Introducción

Hola, ¡Te damos la bienvenida a nuestra segunda clase! 👋

Como ya vimos en la clase anterior, Git es una herramienta muy poderosa, y como todo gran poder trae consigo una gran responsabilidad, suele ser común que se generen algunos conflictos...

¡Pero no te preocupes! Esto es justamente lo que vamos a ver hoy; por qué se producen, cómo manejarlos y algunos comandos útiles para gestionar tus cambios en Git.

¡Comencemos! 🚀

Conflictos de fusión

Los **conflictos de fusión** (*"merge conflicts"*) suelen ocurrir cuando se trata de combinar archivos que están en conflicto; puede ser cuando se ha

cambiado la misma línea de un archivo, el mismo texto o el mismo elemento.

Esto también puede ocurrir cuando estamos trabajando con otra persona.

Git no puede resolver estos conflictos automáticamente y necesita tu ayuda para decidir qué versión mantener.

Entonces, **¿qué hace Git cuando esto sucede?** Git pausará el proceso de fusión y te pedirá que resuelvas los conflictos. Los archivos con conflictos tendrán marcas especiales alrededor del código que te ayudarán a identificar el problema. Tendrás que editar el archivo para resolver el conflicto y luego puedes continuar con la fusión.

Por ejemplo, si hemos realizado cambios que entran en conflicto con los cambios de otra persona y los hemos confirmado, Git nos informará sobre el problema en la terminal y nos indicará que la fusión automática ha fallado:

```
Egg@DESKTOP-GTMBQCU MINGW64 ~/Desktop/miprimerapagina (main)
$ git merge merge-test
Auto-merging testing.txt
CONFLICT (content): Merge conflict in testing.txt
Automatic merge failed; fix conflicts and then commit the result.
```

Incluso si hubiéramos pasado por alto ese mensaje, nos recordará acerca del conflicto la próxima vez que escribamos "git status".

Muy bien, ¡es momento de recrearlo! En la siguiente actividad, trabajaremos en un entorno local simulando un escenario de conflicto de fusión.

¡Manos a la obra!

❗ *Recuerda primero crear una nueva carpeta con el nombre de "Clase 2 - Resolución de conflictos" y hacer una copia de los documentos sobre los que venías trabajando.*

💡 Te recomendamos compartir tu pantalla con la mesa para que vean el conflicto que ha ocurrido.

1. Abrir la carpeta de *"Clase 2- Resolución de conflictos"* en Visual Studio Code y crear dentro un archivo de texto (como por ejemplo, *conflictos.txt*).

2. En el archivo de texto agregar contenido. Por ejemplo: *"Este es un contenido para hacer pruebas.".* No olvides de luego guardar los cambios.

3. En tu ordenador, ir a donde tengas ubicada la carpeta de *"Clase 2- Resolución de conflictos",* hacer clic derecho sobre la misma y seleccionar la opción de **"Git Bash Here"**.

4. Agregar todos los archivos de la carpeta al "área de preparación" con el comando: **git add .** y realizar un commit de los cambios utilizando el comando: **git commit -m "Mensaje descriptivo"**.

5. Crear una nueva rama utilizando el comando: **git branch nombre-rama**. Esta rama será utilizada para simular el conflicto de fusión.

6. Cambiar a la nueva rama utilizando el comando: **git switch nombre-rama**.

7. Volver a Visual Studio Code y realizar cambios en la misma línea del archivo de texto creado estando esta nueva rama. Puedes agregar, modificar o eliminar contenido en el archivo según tus preferencias, y luego guardar los cambios.

8. Realizar un commit de los cambios en la nueva rama utilizando los comandos: **git add nombre-archivo** y **git commit -m "Mensaje descriptivo".**

9. Cambiar nuevamente a la rama principal (por ejemplo, *git switch main*).

10. Nuevamente en Visual Studio Code, realizar cambios en la misma línea del mismo archivo de texto estando en la rama principal. Puedes hacer cambios diferentes a los realizados en la rama anterior.

11. Realizar un commit de los cambios en la rama principal utilizando los comandos: **git add .** y **git commit -m "Mensaje descriptivo".**

12. Intentar fusionar los cambios de la rama secundaria a la rama principal utilizando el comando: **git merge nombre-rama**.

En este punto, **se producirá un conflicto de fusión**.

13. Abrir el archivo con conflicto en Visual Studio Code y verás que contiene marcas especiales que indican las diferencias entre las versiones. Edita el archivo para resolver el conflicto haciendo clic en el botón de **“Resolve in Merge Editor”**.

14. Puedes elegir mantener tus cambios, los cambios de la rama secundaria o combinar ambas versiones según tus necesidades. Luego, hacer clic en **“Complete Merge”** para resolver el conflicto.

¡Felicitaciones! 👐 Has resuelto un conflicto de fusión en un entorno local.

Recuerda que en un entorno colaborativo real, trabajarías con repositorios remotos compartidos y el proceso de fusión se realizaría entre colaboradores.

Te compartimos el siguiente video por si quedaron dudas de cómo resolver la fusión de conflictos:

Git stash

Existen otras ocasiones en las que podrás encontrarte con un conflicto o una interrupción en el momento de escribir código. En este caso, vamos a explorar cómo manejar situaciones en las que necesitas cambiar rápidamente de una tarea a otra, sin tener que hacer un commit de los cambios que has realizado en el código.

Por ejemplo, podrías encontrarte en medio de la implementación de una nueva funcionalidad y recibir una solicitud urgente para solucionar un error en la rama principal. En lugar de hacer un commit de los cambios actuales y cambiar de rama, podrías guardar temporalmente esos cambios sin comprometer su integridad.

¡Pongámoslo en práctica!

1. Crear una rama en tu repositorio que se llame **“feature-1"** y agregarle un archivo HTML. Puedes usar el archivo de tu sitio web o cualquier otro.

Ahora, imagina que estás trabajando en una nueva característica en la rama "feature-1", pero de repente te piden que soluciones un error crítico en la rama principal (master).

2. Realizar algunos cambios en los archivos de la rama *"feature-1"* como modificar algunos estilos CSS, agregar un nuevo elemento HTML o cambiar el contenido de una sección. *No hagas commit todavía.*

3. Utilizar el comando **"git stash"** para guardar temporalmente tus cambios. Esto almacenará tus modificaciones en un área especial de Git y limpiará tu directorio de trabajo.

4. Cambiar a la rama principal utilizando el comando **"git switch master"** (o el nombre de tu rama principal).

5. Realizar los cambios necesarios para solucionar el “error crítico” y hacer commit de ellos como lo harías normalmente.

6. Una vez que hayas finalizado los cambios en la rama principal, volver a la rama "feature-1" utilizando el comando "git switch feature-1" (o el nombre de tu rama).

7. Recuperar tus cambios guardados utilizando el comando **"git stash pop"**. Esto restaurará tus modificaciones y las aplicará en la rama actual.

8. Verificar que los cambios estén presentes y continuar trabajando donde lo dejaste.

¡Excelente! Has utilizado con éxito **git stash** para guardar temporalmente tus cambios en una rama, cambiar a otra rama para solucionar un error y luego recuperar tus cambios guardados sin perder nada.

Entonces repasemos, **¿por qué es útil git stash? Porque te permite guardar temporalmente cambios para poder alternar rápidamente entre tareas sin hacer commit de dichos cambios que pueden estar incompletos.** Y, cuando quieras volver a tus cambios, puedes usar “git stash pop” para aplicar los cambios guardados.

💡 *También puedes usar **git stash list** para ver la lista de cambios guardados en el stash, **git stash -save “nombre”** para guardar un cambio en el stash con un mensaje que quieras, **git stash apply** para restaurar el mensaje guardado pero sin eliminarlo de la cola, como lo hace **git stash pop**, y **git stash clear** para eliminar todos los cambios guardados en la cola del stash.*

Git cherry-pick

A continuación, vamos a explorar cómo manejar situaciones en las que necesitas aplicar cambios específicos de un commit en una rama distinta sin tener que fusionar todo el historial.

Imagina nuevamente que estás trabajando en una rama específica, implementando una nueva funcionalidad, y de repente recibes una solicitud urgente para solucionar un error crítico en la rama principal. En lugar de fusionar toda la rama y arrastrar todos los cambios, vas a poder seleccionar y aplicar solo los cambios necesarios de un commit específico.

¡Pongámoslo en práctica!

1. Crear una nueva rama en tu repositorio con el nombre **"new-branch"** (o cualquier otro nombre que prefieras).

2. Realizar dos cambios con dos commits en uno de los archivos dentro la rama *"new-branch"*. Estos cambios pueden incluir agregar nuevas funcionalidades, modificar estilos o cambiar contenido.

3. Anotar el **ID del primer commit** (con los primeros 7 caracteres es suficiente) de los dos que hiciste. Puedes obtener el ID utilizando el comando **"git log"**.

```
Egg@eggpf3klttm MINGW64 ~/Dropbox/Projects/GIT_GITHUB/miprimerwebconchatgpt (nue
va-rama)
$ git log
commit 5ae2f11d6113794521f15cd82b4eea1ca4529de4 (HEAD -> nueva-rama)
Author: apalamara <apalamara@gmail.com>
Date:   Tue May 30 18:09:16 2023 -0300

    cambio color header a amarillo
```

4. Cambiar a la rama principal utilizando el comando **"git switch master"** (o el nombre de tu rama principal).

5. Utilizar el comando cherry-pick junto con el **ID del commit** anotado en el paso anterior para seleccionar y aplicar los cambios específicos de ese commit en la rama principal **"git cherry-pick <commit-id>"** .

6. Verificar que los cambios se hayan aplicado correctamente en la rama principal.

¡Excelente! Has utilizado exitosamente *git cherry-pick* para seleccionar y aplicar cambios específicos de un commit en la rama principal, evitando la necesidad de fusionar todo el historial de otra rama.

Repasemos, **¿por qué es útil git cherry-pick?** Porque es un comando poderoso que **te permite aplicar los cambios de un commit específico a tu rama actual**, siendo útil si necesitas incluir algunos cambios específicos sin fusionar una rama entera.

Git reset & Git revert

En esta parte, exploraremos diferentes ejercicios de cómo deshacer cambios hechos en el repositorio, lo que puede resultar útil para cuando cometemos errores en los commits y necesitamos revertir o deshacer los cambios realizados.

¡Manos a la obra!

Ejercicio 1: Deshacer el último commit

Supongamos que has realizado un commit en tu repositorio, pero te has dado cuenta de que cometiste un error.

Afortunadamente, vas a poder mover los cambios de ese commit a tu área de preparación (staging area) sin perderlos siguiendo estos pasos:

1. Realizar un commit en tu repositorio con los cambios que deseas deshacer.

2. Utilizar el comando **"git reset --soft HEAD~1"** para deshacer el último commit y mover los cambios a tu área de preparación.

3. Verificar que los cambios estén en el área de preparación utilizando el comando "**git status**".

4. Si es necesario, realiza las correcciones pertinentes en tu código.

5. Haz un nuevo commit de los cambios corregidos si deseas guardarlos.

¡Excelente! Has utilizado **git reset** para deshacer el último commit y mover los cambios a tu área de preparación, lo que te brinda la oportunidad de corregir errores antes de hacer un nuevo commit.

Ejercicio 2: Revertir un commit

En ocasiones, es posible que necesites deshacer los cambios realizados en un commit anterior manteniendo el historial de tu proyecto. Para esto, sigue estos pasos:

1. Realizar un cambio en cualquier archivo y haz un commit de ese cambio.

2. Utilizar el comando "git log" para obtener el ID del commit que deseas revertir.

3. Utilizar el comando "git revert <commit-id>" reemplazando "<commit-id>" por el ID del commit que deseas revertir.

4. Si nos aparece en la consola un mensaje como el siguiente:

```
Revert "add new file"

This reverts commit 82a1c0041023791f518faf1687051d29735060a1.

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# On branch segunda-rama
# Changes to be committed:
```

Entonces simplemente vamos a escribir “:q” y luego apretaremos enter

5. Si en lugar del mensaje anterior se produjo un conflicto durante el revert, sencillamente debemos resolverlo como ya hemos aprendido, es decir, ver con que cambios te quedas, luego hacer un **“git add .”** y un **“git commit”**.

6. Ahora si ejecutas un **“git log”** deberías ver el commit nuevo y que el historial de commits se mantiene pero pudiste realizar cambios que involucraban a un commit específico.

4. Verificar que los cambios se hayan revertido correctamente.

¡Genial! Has utilizado **git revert** para crear un nuevo commit que deshace los cambios realizados en un commit anterior, permitiéndote corregir errores sin alterar el historial de tu proyecto.

Pero ahora bien, **¿cuál es la diferencia entre git revert y git reset?** Y, **¿en qué casos sería útil cada uno de ellos?**

La diferencia principal entre **git revert** y **git reset** radica en cómo manejan los cambios y el historial del proyecto.

- **git revert** crea un nuevo commit que *deshace los cambios realizados en un commit anterior*, dejando intacto el historial de tu proyecto. Es útil cuando deseas deshacer los efectos de un commit específico sin eliminarlo por completo y sin afectar los commits posteriores.
- **git reset**, por otro lado, *puede eliminar commits y modificar el historial de tu proyecto*. Puedes utilizar git reset para deshacer cambios hasta un punto específico en el historial. Sin embargo, debes tener precaución al utilizarlo, ya que los commits eliminados no se pueden recuperar fácilmente.

Ejercicio 3: Deshacer todos los cambios

En algunos casos, es posible que desees deshacer todos los cambios realizados en tus archivos y volver a commit anterior, como si comenzaras desde cero. Para lograr esto, sigue estos pasos:

1. Con el comando **“git log”** elegir el commit desde el cual quieres comenzar de nuevo.

2. Utilizar el comando **"git reset --hard <commit-id>"** para deshacer todos los cambios y volver al estado anterior. Ten en cuenta que este comando eliminará permanentemente todos los commits hay entre el commit destino y el commit actual, por lo que debes tener precaución al utilizarlo.

💡 *También puedes usar* ***"git reset --hard HEAD~1"****: “****HEAD”*** *es una referencia que apunta al commit actual donde te encuentras, y “****~1”*** *representa el número de commits que quieres retroceder en el historial, por eso puedes cambiar el “1” por cualquier otro número.*

3. Verificar con **"git status"** que todos tus cambios se hayan deshecho y tus archivos estén en el estado anterior.

¡Excelente! Has utilizado **git reset --hard** para deshacer todos los cambios realizados en tus archivos y volver al estado anterior, como si comenzaras desde cero.

Git restore

Puede ocurrir que cometamos errores al realizar cambios en nuestros archivos y que necesitemos restaurarlos a como estaban desde el último commit. Para esos casos, vamos a poder revertir los cambios no deseados.

¡Pongámoslo en práctica!

1. Realizar algunos cambios en uno o varios archivos de tu repositorio, como modificar el contenido de un archivo o eliminar una sección importante. No hagas commit de estos cambios.

2. Utilizar el comando **"git status"** para ver los cambios realizados en los archivos. Verifica que los archivos modificados aparezcan en la sección *"Changes not staged for commit"*.

3. Ahora, restaurar los cambios no deseados utilizando **"git restore"**.

- Si deseas restaurar un archivo específico, utiliza el comando **"git restore <nombre del archivo>"**. Por ejemplo, "git restore index.html".
- Si quieres restaurar todos los archivos modificados en el directorio de trabajo, utiliza el comando **"git restore ."**.

4. Verificar nuevamente el estado de los archivos con **"git status"**. Deberías ver que los cambios no deseados han sido revertidos y los archivos vuelven a su estado anterior.

¡Muy bien! Has utilizado el comando **"git restore"** para revertir los cambios no deseados. Esta herramienta es útil cuando necesitas deshacer modificaciones en uno o varios archivos sin afectar otros cambios o commits en tu repositorio.

Repasemos, **¿por qué es útil git restore? Porque nos permite restaurar archivos a su estado previo sin afectar otros archivos o commits**. Podemos utilizarlo para deshacer cambios no deseados en archivos específicos o en todo el directorio de trabajo.

💡 Recuerda que "git restore" solo afecta los archivos en el área de trabajo y no modifica el historial de commits. Si deseas deshacer cambios en commits anteriores, puedes explorar otras herramientas como "git revert" o "git reset".

Desafío

Ahora que has probado todos los comandos útiles de Git te invitamos a realizar el siguiente desafío sobre **integración de características con conflictos**.

¡A resolver!

1. Crear un nuevo repositorio y dentro de él, un archivo llamado **“features.txt”**. Escribe una lista de 5 características que te gustaría tener en una aplicación.

2. Luego, crear *dos ramas nuevas* (“**feature-branch-1”** y **“feature-branch-2”**).

3. En cada rama, seleccionar y cambiar la descripción de dos características diferentes.

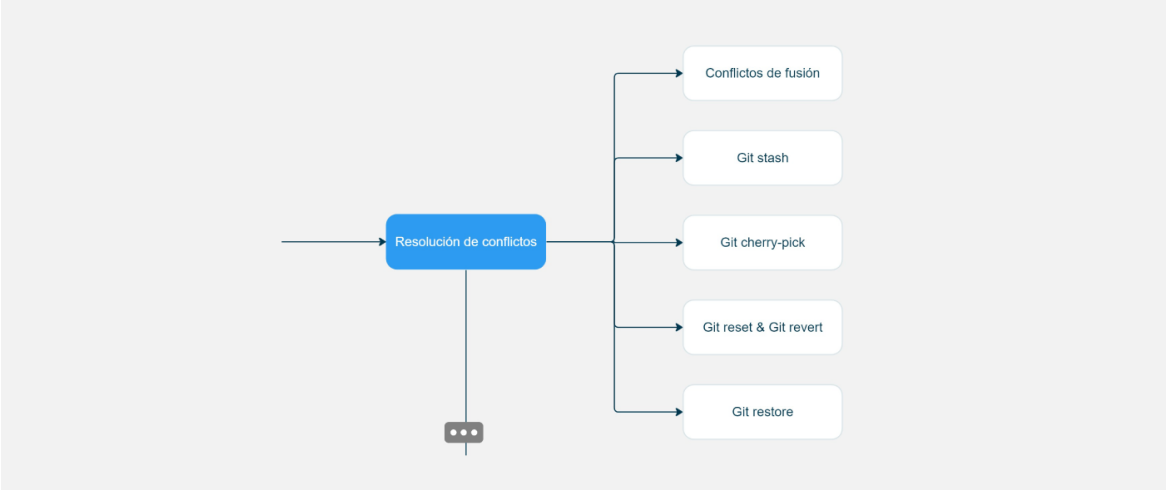
4. Finalmente, intentar fusionar ambas ramas con la rama principal (master o main). Tendrás que resolver los conflictos de merge antes de que la fusión pueda completarse.

Este ejercicio te dará una práctica valiosa con conceptos reales de Git y GitHub que son muy comunes en el desarrollo de software.

¡Diviértete practicando! 👐

Mapa de conceptos vistos

¡Fantástico! **Hemos cubierto mucho terreno hoy; desde la resolución de conflictos de merge y la experimentación con algunos de los comandos más avanzados de Git**. ¡Bien hecho!



Es importante recordar que dominar Git lleva tiempo y mucha práctica. La belleza del proceso consiste en que siempre hay nuevas cosas que aprender y formas más eficientes de hacer las cosas.

Para seguir practicando, te animamos a continuar trabajando en tus propios proyectos para que apliques lo que has aprendido hoy. Explora

más comandos de Git y, lo más importante, no tengas miedo de cometer errores. Muchos de los mejores aprendizajes en la programación vienen de resolver problemas y superar desafíos.

Gracias por tu participación en la clase de hoy. ¡Sigue practicando y no dudes en consultar en Discord si tienes más preguntas! 👋