Unidad 6:

6.1.Introducción a Git y GitHub

Git es un **sistema de control de versiones distribuido** que permite **rastrear cambios**, **colaborar de forma paralela mediante ramas** y **mantener un historial íntegro y verificable**.
Su uso correcto requiere **configurar identidad, editor, rama por defecto y reglas de fin de línea** desde el inicio para evitar inconsistencias en proyectos con distintos sistemas operativos.

🌟 Características Clave de GitHub

GitHub no es solo un lugar para alojar repositorios: es una plataforma completa de colaboración, automatización y seguridad para proyectos de software. A continuación, se detallan sus principales características, actualizadas con las prácticas modernas que hoy son estándar en equipos de desarrollo:

📁 Repositorios

Los repositorios almacenan tu proyecto, incluyendo el código fuente, la historia completa de commits y ramas, issues, documentación y más. Podés crear repositorios públicos (visibles para toda la comunidad) o privados (accesibles solo para tus colaboradores).
Cada repositorio funciona como un espacio centralizado donde los miembros del equipo trabajan de forma paralela utilizando ramas y pull requests, sin pisarse entre sí.

🤝 Colaboración

GitHub facilita el trabajo en equipo mediante **pull requests**, revisiones de código, asignación de tareas, discusiones y etiquetado.
Los pull requests permiten proponer cambios desde una rama y someterlos a revisión antes de integrarlos en la rama principal.
Además, podés usar **branch protection rules** para exigir revisiones, status checks aprobados y políticas de fusión claras.

> 💡 Esto permite implementar flujos profesionales (como *GitHub Flow* o *Trunk-Based Development*) con revisiones obligatorias y CI automáticas, manteniendo la rama principal siempre estable.

🔐 Autenticación moderna y seguridad

Hoy en día, **GitHub ya no permite autenticarse con usuario y contraseña para operaciones Git**. En su lugar, se utilizan dos métodos principales:

- **Tokens personales (PAT)**: se generan desde tu cuenta de GitHub y se usan como contraseña en push/pull sobre HTTPS.
- **Claves SSH**: proporcionan autenticación segura basada en criptografía de clave pública. Es el método recomendado para proyectos frecuentes.

Además, GitHub ofrece medidas de seguridad integradas:

- **Autenticación en dos pasos (2FA)**: altamente recomendada para proteger tu cuenta.
- **Branch protection rules**: definen quién puede hacer push, requieren revisiones antes de fusionar, y pueden bloquear force-pushes a ramas críticas como `main`.
- **Required reviews** y **status checks obligatorios**: aseguran que cada cambio pase por revisión humana y pruebas automatizadas.
- **Firmas GPG opcionales** para commits, agregando integridad criptográfica al historial.

> ⚠️ Configurar correctamente estos mecanismos desde el inicio es clave para evitar incidentes de seguridad y garantizar la trazabilidad de cada cambio.

⚙️ Integraciones y Automatización (CI/CD)

GitHub se integra con numerosas herramientas externas y, además, incluye su propio sistema de automatización: **GitHub Actions**.
Con Actions, podés configurar flujos de integración continua (CI) y entrega continua (CD) directamente desde tu repositorio mediante archivos YAML.
Ejemplo: ejecutar tests automáticamente en cada pull request, hacer deploy a producción al fusionar en `main`, o verificar estándares de código antes de aceptar cambios.

> 🚀 Esto elimina la necesidad de depender de plataformas externas para automatizar tareas comunes, mejorando la velocidad de desarrollo.

🌍 GitHub Pages

GitHub Pages permite alojar sitios web estáticos de forma gratuita directamente desde un repositorio. Es ideal para portfolios, landing pages, blogs generados con frameworks estáticos o documentación de proyectos.
Solo tenés que habilitar la opción en la configuración del repo y seleccionar la rama y carpeta donde está tu sitio.

⚠️ Aclaración: hoy se recomienda Pages con **Actions** y que para repos privados o orgs conviene usar **"Source: GitHub Actions"**. Mencionar rutas base (`base href`) y carpeta `docs/` como alternativa simple.

Configurar Git correctamente desde el inicio es crucial para un flujo de trabajo eficiente. En este bloque, aprenderás cómo instalar y configurar Git en tu sistema.

Windows

1. Descarga el instalador de Git desde git-scm.com.
2. Ejecuta el instalador y sigue las instrucciones en pantalla.
3. Acepta las opciones predeterminadas a menos que tengas una razón específica para cambiarlas.
4. Una vez completada la instalación, abre la terminal de Git Bash para verificar la instalación con el siguiente comando: git --version

macOS

1. Abre la Terminal.
2. Instala Git utilizando "Homebrew" con el siguiente comando: brew install git
3. Verifica la instalación con el comando: git --version

Linux

1. Abre tu terminal.
2. Instala Git utilizando el gestor de paquetes de tu distribución.

 # En Debian/Ubuntu:

sudo apt-get install git

 # En Fedora:

```
sudo dnf install git
```

3. Verifica la instalación con el comando: `git --version`

⚙️ Configuración inicial recomendada

Antes de usar Git en un proyecto, es fundamental realizar una configuración base para evitar conflictos posteriores:

1. Abre la terminal.
2. Configura tu nombre de usuario y tu correo electrónico:

```
git config --global user.name "Tu Nombre"
```

```
git config --global user.email "tuemail@example.com"
```

Verificación de la Configuración

Puedes verificar la configuración de Git en cualquier momento utilizando el siguiente comando:

```
git config --show-origin --list
```

Este comando muestra todas las configuraciones actuales. Es útil para asegurarte de que tu nombre, email y demás opciones están correctas.

Configuración de un Editor de Texto

Git usa un editor para escribir mensajes de commit, editar merges o realizar operaciones interactivas.
Podés elegir el que más uses en tu día a día. Por ejemplo, para usar **Visual Studio Code**:

```
git config --global core.editor "code --wait"
```

📌 El flag `--wait` le indica a VS Code que Git debe esperar a que cierres el editor antes de continuar con la operación.
Si preferís otro editor (nano, vim, Sublime, etc.), podés configurarlo también.

Definí la rama por defecto

De forma predeterminada, Git solía crear la rama `master` como inicial. Hoy en día, la convención moderna (incluyendo GitHub) es usar **`main`** como rama por defecto.
Esto mejora la integración con servicios externos y evita confusiones al inicializar nuevos repositorios.

```
git config --global init.defaultBranch main
```

✅ De esta forma, cada vez que crees un repositorio nuevo con `git init`, la rama inicial será `main` automáticamente, sin necesidad de renombrarla después.

Configurá el manejo de fin de línea (EOL)

Este paso es crucial en **equipos que usan diferentes sistemas operativos** (Windows, macOS, Linux).
Cada sistema trata los saltos de línea de forma distinta, lo que puede generar "ruido" en los commits si no se normaliza desde el principio.

```
# macOS / Linux
git config --global core.autocrlf input

# Windows
git config --global core.autocrlf true
```

- **`input`** (recomendado en macOS/Linux): Git convierte saltos de línea CRLF a LF al commitear, pero no modifica al hacer checkout.
- **`true`** (recomendado en Windows): Git convierte saltos de línea LF a CRLF al hacer checkout y los normaliza a LF al commitear.

> ⚠️ Si no se configura correctamente, los cambios de EOL pueden aparecer como modificaciones en todos los archivos, generando conflictos innecesarios.

6.2. Creación y Gestión de Repositorios

Para iniciar un nuevo proyecto con Git, primero debes crear un nuevo repositorio. Sigue estos pasos:

1. Crea una nueva carpeta para tu proyecto.
2. Navega a esa carpeta en la terminal.
3. Inicializa un nuevo repositorio con el siguiente comando: git init

Clonar un Repositorio

Clonar un repositorio es uno de los primeros pasos para trabajar con un proyecto existente en Git. Utiliza el siguiente comando para clonar un repositorio:

```
git clone https://github.com/usuario/repositorio.git
```

Resumen

Configurar Git correctamente es un paso fundamental para cualquier desarrollador. Este proceso incluye la instalación de Git, la configuración de tu identidad de usuario, la verificación y la configuración de un editor de texto. Además, aprender a clonar repositorios y a crear nuevos repositorios te preparará para gestionar y colaborar en proyectos de desarrollo de manera eficiente.

Y si acaso en un futuro necesitas instalar y configurar Git de nuevo, puedes acceder al sitio de Git y encontrar allí la documentación para seguir el paso a paso. Leer la documentación es una buena práctica, ¡adoptala!

Próximos Pasos

En el siguiente bloque, aprenderemos comandos básicos de Git para añadir archivos, realizar commits y trabajar con ramas. Estas habilidades te permitirán comenzar a utilizar Git de manera efectiva en tus proyectos de desarrollo web.

6.3.Estados y Flujo de Trabajo en Git

Imaginá que tu proyecto es como una cámara:

1. **Working Directory (directorio de trabajo)** → donde sacás la foto (editás tus archivos).
2. **Staging Area (área de preparación)** → elegís qué fotos querés guardar (`git add`).
3. **Repository (repositorio local)** → guardás la foto en tu álbum (`git commit`).
4. **Remote (repositorio remoto)** → subís el álbum a la nube (`git push`).

> ⚠️ Importante: `git push` **no forma parte del repositorio local**. Solo funciona si ya configuraste un remoto (por ejemplo, GitHub) y tenés permisos.

-

Áreas en Git

Estos estados se corresponden con tres áreas:

- **Directorio de Trabajo (Working Directory):** Área donde se trabaja con los archivos. Pueden estar en estado modificado.
- **Área de Preparación (Staging Area):** Área donde se preparan los archivos para confirmar. Están en estado preparado.
- **Directorio de Git (Git Directory):** Área donde se almacenan los cambios confirmados.

Flujo de Trabajo en Git

El flujo de trabajo básico en Git implica mover archivos entre estas áreas. A continuación se describe el proceso típico:

1. **Modificar Archivos en el Directorio de Trabajo:**
    - Realizás cambios en los archivos de tu proyecto.
    - Estos archivos están en estado modificado.
2. **Agregar Archivos al Área de Preparación:**
    - Utiliza el comando `git add <archivo>` para mover los archivos modificados al área de preparación.
    - Ahora, estos archivos están en estado preparado.
3. **Confirmar Cambios en el Directorio de Git:**
    - Utiliza el comando `git commit -m "mensaje"` para confirmar los cambios preparados.
    - Los archivos pasan al estado confirmado y se almacenan en el directorio de Git.

Analogía:

Te reunes con amigos y les decís que se van a sacar una foto para recordar ese momento tan importante, por lo cual todos se preparan para ello abrazándose y juntándose para la foto (git add - todos están preparados). Una vez sacaste la foto, la estás preparando para subirla a tu red social favorita, y en la descripción ponés una frase acorde al momento (git commit - descripción del momento vivido). Posterior a eso, decidís subirla para que la gente la vea (git push - la subiste).

Te estarás preguntando: ¿Qué significa git push y dónde se suben los archivos? Eso lo veremos en la próxima sección.

Comandos Básicos

- git status: Verifica el estado de los archivos en el directorio de trabajo y el área de preparación.
- git add: Añade un archivo modificado al área de preparación.

- git commit -m "mensaje": Confirma los cambios en el área de preparación con un mensaje descriptivo.
- git log: Muestra el historial de commits en el repositorio.
- `git switch -c feature-x` (crear/cambiar rama)

- `git restore --staged archivo` (sacar del stage)

Ejemplo de Flujo de Trabajo

Se modifica o se crea un nuevo archivo en nuestro proyecto:

Ver el estado de los archivos → git status

Preparar el archivo para confirmación → git add archivo.txt

Confirmar los cambios → git commit -m "Añadir archivo de ejemplo"

Ver el historial de commits → git log

6.4.GitHub: Control de Versiones

El control de versiones es una práctica esencial en el desarrollo de software, permitiendo gestionar y registrar cambios en el código de manera eficiente.

Beneficios del Control de Versiones

- **Historial de cambios:** Registro detallado de modificaciones en el código a lo largo del tiempo.
- **Colaboración eficiente:** Múltiples desarrolladores pueden trabajar en el mismo proyecto sin conflictos.
- **Deshacer cambios:** Posibilidad de revertir a versiones anteriores del código si es necesario.
- **Ramas (Branches):** Trabajo simultáneo en diferentes características o correcciones sin interferencias.

Flujo de Trabajo Básico en GitHub

1. **Clonar un Repositorio:** Obtener una copia local de un proyecto.
2. **Crear una rama:** Desarrollar nuevas características o corregir errores en una rama separada.
3. **Hacer commits:** Guardar los cambios localmente con descripciones significativas.
4. **Hacer push:** Enviar los commits a GitHub.

5. **Crear un pull request:** Solicitar la integración de los cambios en la rama principal del proyecto.
6. **Revisar y fusionar:** Revisar el código y fusionar los cambios después de la aprobación.

6.5. Creando un repositorio

Referencias

- Introducción a GitHubControl de Versiones con Git

Ahora tienes las herramientas necesarias para poner tu proyecto en línea de manera rápida y sencilla. Practica estos pasos y pronto te sentirás cómodo utilizando GitHub Pages en tus proyectos web.

6.6.GitHub Copilot: Un Asistente de Programación Impulsado por IA

La programación ha avanzado enormemente en las últimas décadas, pero incluso con las mejores herramientas de desarrollo disponibles, sigue siendo una tarea compleja y laboriosa. La escritura de código manualmente, la búsqueda de soluciones en línea y el ciclo constante de pruebas y correcciones pueden consumir tiempo y energía valiosos. Aquí es donde GitHub Copilot, un asistente de codificación impulsado por inteligencia artificial, está cambiando las reglas del juego.

GitHub Copilot utiliza modelos avanzados de IA para sugerir fragmentos de código en tiempo real mientras el desarrollador escribe. Desde líneas simples hasta funciones completas, la IA está diseñada para comprender el contexto del código que estás escribiendo y sugerir automáticamente las soluciones más adecuadas. Esto no solo aumenta la productividad, sino que también reduce significativamente el tiempo necesario para completar tareas repetitivas o tediosas.

1. ¿Qué es GitHub Copilot?

GitHub Copilot hoy se basa en **modelos de OpenAI de nueva generación (GPT-4 y GPT-4o)** integrados por GitHub. En las versiones **Business** y **Enterprise**, se pueden activar controles de **filtrado, retención cero y exclusión de telemetría**, ofreciendo más transparencia y seguridad en el uso de datos.
Es recomendable siempre revisar las **licencias del código sugerido**, aplicar **secret scanning** y mantener **tests automáticos** para garantizar buenas prácticas y evitar exposición de información sensible.

2. ¿Cómo funciona?

GitHub Copilot funciona gracias a un modelo de IA llamado **Codex**, desarrollado por OpenAI. Este modelo ha sido entrenado utilizando millones de repositorios públicos en GitHub, lo que le permite aprender patrones de código y sugerir soluciones basadas en el contexto del proyecto.

- **Análisis en tiempo real**: A medida que el desarrollador escribe código, Copilot analiza el contexto y ofrece sugerencias basadas en patrones de código similares que ha aprendido.
- **Predicción de código**: Copilot puede predecir el siguiente fragmento de código que necesitas, ahorrando tiempo en la escritura manual.
- **Aprendizaje adaptativo**: A medida que usas la herramienta, GitHub Copilot se ajusta a tu estilo de programación y patrones de codificación, volviéndose más preciso y relevante con el tiempo.

3. Funciones clave

- **Completar código automáticamente**: Desde una simple función hasta una línea compleja de lógica condicional, Copilot puede sugerir código que encaje con lo que estás desarrollando.
- **Generación de funciones completas**: Si le proporcionas un comentario bien estructurado, puede escribir una función completa con todas las piezas necesarias para funcionar.
- **Soporte para múltiples lenguajes**: GitHub Copilot admite varios lenguajes de programación, incluidos HTML, CSS, Python, JavaScript, TypeScript, Ruby, Go, y muchos más.

Ejemplo de uso con HTML

Vamos a suponer que estás creando una estructura básica de HTML para una página web. En lugar de escribir todo manualmente, con **GitHub Copilot** puedes simplemente escribir un comentario que describa la estructura deseada, y la IA generará el código correspondiente.

Prompt:

```
<!-- Crea una página HTML con un título, un párrafo de bienvenida y una lista de enlaces. -->
```

Estructura sugerida por GitHub Copilot:

```
<!DOCTYPE html>

<html lang="es">
```

```
<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width,
initial-scale=1.0">

<title>Página de Inicio</title>

</head>

<body>

<h1>Bienvenido a mi sitio web</h1>

<p>Gracias por visitar mi página. A continuación, encontrarás
algunos enlaces útiles:</p>

<ul>

<li><a href="#">Enlace 1</a></li>

<li><a href="#">Enlace 2</a></li>

<li><a href="#">Enlace 3</a></li>

</ul>

</body>

</html>
```

En este caso, **GitHub Copilot** sugiere la estructura completa después de que escribas el comentario. Esto acelera el proceso de creación de una página HTML básica, asegurando que sigas buenas prácticas y evitando posibles errores.

4. Importancia

GitHub Copilot representa un avance significativo en la forma en que los desarrolladores abordan sus proyectos. Al automatizar tareas repetitivas y proporcionar sugerencias de código, permite que los programadores se concentren en resolver problemas más complejos y en optimizar el código en lugar de perder tiempo con tareas mundanas.

- **Aumento de la productividad**: Gracias a las sugerencias automáticas, los desarrolladores pueden completar tareas en menos tiempo, lo que les permite enfocarse en los aspectos más críticos de su proyecto.
- **Reducción de errores**: Copilot sugiere código basado en millones de ejemplos reales, lo que ayuda a evitar errores comunes y mejora la calidad del código desde el principio.
- **Facilidad de uso**: Desde desarrolladores junior hasta experimentados, todos pueden beneficiarse de las capacidades de Copilot. Los programadores menos experimentados pueden aprender mejores prácticas, mientras que los más avanzados pueden automatizar tareas y ahorrar tiempo.

5. Casos de uso

- **Desarrolladores junior**: Aprender a escribir código de manera eficiente y siguiendo buenas prácticas es uno de los mayores desafíos para los desarrolladores principiantes. GitHub Copilot puede actuar como un tutor, sugiriendo las mejores soluciones a problemas comunes y ayudando a los nuevos programadores a comprender mejor el flujo del código.
- **Proyectos colaborativos**: En equipos de desarrollo grandes, la consistencia del código es clave. Copilot puede ayudar a mantener un estilo uniforme en el código que escriben diferentes miembros del equipo, lo que facilita el trabajo colaborativo.
- **Tareas repetitivas**: Los desarrolladores con experiencia a menudo se encuentran escribiendo las mismas líneas de código o funciones una y otra vez. Con GitHub Copilot, pueden automatizar estas tareas, permitiéndoles centrarse en problemas más complejos y personalizados.

6. Ventajas adicionales

- **Integración fluida**: GitHub Copilot está completamente integrado en editores como **Visual Studio Code**, por lo que no es necesario cambiar de herramienta o plataforma.
- **Compatibilidad con repositorios privados**: Aunque Copilot fue entrenado en repositorios públicos, también es capaz de ofrecer sugerencias útiles en proyectos privados.
- **Mejoras continuas**: GitHub y OpenAI están trabajando constantemente en mejorar Copilot, añadiendo nuevas capacidades y ampliando la cantidad de lenguajes y patrones de código que puede manejar.

7. Limitaciones

Aunque GitHub Copilot es una herramienta revolucionaria, no es perfecta. Algunas limitaciones incluyen:

- **Calidad variable de sugerencias**: Dado que Copilot ha sido entrenado en una variedad de fuentes, la calidad de las sugerencias puede variar. A veces puede proponer soluciones innecesarias o subóptimas.

- **No siempre es óptimo para proyectos complejos**: En proyectos más grandes y complejos, puede ser necesario ajustar manualmente el código propuesto por Copilot para que se alinee con las especificaciones exactas del proyecto.
- **Privacidad y seguridad**: Algunos desarrolladores han expresado preocupaciones sobre cómo Copilot podría proponer fragmentos de código que hayan sido copiados de repositorios públicos, lo que podría causar problemas con licencias o propiedad intelectual.

- Aunque Copilot puede acelerar el desarrollo, siempre hay que **evaluar la privacidad y la procedencia del código generado**.
  En entornos empresariales, se recomienda usar **Copilot Business o Enterprise**, donde se pueden aplicar políticas de **retención cero de datos** y **exclusión de telemetría**.
  Recordá: toda sugerencia debe ser revisada por un humano antes de integrarla al código productivo.

Enlace de referencia

- GitHub Copilot

GIT expert + IA

GIT Expert puede actuar como asistente opcional para aprender comandos Git mediante IA. Sin embargo, es importante **verificar los permisos del repositorio**, las **políticas de datos** y la **retención de logs** antes de integrarlo.
Siempre se debe priorizar el uso de **herramientas oficiales** de Git y GitHub (CLI, Actions o Copilot) salvo necesidad explícita de servicios externos.

¿Cómo funciona?

Debes crear una cuenta en la siguiente web: https://codegpt.co/agents/git-expert, y configura GIT Expert. Luego, selecciónalo en tu proyecto e intégralo en tu IDE favorito para comenzar a trabajar junto al agente.