Git y GitHub: El Flujo de Trabajo

Conectando tu PC con la Nube

Matemáticas Computacionales

Universidad Industrial de Santander

12 de febrero de 2026

Paso 0: Configuración de Identidad

Solo se hace una vez en la vida

Antes de guardar nada, Git necesita saber **quién** eres para firmar tu trabajo.

Abre la terminal (Git Bash) y ejecuta:

```
# 1. Tu nombre (Saldr  en el historial)
$ git config --global user.name "Pepito Perez"

# 2. Tu correo (EL MISMO que usaste en GitHub)
$ git config --global user.email "pepito@correo.edu.co"

# 3. Definir la rama principal como 'main' (Est ndar moderno)
$ git config --global init.defaultBranch main
```

Paso 1: Nacer (git init)

Para que una carpeta deje de ser "normal" se convierta en un **Repositorio** con superpoderes (historial), debemos inicializarla.

Ejercicio Rápido:

```
$ cd ~/Desktop              # Ir al escritorio
$ mkdir ProyectoClase2      # Crear carpeta
$ cd ProyectoClase2         # Entrar
$ git init                  #   MAGIA !
```

Paso 1: Nacer (git init)

Para que una carpeta deje de ser "normalz se convierta en un **Repositorio** con superpoderes (historial), debemos inicializarla.

Ejercicio Rápido:

```
$ cd ~/Desktop              # Ir al escritorio
$ mkdir ProyectoClase2      # Crear carpeta
$ cd ProyectoClase2         # Entrar
$ git init                  #   MAGIA !
```

Si te sale *"Initialized empty Git repository"*, ya tienes el sistema de vigilancia activado en esa carpeta.

El Ciclo de Vida: Add, Commit, Push

Imaginen que van a tomar una foto grupal:

1. Working Directory
(El Escenario)
Aquí trabajas, editas y guardas archivos en tu editor.

2. Staging Area
(La Tarima)
Subes al escenario a quienes van a salir en la foto. `git add`

3. Repository
(La Foto Tomada)
Tomas la foto y la guardas en el álbum. `git commit`

Comandos Locales: Guardar Cambios

Vamos a crear un archivo y guardarlo en el historial.

```
# 1. Crear algo
$ touch formulas.py

# 2. Verificar estado (¿Qué ha pasado?)
$ git status
> (Saldrá en ROJO: Untracked files)

# 3. Preparar la foto (Staging)
$ git add formulas.py
> (Si das 'git status' ahora saldrá en VERDE)

# 4. Tomar la foto (Commit)
$ git commit -m "Mi primer commit matemático"
```

Paso Crítico: Crear el Repositorio en la Nube

Ahora queremos subir esto a Internet.

Instrucciones (Ir al navegador):

1. Entra a **github.com** y loguéate.
2. Click en el botón **New** (verde) o el símbolo +.
3. **Repository name:** `ProyectoClase2`
4. Click en **Create repository**.

Escenario A: Conectar MI carpeta a MI GitHub

Situación: Tienes la carpeta `MathComp2026` en tu PC con tareas y quieres subirla a tu cuenta nueva.

1. Ve a GitHub, crea un repo **VACÍO** y copia la URL (HTTPS).
2. En tu terminal, asegúrate de estar **DENTRO** de la carpeta.

```
# 1. Iniciar el rastreo (Si no lo hiciste la clase pasada)
$ git init

# 2. Agregar TODO el contenido (El punto es clave)
$ git add .

# 3. Guardar la versi n en el historial
$ git commit -m "Subiendo mi tarea de la Clase 1"

# 4. Crear el puente con la nube (Pega TU link aqu )
$ git remote add origin https://github.com/TU_USUARIO/MathComp2026.
   git
```

Escenario B: Conectarse al Repo de la Clase

Situación: Quieres descargar los apuntes o el código que hizo el profesor.

¡Regla de Oro!

NUNCA clones un repositorio dentro de otro repositorio. Sal al Escritorio o a Documentos antes de ejecutar esto.

Pasos:

1. Consigue el Link del profesor (Ej: `.../Clase_Profesor.git`)

1. Ejecuta en tu terminal:

```
# 1. Salir de tu carpeta actual
$ cd ~/Desktop

# 2. Descargar una copia exacta de la nube
$ git clone https://github.com/Yazmin246/Matem-ticas-computacionales
   -2026-1.git
```

El Salto a la Nube: Push

Finalmente, enviamos nuestros commits locales a GitHub.

```
$ git push -u origin main
```

Momentos de Tensión

- Si es la primera vez, saldrá una ventana emergente (Git Credential Manager).
- Dale **"Sign in with your browser"**.
- Autoriza el acceso.

Si sale Success... **¡Refresca la página de GitHub!**

Resumen: El Mantra Diario

Cada vez que trabajen en clase, repetirán esto:

1. **Modificar:** Editar código.
2. **Verificar:** `$ git status`
3. **Preparar:** `$ git add .` (El punto agrega TODO).
4. **Guardar:** `$ git commit -m "Mensaje descriptivo"`
5. **Subir:** `$ git push`

Ramas (Branches): Universos Paralelos

Una **Rama** es una línea de tiempo alternativa. Te permite probar ideas sin dañar el código principal (`main`).

Comandos Esenciales:

```
# 1. Ver ramas (La que tiene * es
   la actual)
$ git branch

# 2. Crear una rama nueva
$ git branch experimental

# 3. Moverse a esa rama (Cambiar
   universo)
$ git checkout experimental
# O la versi n moderna:
$ git switch experimental
```

Git Log Gráfico

Para ver cómo se bifurca la historia: `$ git log --oneline --graph --all`

Fusionar (Merge) y Clonar

Una vez tu experimento funciona, debes unirlo a la historia principal.

```
# Paso 1: Volver a la rama principal (OBLIGATORIO)
$ git checkout main

# Paso 2: Fusionar la rama experimental aqu
$ git merge experimental
```

Nota sobre `git clone`**:** Si quieres descargar un repo para empezar a crear ramas desde cero:

```
$ git clone https://github.com/Usuario/Repo.git
```

Deshacer: ¿Borrar historia o Corregirla?

A. Git Revert (Seguro) Crea un *nuevo commit* que invierte lo que hizo el anterior. No borra historia. Ideal para equipos.

```
$ git revert <hash_commit>
```

1. Git Tag (Etiquetas): Poner nombre a un commit importante (v1.0). v1 *gitshowv*1,0*Verdetallesdeltag* git push –tags Subir las etiquetas a la nube

2. Checkout al Pasado (Time Travel) Puedes ir a cualquier punto de la historia para ver cómo era el código.

```
$ git checkout <hash_del_commit>
```

Advertencia: Entrarás en modo "Detached HEAD". Puedes mirar, pero si haces cambios, se perderán a menos que crees una rama nueva.

Conflictos de Fusión

Ocurre cuando dos ramas modifican **la misma línea** de forma distinta. Git entra en pánico y te pide ayuda.

Pasos para resolverlo:

1. Git te dirá: *CONFLICT (content): Merge conflict in archivo.py*.
2. Abre el archivo. Verás esto:

```
<<<<<<< HEAD
print("Hola Mundo")
=======
print("Hello World")
>>>>>>> experimental
```

3. **Decide:** Borra las marcas ($<<, ==, >>$) y deja el código correcto.
4. **Finaliza:**

```
$ git add archivo.py
$ git commit -m "Conflicto resuelto"
```

Reto Integrador: .El Multiverso"

Objetivo: Crear una realidad alterna, dañarla y arreglarla.

1. Crea una rama llamada `features`: `$ git branch features`
2. Cámbiate a ella: `$ git checkout features`
3. Crea `error.txt`, agrégalo y haz commit.
4. **Prueba de fuego:** Borra ese commit destructivamente: `$ git reset --hard HEAD~1` (El archivo desaparecerá).
5. Crea `acierto.txt`, haz commit.
6. Vuelve al inicio: `$ git checkout main`
7. Fusiona tu trabajo: `$ git merge features`
8. Etiquétalo: `$ git tag v1.0`
9. Verifica tu árbol: `$ git log --oneline --graph`

Resultado

Debes ver el grafo con la bifurcación y unión, y tener el archivo `acierto.txt` en tu rama main.

Gran Reto Final: El Proyecto "Phoenix"

Objetivo: Simular el ciclo de vida completo de una funcionalidad (Feature Branch Workflow).

Instrucciones paso a paso:

1. **Preparación:** Asegúrate de estar en la rama `main` y actualizado (`git pull`).
2. **Ramificación:** Crea y salta a la rama `experimento`.
3. **Trabajo:** Crea el archivo `analisis.py` con un código simple (ej. `print("Hola")`). Haz **commit**.
4. **El Error:** Modifica el archivo agregando ."ERROR CRÍTICO", guarda y haz un **segundo commit** (simulando una metida de pata).
5. **Corrección (Reset):** Usa `git reset --hard HEAD~1` para eliminar ese último commit y volver al estado limpio.
6. **Unión:** Regresa a `main` y fusiona tu rama `experimento`.
7. **Lanzamiento:** Crea una etiqueta: `git tag v1.0`.
8. **Despliegue:** Sube todo a la nube: `git push origin main --tags`.