«Talento Tech»

# Front-End JS

Clase 08

Buenos Aires Ciudad

# Clase 08 | Git & Github

**Temario:**

1. **Git y GitHub**
    - ¿Qué es Git?
    - ¿Qué es GitHub?
    - Instalación y Configuración de Git
    - Conceptos Clave en Git
    - Comandos Básicos de Git
    - Crear ramas y trabajar con ellas
    - Flujo de Trabajo con Git y GitHub
2. **Ejercicios Prácticos**

## Objetivo de la clase:

En esta clase vamos a introducirnos en el uso de **Git** y **GitHub**, dos herramientas fundamentales para el control de versiones y el trabajo colaborativo en proyectos de desarrollo. Vamos a conocer qué es Git y qué es GitHub, cómo instalar Git y configurarlo correctamente en nuestra computadora.

Exploraremos los **conceptos clave** como repositorio, rama (*branch*), *commit* y *merge*, y aprenderemos los **comandos básicos** para crear un repositorio, registrar cambios, y sincronizarlos con GitHub. Además, practicaremos cómo crear ramas, hacer cambios en ellas, fusionarlas con la rama principal, y cómo resolver conflictos en el proceso.

Para cerrar, veremos cómo crear un repositorio en GitHub, vincularlo con nuestro proyecto local y mantenerlos sincronizados. Con esta base, vas a poder organizar mejor tu trabajo, llevar registro de tus avances y colaborar con otras personas de forma eficiente y profesional.

# Git y GitHub

**Git** es un sistema de control de versiones. En pocas palabras: es una herramienta que te permite ir guardando versiones de tu proyecto. Si alguna vez rompiste todo el código y deseaste volver a una versión anterior, ¡Git es la solución! Gracias a esta herramienta podés "viajar en el tiempo" y recuperar una versión anterior del proyecto. Y no solo eso, también te permite trabajar en equipo sin “pisarte” con otras personas que desarrollen el mismo proyecto.

**GitHub** es, de alguna forma, “el hermano” de Git, pero con una diferencia: opera en la nube. Es una plataforma donde podés subir tu código y compartirlo con otras personas desarrolladoras o con el mundo. En este espacio virtual podés colaborar en proyectos, subir cambios y ver el progreso que va teniendo tu proyecto a lo largo del tiempo.

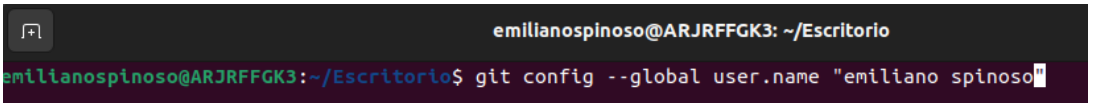
GitHub es básicamente un lugar donde guardás tus repositorios (que son como carpetas de proyectos con historial) para poder trabajar con más gente o desde cualquier lugar.

## Instalación y Configuración de Git

- **Windows**: Descargá el instalador desde Git SCM y seguí las instrucciones.
- **Mac**: Si tenés **Homebrew**, podés instalarlo directamente desde la terminal con:
  `brew install git`
- **Linux**: Para distribuciones basadas en Debian, como **Ubuntu**, escribí en la terminal:
  `sudo apt-get install git`

### Configuración

Lo primero que necesitás después de instalar Git es configurarlo. Esto lo hacés desde la **terminal o consola.** ¡Veamos cómo!




```
git config --global user.name "Tu Nombre"
git config --global user.email "tuemail@ejemplo.com"
```

Esto le dice a Git cuál es tu nombre y tu email para que puedas hacer *commits* (que son como "guardados" o "versiones").

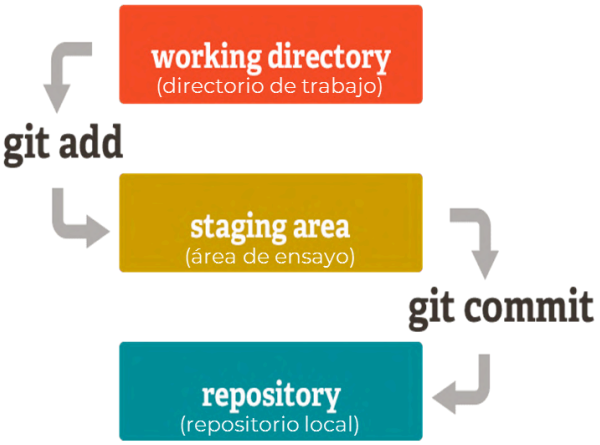
## Conceptos Clave en Git

- **Repositorio (o “Repo”)**: Se trata del espacio donde va a estar guardado todo tu proyecto, junto con su historial de cambios. Puede ser un repositorio local (almacenado en tu computadora) o remoto (en GitHub).
- **Branch (o “Rama”)**: Imaginá que querés probar algo nuevo en tu código, pero sin romper lo que ya funciona. Para eso hacés una *"rama"*. Podés trabajar tranquilamente y, si te gusta el resultado, lo unís a la rama principal (generalmente llamada `main` o `master`).
- **Add**:Cuando modificás archivos, Git no los guarda automáticamente. Primero tenés que **agregar los cambios al área de preparación** (staging area) usando `git add`. Es como decirle a Git: “estos archivos están listos para guardar”.
- **Commit**: Un **commit** guardará el estado actual de tu proyecto. Cada vez que hacés un commit de alguna forma estás “sacando una foto” de tu proyecto, pero más importante: estás guardando los cambios hechos hasta ese momento.
- **Git Status:** Este comando (`git status`) te muestra **el estado actual de tu proyecto**. Te dice qué archivos cambiaste, cuáles están listos para hacer commit y cuáles no. Es como una “foto informativa” para saber en qué punto estás.
- **Push**:Una vez que hiciste los commits, `git push` se usa para **enviar esos cambios al repositorio remoto**, como GitHub. Es el paso que sube tu trabajo local a la nube para compartirlo o respaldarlo.
- **Merge (o “fusión”)**: Es el proceso de unir dos ramas. Después de probar algo nuevo en una rama, podés unirlo al código principal mediante un **merge**.

# Comandos básicos de Git

**git init**: Inicializa un nuevo repositorio en la carpeta actual. Esto crea un `.git`, que es la carpeta donde se guardan todos los cambios que vas a realizar.

```
git init
```




**git add**: Este comando selecciona los archivos que querés "guardar". Podés agregar archivos específicos o todo de una, con:

```
git add .
```

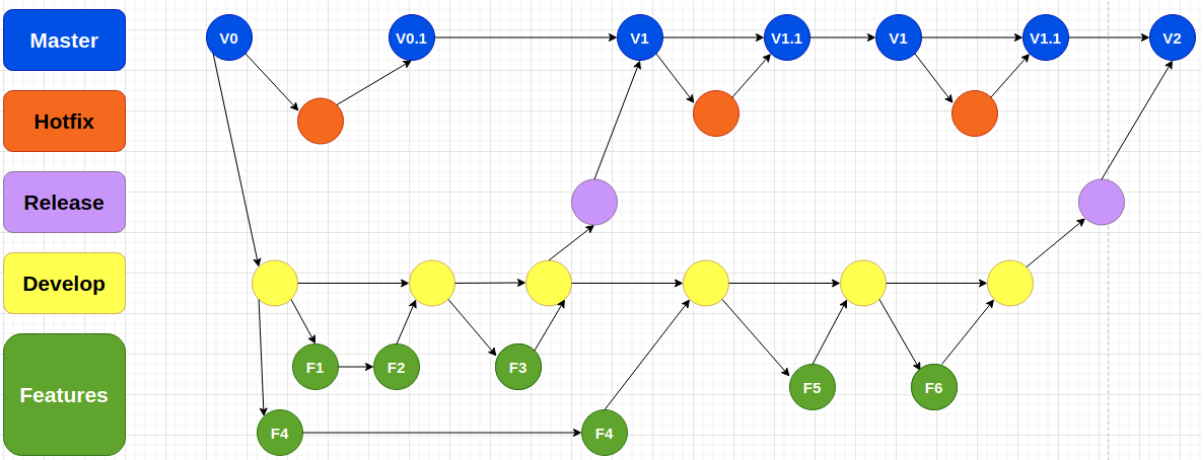
**git commit**: Crea un "snapshot" de los archivos agregados y guarda un mensaje descriptivo de los cambios.

```
git commit -m "Mensaje descriptivo de los cambios"
```

**git push**: Sube los commits de tu repositorio local a uno remoto en GitHub.

```
git push origin main
```

# Crear ramas y trabajar con ellas en Git.




En Git, trabajar con ramas es esencial para mantener un flujo de trabajo organizado, especialmente cuando trabajás en equipo o desarrollás diferentes funcionalidades (features) de un proyecto. Las ramas te permiten experimentar con nuevos cambios o corregir errores sin afectar la rama principal (main o master).

### ¿Qué es una Rama en Git?

Una rama (branch) es una línea paralela de desarrollo que te permite trabajar en nuevas funcionalidades, arreglos o pruebas sin modificar la rama principal. Al final, podés fusionar (merge) los cambios de tu rama con la rama principal si todo funciona correctamente.

Por ejemplo, supongamos que recibís un bug en tu proyecto o querés agregar una nueva funcionalidad. Lo mejor es crear una nueva rama para trabajar en esos cambios, mientras la rama principal sigue funcionando correctamente para otros usuarios o miembros del equipo.

## Comandos Básicos de Ramas

### Ver ramas existentes

```
git branch
```

- Muestra todas las ramas y te indica en cuál estás parado.

### Crear una nueva rama

```
git branch nombreRama
```

Ejemplo:

```
git branch feature-contacto
```

### Cambiar de rama

```
git checkout nombreRama
```

Ejemplo:

```
git checkout feature-contacto
```

- Recomendación: Usá `git checkout -b nombreRama` para **crear y moverte** a una rama en un solo paso.

### Renombrar una rama

```
git branch -m ramaVieja ramaNueva
```

### Eliminar una rama

```
git branch -d nombreRama
```

**¡Ojo!** No podés eliminar una rama si estás parado sobre ella. Primero cambiá a otra rama.

Si necesitás forzar el borrado:

```
git branch -D nombreRama
```

## Trabajar con Archivos en Ramas

Creás una rama y te cambiás a ella:

```
git branch rama1
git checkout rama1
```

Agregás archivos:

```
touch archivo1.txt
      touch archivo2.txt
```

Registrás los cambios:

```
git add .
git commit -m "Creación de archivos"
```

Ahora, si cambiás de rama, vas a ver que cada rama tiene sus propios archivos y cambios.

## Unir Ramas (Merge)

### Reglas básicas antes de hacer merge:

- Parate en la rama que querés actualizar (ej. `main`).
- Mergeá desde la rama con los cambios.

**Ejemplo:**

```
git checkout main
git merge feature-contacto
```

# Revertir un Merge

### Si no hiciste commit:

```
git merge --abort
```

### Si ya hiciste commit:

Buscá el hash del commit con:
```
git log --oneline
```

1. Luego:
   ```
   git revert -m 1 <hash>
   ```
2. `-m 1` indica que se mantiene el primer "padre", normalmente la rama principal.

# Recuperar una Rama Eliminada

### Ver historial oculto:

```
git reflog
```

### Recuperar una rama perdida:

```
git checkout -b nombre <hash>
```

# Reset de Commits

| Comando | Qué hace |
|---|---|
| `git reset --soft <hash>` | Borra commit, mantiene cambios stageados |
| `git reset --mixed <hash>` | Borra commit, mantiene cambios no stageados |
| `git reset --hard <hash>` | Borra commit y también los cambios locales |

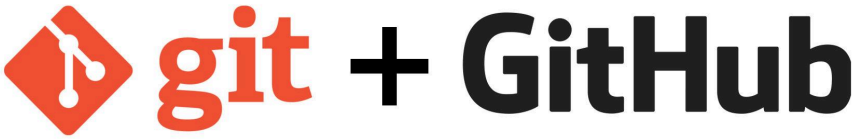
## Git Restore

| Acción | Comando | Resultado |
|---|---|---|
| Deshacer cambios locales | `git restore archivo` | Vuelve al último commit |
| Quitar del staging | `git restore --staged archivo` | Saca archivo del área de staging |
| Traer versión de un commit anterior | `git restore --source <hash> archivo` | Recupera una versión vieja |

## Resumen

| Acción | Comando |
|---|---|
| **Ver ramas** | `git branch` |
| **Crear rama** | `git branch nombre` |
| **Moverse a rama** | `git checkout nombre` |
| **Crear + cambiar** | `git checkout -b nombre` |
| **Eliminar rama** | `git branch -d nombre` |
| **Renombrar rama** | `git branch -m vieja nueva` |
| **Subir rama** | `git push origin nombre` |
| **Unir ramas** | `git merge nombre` |
| **Revertir merge** | `git merge --abort` / `git revert -m 1 <hash>` |
| **Ver historial oculto** | `git reflog` |

# Flujo de trabajo con Git y GitHub.




**Crear un repositorio en GitHub**: Vas a GitHub, creás un nuevo repositorio y le ponés un nombre descriptivo.

**Vincular repositorio local con GitHub**: Abrí tu terminal y escribí este comando para vincular tu repositorio local con el que creaste en GitHub:

```
git remote add origin https://github.com/tu-usuario/proyecto.git
```

**Subir cambios a GitHub**: Después de hacer algunos cambios y hacer commits, podés subirlos a GitHub con el comando:

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

## Resolución de conflictos en Git

Los **conflictos en Git** surgen cuando dos personas (o vos en dos ramas diferentes) modifican las mismas líneas de código. Git no sabe cuál de las dos versiones debe mantener, así que te pide que lo resuelvas manualmente.

## Pasos para resolver un conflicto:

**Identificar el conflicto**:
Después de intentar hacer un merge, si hay un conflicto, Git te lo va a avisar. Podés ver los archivos en conflicto ejecutando:

```
git status
```

Los archivos en conflicto **aparecerán en rojo.**

**Abrir el archivo en conflicto**:
Accedé al archivo en conflicto por medio de tu editor de código preferido. Git va a marcar las áreas en conflicto de esta forma:

```
<<<<<<< HEAD

Código en tu rama actual

=======

Código en la otra rama

>>>>>> nombre-de-la-rama
```

Todo lo que está entre `<<<<<<< HEAD` y `=======` es tu código. Lo que está entre `=======` y `>>>>>>> nombre-de-la-rama` es el código de la otra rama.

**Resolver el conflicto**:
Tenés que elegir qué código querés mantener. Podés optar por una de las versiones o hacer una mezcla de ambas. Una vez que decidas, eliminás las marcas (`<<<<<<<`, `=======`, `>>>>>>>`) y dejás el código como corresponde.

**Agregar el archivo resuelto**:
Una vez resuelto el conflicto, volvés a agregar el archivo al *staging area*:

```
git add nombre-del-archivo
```

**Hacer un commit**:
Luego de agregar el archivo, hacés un commit que indique que resolviste el conflicto:

```
git commit -m "Conflicto resuelto"
```

**Continuar con el flujo**:
Después de resolver el conflicto y hacer el *commit,* podés continuar con el flujo de trabajo normal (push, merge, etc.).

# Usar Git con GitHub Desktop y Visual Studio Code

Aunque podés trabajar con Git desde la terminal, existen herramientas que te harán muchísimo más sencillas tus tareas, como **GitHub Desktop** y **Visual Studio Code**.

**GitHub Desktop** es una aplicación con interfaz gráfica que te permite gestionar tus repositorios de Git sin necesidad de usar la terminal. Es ideal para quienes están empezando con Git o prefieren una experiencia más visual. Recordá que usar GitHub Desktop es opcional, y muchas personas no lo utilizan, prefiriendo directamente utilizar la consola y las líneas de comando para trabajar

**Ventajas**:

- Interfaz sencilla e intuitiva.
- Permite gestionar commits, ramas y merges en pocos pasos.
- Te muestra las diferencias entre versiones de manera visual.

**¿Cómo usarlo?**:

1. **Descargar GitHub Desktop** desde desktop.github.com.
2. **Clonar un repositorio**: Podés clonar un repositorio existente o crear uno nuevo directamente desde la app.
3. **Hacer commits y push**: Cada vez que modifiques algo en tu proyecto, podés hacer commits y enviar esos cambios a GitHub sin necesidad de abrir la terminal.

## Visual Studio Code (VS Code)

**Visual Studio Code** es un editor de código que tiene integración nativa con Git. Te permite ver los cambios, hacer commits y resolver conflictos directamente desde la interfaz del editor.

**Ventajas**:

- Podés hacer todo sin salir del editor de código.
- Muestra claramente los cambios hechos en cada archivo.
- Facilita la resolución de conflictos visualmente.**¿Cómo usarlo?**:

1. **Abrí tu proyecto en VS Code**.
2. **Usá la barra lateral de Git**: Hacé clic en el ícono de Source Control en la barra lateral para ver los cambios, hacer commits y sincronizar con GitHub.
3. **Sincronización con GitHub**: Podés conectarte con tu cuenta de GitHub y gestionar repositorios remotos directamente desde VS Code.

Con esto, cerramos el tema de **Git y GitHub**, que es esencial para gestionar proyectos y trabajar en equipo de manera eficiente. Recordá que usar Git bien desde el principio te ahorra dolores de cabeza en el futuro, ¡así que a practicar!

💭**Storytelling**

# Control de versiones con Git y GitHub

**Tomás (Desarrollador Senior)**

Hasta ahora vinieron trabajando genial en HTML y CSS. Pero en el mundo real del desarrollo, es vital tener un **registro de los cambios**, colaborar sin pisarse el código y poder volver atrás si algo se rompe.
Acá entra en juego **Git**. Hoy damos un paso grande: aprendemos a versionar y a trabajar en equipo con Git y GitHub.

# Ejercicio práctico #1

## Iniciar repositorio Git y guardar cambios

**Lucía (Product Owner)**

Antes de empezar a trabajar colaborativamente, es importante que tu proyecto esté bajo control de versiones. Esto te va a permitir registrar cambios, volver atrás si algo sale mal, y organizar mejor tu progreso.

**Instrucciones**:

1. **Iniciar Git en tu proyecto local**
    - Instalá Git: https://git-scm.com/
    - Abrí la terminal en la carpeta de tu proyecto y ejecuta:
      `git init`
2. **Guardar tu primer cambio**
    - Asegurate de tener al menos un archivo HTML y un CSS vinculado. Ejecutá:
    - `git add .`
    - `git commit -m "Inicio del proyecto con estructura HTML y`

`estilos básicos"`

3. **Consejos del equipo**
    - Hacé commits frecuentes a medida que avances.
    - Usá mensajes descriptivos (ej: `"Agregado formulario de contacto"` o `"Estilos de la barra de navegación"`).

🛑Podés verificar tus commits con: `git log --oneline`

# Ejercicio práctico #2:

## Subir el proyecto a GitHub y publicarlo con GitHub Pages

Ahora que tenés tu proyecto andando y con cambios registrados en Git, es momento de **publicarlo online** para que cualquiera pueda verlo desde cualquier dispositivo.

## ¿Qué tenés que hacer?

1. **Crear un repositorio en GitHub**
    - Andá a github.com y creá un nuevo repositorio.
    - Nombre sugerido:
      `proyecto-final-ecommerce-[tu-nombre-apellido]`

2. **Vincular tu proyecto local con GitHub**

En la terminal (dentro de la carpeta de tu proyecto), ejecutá:

```
git remote add origin https://LINK DEL REPOSITORIO
```

```
git branch -M main
```

🛑Este comando **renombra la rama actual a `main`**, forzando el cambio si la rama ya existía.

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

🛑Este comando **sube tu rama main al repositorio remoto (origin)** y además le indica a Git que esa es la rama por defecto para futuros push.

3. **Publicar tu sitio con GitHub Pages**
    - Ingresá al repositorio en GitHub.
    - Andá a la pestaña **Settings > Pages**.
    - En “Branch”, seleccioná `main` y la carpeta `/ (root)` si tu `index.html` está en la raíz.
    - Guardá los cambios.
    - GitHub te dará una URL como: `https://tuusuario.github.io/proyecto-final-ecommerce-nombre-apellido`
4. **Compartir**
    - Copiá ese enlace y compartilo en el aula virtual para que el equipo pueda ver tu progreso.

## Objetivo de la clase:

- Subir tu proyecto completo a GitHub.
- Publicarlo usando GitHub Pages para que se vea online.
- Entender cómo compartir tu trabajo de forma profesional.

# Pre-Entrega de Proyecto

Buenos Aires aprende

Agencia de Habilidades para el Futuro

Buenos Aires Ciudad