**Lección 7: Actividad para portafolios N° 7 “Git y Github”**

1. Necesidad de un repositorio de código fuente:

### R: Primero se responderá por qué es importante de utilizar un repositorio de código fuente, y luego sobre los beneficios que tiene el desarrollo colaborativo:

* **Historial y Versionamiento:**

Un repositorio de código mantiene un historial completo de cambios en el código a lo largo del tiempo. Permite revertir a versiones anteriores si es necesario. Facilita la identificación de quién hizo qué cambio y cuándo.

* **Colaboración:**

Proporciona un lugar centralizado para que múltiples desarrolladores trabajen en un proyecto simultáneamente. Evita conflictos al gestionar las contribuciones de cada desarrollador de manera organizada.

* **Seguimiento de Problemas y Soluciones:**

Ofrece herramientas para realizar un seguimiento de problemas y tareas pendientes. Facilita la colaboración y discusiones alrededor de problemas específicos.

* **Facilita la Distribución:**

Permite la distribución del código fuente entre diferentes entornos, ya sea entre desarrolladores, equipos o incluso en diferentes máquinas.

**Beneficios para el Desarrollo Colaborativo:**

* **Paralelismo y Concurrencia:**

Múltiples desarrolladores pueden trabajar simultáneamente en diferentes aspectos del proyecto sin interferencias.

* **Gestión de Conflictos:**

Git facilita la fusión de cambios, pero cuando hay conflictos, proporciona herramientas para resolverlos. La gestión eficiente de conflictos es crucial en proyectos colaborativos.

* **Revisión de Código:**

Facilita la revisión de código entre colaboradores antes de que se fusionen los cambios en la rama principal. Mejora la calidad del código y promueve buenas prácticas de desarrollo.

* **Trabajo Distribuido:**

Permite a los desarrolladores trabajar desde ubicaciones geográficas diferentes y contribuir al proyecto de manera eficiente.

* **Transparencia y Responsabilidad:**

Todos los cambios son rastreables y atribuibles a un desarrollador específico, lo que fomenta la transparencia y la responsabilidad.

* **Comentarios y Comunicación:**

GitHub, por ejemplo, ofrece funciones para comentarios y discusiones directamente en el código, lo que mejora la comunicación entre colaboradores.

Así que es útil usar un repositorio de código fuente, especialmente cuando se usa junto con sistemas de control de versiones como Git y plataformas como GitHub, facilita el desarrollo colaborativo eficiente y ordenado. Proporciona las herramientas necesarias para gestionar cambios, colabora de manera efectiva y mantiene la integridad del código a lo largo del tiempo.

1. **Instalación, configuración y comandos básicos:**

En Windows:

1. Descargar el instalador desde el sitio oficial de Git
2. Ejecutar el instalador y seguir las instrucciones predeterminadas.

* Para la configuración del Nombre y Correo:

1. Debemos abrir un terminal, Git Bash en Windows y ejecutar los siguientes comandos:

* git config --global user.name "Tu Nombre"
* git config --global user. email tu@email.com

Se debe reemplazar “Tu Nombre” con el nombre de la persona, y lo mismo para el correo electrónico.

* Para crear un repositorio Git y añadir archivos

1. Crear un nuevo directorio para un proyecto, y navegar hacia él en el terminal Git Bash

* mkdir mi\_proyecto
* cd mi\_proyecto

1. Para inicializar un nuevo repositorio en Git usamos el comando:

-git init

3. si queremos crear algunos archivos usamos el comando:

-git add .

**3. Commits y restauración de archivos:**

->Creamos un archivo y entramos a el con:

-mkdir mi\_proyecto

-cd mi\_proyecto

->Luego inicializamos con:

-git init

->Creamos un archivo y le añadimos información con:

- echo "Contenido inicial" > archivo.txt

->Unimos el archivo en la carpeta y empezamos escribiendo un “commit”:

-git add archivo.txt

-git commit -m "Primer commit - Contenido inicial"

->Si deseamos realizar otros cambios y agregar otro “commit” lo hacemos con:

-echo "Nueva línea" >> archivo.txt

-git add archivo.txt

-git commit -m "Segundo commit – nueva línea"

->Si deseamos realizar otro cambio más y agregar otro “commit” lo hacemos nuevamente con:

-echo "nueva modificación" >> archivo.txt

-git add archivo.txt

-git commit -m "Tercer commit – nueva modificación"

Si deseamos poder restaurar una versión anterior, usamos el comando:

-git log

Vemos los Commits para saber cuál versión deseamos restaurar:

-# Reemplaza 'hash\_del\_commit' con el hash del commit al que deseas regresar

-git checkout hash\_del\_commit archivo.txt

Ahora podemos verificar el archivo.txt con:

-git status

4**. Cambios de nombre:**

Si deseamos cambiar el nombre de un archivo en Git, usamos el comando “git mv” en lugar de simplemente cambiar el nombre del archivo manualmente.

“Git mv” realiza el cambio de nombre de manera que Git pueda rastrear adecuadamente la modificación. Esto permite que Git comprenda o actualice el cambio de nombre y así mantiene el historial del archivo de los commis,

**5. Ignorando archivos:**

Creamos un archivo “gitignore” en la raíz de un repositorio. Para editarlo lo abrimos con un editor de texto, luego le añadimos patrones para los archivos o directorios que deseamos ignorar:

“ \* ”, “ ? ”, también se usa “ # ” en las líneas que estén comenzadas con “ # “.

Por ejemplo:

# Ignorar archivos temporales

\*.log

\*.tmp

--------------------------------------------------------------------------

# Ignorar directorios de dependencias

node\_modules/

--------------------------------------------------------------------------

# Ignorar archivos de configuración local

config.local

--------------------------------------------------------------------------

Para que los cambios tengan efecto, guardamos el archivo “.gitignore”.

Git ignora archivos especificados con los caracteres o patrones mencionados, así no los incluye en el control de versiones.

6. **Ramas, uniones, conflictos y tags:**

Para crear una nueva rama usamos:

-git Branch nueva\_rama

Para cambiar a la nueva rama usamos:

-git checkout nueva\_rama

También se puede usar el comando:

-git switch nueva\_rama

Pero si deseamos unir ambas ramas, usamos el comando:

-git checkout -b nueva\_rama

También se puede usar el comando:

-git switch -c nueva\_rama

----------------------------------------------------------------------------------------------------------------

En caso de que deseáramos hacer cambios hacemos lo siguiente:

-echo “Nuevo contenido” > archivo.txt

-git add archivo.txt

-git commit -m “Nuevos cambios en Nueva\_rama”

----------------------------------------------------------------------------------------------------------------

Si deseamos fusionar ramas, hacemos lo siguiente:

-git checkout main

Y para fusionar:

-git merge nueva\_rama

----------------------------------------------------------------------------------------------------------------

Si deseamos etiquetar una versión específica usamos:

-git log -> verificamos el historial de Commits y así buscamos el “ # “ del commit deseado.

Para etiquetar usamos:

-git log -a v1.0.0 -m “versión 1.0.0”

7. **Stash y Rebase:**

Para almacenar temporalmente archivos que no están listos, usamos “ Git stash”:

-git stash sabe “archivo\_en stash”

Si deseamos aplicar esos archivos cuando sea necesario usamos:

-git stash apply

En caso de que tengamos varios Stashes, podemos especificar el stash que deseamos aplicar:

-git stash apply stash@{2}

-------------------------------------------------------------------------------------------------

Para hacer un rebase interactivo, escribimos la siguiente línea:

-git rebase -i HEAD-n

“n” es el númerode Commits que quieres incluir en el rebase.

Usamos el editor con la lista de Commits:

-pick abc123 primer commit

-pick efgh456 segundo commit

“pick” se puede reemplazar por “squash” (combinar), “edit”(editar), “drop”(eliminar), entre otras opciones.

Finalmente usamos:

-git push origin nombre\_rama - - force

8. **Repositorios remotos, Push y Pull:**

Para configurar un repositorio remoto en github,

1. Iniciar sesión en Github
2. Hacer click en “+”, luego en “Nuevo Repositorio”
3. Copiamos el URL del repositorio de Github,
4. Luego en git Bash escribimos:

-git remote add origin URL del repositorio

------------------------------------------------------------------------------------------

Para usar el Comando “PUSH”

-git add .

-git commit -m “detalle de los cambios”

Ahora hacemos un push al repositorio remoto:

-git push -u origin nombre de la rama

------------------------------------------------------------------------------------------

Para usar el Comando “PULL”

-git pull origin nombre de la rama

9. **Fetch v/s Pull:**

“Git fetch” y “Git pull” son comandos que permiten traer cambios desde un repositorio remoto, pero difieren en cómo se manejan esos cambios y cómo afectan al repositorio local.

Git fetch: este comando descarga los cambios del repositorio remoto a tu repositorio local, pero no los aplica automáticamente a tu rama actual.

Y se usa cuando deseamos obtener información actualizada sobre el estado del repositorio remoto sin fusionar los cambios en tu rama actual.

-git fetch origin

Git pull: este comando descarga los cambios desde el repositorio remoto llamado “origin” y los fusiona con rama local actual.

Se usa cuando deseamos actualizar la rama local y aplicar cambios automáticamente los cambios del remoto.

-git pull origin main

**10. Clonando un repositorio**

El comando “git clone” se usa para clonar(copiar) un repositorio Git existente en un nuevo directorio. Esto es muy útil cuando deseas obtener una copia completa de un proyecto, incluyendo todo su historial de cambios, ramas y archivos.

1. Obtener la URL del repositorio: en la página del repositorio en GitHub buscamos el botón “Code” o “Clone” para obtener la URL del repositorio.
2. Abrir git Bash: ejecutamos el comando:

-git clone “URL del repositorio”

3. Verificar el resultado: desúés de que el comando se complete, habrá una nueva carpeta con el nombre del repositorio el cual contendrá todos los archivos del repositorio clonado.

**11. Documentando un proyecto con Markdown**

**# Proyecto Ficticio**

**Bienvenido al proyecto ficticio. Este documento proporciona información esencial sobre el proyecto, su propósito y cómo comenzar.**

**## Descripción**

**Este proyecto ficticio es una aplicación web diseñada para [describir brevemente el propósito del proyecto].**

**## Características Principales**

**- [Lista de características importantes]**

**- [Otra característica clave]**

**- [Y otra característica]**

**## Instalación**

**A continuación, se detallan los pasos para instalar y ejecutar el proyecto en tu entorno local:**

**1. Clona este repositorio:**

**```bash**

**git clone https://github.com/usuario/proyecto-ficticio.git**

luego con git Bash usamos:

-cd proyecto-ficticio >> esto nos permite entrar en el directorio del proyecto

npm install >> instalamos las dependencias

npm start >> ejecutamos la aplicación.

Si deseamos contribuir al proyecto podemos hacer lo siguiente:.

1. Crea una nueva rama para tu contribución: git checkout -b mi-contribucion.
2. Realiza tus cambios y confirma: git commit -m "Añadir nueva característica".
3. Envía tus cambios al repositorio: git push origin mi-contribucion.
4. Abre un Pull Request en GitHub.

**12. Administración Pull Request:**

Pasos:

1. Clonar el repositorio con >> git clone http://github.com/user/repo.git

>> cd repo

1. Creamos una nueva rama >> git checkout -b nueva rama
2. Realizamos y confirmamos los cambios >> git add .

>> git commit -m “descripción cambios”

1. Luego enviamos los cambios al repositorio remoto

>> git push origin nueva rama

1. Abrimos GitHub y navegamos al Repositorio:
   1. vamos al repositorio en GitHub (por ejemplo, https://github.com/user/repo).
2. Cambiamos a la Nueva Rama:
   1. Hacemos clic en el menú desplegable de la rama principal y seleccionamos mi rama.
3. Hacemos clic en "Compare & pull request":
   1. Seleccionamos la opción "Compare & pull request" para iniciar el proceso de creación del Pull Request.
4. Completamos la Información del Pull Request**:**
   1. Proporcionamos una descripción clara de los cambios que hemos realizado.
   2. Mencionamos cualquier problema que estemos resolviendo.
5. Revisión del Pull Request:
   1. Los revisores revisarán los cambios antes de que se fusionen.
6. Hacer clic en "Create pull request":
   1. Una vez que hayas revisado tu Pull Request y estés listo para fusionar, haz clic en "Create pull request".
7. Para fusionar los cambios
   1. Si los revisores confirman los cambios y no hay conflictos, el propietario del repositorio puede fusionar el Pull Request haciendo clic en “Merge pull request”.
8. **Flujos de trabajo con GitHub:**

**Modelo de Bifurcación (Forking Workflow):**

En este modelo, los contribuidores crean bifurcaciones del repositorio principal, realizan sus cambios en sus propias bifurcaciones y luego abren Pull Request para que los cambios se integren en el repositorio principal.

Pasos a seguir:

1. Bifurcar el repositorio principal:
2. Clonamos la Bifurcación en el repositorio local

>> git clone https://github.com/contribuidor/repo.git

>> cd git

1. Creamos una rama y realizamos cambios:

>> git checkout -b mi-repo

1. Luego subimos la rama a la bifurcación

>> git push origin mi-repo

1. Abrir un Pull Request:

>> el contribuidor va a Github y abre un Pull Request desde su bifurcación a la rama principal del repositorio principal.

1. Revisión y Fusión:

>> Los colaboradores revisan el Pull Request, realizan comentarios y, si todo está bien, lo fusionan en el repositorio principal.

**Modelo de Rama por Característica (Feature Branch Workflow):**

Pasos a seguir:

1. Crear una rama para la característica:

>> git checkout -b mi-carpeta main

1. Realizamos cambios en la rama:

>> git add .

>> git commit

1. Subimos a la rama al repositorio remoto:

>> git push origin mi-carpeta

1. Revisamos y fusionamos en la rama principal:

>> git checkout main

>> git merge mi-carpeta