Introducción a Git y GitHub

Objetivos del curso:

- Entender los conceptos básicos de Git y GitHub.
- Aprender a crear y gestionar un repositorio en GitHub.
- Realizar operaciones básicas con un repositorio: clonarlo, modificarlo y subir cambios.

Sección 1: Introducción a Git y GitHub

1. Git: Un Sistema de Control de Versiones (SCV) Distribuido

¿Qué es Git?

- Git es un **sistema de control de versiones distribuido** que permite a los desarrolladores rastrear los cambios en su código a lo largo del tiempo. Cada cambio se guarda como un "commit", lo que permite a los usuarios ver exactamente quién cambió qué y cuándo.

Desarrollo de software en el pasado:

- Antes de Git, se usaban sistemas de control de versiones centralizados como **CVS** o **Subversion (SVN)**, donde un servidor central almacenaba el código y los desarrolladores tenían que conectarse a este servidor para obtener la última versión del proyecto. Esto presentaba desventajas, como:
  - **Dependencia del servidor**: Si el servidor fallaba, todos los desarrolladores se quedaban sin acceso.
  - **Conflictos frecuentes**: Cuando varios desarrolladores trabajaban en un archivo, se generaban muchos conflictos que eran difíciles de resolver sin perder cambios.

Ventajas de Git sobre los sistemas anteriores:

- **Distribuido**: Cada desarrollador tiene una copia completa del historial del proyecto en su máquina local, lo que permite trabajar sin conexión y realizar cambios que luego se pueden sincronizar.
- **Velocidad**: Al tener el historial local, muchas operaciones en Git son más rápidas que en sistemas centralizados.
- **Mejor manejo de ramas (branches)**: Git facilita la creación de branches, permitiendo a los desarrolladores trabajar en nuevas funcionalidades o corregir errores sin afectar el código principal (branch `main`).

2. GitHub: Un repositorio remoto para Git

¿Qué es GitHub?

- GitHub es una plataforma basada en la web que utiliza Git como base para almacenar proyectos de software y permite la colaboración entre equipos distribuidos. Es particularmente popular en proyectos de código abierto.

Colaboración en el pasado:

- En las primeras etapas del desarrollo de software, colaborar en proyectos era complicado:
  - Los equipos de desarrollo compartían archivos a través de correos electrónicos o servidores FTP, lo que generaba **versiones duplicadas** del mismo archivo y dificultades para fusionar los cambios realizados por diferentes personas.
  - **Confusión de versiones**: No existía un seguimiento claro de quién había cambiado qué parte del código y cuándo.

Ventajas de GitHub sobre los métodos anteriores:

- **Almacenamiento centralizado y accesible**: GitHub permite tener una copia del proyecto en la nube, accesible para todo el equipo, y el control de versiones está integrado.
- **Facilita la colaboración**: Los **Pull Requests** permiten a los desarrolladores proponer cambios que otros pueden revisar antes de fusionar, facilitando la colaboración en proyectos.
- **Historial y seguridad**: GitHub guarda un historial detallado de los cambios, lo que facilita la revisión y la restauración de versiones anteriores si es necesario.

Sección 2: Creación y gestión de un repositorio

1. Creación de un Repositorio en GitHub

A través de la plataforma web:

1. **Iniciar sesión en GitHub**:
   - Dirígete a GitHub e inicia sesión en tu cuenta.
2. **Crear un nuevo repositorio**:
   - En la esquina superior derecha, haz clic en el ícono **"+"** y selecciona "New repository" (Nuevo repositorio).
3. **Configurar el repositorio**:
   - **Repository name (Nombre del repositorio)**: Escribe un nombre para tu repositorio (ej. `mi-primer-repo`).
   - **Description (opcional)**: Puedes agregar una breve descripción del propósito del repositorio.
   - **Visibility**: Elige si el repositorio será público (accesible por cualquiera) o privado (solo para ti y las personas que invites).
   - **Initialize this repository with a README**: Si seleccionas esta opción, GitHub generará un archivo README.md con el cual tu proyecto comenzará a tener documentación.
4. **Crear el repositorio**:
   - Haz clic en **Create repository** para finalizar el proceso.

Creación de un Repositorio desde la Consola

A continuación, te muestro cómo crear un repositorio localmente utilizando la consola (terminal) y luego conectarlo con GitHub.

0. Descargar e instalar git

1. Dirígete a https://git-scm.com/downloads y baja el instalador para tu sistema operativo.
2. Sigue los pasos del instalador
3. Desde ahora cuentas con la posibilidad de abrir la consola con la opcion Open Git Bash here

1. Inicializar Git en tu directorio local:

1. Abre una terminal (puedes usar Git Bash, PowerShell o cualquier terminal de tu elección).

2. Crea una carpeta en tu máquina local donde estará el proyecto:

```
mkdir mi-proyecto
cd mi-proyecto
```

3. Inicializa un repositorio de Git en ese directorio:

```
git init
```

   Esto crea un nuevo repositorio vacío en tu máquina local. Git ahora comenzará a rastrear los cambios en los archivos de este directorio.

4. (Opcional) Crea un archivo README para describir el proyecto:

```
echo "# Mi primer repositorio" > README.md
```

5. Añade este archivo al área de preparación (staging):

```
git add README.md
```

6. Haz un commit para guardar este cambio en el historial del repositorio:

```
git commit -m "Inicializar el repositorio con README"
```

2. Crear un repositorio en GitHub y conectarlo a tu repositorio local:

1. Crea un nuevo repositorio en GitHub siguiendo los pasos descritos anteriormente (plataforma web). **No inicialices con README** si ya lo creaste en local.

2. Una vez creado el repositorio en GitHub, verás una URL similar a esta: `https://github.com/tu-usuario/mi-proyecto.git`. Este será el repositorio remoto donde alojarás tu código.

3. Conecta tu repositorio local con el repositorio remoto en GitHub:

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

4. Sube los cambios de tu repositorio local al repositorio remoto (GitHub):

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

---

2. Clonar un Repositorio

Clonar un repositorio significa crear una copia exacta del repositorio remoto en tu máquina local. Así puedes trabajar en el código de forma local.

1. Ve al repositorio que quieras clonar en GitHub y copia la URL del repositorio (puedes usar HTTPS o SSH).

2. En tu terminal, usa el siguiente comando para clonar el repositorio:

   ```
   git clone https://github.com/tu-usuario/mi-proyecto.git
   ```

Este comando descargará una copia completa del repositorio en tu máquina local, con todo el historial de commits.

---

3. Cambiar de Ramas (Branches)

Un **branch** es una rama de desarrollo. Te permite trabajar en nuevas funcionalidades o corregir errores sin afectar el código principal (normalmente en la rama `main` o `master`).

Crear y cambiar a una nueva rama:

1. Crea un nuevo branch:

   ```
   git checkout -b nombre-del-branch
   ```

   Esto crea un nuevo branch y te cambia automáticamente a esa rama para que empieces a trabajar. Por ejemplo, podrías crear una rama llamada `nueva-funcionalidad`:

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

2. Cambiar entre ramas existentes:

   ```
   git checkout main
   ```

   Con este comando, vuelves a la rama principal (`main` o `master`). Puedes cambiar entre cualquier rama que hayas creado.

---

4. Hacer cambios y subirlos a GitHub

Una vez que has hecho cambios en tu proyecto (por ejemplo, has editado o creado archivos), debes seguir una serie de pasos para que esos cambios se guarden en el historial de Git y se suban a GitHub.

Hacer cambios y confirmar (commit) los cambios:

1. Asegúrate de que los archivos modificados estén listos para ser rastreados por Git:

   ```
   git add archivo-modificado.txt
   ```

   O puedes añadir todos los archivos cambiados:

   ```
   git add .
   ```

2. Haz un commit con un mensaje que describa los cambios:

   ```
   git commit -m "Descripción clara de los cambios"
   ```

Subir los cambios a GitHub:

1. Si estás en la rama en la que has hecho cambios, sube esos cambios al repositorio remoto (GitHub):

   ```
   git push origin nombre-del-branch
   ```

   Por ejemplo, si estás en la rama `nueva-funcionalidad`, el comando sería:

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

---

5. Pull Request: Qué es y cómo hacerlo

Un **Pull Request (PR)** es una solicitud para fusionar los cambios de una rama (branch) en otra, generalmente de una rama de desarrollo (`nueva-funcionalidad`) a la rama principal (`main`).

¿Por qué usar un Pull Request?

- Permite a otros revisores inspeccionar los cambios antes de fusionarlos en el código principal.
- Facilita la colaboración en equipo y la revisión del código.

Cómo crear un Pull Request en GitHub:

1. Una vez que has subido los cambios a un nuevo branch (por ejemplo, `nueva-funcionalidad`), ve a GitHub.
2. GitHub te sugerirá automáticamente crear un Pull Request desde la rama `nueva-funcionalidad` hacia `main`. Haz clic en **Compare & pull request**.
3. Proporciona un título descriptivo y una descripción de los cambios.
4. Haz clic en **Create pull request**.

Otros colaboradores pueden ahora revisar tu código y discutir los cambios antes de que se fusionen en el código principal.

---

6. Fusionar un Pull Request (Merge)

Después de que los cambios hayan sido revisados y aprobados, puedes fusionar la rama `nueva-funcionalidad` en `main`.

1. Desde la página del Pull Request en GitHub, haz clic en **Merge pull request**.
2. Confirma la fusión haciendo clic en **Confirm merge**.

Fusionar ramas desde la consola:

Si prefieres hacer la fusión desde la terminal, sigue estos pasos:

1. Cambia a la rama `main` (o la rama que desees fusionar):

```
git checkout main
```

2. Fusiona la rama `nueva-funcionalidad` en `main`:

```
git merge nueva-funcionalidad
```

3. Después de fusionar, puedes eliminar la rama que ya no necesitas:

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

Comparación: Desarrollo de Software Antes vs. Ahora

Antes	Ahora (Con Git y GitHub)
Versionado manual (copiar archivos)	Versionado automático con Git: Cada cambio se guarda como un commit y está registrado.
Comunicación vía correo o FTP	Colaboración centralizada en GitHub: Acceso y revisión de código en un solo lugar.
Difícil resolución de conflictos	Git permite trabajar en branches, lo que reduce conflictos y facilita la revisión de cambios.
Confusión de versiones y falta de historial	Historial claro y preciso con Git: Puedes ver qué cambió, quién lo cambió y cuándo.
Dependencia del servidor central (SVN, CVS)	Git es distribuido: Puedes trabajar offline y sincronizar los cambios cuando sea necesario.

Ventajas de Git y GitHub en el Desarrollo Moderno

1. **Colaboración sin fronteras**:

- Equipos distribuidos en todo el mundo pueden colaborar sin barreras geográficas.
- Los Pull Requests y las revisiones de código promueven buenas prácticas en el desarrollo colaborativo.

2. **Mejora en la calidad del código**:
   - GitHub fomenta la revisión continua del código, ya que otros desarrolladores pueden revisar los cambios antes de integrarlos en el código principal.
   - Facilita la integración de herramientas de CI/CD (Integración Continua/Despliegue Continuo) para la automatización de pruebas.
3. **Transparencia y trazabilidad**:
   - Git proporciona un historial completo de cambios, lo que hace que el código sea más transparente y los errores sean más fáciles de rastrear.
   - GitHub integra funciones como Issues y Proyectos para gestionar el ciclo de vida de un proyecto de software, permitiendo un seguimiento eficiente de problemas y tareas.
4. **Código abierto y comunidad**:
   - GitHub es la plataforma líder para proyectos de código abierto, lo que ha permitido el desarrollo de grandes proyectos de software colaborativo como **Linux**, **React**, **TensorFlow**, entre otros.
   - Los desarrolladores pueden contribuir a proyectos públicos y aprender de otros mediante la interacción con el código fuente.

Sección 3: Operaciones básicas en un repositorio

1. **Modificar un Repositorio**:
   - Explicación de cómo editar o añadir archivos al repositorio.
   - Comandos básicos:
     - `git add <archivo>`: Añadir archivos al área de preparación (staging).
     - `git commit -m "Mensaje del commit"`: Guardar los cambios en el historial de Git.
     - `git push`: Subir los cambios al repositorio remoto en GitHub.
2. **Ver el historial de commits**:
   - Comando: `git log`.
3. **Creación de Branches**:
   - Crear un branch nuevo: `git checkout -b <nombre_del_branch>`.
   - Cambiar de branch: `git checkout <nombre_del_branch>`.

Git & GitHub Cheat Sheet:

- `git init`: Inicializar un nuevo repositorio de Git.

- `git clone <url>`: Clonar un repositorio existente.
- `git add <archivo>`: Añadir archivos a la zona de preparación.
- `git commit -m "mensaje"`: Crear un commit con los cambios.
- `git push`: Subir cambios al repositorio remoto.
- `git checkout -b <nombre>`: Crear y cambiar a un branch nuevo.
- `git merge <branch>`: Combinar los cambios de otro branch en el actual.

Flujo de Trabajo en Git:

1. Trabaja localmente en tu máquina.
2. Usa `git add` y `git commit` para guardar cambios.
3. Usa `git push` para sincronizar con el repositorio en GitHub.
4. Usa Pull Requests para colaborar y hacer revisiones de código.

Sección 4: Práctico

Instrucciones del Práctico:

1. **Crear un Repositorio en GitHub**:
   - Dirígete a GitHub y crea un nuevo repositorio llamado `mi-repositorio-ejemplo`. Deja la opción "Initialize this repository with a README" activada.
2. **Clonar el Repositorio**:
   - Abre tu terminal y clona el repositorio en tu máquina local usando el siguiente comando:

```
git clone https://github.com/tu-usuario/mi-repositorio-ejemplo.git
```

3. **Modificar el Repositorio**:
   - En tu máquina local, crea un archivo de texto llamado `archivo.txt` dentro del repositorio clonado y añade el texto:

```
Este es un archivo de ejemplo para el curso de GitHub.
```

4. **Subir los Cambios a GitHub**:
   - Añade los cambios a la zona de preparación (staging):

```
git add archivo.txt
```

   - Realiza un commit con un mensaje descriptivo:

```
git commit -m "Añadir archivo de ejemplo"
```

   - Sube los cambios a GitHub:

```
git push origin main
```

5. **Crear un Branch Nuevo y Modificarlo**:
   - Crea un branch nuevo llamado `nueva-funcionalidad`:

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

   - Modifica el archivo `archivo.txt` agregando una segunda línea:

     ```
     Esta es una segunda línea en el branch nueva-funcionalidad.
     ```

   - Realiza los pasos de `git add`, `git commit`, y `git push` nuevamente, pero esta vez en el nuevo branch:

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

6. **Realizar un Pull Request en GitHub**:
   - Dirígete a GitHub y verás una opción para crear un **Pull Request** desde el branch `nueva-funcionalidad` hacia el branch `main`. Crea el pull request.

Conclusión

Git y GitHub han revolucionado la manera en que desarrollamos software, comparado con las prácticas anteriores. Mientras que en el pasado se dependía de sistemas centralizados y la gestión manual de versiones, ahora Git permite un control eficiente, distribuido y seguro del código. GitHub, además, agrega una capa de colaboración esencial para equipos distribuidos, facilitando el desarrollo de software de manera más profesional y organizada.