Unidad 2 – Control de versiones con Git y GitHub

Sesión 7 – Ramas, merges y resolución de conflictos

1. Introducción a Git y las ramas

- Git es un sistema de **control de versiones distribuido** que permite registrar los cambios realizados en el código fuente de un proyecto y coordinar el trabajo entre varios desarrolladores.

- A diferencia de los sistemas centralizados, en Git cada usuario tiene una copia completa del repositorio, incluyendo su historial y ramas.

- Esto hace posible trabajar sin conexión y experimentar sin riesgo.

- Las ramas (branches) representan líneas independientes de desarrollo.

- Cada rama es un puntero que apunta a un conjunto de commits.

- Trabajar con ramas permite aislar desarrollos, probar nuevas ideas o corregir errores sin afectar la versión principal del proyecto (generalmente llamada main o master).

2. Concepto de merge y conflictos

- Cuando se combinan ramas en Git se utiliza el comando **merge**.
- Este proceso **fusiona los cambios de una rama en otra**, manteniendo la historia completa de ambas.
- Si los cambios se realizan en archivos diferentes, **Git los combina automáticamente.**
- Sin embargo, **si dos ramas modifican las mismas líneas de un archivo, aparecerá un conflicto que debe resolverse manualmente.**
- Durante un **conflicto**, Git marca las diferencias en el archivo usando los **delimitadores <<<<<<<, ======= y >>>>>>>.**
- El desarrollador debe decidir qué versión conservar o combinar los cambios de ambas ramas.

- Una vez resuelto el conflicto, se ejecuta **'git add'** para marcar el archivo como resuelto y **'git commit'** para finalizar la fusión.

3. Ejemplo práctico de conflicto

Supongamos que dos programadores trabajan sobre el archivo index.html.

Uno modifica el encabezado, y otro cambia el mismo texto en su propia rama.

Cuando se intenta hacer un merge, aparece el conflicto:

<<<<<<< HEAD
<h1>Bienvenido a nuestra web</h1>
=======
<h1>Bienvenido al portal</h1>
>>>>>>> feature/texto-encabezado

Para resolverlo, basta con editar el archivo, eliminar los marcadores y dejar la versión final deseada:

<h1>Bienvenido al portal principal</h1>

4. Estrategias de branching

Las estrategias de **branching** definen cómo se organizan las ramas y cómo fluye el trabajo entre ellas.

Existen varios modelos adaptados a distintos tipos de proyectos y equipos:

Estrategia	Descripción	Ventajas	Desventajas	Casos de uso	Ejemplo real
Git Flow	Usa ramas fijas (main, develop, feature, release, hotfix).	Control exhaustivo y gestión de versiones estables.	Complejo y menos ágil para despliegues frecuentes.	Software empresarial con ciclos de liberación largos.	Siemens, Mozilla Firefox
GitHub Flow	Solo main y ramas cortas (feature/*) con Pull Requests.	Sencillo, ideal para integración continua y despliegue rápido.	No contempla versiones simultáneas ni releases largas.	Startups, aplicaciones web y servicios en la nube.	Netflix, Shopify
Trunk Based	Todos los desarrolladores trabajan sobre	Permite CI/CD continuo y desplie	Requiere alto nivel de testing automatizado.	Grandes equipos DevOps y proyectos CI/CD.	Google, Facebook

	main con commits frecuentes.	gues diarios.			
Hotfix	Ramas de emergencia creadas para corregir errores críticos.	Rápido y enfocado a producción.	Temporal y no estructurado a largo plazo.	Entornos de mantenimiento y soporte.	GitLab, empresas de hosting

5. Casos de uso empresarial

- Netflix utiliza GitHub Flow para desplegar actualizaciones diarias de su plataforma de streaming.
- Siemens adopta Git Flow para garantizar control y trazabilidad en software industrial.
- Google aplica Trunk Based Development con integración continua y revisiones automáticas.
- GitLab combina GitHub Flow y Hotfixes para gestionar incidencias críticas en producción.

6. Actividades prácticas

Actividad 1: Creación y fusión de ramas.

1. Crea un repositorio local y añade un archivo README.md.

```
mkdir mi-repo && cd mi-repo
git init
echo "# Mi proyecto" > README.md
git add README.md
git commit -m "chore: init repo with README"
```

2. Crea una rama llamada feature/presentacion.

```
git switch -c feature/presentacion
# (alternativa antigua) git checkout -b feature/presentacion
```

3. Añade un cambio y haz commit.

```
echo "- Presentación inicial del proyecto" >> README.md
```

git add README.md

git commit -m "feat: añadir sección de presentación al README"

4. Cambia a main y fusiona los cambios.

git switch main

si tu rama principal se llama master: git switch master

git merge feature/presentacion --no-ff -m "merge: integrar presentación"

5. Elimina la rama.

git branch -d feature/presentacion

si ya la subiste al remoto y quieres borrarla allí:

git push origin --delete feature/presentación

git log --oneline --graph --decorate –all

Actividad 2: Simulación de conflicto.

1. Dos alumnos modifican la misma línea de un archivo en ramas distintas.

en main crea un archivo base

echo "línea 1" > notas.txt

echo "línea 2 - ORIGINAL" >> notas.txt

echo "línea 3" >> notas.txt

git add notas.txt

git commit -m "feat: archivo base para conflicto"

Alumno Rama A:

git switch -c rama-a

editar la MISMA línea 2

sed -i '' 's/ORIGINAL/EDITADA POR A/' notas.txt 2>/dev/null || sed -i 's/ORIGINAL/EDITADA POR A/' notas.txt

git commit -am "feat: A cambia la línea 2"

git switch main

Alumno Rama B:

git switch -c rama-b

editar la MISMA línea 2 pero con texto distinto

sed -i '' 's/ORIGINAL/EDITADA POR B/' notas.txt 2>/dev/null ||
sed -i 's/ORIGINAL/EDITADA POR B/' notas.txt

git commit -am "feat: B cambia la línea 2"

2. Intentan hacer merge y resuelven el conflicto manualmente.

Intentar fusionar B (provoca conflicto)

git merge rama-b

Esperado: CONFLICT (content): Merge conflict in notas.txt

Resolver el conflicto (manual)

1. Abre notas.txt: verás marcadores así:
2. línea 1
3. <<<<<<< HEAD
4. línea 2 - EDITADA POR A
5. =======
6. línea 2 - EDITADA POR B
7. >>>>>>> rama-b
8. línea 3

9. Decide el resultado. Ejemplo, **combinar**:
10. línea 1
11. línea 2 - EDITADA POR A y B
12. línea 3
13. Añade y finaliza el merge:
14. git add notas.txt
15. git commit -m "fix: resolver conflicto combinando cambios de A y B"

Limpieza opcional

git branch -d rama-a rama-b

Actividad 3: Estrategia GitHub Flow.

1. Crea una rama feature/footer.
2. Realiza commits, sube a GitHub y crea un Pull Request.
3. Fusiona tras la revisión y elimina la rama.

Requiere tener un repositorio en GitHub vacío creado y el remoto configurado.

0) Configurar remoto (si no lo tienes)

crea repo en GitHub (p.ej. https://github.com/tu-usuario/mi-repo)

git remote add origin git@github.com:tu-usuario/mi-repo.git

git push -u origin main   # primer push de main

1) Crear rama feature/footer

git switch -c feature/footer

echo "<footer>© 2025 Mi Web</footer>" > footer.html

git add footer.html

git commit -m "feat(footer): estructura inicial del pie de página"

2) Realizar commits y subir a GitHub

otro commit de mejora

echo "<footer>© 2025 Mi Web · Contacto</footer>" > footer.html

git commit -am "feat(footer): añadir enlace de contacto"

subir rama

git push -u origin feature/footer

3) Crear Pull Request

- Ve a GitHub → el repo → verás un banner “Compare & pull request” → **Create pull request**.
- Rellena título/descr. (qué y por qué), asigna reviewers.

(Opcional, vía CLI con GitHub CLI si la tienes: gh pr create --fill)

4) Revisar, fusionar y eliminar

- Tras la revisión: **Merge** (preferible *Squash & merge* o *Merge commit* según política).
- Marca **“Delete branch”** para borrar feature/footer en remoto.
- Localmente:
- git switch main
- git pull origin main
- git branch -d feature/footer

Verificación

git log --oneline --graph --decorate --all

git branch -a

7. Evaluación por competencias

Competencia	Indicadores de logro	Nivel esperado
Uso técnico de Git	Aplica comandos correctamente para gestionar ramas y merges.	Realiza merges exitosos sin errores.
Colaboración	Coordina ramas y revisiones mediante Pull Requests.	Participa activamente en el flujo de trabajo del equipo.
Resolución de conflictos	Identifica y soluciona conflictos en el código.	Resuelve conflictos correctamente y documenta el proceso.
Documentación	Escribe mensajes de commit y PR claros.	Mensajes concisos, descriptivos y normalizados.