VAMOS A CREAR UN REPOSITORIO LLAMADO **"PROYECTO-WEB"** DONDE **APLICAREMOS GIT STASH Y GIT REBASE:**

**Paso 1: Crear el repositorio y estructura inicial**

**Abrimos GitBash**

# Crear carpeta e inicializar repositorio

mkdir proyecto-web

cd proyecto-web

git init

# Crear archivos base

echo "<h1>Bienvenido</h1>" > index.html

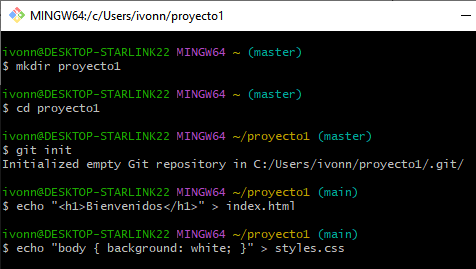
echo "body { background: white; }" > styles.css

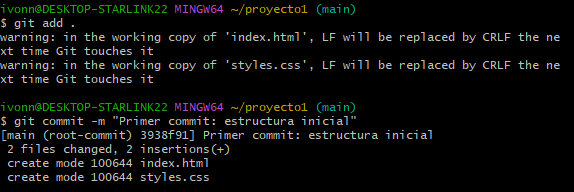
# Primer commit

git add .

git commit -m "Primer commit: estructura inicial"

**En GitBash se observará de esta manera:**





**Paso 2: Hacer cambios y guardarlos en stash**

1. Vamos a modificar archivos:

echo "<p>Nuevo párrafo</p>" >> index.html

echo "h1 { color: blue; }" >> styles.css

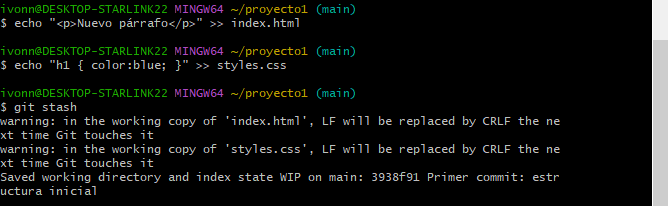
1. Guardamos los cambios en stash (sin hacer commit):

git stash

Tendremos la siguiente Salida:

Saved working directory and index state WIP on main: 5a3d2e1 Primer commit

**Entonces escribimos en GitBash:**



**Paso 3: Vamos a crear una rama para trabajar en una nueva función.**

git checkout -b feature/navbar

1. Vamos a añadir navbar en index.html:

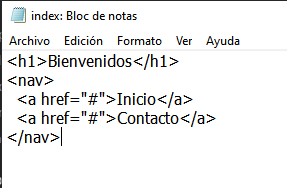
Entonces en html pondremos

<nav>

<a href="#">Inicio</a>

<a href="#">Contacto</a>

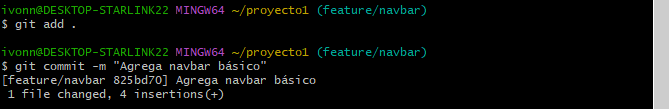
</nav>



1. **En GitBash vamos a hacer commit**:

git add .

git commit -m "Agrega navbar básico"



**Paso 4: Recuperar cambios del stash en la rama actual**

1. **Aplicar el stash guardado**:

**En GitBash escribimos**

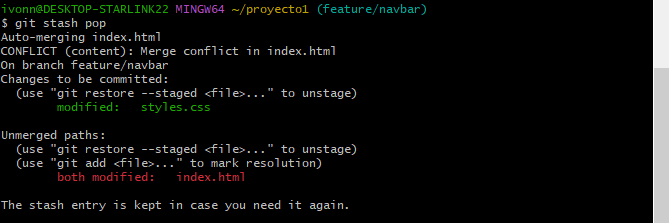
git stash pop

**Salida**:

Auto-merging index.html

CONFLICT (content): Merge conflict in index.html

(Simulamos un conflicto para resolverlo).



1. **Resolver el conflicto manualmente:**

Abre index.html y combina los cambios:

<h1>Bienvenido</h1>

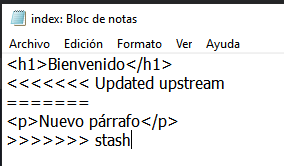
<<<<<<< Updated upstream

=======

<p>Nuevo párrafo</p>

>>>>>>> stash

Entonces tenemos asi el Html y se edito manualmente



**¿Qué cambió?**

1. **Eliminamos los marcadores de Git** (<<<<<<<, =======, >>>>>>>) que aparecen por un conflicto al hacer merge.
2. **Mantuvimos ambos cambios**:
   * El <h1> con el texto "Bienvenidos" (versión actualizada).
   * El nuevo párrafo <p>Nuevo párrafo</p> (que estaba en el "stash").
3. **Dejamos la estructura completa** con el <nav> y los enlaces.

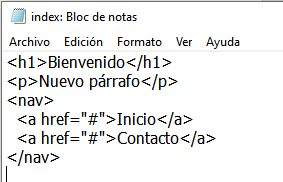
**Versión final**:

En Html

<h1>Bienvenido</h1>

<p>Nuevo párrafo</p>

Por lo que el Html quedo asi:

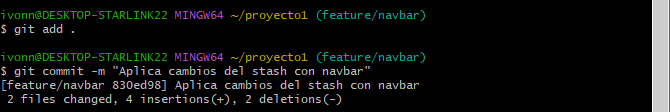


**Finalizar la resolución**:

**En GitBash**

git add .

git commit -m "Aplica cambios del stash con navbar"



**Paso 5: Usar Git Rebase para integrar cambios de main o Master**

1. **Cambiar a main y actualizarla**:

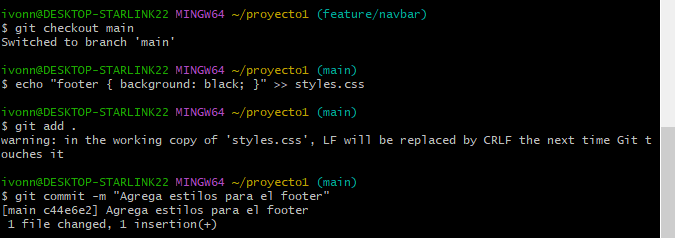
**En GitBash**

git checkout main

echo "footer { background: black; }" >> styles.css

git add .

git commit -m "Agrega estilos para el footer"

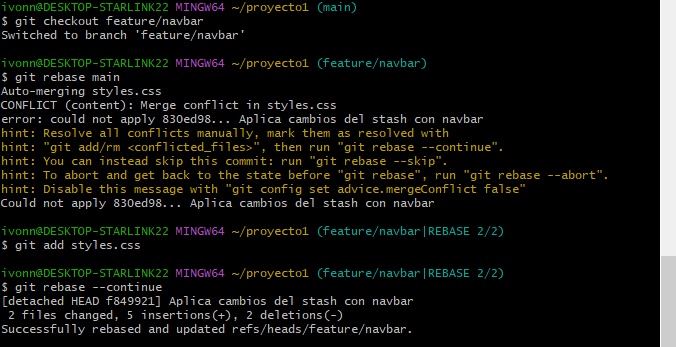


Ahora vamos a rebasear la rama feature/navbar:

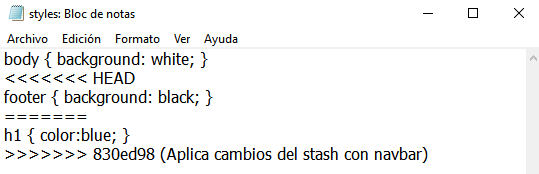
**En GitBash**

git checkout feature/navbar

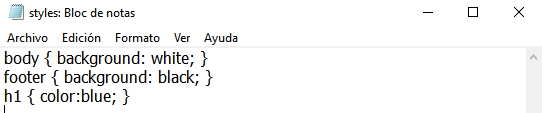
git rebase main



**Marca error por esto en el archivo styles.css, así que hay que editar y cambiarlo de esto:**



**A que al final quede asi y lo guardamos:**



**La Salida debe ser**:

Successfully rebased and updated refs/heads/feature/navbar.

**Pero si falla y muestra error:**

Hay que modificar el archive styles.css:

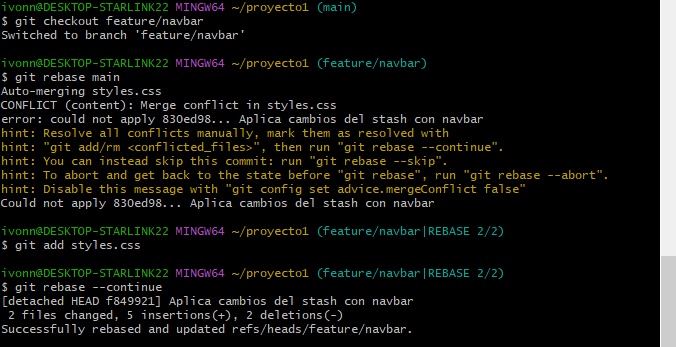
**# 1.** Resolvemos el conflicto en styles.css y guardamos el cambio en el archivo

**# 2.** De esta forma se marca el archivo como resuelto con el comando siguiente:

git add styles.css

**# 3.** Continúa el rebase

git rebase --continue



**Paso 6: Resolver conflictos durante el rebase (si los hay)**

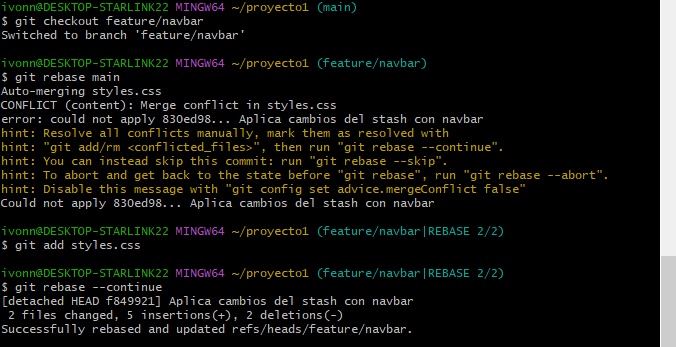
**Si Git muestra un conflicto durante el rebase:**

1. Edita los archivos marcados.
2. Ejecuta:

**En GitBash**

git add .

git rebase --continue

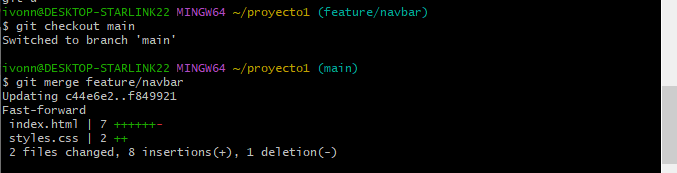


**Paso 7: Fusionar los cambios en main**

**En GitBash**

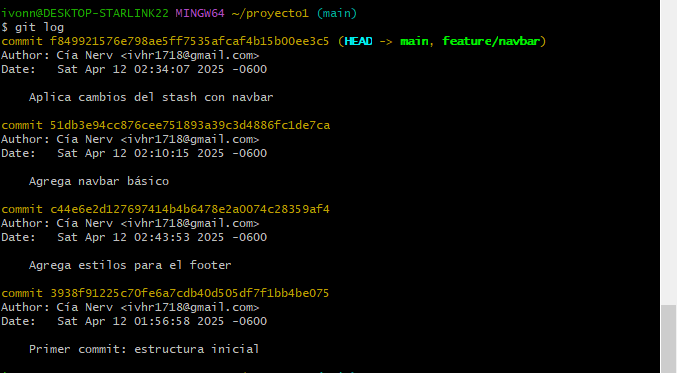
git checkout main

git merge feature/navbar



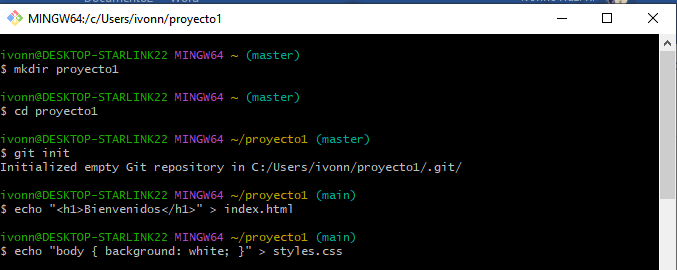
**Resultado Final verificaremos con el comando git log y damos enter, que nos muestre el historial de todos los commits que se hicieron en el proyecto1 dentro del main:**

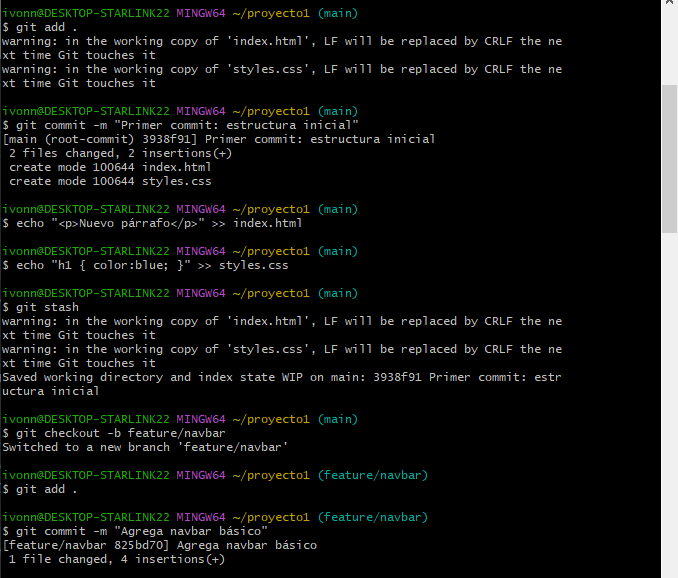
* **Historial de commits (git log)** hechos en el ejercicio.

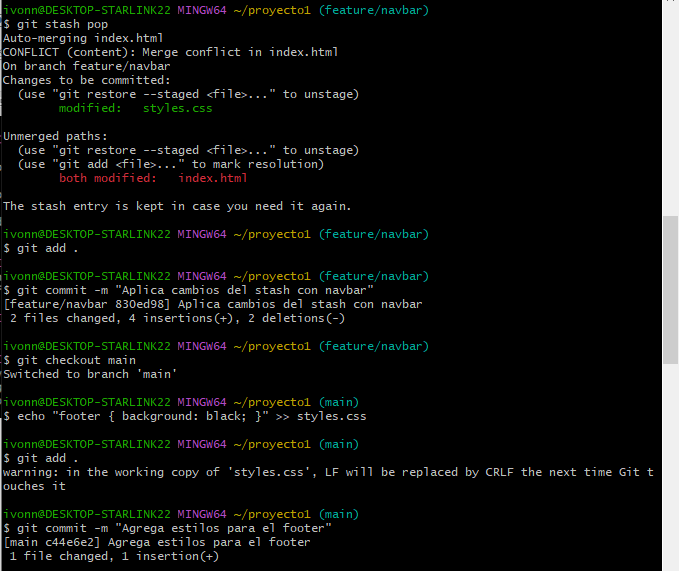


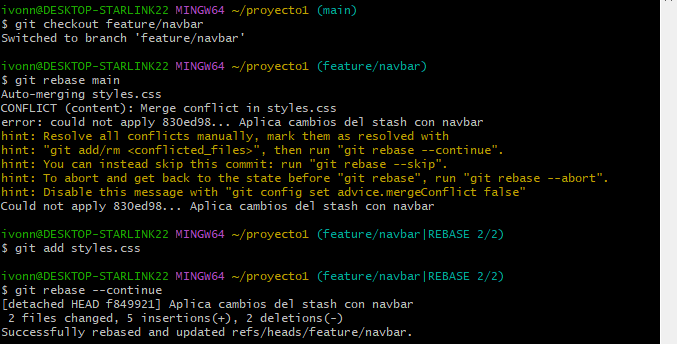
1. **Stash**: Guardar cambios temporales sin perder trabajo.
2. **Rebase**: Mantiene un historial lineal y limpio.
3. **Ramificación**: Trabajar en features aisladas.

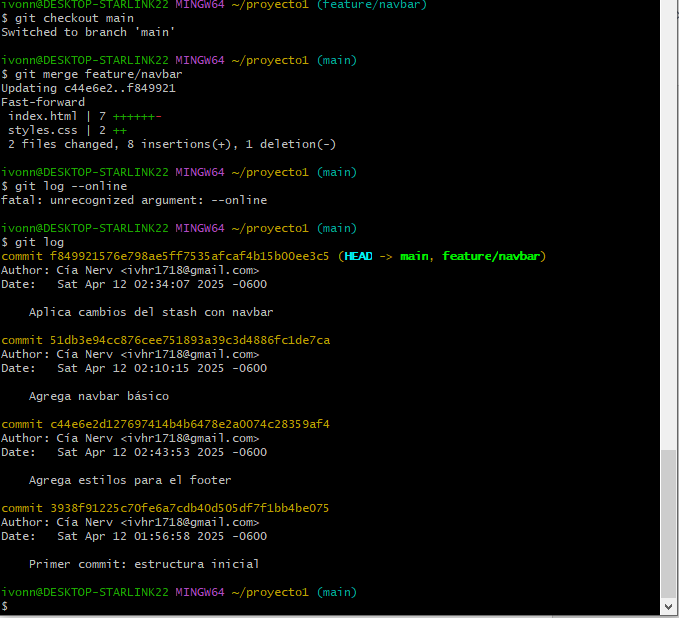
**A continuación se muestra el código de GitBash para este ejercicio:**

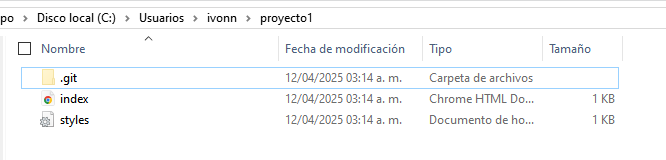












**Stash**

Qué es:

El stash es un área temporal donde Git guarda cambios no commitidos (tanto en el working directory como en el staging area), permitiéndote cambiar de contexto sin perder tu trabajo.

**Para qué sirve:**

Guardar cambios rápidamente: Si estás modificando archivos pero necesitas cambiar de rama o trabajar en otra tarea.

Evitar commits "sucios": No quieres hacer commit de cambios a medio terminar.

**Comandos:**

**GitBash**

|  |  |
| --- | --- |
| git stash | # Guarda los cambios en el stash |
| git stash list | # Muestra la lista de stashes guardados |
| git stash apply | # Recupera el último stash (sin borrarlo) |
| git stash pop | # Recupera y elimina el último stash |

**Ejemplo:**

En GitBash

echo "nuevo cambio" > archivo.txt

git stash # Guarda cambios

git checkout otra-rama

git stash pop # Recupera cambios en la nueva rama

**2. Rebase.**

El rebase reescribe el historial de commits al mover una rama para que comience desde un punto más reciente (generalmente el último commit de otra rama).

**Para qué sirve:**

Mantener un historial lineal: Evita los commits de fusión (merge commits) y crea un historial más limpio.

Integrar cambios de una rama base: Por ejemplo, actualizar tu rama feature con los últimos cambios de main.

**Comandos:**

**En GitBash**

git checkout feature

git rebase main # Reubica la rama feature sobre main

Ejemplo:

Antes del rebase:

main: A — B — C

feature: — D — E

Después del rebase:

main: A — B — C

feature: — D' — E'

No uses rebase en ramas públicas (como main), ya que altera el historial y puede causar conflictos para otros colaboradores.

**3. Ramificación (Branching)**

**Qué es:**

Las ramas son líneas de desarrollo independientes. Permiten trabajar en múltiples características o correcciones sin afectar la rama principal (ej: main).

**Para qué sirve:**

Desarrollo paralelo: Trabajar en una nueva funcionalidad (feature/login) mientras corriges un error (hotfix/bug).

Aislar cambios: Evitar que código inestable llegue a producción.

**Comandos:**

**En GitBash**

git branch nueva-rama # Crea una rama

git checkout nueva-rama # Cambia a la rama

git switch nueva-rama # Alternativa moderna a checkout

git merge nueva-rama # Fusiona la rama en la actual

**Ejemplo:**

**En GitBash**

git checkout -b feature/login # Crea y cambia a la rama

# ...haz commits...

git checkout main

git merge feature/login # Fusiona feature/login en main

Diferencias clave entre Merge y Rebase

git merge git rebase

Crea un commit de fusión. Reescribe el historial.

Ideal para ramas públicas. Ideal para ramas locales.

Mantiene el orden cronológico. Crea un historial lineal.

**COMANDOS QUE SE UTILIZAN:**

* Para listar ramas disponibles: **git branch**
* Eliminar una rama llamada "bugfix-123": **git branch -d bugfix-123**
* Ver rama en la que actualmente estas trabajando: **git branch --current**
* Diferencias entre commit y git stash: **Un git stash guarda cambios temporalmente, mientras que un commit los guarda permanentemente.**
* Se guardan los cambios actuales de trabajo en stash: **git stash**
* Para crear una rama nueva a partir de un stash existente: **git stash branch <branch-name>**
* Para aplicar un stash específico identificado por "stash@{2}" **git apply stash@{2}**
* Ver listado de todos los stashes guardados: **git stash list**
* Aplica cambios guardados en el stash y eliminalos de la lista de stash al mismo tiempo: **git stash pop**
* Aplica cambios guardados en el stash a la rama actual y sin eliminar el stash: **git stash apply**
* Elimina el stash aplicado recientemente: **git stash drop**

**Merge ramas, unir ramas, conflictos, unión automática.**

* Hacer merge de una rama en otra: **git merge abeh\_name**
* Hacer merge de rama feature en la rama main: **git checkout main y luego git merge feature**
* Fusionar (merge) con una rama en la rama principal: **Combina los cambios de la rama en la rama principal**
* Conflicto merge: **Hace modificaciones simultáneas en la misma parte del archivo desde diferentes ramas.**
* Abortar un merge en caso de conflictos: **git merge –abort**
* Unión (merge) en Git: **Ayuda a combinar los cambios de dos ramas en una sola**
* Decimos que es conflicto en git: **Cuando dos ramas contienen cambios incompatibles en el mismo archivo**
* Si hay conflictos durante el merge, paso inicial para resolver es: **Editar manualmente los archivos conflictivos**
* Después de resolver los conflictos y guardar los cambios, para terminar el merge utiliza: **git commit**
* merge fast-forward en Git: **Un merge que simplemente mueve el puntero de la rama a la punta de la rama fusionada**
* Caracteriza a un merge fast-forward: **a que la** **rama simplemente avanza al último commit de la rama fusionada.**
* Ventaja de implementar un merge fast-forward: **Mantiene un historial de commits lineal y limpio**
* Un merge de unión automática en Git se refiere a: **Un merge que Git puede realizar automáticamente sin conflictos**
* Para realizar una unión automática de ramas se utiliza: **git merge**
* Situación donde podemos implementar un merge fast-forward en Git: **Cuando la rama de destino no ha tenido nuevos commits desde que se creó la rama fuente.**
* Qué sucede con el historial de commits cuando se realiza un merge fast-forward: **El historial de commits permanece lineal.**
* Conflicto de fusión en Git: **Que hay divergencias en los archivos que deben resolverse**
* Resuelven conflictos: **git add y git commit**
* Después de resolver un conflicto de fusión, implementa para completar una fusión: **git commit**
* Qué tipo de merge sucede cuando la rama de destino no tiene nuevos commits desde que se creó la rama fuente:**Fast-forward merge**

**Git Stash y Git Rebase – Para realizar cambios de emergencia**

* ¿Qué es Git Stash?: **Es un área que permite almacenar temporalmente los cambios realizados en un proyecto sin enviarlos al repositorio mediante un commit.**
* Permite realizar un guardado rápido en nuestros archivos del proyecto: **git stash o git abe**
* Se visualiza de forma numerada los guardados rápidos almacenados: **git stash list**
* Como se aplica uno de los guardados rápidos almacenados mediante su nombre de lista: **git stash apply stash@{2}**
* Acción realiza el comando ***git stash pop***: **Permite hacer entrada al guardado, aplicar los cambios y eliminar los archivos de dicha área.**
* Con ***git stash show***: **Se muestra un resumen de los cambios realizados en el stash.**
* Borra los cambios guardados en stash sin aplicarlos: **git stash drop *nombre-del-stash***
* Limpia todo en el área de stash: **git stash clear**
* Qué acción se ejecuta cuando añadimos la opción ***–keep-index*** al comando ***stash sabe***: **Le indica a Git que no guarde nada que no se haya almacenado ya mediante el comando git add.**
* De qué forma se puede realizar un guardado rápido en el que se almacene cualquier archivo que no está bajo el control de la última versión del proyecto: **git stash –include-untracked**

**Git Rebase.**

* ¿Qué es git rebase?: **Es un proceso de reorganización enfocado en integrar modificaciones de una rama a otra, organizando la base de una rama de un commit a otra, manteniendo un historial del proyecto lineal.**
* Función de squash en el comando git rebase: **Permite combinar uno o más commits en uno solo.**
* ¿Cómo funciona la opción reword aplicada en git rebase?: **Después de su ejecución, la reorganización se detiene y permite modificar el mensaje de commit**.
* git rebase edit ayuda: **En permitir modificar el commit, es decir, agregar o eliminar información de éste**.
* Pick sirve para: **Indicar que se incluye la confirmación del commit.**
* Desventaja de hacer un rebase:**Hacer un rebase en una rama o repositorio público fusiona la rama principal en el repositorio local, creando una versión diferente o una discordancia en el proyecto.**
* Diferencia entre git merge y git rebase: **Git merge deja intactas las ramas existentes conservando todo el historial, mientras que git rebase reorganiza nuestro proyecto consiguiendo un historial limpio y lineal sin bifurcaciones.**
* Diferencia entre usar la opción fixup y la opción squash:**Squads permite combinar dos o más commits en uno solo con la posibilidad de escribir un nuevo mensaje de confirmación, fixup descarta este mensaje y sólo combina los commits en uno solo.**
* git rebase –i: **Accede a la versión interactiva del comando, en la cual podemos seleccionar, modificar, mover o eliminar los commits según las propias necesidades.**
* Diferencia entre realizar un rebase standard y un rebase interactivo: **El rebase standard toma automáticamente las confirmaciones de la rama de trabajo actual y las aplica al encabezado de la rama anterior, mientras que el rebase interactivo abre un editor en el cual se pueden adaptar los commits a nuestras necesidades.**