modificación del Historial

34. Eliminar Commits del Historial (Reescribir Historia)

- `git reset <commit>`
  Mueve la rama actual a un commit anterior, deshaciendo los commits posteriores.
- `git reset --hard <commit>`
  Borra completamente los cambios posteriores a un commit específico.

35. Revertir Múltiples Commits

- `git revert -n <commit>^..<commit>`
  Revierte los cambios de varios commits sin hacer un nuevo commit (usado para correcciones grandes).

36. Borrar un Commit Público (No Recomendado)

- `git push origin --force`
  Empuja cambios forzadamente y reescribe el historial remoto (usado con precaución).

👫 Colaboración y Resolución de Conflictos

37. Marcar un Conflicto como Resuelto

- **`git add <archivo_conflictivo>`**
  Después de resolver manualmente un conflicto en un archivo, lo añades para indicar que está resuelto.

38. Saltar un Commit Durante el Rebase

- **`git rebase --skip`**
  Si encuentras un conflicto en un commit durante un rebase, puedes saltar ese commit y continuar.

39. Rehacer un Merge Desastroso

- **`git reflog`**
  Muestra el historial de todas las operaciones de Git, incluidas las que no están en el log tradicional (ideal para deshacer errores graves).
- **`git reset --hard <reflog_id>`**
  Vuelve a un estado anterior usando el identificador de `reflog`.

🎨 Herramientas de Personalización y Configuración

40. Configurar Nombre y Correo Electrónico Global

- **`git config --global user.name "Nombre"`**
  Establece el nombre de usuario global.
- **`git config --global user.email "email@example.com"`**
  Establece el correo electrónico global.

41. Configurar Alias de Comandos

- **`git config --global alias.<alias> "<comando_git>"`**
  Define alias personalizados para acortar comandos. Ejemplos:
  - **`git config --global alias.co checkout`**
    Permite usar `git co` en lugar de `git checkout`.
  - **`git config --global alias.st status`**
    Para usar `git st` en lugar de `git status`.

🧩 Comandos Especiales para Entornos Avanzados

42. Trabajar con Submódulos (Submodules)

- `git submodule add <url>`
  Añade un repositorio Git dentro de otro (submódulo).
- `git submodule update --init`
  Inicializa y clona todos los submódulos definidos en el proyecto.
- `git submodule foreach <comando>`
  Ejecuta un comando en cada submódulo.

43. Shallow Clone (Clonar sin Historial Completo)

- `git clone --depth 1 <url>`
  Clona el repositorio sin todo el historial (solo los últimos commits), útil para proyectos grandes.

🛡️ Comandos de Seguridad y Verificación

44. Verificar Firmas de Commits

- `git log --show-signature`
  Muestra si los commits están firmados y verificados.

45. Firmar Commits

- `git commit -S -m "mensaje"`
  Crea un commit firmado digitalmente (requiere configurar GPG).

💼 Comandos para Automatización y Despliegue

46. Hooks de Git

`pre-commit`, **`post-commit`**, **`pre-push`**
Git permite crear hooks (scripts automatizados) que se ejecutan en momentos clave del flujo de trabajo. Los archivos de hooks se almacenan en `.git/hooks/`.
Ejemplo: Un **hook** **`pre-commit`** que verifica el código antes de cada commit.
bash

#!/bin/bash

npm run lint

⚡ Comandos de Optimización

47. Compactar el Repositorio (Garbage Collection)

- **`git gc`**
  Ejecuta la recolección de basura para limpiar objetos innecesarios y reducir el tamaño del repositorio.

48. Optimizar el Rendimiento en Repositorios Grandes

- **`git repack`**
  Reempaqueta objetos en un repositorio para mejorar el rendimiento.

🔍 Comandos para Análisis y Debugging

49. Buscar Commits que Introducen Bugs (Bisect)

- `git bisect start`
  Inicia una búsqueda binaria para encontrar el commit que introdujo un bug.
- `git bisect good <commit>`
  Marca un commit como "bueno".
- `git bisect bad <commit>`
  Marca un commit como "malo". Git reducirá los commits sospechosos hasta encontrar el origen del bug.

50. Rastrear Cambios en una Línea (Blame)

- `git blame <archivo>`
  Muestra línea por línea quién cambió qué en un archivo.

51. Buscar en el Historial del Repositorio

- `git grep <palabra>`
  Busca palabras o patrones específicos dentro del repositorio.

🌍 Trabajar con Flujos Distribuidos

52. Fetch para Mantener Repositorio Local Actualizado

- `git fetch <remote>`
  Descarga los cambios de un repositorio remoto, pero no los fusiona automáticamente.

53. Actualizar el Rastreado de una Rama Remota

- `git branch -u <remote>/<branch>`
  Establece la rama remota como la rama de seguimiento.

🧭 Resumen de Flujos Avanzados

Flujo para Rebases:

1. **Crear rama**:
   `git checkout -b feature-rama`
2. **Rebase para mantener tu rama actualizada con `main`**:
   `git checkout feature-rama`
   `git rebase main`
3. **Resolver conflictos si es necesario**:
   `git rebase --continue` o `git rebase --skip`
4. **Fusionar cambios a `main`**:
   `git checkout main`
   `git merge feature-rama`

Flujo de Submódulos:

1. **Añadir un submódulo**:
   `git submodule add <url>`
2. **Actualizar submódulos**:
   `git submodule update --remote`
3. **Eliminar un submódulo**:
   Edita `.gitmodules` y `.git/config`, y borra la carpeta del submódulo.

Preguntas Tipo Test sobre Git

1. **¿Qué comando se utiliza para inicializar un nuevo repositorio Git?**
   - A) `git start`
   - B) `git init`
   - C) `git new`
   - D) `git create`
   - **Respuesta:** B) `git init`
2. **¿Qué comando clona un repositorio existente en tu máquina local?**
   - A) `git copy`
   - B) `git clone`
   - C) `git pull`
   - D) `git fetch`
   - **Respuesta:** B) `git clone`
3. **¿Cuál es la función del comando `git status`?**
   - A) Muestra el historial de commits.
   - B) Muestra el estado del árbol de trabajo y el área de staging.
   - C) Realiza un commit.
   - D) Restaura archivos a su versión anterior.
   - **Respuesta:** B) Muestra el estado del árbol de trabajo y el área de staging.
4. **Para añadir todos los cambios de archivos al área de staging, se utiliza:**
   - A) `git add *`
   - B) `git add .`
   - C) `git stage .`
   - D) `git commit -a`
   - **Respuesta:** B) `git add .`
5. **¿Qué hace el comando `git commit -m "mensaje"`?**
   - A) Realiza un commit y muestra el historial.
   - B) Guarda los cambios en el área de staging con un mensaje.
   - C) Muestra los archivos que han cambiado.
   - D) Aplica cambios de otro branch.
   - **Respuesta:** B) Guarda los cambios en el área de staging con un mensaje.
6. **¿Cómo se crea una nueva rama en Git?**
   - A) `git branch <nombre_rama>`
   - B) `git create branch <nombre_rama>`
   - C) `git new branch <nombre_rama>`
   - D) `git checkout <nombre_rama>`
   - **Respuesta:** A) `git branch <nombre_rama>`
7. **¿Qué comando se utiliza para fusionar dos ramas?**
   - A) `git merge <rama>`
   - B) `git branch <rama>`
   - C) `git commit <rama>`
   - D) `git combine <rama>`
   - **Respuesta:** A) `git merge <rama>`

8. **¿Cuál es la diferencia entre `git pull` y `git fetch`?**
    - A) `git pull` solo descarga cambios, mientras que `git fetch` descarga y fusiona.
    - B) `git fetch` solo descarga cambios, mientras que `git pull` descarga y fusiona.
    - C) No hay diferencia, son lo mismo.
    - D) `git pull` solo funciona en ramas locales.
    - **Respuesta:** B) `git fetch` solo descarga cambios, mientras que `git pull` descarga y fusiona.
9. **¿Qué comando se utiliza para eliminar un archivo del área de staging?**
    - A) `git remove <archivo>`
    - B) `git unstage <archivo>`
    - C) `git reset <archivo>`
    - D) `git delete <archivo>`
    - **Respuesta:** C) `git reset <archivo>`
10. **¿Qué hace el comando `git stash`?**
    - A) Hace un commit de todos los cambios actuales.
    - B) Guarda cambios no confirmados temporalmente.
    - C) Restaura archivos borrados.
    - D) Crea una nueva rama.
    - **Respuesta:** B) Guarda cambios no confirmados temporalmente.
11. **¿Cómo se muestra el historial de commits en Git?**
    - A) `git history`
    - B) `git log`
    - C) `git show`
    - D) `git commits`
    - **Respuesta:** B) `git log`
12. **Para revertir un commit específico, se utiliza:**
    - A) `git reset <commit>`
    - B) `git undo <commit>`
    - C) `git revert <commit>`
    - D) `git back <commit>`
    - **Respuesta:** C) `git revert <commit>`
13. **¿Qué comando se usa para crear una etiqueta (tag) en Git?**
    - A) `git label <nombre_etiqueta>`
    - B) `git tag <nombre_etiqueta>`
    - C) `git create-tag <nombre_etiqueta>`
    - D) `git mark <nombre_etiqueta>`
    - **Respuesta:** B) `git tag <nombre_etiqueta>`

14. **¿Qué es un conflicto de fusión?**
    - A) Cuando no se pueden realizar commits.
    - B) Cuando dos ramas han cambiado el mismo archivo y Git no puede resolverlo automáticamente.
    - C) Cuando un repositorio está corrupto.
    - D) Cuando un usuario no tiene permisos para fusionar.
    - **Respuesta:** B) Cuando dos ramas han cambiado el mismo archivo y Git no puede resolverlo automáticamente.
15. **¿Qué hace el comando `git clean -fd`?**
    - A) Elimina archivos no rastreados y directorios.
    - B) Restaura archivos eliminados.
    - C) Elimina commits no confirmados.
    - D) Limpia el área de staging.
    - **Respuesta:** A) Elimina archivos no rastreados y directorios.
16. **¿Qué es `git cherry-pick`?**
    - A) Aplica un commit de otra rama a la rama actual.
    - B) Crea una copia de un commit.
    - C) Elimina un commit específico.
    - D) Muestra cambios de la rama actual.
    - **Respuesta:** A) Aplica un commit de otra rama a la rama actual.
17. **¿Cómo se muestra el estado de las ramas locales y remotas?**
    - A) `git branch -a`
    - B) `git branches`
    - C) `git show branches`
    - D) `git all-branches`
    - **Respuesta:** A) `git branch -a`
18. **¿Qué comando se utiliza para restablecer el último commit sin perder cambios?**
    - A) `git reset --hard HEAD`
    - B) `git reset HEAD~1`
    - C) `git revert HEAD`
    - D) `git undo`
    - **Respuesta:** B) `git reset HEAD~1`
19. **¿Cuál es el propósito de `git bisect`?**
    - A) Para buscar errores en el código.
    - B) Para revertir múltiples commits.
    - C) Para encontrar el commit que introdujo un bug.
    - D) Para fusionar ramas.
    - **Respuesta:** C) Para encontrar el commit que introdujo un bug.

20. **¿Qué comando se usa para restaurar un archivo a su última versión confirmada?**
    - A) `git restore <archivo>`
    - B) `git reset <archivo>`
    - C) `git checkout <archivo>`
    - D) Ambas A y C son correctas.
    - **Respuesta:** D) Ambas A y C son correctas.

Preguntas Tipo Test sobre Git

1. **¿Qué comando se utiliza para inicializar un nuevo repositorio Git?**
    - A) `git start`
    - B) `git init`
    - C) `git new`
    - D) `git create`
    - **Respuesta: B**
2. **¿Qué comando clona un repositorio existente en tu máquina local?**
    - A) `git copy`
    - B) `git clone`
    - C) `git pull`
    - D) `git fetch`
    - **Respuesta: B**
3. **¿Cuál es la función del comando `git status`?**
    - A) Muestra el historial de commits.
    - B) Muestra el estado del árbol de trabajo y el área de staging.
    - C) Realiza un commit.
    - D) Restaura archivos a su versión anterior.
    - **Respuesta: B**
4. **Para añadir todos los cambios de archivos al área de staging, se utiliza:**
    - A) `git add *`
    - B) `git add .`
    - C) `git stage .`
    - D) `git commit -a`
    - **Respuesta: B**
5. **¿Qué hace el comando `git commit -m "mensaje"`?**
    - A) Realiza un commit y muestra el historial.
    - B) Guarda los cambios en el área de staging con un mensaje.
    - C) Muestra los archivos que han cambiado.
    - D) Aplica cambios de otro branch.
    - **Respuesta: B**

6. **¿Cómo se crea una nueva rama en Git?**
   - A) `git branch <nombre_rama>`
   - B) `git create branch <nombre_rama>`
   - C) `git new branch <nombre_rama>`
   - D) `git checkout <nombre_rama>`
   - **Respuesta: A**
7. **¿Qué comando se utiliza para fusionar dos ramas?**
   - A) `git merge <rama>`
   - B) `git branch <rama>`
   - C) `git commit <rama>`
   - D) `git combine <rama>`
   - **Respuesta: A**
8. **¿Cuál es la diferencia entre `git pull` y `git fetch`?**
   - A) `git pull` solo descarga cambios, mientras que `git fetch` descarga y fusiona.
   - B) `git fetch` solo descarga cambios, mientras que `git pull` descarga y fusiona.
   - C) No hay diferencia, son lo mismo.
   - D) `git pull` solo funciona en ramas locales.
   - **Respuesta: B**
9. **¿Qué comando se utiliza para eliminar un archivo del área de staging?**
   - A) `git remove <archivo>`
   - B) `git unstage <archivo>`
   - C) `git reset <archivo>`
   - D) `git delete <archivo>`
   - **Respuesta: C**
10. **¿Qué hace el comando `git stash`?**
    - A) Hace un commit de todos los cambios actuales.
    - B) Guarda cambios no confirmados temporalmente.
    - C) Restaura archivos borrados.
    - D) Crea una nueva rama.
    - **Respuesta: B**
11. **¿Cómo se muestra el historial de commits en Git?**
    - A) `git history`
    - B) `git log`
    - C) `git show`
    - D) `git commits`
    - **Respuesta: B**
12. **Para revertir un commit específico, se utiliza:**
    - A) `git reset <commit>`
    - B) `git undo <commit>`
    - C) `git revert <commit>`
    - D) `git back <commit>`
    - **Respuesta: C**

13. **¿Qué comando se usa para crear una etiqueta en Git?**
    - A) `git label <nombre_etiqueta>`
    - B) `git tag <nombre_etiqueta>`
    - C) `git create-tag <nombre_etiqueta>`
    - D) `git mark <nombre_etiqueta>`
    - **Respuesta: B**
14. **¿Qué es un conflicto de fusión?**
    - A) Cuando no se pueden realizar commits.
    - B) Cuando dos ramas han cambiado el mismo archivo y Git no puede resolverlo automáticamente.
    - C) Cuando un repositorio está corrupto.
    - D) Cuando un usuario no tiene permisos para fusionar.
    - **Respuesta: B**
15. **¿Qué hace el comando `git clean -fd`?**
    - A) Elimina archivos no rastreados y directorios.
    - B) Restaura archivos eliminados.
    - C) Elimina commits no confirmados.
    - D) Limpia el área de staging.
    - **Respuesta: A**
16. **¿Qué comando se usa para restablecer el último commit sin perder cambios?**
    - A) `git reset --hard HEAD`
    - B) `git reset HEAD~1`
    - C) `git revert HEAD`
    - D) `git undo`
    - **Respuesta: B**
17. **¿Cuál es el propósito de `git bisect`?**
    - A) Para buscar errores en el código.
    - B) Para revertir múltiples commits.
    - C) Para encontrar el commit que introdujo un bug.
    - D) Para fusionar ramas.
    - **Respuesta: C**
18. **¿Qué comando se utiliza para ver las ramas remotas?**
    - A) `git branch -r`
    - B) `git remote show`
    - C) `git list-remote`
    - D) `git branches`
    - **Respuesta: A**
19. **¿Cómo se crea una nueva rama y se cambia a ella al mismo tiempo?**
    - A) `git new branch <nombre_rama>`
    - B) `git checkout -b <nombre_rama>`
    - C) `git branch <nombre_rama> -c`
    - D) `git switch -b <nombre_rama>`
    - **Respuesta: B**
    -

20. **¿Qué comando muestra las diferencias entre el área de staging y el último commit?**
    - A) `git status`
    - B) `git diff`
    - C) `git log`
    - D) `git show`
    - **Respuesta: B**
21. **¿Qué significa `HEAD` en Git?**
    - A) La última versión de un archivo.
    - B) El último commit en la rama actual.
    - C) La rama principal del repositorio.
    - D) Un comando de Git.
    - **Respuesta: B**
22. **¿Qué hace el comando `git cherry-pick <commit>`?**
    - A) Aplica el commit especificado a la rama actual.
    - B) Crea una copia del commit especificado.
    - C) Elimina el commit especificado.
    - D) Fusiona dos ramas.
    - **Respuesta: A**
23. **¿Qué comando muestra la configuración de Git?**
    - A) `git config`
    - B) `git settings`
    - C) `git show config`
    - D) `git info`
    - **Respuesta: A**
24. **¿Cómo se elimina una rama en Git?**
    - A) `git branch -d <nombre_rama>`
    - B) `git delete <nombre_rama>`
    - C) `git remove <nombre_rama>`
    - D) `git branch remove <nombre_rama>`
    - **Respuesta: A**
25. **¿Qué hace el comando `git push origin --delete <nombre_rama>`?**
    - A) Elimina una rama local.
    - B) Elimina una rama remota.
    - C) Elimina un commit en el remoto.
    - D) Cancela el último push.
    - **Respuesta: B**
26. **¿Cómo se traen los cambios de una rama remota?**
    - A) `git fetch`
    - B) `git pull`
    - C) `git update`
    - D) Ambas A y B son correctas.
    - **Respuesta: D**

27. **¿Qué hace el comando `git fetch --prune`?**
    - A) Elimina ramas locales no rastreadas.
    - B) Actualiza la información de ramas remotas y elimina las que ya no existen.
    - C) Muestra los commits no confirmados.
    - D) Limpia el área de staging.
    - **Respuesta: B**
28. **¿Cómo se configura un alias para un comando en Git?**
    - A) `git alias <nombre> <comando>`
    - B) `git config --global alias.<nombre> <comando>`
    - C) `git set alias <nombre> <comando>`
    - D) `git command alias <nombre> <comando>`
    - **Respuesta: B**
29. **¿Qué es un submódulo en Git?**
    - A) Una rama secundaria.
    - B) Un repositorio dentro de otro repositorio.
    - C) Un commit específico.
    - D) Un tipo de etiqueta.
    - **Respuesta: B**
30. **¿Qué comando se utiliza para ver la configuración remota de un repositorio?**
    - A) `git remote`
    - B) `git remote show`
    - C) `git remote -v`
    - D) Todas las anteriores.
    - **Respuesta: D**
31. **¿Qué significa `git reset` sin ningún argumento?**
    - A) Restaura el último commit.
    - B) Elimina todos los cambios en el área de staging.
    - C) Restaura el área de staging al último commit, pero mantiene los cambios en el directorio de trabajo.
    - D) Elimina todos los cambios locales.
    - **Respuesta: C**
32. **¿Cómo se mueve un archivo de una rama a otra?**
    - A) `git move <archivo> <nueva_rama>`
    - B) `git checkout <nueva_rama> -- <archivo>`
    - C) `git branch --move <archivo> <nueva_rama>`
    - D) `git checkout -b <nueva_rama> <archivo>`
    - **Respuesta: B**
33. **¿Qué comando se utiliza para restaurar todos los archivos a su versión en el último commit?**
    - A) `git restore .`
    - B) `git reset .`
    - C) `git checkout .`
    - D) Todas las anteriores son correctas.
    - **Respuesta: D**

34. **¿Qué comando se utiliza para restaurar un archivo específico al último commit?**
    - A) `git checkout <archivo>`
    - B) `git reset <archivo>`
    - C) `git restore <archivo>`
    - D) Ambas A y C son correctas.
    - **Respuesta: D**
35. **¿Cuál de las siguientes afirmaciones es verdadera sobre las ramas en Git?**
    - A) Las ramas son copias completas del repositorio.
    - B) Las ramas permiten trabajar en diferentes características sin afectar la rama principal.
    - C) No se pueden fusionar ramas en Git.
    - D) Las ramas no pueden eliminarse una vez creadas.
    - **Respuesta: B**
36. **¿Qué significa el comando `git merge --no-ff <rama>`?**
    - A) Realiza un merge sin crear un commit de merge.
    - B) Realiza un merge y siempre crea un commit de merge.
    - C) Elimina la rama después de hacer el merge.
    - D) Crea un nuevo branch.
    - **Respuesta: B**
37. **¿Qué comando se utiliza para ver las diferencias entre dos commits?**
    - A) `git diff <commit1> <commit2>`
    - B) `git log <commit1> <commit2>`
    - C) `git compare <commit1> <commit2>`
    - D) `git show <commit1> <commit2>`
    - **Respuesta: A**
38. **¿Qué comando se utiliza para aplicar un stash guardado?**
    - A) `git apply stash`
    - B) `git stash apply`
    - C) `git restore stash`
    - D) `git merge stash`
    - **Respuesta: B**
39. **¿Qué hace el comando `git reset --hard`?**
    - A) Restaura todos los archivos al último commit.
    - B) Elimina todos los cambios no confirmados.
    - C) Ambas A y B son correctas.
    - D) Solo elimina cambios en el área de staging.
    - **Respuesta: C**
40. **¿Cómo se verifica el tamaño de un repositorio en Git?**
    - A) `git size`
    - B) `git repo-size`
    - C) `git count-objects -v`
    - D) `git check`
    - **Respuesta: C**
    -

41. **¿Qué comando se utiliza para buscar un commit específico?**
    - A) `git find <commit>`
    - B) `git search <commit>`
    - C) `git log --grep="<mensaje>"`
    - D) `git locate <commit>`
    - **Respuesta: C**
42. **¿Cuál es el comando para cambiar a una rama diferente?**
    - A) `git switch <rama>`
    - B) `git checkout <rama>`
    - C) `git change <rama>`
    - D) Ambas A y B son correctas.
    - **Respuesta: D**
43. **¿Qué comando se utiliza para renombrar una rama?**
    - A) `git rename <rama_antigua> <rama_nueva>`
    - B) `git branch -m <rama_antigua> <rama_nueva>`
    - C) `git branch rename <rama_antigua> <rama_nueva>`
    - D) `git change-branch <rama_antigua> <rama_nueva>`
    - **Respuesta: B**
44. **¿Qué hace el comando `git commit --amend`?**
    - A) Modifica el último commit con nuevos cambios.
    - B) Elimina el último commit.
    - C) Crea un nuevo commit con el mensaje del último commit.
    - D) Deshace todos los cambios en el último commit.
    - **Respuesta: A**
45. **¿Qué hace el comando `git tag -d <nombre_etiqueta>`?**
    - A) Crea una nueva etiqueta.
    - B) Elimina una etiqueta existente.
    - C) Muestra todas las etiquetas.
    - D) Muestra la información de una etiqueta.
    - **Respuesta: B**
46. **¿Cuál es el propósito de un "pull request"?**
    - A) Solicitar la eliminación de una rama.
    - B) Solicitar la revisión y fusión de cambios en una rama.
    - C) Solicitar cambios en el repositorio remoto.
    - D) Solicitar el renombramiento de una rama.
    - **Respuesta: B**
47. **¿Qué comando se utiliza para verificar los cambios en un archivo específico?**
    - A) `git show <archivo>`
    - B) `git status <archivo>`
    - C) `git diff <archivo>`
    - D) `git log <archivo>`
    - **Respuesta: C**

48. **¿Qué hace el comando `git blame <archivo>`?**
    - A) Muestra quién hizo cambios en cada línea de un archivo.
    - B) Muestra el historial de cambios en un archivo.
    - C) Muestra el contenido de un archivo.
    - D) Restaura un archivo a su versión anterior.
    - **Respuesta: A**
49. **¿Qué es un "fork" en GitHub?**
    - A) Una copia de un repositorio para realizar cambios de manera independiente.
    - B) Un tipo de rama.
    - C) Un commit especial.
    - D) Un submódulo.
    - **Respuesta: A**
50. **¿Qué significa `git init --bare`?**
    - A) Crea un repositorio sin un árbol de trabajo.
    - B) Crea un nuevo branch.
    - C) Inicializa un repositorio local.
    - D) Elimina un repositorio existente.
    - **Respuesta: A**