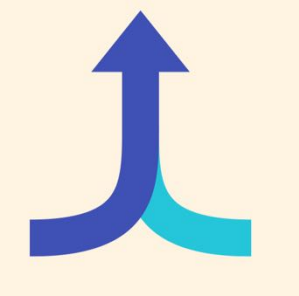
🔀 Fusión de ramas en Git

¡Llegó el momento de unir caminos! 🚧



Ahora que ya dominamos cómo **crear ramas** y **movernos entre ellas**, hablemos de algo esencial en Git: **la fusión de ramas** (*git merge*).

Piensa en las ramas como **carriles separados en una autopista** 🛣️.
Cada persona puede avanzar en su propio carril sin chocar con los demás. Pero en algún punto, necesitamos volver a juntarnos en el mismo camino para seguir el viaje juntos. 🚗🚙➡️

Eso es justamente lo que hace la fusión: **combinar el trabajo de una rama con otra**.

Por ejemplo:
👉 Tú creas una nueva función,
👉 Yo arreglo un error,
👉 Y al final queremos **reunir ambos aportes** en una sola versión del proyecto.

A veces una rama se descarta (como una calle cerrada 🚧), pero normalmente, cuando el trabajo está listo, lo **fusionamos**.
Y el comando mágico para lograrlo es:

```
git merge
```

🌳 El flujo de trabajo más común

La mayoría de los equipos tratan la rama principal —ya sea `main`, `master` o `trunk`— como **la base más estable del proyecto**, la "versión oficial". 📦

No hacemos experimentos allí.
Por eso, creamos **ramas de características** (*feature branches*) donde probamos ideas, corregimos errores o agregamos funciones nuevas.

Una vez que todo está probado y aprobado ✅, **fusionamos esos cambios en la rama principal**.

De esta forma, cada integrante del equipo puede trabajar libremente en su propia rama sin romper el código de los demás. 💪

🧩 Conceptos clave sobre la fusión

Hay dos reglas de oro que debes recordar 🌟:

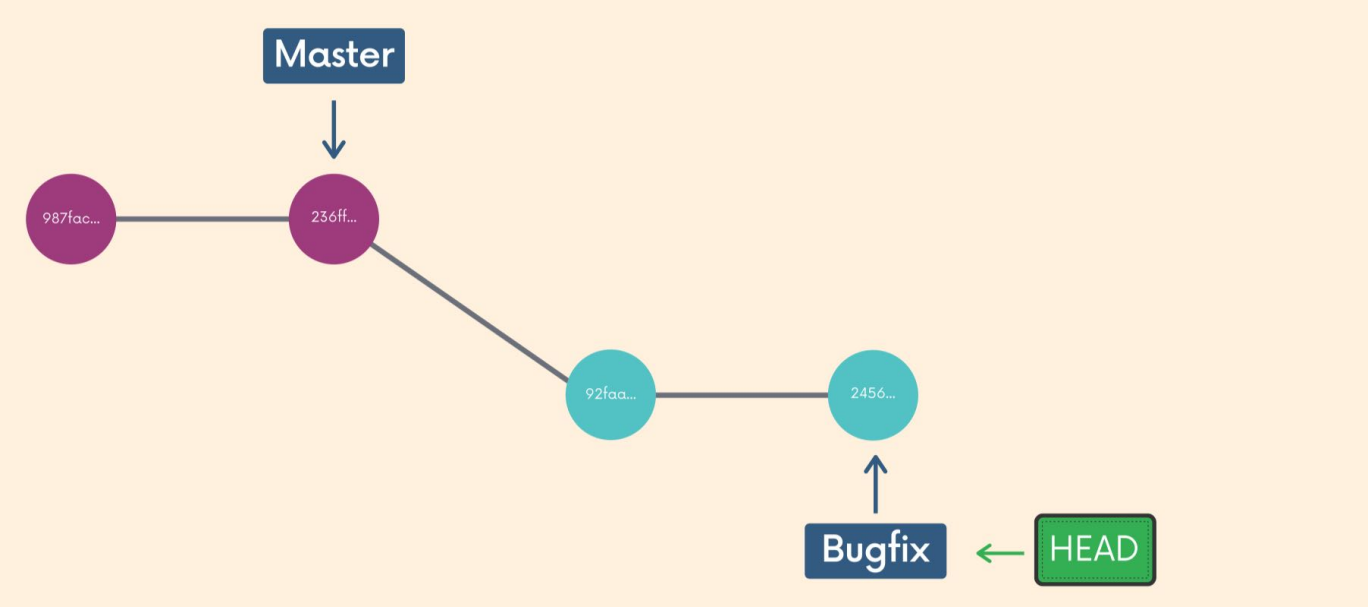
1. **Fusionamos ramas, no commits individuales.**
   Git no mezcla confirmaciones una por una, sino **todo el historial** de una rama con otra. 📜
2. **La fusión ocurre en la rama actual.**
   En otras palabras, Git fusiona *hacia* donde estás parado.
   Si estás en `master`, los cambios de la otra rama se integran *en* `master`. 🎯

💡 Ejemplo práctico

Imagina que tenemos estas dos ramas:

- 🧭 `master` → la rama principal
- 🔪 `bug-fix` → una rama donde corregimos un error

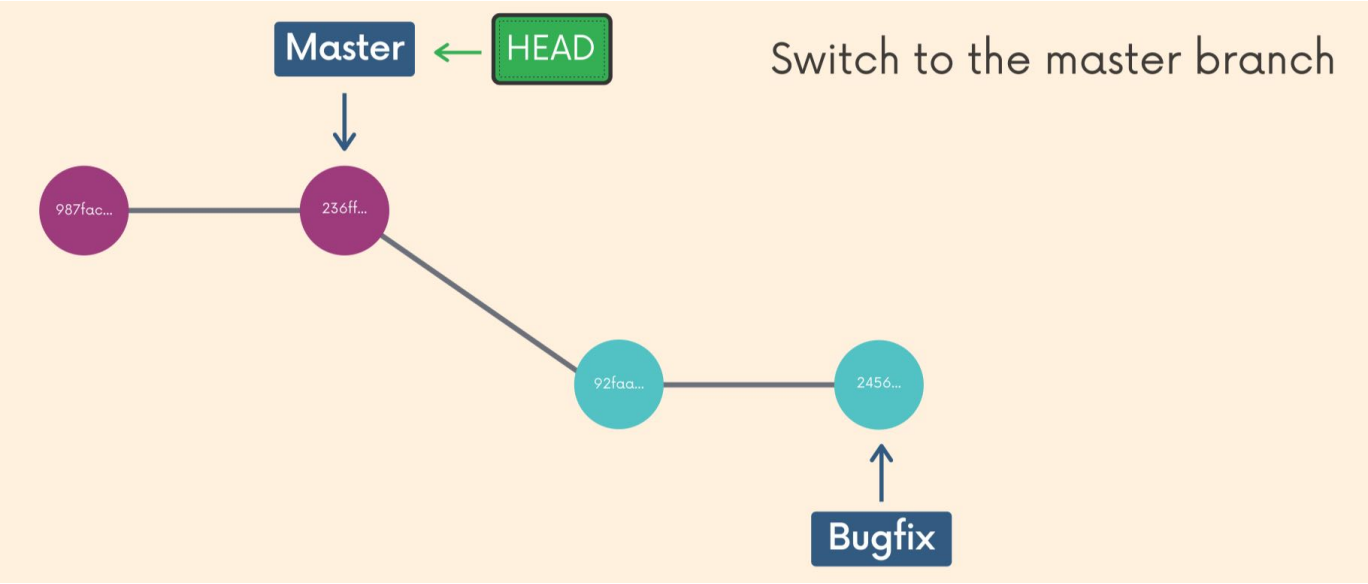
En `bug-fix` hicimos un par de commits 🪄 para resolver un problema, y ahora queremos **fusionar esos cambios** con `master`.



Los pasos serían:

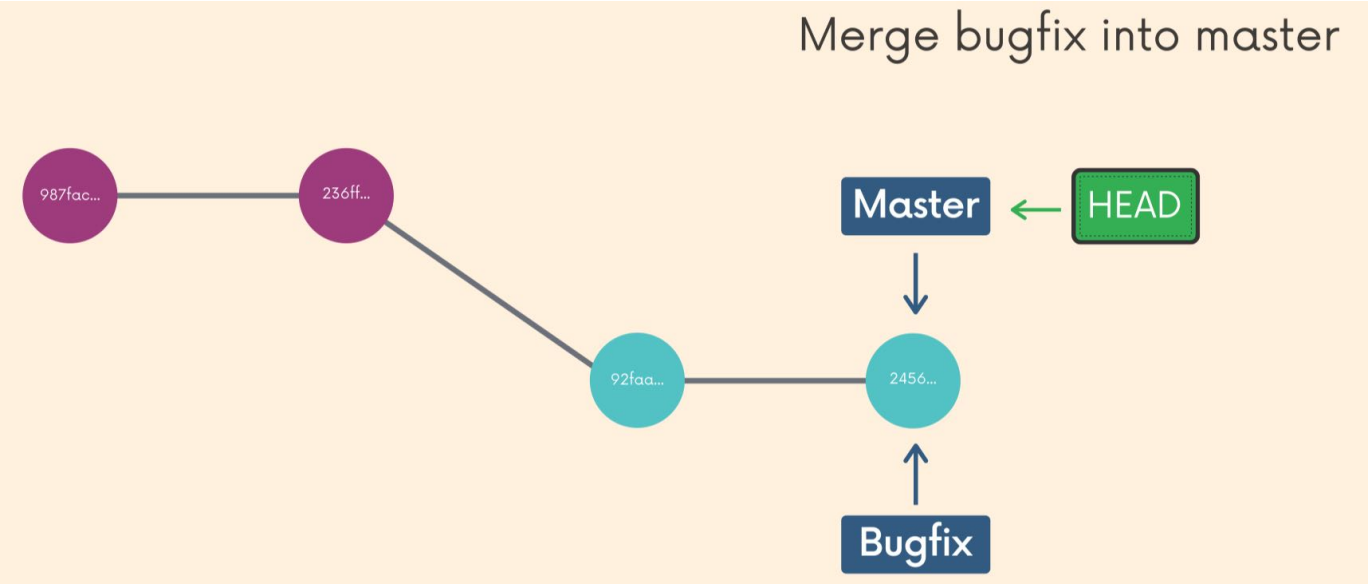
1️⃣ Cambiamos a la rama que recibirá los cambios (en este caso, **`master`**):

```
git switch master
```



2️⃣ Luego ejecutamos el comando de fusión:

```
git merge bug-fix
```



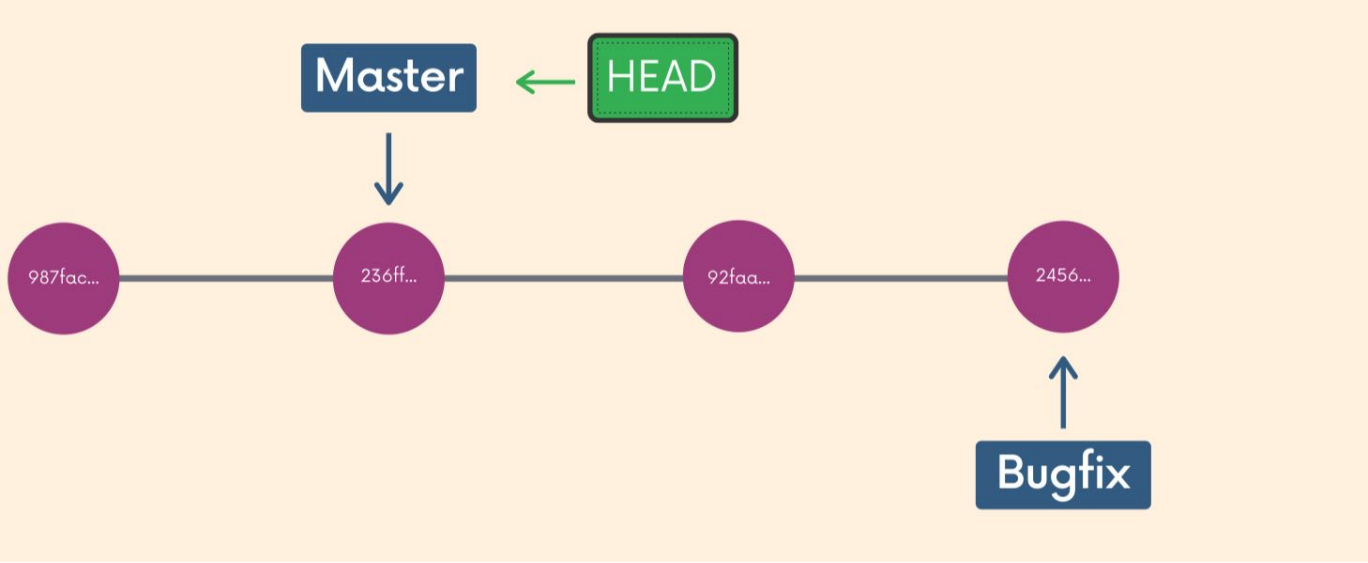
🎉 ¡Listo! Git toma los commits de `bug-fix` y los incorpora en `master`.

⚡ Fusión “fast-forward” (avance rápido)

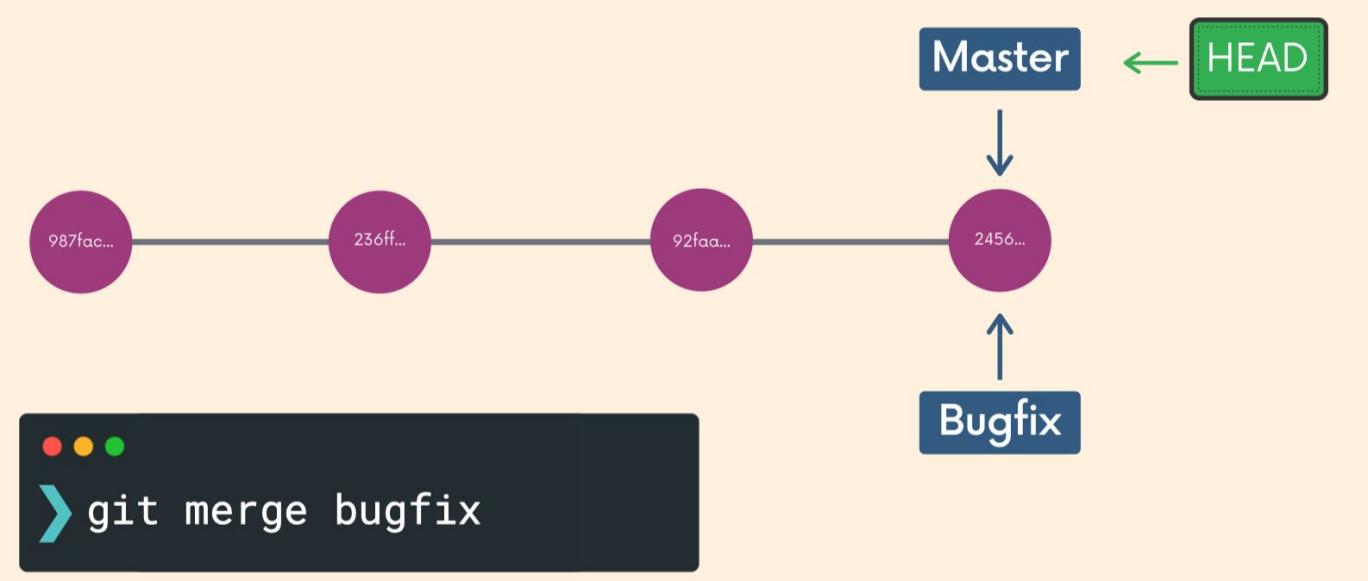
En este caso, la fusión es muy sencilla porque **no se hicieron nuevos commits en `master`** desde que creamos `bug-fix`.
Git simplemente **mueve el puntero** de `master` hacia el final de `bug-fix`, como si avanzara el marcador de posición. 🏃‍♂️💨

A este tipo de fusión se le llama **“fast-forward” (avance rápido)** porque Git no necesita crear un nuevo commit de fusión; solo **avanza en línea recta**.

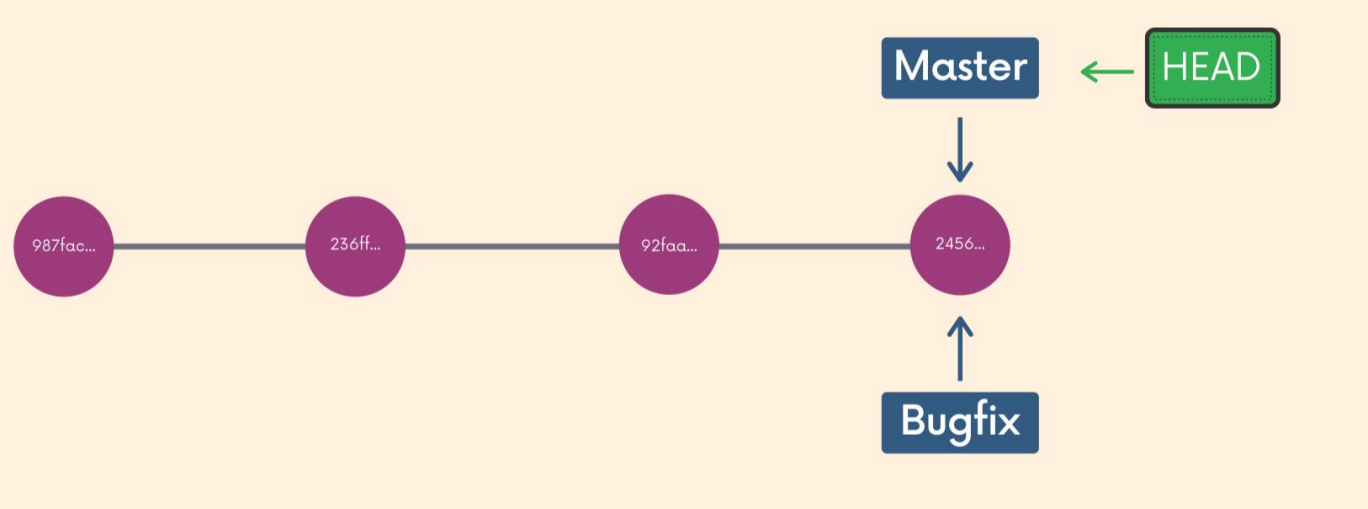
Visualmente sería así:



Después de la fusión:



En resumen: Git solo “adelanta” la rama principal para ponerse al día. ⏩



🧠 Más adelante veremos fusiones más complejas, donde ambas ramas tienen commits diferentes y podrían aparecer **conflictos** (cuando Git no sabe qué versión elegir 🤯).

Pero por ahora, ya dominas la fusión más fácil y común: la **fusión de avance rápido**. 🚀

⚡ Fusión “Fast-Forward” (avance rápido) en Git

Ahora vamos a practicar una **fusión de avance rápido**, que —solo para recapitular— es un **caso especial** de fusión en Git. 🔀

A diferencia de las fusiones más complejas, la “fast-forward” es **muy sencilla**: los comandos son los mismos, pero para Git es pan comido 🍞 porque lo único que debe hacer es **mover un puntero** hacia adelante hasta alcanzar el último commit de otra rama.

💡 Imagina que tu rama principal (`master`) se quedó un poco atrás en una carrera 🏃‍♂️🏁.
Cuando fusionas una rama más actualizada (por ejemplo, `bug-fix`), Git simplemente **hace que `master` corra hacia adelante** hasta alcanzar la misma línea de meta. 🏅

Manos a la obra 👉🏽:

🎧 Ejemplo: "MusicWave" — Un sitio web de música

Imaginemos que estamos desarrollando un sitio web llamado **MusicWave**, donde los usuarios pueden escuchar música, ver listas de reproducción y descubrir artistas nuevos.

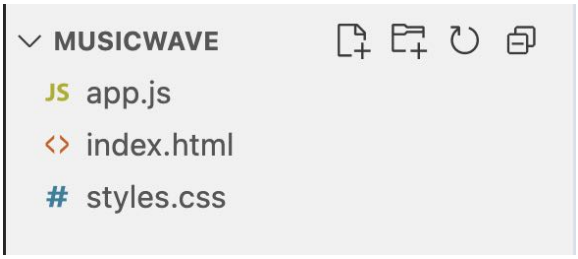
🎬 Situación inicial

La rama principal del proyecto (`master o main`) contiene el sitio base:

- Una página de inicio (`index.html`)
- Una hoja de estilos (`styles.css`)
- Un archivo JavaScript básico (`app.js`)

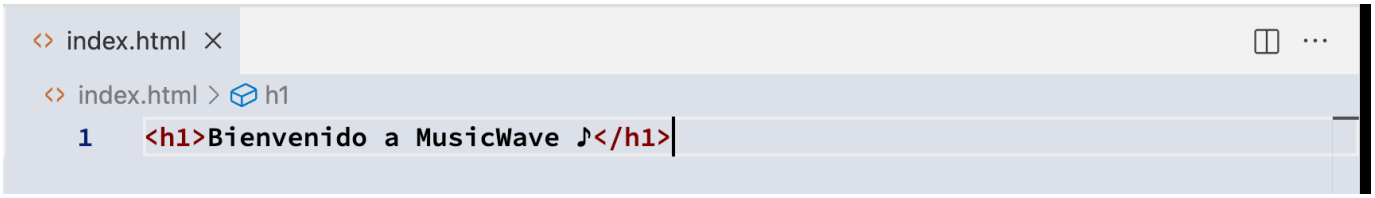
📂 Estructura inicial:

```
musicwave/
├── index.html
├── styles.css
└── app.js
```



📜 En `index.html` solo aparece el título:

```
<h1>Bienvenido a MusicWave 🎵</h1>
```



Iniciar el repositorio. Y hacer un commit

```
(base) josegarcia@Mac-mini-de-Jose musicwave % git init
Initialized empty Git repository in /Users/josegarcia/Documents/musicwave/.git/
(base) josegarcia@Mac-mini-de-Jose musicwave % git add .
(base) josegarcia@Mac-mini-de-Jose musicwave % git commit -m "Iniciando proyecto"
[main (root-commit) 6908bdd] Iniciando proyecto
 3 files changed, 1 insertion(+)
 create mode 100644 app.js
 create mode 100644 index.html
 create mode 100644 styles.css
(base) josegarcia@Mac-mini-de-Jose musicwave %
```

🌿 Paso 1: Crear una nueva rama

Un integrante del equipo, Ana, decide agregar una nueva sección para mostrar **listas de reproducción populares**.
Crea una nueva rama desde `master` (o `main`) llamada `playlists-populares`:

`git switch -c playlists-populares`

```
(base) josegarcia@Mac-mini-de-Jose musicwave % git switch -c playlists-populares
Switched to a new branch 'playlists-populares'
```

`En este momento ambas ramas están apuntando al mismo lugar.`

🎨 Paso 2: Trabajar en la nueva función

En la rama `playlists-populares`, Ana edita `index.html` y agrega lo siguiente:

```
<section id="playlists">
  <h2>Listas de reproducción populares</h2>
  <ul>
    <li>Top Hits 2025</li>
    <li>Clásicos del Rock</li>
    <li>Lo mejor del Pop Latino</li>
  </ul>
</section>
```

index.html M

index.html > ...

```html
<h1>Bienvenido a MusicWave ♪</h1>

<section id="playlists">
  <h2>Listas de reproducción populares</h2>
  <ul>
    <li>Top Hits 2025</li>
    <li>Clásicos del Rock</li>
    <li>Lo mejor del Pop Latino</li>
  </ul>
</section>
```

Luego hace un commit con su cambio:

```
git add index.html
git commit -m "Agregar sección de listas de reproducción populares"
```

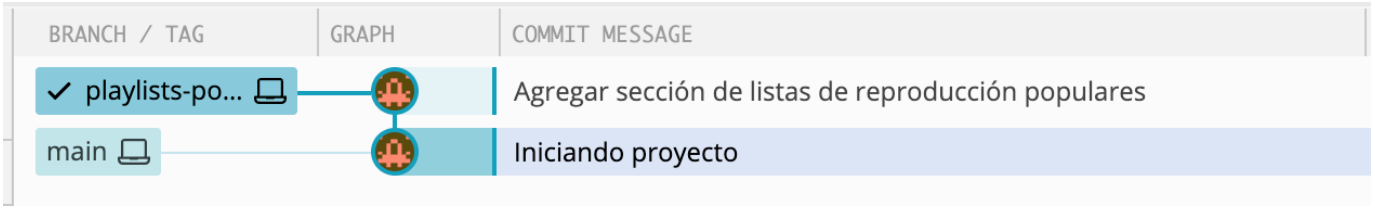
```
(base) josegarcia@Mac-mini-de-Jose musicwave % git add index.html
(base) josegarcia@Mac-mini-de-Jose musicwave % git commit -m "Agregar sección de listas de reproducción populares"
[playlists-populares 42f9803] Agregar sección de listas de reproducción populares
 1 file changed, 10 insertions(+), 1 deletion(-)
```

🌳 Paso 3: Verificar la situación de las ramas

Hasta ahora tenemos dos ramas:

- **`main`** ➡️ solo tiene el mensaje de bienvenida.
- **`playlists-populares`** ➡️ incluye la nueva sección de listas de reproducción.

No se ha hecho ningún cambio adicional en `main` desde que Ana creó su rama, por lo que Git podrá hacer una **fusión fast-forward**. ⚡



🔀 Paso 4: Fusionar con “fast-forward”

Ana cambia de nuevo a la rama `main`:

```
git switch main
```

Y fusiona su trabajo:

```
git merge playlists-populares
```

💡 Git responderá algo como:

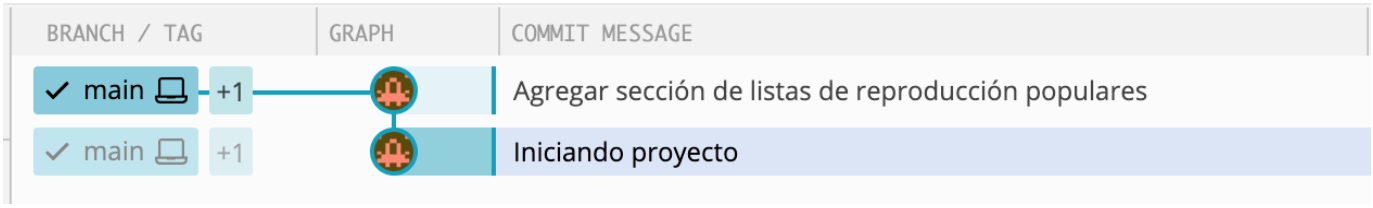
```
Updating a1b2c3d..d4e5f6g
Fast-forward
 index.html | 7 +++++++
 1 file changed, 7 insertions(+)
```

```
(base) josegarcia@Mac-mini-de-Jose musicwave % git switch main
Switched to branch 'main'
(base) josegarcia@Mac-mini-de-Jose musicwave % git merge playlists-populares
Updating 6908bdd..42f9803
Fast-forward
 index.html | 11 ++++++++++-
 1 file changed, 10 insertions(+), 1 deletion(-)
```

✅ Resultado

La rama `main` ahora incluye la nueva sección de listas de reproducción.
Git no creó un nuevo commit de fusión porque **solo adelantó el puntero** de `main` hasta el último commit de `playlists-populares`.

Después de la fusión:



Si pasamos el puntero del ratón donde dice +1 veremos que las 2 ramas están en el mismo lugar

---

🧩 Explicación con metáfora

Piensa que `main` es el **álbum principal** 🎶 y `playlists-populares` es una **versión extendida** con nuevas canciones.
Cuando haces una fusión fast-forward, simplemente **actualizas el álbum principal** para incluir todas las canciones nuevas sin tener que regrabar nada. 🎤

🧩 Importante: la rama fusionada no desaparece

Después de la fusión, **la rama `playlists-populares` sigue existiendo**. Git no la borra automáticamente.
Piensa en ella como una copia de seguridad o un laboratorio donde puedes seguir experimentando 🧪 sin afectar el contenido principal.

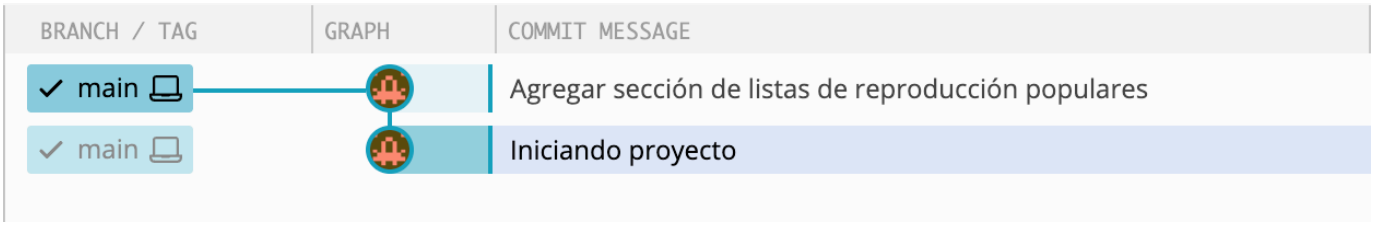
Aunque ahora ambas ramas (`main` y `playlists-populares`) apuntan al mismo commit, siguen siendo **dos contextos separados**.
Puedes seguir trabajando en `playlists-populares` sin que eso cambie automáticamente lo que hay en `main`.

🪄 Paso extra

Después de fusionar, Ana puede eliminar la rama porque ya se integró su trabajo:

```
git branch -d playlists-populares
```

```
(base) josegarcia@Mac-mini-de-Jose musicwave % git branch -d playlists-populares
Deleted branch playlists-populares (was 42f9803).
```



Esto mantiene el repositorio limpio y organizado. 🧹

🔀 Fusiones sin avance rápido en Git

Como mencioné antes, **no todas las fusiones (merge)** en Git son fusiones de **avance rápido (fast-forward)**. Veamos un ejemplo para entender mejor qué ocurre en esos casos 👇

Imagina que estás trabajando en un proyecto en equipo. Tienen la **rama principal** (`main` o `master`) y tú creas una **rama nueva** para corregir un error o probar una idea.
Después de unas horas de trabajo, quedas satisfecho con los cambios y decides **fusionar tu rama** con la rama principal.

💡 Pero aquí viene el detalle: mientras tú trabajabas en tu rama, **otra persona del equipo también hizo cambios** en la rama principal.
Eso significa que ahora la rama principal tiene *commits nuevos* que tú no tienes, y tu rama tiene cambios que la principal tampoco conoce.

En este punto, **ya no es posible hacer una fusión de avance rápido**, porque Git no puede simplemente "poner al día" una de las ramas.
Ambas contienen información diferente que debe combinarse.

🤔 ¿Qué hace Git en este caso?

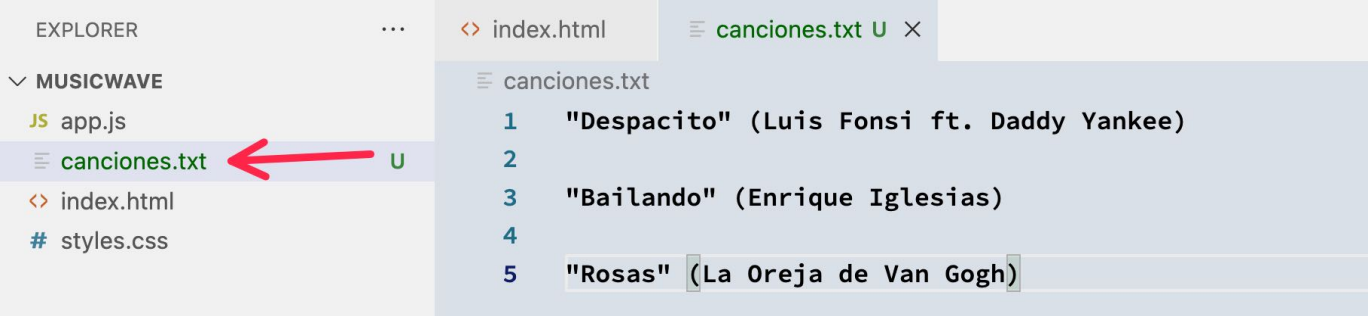
Git realiza lo que se llama un **commit de fusión (merge commit)**.
Este commit se crea automáticamente para registrar el punto en el que ambas ramas se unieron. Es importante notar que este nuevo commit **tiene dos padres**, uno por cada rama que se está fusionando.

Normalmente, Git puede hacer esta fusión de manera automática.
Sin embargo, si hay **conflictos** —por ejemplo, si tú y tu compañero editaron la misma línea de un archivo de forma diferente—, entonces Git no puede decidir por sí solo qué versión conservar.
(Veremos cómo resolver esos conflictos en otro ejemplo más adelante 😉).

Manos a la obra 👉🏽:

Vamos a usar el repositorio que hemos creado para visualizar mejor el proceso:

1. En la rama principal (`main`), añadimos un archivo llamado `canciones.txt` con algunas canciones.



Hacemos un **commit** con el mensaje:
`Agregando canciones.txt con canciones pop.`

```
(base) josegarcia@Mac-mini-de-Jose musicwave % git add .
(base) josegarcia@Mac-mini-de-Jose musicwave % git commit -m "Agregando canciones.txt con canciones pop."
[main b98d9f7] Agregando canciones.txt con canciones pop.
 1 file changed, 5 insertions(+)
 create mode 100644 canciones.txt
(base) josegarcia@Mac-mini-de-Jose musicwave %
```

2. Luego creamos una nueva rama llamada **`mas-exitos`**: y cambiamos a esa rama

```
git branch mas-exitos
git switch mas-exitos
```

```
(base) josegarcia@Mac-mini-de-Jose musicwave % git branch mas-exitos
(base) josegarcia@Mac-mini-de-Jose musicwave % git switch mas-exitos
Switched to branch 'mas-exitos'
```

En esta rama agregamos 2 canciones al archivo canciones.txt y hacemos un commit.

"Agregando 2 canciones"

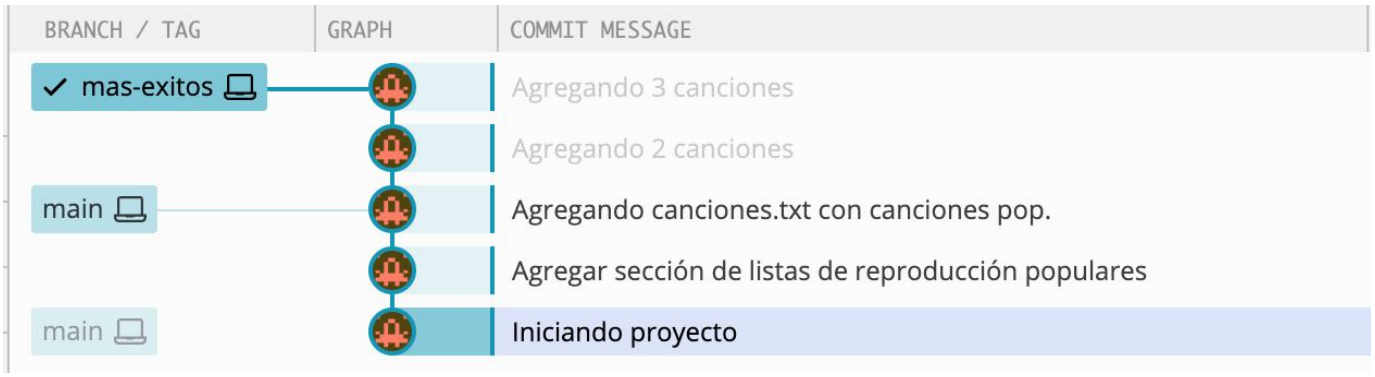
```
(base) josegarcia@Mac-mini-de-Jose musicwave % git add .
(base) josegarcia@Mac-mini-de-Jose musicwave % git commit -m "Agregando 2 canciones"
 [mas-exitos 01227bb] Agregando 2 canciones
  1 file changed, 5 insertions(+), 1 deletion(-)
```

Después del commit agregamos 3 canciones al archivo **`canciones.txt`** y nuevamente hacemos un commit.

"Agregando 3 canciones"

```
(base) josegarcia@Mac-mini-de-Jose musicwave % git add .
(base) josegarcia@Mac-mini-de-Jose musicwave % git commit -m "Agregando 3 canciones"
 [mas-exitos c8505f9] Agregando 3 canciones
  1 file changed, 7 insertions(+), 1 deletion(-)
```

Hasta aquí, si fusionáramos `mas-exitos` con `main`, sería una **fusión de avance rápido**, porque `main` no ha cambiado desde que creamos `mas-exitos`.

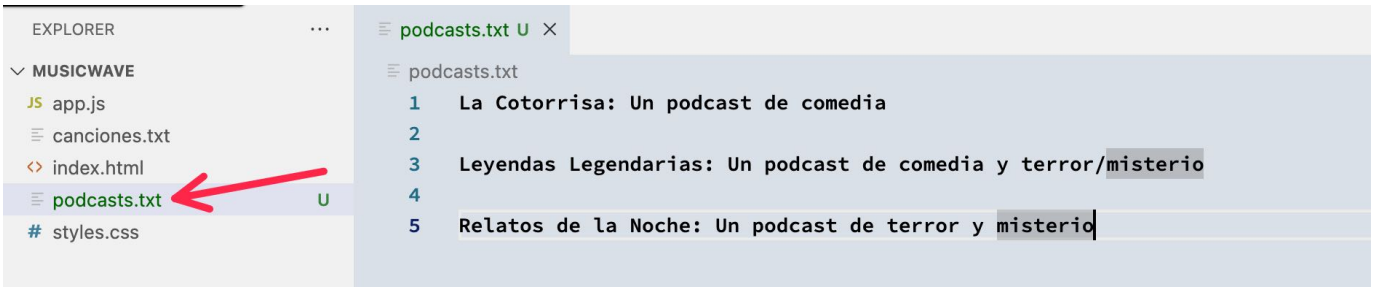


Pero agreguemos más acción...

Antes de fusionar, volvamos a la rama `main`

```
(base) josegarcia@Mac-mini-de-Jose musicwave % git switch main
Switched to branch 'main'
```

y agreguemos un nuevo archivo, por ejemplo `podcasts.txt`, con algunos nombres de podcasts.

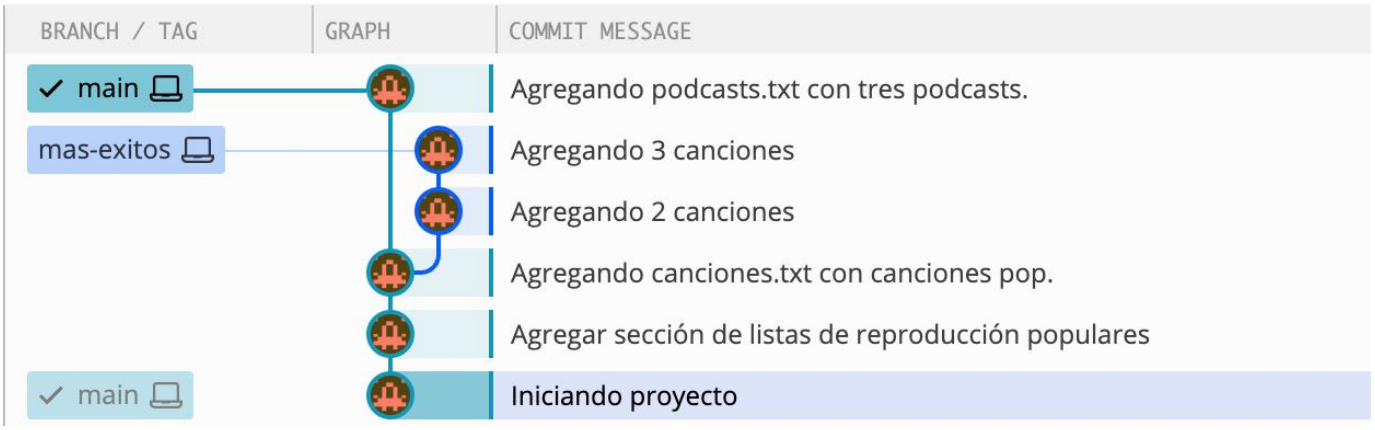


Hacemos un **commit** con el mensaje:
`Agregando podcasts.txt con tres podcasts.`

```
(base) josegarcia@Mac-mini-de-Jose musicwave % git add .
(base) josegarcia@Mac-mini-de-Jose musicwave % git commit -m "Agregando podcasts.txt con tres podcasts."
[main af4de3f] Agregando podcasts.txt con tres podcasts.
 1 file changed, 5 insertions(+)
 create mode 100644 podcasts.txt
```

Ahora tenemos:

- En `mas-exitos`: los commits con las nuevas canciones.
- En `main`: los commits con podcasts.



Si intentamos fusionar ahora `mas-exitos` en `main`, Git **no podrá hacer un fast-forward**, porque ambas ramas avanzaron por caminos diferentes.
Así que realizará un **merge commit**.

🧩 Realizando la fusión

Nos aseguramos de estar en la rama principal:

```
git switch master
```

```
(base) josegarcia@Mac-mini-de-Jose musicwave % git switch main
Already on 'main'
```

Y luego ejecutamos:

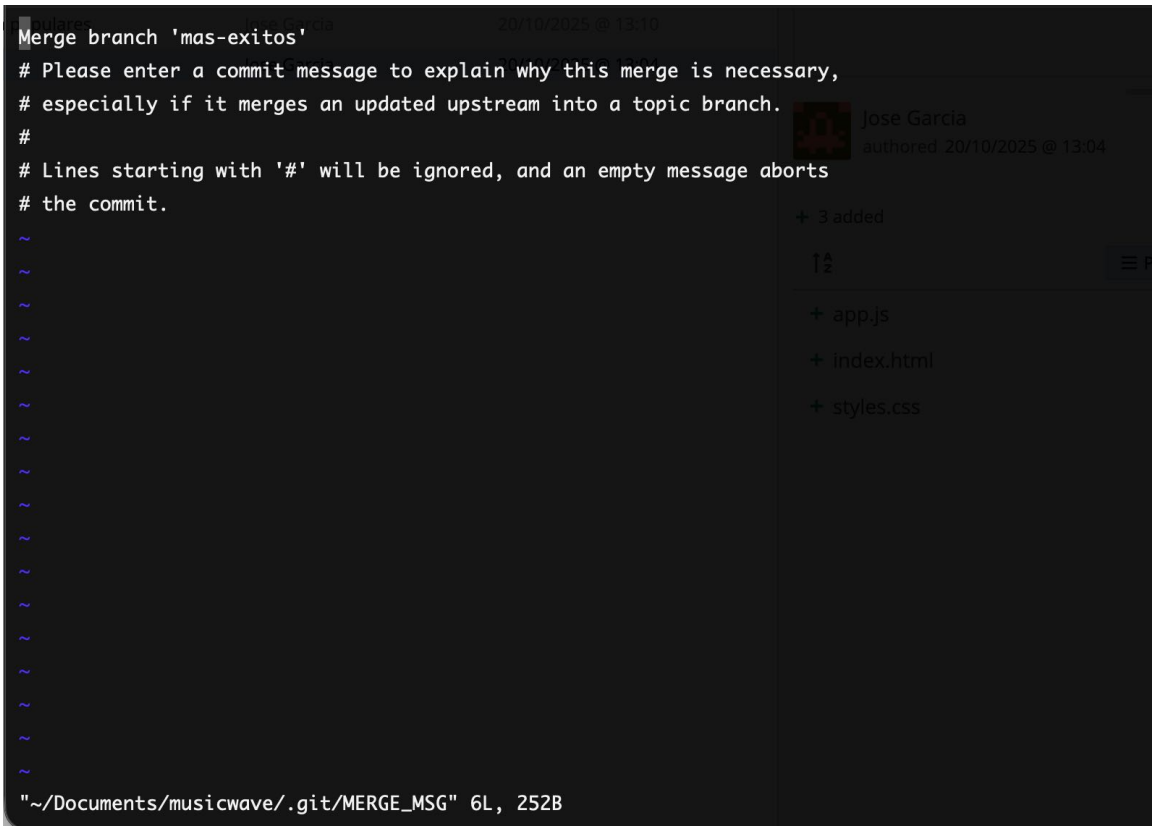
```
git merge mas-exitos
```

```
(base) josegarcia@Mac-mini-de-Jose musicwave % git merge mas-exitos
```

Git abrirá nuestro editor (por ejemplo, VS Code) y mostrará un mensaje como:

```
Merge branch mas-exitos
```

NOTA: En MAC se abre el programa VI



- Asegúrate de estar en el **modo de comandos** (presiona la tecla `Esc` si estás en modo de inserción).
- Escribe `:wq`

- Presiona `Enter`.

Podemos dejar el mensaje por defecto o editarlo si queremos.

Cuando cerramos el editor, Git crea el **commit de fusión** automáticamente y muestra un mensaje como:

```
Merge made by the 'recursive' strategy.
```

```
(base) josegarcia@Mac-mini-de-Jose musicwave % git merge mas-exitos
Merge made by the 'ort' strategy.
 canciones.txt  | 12 ++++++++++-
 1 file changed, 11 insertions(+), 1 deletion(-)
```

🔍 Revisando el resultado

Si revisamos el historial con:

```
git log --oneline --graph
```

Veremos un nuevo commit llamado:

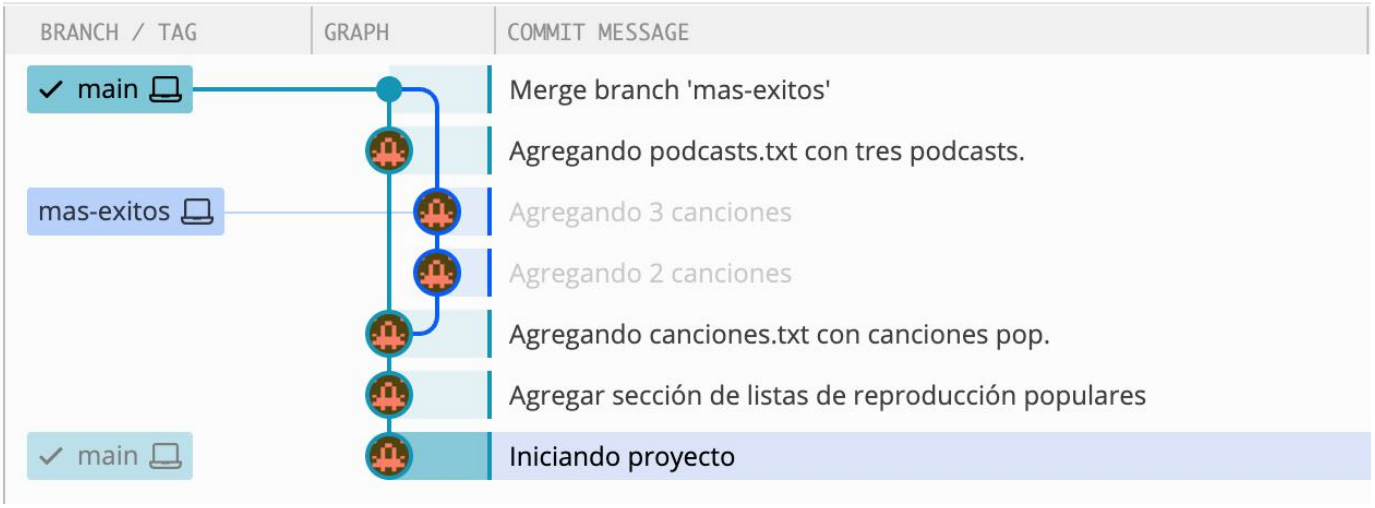
```
Merge branch 'mas-exitos'
```

```
(base) josegarcia@Mac-mini-de-Jose musicwave % git log --oneline --graph
*   9892b11 (HEAD -> main) Merge branch 'mas-exitos'
|\
| * c8505f9 (mas-exitos) Agregando 3 canciones
| * 01227bb Agregando 2 canciones
* | af4de3f Agregando podcasts.txt con tres podcasts.
|/
* b98d9f7 Agregando canciones.txt con canciones pop.
* 42f9803 Agregar sección de listas de reproducción populares
* 6908bdd Iniciando proyecto
```

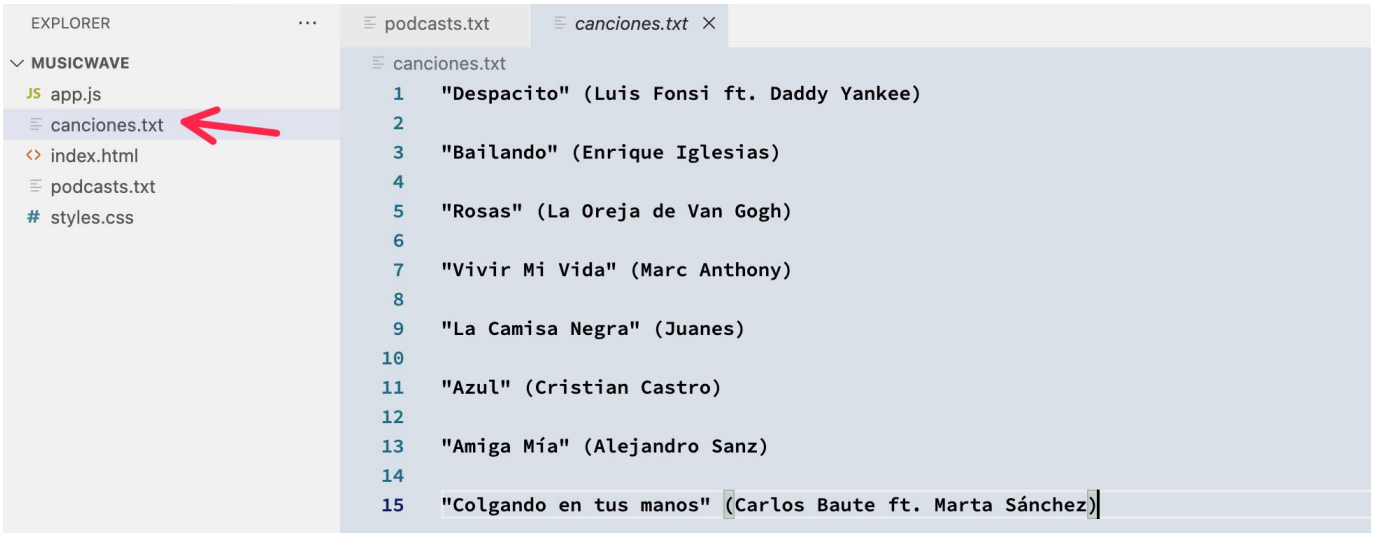
Este commit combina los cambios de ambas ramas:

- Las nuevas canciones de `mas-exitos`.
- El archivo `podcasts.txt` de `main`.

Y lo mejor de todo: **no hubo conflictos** 🎉
Git logró combinar todo sin problemas porque los cambios estaban en archivos distintos.



Estando en la rama `main` podemos ver que se tienen todas las canciones:



✅ Conclusión

Este es un ejemplo de una **fusión sin avance rápido**, en la que Git:

- Crea un **commit de fusión** para unir dos ramas con historias diferentes.
- Mantiene el historial completo de ambas.
- Puede hacerlo automáticamente, siempre que no haya conflictos.

En el siguiente paso aprenderemos qué ocurre **cuando sí hay conflictos** y cómo resolverlos manualmente 🧩💪

⚔️ Conflictos de fusión en Git

Muy bien, ahora hablemos sobre los **conflictos de fusión** 🧩.
Cuando fusionamos dos ramas, Git siempre intentará combinar los cambios **automáticamente**, ya sea mediante un **avance rápido (fast-forward)** o creando un **commit de fusión (merge commit)**, como vimos antes.

En la mayoría de los casos, Git lo logra sin problema ✅.
Pero hay situaciones en las que **no puede decidir por sí solo qué hacer**, y ahí es donde surgen los conflictos.

😬 ¿Cuándo ocurren los conflictos?

Algunos ejemplos típicos:

- En una rama, alguien **modifica un archivo**, y en otra rama alguien **borra ese mismo archivo**.
- En una rama, se edita la **línea 77** de un archivo, y en la otra rama **esa misma línea** se cambia de forma distinta.

En estos casos, Git se confunde 🤯.
No sabe qué versión conservar, así que **te pide ayuda** para resolverlo manualmente.

🧭 ¿Qué hace Git cuando detecta un conflicto?

Cuando ejecutas:

```
git merge <rama>
```

y hay conflictos, Git mostrará un mensaje parecido a:

```
Auto-merging archivo.txt
CONFLICT (content): Merge conflict in archivo.txt
Automatic merge failed; fix conflicts and then commit the result.
```

Esto significa que:

1. Git **intentó fusionar los cambios**,
2. encontró diferencias imposibles de resolver automáticamente,
3. y **necesita que tú intervengas** para decidir qué versión conservar.

✏️ Cómo se marcan los conflictos en los archivos

Cuando abres el archivo en conflicto, verás algo como esto:

```
<<<<<<< HEAD
Este es el contenido de mi rama principal
=======
Este es el contenido de la otra rama (por ejemplo, bug-fix)
>>>>>>> bug-fix
```

Estos símbolos son llamados **marcadores de conflicto**:

- `<<<<<<< HEAD` → indica el contenido actual en tu rama (la receptora).
- `=======` → separa las dos versiones.
- `>>>>>>>` → muestra el nombre de la rama desde la cual estás intentando fusionar.

Tu tarea es **editar el archivo** y decidir qué partes conservar.
Puedes quedarte con una versión, combinar ambas o reescribir el contenido para que tenga sentido según el contexto.

🪄 Pasos para resolver un conflicto

1. **Identificar los archivos en conflicto**
   Git los mostrará en el mensaje de error o puedes listarlos con:

   ```
   git status
   ```

2. **Abrir y editar los archivos**
   Decide qué contenido conservar y **elimina los marcadores** (`<<<<<<<`, `=======`, `>>>>>>>`).
3. **Guardar los cambios**
   Una vez resuelto el conflicto, guarda los archivos editados.
4. **Agregar los archivos al área de preparación (staging area)**

   ```
   git add <archivo>
   ```

5. **Hacer el commit de la resolución**

   ```
   git commit
   ```

   Git registrará que resolviste el conflicto con un nuevo **commit de fusión**.

💡 En resumen

Los conflictos de fusión:

- Son totalmente normales 😌.
- No significan que algo esté mal.
- Solo indican que **Git necesita tu decisión** para combinar los cambios correctamente.

Una vez que los resuelves y haces commit, la fusión se completa con éxito 🎉.

En el siguiente paso veremos una **demostración práctica** 👨‍💻 para entender cómo detectar, editar y resolver un conflicto directamente en Git.

Manos a la obra 👉:

💥 Demostración práctica: cómo generar, detectar y resolver un conflicto de fusión en Git

Muy bien 👋 En esta parte, te mostraré **cómo generar un conflicto de fusión** en Git, **resolverlo paso a paso** y entender lo que ocurre detrás de escena.

Seguiremos usando el mismo repositorio sencillo que ya tiene dos ramas: `main` y `mas-exitos`.
Pero como ya fusionamos los cambios anteriores, podemos eliminar esa rama extra para empezar desde cero.

🧹 Eliminando una rama innecesaria

Hay varias formas de eliminar una rama. Una muy común es:

`git branch -d mas-exitos`

```
(base) josegarcia@Mac-mini-de-Jose musicwave % git branch -d mas-exitos
Deleted branch mas-exitos (was c8505f9).
```

(La `-d` minúscula elimina la rama **solo si ya fue fusionada**. Si no lo estuviera, habría que usar `-D` en mayúscula para forzar la eliminación.)

Ahora solo nos queda la rama `main`, lista para seguir trabajando ✅.

```
(base) josegarcia@Mac-mini-de-Jose musicwave % git branch
* main
```

🌿 Creando nuevas ramas

Vamos a crear **dos nuevas ramas** para simular a dos compañeros de equipo:

- **Benito**, que agregará sus propias canciones.
- **Sonia**, que también hará su lista de reproducción.

Cada uno trabajará de forma independiente en su rama.

```
git branch sonia
git branch benito
```

```
(base) josegarcia@Mac-mini-de-Jose musicwave % git branch sonia
(base) josegarcia@Mac-mini-de-Jose musicwave % git branch benito
(base) josegarcia@Mac-mini-de-Jose musicwave % git branch
  benito
* main
  sonia
```

Ahora tenemos tres ramas: `main`, `sonia` y `benito`.
Ambas nuevas ramas se crearon a partir del mismo punto del repositorio.

🎶 Trabajando en la rama de Benito

Cambiamos a la rama de Benito y hacemos algunos cambios:

`git switch benito`

```
(base) josegarcia@Mac-mini-de-Jose musicwave % git switch benito
Switched to branch 'benito'
```

Benito no es muy fan de Luis Fonsi 😅, pero le gustan las otras canciones, así que las deja.
También agrega nuevas canciones:

Después de guardar hace un commit

```
(base) josegarcia@Mac-mini-de-Jose musicwave % git add .
(base) josegarcia@Mac-mini-de-Jose musicwave % git commit -m "Agregar canciones de benito"
[benito c72995d] Agregar canciones de benito
 1 file changed, 7 insertions(+), 3 deletions(-)
```

Perfecto, Benito tiene un commit nuevo en su rama.

🎤 Trabajando en la rama de Sonia

Ahora cambiamos a la rama de Sonia:

`git switch sonia`

```
(base) josegarcia@Mac-mini-de-Jose musicwave % git switch sonia
Switched to branch 'sonia'
```

Sonia es fan de algunos artistas y decide agregar *algunas canciones*:

podcasts.txt | canciones.txt M

canciones.txt

```
1  "Despacito" (Luis Fonsi ft. Daddy Yankee)
2
3  "Bailando" (Enrique Iglesias)
4
5  "Rosas" (La Oreja de Van Gogh)
6
7  "Vivir Mi Vida" (Marc Anthony)
8
9  "La Camisa Negra" (Juanes)
10
11 "Azul" (Cristian Castro)
12
13 "Amiga Mía" (Alejandro Sanz)
14
15 "Colgando en tus manos" (Carlos Baute ft. Marta Sánchez)
16
17 Hawái" (Maluma)
18
19 "Tusa" (Karol G, Nicki Minaj)
20
21 "Ella Baila Sola" (Eslabón Armado y Peso Pluma)
22
23 "Malamente" (Rosalía)
24
25 "Vida de Rico" (Camilo)
```

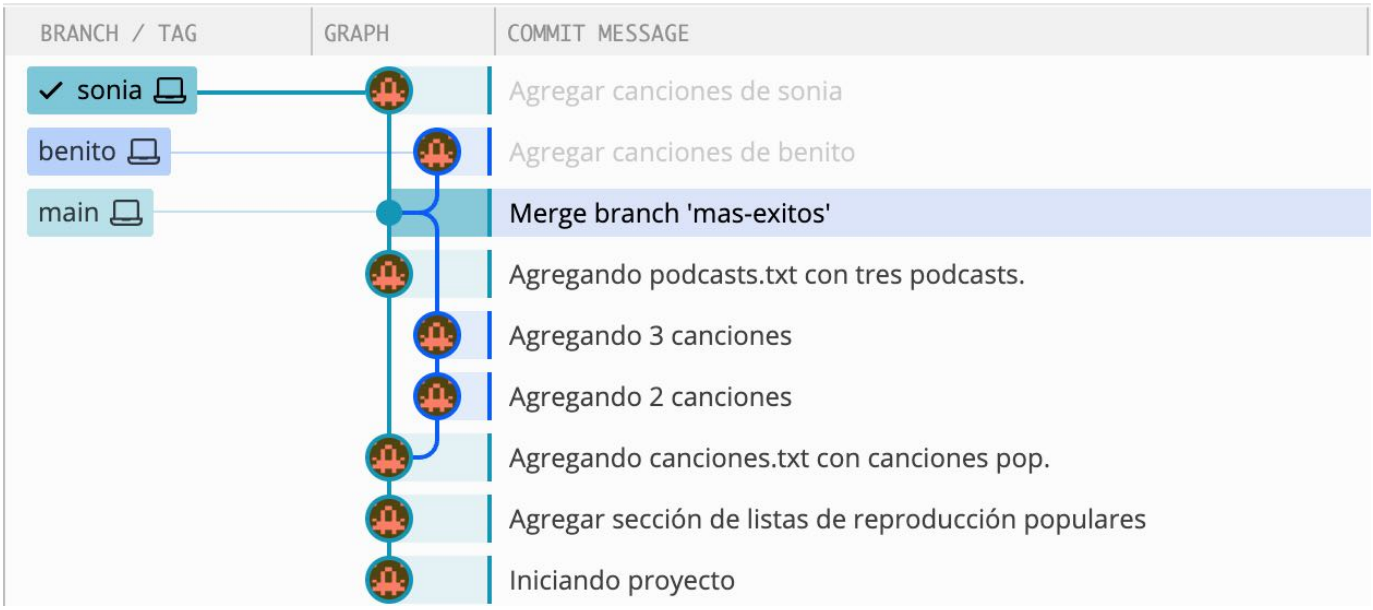
Hace un commit

```
(base) josegarcia@Mac-mini-de-Jose musicwave % git add .
(base) josegarcia@Mac-mini-de-Jose musicwave % git commit -m "Agregar canciones de sonia"
[sonia 8e4cf73] Agregar canciones de sonia
 1 file changed, 11 insertions(+), 1 deletion(-)
```

Hasta ahora tenemos esto:

- **Rama Benito:** un commit con canciones de Benito.
- **Rama Sonia:** un commit con canciones de Sonia.

Ambos modificaron **el mismo archivo (`canciones.txt`)**, y posiblemente **las mismas líneas** 🎧.
Esto significa que al fusionarlas, habrá **conflicto**.



⚡ Provocando el conflicto

Crearemos una nueva rama para hacer la fusión:

```
git switch benito
git switch -c combo
```

```
(base) josegarcia@Mac-mini-de-Jose musicwave % git switch benito
Switched to branch 'benito'
(base) josegarcia@Mac-mini-de-Jose musicwave % git switch -c combo
Switched to a new branch 'combo'
```

La rama `combo` parte de la rama `benito`.

Ahora intentemos fusionar los cambios de Sonia en ella:

```
git merge sonia
```

```
(base) josegarcia@Mac-mini-de-Jose musicwave % git merge sonia
Auto-merging canciones.txt
CONFLICT (content): Merge conflict in canciones.txt
Automatic merge failed; fix conflicts and then commit the result.
```

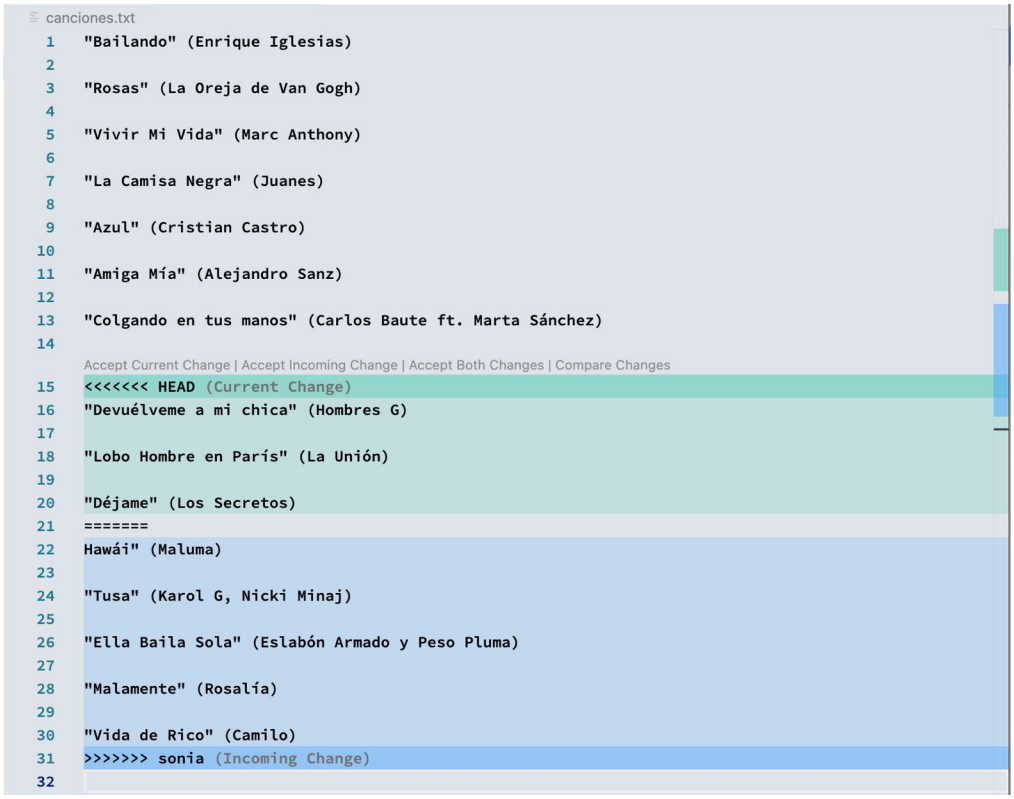
Git intentará hacer la fusión automática, pero… ❌
aparecerá un mensaje como este:

```
Auto-merging canciones.txt
CONFLICT (content): Merge conflict in canciones.txt
Automatic merge failed; fix conflicts and then commit the result.
```

¡Tenemos un conflicto de fusión! 😬

🧭 Resolviendo el conflicto

Abrimos el archivo en conflicto (`canciones.txt`) en nuestro editor (por ejemplo, VS Code).
Verás algo como esto:



Los marcadores de conflicto significan:

- `<<<<<<< HEAD` → los cambios de la rama actual (`combo`, basada en Benito).
- `=======` → separación entre versiones.
- `>>>>>>> sonia` → los cambios que vienen de la rama que intentamos fusionar.

Ahora debemos **editar el archivo** y decidir qué conservar.
Podemos mantener ambos conjuntos de canciones, eliminar duplicados o dejar solo una parte.

Por ejemplo, podríamos dejarlo así:

Y eliminamos todos los marcadores (`<<<<<<<`, `=======`, `>>>>>>>`).

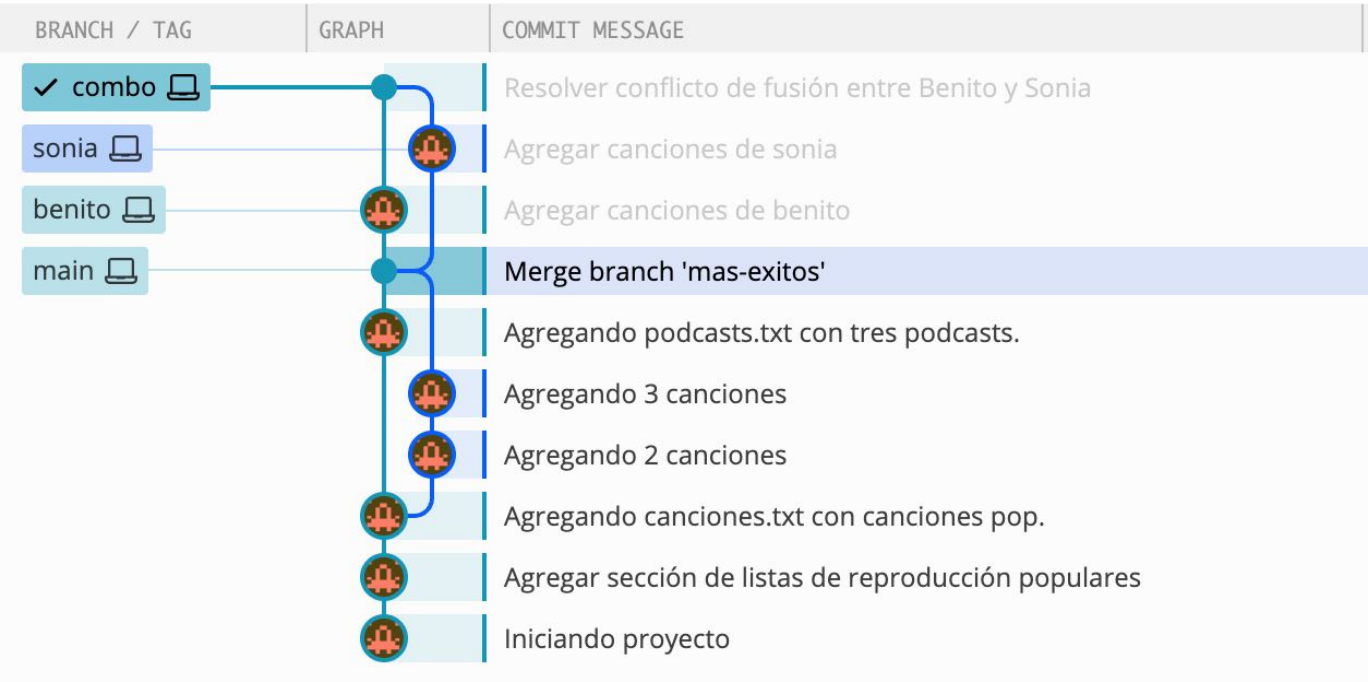
💾 Guardar y confirmar los cambios

Una vez editado, seguimos estos pasos:

```
git add .
git commit -m "Resolver conflicto de fusión entre Benito y Sonia"
```

```
(base) josegarcia@Mac-mini-de-Jose musicwave % git add .
(base) josegarcia@Mac-mini-de-Jose musicwave % git commit -m "Resolver conflicto de fusión entre Benito y Sonia"
[combo 15fcf7f] Resolver conflicto de fusión entre Benito y Sonia
(base) josegarcia@Mac-mini-de-Jose musicwave %
```

Git registra ahora un **merge commit** que representa la resolución del conflicto.



🎉 Resultado final

¡Listo! Hemos resuelto el conflicto de forma manual 💪.
La nueva rama `combo` contiene ahora las canciones de **Benito** y **Sonia**, sin duplicados y sin errores.

Git creó un nuevo **commit de fusión**, pero esta vez **tuvimos que intervenir nosotros** para decidir qué conservar.
Así aprendemos que los conflictos no son errores, sino **momentos donde el desarrollador decide el resultado final** 🧠💻.

💻 Ejercicio de Fusión en Git

Este ejercicio es un poquito diferente 🤓. En lugar de seguir paso a paso mis instrucciones, ¡quiero que crees tus **propios escenarios** que cumplan con los requisitos que te doy!

Comienza creando un **nuevo repositorio** 🆕. Escoge un tema sobre el cual trabajar (animales, películas, tecnología, etc..)
Dentro de él, crea uno o dos archivos para practicar.

Si necesitas un poco de inspiración... yo estaré trabajando con un archivo llamado `saludos.txt`, que contendrá saludos en diferentes idiomas 🌍.

🟢 Parte 1: Fusión "Fast Forward"

🎯 **Tu objetivo:** generar una fusión *fast forward* (avance rápido).
Demuestra que entiendes cómo funcionan este tipo de fusiones creando una tú mismo.

1. Crea una **nueva rama** 🌿.
2. Realiza algunos cambios en el repositorio.
3. Luego, fusiona esa rama con la rama `master` (o `main`).
4. Verifica si la fusión fue de tipo *fast forward* ✅.

🟡 Parte 2: Fusión con *Merge Commit* (sin conflictos)

🎯 **Tu objetivo:** realizar una fusión que cree un *merge commit*, pero **sin conflictos** 😌.

1. Crea una nueva rama 🌱.
2. Haz cambios en el repositorio de forma que, al fusionar con `master`, Git tenga que crear un *merge commit*.
3. Asegúrate de que no se generen conflictos 🤹.
4. Fusiona la rama y comprueba el resultado 👀.

🔴 Parte 3: ¡Conflictos! ⚔️

🎯 **Tu objetivo:** ¡provocar un conflicto de fusión intencionalmente! 😈

1. Crea una nueva rama 🌿.
2. Realiza cambios en el mismo archivo (y en las mismas líneas) que en la rama `master`.
3. Luego, intenta fusionarlas.
4. Si todo sale bien... ¡Git te mostrará un **conflicto de fusión**! 😱
5. Finalmente, **resuélvelo** 🧩 y completa la fusión con éxito 💪.