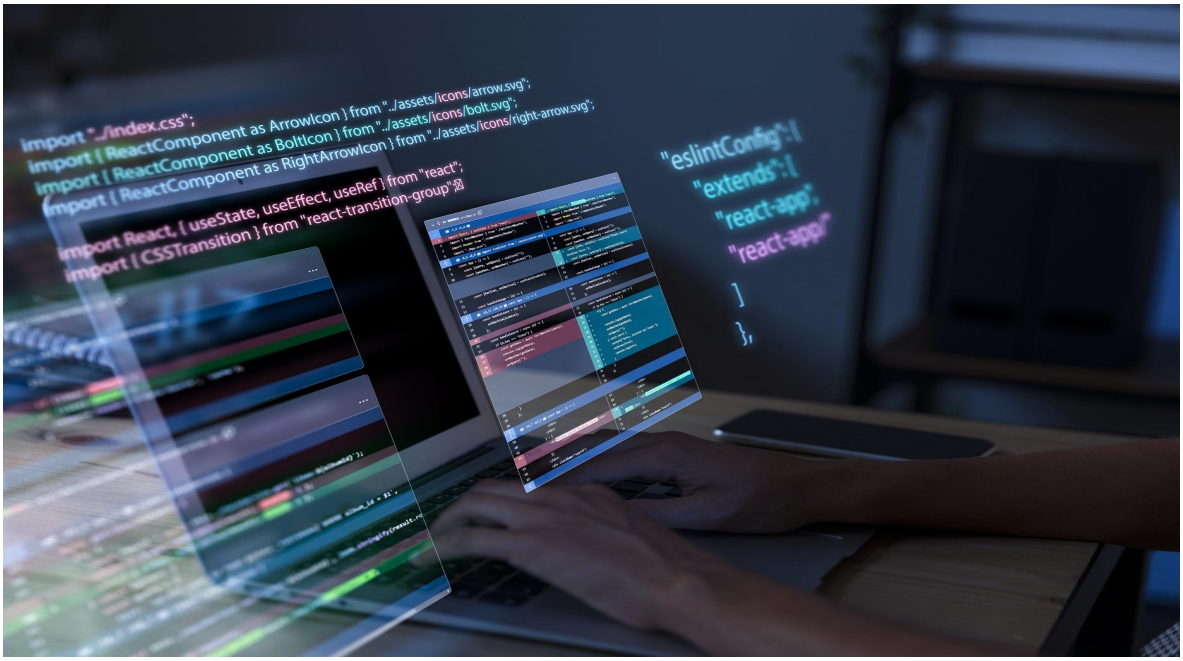
DAM & DAW /ENTORNOS DE DESARROLLO

UNIDAD 2

Control de versiones con Git y GitHub



Autor: Luz María Álvarez Moreno

Fecha: 21/10/2025

Sesión 5 – Introducción Teórica a Git: Repositorio, Commit e Historial

1. Objetivos de aprendizaje

Al finalizar la sesión, el alumnado será capaz de:

- Comprender el concepto de **control de versiones** y su importancia.
- Entender qué es **Git** y por qué es una herramienta esencial para programadores y equipos.
- Definir los conceptos de **repositorio**, **commit** e **historial**.
- Reconocer las ventajas de Git frente a métodos tradicionales de guardar archivos o trabajar en la nube.
- Conocer cómo se instala y configura Git paso a paso (Windows, macOS y Linux).
- Comprender la diferencia entre un **repositorio local** y uno **remoto** (como GitHub).

2. Introducción: el problema sin Git

2.1. Antes de Git

Cuando los programadores no tenían control de versiones, usaban carpetas para guardar diferentes versiones de un proyecto:

- Proyecto_final
- Proyecto_final_v2
- Proyecto_definitivo_ahora_sí

Esto genera confusión:

- ¿cuál es la última versión?
- ¿Qué cambió?
- ¿Quién lo cambió?

2.2. ¿Qué necesitamos?

Un sistema que:

- Guarde cada cambio de forma ordenada.
- Permita **volver atrás** si algo falla.
- Muestre **qué se modificó**, **cuándo** y **por quién**.
- Permita **trabajar en equipo** sin perder información.

2.3. Solución: los sistemas de control de versiones (VCS)

Un **Sistema de Control de Versiones** (VCS) permite llevar un registro de los cambios realizados sobre un conjunto de archivos.

Ejemplos:

- CVS (Concurrent Versions System)
- SVN (Subversion)
- **Git (el más popular)**

Git fue creado por **Linus Torvalds** (creador de Linux) en 2005 para mejorar el manejo del código del sistema operativo.

3. Comparativa: Git vs trabajar en la nube o en local

Característica	Trabajar solo en local	Trabajar en la nube (Drive, Dropbox…)	Trabajar con Git
Dónde se guarda	En el ordenador personal	En un servidor de almacenamiento en línea	En tu equipo (local) y opcionalmen

			te en un servidor remoto (GitHub, GitLab...)
Control de versiones	Manual (copiando carpetas)	Automático, pero no entiende el código	Total, línea por línea, con historial completo
Trabajo colaborativo	Difícil: se sobrescriben archivos	Puede haber conflictos de sincronización	Integrado, con ramas y fusiones controladas
Revisión de cambios	No hay historial	Solo versiones de archivos	Historial detallado con autor, fecha y descripción
Necesidad de conexión a Internet	No necesaria	Obligatoria	No necesaria (local), pero útil para sincronizar con remoto

Uso profesional en desarrollo	Muy limitado	No especializado	Estándar en la industria
Seguridad de los datos	Riesgo de pérdida si falla el disco	Depende del servicio en la nube	Copias distribuidas en cada clon
Gestión de errores	Manual (sobrescribir)	Revisión básica de archivos	Permite revertir cambios fácilmente

Conclusión:

- Git combina **lo mejor de ambos mundos**: la rapidez del trabajo local y la seguridad del respaldo en la nube.
- Permite trabajar **sin conexión**, guardar **versiones profesionales** y colaborar sin riesgos.

Ejemplo práctico:

- Si trabajas en Google Drive y alguien borra tu archivo, lo pierdes salvo que haya copia.
- Si trabajas en Git, puedes volver al commit anterior y recuperar el archivo tal como estaba.

4. ¿Qué es Git?

Git es un **sistema de control de versiones distribuido**.

Esto significa que cada persona que trabaja en un proyecto tiene **una copia completa del historial del proyecto**.

4.1. Ventajas principales

- **Seguridad:** el historial se guarda en cada ordenador.
- **Velocidad:** trabaja localmente, sin depender de un servidor central.
- **Trabajo sin conexión:** puedes hacer cambios aunque no tengas internet.
- **Colaboración:** varias personas pueden trabajar al mismo tiempo sin sobrescribir los cambios de otros.

4.2. Concepto de “control de versiones”

Git toma “**fotografías**” de tu proyecto (commits) cada vez que guardas los cambios.

Luego puedes retroceder a cualquier punto anterior.

5. Conceptos básicos

5.1. Repositorio

Un **repositorio** es la carpeta donde Git guarda toda la información del proyecto y su historial.

Puede ser:

- **Local:** en tu ordenador.
- **Remoto:** en una plataforma online (GitHub, GitLab, Bitbucket…).

5.2. Commit

Un **commit** es una "foto" de los archivos en un momento determinado.

Cada commit guarda:

- Qué se cambió.
- Cuándo se cambió.
- Quién lo cambió.

5.3. Historial

El **historial** es la secuencia ordenada de todos los commits realizados.

5.4. Ramas (Branch)

Las **ramas** son líneas de trabajo paralelas.

Permiten desarrollar nuevas ideas sin afectar la versión principal.

Luego pueden fusionarse (merge).

6. Instalación de Git

6.1. Descarga oficial

Ir a https://github.com/

Haz clic en **“Download for SSOO”**

6.2. Instalación en Windows

1. Ejecuta el archivo Git-<versión>-64-bit.exe.
2. Acepta la licencia.
3. Deja las opciones por defecto.
4. Selecciona **Visual Studio Code** como editor si lo tienes.
5. Finaliza e inicia **Git Bash**.

6.3. macOS

- Opción A: descarga desde la web oficial.
- Opción B: usa Homebrew.

brew install git

6.4. Linux (Ubuntu/Debian)

sudo apt update

sudo apt install -y git

6.5. Comprobación de instalación

git --version

6.6. Configuración inicial

git config --global user.name "Tu Nombre"

git config --global user.email "tu@email.com"

7. Git local y remoto

7.1. Git local

- Instalada en tu ordenador.
- Guarda el historial completo.
- Puedes trabajar sin internet.

7.2. Git remoto

- Copia alojada en un servidor.
- Permite compartir el proyecto.
- GitHub, GitLab y Bitbucket son los más conocidos.

7.3. Ejemplo visual

Tu PC (repositorio local)

⇕ git push / git pull ⇕

GitHub (repositorio remoto)

8. Conclusiones

- Git es una herramienta profesional que **controla versiones** y **facilita el trabajo en equipo**.
- Combina las ventajas del trabajo local (rapidez) y de la nube (colaboración y respaldo).
- Saber usar Git es un requisito básico en cualquier área de programación moderna.

9. Glosario

- **Repositorio:** carpeta con el historial del proyecto.
- **Commit:** registro de un cambio guardado.
- **Historial:** secuencia de commits.
- **Branch:** rama de desarrollo.

- **Git local:** versión de Git en tu equipo.
- **Git remoto:** versión en la nube (GitHub, GitLab, etc.).

10. Actividades

1. **Debate:** ¿Qué ventajas tiene usar Git frente a Google Drive o Dropbox?
2. **Ejercicio visual:** dibuja un flujo de commits y cómo se guarda la historia.
3. **Exploración:** busca un repositorio en GitHub y observa su historial de commits.

11. Comandos esenciales de Git

Pensado para primera toma de contacto. Cada comando incluye **qué hace**, **cuándo usarlo** y **un ejemplo** listo para copiar.

11.1. Comandos de sistema y ayuda

git - - version (los guiones van sin espacio)

- **Qué hace:** muestra la versión instalada.
- **Cuándo usar:** tras instalar o para comprobar requisitos.

Ejemplo:

git --version

git help <comando> / <comando> - -help

- **Qué hace:** abre la ayuda del comando.
- **Cuándo usar:** para ver opciones y ejemplos oficiales.

Ejemplo:

git help commit

 # o

git commit --help

11.2. Configuración inicial

git config --global user.name "Nombre"

git config --global user.email correo@ejemplo.com

- **Qué hace:** define tu identidad para los commits.
- **Cuándo usar:** la primera vez (o si cambias de identidad).

Ejemplo:

git config --global user.name "Ana García"

git config --global user.email "ana@example.com"

git config --global init.defaultBranch main

- **Qué hace:** fija main como rama por defecto al crear repos.

Ejemplo:

git config --global init.defaultBranch main

git config –list

- **Qué hace:** muestra la configuración actual.

Ejemplo:

git config --list

11.3. Crear y preparar un repositorio

git init

- **Qué hace:** convierte la carpeta actual en un repositorio Git (crea .git).
- **Cuándo usar:** para empezar a versionar un proyecto local.

Ejemplo:

mkdir mi-proyecto **&&** cd mi-proyecto

git init

git status

- **Qué hace:** muestra el estado (archivos nuevos, modificados, preparados).

- **Cuándo usar:** constantemente, antes de add o commit.

Ejemplo:

git status

git add <archivo> / git add .

- **Qué hace:** mueve cambios al **Staging Area** (preparados).
- **Cuándo usar:** antes de crear un commit.

Ejemplo:

echo "Hola" > README.md

git add README.md

 # o todo lo cambiado

git add .

git commit -m "mensaje"

- **Qué hace:** guarda un **snapshot** de los cambios preparados.
- **Cuándo usar:** tras git add, con mensaje claro.

Ejemplo:

git commit -m "feat: añade README inicial"

git log / git log --oneline

- **Qué hace:** muestra el **historial** de commits.
- **Cuándo usar:** para revisar versiones y autores.

  - **Ejemplo:**

git log --oneline

git show <hash>

- **Qué hace:** enseña detalles de un commit específico.

Ejemplo:

git show HEAD

git diff

- **Qué hace:** compara cambios no preparados respecto a HEAD.
- **Cuándo usar:** antes de añadir o confirmar, para ver “qué cambió”.

Ejemplo:

echo "Nueva línea" >> README.md

git diff

echo + .gitignore

- **Qué hace:** crear/editar archivo para **ignorar** rutas o patrones.

Ejemplo:

echo -e "node_modules/

.env

.DS_Store" > .gitignore

git add .gitignore

git commit -m "chore: añade .gitignore"

11.4. Trabajo con ramas (básico)

git Branch

- **Qué hace:** lista ramas locales.

Ejemplo:

git branch

git switch -c <nombre>

- **Qué hace:** crea y cambia a una rama nueva.

Ejemplo:

git switch -c experimento

git switch <nombre>

- **Qué hace:** cambia a una rama existente.

Ejemplo:

git switch main

git merge <rama>

- **Qué hace:** fusiona la rama indicada en la rama actual.
- **Cuándo usar:** para integrar trabajo terminado.

Ejemplo:

git switch main
git merge experimento