1. Introducción a Git y GitHub

Git es un sistema de control de versiones distribuido que te permite llevar un historial de cambios de tu código. GitHub es una plataforma que hospeda repositorios Git de forma remota, facilita la colaboración y ofrece tools (pull requests, issues, actions...).

---

2. Flujo básico de trabajo

1. **Clonar un repositorio remoto**

```bash
git clone https://github.com/usuario/proyecto.git
```

Crea una copia local ligada al remoto.

2. **Estado de tu copia**

```bash
git status
```

Te dice archivos modificados, sin seguimiento o listos para commit.

3. **Añadir cambios al área de staging**

```bash
git add archivo1.java carpeta/archivo2.txt
```

   - git add . añade *todo* lo modificado.
   - **Peligro**: si incluyes archivos no deseados (credenciales, binarios), los subirás a tu historial.

4. **Registrar los cambios en el repositorio local**

```bash
git commit -m "Mensaje claro: qué y por qué"
```

Muy importante que sean Mensajes breves pero descriptivos que ayuden a entender el historial.

- **Peligro**: commits demasiado grandes o con mensajes genéricos dificultan la revisión.
-

5. **Enviar commits al repositorio remoto**

```bash
git push origin main
```

Aquí actualizaras la rama de tu repositorio remoto con tus cambios, origin es el nombre por defecto del remoto, main (o master) la rama, usualmente en los repositorios de la universidad se les llama main.

- **Peligro**: si tu local está desactualizado, el push fallará y deberás hacer un pull y resolver conflictos antes de volver a intentar.

---

3. Gestión de ramas (branches)

- **Ver ramas locales y remotas**

```bash
git branch          # locales
git branch -r       # remotas
git branch -a       # todas
```

Crear una rama nueva

```bash
git branch nombre-rama
```

- **Cambiarse a otra rama (checkout)**

```bash
git checkout nombre-rama
```

ó, en Git ≥ 2.23:

```bash
git switch nombre-rama
```

Crear y cambiarse en un solo comando

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

ó:

```bash
git switch -c feature/login
```

- **Sincronizar tu rama con el remoto**

```bash
git fetch origin
git merge origin/nombre-rama
```

ó, de forma abreviada:

```bash
git pull
```

git pull hace fetch + merge

.
Peligro: si has modificado archivos y cambias de rama sin *stash (Guarda temporalmente cambios)* ni commit, se pueden perder o quedar mezclados.

- **¿Qué pasa si tu rama local está muy desfasada?**
    - Guarda tus cambios locales sin commit:

```bash
git stash
```

    - Actualiza la rama:

```bash
git pull
```

Recupera tus cambios:

```bash
git stash pop
```

- Esto evita sobreescribir tu trabajo al hacer git pull.

---

4. Fusionar cambios (merge)

1. **Desde línea de comandos**

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

Si no hay conflictos, Git crea un commit de merge.

   - **Peligro**: puede generar conflictos si ambos modificaron las mismas líneas.

2. **Conflictos de merge**
   - Git marca los archivos en conflicto con <<<<<<<, =======, >>>>>>>.
   - Edita manualmente con tu editor de código, decide qué conservar y luego:

```bash
git add archivo-en-conflicto
git commit   # termina el merge
```

3. **Merge en GitHub (Pull Request)**
   - Crea un Pull Request desde feature/login (o desde cualquier rama a la que quieras implementar) de hacia main.
   - Revisa “diffs”, deja comentarios y, al aprobarse, haz **Merge** mediante la interfaz.
   - **Peligro**: si alguien fuerza un *push* directo a main (git push --force), puedes borrar commits de otros y corromper el historial.

---

5. Actualizaciones forzadas y “peligros”

- **git pull --rebase**
    - Reaplica tus commits encima de la última versión remota, dejando el historial más lineal.
- **git reset --hard origin/main**
    - Sincroniza tu rama local con la remota, **descartando** todos los cambios no comiteados de tu repositorio local.
    - **Muy peligroso** si no guardaste tu trabajo: perderás lo no commiteado para siempre.
- **git push --force / --force-with-lease**
    - Reemplaza la historia remota.
    - **Extremadamente peligroso**: si otros han trabajado en esa rama, sus commits desaparecen; usa --force-with-lease para minimizar riesgos.

---

6. Buenas prácticas finales

- **Commits pequeños y frecuentes** ayudan a aislar cambios y facilitan rollbacks.
- **Ramas temáticas** (feature/, bugfix/) mantienen tu main limpio.
- **Revisiones de código** mediante Pull Requests garantizan calidad.
- **README y CONTRIBUTING**: incluye instrucciones para clonar, compilar y contribuir.
- **Git ignore**: mantén un .gitignore actualizado para no subir binarios, credenciales ni artefactos de compilación.