Práctico 2: Git y GitHub

Objetivo:

El estudiante desarrollará competencias para trabajar con Git y GitHub, aplicando conceptos fundamentales de control de versiones, colaboración en proyectos y resolución de conflictos, en un entorno simulado y guiado.

Resultados de aprendizaje:

1. Comprender los conceptos básicos de Git y GitHub: Identificar y explicar los principales términos y procesos asociados con Git y GitHub, como repositorios, ramas, commits, forks, etiquetas y repositorios remotos.
2. Manejar comandos esenciales de Git: Ejecutar comandos básicos para crear, modificar, fusionar y gestionar ramas, commits y repositorios, tanto en local como en remoto.
3. Aplicar técnicas de colaboración en GitHub: Configurar y utilizar repositorios remotos, realizar forks, y gestionar pull requests para facilitar el trabajo colaborativo.
4. Resolver conflictos en un entorno de control de versiones: Identificar, analizar y solucionar conflictos de merge generados en un flujo de trabajo con múltiples ramas.

Actividades

1) Contestar las siguientes preguntas utilizando las guías y documentación proporcionada (Desarrollar las respuestas) :

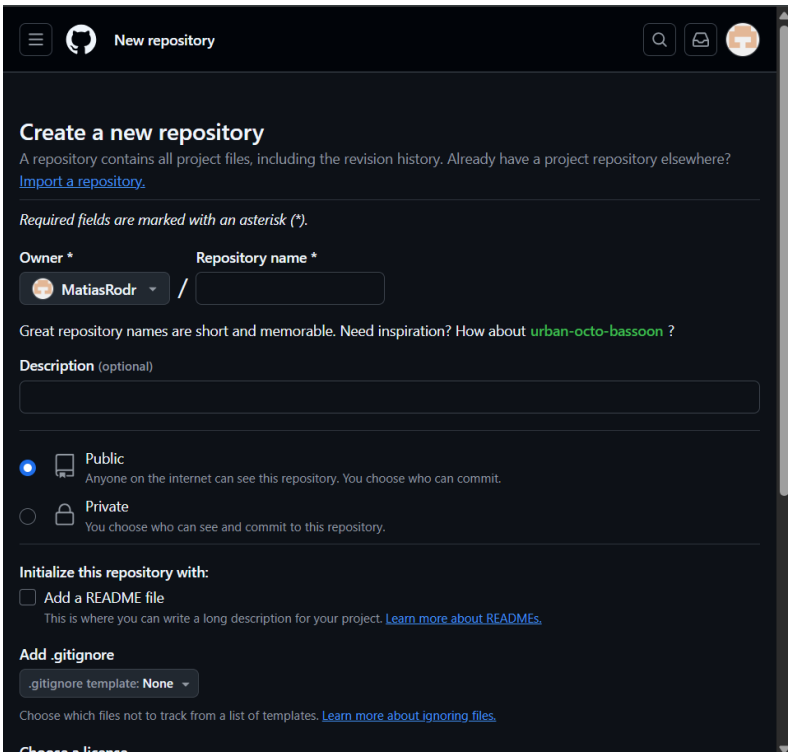
- ¿Qué es GitHub?

  GitHub es una plataforma de desarrollo colaborativo basada en la web que utiliza el sistema de control de versiones Git. Es ampliamente utilizado por desarrolladores para alojar, revisar y gestionar proyectos de software, aunque también puede usarse para otros tipos de proyectos (como documentación o incluso libros).

- ¿Cómo crear un repositorio en GitHub?

  Para crear un repositorio en GitHub, primero deberemos iniciar sesión o crear una cuenta. Una vez que ingresemos debemos ir al panel en la esquina superior derecha y seleccionar “New repository”.

  Una vez ahí debemos ponerle un nombre, una descripción, si queremos que sea que el repositorio sea público o privado, etc.



Cuando terminemos de configurar, deberemos hacer click en crear repositorio. Y listo el repositorio ya va a contar con una URL .

- ¿Cómo crear una rama en Git?

Para crear una rama en GIT, primero podemos verificar nuestra rama actual con:

```
$ git branch
```

Con este código se vera una lista de las ramas existentes y la rama actual estará marcada con un *. Para crear una nueva rama usaremos:

```
$ git branch nueva-rama
```

- ¿Cómo cambiar a una rama en Git?

Para cambiar de una rama a otra, deberemos utilizar el comando:

```
$ git checkout nueva-rama
```

- ¿Cómo fusionar ramas en Git?

Fusionar ramas en Git es el proceso de combinar los cambios de una rama con otra.

Primero, deberemos estar en la rama a la que quieres fusionar los cambios. Por lo general, es a la rama principal (master o main) o cualquier otra rama en la que estés integrando los cambios.

Por ejemplo queremos fusionar “rama-nueva” a la rama master, entonces vamos a pararnos en la principal:

```
$ git checkout master
```

Una en la rama principal, usaremos el comando git merge para fusionar la rama de origen en la rama actual:

```
$ git merge nueva-rama
```

Este comando incorpora los cambios de nueva-rama en master.

- ¿Cómo crear un commit en Git?

Crear un commit en Git es el proceso de guardar cambios en tu repositorio local con un mensaje descriptivo. Para realizar un commit en un proyecto primero deberemos realizar los cambios, una vez listo deberemos preparas los archivos para el commit con el comando:

`$ git add nombre-del-archivo` (para un archivo especifico)

`$ git add .` (para todos los archivos modificados)

Una vez que los cambios están en el área de preparación, podemos realizar el comando commit que va acompañado de un mensaje que describa los cambios realizados:

```
$ git commit m – “Cambio realizados”
```

- ¿Cómo enviar un commit a GitHub?

Primero debemos clonar el repositorio de GitHub a nuestra máquina local:

```
$ git clone https://github.com/tu_usuario/tu_repositorio.git  cd tu_repositorio
```

Haz cambios en los archivos del repositorio y agrégalos:

```
$ git add nombre_del_archivo o git add .
```

Crea un commit de tus cambios:

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

Para enviar los commits al repositorio en GitHub, debemos empujarlos con:

```
$ git push origin nombre_de_la_rama
```

- ¿Qué es un repositorio remoto?

Un repositorio remoto en Git es una versión de tu proyecto alojada en un servidor externo (como GitHub, GitLab o Bitbucket) que permite colaboración, respaldo y

sincronización de código entre múltiples desarrolladores.

- ¿Cómo agregar un repositorio remoto a Git?

  Para vincular mi repositorio local con uno remoto usaremos el codigo

  ```
  $ git remote add [nombre] [url]
  ```

  - [nombre]: Un alias para referenciar el remoto (por defecto suele ser origin).
  - [url]: La dirección del repositorio remoto (HTTPS o SSH).

  Para asegurarte de que el remoto se ha agregado correctamente y verificar el nombre que le has dado, usa el comando:

  ```
  $ git remote –v
  ```

  Esto mostrará una lista de todos los remotos configurados con sus URLs: [nombre] [url]
  Para traer toda la información del repositorio remoto que aún no tienes en tu repositorio local, usa el comando git fetch seguido del nombre del remoto:

  ```
  $ git fetch [nombre]
  ```

  Este comando descarga todos los cambios del repositorio remoto asociado con el nombre, pero no fusiona esos cambios con tu rama actual. Es útil para ver qué cambios están disponibles en el remoto.

- ¿Cómo empujar cambios a un repositorio remoto?

  Antes de empujar tus cambios, es una buena práctica obtener los últimos cambios del repositorio remoto para evitar conflictos:

  ```
  $ git pull origin nombre_de_la_rama
  ```

  Empuja tus cambios al repositorio remoto:

  ```
  $ git push origin nombre_de_la_rama
  ```

- ¿Cómo tirar de cambios de un repositorio remoto?

  Como mencionamos en el punto anterior, para tirar los cambios del repositorio remoto Usa el comando git pull para descargar y fusionar los cambios del repositorio remoto con tu rama local:

  ```
  $ git pull origin nombre_de_la_rama
  ```

  Por ejemplo, si estás trabajando en la rama main:

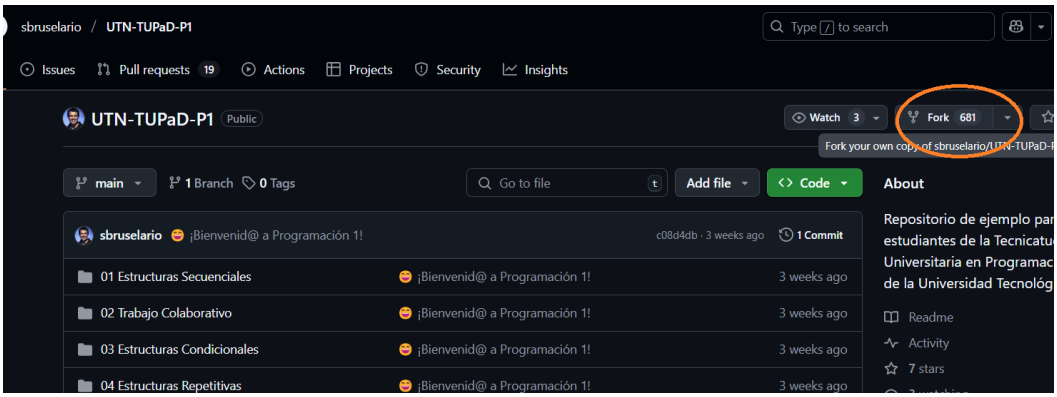
  ```
  $ git pull origin main
  ```

- ¿Qué es un fork de repositorio?

  Un fork es una copia personal de un repositorio de Git (generalmente en plataformas como GitHub, GitLab o Bitbucket) que te permite experimentar, modificar y contribuir a un proyecto sin afectar el repositorio original.

- ¿Cómo crear un fork de un repositorio?

  Para clonar un proyecto (fork) de algún repositorio, deberemos acceder a este y en la esquina superior, hacer clic en el botón “Fork”.



  De ahí deberemos ponerle un nombre y una descripción (opcional), una vez hecho git te llevara a tu copia del repositorio.

- ¿Cómo enviar una solicitud de extracción (pull request) a un repositorio?

  Para hacer un pull request, nos deberemos dirigir a la solapa de Pull requests. Alli podremos hacerlo seleccionado New, una ventana en modo resumen mostrara los cambios que realizamos nosotros en nuestro fork. Daremos click en Create pull request donde deberemos colocar un mensaje global junto un texto en el que detallaremos los cambios realizados por nosotros.

- ¿Cómo aceptar una solicitud de extracción?

  El autor del repositorio verá en sus pull requests el mensaje que le hemos enviado, para que lo pueda observar y si lo considera realizar el cambio pertinente (además de poder responderle al usuario que le ha propuesto ese cambio). Lo bueno de todo esto es que si el usuario original considera que esta modificación es buena y no genera conflictos con la rama maestra de su repositorio local remoto, puede clickear en Merge pull request y de esta manera sumará a su repositorio los cambios que hizo un usuario (en modo de ayuda).

- ¿Qué es un etiqueta en Git?

  Una etiqueta (o tag en inglés) en Git es una referencia especial que apunta a un commit específico en el historial de tu repositorio. Las etiquetas se utilizan comúnmente para marcar versiones importantes del proyecto, como lanzamientos (releases), puntos de referencia o hitos importantes en el desarrollo.

A diferencia de las ramas, que siguen avanzando con nuevos commits, las etiquetas son estáticas y no cambian. Esto significa que una vez que creas una etiqueta en un commit, esta etiqueta siempre apuntará a ese commit específico, lo cual es útil para señalar versiones de código que ya han sido "congeladas" o liberadas.

- ¿Cómo crear una etiqueta en Git?

  Git utiliza dos tipos principales de etiquetas: ligeras y anotadas.

  Una etiqueta ligera es muy parecida a una rama que no cambia - simplemente es un puntero a un commit específico. Ejemplo:

  ```
  $ git tag v1.0
  ```

  Sin embargo, las etiquetas anotadas se guardan en la base de datos de Git como objetos enteros. Tienen un checksum; contienen el nombre del etiquetador, correo electrónico y fecha; tienen un mensaje asociado; y pueden ser firmadas y verificadas con GNU Privacy Guard (GPG). Ejemplo:

  ```
  $ git tag -a v1.0 -m "Versión 1.0 lista para producción"
  ```

  Normalmente se recomienda que crees etiquetas anotadas, de manera que tengas toda esta información; pero si quieres una etiqueta temporal o por alguna razón no estás interesado en esa información, entonces puedes usar las etiquetas ligeras.

- ¿Cómo enviar una etiqueta a GitHub?

  Para subir una etiqueta a GitHub, deberemos usar el push y agregar una etiqueta en especial o todas las etiquetas creadas localmente.

  Para subir una etiqueta específica:

  ```
  $ git push origin v1.0
  ```

  Para subir todas las etiquetas locales:

  ```
  $ git push origin --tags
  ```

- ¿Qué es un historial de Git?

  El historial de Git es el registro de todos los cambios que se han hecho en un repositorio a lo largo del tiempo. Cada vez que hacemos un commit, Git guarda el contenido del cambio, el autor, la fecha y el mensaje que escribimos en la commit y un identificador único. Todo esto forma parte del historial donde se sabe cada cambio que se realizó, quien lo hizo, cuando y por qué.

- ¿Cómo ver el historial de Git?

  Puedes ver el historial con este comando:

  ```
  $ git log
  ```

  Esto te muestra los commits en orden, del más reciente al más antiguo

- ¿Cómo buscar en el historial de Git?

  Si quieres buscar un commit con cierta palabra en su mensaje:

```
$ git log --grep="palabra"
```

Para buscar commits que han modificado un archivo específico:

```
$ git log -- nombre_del_archivo
```

Para buscar commits en un rango de fechas específico:

```
$ git log --since="2024-01-01" --until="2024-01-31"
```

Para encontrar commits hechos por un autor específico:

```
$ git log --author="Nombre del Autor"
```

- ¿Cómo borrar el historial de Git?

  Borrar el historial de Git es una acción delicada y potencialmente destructiva, pero puede hacerse si tenés una razón válida.

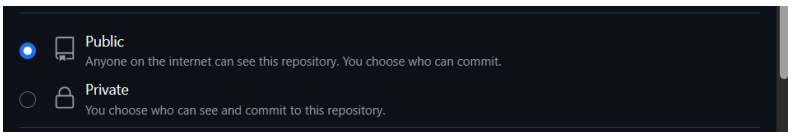
```
$ rm -rf .git
```

- ¿Qué es un repositorio privado en GitHub?

  Un repositorio privado en GitHub es un tipo de repositorio en el que el contenido solo es accesible para usuarios específicos que han sido autorizados. A diferencia de los repositorios públicos, donde cualquier persona puede ver y clonar el contenido, un repositorio privado limita el acceso a los colaboradores que tú elijas. Esto es útil para proyectos que contienen información sensible o que aún están en desarrollo y no deseas que estén disponibles públicamente.

- ¿Cómo crear un repositorio privado en GitHub?

  Al crear un repositorio en GitHub, nos ofrece activar la opcion "Private" para hacerlo privado.



- ¿Cómo invitar a alguien a un repositorio privado en GitHub?

Para invitar a alguien es sencillo, deberemois ir a "Settings" del proyecto, luego en "Collaborators" y clickeando el boton "Add people" para buscar el perfil de la persona . Selecciona el nivel de acceso que deseas otorgar: Read, Triage, Write, Maintain, o Admin. Haz clic en el botón "Add" para enviar la invitación.

- ¿Qué es un repositorio público en GitHub?

  Un repositorio público en GitHub es un repositorio cuyo contenido es accesible a cualquier persona en Internet. A diferencia de un repositorio privado, que está restringido a un grupo específico de colaboradores, un repositorio público permite que cualquier persona pueda ver, clonar y, si tienen los permisos adecuados, contribuir al proyecto.

- ¿Cómo crear un repositorio público en GitHub?

  Cuando creamos el repositorio, nos da la opción al igual que si queremos que sea privado

- ¿Cómo compartir un repositorio público en GitHub?

La forma más sencilla de compartir tu repositorio es proporcionar el enlace directo al mismo. Accede a tu repositorio, copia la URL de tu repositorio que se encuentra en un cuadro de texto que dice "<> Code”:

2) Realizar la siguiente actividad:

- Crear un repositorio.
  - Dale un nombre al repositorio.
  - Elije el repositorio sea público.
  - Inicializa el repositorio con un archivo.

- Agregando un Archivo
  - Crea un archivo simple, por ejemplo, "mi-archivo.txt".
  - Realiza los comandos git add . y git commit -m "Agregando mi-archivo.txt" en la línea de comandos.
  - Sube los cambios al repositorio en GitHub con git push origin main (o el nombre de la rama correspondiente).

- Creando Branchs

    - Crear una Branch
    - Realizar cambios o agregar un archivo
    - Subir la Branch

3) Realizar la siguiente actividad:

Paso 1: Crear un repositorio en GitHub

- Ve a GitHub e inicia sesión en tu cuenta.
- Haz clic en el botón "New" o "Create repository" para crear un nuevo repositorio.
- Asigna un nombre al repositorio, por ejemplo, conflict-exercise.
- Opcionalmente, añade una descripción.
- Marca la opción "Initialize this repository with a README".
- Haz clic en "Create repository".

Paso 2: Clonar el repositorio a tu máquina local

- Copia la URL del repositorio (usualmente algo como https://github.com/tuusuario/conflict-exercise.git).
- Abre la terminal o línea de comandos en tu máquina.
- Clona el repositorio usando el comando:

```
git clone https://github.com/tuusuario/conflict-exercise.git
```

- Entra en el directorio del repositorio:

```
cd conflict-exercise
```

Paso 3: Crear una nueva rama y editar un archivo

- Crea una nueva rama llamada feature-branch:

```
git checkout -b feature-branch
```

- Abre el archivo README.md en un editor de texto y añade una línea nueva, por ejemplo:

Este es un cambio en la feature branch.

- Guarda los cambios y haz un commit:

```
git add README.md
```

```
git commit -m "Added a line in feature-branch"
```

Paso 4: Volver a la rama principal y editar el mismo archivo

- Cambia de vuelta a la rama principal (main):

```
git checkout main
```

- Edita el archivo README.md de nuevo, añadiendo una línea diferente:

Este es un cambio en la main branch.

- Guarda los cambios y haz un commit:

```
git add README.md
```

```
git commit -m "Added a line in main branch"
```

Paso 5: Hacer un merge y generar un conflicto

- Intenta hacer un merge de la feature-branch en la rama main:

```
git merge feature-branch
```

- Se generará un conflicto porque ambos cambios afectan la misma línea del archivo README.md.

Paso 6: Resolver el conflicto

- Abre el archivo README.md en tu editor de texto. Verás algo similar a esto:

```
<<<<<<< HEAD
```

Este es un cambio en la main branch.

```
=======
```

Este es un cambio en la feature branch.

```
>>>>>>> feature-branch
```

- Decide cómo resolver el conflicto. Puedes mantener ambos cambios, elegir uno de ellos, o fusionar los contenidos de alguna manera.
- Edita el archivo para resolver el conflicto y guarda los cambios(Se debe borrar lo marcado en verde en el archivo donde estes solucionando el conflicto. Y se debe borrar la parte del texto que no se quiera dejar).
- Añade el archivo resuelto y completa el merge:

```
git add README.md
```

```
git commit -m "Resolved merge conflict"
```

Paso 7: Subir los cambios a GitHub

- Sube los cambios de la rama main al repositorio remoto en GitHub:

```
git push origin main
```

- También sube la feature-branch si deseas:

```
git push origin feature-branch
```

Paso 8: Verificar en GitHub

- Ve a tu repositorio en GitHub y revisa el archivo README.md para confirmar que los cambios se han subido correctamente.
- Puedes revisar el historial de commits para ver el conflicto y su resolución.