Programación 1

Practico 2: Git y GitHub

Ovelar Isaias Javier

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?
- ¿Cómo crear un repositorio en GitHub?
- ¿Cómo crear una rama en Git?
- ¿Cómo cambiar a una rama en Git?
- ¿Cómo fusionar ramas en Git?
- ¿Cómo crear un commit en Git?
- ¿Cómo enviar un commit a GitHub?
- ¿Qué es un repositorio remoto?
- ¿Cómo agregar un repositorio remoto a Git?
- ¿Cómo empujar cambios a un repositorio remoto?
- ¿Cómo tirar de cambios de un repositorio remoto?
- ¿Qué es un fork de repositorio?
- ¿Cómo crear un fork de un repositorio?

- ¿Cómo enviar una solicitud de extracción (pull request) a un repositorio?
- ¿Cómo aceptar una solicitud de extracción?
- ¿Qué es un etiqueta en Git?
- ¿Cómo crear una etiqueta en Git?
- ¿Cómo enviar una etiqueta a GitHub?
- ¿Qué es un historial de Git?
- ¿Cómo ver el historial de Git?
- ¿Cómo buscar en el historial de Git?
    - ¿Cómo borrar el historial de Git?
- ¿Qué es un repositorio privado en GitHub?
- ¿Cómo crear un repositorio privado en GitHub?
- ¿Cómo invitar a alguien a un repositorio privado en GitHub?
- ¿Qué es un repositorio público en GitHub?
- ¿Cómo crear un repositorio público en GitHub?
- ¿Cómo compartir un repositorio público en GitHub?

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.

1)

A. GitHub es una plataforma de desarrollo que permite gestionar, compartir y almacenar código fuente, se puede usar mediante web pero por defecto se usa mediante interfaces de texto (CLI), aún así existen herramientas que permiten su gestión de forma visual

B. Para crear un repositorio en GitHub, primero debemos registrarnos. Luego nos movemos a la página de **Dashboard** y hacemos click en el botón verde de **New**. Pedirá un nombre y verificara si el mismo está disponible, debemos definir su visibilidad (publico/privado), una descripción, si deseamos agregar un **readme**, licencia y si deseamos usar **.gitignore** (una lista negra de archivos que no queremos incluir en el proyecto).
C. Se pueden crear ramas mediante el comando branch

```
git branch nueva_rama
```

D. Para cambiar ramas se puede utilizar el comando **checkout** o **switch**. Tener en cuenta de que **checkout** tiene funcionalidades adicionales, pero para este propósito sirve.

```
git switch rama
git checkout rama
```

E. Para fusionar ramas se usa el comando **merge**. Primero nos ubicamos en la rama a la cual queremos traer código, luego con **merge** indicamos desde cual rama deseamos traer

```
git switch main
git merge rama_a_traer
```

F. Para realizar un commit, primero debemos incorporar archivos mediante **add** nombre_archivo o con “.” que actúa como wildcard para seleccionar todos los archivos que sufrieron cambios y seleccionarlos para el **stagging área**, que es un área previa al commit.
Luego con commit -m “mensaje” lo creamos y lo guarda en el

repositorio local.

```
git add .
git commit -m “mensaje”
```

G. Con el comando **push** enviamos el ultimo commit (por defecto) a github

```
git push
```

H. Un repositorio remoto es almacenamiento fuera del espacio físico del pc (nube).

I. Para agregar un respositorio remoto primero debemos inicializar un repositorio local con **init**. Luego debemos crear un respositorio en **github** y copiar la url, y se agregar con el comando **remote.**

```
git init
git remote add origin git@github.com:User/UserRepo.git
```

J. Mediante **push**.

```
git push rama_local rama_remota
```

K. Mediante **pull.** Nos sitiamos en que rama local deseamos traer los cambios del repositorio remoto.

```
git pull
```

L. Un fork es una copia de otro repositorio, que se almacena en nuestra cuenta de github. Normalmente un fork es una versión alternativa de un programa/proyecto.

M. En GitHub nos sitiamos en la página del proyecto a realizar el fork, y vamos al icono etiquetado **fork**. Luego elegimos en cual repositorio almacenar y el nombre (opcional) que deseamos.

N. Para hacer un PR debemos primero haber hecho un fork de un proyecto, realizar modificación y luego hacer **push** al repositorio remoto. Luego en la página del proyecto ir al apartado de **pull request** y completar la solicitud. La persona encargada de administrar el GitHub decidirá si se fusionara o no con el repositorio remoto.

O. En el apartado de **pull request** aparecerán notificaciones indicando que hay Código que puede fusionarse con el proyecto.

P. Una etiqueta en **git** es una etiqueta que permite hacer una referencia a una versión en particular del proyecto dentro del historial.

Q. Para realizar una etiqueta, luego de haber hecho un **commit**, agregamos la etiqueta

```
git tag -a nombreEtiqueta -m “mensaje”
```

Luego debemos hacer **push**

```
git push origin nombre-del-tag
```

R. El historial de git es una lista que indica las distintas versiones que fueron guardadas en el repositorio local. Indica fecha, mail de las personas que realizo la copia y comentarios. Vienen identificadas un equívocamente por un código hash.

S. Para ver el historial se utiliza **log**, recomendable usar con **–oneline** para que sea más conciso

```
git log --oneline
```

T. Hay varias formas de buscar en los commits, en función de lo que se desee encontrar:

buscar por fecha

```
git log –since=”2020-12-01” –until=”2020-12-12”
```

Buscar por fragmento de comentario

```
git log –grep=”mensaje a buscar”
```

U. Técnicamente no podemos borrar directamente el historial de git, pero se pueden hacer cosas que se asemejen.

Rebase: Compacta uno o varios commits en un solo,

Reset: Permite retroceder a un estado previo (commit):

Rest –soft, deshace el commit pero no modifica los archivos

```
git reset –soft HEAD~1
```

reset –hard, deshace el commit y modifica los archivos

```
git reset –hard HEAD~1
```

Se pueden eliminar múltiples commits con HEAD~n

Si se realizo push, luego de borrar los commits hay que volver a hacerlo

```
git push --force
```

V. Un repositorio privado es aquel en el cual tiene su visibilidad limitada, y esta solo disponible por invitación.

W. Para crear un repositorio privado, simplemente hay que marcar la opción de privado en GitHub al momento de crear un repositorio.

X. Para agregar personas a un repositorio remoto, ir a Settings -> Collaborators -> Add People, y buscar por email, nombre completo o usuario, para luego enviar una invitación.

Y. Un repositorio público es aquel que es visible por todos los usuarios de GitHub.

Z. Al crear un repositorio, por defecto es público.

AA. Para compartir un repositorio público, por lo general al ser publico tiene visible la url para poder clonar/hacer fork en el boto verde de **code**.