Clase 4: Fusiones y Resolución de Conflictos para Científicos de Datos

Objetivos de la Clase

Al finalizar esta clase, serás capaz de:

- Comprender el concepto de **fusión de ramas (**`merge`**)** y su importancia en el flujo de trabajo colaborativo.
- Identificar y **resolver conflictos de fusión** que puedan surgir al integrar cambios.
- Explorar los diferentes **tipos de fusión** (`Fast-forward`, `No fast-forward`, `Squash`) y cuándo usarlos.
- Aprender a **eliminar ramas** que ya no son necesarias en tu repositorio.

1. Fusión de Ramas (`git merge`)

En proyectos de ciencia de datos, es común que trabajes en una rama separada para un experimento, una nueva funcionalidad o una corrección. Una vez que tu trabajo en esa rama es estable y ha sido validado, necesitas integrar esos cambios de vuelta a la rama principal (ej. `main`) o a otra rama de integración. Este proceso se llama **fusión** o `merge`.

1.1. ¿Qué es `git merge`?

`git merge` es el comando que te permite incorporar los cambios de una rama en otra. Cuando ejecutas `git merge <nombre_rama_origen>` desde una rama de destino, Git intentará combinar el historial de la rama de origen con el de la rama de destino.

Concepto Clave:

- La rama desde la que ejecutas `git merge` es la **rama de destino** (la que recibirá los cambios).
- La rama que pasas como argumento es la **rama de origen** (la que aporta los cambios).
- Git no te permitirá realizar una fusión si tienes cambios sin guardar (sin `commit` o `stash`) en tu Directorio de Trabajo de la rama de destino, ya que esto podría causar la pérdida de trabajo.

Sintaxis Básica:

```
Unset

# Primero, asegúrate de estar en la rama de destino (ej. 'main')
git switch main

# Luego, fusiona la rama de origen (ej. 'feat-nuevo-modelo') en la rama actual ('main')
git merge feat-nuevo-modelo
```

Image of Git Merge Operation

2. Tipos de Fusión (Merge)

Git tiene varias estrategias para realizar fusiones, que se aplican automáticamente o pueden ser controladas con parámetros, dependiendo de la relación entre los historiales de las ramas.

2.1. Fast-forward Merge (Avance Rápido)

Este tipo de fusión ocurre cuando la rama de destino no ha tenido ningún commit nuevo desde que la rama de origen se bifurcó de ella. Es decir, el historial de la rama de origen está "directamente por delante" de la rama de destino.

- **¿Cómo funciona?** Git simplemente mueve el puntero de la rama de destino al último commit de la rama de origen. No se crea un nuevo commit de fusión. El historial resultante es una línea recta, más limpia.
- **Cuándo ocurre:**
    - La rama de origen comparte el historial de commits de la rama destino.
    - La rama de origen tiene commits por delante que la de destino no tiene.
    - La rama de destino **no debe tener ningún commit** que no esté en la rama de origen.

Image of Git Fast-forward Merge

Ejemplo:

```
Unset
# Suponiendo que 'main' no ha tenido commits desde que 'feat-modelo' se bifurcó
```

```
git switch main
git merge feat-modelo
# Salida: Updating <hash_original>..<hash_final>
#          Fast-forward
#          <lista de archivos modificados>
```

Puedes forzar a Git a solo hacer `fast-forward` si es posible, o fallar de lo contrario, con `git merge --ff-only`.

2.2. `No Fast-forward` Merge (Merge Commit)

Este es el tipo de fusión más común cuando las ramas han **divergido**. Es decir, tanto la rama de origen como la rama de destino han tenido nuevos commits desde el punto en que se bifurcaron.

- **¿Cómo funciona?** Git crea un **nuevo commit de fusión** (un "merge commit") que tiene dos commits padre (los últimos commits de ambas ramas). Este commit combina el historial de ambas ramas y explícitamente registra la fusión.
- **Cuándo ocurre:** Cuando la rama de destino tiene commits que la rama de origen no tiene, y viceversa (historias divergentes). Git no puede simplemente "avanzar" el puntero.

[Image of Git No Fast-forward Merge]

Ejemplo:

```
Unset

# Suponiendo que 'main' y 'feat-modelo' han tenido commits
independientes
git switch main
git merge feat-modelo
# Salida: Merge made by the 'recursive' strategy.
#          <lista de archivos modificados>
#          <Git puede abrir un editor para el mensaje del merge
commit
```

Puedes forzar este tipo de fusión, incluso si un `fast-forward` fuera posible, con `git merge --no-ff`. Esto es útil para mantener explícito el registro de cada fusión en el historial.

2.3. `Squash` Merge

El `squash` merge es una estrategia de fusión que combina todos los commits de la rama de origen en un **solo commit nuevo** en la rama de destino.

- **¿Cómo funciona?** Git aplica todos los cambios de la rama de origen al Directorio de Trabajo y al Área de Staging de la rama de destino, pero sin preservar los commits individuales de la rama de origen. El resultado es un único commit "limpio" que contiene todos los cambios de la rama origen.
- **Cuándo usarlo:** Cuando quieres mantener un historial de la rama de destino muy lineal y limpio, y no te interesa conservar el historial detallado de commits de la rama de origen (ej., si la rama de origen tuvo muchos "commits de trabajo en progreso").

Image of Git Squash Merge

Sintaxis:

```
Unset


git switch main
git merge --squash feat-experimento-limpieza
# Salida: (Git te dejará los cambios en staging, listos para un
nuevo commit)
git commit -m "feat: Integrar resultados de experimento de limpieza
de datos"
```

Ventajas: Historial más limpio, ideal para funcionalidades autocontenidas.

Desventajas: Se pierde la trazabilidad de los commits individuales de la rama fusionada.

3. Resolviendo Conflictos de Fusión

Los **conflictos de fusión** son una parte inevitable del trabajo colaborativo con Git. Ocurren cuando Git no puede combinar automáticamente los cambios de dos ramas porque la misma parte de un archivo ha sido modificada de manera diferente en ambas ramas.

Image of Git Merge Conflict

3.1. ¿Cómo surgen los conflictos?

Un conflicto surge cuando:

- Dos ramas modifican las **mismas líneas de código** en el mismo archivo.
- Una rama elimina un archivo que otra rama ha modificado.
- Dos ramas crean archivos con el **mismo nombre** en la misma ubicación.

Cuando un `git merge` resulta en un conflicto, Git pausará el proceso y te notificará.

Ejemplo de Notificación de Conflicto:

```
Unset

git switch main
git merge feature/ajuste-parametros
# Salida:
# Auto-merging data_preprocessing.py
# CONFLICT (content): Merge conflict in data_preprocessing.py
# Automatic merge failed; fix conflicts and then commit the result.
```

En este punto, `git status` te indicará que tienes "Unmerged paths" (rutas sin fusionar).

3.2. Proceso de Resolución de Conflictos

1. **Identificar los archivos en conflicto:**

```
Unset
git status

# Te mostrará algo como:
# Unmerged paths:
#   (use "git add <file>..." to mark resolution)
#   both modified:    data_preprocessing.py
```

2. **Inspeccionar el archivo en conflicto:** Abre el archivo (`data_preprocessing.py` en este caso) con tu editor de texto. Verás unas marcas especiales que Git añade para indicar el conflicto:

```
Unset
# data_preprocessing.py
```

```
<<<<<<< HEAD
# Código de la rama actual (main)
learning_rate = 0.01
num_iterations = 100
=======
# Código de la rama que estás fusionando (feature/ajuste-parametros)
learning_rate = 0.005
num_iterations = 500
>>>>>>> feature/ajuste-parametros
```

    - `<<<<<<< HEAD`: Marca el inicio de los cambios de tu rama actual (donde estás `HEAD`).
    - `=======`: Separa tus cambios de los cambios de la otra rama.
    - `>>>>>>> <nombre_rama_origen>`: Marca el final de los cambios de la otra rama.
3. **Resolver el conflicto:** Edita el archivo para eliminar las marcas (`<<<<<<<`, `=======`, `>>>>>>>`) y deja el código como quieres que quede. Debes decidir qué versión de cada sección conflictiva quieres conservar, o si quieres combinarlas. **Ejemplo de resolución (combinando):**

```
Unset
# data_preprocessing.py (después de resolver)

# Código final deseado
learning_rate = 0.005 # Decidimos usar el learning rate de la otra rama
num_iterations = 100 # Pero mantenemos las iteraciones de nuestra rama
```

4. **Marcar el archivo como resuelto:** Una vez que hayas editado el archivo y eliminado todas las marcas de conflicto, debes decirle a Git que el conflicto ha sido resuelto y añadir el archivo al área de staging.

```
Unset
git add data_preprocessing.py
```

5. Puedes ejecutar `git status` de nuevo para confirmar que el archivo ya no aparece como "unmerged".
6. **Completar la fusión:** Finalmente, crea un nuevo commit de fusión. Git generará un mensaje de commit predeterminado que puedes aceptar o modificar.

```
Unset
git commit
# Esto abrirá tu editor para el mensaje del commit de fusión.
# Puedes aceptar el mensaje por defecto o añadir más detalles.
```

3.3. Abortar una Fusión (`git merge --abort`)

Si te has metido en un lío de conflictos y quieres empezar de nuevo, puedes abortar el proceso de fusión.

```
Unset

git merge --abort
```

Esto revertirá tu repositorio a su estado exacto antes de que intentaras la fusión.

4. Eliminando Ramas (`git branch -d` / `-D`)

Una vez que has fusionado una rama de funcionalidad o experimento en la rama principal, y ya no necesitas la rama de origen, es una buena práctica eliminarla para mantener tu lista de ramas limpia y organizada.

4.1. Eliminar Ramas Locales

- `git branch --delete <nombre_rama>` (o `-d`): Elimina la rama local **solo si sus cambios ya han sido fusionados** en la rama actual o en alguna otra rama. Git te protege para evitar la pérdida accidental de trabajo no fusionado.

```
Unset
# Asegúrate de no estar en la rama que quieres eliminar

git switch main

# Elimina la rama 'feat-visualizacion-datos' (si ya fue fusionada)
git branch -d feat-visualizacion-datos
```

```
# Salida: Deleted branch feat-visualizacion-datos (was <hash>).
```

- `git branch --delete --force <nombre_rama>` (o `-D`): Elimina la rama local **sin importar si sus cambios han sido fusionados o no**.
    - **¡ADVERTENCIA!** Utiliza esto con extrema precaución. Si eliminas una rama no fusionada con `-D`, perderás el historial de commits de esa rama si no está respaldada en otro lugar (ej. un repositorio remoto).

```
Unset
# Elimina la rama 'exp-fallido' (aunque no se haya fusionado)
git branch -D exp-fallido
# Salida: Deleted branch exp-fallido (was <hash>).
```

4.2. Eliminar Ramas Remotas (Introducción)

Aunque el comando `git branch` solo afecta a las ramas locales, para eliminar una rama del repositorio remoto (ej. GitHub/GitLab), usarás `git push`.

```
Unset

# Elimina la rama 'feat-visualizacion-datos' del remoto 'origin'
git push origin --delete feat-visualizacion-datos
```

Esto se profundizará en la clase de Git Remoto.

4.3. Podar Ramas Locales Obsoletas (`git remote prune`)

A veces, un compañero elimina una rama remota, pero tú todavía tienes una referencia local a ella. `git remote prune` te ayuda a limpiar estas referencias obsoletas.

```
Unset

# Simula la poda (dry run) para ver qué se eliminaría
git remote prune origin --dry-run
# Salida: * [would prune] origin/old-feature-branch
```

```
# Realiza la poda
git remote prune origin
# Salida: * [pruned] origin/old-feature-branch
```

5. Actividad Práctica (Laboratorio 4)

Vamos a simular un escenario de colaboración donde surgirán conflictos y practicaremos su resolución.

Paso 1: Preparación del Entorno

1. Asegúrate de estar en un directorio de trabajo limpio. Si continuas desde la clase anterior, usa `cd ..` y luego `mkdir clase4-lab-ds` y `cd clase4-lab-ds`.
2. Inicializa un nuevo repositorio Git:

```
Unset
git init
```

3. Crea un archivo `config.py` con contenido inicial:

```
Unset
# config.py

# Configuración de parámetros del modelo
learning_rate = 0.01
epochs = 100
```

4. Realiza el primer commit:

```
Unset
git add config.py
git commit -m "Initial commit: Add base configuration file"
```

5. Verifica con `git log --oneline`.

Paso 2: Escenario de Fusión Simple (Fast-forward)

1. Crea una nueva rama `feat-ajuste-epochs` y cámbiate a ella:

```
Unset
git switch -c feat-ajuste-epochs
```

2. Modifica `config.py` para cambiar el número de `epochs`:

```
Unset
# config.py
# Configuración de parámetros del modelo
learning_rate = 0.01
epochs = 200 # Aumentado el número de epochs
```

3. Realiza un commit en esta rama:

```
Unset
git add config.py
git commit -m "feat: Aumentar epochs a 200 para mejor convergencia"
```

4. Vuelve a la rama `main`:

```
Unset
git switch main
```

5. Fusiona `feat-ajuste-epochs` en `main`. Observa que será un `fast-forward` merge porque `main` no ha tenido nuevos commits.

```
Unset
git merge feat-ajuste-epochs

# Salida: Fast-forward
```

6. Verifica el historial con `git log --oneline --graph`. Deberías ver una línea recta.
7. Elimina la rama `feat-ajuste-epochs`:

```
Unset
git branch -d feat-ajuste-epochs
```

Paso 3: Escenario de Fusión con Conflicto

1. Asegúrate de estar en la rama `main`.
2. Crea una nueva rama `feat-nueva-metrica` y cámbiate a ella:

```
Unset
git switch -c feat-nueva-metrica
```

3. Modifica `config.py` en `feat-nueva-metrica`, añade una nueva métrica justo después de `epochs`:

```
Unset
# config.py

# Configuración de parámetros del modelo
learning_rate = 0.01
epochs = 200
metric_threshold = 0.85 # Nueva métrica para el experimento A
```

4. Realiza un commit en esta rama:

```
Unset
git add config.py
git commit -m "feat: Añadir umbral de métrica para nuevo
experimento"
```

5. Vuelve a la rama `main`:

```
Unset
git switch main
```

6. Ahora, **en la rama** `main`, modifica `config.py` en la **misma línea o cerca** de donde la otra rama añadió la métrica. Por ejemplo, añade un comentario o una variable relacionada con el `learning_rate` justo después de `epochs`:

```
Unset
# config.py
# Configuración de parámetros del modelo
learning_rate = 0.01
epochs = 200
# Comentario: El learning rate se ajustará dinámicamente.
```

7. Realiza un commit en la rama `main`:

```
Unset
git add config.py
git commit -m "docs: Añadir comentario sobre ajuste dinámico de
learning rate"
```

8. Intenta fusionar `feat-nueva-metrica` en `main`. **¡Esto generará un conflicto!**

```
Unset
git merge feat-nueva-metrica
# Salida: CONFLICT (content): Merge conflict in config.py
```

9. **Resuelve el conflicto:**
    - Verifica `git status`. `config.py` estará como "Unmerged".
    - Abre `config.py` con tu editor. Verás las marcas `<<<<<<< HEAD`, `=======`, `>>>>>>>`.
    - Edita el archivo para combinar ambos cambios como desees. Por ejemplo, puedes mantener ambos y eliminar solo las marcas:

```
Unset
# config.py (después de la resolución manual)
# Configuración de parámetros del modelo
learning_rate = 0.01
epochs = 200
metric_threshold = 0.85 # Nueva métrica para el experimento A
# Comentario: El learning rate se ajustará dinámicamente.
```

    - Guarda el archivo.
    - Marca el archivo como resuelto:

```
Unset
git add config.py
```

- Completa la fusión con un commit:

```
Unset
git commit -m "Merge branch 'feat-nueva-metrica' with conflict
resolution"
```

10. Verifica el historial con `git log --oneline --graph --all`. Deberías ver un "merge commit" con dos padres.
11. Elimina la rama `feat-nueva-metrica`:

```
Unset
git branch -d feat-nueva-metrica
```

12. *Si te da error al eliminar la rama por no estar completamente fusionada, es posible que no hayas completado el commit de fusión correctamente. Revisa `git status` o usa `git branch -D` si estás seguro.*

Paso 4: Escenario de Fusión con Squash

1. Asegúrate de estar en la rama `main`.
2. Crea una nueva rama `exp-pruebas-rapidas` y cámbiate a ella:

```
Unset
git switch -c exp-pruebas-rapidas
```

3. Realiza varias modificaciones pequeñas y commits separados en esta rama, por ejemplo:
   - **Primer commit:** Modifica `config.py` para añadir `seed = 42`.

```
Unset
# config.py
```

```
# Configuración de parámetros del modelo
learning_rate = 0.01
epochs = 200
metric_threshold = 0.85
# Comentario: El learning rate se ajustará dinámicamente.
seed = 42 # Añadir semilla para reproducibilidad
```bash
git add config.py
git commit -m "exp: Add random seed to config"
```

- **Segundo commit:** Crea un archivo `temp_test.py` con un print simple.

```
Unset
# temp_test.py
print("Esto es una prueba temporal.")
```bash
git add temp_test.py
git commit -m "exp: Create temp test script"
```

4. Vuelve a la rama `main`:

```
Unset
git switch main
```

5. Fusiona `exp-pruebas-rapidas` usando `squash`:

```
Unset
git merge --squash exp-pruebas-rapidas

# Git te mostrará que los cambios están en staging, listos para un commit.
```

6. Realiza el commit. Git te sugerirá un mensaje que combina los de la rama de origen, pero puedes escribir uno nuevo que resuma el "resultado final" del experimento:

```
Unset
git commit -m "feat: Integrar resultados de pruebas rápidas (incluye seed y script temporal)"
```

7. Verifica el historial con `git log --oneline --graph --all`. Observa que no hay un "merge commit" y que los commits individuales de `exp-pruebas-rapidas` no se muestran en la línea de `main`.

6. Posibles Errores y Soluciones

Aquí te presento algunos errores comunes que podrías encontrar en esta clase y cómo solucionarlos.

1. **Error:** `fatal: You have not concluded your merge (MERGE_HEAD exists).`
   - **Causa:** Intentaste ejecutar otro comando de Git (como `git status`, `git commit`, `git switch`) mientras estabas en medio de un conflicto de fusión. Git te está esperando para que resuelvas el conflicto.
   - **Solución:** Resuelve los conflictos en los archivos, luego usa `git add <archivo_resuelto>` y finalmente `git commit` para completar la fusión. Si quieres abortar, usa `git merge --abort`.
2. **Error:** `error: The branch 'X' is not fully merged.` **al usar** `git branch -d`
   - **Causa:** Intentaste eliminar una rama con `git branch -d`, pero Git detecta que hay commits en esa rama que aún no han sido fusionados en tu rama actual (o en el `upstream` configurado). Git te protege de la pérdida de trabajo.
   - **Solución:**
     - **Opción A (Recomendada):** Asegúrate de que todos los commits importantes de esa rama se hayan fusionado correctamente en la rama de destino deseada (ej. `main`). Luego intenta `git branch -d` de nuevo.
     - **Opción B (Con precaución):** Si estás absolutamente seguro de que no necesitas esos commits y quieres descartar la rama, puedes forzar la eliminación con `git branch -D <nombre_rama>`. **¡Cuidado, esto puede resultar en pérdida de trabajo!**
3. **Conflictos inesperados al hacer** `git merge --squash`**:**

    - **Causa:** Aunque `squash` agrupa commits, sigue siendo una fusión a nivel de contenido. Si las mismas líneas fueron modificadas en `main` y en la rama que estás "squasheando", Git reportará un conflicto de contenido.
    - **Solución:** Resuelve los conflictos manualmente en el archivo, luego `git add <archivo_resuelto>`, y finalmente `git commit` para completar el proceso.
4. `git merge` **dice** `Already up to date.` **pero crees que hay cambios:**
    - **Causa:** Esto significa que la rama que intentas fusionar no tiene commits nuevos que no estén ya en tu rama actual. O bien no has commiteado los cambios en la rama de origen, o ya se fusionaron previamente.
    - **Solución:** Usa `git log --oneline --graph --all` para visualizar el historial de ambas ramas y ver si la rama de origen realmente tiene commits nuevos que aún no están en la de destino. Asegúrate de haber hecho `git add` y `git commit` en la rama de origen.

Conclusión de la Clase 4

¡Excelente trabajo, científicos de datos! En esta clase, hemos dominado una de las operaciones más importantes de Git: la **fusión de ramas (**`merge`**)**. Has aprendido a diferenciar entre los tipos de fusión (Fast-forward, No fast-forward, Squash) y, crucialmente, has practicado la **resolución de conflictos**, una habilidad indispensable en el trabajo colaborativo. Finalmente, sabes cómo mantener tu repositorio limpio eliminando ramas que ya no necesitas.

Estás listo para dar el siguiente gran paso: trabajar con **repositorios remotos**. Esto nos abrirá las puertas a la verdadera colaboración en equipo, que será el enfoque de nuestra próxima clase.