

«Talento Tech»

Front-End JS

Clase 08



Clase 08 | Git & Github

Temario:

1. Git y GitHub

- ¿Qué es Git?
- ¿Qué es GitHub?
- Instalación y Configuración de Git
- Conceptos Clave en Git
 - Repositorio
 - Branch
 - Commit
 - Merge
- Comandos Básicos de Git
 - git init
 - git add
 - git commit
 - git push
- Crear ramas y trabajar con ellas
 - Crear una nueva rama
 - Hacer cambios y commits en la rama
 - Subir la rama a GitHub
 - Hacer merge de la rama con la principal
 - Resolver conflictos de merge
 - Eliminar ramas que ya no son necesarias
- Flujo de Trabajo con Git y GitHub
 - Crear un Repositorio en GitHub
 - Vincular Repositorio Local con GitHub
 - Sincronización entre Repositorios

2. Ejercicios Prácticos

1. Git y GitHub

¿Qué es Git?



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.

¿Qué es GitHub?



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

Instalación

- **Windows:** Descargá el instalador desde [Git SCM](#) y seguí las instrucciones.

Mac: Si tenés **Homebrew**, podés instalarlo directamente desde la terminal con:

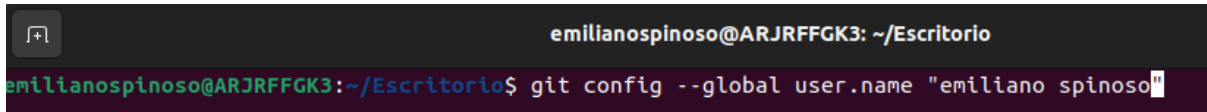
```
brew install git
```

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!

A screenshot of a terminal window. The title bar at the top says "emilianospinoso@ARJRFFGK3: ~/Escritorio". The terminal text shows the command "git config --global user.name 'emiliano spinoso'" being entered at the prompt "emilianospinoso@ARJRFFGK3:~/Escritorio\$".

```
emilianospinoso@ARJRFFGK3:~/Escritorio$ git config --global user.name "emiliano spinoso"
```

```
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`).
- **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.
- **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:

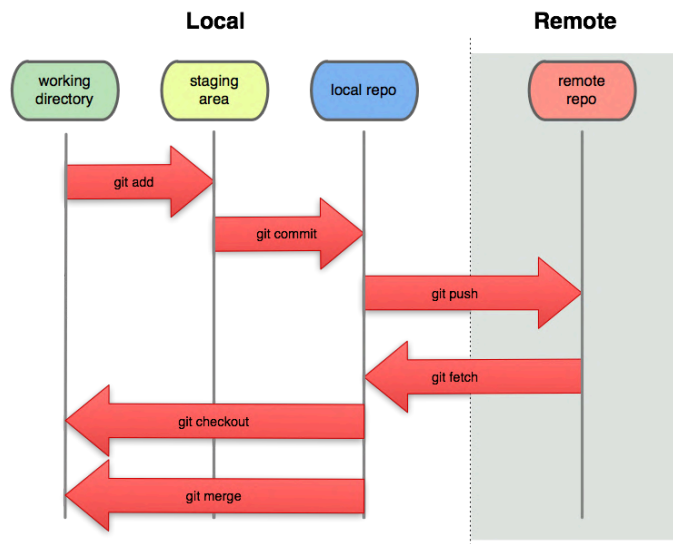
`git add .`

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

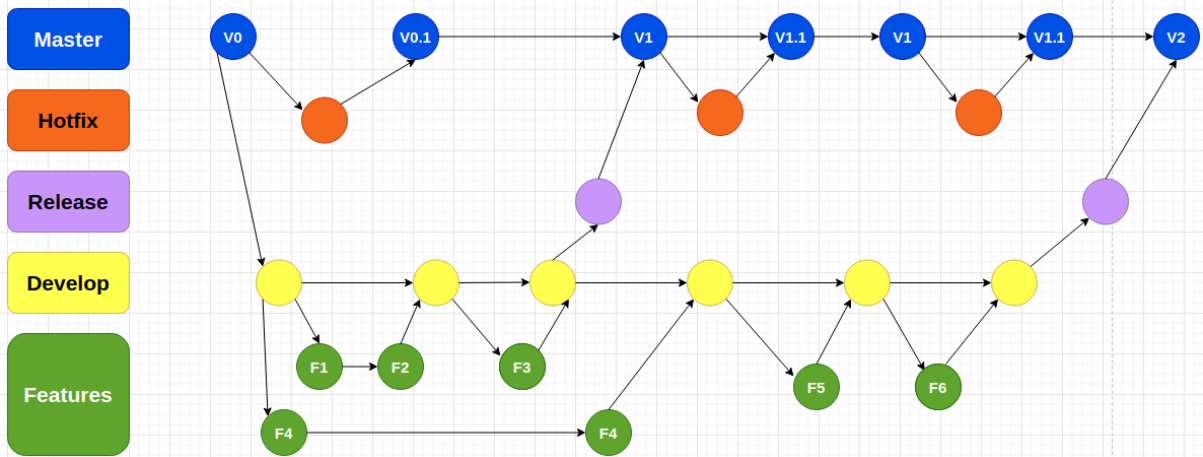
`git commit -m "Mensaje descriptivo de los cambios"`

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.

Ejemplo: Crear y trabajar con ramas

Crear una Nueva Rama: Cuando empezás a trabajar en una nueva funcionalidad o en la corrección de un error, lo primero que hacés es crear una nueva rama. Esto te permite aislar tus cambios y no afectar la rama principal hasta que estés seguro de que todo funciona correctamente.

Comando para crear una nueva rama:

```
git checkout -b nombre-de-la-rama
```

`git checkout -b` crea una nueva rama y cambia automáticamente a esa rama.

Reemplazá `nombre-de-la-rama` con un nombre descriptivo para la rama, como `feature-nueva-funcionalidad` o `bugfix-corregir-bug`.

Ejemplo:

```
git checkout -b feature-agregar-formulario-contacto
```

Ahora estás trabajando en una nueva rama llamada `feature-agregar-formulario-contacto`.

Hacer Cambios en la Nueva Rama: Podés realizar todos los cambios necesarios en esta rama sin afectar el resto del proyecto. Modifica el código, agregá archivos, y asegurate de hacer commits regularmente para registrar tus cambios.

Ejemplo:

```
git add .
```

```
git commit -m "Agregado nuevo formulario de contacto"
```

Subir la Rama al Repositorio Remoto (GitHub): Una vez que estás listo para compartir tu rama o realizar una revisión del código, podés subirla a GitHub.

Comando:

```
git push origin nombre-de-la-rama
```

Ejemplo:

```
git push origin feature-agregar-formulario-contacto
```

Revisar y Fusionar la Rama con la Rama Principal: Después de terminar el trabajo en tu rama, podés fusionarla con la rama principal (main o master). Para esto, primero asegurate de estar en la rama principal:

Comando para cambiar a la rama principal:

```
git checkout main
```

Luego, podés fusionar tu rama con la rama principal usando el comando `git merge`:

Comando para hacer merge:

```
git merge nombre-de-la-rama
```

Ejemplo:

```
git merge feature-agregar-formulario-contacto
```

Esto va a combinar los cambios de la rama

`feature-agregar-formulario-contacto` con la rama principal.


Eliminar la Rama (Opcional): Una vez que la rama ha sido fusionada y no la necesitás más, podés eliminarla para mantener el repositorio organizado:

Comando:

```
git branch -d nombre-de-la-rama
```

Ejemplo:

```
git branch -d feature-agregar-formulario-contacto
```

 **Resolver Conflictos:** Si durante el merge Git detecta que hay conflictos entre tu rama y la principal, tendrás que resolverlos manualmente. Git te mostrará en el archivo afectado dónde están los conflictos, y tendrás que decidir qué cambios conservar.

Flujo de Trabajo con Ramas:

Inicio de una nueva tarea: Crear una nueva rama para trabajar en una nueva funcionalidad o bug.

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

Desarrollo: Realizás cambios y los registrás en la rama mediante commits:

```
git add .
```

```
git commit -m "Agregada nueva funcionalidad"
```

Subida de cambios al repositorio remoto: Subís tu rama al repositorio para compartirla o hacer una revisión:

```
git push origin feature-nueva-funcionalidad
```

Finalización: Fusionás los cambios con la rama principal una vez que la funcionalidad está lista:

```
git checkout main
```

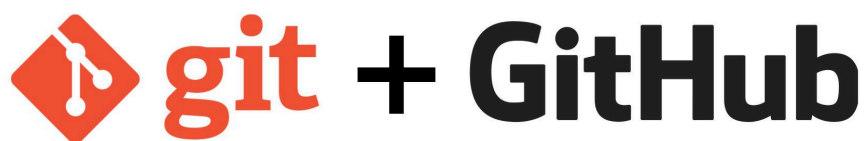
```
git merge feature-nueva-funcionalidad
```

Limpieza: Eliminás la rama si ya no es necesaria:

```
git branch -d feature-nueva-funcionalidad
```

Este flujo de trabajo con ramas es útil para organizar los cambios, realizar revisiones y mantener un entorno de desarrollo estable.

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, volvé 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. **Abrió 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!

Ejercicio 1: Crear una nueva rama en Git

Crear una nueva rama en tu repositorio Git además de la rama principal (main o master). Nombrá la nueva rama de acuerdo con la funcionalidad o característica que estás desarrollando. Asegurate de hacer un commit en la nueva rama y luego subila al repositorio remoto en GitHub.

Tips:

- Creación de la rama: Usá el comando `git checkout -b nombre-de-la-rama` para crear y cambiarte a la nueva rama al mismo tiempo.
 - Verificar la rama actual: Ejecutá `git branch` para verificar que estés en la rama correcta antes de hacer cambios.
 - Hacer commits en la nueva rama: No olvides usar `git add .` seguido de `git commit -m "Descripción de los cambios"` para registrar los cambios en la nueva rama.
 - Subir la nueva rama a GitHub: Usá `git push origin nombre-de-la-rama` para subir tu nueva rama al repositorio remoto.
-

Ejercicio 2: Repositorio en Github

Crear un repositorio en GitHub para tu proyecto, subir el código actual y realizar commits documentando los cambios. Asegurarse de que el repositorio esté configurado para futuras actualizaciones. A continuación te mostramos cómo nombrar el repositorio.

Nombre del repositorio: proyecto-final-ecommerce-[nombre-apellido]

Tips:

- **Organización del repositorio:** asegurate de que el nombre del repositorio siga el formato correcto: proyecto-final-ecommerce-[nombre-apellido]. Esto te ayudará a mantener un esquema de nombres claro y profesional.
- **Documentación de commits:** Siempre que hagas un commit, utilizá mensajes claros y descriptivos que expliquen los cambios realizados. Ejemplo: git commit -m "Añadido formulario de contacto".
- **Mantener el repositorio actualizado:** A medida que avances en tu proyecto, asegurate de hacer commits frecuentemente y subir tus cambios a GitHub para que tu trabajo esté siempre respaldado.
- **Futuras actualizaciones:** Configuraré tu repositorio para futuras actualizaciones asegurándote de hacer pull antes de realizar push cuando trabajes desde distintas ubicaciones o dispositivos.



Buenos Aires
~ aprende ~
Agencia de Habilidades para el Futuro

BA Buenos
Aires
Ciudad