Introducción a Git y GitHub

Git y GitHub (GIT) - 2026

Introducción a Git y GitHub

1. El caos sin control de versiones

Imaginen que están escribiendo un ensayo en grupo por correo electrónico. Cada quien edita el Word y lo reenvía:

- `ensayo_final.docx`
- `ensayo_final_v2.docx`
- `ensayo_final_v2_JUAN.docx`
- `ensayo_final_v2_JUAN_correcciones_MARIA.docx`

Preguntas para la clase:

- ¿Cuál es la versión buena?
- ¿Qué pasa si dos personas editan el mismo párrafo a la vez?
- ¿Cómo saben quién cambió qué y por qué?
- ¿Cómo revierten un cambio si alguien se equivocó hace tres días?

Conclusión: Git resuelve esto con un historial versionado, cambios rastreables, y la capacidad de trabajar en paralelo sin interferir entre todos.

---

2. ¿Qué es Git?

Git es un sistema de control de versiones distribuido. Permite:

- Guardar el historial completo de cambios de un proyecto.
- Trabajar en paralelo sin sobrescribir el trabajo de otros.
- Volver a cualquier punto anterior del proyecto.
- Trabajar sin conexión (todo el historial vive en tu máquina).

Los tres estados de un archivo

```
Working Directory  --git add-->  Staging Area  --git commit-->  Repositorio (historial)
   (tus archivos)                 (lo que vas                   (snapshot permanente)
                                    a guardar)
```

- **Working directory:** donde editas tus archivos normalmente.
- **Staging area:** una "caja" donde eliges qué cambios van a formar parte del próximo commit.
- **Repositorio:** el historial de commits, guardado localmente.

Conceptos clave

- **Repositorio (repo):** la carpeta con memoria — contiene tu proyecto y todo su historial.
- **Commit:** una fotografía (snapshot) del proyecto en un momento dado, con un mensaje que explica el cambio.

---

3. Comandos básicos (demo en vivo)

| Comando | Analogía | Qué hace | |---|---|---| | `git clone <url>` | Sacar una copia de la biblioteca a tu casa | Descarga un repositorio remoto completo, con su historial | | `git add <archivo>` | Meter documentos en una caja para enviar | Mueve cambios del working directory al staging area | | `git commit -m "mensaje"` | Sellar la caja con una nota de qué contiene | Guarda un snapshot permanente en el historial local | | `git push` | Enviar la caja a la bodega central | Sube tus commits locales al repositorio remoto | | `git pull` | Traer las cajas nuevas de la bodega | Descarga y fusiona los cambios del remoto a tu copia local |

Guion de la demo

1. `git clone <repo-vacío>` — mostrar que se crea la carpeta local.
2. Crear un archivo nuevo (ej. `notas.txt`).
3. `git status` — mostrar el archivo como "untracked".
4. `git add notas.txt` — mostrar el cambio a "staged".
5. `git commit -m "Agrega notas de la clase"`.
6. `git push` — mostrar el commit apareciendo en GitHub.
7. Editar el archivo directamente en GitHub (simulando a "otra persona").
8. `git pull` en la máquina local — mostrar que el cambio remoto llega.

Nota: este es el mejor momento para reforzar que `git status` es el comando que van a usar todo el tiempo para saber "¿en qué estado estoy?".

---

4. Trabajo en paralelo: branches, merge, fetch y fork

Branches (ramas)

Analogía: líneas de tiempo paralelas. Puedes experimentar en una rama sin afectar la rama principal (`main`).

- `git branch <nombre>` — crea una rama nueva.
- `git checkout -b <nombre>` — crea y cambia a la rama en un solo paso.
- `git checkout main` — vuelve a la rama principal.

Ejemplo de uso: crear una rama por cada nueva funcionalidad (`feature/login` , `feature/reporte-ventas`) para no romper el código que ya funciona en `main` .

Merge

Unir dos líneas de tiempo en una. Cuando terminas el trabajo en una rama, la fusionas de vuelta a `main` :

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

- Si los cambios no chocan, Git los combina automáticamente.
- Si dos ramas modificaron la misma línea del mismo archivo, ocurre un conflicto de merge y Git pide que la persona decida manualmente qué versión dejar. (Mención breve — no es el foco de esta clase introductoria.)

Fetch vs Pull

- `git fetch` — trae los cambios del remoto pero no los aplica todavía. Sirve para "mirar antes de mezclar".
- `git pull` = `git fetch` + `git merge` . Trae los cambios y los combina de una vez con tu rama actual.

Fork

Analogía: fotocopiar el libro completo para tenerlo en tu propia biblioteca, con la opción de proponerle cambios al autor original.

- Un fork crea una copia completa del repositorio en tu propia cuenta de GitHub.
- Diferencia clave con branch:
  - **Branch:** ocurre dentro del mismo repositorio.
  - **Fork:** ocurre a nivel de repositorio completo, usualmente en otra cuenta.
- Se usa típicamente cuando no tienes permisos de escritura sobre el repositorio original (ej. proyectos open source).

---

5. GitHub

Git ≠ GitHub. Git es la herramienta; GitHub es una plataforma/nube que la hospeda y le agrega colaboración.

- **Pull Request (PR):** la manera de proponer que tus cambios (de una rama o de un fork) se incorporen a otra rama. Permite revisión de código antes de hacer merge.
- **Issues:** para reportar bugs o proponer tareas.
- **README:** la carta de presentación del proyecto.
- **Actions:** automatización (CI/CD) — mención breve, sin entrar en detalle.
- **Permisos de colaboración:** Read / Write / Maintain / Admin — quién puede hacer qué dentro de un repo.

---

6. Cierre

Flujo típico resumido

```
fork o clone → branch → add/commit → push → pull request → merge
```

Para practicar

- github.com/skills — ejercicios interactivos oficiales de GitHub.

Preguntas de repaso sugeridas

1. ¿Cuál es la diferencia entre `git fetch` y `git pull`?
2. ¿Cuándo usarían un fork en vez de un branch?
3. ¿Qué comando usarían para guardar un cambio de forma permanente en el historial local?