Juan Ignacio Ceballo

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 pensada para alojar repositorios de software en la nube y acceder a ellos de forma remota. Este se basa en Git, un software utilizado para gestionar versionado de código. Github permite llevar a cabo trabajo colaborativo entre varios desarrolladores en un mismo proyecto mediante *branches* (ramas) y *pull requests* (solicitudes de cambio).

- ¿Cómo crear un repositorio en GitHub?

  Para crear un repositorio en GitHub se debe acceder a la opción “Cerate New…” en el header de la home page de Github, y al desplegarse las opciones seleccionar “New Repository”. Se pasará a la page donde se pedirá información sobre el nuevo repositorio a crear tal como: Owner, nombre y descripción del repo. También se incluye una opción para elegir permitir acceso público al repo o, en cambio, dejarlo como privado. Incluidas algunas opciones más, al final del form, habrá un botón para enviar la información del repo y efectivamente crearlo.

- ¿Cómo crear una rama en Git?

  Una vez posicionado sobre la carpeta del repositorio en Git bash, se ingresa el comando: git branch [nombre de la nueva rama].

- ¿Cómo cambiar a una rama en Git?

  Para cambiar de rama en Git, se debe ingresar el comando: git checkout [nombre de la rama a la que se quiere apuntar]

- ¿Cómo fusionar ramas en Git?

  Primero se debe cambiar a la rama donde se quiere hacer el merge, para ello se utiliza el comando: git checkout [rama elegida donde se hará el merge].
  Luego se debe ingresar el comando: git merge [rama seleccionada para fusionar].

- ¿Cómo crear un commit en Git?

  Para crear cambios en el repositorio con un commit primero se deben trackear los cambios realizados con respecto a la versión anterior del repo, para ello se utiliza el comando: git add . (para agregar todos los cambios realizados) o git add [nombre y extensión del archivo a tener en cuenta]. Luego se utiliza el comando: git commit -m “” (entre comillas el nombre del commit o una breve descripción de los cambios que se hicieron).

- ¿Cómo enviar un commit a GitHub?

  Siguiendo el punto anterior, se debe utilizar el comando:

  ```
  git push -u origin main (si es el primer push que se hace)
  ```

  O sino:

  ```
  git push
  ```

- ¿Qué es un repositorio remoto?

  Un repositorio remoto es uno que no está alojado en la maquina local del usuario, sino que se aloja en alguna plataforma de repositorios como son GitHub o Gitlab

- ¿Cómo agregar un repositorio remoto a Git?

  Si no se tiene un repositorio en GitHub, se debe crear y conseguir la URL de éste. Luego se debe dirigir hacia la carpeta del repositorio en Git bash y darle inicio con el comando:

  ```
  git init
  ```

  Luego agregar el repositorio remoto con el comando:

  ```
  git remote add origin [URL del repositorio en GitHub]
  ```

- ¿Cómo empujar cambios a un repositorio remoto?

  Primero hay que trackear los cambios ya se con:

  ```
  git add .
  ```

  o especificando el archivo que se quiere cambiar/borrar. Luego se debe hacer el commit con:

git commit -m “Descripción del commit”

Como último se debe usar el comando:

git push origin main

- ¿Cómo tirar de cambios de un repositorio remoto?

  Se utiliza el comando:

  git pull origin main

- ¿Qué es un fork de repositorio?

  Es la copia de un repositorio de otro usuario de GitHub en la cuenta de uno la cual permite generar cambios, sin alterar el repositorio original.

- ¿Cómo crear un fork de un repositorio?

  Se debe acceder al repositorio del usuario del cual se desea hacer un fork. Una vez que seleccionado este, se debe acceder a la opción fork la cual llevará a un page donde se podrán seleccionar opciones sobre la nueva copia del repositorio original. Finalizado esto último, se podrá crear el fork deseado.

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

  Primero se debe clonar el repositorio forkeado en la maquina local con el siguiente comando en git bash:

  ```
  git clone [URL del repositorio forkeado]
  ```

  Luego se realizan los cambios y a continuación la serie de comandos que se vieron con anterioridad:

  ```
  git add .
  git commit -m “Cambios en…”
  git push origin main (o rama deseada)
  ```

  Luego en tu repositorio forkeado en GitHub, se debe deplegar la opciones de la etiqueta “Contribute” y seleccionar la opción “Open pull request”. Así, se desplegará un form para indicar nombre del request, una descripción y la opción de envío “Create pull request”.

- ¿Cómo aceptar una solicitud de extracción?

  Primero se debe seleccionar y abrir el repositorio en el cual se haya hecho el pull request. Una vez seleccionado, se deben visualizar los pull requests pertinentes al repositorio accediendo a la pestaña "Pull requests”. Se selecciona el pull request que se desea revisar y una vez abierto, se puede ver qué archivos han sido modificados en la pestaña "Files changed". También se pueden solicitar cambios y dejar comentarios. AL finalizar la revisión, si los cambios son correctos, se usa el botón "Merge pull request" y luego en "Confirm merge" para efectuar los cambios propuestos.

- ¿Qué es un etiqueta en Git?

  Una etiqueta (tag) en Git es un marcador que se usa para señalar versiones importantes en la historia de un repositorio, como versiones estables de un proyecto (v1.0, v2.0). Las etiquetas se usan comúnmente para gestionar versiones de software, especialmente en combinación con herramientas como GitHub Releases. Son como ramas que no cambian, una vez creada una etiqueta, no se pueden añadir más confirmaciones. Se pueden adjuntar a confirmaciones, ramas o incluso archivos individuales.

- ¿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. 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). Normalmente se recomienda crear etiquetas anotadas, para así tener toda esta información.
  Crear una etiqueta anotada en Git es sencillo. La forma más fácil de hacerlo es especificar la opción -a cuando ejecutas el comando git tag:

  ```
  git tag -a v1.4 -m 'my version 1.4'
  ```

  La opción -m especifica el mensaje de la etiqueta, el cual es guardado junto con ella. Si no se especifica el mensaje de una etiqueta anotada, Git abrirá el editor de texto para que se agregue dicho mensaje.

  La otra forma de etiquetar un commit es mediante una etiqueta ligera. Una etiqueta ligera no es más que el checksum de un commit guardado en un archivo, no incluye más información. Para crear una etiqueta ligera, no se deben incluir las opciones -a, -s ni -m:

  ```
  $ git tag v1.4-lw
  ```

- ¿Cómo enviar una etiqueta a GitHub?

  Por defecto, el comando git push no transfiere las etiquetas a los servidores remotos. Se deben enviar las etiquetas de forma explícita al servidor luego de que se hayan creado. Este proceso es similar al de compartir ramas remotas, se puede ejecutar git push origin [etiqueta].

  ```
  $ git push origin v1.5
  ```

  Se pueden enviar varias etiquetas a la vez usando la opción --tags del comando git push. Esto enviará al servidor remoto todas las etiquetas que aún no existen en él.

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

  Por lo tanto, cuando alguien clone o traiga información del repositorio, también obtendrá todas las etiquetas.

- ¿Qué es un historial de Git?

  El historial de Git es el registro de todos los cambios realizados en un repositorio, organizados en commits. Cada commit tiene información sobre:

  Quién hizo el cambio (autor).
  Cuando se hizo (fecha y hora).
  Qué se modificó (archivos cambiados).
  Un identificador único (hash del commit).
  Un mensaje descriptivo sobre el cambio.

- ¿Cómo ver el historial de Git?

  La primera opción utilizando el comando más básico para ver el historial de git es:

  ```
  git log
  ```

  Esto muestra una lista de commits en orden cronológico inverso (los más recientes primero).

  También se puede utilizar:

  ```
  git log –oneline
  ```

  Cada commit se mostrará en una sola línea, con su hash corto y mensaje.

  También se puede ver el historial con más detalles utilizando:

  ```
  git log --graph --decorate --all –oneline
  ```

  Esto muestra el historial con un gráfico de ramas, útil para ver la estructura del proyecto.

- ¿Cómo buscar en el historial de Git?

  Si se quiere encontrar commits que contengan una palabra en su mensaje:

  ```
  git log --grep=”[palabra clave]”
  ```

  Ejemplo:

  ```
  git log --grep="bug"
  ```

  Esto muestra todos los commits que mencionan "bug" en su mensaje.

  Si se necesita ver solo los commits de un desarrollador específico:

  ```
  git log --author="[Nombre del autor]"
  ```

  Para buscar commits en un rango de fechas se utiliza:

  ```
  git log --since="2024-01-01" --until="2024-03-30"
  ```

  En este ejemplo se muestran todos los commits entre el 1 de enero y el 30 de marzo de 2024.

  Si se necesita ver el historial de un archivo en particular:

  ```
  git log -- archivo.txt
  ```

  Esto muestra todos los commits que modificaron archivo.txt.

Si se sabe una línea de código específica se busca ver quién la modificó:

```
git blame archivo.txt
```

Con esto se muestra línea por línea quién hizo cada cambio y en cual commit.

Para buscar un commit por su hash parcial, es decir, si se teine una parte del identificador (hash) de un commit, se utiliza:

```
git log --oneline | grep "abc123"
```

Esto encuentra el commit cuyo hash comienza con "abc123".

- ¿Cómo borrar el historial de Git?

En Git, el historial de commits es inmutable, lo que significa que no se puede borrar de forma directa sin afectar el repositorio. Sin embargo, existen algunas formas de eliminarlo o reescribirlo según lo que se necesite hacer.

Si solo se quiere reiniciar el historial sin eliminar los archivos actuales, se pueden hacer lo siguientes pasos:

```
rm -rf .git  # Borra la carpeta de Git permanentemente
git init     # Inicializa un nuevo repositorio vacío
git add .
git commit -m "Reiniciando el historial"
git remote add origin https://github.com/usuario/repositorio.git
git push -f origin main  # Sobrescribe el historial remoto
```

Este método borra todo el historial y crea un nuevo inicio para el repositorio.

Si solo se busca eliminar los últimos commits sin borrar todo el historial:

Eliminar el último commit (manteniendo los cambios en el área de trabajo):

```
git reset --soft HEAD~1
```

Eliminar el último commit y los cambios:

```
git reset --hard HEAD~1
```

Eliminar varios commits (ejemplo: los últimos 3):

```
git reset --hard HEAD~3
```

Si ya se subieron los commits al repositorio remoto, se deberá forzar la actualización:

```
git push origin main –force
```

Si se quiere reescribir o modificar varios commits anteriores:

```
git rebase -i HEAD~3  # Edita los últimos 3 commits
```

Esto abrirá un editor donde se pueden modificar, combinar o eliminar commits.

Si se busca eliminar un commit en particular pero no los demás:

```
git rebase -i <hash-del-commit-anterior>
```

En el editor, se debe cambiar pick por drop en el commit que se desea eliminar.

Si ya subiste los commits y quieres eliminar todo el historial en GitHub:

```
git push origin –mirror
```

- ¿Qué es un repositorio privado en GitHub?

  Un repositorio privado en GitHub es un repositorio que el usuario que lo creó y las personas que éste invite pueden ver y acceder. A diferencia de los repositorios públicos, los privados no son visibles para otros usuarios en GitHub. Como se mencionó, solo es accesible para usuarios. Estos no aparecen en búsquedas públicas ni en el perfil del creador a menos que éste los comparta. Son compatibles con todas las funciones de GitHub, como git push, git pull y GitHub Actions. Cabe mencionar, que resultan ideales para proyectos personales, privados o comerciales.

- ¿Cómo crear un repositorio privado en GitHub?

  Una vez logueado en GitHub, se debe utilizar la opción "New repository". Se deben completar el form con la información mencionada anteriormente y llegado a la sección "Visibility", seleccionar "Private". Para finalizar el envío del form y efectuar le creación del repositorio se utiliza el botón "Create repository").

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

  En la página del repositorio en GitHub, se debe entra en la pestaña **"Settings".** Luego, en el menú lateral, seleccionar **"Collaborators".** Hacer click en **"Add people"**. Aquí se ingresa el nombre de usuario o correo de la persona y se clickea en **"Add"**. Con esto, GitHub enviará una invitación que el colaborador deberá aceptar.

- ¿Qué es un repositorio público en GitHub?

  Un repositorio público en GitHub es un repositorio que cualquier persona puede ver y acceder, sin necesidad de permisos especiales. Es ideal para proyectos de código abierto, documentación o cualquier contenido se desee compartir con la comunidad. Cualquier persona puede clonarlo o bifurcarlo (fork). Solo los propietarios y colaboradores pueden hacer cambios directamente, pero, puede recibir contribuciones de la comunidad a través de pull requests.

- ¿Cómo crear un repositorio público en GitHub?

  Al igual que con la el repositorio privado, se debe estar logueado en GitHub. Se selecciona la opción **"New repository"**. Se introduce la información que pide el form y en **"Visibility"**, se selecciona **"Public".** Para finalizar, enviar el form y crear el repositorio, se clickea en **"Create repository"**.

- ¿Cómo compartir un repositorio público en GitHub?

  Existen varias formas de hacerlo.

  La forma más sencilla es copiar la URL del repositorio y enviarla por correo, chat o redes sociales.

  Si alguien quiere trabajar el código, puede clonarlo con:

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

  Esto descargará una copia del repositorio en su computadora.

  Si se busca que alguien más tenga permisos para modificar el código, se puede agregarlo como colaborador siguiendo los pasos mencionados en puntos anteriores. Este método es útil si se busca que otros editen el repositorio sin necesidad de forks o pull requests.

  GitHub también ofrece un botón para compartir directamente en Twitter, LinkedIn y Facebook.

  Si se busca que otros contribuyan al proyecto sin darles acceso directo:

  Un usuario hace un fork del repositorio.
  Clona su propia copia con git clone.
  Realiza cambios en una nueva rama y los sube.
  Envía un Pull Request para que dichos cambios se revisen e integren.

  Este método es ideal para proyectos de código abierto.

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 Branches
    - 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.