

TRABAJO PRACTICO N° 2 - Programación I

1) CONTESTAR LAS SIGUIENTES PREGUNTAS UTILIZANDO LAS GUÍAS Y DOCUMENTACIÓN PROPORCIONADA (DESARROLLAR LAS RESPUESTAS) :

1. ¿Qué es GitHub?

Es una plataforma donde puedes guardar y compartir proyectos de código (o cualquier archivo).
Te permite

- Almacenar proyectos (repositorios) en la nube.
- Trabajar en equipo (varias personas pueden editar el mismo código).
- Controlar versiones (sabes quién hizo cambios y cuándo).
- Publicar proyectos (como páginas web o apps).
- Contribuir a software libre (como WordPress o Linux).

2. ¿Cómo crear un repositorio en GitHub?

1. Entrar a github.com, crear una cuenta e iniciar sesión.

2. Crear el repositorio

- Haz clic en el botón verde "+ New" (arriba a la derecha, junto a tu foto de perfil).
- Rellena los datos:
 - ❖ Repository name: Ponle un nombre (ej: mi-primer-proyecto).
 - ❖ Usa letras minúsculas y guiones (-) en lugar de espacios.
 - ❖ Description: Descripción opcional (ej: "Mi primera calculadora en Python").
 - ❖ Public/Private: Elige Public (gratis y visible para todos) o Private (solo tú lo ves).
 - ❖ Initialize this repository with a README: Marca esta casilla si quieres un archivo inicial de presentación (recomendado para principiantes).
- Haz clic en Create repository (botón verde al final).

3. ¡Listo! Ya tenemos el repositorio

- Veremos una página con un nuevo proyecto vacío. Aquí podemos:
- Subir archivos manualmente (arrastrándolos a la página).
- Clonarlo en nuestra computadora (para trabajar localmente, usando Git)

3. ¿Cómo crear una rama en Git?

```
git branch nombre-de-la-rama # Crea la rama
git checkout nombre-de-la-rama # Te mueve a ella
```

La rama master (o main, como ahora se llama por defecto en GitHub) es la rama principal de un proyecto. Es como la versión "oficial" y estable del código, donde todo debe funcionar correctamente.

Es como el tronco de un árbol: De aquí salen otras ramas (branches) para desarrollar nuevas funciones sin afectar el proyecto principal.

Nunca se trabaja directamente en master: Para hacer cambios, creas ramas nuevas y luego se las fusiona cuando estén listas.

4. ¿Cómo cambiar a una rama en Git?

El comando para cambiar entre ramas es el git checkout

```
git checkout -b nombre-de-la-rama
```

Qué hace:

- Crea una rama nueva con el nombre que le pongas.
- Automáticamente te mueve a esa rama para que empieces a trabajar.

5. ¿Cómo fusionar ramas en Git?

Fusionar (merge) une los cambios de una rama con otra (generalmente con la rama principal main o master).

Pasos Básicos

Cambiarse a la rama que recibirá los cambios (normalmente main):

```
git checkout master
```

```
git merge ramanueva
```

Este comando incorpora los cambios de nuevaRama en master.

6. ¿Cómo crear un commit en Git?

Un commit es un guardado de los cambios en tu proyecto. Es como una foto instantánea de los archivos en un momento determinado. Cada commit tiene:

- Un identificador único (hash como abc123).
- Un mensaje que explica qué cambiaste (ej: "Arreglé el botón de login").
- Autor y fecha.

Pasos para crear un commit:

Prepara los archivos (añádelos al staging area):

- `git add nombre-del-archivo` # Para un archivo específico
- `git add .` # Para todos los archivos modificados

Crea el commit con un mensaje claro:

- `git commit -m "Mensaje que describe los cambios"`

Ejemplo:

```
git commit -m "Agregué función de búsqueda"
```

Comandos útiles

Ver commits recientes: `git log`

Corregir el último commit (si olvidaste algo): `git commit --amend -m "Nuevo mensaje"`

7. ¿Cómo enviar un commit a GitHub?

Si es la primera vez trabajando en el proyecto (no tenemos los archivos en la PC):

```
git clone https://github.com/tu-usuario/tu-repo.git
cd tu-repo # Entrás a la carpeta del proyecto
```

Ejemplo:

```
git clone https://github.com/ejemplo/mi-proyecto.git
cd mi-proyecto
```

Para enviar un commit a GitHub

1. Primero, crea tu commit local (si no lo has hecho):

```
git add .          # Prepara todos los cambios
git commit -m "Mensaje claro" # Ej: "Arreglé el error de login"
```

2. Sube (push) el commit a GitHub:

```
git push origin nombre-de-tu-rama
```

Ejemplo:

```
git push origin main      # Si trabajas en la rama main
git push origin feature/login # Si es una rama secundaria
```

3. Verifica en GitHub:

Entra a tu repositorio en github.com
Haz clic en la pestaña "Commits" para ver tu cambio subido.

8. ¿Qué es un repositorio remoto?

Un repositorio remoto es una copia del proyecto alojada en un servidor en internet (como GitHub). Permite guardar el historial de cambios, compartir el código con otros y actuar como respaldo. Ejemplo: Si mi computadora falla, puedo recuperar el proyecto desde el remoto

Los repositorios remotos centralizan el trabajo del equipo, evitan pérdida de datos y permiten resolver conflictos entre versiones. Además, facilitan la revisión de código mediante herramientas como pull requests

9. ¿Cómo agregar un repositorio remoto a Git?

Ejecutar en la terminal (dentro de tu proyecto local):

```
git remote add origin URL-del-repositorio
```

origin: Nombre por defecto del remoto (puedes cambiarlo).

Ejemplo:

```
git remote add origin https://github.com/usuario/mi-proyecto.git
```

Verificar que se agregó correctamente

```
git remote -v
```

Mostrará algo como:

```
origin https://github.com/usuario/mi-proyecto.git (fetch)
```

```
origin https://github.com/usuario/mi-proyecto.git (push)
```

4. Envía tu código al remoto (primer push)

```
git push -u origin main
```

-u: Guarda origin main como referencia para futuros push

10. ¿Cómo empujar cambios a un repositorio remoto?

Pasos Básicos

- Prepara tus cambios (agrégalos al staging area):

```
git add .          # Todos los archivos modificados
git add archivo.txt # Solo un archivo específico
```
- Crea un commit (registra los cambios localmente):

```
git commit -m "Descripción clara de los cambios"
```
- Sube los cambios al remoto (a GitHub/GitLab/etc.):

```
git push origin nombre-de-la-rama
```
- Ejemplo:

```
git push origin main      # Para la rama principal
git push origin feature/login # Para una rama secundaria
```

11. ¿Cómo tirar de cambios de un repositorio remoto?

Comando Básico

```
git pull origin nombre-de-la-rama
```

(Descarga cambios y los fusiona automáticamente con tu rama local).

12. ¿Qué es un fork de repositorio?

Un fork es una copia personal de un repositorio de GitHub (o GitLab/Bitbucket) que se guarda en tu propia cuenta.

Ejemplo cotidiano

Imagina un libro en una biblioteca pública:

Fork = Hacer una fotocopia para añadir tus propias notas.

Pull Request = Enviar tus notas al autor por si quiere incluirlas en el libro original.

13. ¿Cómo crear un fork de un repositorio?

En GitHub:

Ir al repo que quieres copiar (ej: <https://github.com/usuario/proyecto>).

Hacer clic en "Fork" (esquina superior derecha).

Botón Fork en GitHub

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

Pasos

1. Haz un Fork del Repositorio

- Entra al repo original en GitHub (ej: github.com/autor/proyecto).
- Haz clic en "Fork" (arriba a la derecha). Esto crea una copia en tu cuenta.

2. Clona tu Fork (a tu computadora):

```
git clone https://github.com/tu-usuario/proyecto.git  
cd proyecto
```

3. Crea una Rama Nueva (para tus cambios):

```
git checkout -b mi-mejora
```

4. Haz tus Cambios

- Edita archivos, guarda, y haz commit:

```
git add .  
git commit -m "Descripción clara de los cambios"
```

5. Sube los Cambios a tu Fork:

```
git push origin mi-mejora
```

6. Abre el Pull Request

1. Entra a tu fork en GitHub (en el navegador).
2. Verás un botón "Compare & pull request" (GitHub lo sugiere automáticamente).
3. Completa:
 - Título: Breve y descriptivo (ej: "Arregla el error en el login").
 - Descripción: Detalla qué cambiaste y por qué.
4. Haz clic en "Create pull request".

15. ¿Cómo aceptar una solicitud de extracción?

1. Entrar al PR (la solicitud de cambios)

Ir a tu proyecto en GitHub.

Hacer clic en la pestaña "Pull requests" (arriba).

Seleccionar el PR que quieres revisar (verás el título y quién lo envió).

2. Revisar los cambios

Hacer clic en "Files changed" para ver qué modificó la persona:

Líneas en verde: Son cosas nuevas que agregó.

Líneas en rojo: Son cosas que eliminó.

Si todo parece correcto, pasar al siguiente paso.

3. Elegir cómo mezclar los cambios

Se ve un botón verde "Merge pull request". Al hacer clic, te dará 3 opciones:

"Merge" (Recomendado para principiantes): Guarda todos los cambios manteniendo el historial completo.

"Squash and merge": Combina todas las modificaciones en un solo cambio (como resumir un capítulo largo en un párrafo).

"Rebase and merge": Organiza los cambios de forma lineal (para proyectos avanzados).

4. Confirmar y terminar

Escribir un mensaje breve (ej: "Acepto los cambios de María en el menú").

Hacer clic en "Confirm merge". ¡Listo! Los cambios ya son parte del proyecto.

16. ¿Qué es un etiqueta en Git?

En Git, una etiqueta (tag) es como un "marcador" permanente que se usa para señalar un momento importante en el historial del proyecto, generalmente para versiones (ej: v1.0.0, v2.1.3)

17. ¿Cómo crear una etiqueta en Git?

Tipos de Etiquetas

- Ligeras (Lightweight): Solo un nombre que apunta a un commit (como un bookmark).
`git tag v1.0.1`
- Anotadas (Annotated): Incluyen metadata (autor, fecha, mensaje descriptivo).
`git tag -a v1.0.0 -m "Primera versión estable"`

18. ¿Cómo enviar una etiqueta a GitHub?

- Crea la etiqueta (localmente):
`git tag -a v1.0.0 -m "Versión estable inicial" # Etiqueta anotada`
(O usa `git tag v1.0.0` para una ligera).
- Súbela a GitHub:
`git push origin v1.0.0 # Envía SOLO esa etiqueta`
(O todas las etiquetas nuevas con `git push -tags`).

19. ¿Qué es un historial de Git?

El historial de Git es como el libro de registro de tu proyecto:

Guarda cada modificación que realizas en los archivos (llamadas commits).

Detalla:

Autor: Quién hizo el cambio.

Fecha: Cuándo se implementó.

Descripción: Breve resumen de la modificación (ej: "Corregí error en cálculo de precios").

20. ¿Cómo ver el historial de Git?

Ver historial completo (orden cronológico inverso)

`git log`

Muestra: autor, fecha, mensaje del commit y su ID único (hash).

(Orden: del más reciente al más antiguo)

(Presiona q para salir)

Versión simplificada (1 línea por commit)

`git log --oneline`

21. ¿Cómo buscar en el historial de Git?

1. Por mensaje de commit

```
git log -grep="texto"
```

2. Por autor

```
git log -author="nombre"
```

3. Por rango de fechas

```
git log --since="fecha-inicio" --until="fecha-fin"
```

4. En un archivo específico

```
git log -- archivo
```

5. Por contenido en cambios

```
git log -p -S "texto"
```

22. ¿Cómo borrar el historial de Git?

- git reset (Deshacer commits locales)

Tipos de reset:

Comando

```
git reset --soft <commit>
```

```
git reset --mixed <commit>
```

```
git reset --hard <commit>
```

Efecto

Borra commits pero guarda cambios en staging.

Borra commits y saca cambios del staging (por defecto).

Borra commits y cambios (peligroso).

- git rebase (Reescribir historial)

Reescribir commits:

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

Opciones en modo interactivo:

- squash: Combina commits.
- drop: Elimina un commit.
- edit: Modifica un commit.

Limpiar commits antes de hacer push/Unificar mensajes de commits desordenados

23. ¿Qué es un repositorio privado en GitHub?

Un repositorio privado en GitHub es un proyecto de código al que solo pueden acceder usuarios específicos autorizados por el dueño.

Características clave:

Acceso restringido: Solo tú y los colaboradores que invites pueden ver o modificar el código.

Ideal para:

Proyectos personales/Código empresarial o comercial/Trabajos académicos no públicos.

No visible en búsquedas de GitHub ni para usuarios no autorizados.

Plan requerido:

GitHub ofrece repositorios privados gratis en planes personales y de equipo (con límites).

Para equipos grandes o empresas, existen planes de pago (Team, Enterprise).

24. ¿Cómo crear un repositorio privado en GitHub?

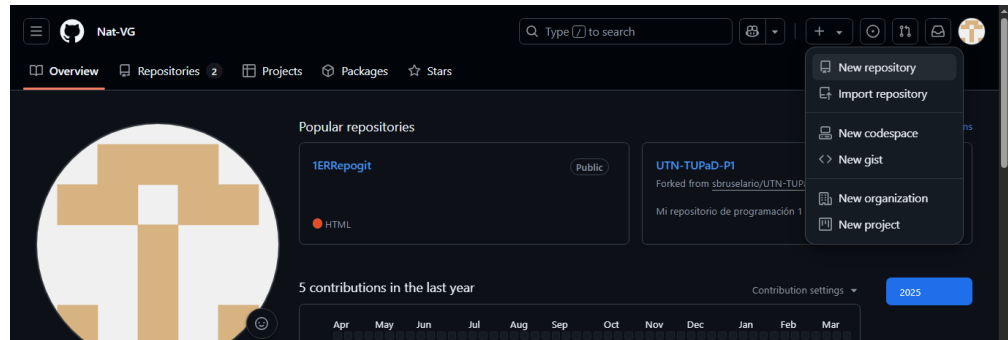
Paso 1: Inicia Sesión en GitHub

Entra a github.com y haz clic en "Sign In".

Ingresa tu usuario y contraseña.

Paso 2: Crea un Nuevo Repositorio

Haz clic en el botón "+" (arriba a la derecha) y selecciona "New repository"



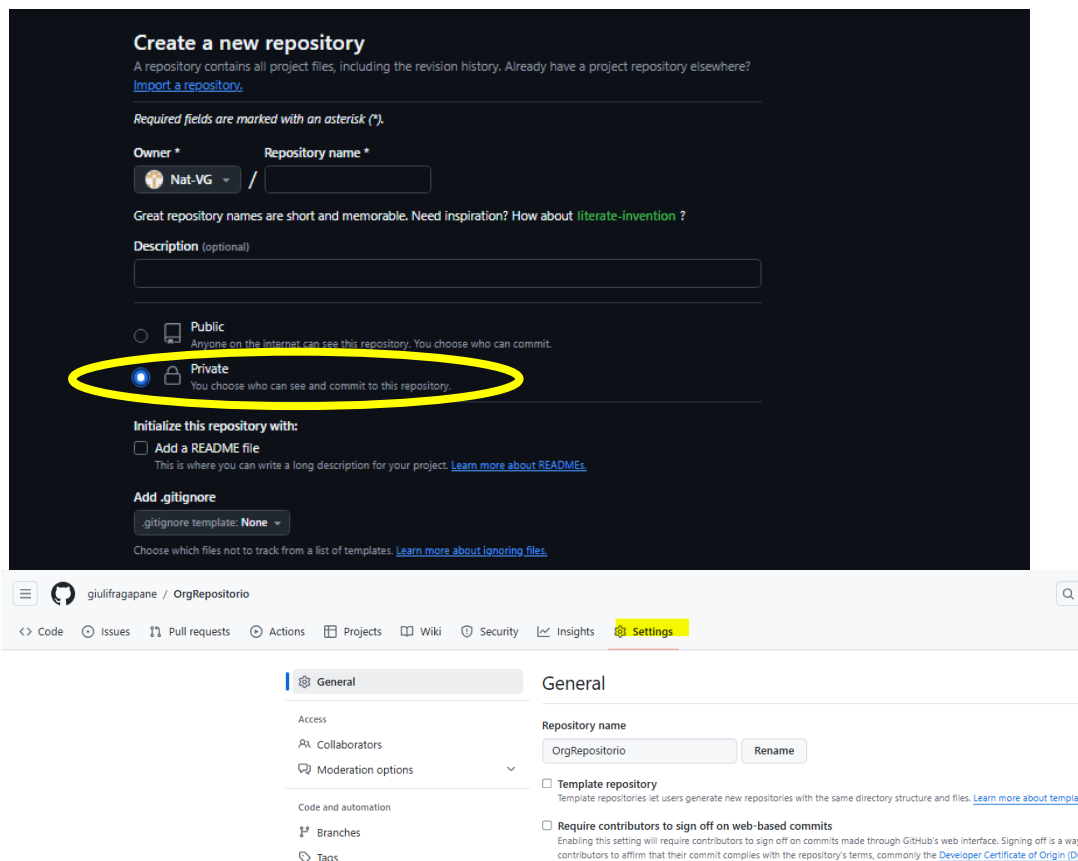
Completa los datos:

Repository name: Ponle un nombre (ej: mi-proyecto-secreto).

Description: Descripción opcional.

Visibilidad: Selecciona "Private".

Initialize this repository with a README: Marca si quieres un archivo inicial.



25. ¿Cómo invitar a alguien a un repositorio privado en GitHub?

Pasos para Invitar:

1. Entra al repositorio privado
Ve a tu repositorio en GitHub (ej: github.com/tu-usuario/repo-privado).
2. Abre la configuración de colaboradores
Haz clic en Settings (pestaña superior derecha).
En el menú izquierdo, selecciona Collaborators > Add people.
Menú Collaborators
3. Invita al usuario
Escribe el nombre de usuario, email o nombre completo del colaborador.
Selecciona su perfil de la lista y haz clic en Add <usuario> to this repository.
Añadir colaborador
4. Elige los permisos
Selecciona el nivel de acceso:
Read: Solo ver el código.
Write: Editar y hacer push.
Admin: Gestionar permisos y configuraciones.
Seleccionar permisos
5. Confirma la invitación
El usuario recibirá un email y una notificación en GitHub.

Debe aceptar la invitación desde:

Su email o github.com/notifications.
Notificación de invitación

26. ¿Qué es un repositorio público en GitHub?

Un repositorio público en GitHub es un proyecto de código abierto visible para cualquier persona en internet, aunque solo los colaboradores autorizados pueden modificarlo.

Características principales:

- Acceso libre: Cualquiera puede ver el código, clonarlo o hacer fork.
- Visibilidad: Aparece en búsquedas de GitHub y en tu perfil.
- Colaboración: Ideal para proyectos open source (otros usuarios pueden sugerir cambios mediante pull requests).
- Gratis: No requiere plan de pago (ilimitados repos públicos).

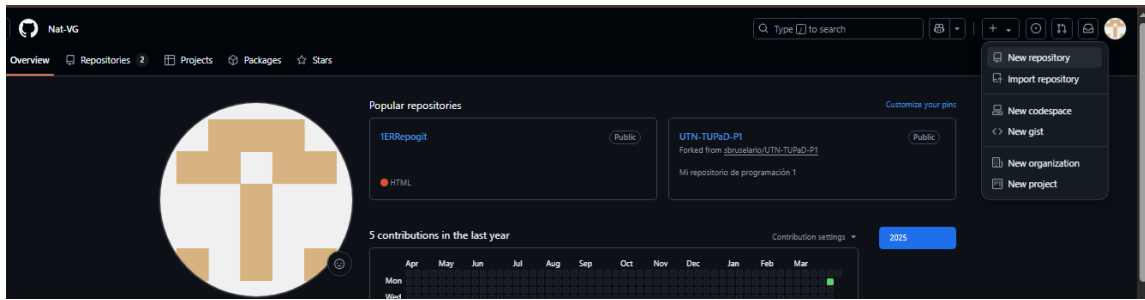
27. ¿Cómo crear un repositorio público en GitHub?

Paso 1: Inicia sesión en GitHub

Ir a <https://github.com> e iniciar sesión con la cuenta.

Paso 2: Crea un nuevo repositorio

Haz clic en el botón "New" (o "Crear repositorio") en la esquina superior derecha.



Paso 3: Configura el repositorio

Completa los detalles del repositorio:

- Repository name: Nombre del repositorio (ej: mi-proyecto).
- Description: Descripción opcional.
- Public: Selecciona "Public" para que sea accesible para todos.
- Opcional: Marca "Add a README file" si quieres un archivo inicial.
- Formulario de creación de repositorio

Paso 4: Confirma la creación

- Haz clic en "Create repository".
- Botón "Create repository"

El repositorio público estará disponible en: <https://github.com/tu-usuario/nombre-repositorio>

Repositorio creado

Notas importantes:

Si no agregaste un README.md, GitHub te dará instrucciones para subir código desde tu computadora.

Puedes clonar el repositorio con:

bash

Copy

git clone <https://github.com/tu-usuario/nombre-repositorio.git>

28. ¿Cómo compartir un repositorio público en GitHub?

Compartir un repositorio público en GitHub es sencillo, ya que al ser público, cualquier persona puede verlo si tiene el enlace. Se puede compartir de las siguientes formas:

Compartir el enlace directo

El repositorio público ya es accesible mediante su URL:

`https://github.com/tu-usuario/nombre-del-repositorio`

Puedes copiar y pegar este enlace en Redes sociales (Twitter, LinkedIn, Facebook).

Usar el botón "Share" de GitHub

Ir a tu repositorio en GitHub. Hacer clic en el botón "Share" (o "Code" → copia la URL).

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).

```
ALBERTO@DESKTOP-AJ16U73 MINGW64 ~/Desktop/GIT PRACTICA/RepoTP2 (main)
$ 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)
    new file:   mi_archivo.txt

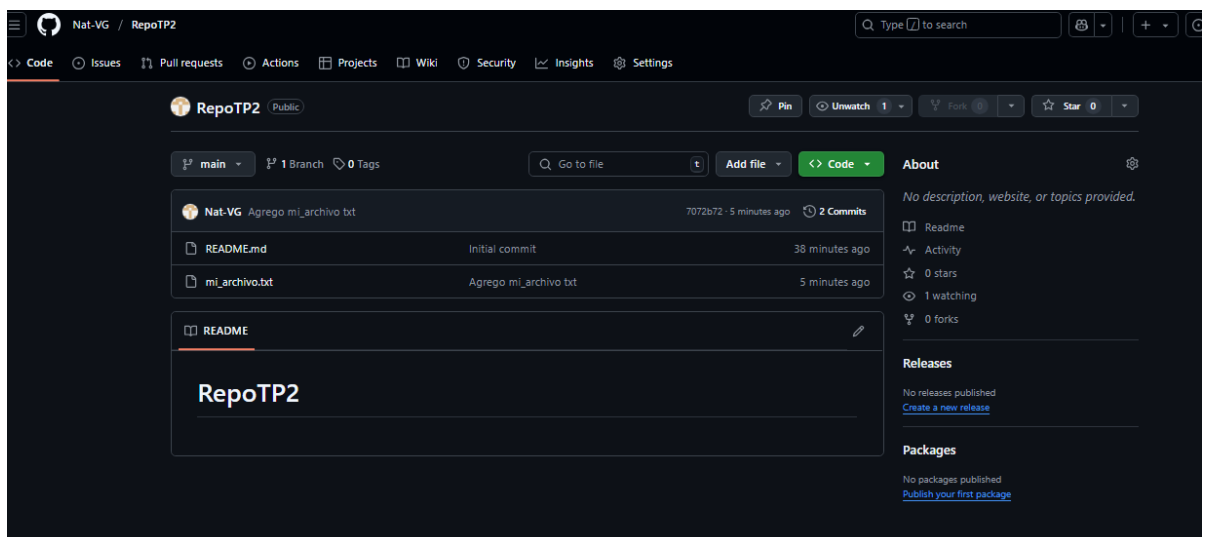
ALBERTO@DESKTOP-AJ16U73 MINGW64 ~/Desktop/GIT PRACTICA/RepoTP2 (main)
$ git commit -m "Agrego mi_archivo txt"
git: 'commit-m' is not a git command. See 'git --help'.

The most similar command is
    commit-tree

ALBERTO@DESKTOP-AJ16U73 MINGW64 ~/Desktop/GIT PRACTICA/RepoTP2 (main)
$ git commit -m "Agrego mi_archivo txt"
[main 7072b72] Agrego mi_archivo txt
 1 file changed, 1 insertion(+)
 create mode 100644 mi_archivo.txt

ALBERTO@DESKTOP-AJ16U73 MINGW64 ~/Desktop/GIT PRACTICA/RepoTP2 (main)
$ git push origin main
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 316 bytes | 158.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/Nat-VG/RepoTP2.git
 e08d9a6..7072b72  main -> main

ALBERTO@DESKTOP-AJ16U73 MINGW64 ~/Desktop/GIT PRACTICA/RepoTP2 (main)
$
```



Creando Branchs

- Crear una Branch

```
MINGW64/c/Users/ALBERTO/Desktop/GIT PRACTICA/RepoTP2
ALBERTO@DESKTOP-AJ16U73 MINGW64 ~/Desktop/GIT PRACTICA/RepoTP2 (main)
$ git checkout -b NuevaRama
Switched to a new branch 'NuevaRama'

ALBERTO@DESKTOP-AJ16U73 MINGW64 ~/Desktop/GIT PRACTICA/RepoTP2 (Nueva
Rama)
$ git branch main
fatal: a branch named 'main' already exists

ALBERTO@DESKTOP-AJ16U73 MINGW64 ~/Desktop/GIT PRACTICA/RepoTP2 (Nueva
Rama)
$ git branch
* NuevaRama
  main

ALBERTO@DESKTOP-AJ16U73 MINGW64 ~/Desktop/GIT PRACTICA/RepoTP2 (Nueva
Rama)
$ git status
On branch NuevaRama
nothing to commit, working tree clean
```

- **Realizar cambios o agregar un archivo**

```
MINGW64/c/Users/ALBERTO/Desktop/GIT PRACTICA/RepoTP2
ALBERTO@DESKTOP-AJ16U73 MINGW64 ~/Desktop/GIT PRACTICA/RepoTP2 (Nueva
Rama)
$ echo "Agregando una modificacion en mi archivo" > mi-archivo.txt

ALBERTO@DESKTOP-AJ16U73 MINGW64 ~/Desktop/GIT PRACTICA/RepoTP2 (Nueva
Rama)
$ git add .
warning: in the working copy of 'mi-archivo.txt', LF will be replaced
by CRLF the next time Git touches it

ALBERTO@DESKTOP-AJ16U73 MINGW64 ~/Desktop/GIT PRACTICA/RepoTP2 (Nueva
Rama)
$ git status
On branch NuevaRama
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   mi-archivo.txt

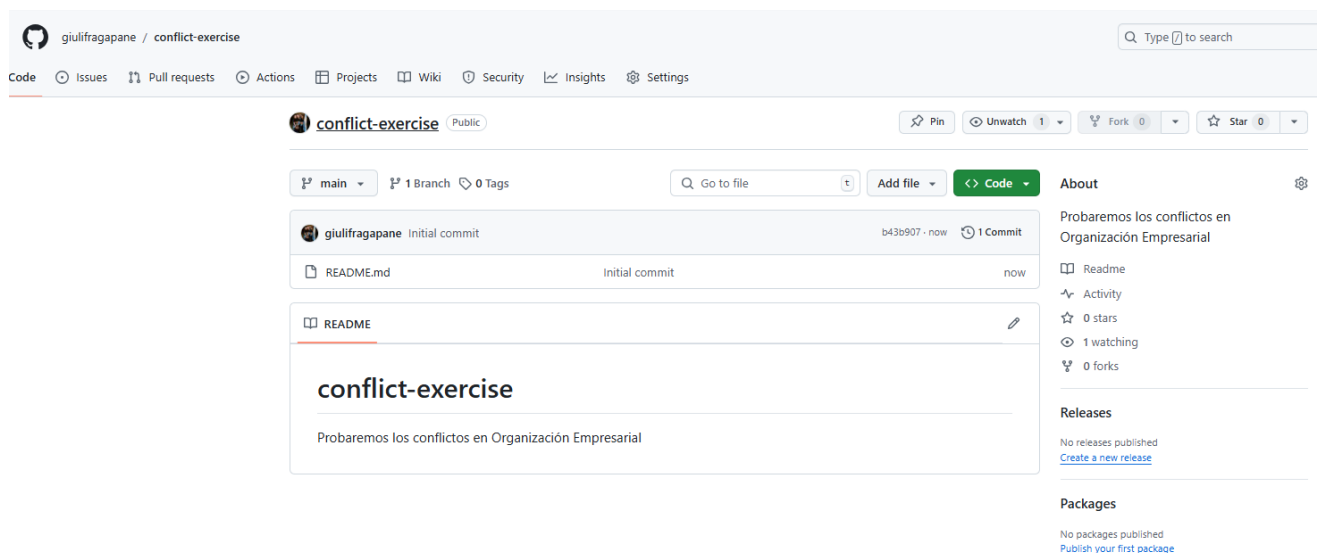
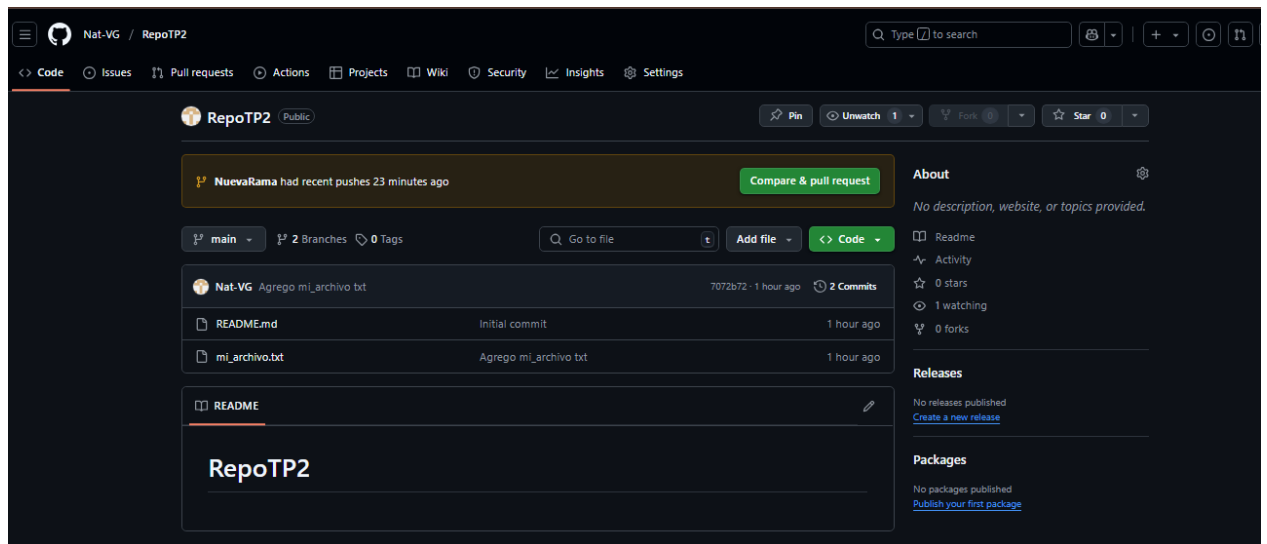
ALBERTO@DESKTOP-AJ16U73 MINGW64 ~/Desktop/GIT PRACTICA/RepoTP2 (Nueva
Rama)
$ git commit -m "Modificacion echa desde NuevaRama"
[NuevaRama 00d757b] Modificacion echa desde NuevaRama
1 file changed, 1 insertion(+)
 create mode 100644 mi-archivo.txt

ALBERTO@DESKTOP-AJ16U73 MINGW64 ~/Desktop/GIT PRACTICA/RepoTP2 (Nueva
Rama)
$ git status
On branch NuevaRama
nothing to commit, working tree clean
```

- **Subir la Branch**

```
MINGW64/c/Users/ALBERTO/Desktop/GIT PRACTICA/RepoTP2
ALBERTO@DESKTOP-AJ16U73 MINGW64 ~/Desktop/GIT PRACTICA/RepoTP2 (Nueva
Rama)
$ git push origin NuevaRama
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 365 bytes | 365.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
remote:
remote: Create a pull request for 'NuevaRama' on GitHub by visiting:
remote:   https://github.com/Nat-VG/RepoTP2/pull/new/NuevaRama
remote:
To https://github.com/Nat-VG/RepoTP2.git
 * [new branch]   NuevaRama -> NuevaRama

ALBERTO@DESKTOP-AJ16U73 MINGW64 ~/Desktop/GIT PRACTICA/RepoTP2 (Nueva
Rama)
$ |
```



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".

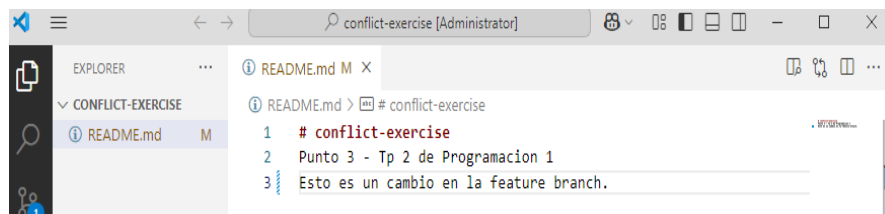
Paso 2: Clonar el repositorio a tu máquina local

- 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"`

```
ALBERTO@DESKTOP-AJ16U73 MINGW64 ~/Desktop/GIT PRACTICA/conflict-exercise (feature-branch)
$ git add README.md

ALBERTO@DESKTOP-AJ16U73 MINGW64 ~/Desktop/GIT PRACTICA/conflict-exercise (feature-branch)
$ git commit -m "Added a line in feature-branch"
[feature-branch 222f9e0] Added a line in feature-branch
1 file changed, 1 insertion(+)
```

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:

```
MINGW64: c:/Users/ALBERTO/Desktop/GIT PRACTICA/conflict-exercise
$ git add README.md

ALBERTO@DESKTOP-AJ16U73 MINGW64 ~/Desktop/GIT PRACTICA/conflict-exercise (feature-branch)
$ git commit -m "Added a line in feature-branch"
[feature-branch 222f9e0] Added a line in feature-branch
1 file changed, 1 insertion(+)

ALBERTO@DESKTOP-AJ16U73 MINGW64 ~/Desktop/GIT PRACTICA/conflict-exercise (feature-branch)
$ git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.

ALBERTO@DESKTOP-AJ16U73 MINGW64 ~/Desktop/GIT PRACTICA/conflict-exercise (main)
$ git add README.md

ALBERTO@DESKTOP-AJ16U73 MINGW64 ~/Desktop/GIT PRACTICA/conflict-exercise (main)
$ git commit -m "Added a line in main branch"
[main 79a736e] Added a line in main branch
1 file changed, 1 insertion(+)
```

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.

```
ALBERTO@DESKTOP-AJ16U73 MINGW64 ~/Desktop/GIT PRACTICA/conflict-exercise (main)
$ git merge feature-branch
Auto-merging README.md
CONFLICT (content): Merge conflict in README.md
Automatic merge failed; fix conflicts and then commit the result.

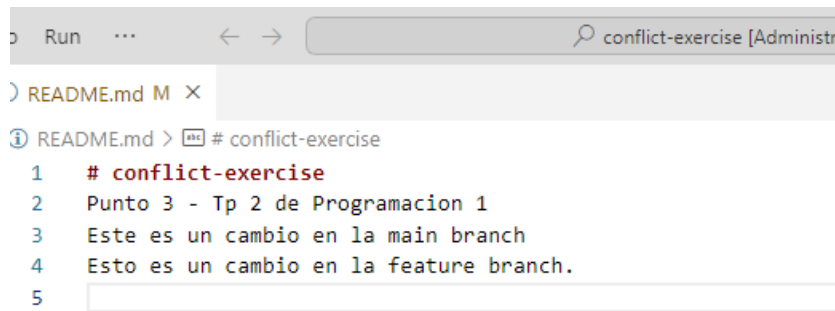
ALBERTO@DESKTOP-AJ16U73 MINGW64 ~/Desktop/GIT PRACTICA/conflict-exercise (main|MER
GING)
$ |
```

Paso 6: Resolver el conflicto

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



```

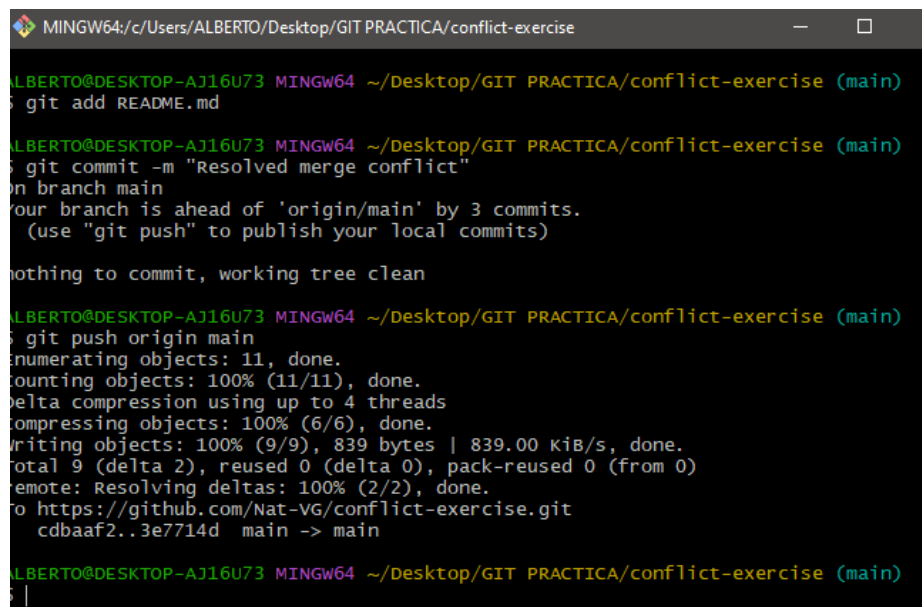
1 # conflict-exercise
2 Punto 3 - Tp 2 de Programacion 1
3 Este es un cambio en la main branch
4 Esto es un cambio en la feature branch.
5

```

- 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:



```

MINGW64:/c/Users/ALBERTO/Desktop/GIT PRACTICA/conflict-exercise
ALBERTO@DESKTOP-AJ16U73 MINGW64 ~/Desktop/GIT PRACTICA/conflict-exercise (main)
$ git add README.md
ALBERTO@DESKTOP-AJ16U73 MINGW64 ~/Desktop/GIT PRACTICA/conflict-exercise (main)
$ git commit -m "Resolved merge conflict"
[master 3e7714d] Resolved merge conflict
1 file changed, 4 insertions(+), 1 deletion(-)
Your branch is ahead of 'origin/main' by 3 commits.
(use "git push" to publish your local commits)

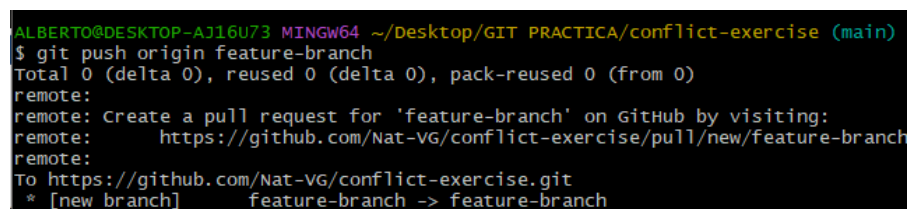
nothing to commit, working tree clean

ALBERTO@DESKTOP-AJ16U73 MINGW64 ~/Desktop/GIT PRACTICA/conflict-exercise (main)
$ git push origin main
Enumerating objects: 11, done.
Counting objects: 100% (11/11), done.
Delta compression using up to 4 threads
Compressing objects: 100% (6/6), done.
Writing objects: 100% (9/9), 839 bytes | 839.00 KiB/s, done.
Total 9 (delta 2), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (2/2), done.
To https://github.com/Nat-VG/conflict-exercise.git
   cdbaaf2..3e7714d  main -> main

```

- También sube la feature-branch si deseas:

`git push origin feature-branch`



```

ALBERTO@DESKTOP-AJ16U73 MINGW64 ~/Desktop/GIT PRACTICA/conflict-exercise (main)
$ 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/Nat-VG/conflict-exercise/pull/new/feature-branch
remote:
To https://github.com/Nat-VG/conflict-exercise.git
 * [new branch]   feature-branch -> 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.

