Actividad guiada

Enunciado del reto

Como parte de un proyecto colaborativo, necesitas **inicializar un repositorio Git**, crear tu primer archivo, hacer un **commit inicial** y luego **gestionar ramas** para organizar tu trabajo. En 30 minutos, practicarás los comandos básicos de Git para simular un flujo de trabajo profesional.

Objetivo(s) de la actividad

- **Inicializar** un repositorio en local y crear el primer commit.
- **Gestionar** archivos con `add`, `commit`, `status` e ignorar archivos con `.gitignore`.
- **Trabajar** con ramas: crear, cambiar, fusionar y manejar conflictos básicos.
- **Comprender** la importancia de GitHub como repositorio remoto y trabajo colaborativo.

Materiales y preparación previa (checklist)

- Git instalado en tu máquina (Windows, Mac o Linux).
- Editor de texto (VS Code, Sublime o similar).
- Carpeta de proyecto vacía.
- Acceso a terminal o Git Bash.

Requisitos técnicos y datos

- Uso de comandos Git en terminal.
- No requiere conexión a internet (flujo local).
- Archivos de texto simples como prueba (`notas.txt`).

Instrucciones paso a paso con micro-checks

Tiempo total: 30 minutos

1. **Inicialización del repositorio (5’)**
    a. Abre una carpeta nueva `mi_proyecto`.
    b. Ejecuta en terminal:

```
git init
```

Micro-check: aparece la carpeta oculta `.git`.

2. **Creación de archivo y primer commit (10’)**
    a. Crea `notas.txt` y escribe:

```
Este es mi primer archivo con Git.
```

    b. Agrega y confirma:

```
git add notas.txt
git commit -m "Primer commit con archivo notas.txt"
```

    c. Verifica estado:

```
git status
```

Micro-check: el archivo aparece como “tracked” y commit registrado.

3. **Gestión de ramas (10’)**
    a. Crea una rama y cámbiate a ella:

```
git branch nueva-funcionalidad
git switch nueva-funcionalidad
```

    b. Edita `notas.txt` agregando una nueva línea:

```
Estoy trabajando en la rama nueva-funcionalidad.
```

    c. Haz commit:

```
git add notas.txt
git commit -m "Actualización en rama nueva-funcionalidad"
```

    d. Vuelve a `main` y fusiona:

```
git switch main
git merge nueva-funcionalidad
```

Micro-check: revisa que `notas.txt` contiene ambas líneas.

4. **Ignorar archivos (5')**
    a. Crea `.gitignore` con:

```
*.log
temp/
```

    b. Crea archivo `debug.log` y carpeta `temp/` y confirma que no aparecen en `git status`.

Criterios de éxito y errores comunes

Éxito:

- Repositorio inicializado con `.git`.
- Commit inicial registrado.
- Ramas creadas, editadas y fusionadas.
- `.gitignore` funcionando.

Errores comunes:

- Olvidar `git add` antes de `git commit`.
- No cambiar a la nueva rama antes de editar.
- Confundir `git switch` con `git checkout`.

Tiempo total: 30'

Plan de contingencia:

- Si `git switch` no funciona, usar:

```
git checkout -b nueva-funcionalidad
```

- Si no puedes crear `.gitignore`, ignora manualmente esos archivos en la revisión.

Extensión opcional (+10'):

- Crear un repositorio en GitHub y subir tu proyecto con:

```
git remote add origin <URL>
git push -u origin main
```

SOLUCIÓN PROPUESTA

```
# Paso 1: Inicialización
mkdir mi_proyecto
cd mi_proyecto
git init

# Paso 2: Crear archivo y primer commit
echo "Este es mi primer archivo con Git." > notas.txt
git add notas.txt
git commit -m "Primer commit con archivo notas.txt"
git status

# Paso 3: Gestión de ramas
git branch nueva-funcionalidad
git switch nueva-funcionalidad
echo "Estoy trabajando en la rama nueva-funcionalidad." >> notas.txt
git add notas.txt
git commit -m "Actualización en rama nueva-funcionalidad"
git switch main
git merge nueva-funcionalidad

# Paso 4: Ignorar archivos
echo "*.log" > .gitignore
echo "temp/" >> .gitignore
mkdir temp
touch debug.log
git status  # Verifica que .log y temp/ están ignorados
```