DAM 1 DAW 1 / Entornos de desarrollo

Control de versiones con Git y GitHub

3.2 Uso avanzado de Git

Autor: Lisa ERIKSEN
Fecha: 2025 / 2026

1. Objetivos de Aprendizaje

En este capítulo exploraremos comandos y flujos de trabajo avanzados que te permitirán aprovechar al máximo las capacidades de Git. Aprenderás a trabajar con ramas (branches), resolver conflictos de fusión (merge conflicts), y usar herramientas como .gitignore para gestionar tu repositorio de manera más eficiente.

2. Trabajar con ramas (branches)

Una rama es una versión paralela de tu repositorio. Las ramas te permiten trabajar en diferentes características, correcciones o experimentos sin afectar la rama principal (main o master).

Comando	Descripción
git branch	Lista las ramas existentes.
git branch nombre_rama	Crea una nueva rama.
git checkout nombre_rama	Cambia a la rama especificada.
git switch nombre_rama	Alternativa moderna a git checkout.
git merge nombre_rama	Fusiona una rama con la actual.
git branch -d nombre_rama	Elimina una rama.
git branch	Lista las ramas existentes.
git branch nombre_rama	Crea una nueva rama.

3. Ejemplo práctico: crear y fusionar ramas

Tu equipo está desarrollando un pequeño juego inspirado en **GTA**, donde el jugador controla un personaje en una ciudad. El proyecto ya incluye un programa básico en Java que muestra el nombre y el dinero del personaje.

Tu misión será:

- Crear una **nueva rama.**
- Crear una **nueva funcionalidad** en una rama separada.
- Fusionar esa rama con la principal (main)
- Resolver un conflicto simple.

3.1 Crea el proyecto

Creación del proyecto

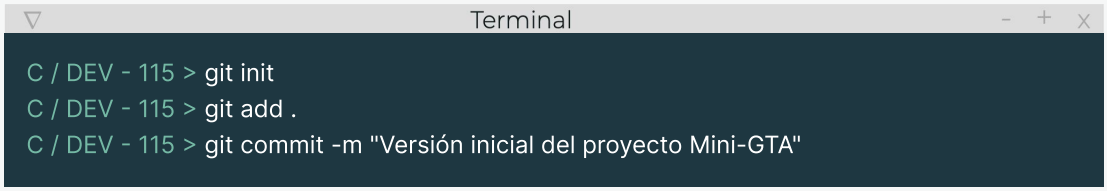
1. Crea una carpeta llamada Git_GTA.
2. Ábrela con VS Code.
3. Cambia el nombre de la rama a main (se llama master por defecto): **git branch -M main**
4. Crea el archivo **Personaje.java.**
5. Copia el siguiente código:

```
public class Personaje {
    private String nom;
    private int dinero;

    public Personaje(String nom, int dinero) {
        this.nom = nom;
        this.dinero = dinero;
    }

    public void mostrarInfo() {
        System.out.println(nom + " tiene " + dinero + "$.");
    }

    public static void main(String[] args) {
        Personaje jugador = new Personaje("Trevor", 500);
        jugador.mostrarInfo();
    }
}
```

3.2 Inicia el repositorio

Inicializa el repositorio

Abre el terminal de VS Code y ejecuta el comando para initializar GIT



3.3 Crea una nueva rama

La nueva funcionalidad será: el personaje puede ganar dinero

Usa la lista de comandos de la parte 2. para:

- Crear una nueva rama que se llama **ganar-dinero**
- Cambiar a esa rama

3.4 Realiza cambios

Implementa una nueva funcionalidad: haz que tu personaje pueda ganar dinero

- Completa el método **ganarDinero** de abajo para aumentar la cantidad de dinero del personaje. El programa debe mostrar por ejemplo: *Trevor gana 250$!*

```
public void ganarDinero ( Añade una variable aquí )  {

        Añade una línea para modificar la variable
        Añade una línea para escribir la frase final
}
```

- Añade el nuevo método en **main**: jugador.ganarDinero(250);

- Prueba el programa para y verifica que todo funciona bien.

3.5 Registra tus cambios en GIT

Registrar los cambios en GIT

Haz un commit para guardar y registrar tus cambios

```
Terminal
C / DEV - 115 > git add .
C / DEV - 115 > git commit -m "Funcionalidad: el personaje puede ganar dinero"
```

3.6 Fusionar con la rama principal

Fusionar con la rama principal main

1. Cambia de rama a main y verifica que no tiene la nueva funcionalidad ganarDinero()
2. Añade los cambios de la rama ganar-dinero en tu rama principal main con;

```
Terminal
C / DEV - 115 > git switch main
C / DEV - 115 > git merge ganar-dinero
```

Preguntas de reflexión:

- ¿Qué comando se usa para cambiar de rama?
- ¿A que sirve de hacer la nueva funcionalidad en una nueva rama antes de fusionar?
- ¿Qué comando se usa para fusionar dos ramas?
- ¿Por qué podemos eliminar la rama después de la fusión?

4. Resolver un conflicto

Creación de un conflicto

1. Crea una nueva rama llamada **conflicto**
2. Cambia el nombre de peronaje en main() a **Franklin** y haz un commit
3. Regresa a main y cambia el nombre a Trevor y
4. Haz un commit del cambio de nombre
5. Fusiona la rama **conflicto** con **main**

👉 Git mostrará un **conflicto**: resuélvelo editando el archivo y eligiendo **un solo nombre**.

5. Ignorar archivos con .gitignore

Durante el desarrollo del juego, pueden existir archivos **que no deben subirse** al repositorio:

- **dinero_debug.log**: archivo temporal que registra ganancias.
- **trucos.txt**: contiene códigos secretos del jugador.

Estos archivos son útiles localmente, pero no forman parte del código del juego → deben ser ignorados.

4.1 Que es un archivo .gitignore?

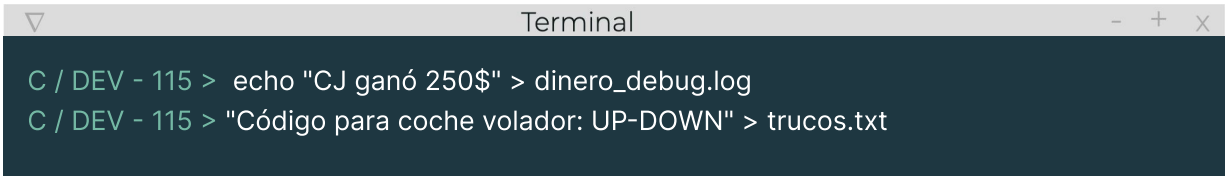
.gitignore es un archivo en tu proyecto donde le dices a Git qué archivos **debe ignorar**.
Esto significa que esos archivos **no aparecerán en git status** y **no podrán añadirse por error**.

4.2 Crear un archivo .gitignore

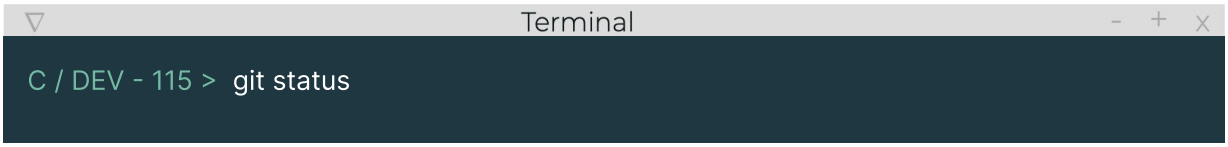
1. En la carpeta principal del proyecto, crea un archivo llamado: .gitignore
2. Agrega las siguientes líneas:

```
# Archivo de registro de pruebas
dinero_debug.log

# Archivo secreto de trucos del jugador
trucos.txt
```

3. Crea los archivos con el terminal para probar:



4. Verifica que Git ignora los archivos especificados:



Resultado esperado:

Los archivos **NO deben aparecer como "archivos no rastreados"** (untracked files).
Si no aparecen → **¡Git los está ignorando correctamente!**

Preguna de reflexion

- En una aplicación empresarial, ¿qué tipo de archivos o información deberíamos evitar subir al repositorio y proteger mediante .gitignore? Explica por qué.