Control de Versiones con Git y GitHub

Del trabajo local a la colaboración en la nube

Juan Ignacio da Torre

Laboratorio de Métodos Cuantitativos Aplicados a la Gestión
Universidad de Buenos Aires

Primer cuatrimestre 2026

Objetivo: Comprender qué es el control de versiones, cómo funciona Git y cómo GitHub facilita el trabajo colaborativo

Objetivo de la clase

¿Qué vamos a ver hoy?

- Entender qué es el **control de versiones** y por qué existe.
- Conocer **Git**: el sistema de control de versiones más utilizado del mundo.
- Aprender el **workflow básico**: cómo registrar cambios en un proyecto.
- Entender el concepto de **branches** (ramas) para trabajar en paralelo.
- Conocer **GitHub**: la plataforma de colaboración en la nube.
- **Tutorial paso a paso:** crear un repo, invitar colaboradores y hacer un commit.

Control de Versiones

El problema que todos tuvimos (y pocos resolvieron bien)

El problema: ¿Te suena familiar?

La carpeta del proyecto sin control

Todos conocemos esta situación:

- `informe_v1.docx`
- `informe_v2_final.docx`
- `informe_v2_final_REAL.docx`
- `informe_definitivo2.docx`
- `informe_USE_THIS_ONE.docx`

Problemas reales

- ¿Cuál es el que vale?
- ¿Qué cambió entre versiones?
- ¿Cómo vuelvo atrás si algo se rompe?
- ¿Cómo trabajo con otra persona sin pisarnos?

¿Qué es el Control de Versiones?

Definición

Un **sistema de control de versiones** (VCS) registra los cambios realizados sobre un archivo o conjunto de archivos a lo largo del tiempo, de modo que se puedan recuperar versiones específicas en cualquier momento.

Un buen sistema de versiones garantiza:

- Registrar cambios de forma concisa.
- Recuperar cualquier versión anterior.
- Revertir cambios sin perder historial.
- Comparar versiones fácilmente.

Analogía: La receta de té

Versión 1: Hervir agua, agregar té, infusionar 5 min, beber.
Versión 2: Igual que v1, pero **agregar azúcar** antes de beber.
Puedo volver a v1 cuando quiera, o saber exactamente qué cambió.

Git

El sistema de control de versiones distribuido más usado del mundo

¿Qué es Git?

Definición

Git es un sistema de control de versiones distribuido, diseñado para rastrear cambios en el código fuente durante el desarrollo de software.

Características clave:

- **Distribuido:** cada desarrollador tiene una copia completa del historial.
- **Gratis y open source:** libre de usar y modificar.
- **Liviano y rápido:** diseñado para eficiencia.
- **Branching y merging:** trabajo paralelo sin conflictos.

Git facilita la colaboración

Git permite que **diferentes equipos** trabajen colaborativamente sobre el mismo código fuente, integrando sus cambios de forma ordenada y trazable.

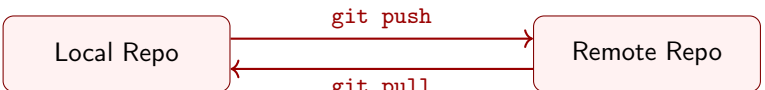
Repositorio Local vs. Repositorio Remoto

Repositorio Local

- Vive en tu computadora.
- Registrás cambios con `commit`.
- Trabajás sin necesidad de internet.
- Contiene todo el historial del proyecto.

Repositorio Remoto

- Vive en un servidor (GitHub, GitLab, etc.).
- Funciona como punto de sincronización.
- Permite el trabajo en equipo.
- Se actualiza con `push` y `pull`.



Workflow Básico

Los comandos esenciales para trabajar con Git

El Commit: La unidad básica de Git

¿Qué es un commit?

Un **commit** es una "foto" del estado del proyecto en un momento dado. Cada commit tiene: un identificador único (hash), una descripción (mensaje), el autor y la fecha.

Buenas prácticas de commits

```
# Mensajes claros y descriptivos
git commit -m "Agrega funcion de calculo de VAN"
git commit -m "Corrige error en importacion de datos"
git commit -m "Actualiza documentacion del modelo"

# Ver diferencias antes de commitear
git diff
git diff --staged
```

Reglas de oro del commit

- Mensajes en presente o imperativo.
- Un commit = un cambio lógico.
- Commitear seguido, no acumular.
- Nunca commitear archivos temporales o contraseñas.

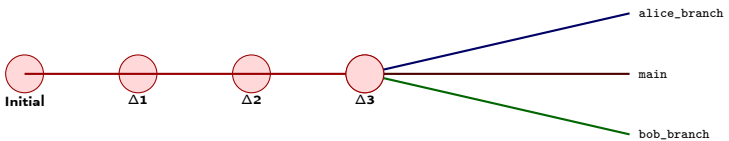
Branches (Ramas)

Trabajar en paralelo sin romper lo que ya funciona

¿Qué es una rama (branch)?

Definición

Una **rama** es una línea de desarrollo independiente. Permite trabajar en nuevas funcionalidades o correcciones sin afectar la versión estable del proyecto (`main` o `master`).



Alice y Bob trabajan cada uno en su rama de forma independiente, sin interferir entre sí.

El flujo colaborativo con branches

Cómo Alice y Bob trabajan juntos

1. **Clonan** el repositorio remoto a sus máquinas locales.
2. Cada uno **crea su propia rama**: `alice_branch`, `bob_branch`.
3. Trabajan de forma independiente, haciendo commits en sus ramas.
4. Al terminar, hacen **push** de su rama al repositorio remoto.
5. Abren un **Pull Request** para que el equipo revise antes de integrar a `main`.
6. Tras la revisión, el código se **mergea** a main con todas las funcionalidades.

Resultado

La rama `main` ahora contiene todos los commits de Alice y Bob, de forma ordenada y revisada. Es una forma más organizada de añadir funcionalidades al proyecto.

GitHub

La plataforma de colaboración para proyectos Git

¿Qué es GitHub?

Definición

GitHub es un servicio de hosting para repositorios Git, accesible en `github.com`. Propiedad de Microsoft, ofrece una capa de colaboración y gestión por encima de Git.

Git $\neq$ GitHub

- **Git:** herramienta de línea de comandos (local).
- **GitHub:** plataforma web que usa Git por debajo.

Alternativas a GitHub:

- GitLab
- Bitbucket
- Azure DevOps

¿Por qué GitHub es el estándar?

- Plan gratuito muy completo.
- Enorme comunidad open source.
- Interfaz visual amigable.
- Integración con CI/CD y otras herramientas.
- Control de acceso y roles.
- Visualización de branches y cambios.

Funcionalidades clave de GitHub

Colaboración

- **Pull Requests:** revisión de código antes de integrar.
- **Issues:** gestión de tareas y bugs.
- **Code Review:** comentarios línea por línea.
- **Roles y permisos:** quién puede hacer qué.

Visibilidad

- **Edición online:** modificar archivos desde el navegador.
- **Visualización de branches:** ver el árbol de ramas.
- **Historial visual:** quién cambió qué y cuándo.

Automatización

- **GitHub Actions:** CI/CD integrado (tests automáticos, deploys).
- **Webhooks:** disparar procesos externos ante eventos.
- **Integración** con herramientas externas (Jira, Slack, etc.).

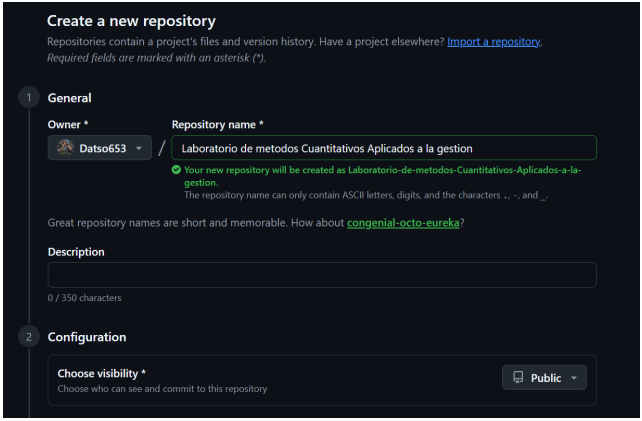
Dato clave

GitHub incluye opción **gratuita** muy potente y planes pagos con funcionalidades adicionales para organizaciones.

Tutorial paso a paso

Crear un repositorio, invitar colaboradores y hacer un commit

Paso 1: Crear un nuevo repositorio



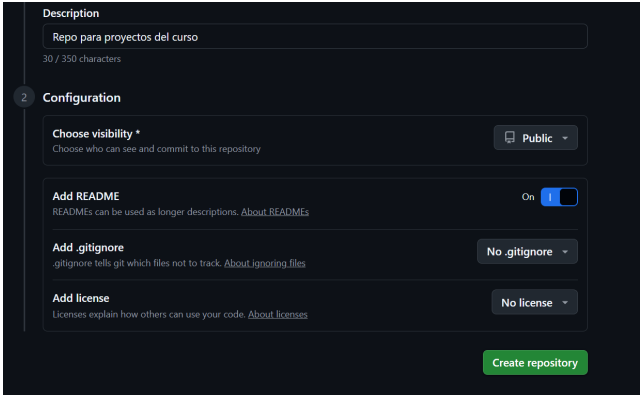
¿Cómo llegar acá?

Desde `github.com`, hacer clic en el botón "+" (arriba a la derecha) y seleccionar **New repository**.

Completar los datos

- **Owner:** tu usuario de GitHub.
- **Repository name:** nombre descriptivo del proyecto.
- GitHub genera automáticamente el nombre con guiones.

Paso 2: Configuración del repositorio



Opciones a configurar

- **Description:** breve descripción del proyecto (opcional pero recomendado).
- **Visibility:** `Public` (visible para todos) o `Private`.
- **Add README:** activar para tener un archivo inicial.

Recomendación

Para proyectos de la materia, usar **Public** y activar el **README**. Luego hacer clic en **Create repository**.

Paso 3: Repositorio creado

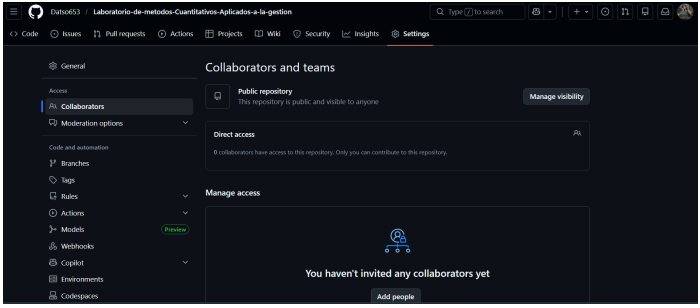
¿Qué vemos?

- El archivo `README.md` creado

Datos del repo

- **1 Branch**, **0 Tags**, **1 Commit**.

Paso 4: Invitar colaboradores (Settings)



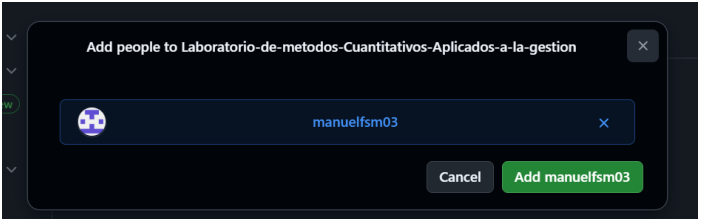
¿Cómo llegar?

1. Ir a la pestaña **Settings** del repositorio.
2. En el menú lateral, seleccionar **Collaborators**.
3. Hacer clic en **Add people**.

Importante

Solo el **dueño** del repositorio puede invitar colaboradores. Los repos **Public** son visibles para todos, pero solo los colaboradores invitados pueden hacer push.

Paso 5: Agregar al compañero



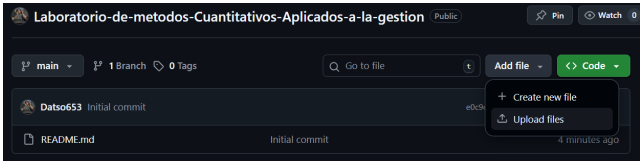
Proceso

1. Escribir el **nombre de usuario** de GitHub del compañero en el buscador.
2. Seleccionarlo de la lista.
3. Hacer clic en **Add [usuario]**.

¿Qué pasa después?

El compañero recibe una **invitación por email** que debe aceptar. Una vez aceptada, puede hacer `push`, `pull` y trabajar directamente en el repositorio.

Paso 6: Subir archivos al repositorio



Desde la interfaz web

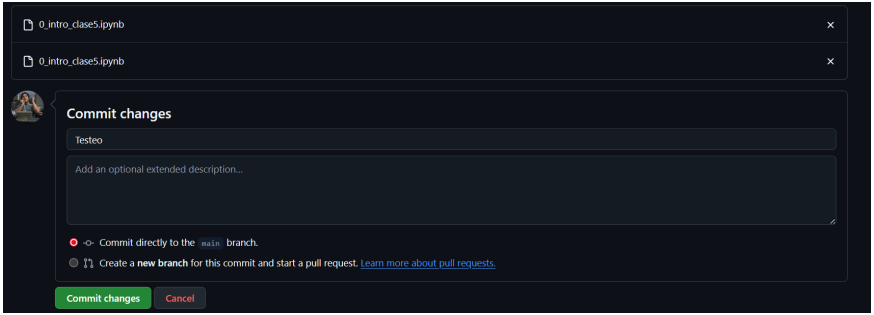
1. En la página principal del repo, hacer clic en **Add file**.
2. Seleccionar **Upload files**.
3. Arrastrar o seleccionar los archivos a subir.

Alternativa por terminal

También se puede hacer con comandos Git:

```
git add archivo.ipynb
git commit -m "Agrega notebook"
git push origin main
```

Paso 7: Hacer un commit (confirmar cambios)



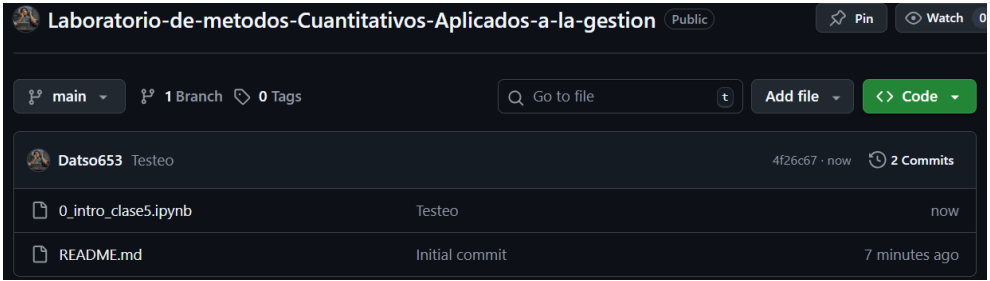
Completar el commit

- Escribir un **mensaje descriptivo** del cambio (ej: "Agrega notebook clase 5").
- Opcionalmente, agregar una descripción extendida.

Hacer clic en Commit changes

Al confirmar, GitHub genera un nuevo commit con un hash único. Los archivos quedan registrados en el historial del proyecto con autor, fecha y mensaje.

Paso 8: Resultado final



¿Qué cambió?

- El repositorio ahora tiene **2 Commits**.
- El archivo `0_intro_clase5.ipynb` aparece con su mensaje de commit.
- El `README.md` sigue intacto.

Resumen del flujo

Crear repo → Configurar → Invitar colaboradores → Subir archivos → Commit → El proyecto está versionado y compartido.

Tabla comparativa: Git vs GitHub

Aspecto	Git	GitHub
Tipo	Herramienta CLI	Plataforma web
Dónde vive	Tu computadora	Nube (servidor)
Requiere internet	No	Sí (para sincronizar)
Colaboración	Limitada	Nativa (PRs, Issues)
Interfaz	Terminal	Visual + Terminal
Costo	Gratuito	Freemium
Dueño	Comunidad (Linux)	Microsoft

Conclusión clave

Git es el motor, GitHub es la carrocería. Podés usar Git sin GitHub, pero GitHub sin Git no existe.

¿Preguntas?

No hay preguntas malas,
solo branches que aún no mergeamos.

Fuentes y Bibliografía

- Pro Git Book — Scott Chacon & Ben Straub (`git-scm.com/book`)
- GitHub Official Documentation — `docs.github.com`
- Atlassian Git Tutorials — `atlassian.com/git/tutorials`
- Git Reference Manual — `git-scm.com/docs`
- The Missing Semester of Your CS Education — MIT (`missing.csail.mit.edu`)

Las imágenes y diagramas son de elaboración propia con fines educativos.