# Metodología de Sistemas II

## Clase 2

## Herramientas de repositorios

Jeremías Fassi

2025

# Introducción a Git

¿Qué es un sistema de control de versiones?

¿Qué es Git?

Jeremías Fassi

2025

# ¿Qué es un sistema de control de versiones?

Un sistema de control de versiones (VCS) es una herramienta que nos permite registrar y gestionar **todos los cambios** que realizamos sobre el código fuente de un proyecto. ¿Por qué es tan importante?

- **Historial completo**: Cada modificación queda guardada en una especie de “línea de tiempo” que registra quién hizo el cambio, cuándo y por qué.

- **Trabajo en equipo**: Permite que varias personas trabajen en paralelo sobre los mismos archivos sin sobrescribir o perder cambios.

# ¿Qué es un sistema de control de versiones?

- **Recuperación de versiones**: Si algo se rompe, podemos volver a una versión anterior estable del código.
- **Colaboración profesional**: El uso de un sistema de control de versiones es un estándar básico, tanto para proyectos pequeños como para grandes desarrollos distribuidos.

# ¿Qué es un sistema de control de versiones?

Ejemplos históricos:

- CVS y Subversion (SVN): sistemas centralizados, donde toda la información estaba en un servidor único. Si el servidor fallaba, era imposible trabajar.

- Git: creado en 2005 por Linus Torvalds para gestionar el kernel de Linux. Hoy es el más utilizado en la industria, tanto en proyectos open source (código abierto) como en empresas privadas.

# ¿Qué es un sistema de control de versiones?

- **Linus Benedict Torvalds** es un ingeniero de software finlandés, conocido por iniciar y mantener el desarrollo del kernel Linux, basándose en el sistema operativo libre Minix creado por Andrew S. Tanenbaum, y también por aportar en algunas herramientas, utilidades y los compiladores desarrollados por el proyecto GNU.

# ¿Qué es Git?

Git es un sistema de control de versiones distribuido. Esto significa que:

- Cada desarrollador tiene una copia completa del repositorio, con todo su historial. Incluso sin conexión a internet, se puede trabajar normalmente (hacer commits, crear ramas, etc.).

- Seguro y performante: Git fue diseñado para ser eficiente en proyectos gigantes (como el kernel de Linux), pero también es rápido y liviano para proyectos pequeños.

# ¿Qué es Git?

- Flexible: Permite adaptar el flujo de trabajo según el tamaño y organización del equipo (trabajo individual, pequeños equipos, grandes organizaciones).

- Ecosistema enorme: Al ser el estándar actual, existen múltiples plataformas y servicios alrededor de Git: GitHub, GitLab, Bitbucket, etc.

En síntesis: Git es la herramienta que permite que los equipos modernos de software trabajen de manera colaborativa, segura y eficiente.

# Conceptos fundamentales

- Repositorio
- Commit
- Branch
- Merge
- Push / Pull

# Conceptos Fundamentales

- Repositorio → Contenedor del proyecto + historial.
- Commit → Registro de cambios confirmados.
- Branch → Línea paralela de desarrollo.
- Merge → Integrar cambios de una rama en otra.
- Push / Pull →
  - Push: subir cambios al remoto.
  - Pull: traer cambios desde el remoto.

# Repositorio

Un repositorio es la unidad básica de trabajo en Git.

- Contiene todos los archivos del proyecto, más el historial completo de cambios.
- Puede estar en:
  - Local: en la compu del desarrollador.
  - Remoto: en una plataforma como GitHub, GitLab o Bitbucket.
- La ventaja: cada copia local es un repositorio completo, con todo el historial incluido.

Ejemplo:

- Ejecutar `git init` crea un nuevo repositorio local en la carpeta actual.

# Commit

Un commit es un “punto de control” (*checkpoint*) en el historial del proyecto.

- Al hacer un commit, confirmamos los cambios en los archivos.
- Cada commit guarda:
  - Qué se cambió.
  - Quién lo hizo.
  - En qué fecha y hora.
  - Un mensaje descriptivo que explica el cambio.
- El commit debe ser claro y atómico: representar una única idea o cambio (ej. “Arreglo de bug en login”).

# Commit

Ejemplo:

- `git add users.js`
- `git commit -m "Fix in user validation"`

# Branch (rama)

Un branch es una línea paralela de desarrollo.

- Permite trabajar en nuevas funcionalidades sin afectar la rama o branch principal.
- La rama principal suele llamarse `main` o `master`.

Buenas prácticas:

- Usar nuevas ramas para *features*: `feature/nueva-funcionalidad`.
- Usar nuevas ramas para *bugs*: `bugfix/error-login`.

→ **Ventaja**: se puede experimentar sin romper el proyecto principal.

# Branch (rama)

Ejemplo:

- `git checkout -b feature/user-login`

# Merge (fusión)

Merge o “mergear” significa combinar los cambios de una rama dentro de otra.

- Caso típico: terminar una feature y “traerla” a main.
- Git intenta unir los cambios automáticamente.
- Si dos personas editaron la misma parte del mismo archivo, aparece un conflicto, que debe resolverse manualmente. Es decir, cuál de las modificaciones es la correcta.

Ejemplo:

- `git checkout main`
- `git merge feature/user-login`

# Push / Pull

Push: enviar nuestros commits locales al repositorio remoto.

- `git push origin main`

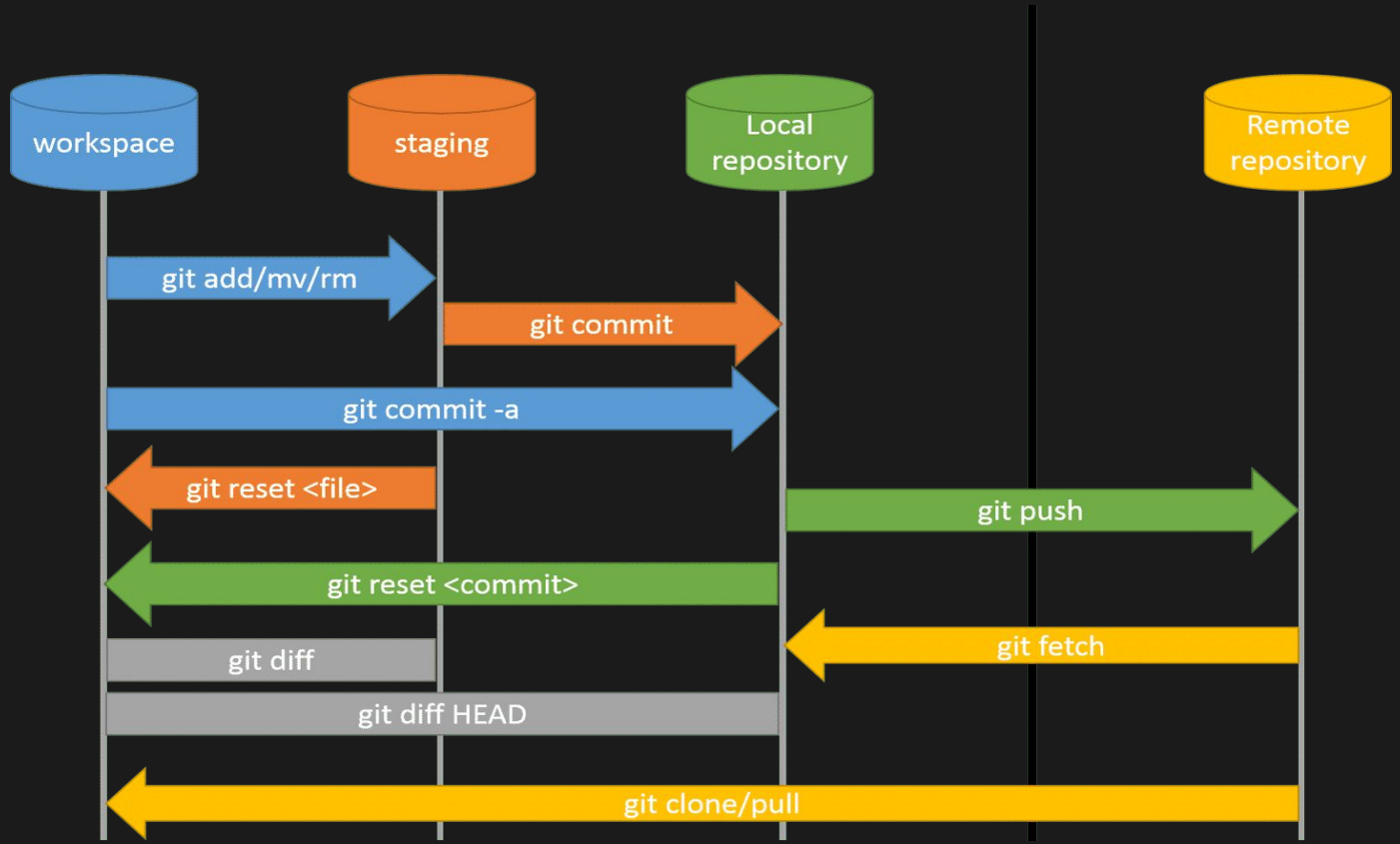
Pull: traer los commits más recientes desde el repositorio remoto.

- `git pull origin main`

Flujo típico de trabajo:

1. Hacemos cambios locales → commit.
2. Subimos con push.
3. Antes de seguir trabajando, actualizamos con pull.

# Flujo


workspace
staging
Local repository
Remote repository
git add/mv/rm
git commit
git commit -a
git reset <file>
git push
git reset <commit>
git fetch
git diff
git diff HEAD
git clone/pull

# GitHub

Es una plataforma para alojar repositorios Git en la nube. Funcionalidades clave:

- Repositorios remotos.
- Trabajo colaborativo (issues, pull requests, wiki).
- Control de acceso y permisos.
- Integración con CI/CD.

Diferencia con Git:

- Git → herramienta local.
- GitHub → servicio en la nube.

# ¿Qué es GitHub?

- GitHub es una plataforma en la nube que permite alojar repositorios Git de manera remota.
- Fue creada en 2008 y adquirida por Microsoft en 2018.
- Hoy es la plataforma de desarrollo colaborativo más utilizada del mundo, tanto en proyectos open source como privados.

# Funcionalidades principales

1. Repositorio remoto
   - GitHub funciona como el “punto de encuentro” donde los desarrolladores sincronizan su trabajo.
   - Los commits que hacemos en local se suben con git push al repositorio remoto.
2. Trabajo colaborativo
   - Issues: sistema de tickets para reportar bugs o discutir nuevas funcionalidades.
   - Pull Requests: mecanismo para proponer cambios y revisarlos antes de integrarlos al proyecto.
   - Wiki: documentación interna del proyecto.
   - Projects: tableros estilo Kanban para gestionar tareas.

# Funcionalidades principales

3. Control de acceso y permisos
    - Un repositorio puede ser público (visible para todos) o privado (acceso restringido).
    - Los colaboradores pueden tener distintos roles: lectura, escritura o administración.
4. Integraciones y automatización
    - GitHub Actions: sistema de CI/CD integrado que permite automatizar pruebas, despliegues y más.
    - Integración con herramientas externas: Slack, Discord, Trello, entre otras.

# Diferencia entre Git y GitHub

- Git: la herramienta en la computadora del desarrollador.
  - Funciona incluso sin internet.
  - Permite crear commits, ramas, merges, etc.
- GitHub: la plataforma en línea que centraliza el trabajo.
  - Permite colaborar con otros desarrolladores.
  - Gestiona repositorios remotos.
  - Agrega valor con herramientas de revisión, documentación y CI/CD.

Analogía:
Git es el motor y el auto. GitHub es el garage en la nube donde guardamos y compartimos el auto.

# Organización básica del repositorio

Archivos iniciales recomendados.

Convenciones de ramas.

Buenas prácticas.

Estructura básica de carpetas.

Jeremías Fassi

2025

# Organización básica del repositorio

## Archivos iniciales recomendados:

README.md

.gitignore

LICENSE (opcional)

## Convenciones de ramas:

main / develop

feature/...

bugfix/...

## Buenas prácticas:

Mensajes de commit claros.

Uso de tags para versiones.

## Estructura básica de carpetas:

```
/docs
/src
/tests
```

# Archivos iniciales recomendados

**README.md**

- Archivo de documentación principal del proyecto.
- Escrito en formato Markdown.
- Debería contener:
    - Nombre del proyecto.
    - Breve descripción.
    - Instrucciones de instalación/uso.
    - Datos de los integrantes/equipo.

# Archivos iniciales recomendados

**.gitignore**

- Lista de archivos o carpetas que Git debe ignorar.
- Ejemplos típicos:
  - Archivos de configuración local (`.env`).
  - Archivos generados automáticamente (`dist/`, `node_modules/`).
  - Archivos temporales del editor (`.vscode/`, `.idea/`).

# Archivos iniciales recomendados

**LICENSE** (opcional pero recomendable en proyectos públicos)

- Indica bajo qué condiciones se puede usar y modificar el software.
- Ejemplos:
  - MIT
  - GPL
  - Apache 2.0

# Convenciones de ramas (branch)

Mantener una estrategia clara de ramas ayuda a la organización:

- `main` o `master`: rama principal, estable, lista para producción.
- `develop`: rama de desarrollo, donde se integran nuevas funcionalidades antes de pasar a main.
- `feature/...`: ramas temporales para implementar nuevas funcionalidades.
- `bugfix/...`: ramas para corregir errores puntuales.

# Convenciones de ramas (branch)

- Ejemplo de flujo de trabajo:

    1. Crear `feature/login`.
    2. Desarrollar y hacer commits.
    3. Hacer un Pull Request hacia `develop`.
    4. Revisar y mergear.
    5. Cuando está probado, se integra `develop` en `main`.

# Buenas prácticas con commits y versiones

- Mensajes de commit claros y descriptivos.
  - Ejemplo correcto: Agrego validación de email en formulario de registro.
  - Ejemplo incorrecto: cambio cosas.
- Commits pequeños y atómicos: un cambio = un commit.
- Uso de tags o versiones: permiten marcar puntos importantes en la historia del proyecto (ejemplo: v1.0.0).

# Organización básica de carpetas

- Estructura sugerida desde el inicio del proyecto:

```
/docs      → Documentación del proyecto
/src       → Código fuente principal
/tests     → Pruebas del sistema
.gitignore
README.md
```

- Esto facilita el crecimiento ordenado del proyecto.
- Al ser una convención, otros desarrolladores podrán entender rápidamente cómo está organizado el código.

# Organización básica de carpetas

```
/docs
    /diagrams        → Diagramas UML, mockups, bocetos
    /manuals         → Manuales de usuario o técnicos
/src
    /config          → Archivos de configuración
    /controllers     → Lógica de negocio
    /models          → Representación de datos / entidades
    /routes          → Definición de rutas/endpoints
    /utils           → Funciones auxiliares
/tests
    /unit            → Pruebas unitarias
    /integration     → Pruebas de integración
    /mocks           → Datos simulados para pruebas
.gitignore           → Archivos/carpetas a ignorar por Git
README.md            → Documentación principal del proyecto
LICENSE              → Licencia (opcional)
```

# Recursos:

- Learn Git Branching
  Permite practicar ramas, merges, rebases con un entorno gráfico y comandos reales de Git.

- Oh My Git! (Juego)
  Un juego interactivo para aprender Git.

- Git y GitHub – Curso práctico (FreeCodeCamp, en español)
  Más de 5 horas, desde lo básico hasta pull requests y ramas.

- GitHub Git Cheat Sheet (PDF)
  Hoja de referencia rápida de comandos básicos.

# Muchas Gracias

Jeremías Fassi

Javier Kinter