

TRABAJO PRÁCTICO UNIDAD 2

Santiago Daniel Vizgarra Soria

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

- ¿Qué es GitHub?

GitHub es una plataforma que proporciona alojamiento de repositorios con control de versiones, permitiendo a los desarrolladores almacenar y gestionar sus proyectos de software. Es una herramienta gratuita y de código abierto que facilita la colaboración en equipo, el intercambio de código y el trabajo conjunto de manera eficiente.

-¿Cómo crear un repositorio en GitHub?

Para crear un repositorio en GitHub, seguimos estos pasos:

1. Inicia sesión en tu cuenta de GitHub.
2. En la página principal, haz clic en el botón verde "New" o "Nuevo" (generalmente en la esquina superior derecha).
3. Asigna un nombre a tu repositorio en el campo "Repository name".
4. Opcionalmente, agrega una descripción.
5. Elige si quieres que el repositorio sea **público** o **privado**.
6. Haz clic en "Create repository".

- ¿Cómo crear una rama en Git?

Para crear una rama en Git, sigue estos pasos:

1. Abre tu terminal o línea de comandos.
2. Navega al directorio de tu repositorio local usando "CD" (si no estás ya en él).
3. Ejecuta el siguiente comando para crear una nueva rama:

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

-¿Cómo cambiar a una rama en Git?

Para cambiarte a la nueva rama, usa el siguiente comando: `git checkout nombre-de-la-rama`

- ¿Cómo fusionar ramas en Git?

Para fusionar ramas en Git, el proceso consiste en incorporar los cambios de una rama a otra.

Primero, debes estar en la rama en la que deseas integrar los cambios. Por lo general, esta será la rama principal, como master o main, o cualquier otra rama en la que quieras combinar los cambios. Por ejemplo, si quieres fusionar la rama ramaNueva a la rama master, lo primero es cambiar a la rama master:

```
"$ git checkout master"
```

ahora usamos utiliza el comando "git merge" para fusionar los cambios de la rama "ramaNueva" en la rama en la que estás:

```
"$ git merge ramaNueva"
```

- ¿Cómo crear un commit en Git?

Crear un commit en Git es el proceso mediante el cual se guardan los cambios realizados en el repositorio. Un commit captura el estado actual del código en un momento dado y lo almacena en el historial del proyecto.

Los cambios realizados los cargamos con git add y luego creamos un commit, con el cual dejamos un mensaje para indicar que cambios fueron realizados.

```
git commit -m "Mensaje del commit"
```

-¿Cómo enviar un commit a GitHub?

los pasos a seguir son :

1. Clonar o inicializar un repositorio.
2. Añadir archivos: `git add .`
3. Hacer un commit: `git commit -m "mensaje"`
4. Enviar los cambios a GitHub: `git push origin main`

-¿Qué es un repositorio remoto?

Un repositorio remoto es una versión de tu proyecto almacenada en un servidor en línea, como GitHub, GitLab o Bitbucket. Permite que varias personas trabajen en el mismo proyecto desde diferentes lugares, sincronizando cambios a través de operaciones como `git push` (enviar cambios) y `git pull` (obtener cambios). Es útil para colaboración en equipo y respaldo de datos.

-¿Cómo agregar un repositorio remoto a Git?

Utiliza el comando git remote add para vincular tu repositorio local con un repositorio remoto y asignar un nombre a ese remoto:

```
git remote add "nombre" / https://github.com/usuario/repositorio.git
```

verificamos que repositorio remoto se haya agregado correctamente:

```
git remote -v
```

- ¿Cómo empujar cambios a un repositorio remoto?

Los cambios que hayamos realizado con "git add", seguido de un commit

usamos git push origin nombre_de_la_rama, para empujar los cambios al repositorio remoto

-¿Cómo tirar de cambios de un repositorio remoto?

Usando el comando git pull para descargar y fusionar los cambios del repositorio remoto con tu rama local:

```
git pull origin nombre_de_la_rama / git pull origin main
```

¿Qué es un fork de repositorio?

Un **fork** de un repositorio es una copia personal de un repositorio que se encuentra en una plataforma como GitHub. Este proceso te permite hacer cambios y experimentar con el código de un proyecto sin afectar al repositorio original.

-¿Cómo crear un fork de un repositorio?

Para crear un fork de un repositorio en GitHub, sigue estos pasos:

Ve al repositorio que deseas hacer fork:

Accede al repositorio original en GitHub.

Haz clic en el botón "Fork":

En la parte superior derecha de la página del repositorio, verás un botón llamado "Fork". Haz clic en él.

Selecciona tu cuenta:

GitHub te pedirá que elijas dónde quieres crear el fork (en tu cuenta personal o en una organización, si tienes acceso a alguna). Elige tu cuenta personal.

Espera a que se complete el proceso:

GitHub creará una copia del repositorio en tu cuenta. Este proceso puede tomar unos segundos.

Una vez creado el fork, tendrás una copia del repositorio original en tu cuenta, y podrás hacer cambios de manera independiente.

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

Primero se realiza un Fork del repositorio (si no tienes permisos):

- Crea una copia del repositorio en tu cuenta.

Clona tu "fork" localmente:

- Descarga la copia a tu computadora.

Crea una rama:

- Haz una rama separada para tus cambios.

Haz tus cambios:

- Modifica los archivos necesarios.

Confirma los cambios:

- Guarda los cambios con `git add` y `git commit`.

Sube tu rama:

- Envía los cambios a tu "fork" en GitHub con `git push`.

Crea la "pull request":

- En GitHub, compara tus cambios y crea la solicitud.

Espera la revisión y fusión:

- revisarán tu solicitud y, si aprueban, fusionarán tus cambios.

- ¿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 clicar en Merge pull request y de esta manera sumará a su repositorio los cambios que hizo un usuario (en modo de ayuda).

-¿Qué es una etiqueta en Git?

En Git, una etiqueta (tag) es una forma de marcar un punto específico en la historia de tu repositorio. Es como una instantánea que puedes usar para referirte a una versión particular de tu proyecto.

- ¿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.

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 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.

`git tag v1.0 (ligera) | git tag -a v1.0 -m "Versión 1.0" (anotadas)`

¿Cómo enviar una etiqueta a GitHub?

Debes enviar las etiquetas de forma explícita al servidor luego de que las hayas creado. Este proceso es similar al de compartir ramas remotas - puedes ejecutar `git push origin [etiqueta]` . Si quieres enviar varias etiquetas a la vez, puedes usar la opción `--tags` del comando `git push` .

-¿Qué es un historial de Git?

El historial de Git es una secuencia de todos los commits realizados en un repositorio a lo largo del tiempo. Este historial muestra cómo ha evolucionado un proyecto, incluyendo los cambios que se han hecho en los archivos, quién los hizo, cuándo los hizo y qué mensaje acompañó a cada cambio.

-¿Cómo ver el historial de Git?

Para ver el historial de Git, usamos `git log` el cual mostrará una lista de commits con detalles como el autor, fecha y mensaje.

Para ver el historial de una rama específica: `git log nombre-de-la-rama`

Historial con una vista más resumida: `git log --oneline`

Historial con cambios en los archivos: `git log --stat`

-¿Cómo buscar en el historial de Git?

Para buscar en el historial de commits de Git, puedes utilizar varios comandos y opciones que te permiten

filtrar y localizar commits específicos.

Para buscar commits que contengan una palabra o frase específica en el mensaje de commit, usamos `git log` con la opción `--grep`: `git log --grep="palabra clave"`

Para buscar commits que han modificado un archivo específico, usa `git log` seguido del nombre del archivo: `git log -- nombre_del_archivo`

Para buscar commits en un rango de fechas específico, usa las opciones `--since` y `--until`:

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

Para encontrar commits hechos por un autor específico, usa `--author`:

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

-¿Cómo borrar el historial de Git?

Borrar el historial de Git puede ser riesgoso, ya que implica eliminar registros importantes de los cambios en el repositorio. Sin embargo, si necesitas hacerlo por razones específicas (como eliminar información sensible o limpiar un repositorio), puedes seguir uno de los métodos a continuación:

- `git reset ->` Quita del stage todos los archivos y carpetas del proyecto.
- `git reset nombreArchivo ->` Quita del stage el archivo indicado.
- `git reset nombreCarpeta/ ->` Quita del stage todos los archivos de esa carpeta.
- `git reset nombreCarpeta/nombreArchivo ->` Quita ese archivo del stage (que a la vez está dentro de una carpeta).
- `git reset nombreCarpeta/*.extensión ->` Quita todos los archivos que cumplan con la condición indicada previamente dentro de esa carpeta del stage.

-¿Qué es un repositorio privado en GitHub?

Un repositorio privado en GitHub es un repositorio cuyo acceso está restringido solo a personas específicas o equipos autorizados. A diferencia de los repositorios públicos, donde cualquier persona puede ver y clonar el código, un repositorio privado sólo es accesible para aquellos que han sido invitados o tienen permisos explícitos para acceder a él.

¿Cómo crear un repositorio privado en GitHub?

Crea un nuevo repositorio:

Ve a GitHub y haz clic en el botón "New" para crear un nuevo repositorio.

Configura la privacidad:

Al crear el repositorio, en la opción "Repository visibility", selecciona "Private".

Al estar en privado solo las personas o equipos autorizados pueden ver o modificar el repositorio.

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

Para invitar a alguien:

Después de crear el repositorio, puedes invitar personas desde la pestaña "Settings" -> "Manage access" y agregar personas a las que deseas otorgar acceso

-¿Qué es un repositorio público en GitHub?

Un repositorio público en GitHub es un repositorio cuyo código y contenido son accesibles para cualquier persona en Internet. Cualquier persona puede ver, clonar, bifurcar (fork) y contribuir al proyecto, siempre que no haya restricciones adicionales establecidas por el propietario del repositorio.

-¿Cómo crear un repositorio público en GitHub?

Crea un nuevo repositorio:

Ve a GitHub y haz clic en el botón "New" para crear un nuevo repositorio.

Configura la privacidad:

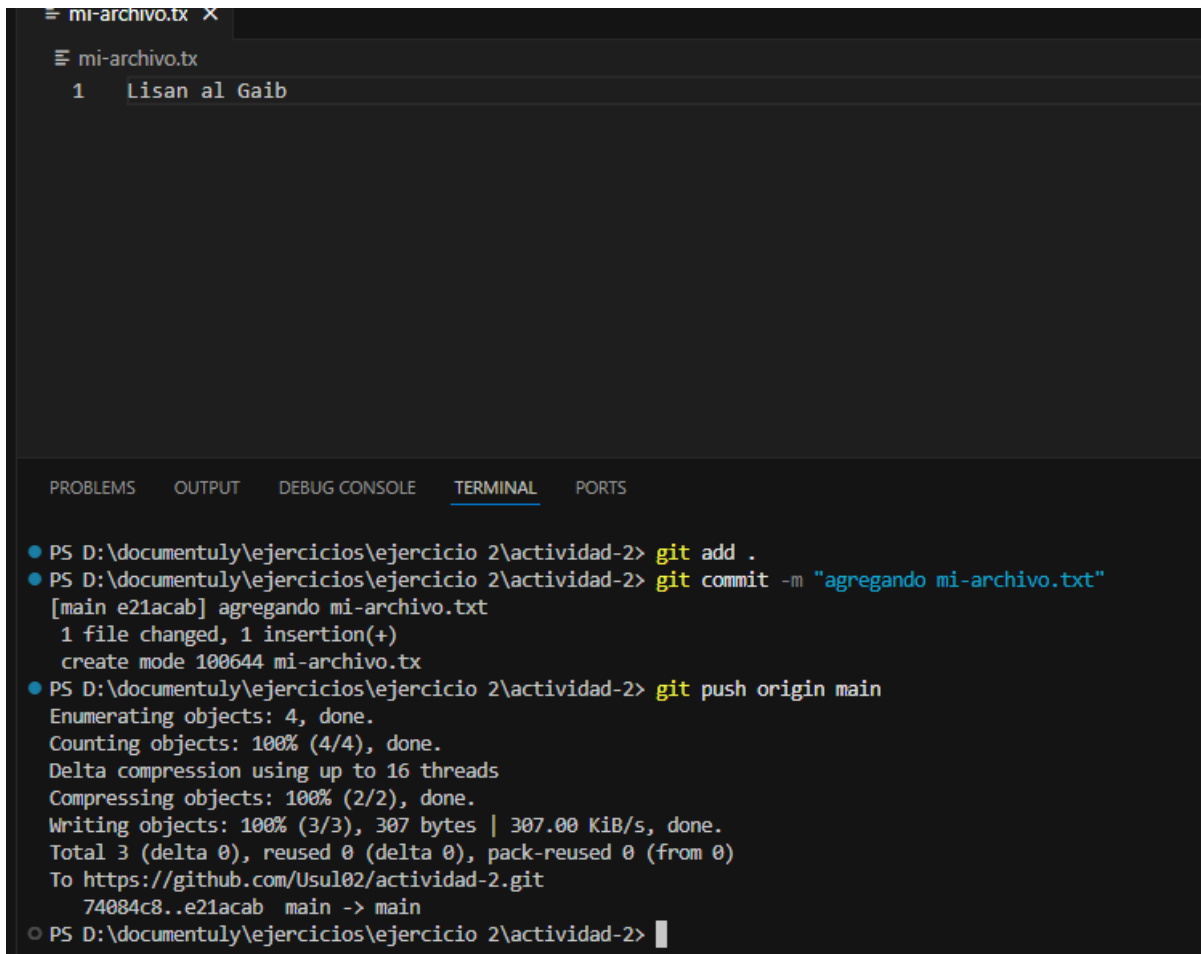
Al crear el repositorio, en la opción "Repository visibility", selecciona "Public".

El repositorio será público y estará disponible para cualquier persona en GitHub.

-¿Cómo compartir un repositorio público en GitHub?

Accede a tu repositorio, copia la URL de tu repositorio que se encuentra en un cuadro de texto que dice "<>Code"

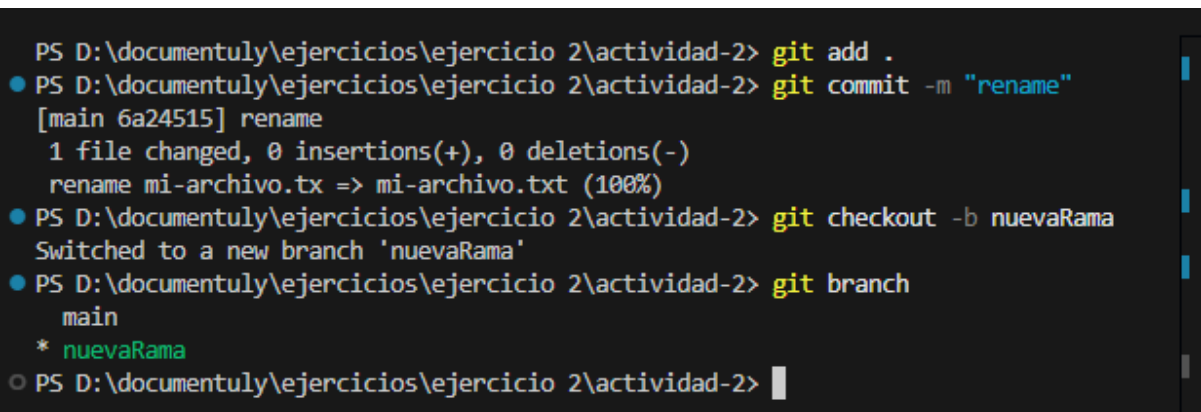
2) Realizar la siguiente actividad:



```
mi-archivo.tx X
mi-archivo.tx
1 Lisan al Gaib

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

● PS D:\documentuly\ejercicios\ejercicio 2\actividad-2> git add .
● PS D:\documentuly\ejercicios\ejercicio 2\actividad-2> git commit -m "agregando mi-archivo.txt"
[main e21acab] agregando mi-archivo.txt
1 file changed, 1 insertion(+)
create mode 100644 mi-archivo.tx
● PS D:\documentuly\ejercicios\ejercicio 2\actividad-2> git push origin main
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 16 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 307 bytes | 307.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/Usul02/actividad-2.git
74084c8..e21acab main -> main
○ PS D:\documentuly\ejercicios\ejercicio 2\actividad-2>
```



```
PS D:\documentuly\ejercicios\ejercicio 2\actividad-2> git add .
● PS D:\documentuly\ejercicios\ejercicio 2\actividad-2> git commit -m "rename"
[main 6a24515] rename
1 file changed, 0 insertions(+), 0 deletions(-)
rename mi-archivo.tx => mi-archivo.txt (100%)
● PS D:\documentuly\ejercicios\ejercicio 2\actividad-2> git checkout -b nuevaRama
Switched to a new branch 'nuevaRama'
● PS D:\documentuly\ejercicios\ejercicio 2\actividad-2> git branch
main
* nuevaRama
○ PS D:\documentuly\ejercicios\ejercicio 2\actividad-2>
```



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

● PS D:\documentuly\ejercicios\ejercicio 2\actividad-2> git branch
* main
  nuevaRama
● PS D:\documentuly\ejercicios\ejercicio 2\actividad-2> git add .
● PS D:\documentuly\ejercicios\ejercicio 2\actividad-2> git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   mi-archivo.txt

○ PS D:\documentuly\ejercicios\ejercicio 2\actividad-2> |
```

3) Realizar la siguiente actividad:

```
PS C:\Users\TIAGO\conflict-exercise> git push origin feature-branch
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
remote:
remote: Create a pull request for 'feature-branch' on GitHub by visiting:
remote:   https://github.com/Usul02/conflict-exercise/pull/new/feature-branc
remote:
To https://github.com/Usul02/conflict-exercise.git
 * [new branch]      feature-branch -> feature-branch
● PS C:\Users\TIAGO\conflict-exercise> git add .
● PS C:\Users\TIAGO\conflict-exercise> git commit -m "Resolved merge conflict"
[main 5293a5b] Resolved merge conflict
 1 file changed, 3 insertions(+), 7 deletions(-)
● PS C:\Users\TIAGO\conflict-exercise> git push origin main
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 16 threads
Compressing objects: 100% (1/1), done.
Writing objects: 100% (3/3), 281 bytes | 281.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/Usul02/conflict-exercise.git
 d936091..5293a5b  main -> main
● PS C:\Users\TIAGO\conflict-exercise> git push origin feature-branch
Everything up-to-date
● PS C:\Users\TIAGO\conflict-exercise> |
```

