Laboratorio 2: Introduccion a Git

1- Introduccion a Git y GitHub 👽

¿Qué es Git?

- Es un sistema de control de versiones distribuido.
- Permite a los desarrolladores colaborar en proyectos de software.
- Seguimiento de cambios en archivos y codigo fuente.
- Fue desarrollado por Linus Torvalds en 2005.

¿Qué es GitHub?

- Plataforma web que utiliza Git.
- Ofrece herramientas de colaboracion, revision de codigo y gestion de proyectos.
- Es utilizada para almacenar y compartir codigo, colaborar en proyectos de código abierto y gestionar repositorios.

2- Conceptos Básicos de Git.

Repositorio (Repo/Repository):

> Lugar donde se almacena el código de un proyecto junto a su historial de cambios.

Commit:

> Una instantanea de los cambios en el repositorio. Cada commit tiene un mensaje que describe los cambios realizados.

Branch(Rama):

> Permite crear "lineas de tiempo" separadas del proyecto para trabajar en diferentes caracteristicas o corregir errores. Ejemplo: **main o master** suelen ser las ramas principales y por lo general se crean algunas como ***devop, develop, feature/nueva-funcionalidad.***

Merge:

> Proceso por el cual se combinan los cambios de una rama con otra.

Pull Request (PR):

> Solicitud para fusionar cambios de una rama en otra, utilizada en Github principalmente para colaborar y revisar codigo.

Clone y Fork:

> - ***Clone:*** Copiar un repositorio en local.
> - ***Fork:*** Crear una copia de un repositorio en tu cuenta de Github.

3- Comandos Básicos de Git.

- **git init : inicializa un nuevo repositorio de Git.**
- **git clone [url]: Clona un repositorio existente.**
- **git add [archivo]: Añade un archivo al área de preparacion.**
- **git commit -m "mensaje": Realiza un commit de los cambios.**
- **git status: Muestra el estado del repositorio.**
- **git push: Envia los commit locales al repositorio remoto.**
- **git pull: Trae y fusiona cambios en el repositorio remoto.**

4- Instalar Git.

> Link para descargarlo: https://git-scm.com/

Para verificar si Git esta instalado correctamente, ejecutar el siguiente codigo en tu terminal.

```
git --version
```

Si esta instalado correctamente deberá imprimir la version instalada.

5- Configurando tu Cuenta de Git.

Configurar tu cuenta es un paso importante, ya que cada commit que realices tendrá tu nombre y correo electrónico asociado. Los comandos de configuración se ejecutan en la terminal o en el símbolo del sistema.

Nos creamos una cuenta en GitHub, https://github.com/

1. Configurar el nombre de usuario:

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

2. Configurar el correo electrónico:

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

3. Verifica la configuracion:

```
git config --global --list
```

6- Uso Básico de Git.

Acá hay dos formas de utilizar Git, la correcta y la que nos ayuda visual Studio Code. La correcta es mediante el uso de la terminal, y es la que veremos a continuacion.

Crea tu primer Repositorio en Git.

1. Crea una carpeta para tu proyecto con los siguientes comandos:

```
mkdir mi-primer-repositorio
cd mi-primer-repositorio
```

2. Inicializa un repositorio de Git en la carpeta:

```
git init
```

3. Crea un archivo de texto, por ejemplo, README.md:

```
echo "#Mi Primer Repositorio" >> README.md
```

4. Añadir el archivo al área de preparacion

```
git add README.md
```

Podes agregar todos los archivos de la carpeta con el siguiente comando:

```
git add .
```

5. Hacer un commit de los cambios:

```
git commit -m "Agrega el archivo README con título inicial"
```

6. Creando y Usando Ramas(Branches).
 Las ramas nos permiten trabajar en caracteristicas o cambios separados al codigo principal, para este primer ejemplo usaremos la principal ***main o master***

```
El comando es:
git branch nombre-de-la-rama

El que utilizaremos:
git branch -M main

Para cambiar entre ramas:
git checkout nombre-de-la-rama
```

7. Para fusionar cambios de una rama a otra:
 Para fusionar cambios, primero cambiamos a la rama principal y luego ejecutamos:

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

8. Para Eliminar una rama:

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

9. Podemos ver el estado del repositorio y el historial con los siguientes comandos:

```
git status

git log
```

7- Conectar con GitHub:

1. En tu cuenta de GitHub, haz clic en el botón New para crear un nuevo repositorio.
2. Para conectarlo, en la terminal agrega el repositorio remoto(GitHub) a tu repositorio local:

```
git remote add origin https://github.com/tuusuario/mi-primer-repositorio.git
```

3. Subir(Push) los Cambios al repositorio Remoto:

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

8- Conectar nuestra cuenta de GitHub en VisualStudioCode.

Poniendo bonito nuestro perfil:

https://gprm.itsvg.in/ https://rahuldkjain.github.io/gh-profile-readme-generator/ https://profilinator.rishav.dev/ https://github.com/lleriayo/markdown-badges?tab=readme-ov-file#table-of-contents

7 - GitIgnore.

El archivo .gitignore es crucial para mantener un repositorio limpio y evitar subir archivos innecesarios o sensibles. Este archivo le indica a Git qué archivos o directorios debe ignorar en el proceso de seguimiento de cambios. Esto es especialmente útil para archivos generados automáticamente, archivos de configuración locales o secretos, y bibliotecas externas.

¿Por qué usar .gitignore?

- Mantiene el repositorio limpio: Evita que archivos innecesarios se suban al repositorio, reduciendo el desorden.
- Protege información sensible: Evita que credenciales, claves de API, o archivos de configuración privada sean expuestos.
- Reduce conflictos: Al ignorar archivos específicos del entorno, se minimizan los conflictos al fusionar ramas.

Como crear y usar el archivo .gitignore:

1. Crear un archivo con el nombre .gitignore en la raiz de tu proyecto.
2. Añadir archivos o directorios al .gitignore:
    - Para ignorar archivos o carpetas específicas, añade sus nombres o rutas:

```
# Ignorar todos los archivos .log
*.log

# Ignorar la carpeta de configuración local
/config/

# Ignorar archivos de entorno
.env
```

3. Aplicar los cambios en el repositorio:

```
git add .gitignore
git commit -m "Añadir archivo .gitignore para ignorar archivos no deseados"
```

4. Tener en cuenta que los archivos ya versionados no se eliminarán del repositorio automáticamente cuando se añaden al .gitignore. Es necesario eliminarlos manualmente.

8. Buenas prácticas al usar Git.

1. ***Realiza commits Pequeños y Frecuentes***: Haz commits de cambios pequeños y relacionados para mantener el historial de commits limpio y fácil de seguir. Esto facilita la revisión del código y la resolución de conflictos.
2. ***Escribe mensajes de commit claros***: Un buen mensaje de commit debe ser descriptivo y responder a la pregunta "¿Qué cambios introduce este commit?". Ejemplo: "Corrige el error de validación en el formulario de registro".
3. ***Usar Ramas para nuevas funcionalidades y correcciones***: Trabaja en ramas separadas (feature branches) para cada nueva funcionalidad o corrección de errores. Esto permite un flujo de trabajo limpio y evita conflictos en el código principal.
4. ***Mantén la Rama main o master Limpia y Estable***: La rama principal debe estar siempre en un estado funcional y listo para producción. Asegúrate de probar los cambios en las ramas de características antes de fusionarlos con main o master.
5. ***Revisa y prueba antes de hacer merge.***
6. ***Sincroniza tu Repositorio Local Regularmente***

EXTRA: Como funciona un esquema de Versiones Semantico o SemVer.

Muchos de nosotros vimos archivos con nombres del tipo "mi_archivo 1.20.3" esto es lo que se conoce como versionado semantico y ayuda a los desarrolladores y a los usuarios a entender el estado de un software y los cambios que se han realizado en diferentes versiones. Cada numero tiene un significado especifico:

MAJOR.MINOR.PATCH

1. MAJOR(Versión Principal):
    1. El primer número indica una versión principal (mayor). Se incrementa cuando hay cambios significativos o incompatibles con versiones anteriores.
    2. Ejemplos: 1.0.0 / 2.0.0
    3. Estos cambios suelen incluir:
        1. Reescrituras completas del software.
        2. Nuevas características que no son compatibles con las versiones anteriores.
        3. Eliminación de funcionalidades obsoletas.
2. MINOR(Versión Menor):
    1. El segundo número indica una versión menor. Se incrementa cuando se agregan nuevas características que son compatibles con la versión actual, pero no alteran su funcionamiento.
    2. Ejemplo: 1.1.0 / 1.5.0
    3. Estos cambios suelen incluir:
        1. Nuevas características o funcionalidades.
        2. Mejoras que no afectan la compatibilidad.
        3. Modificaciones que no rompen las características existentes.

3. PATCH(Revision o parche):
    1. El tercer número indica una revisión o parche. Se incrementa cuando se realizan correcciones de errores o pequeños cambios que no afectan las características ni la compatibilidad con la versión actual.
    2. Ejemplo: 1.1.1 / 1.9.23
    3. Estos cambios suelen incluir:
        1. Corrección de errores.
        2. Mejoras menores de rendimiento.
        3. Ajustes y pequeños arreglos que no añaden nuevas funcionalidades.

Versiones Adicionales.

A veces, se agregan etiquetas adicionales como alpha, beta, rc (release candidate), para indicar versiones de pre-lanzamiento o desarrollo:

- 1.20.3-alpha: Una versión alpha de la versión 1.20.3, generalmente una versión muy temprana para pruebas.
- 1.20.3-beta: Una versión beta de la versión 1.20.3, más estable que alpha pero aún en pruebas.
- 1.20.3-rc.1: Un "release candidate" (candidato a lanzamiento), indicando que está listo para una posible liberación final.

EXTRA 2: Aprende Git Jugando.

Los comandos de git pueden ser un dolor de cabeza, por eso hay paginas para aprender y familiarizarse con ellos de manera didactica, aquí un ejemplo: https://learngitbranching.js.org/?locale=es_AR