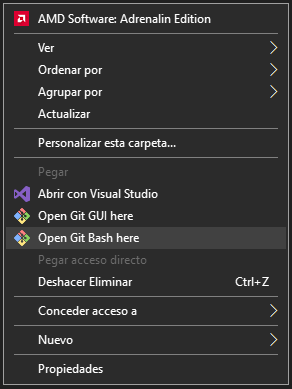
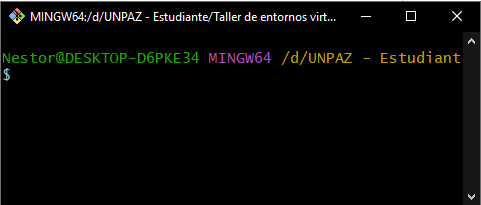
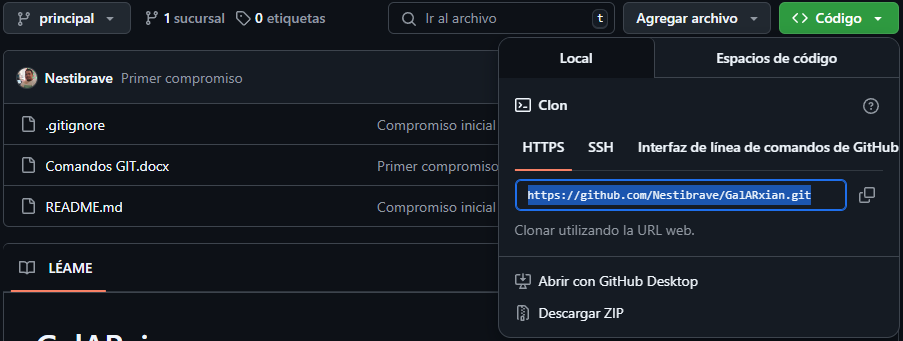
# CONFIGURACIÓN INICIAL

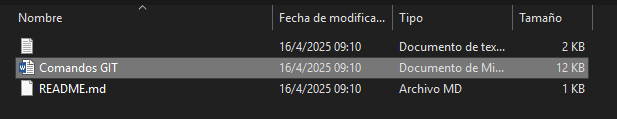
1. Instalar o actualizar git.
2. Crear una carpeta local donde irá el proyecto.
3. Abrir git bash:
   1. Ingresar a la carpeta creada.
   2. Click derecho en el espacio vacío
   3. Click en “Open Git Bash Here”

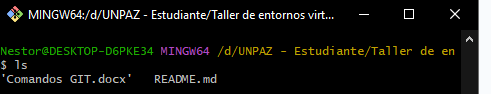
1. Configurar nombre de usuario e e-mail ingresando los siguientes comandos:
   1. git config --global user.name "John Doe"
   2. git config --global user.email johndoe@example.com
2. Crear un proyecto remoto en Github y copiar la dirección del repositorio:



1. En la carpeta local crea un archivo de texto de prueba:



1. Asegurarse de estar en la rama main, caso contrario ejecutar:
   1. git branch -m main
2. **Armar** el paquete de archivos para enviarlo al repositorio remoto añadiendo a ese paquete el archivo creado. A esto se lo conoce como “agregar cambios al staging”:
   1. Para saber el nombre de los elementos de la carpeta ingresar el comando “ls”



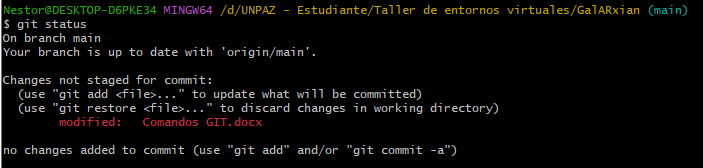
* 1. Luego ejecutar → git add "Comandos GIT.docx"

1. **Cerrar** el paquete de archivos para enviarlo. Esto se conoce como sacarle una foto al paquete o etiquetarlo para su envío:
   1. git commit -m "Primer commit"
2. Conectar el repositorio local al remoto con la dirección que copiamos de Github:
   1. git remote add origin https://github.com/Nestibrave/GalARxian.git
3. En caso de haber conflicto traer el repositorio remoto al local:
   1. git pull --rebase origin main
4. **Enviar** todos los cambios y agregados locales al repositorio remoto. A esto se lo conoce como empujar o coloquialmente “pushear” los cambios. La primera vez lo haremos ejecutando:
   1. git push -u origin main

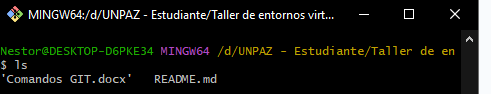
# 

# DESPUÉS DE CADA MODIFICACIÓN DEL REPOSITORIO LOCAL

1. Guardar todas las modificaciones.
2. Ingresar el comando git status para ver los cambios realizados:
   1. git status



1. **Agregar cambios al staging**:
   1. Para saber el nombre de los elementos de la carpeta ingresar el comando “ls”



* 1. Si se desea agregar de un archivo ejecutar → git add "Comandos GIT.docx"
  2. Si se desea agregar todos los cambios en todos los archivos y carpetas reconocidos en el git status ejecutar→ “git add .”

1. **Etiquetar** el paquete de archivos para enviarlo:
   1. git commit -m "Descripción general del paquete"
2. **Enviar** todos los cambios y agregados locales al repositorio remoto. Como esta no es la primera vez lo haremos con el siguiente comando:
   1. git push

# LISTA DE COMANDOS

* **git init** creará un nuevo repositorio local GIT. El siguiente comando de Git creará un repositorio en el directorio actual:
  + git init
* Como alternativa, puedes crear un repositorio dentro de un nuevo directorio especificando el nombre del proyecto:
  + git init [nombre del proyecto]
* **git clone** se usa para copiar un repositorio. Si el repositorio está en un servidor remoto, usa:
  + git clone nombredeusuario@host:/path/to/repository
* A la inversa, ejecuta el siguiente comando básico para copiar un repositorio local:
  + git clone /path/to/repository
* **git add** se usa para agregar archivos al área de preparación. Por ejemplo, el siguiente comando de Git básico indexará el archivo temp.txt:
  + git add <temp.txt>
* **git commit** creará una instantánea de los cambios y la guardará en el directorio git.
  + git commit –m “El mensaje que acompaña al commit va aquí”
* **git config** puede ser usado para establecer una configuración específica de usuario, como el email, nombre de usuario y tipo de formato, etc. Por ejemplo, el siguiente comando se usa para establecer un email:
  + git config --global user.email tuemail@ejemplo.com
* La opción -global le dice a GIT que vas a usar ese correo electrónico para todos los repositorios locales. Si quieres utilizar diferentes correos electrónicos para diferentes repositorios, usa el siguiente comando:
  + git config --local user.email tuemail@ejemplo.com
* **git status** muestra la lista de los archivos que se han cambiado junto con los archivos que están por ser preparados o confirmados.
  + git status
* **git push** se usa para enviar confirmaciones locales a la rama maestra del repositorio remoto. Aquí está la estructura básica del código:
  + git push origin <master>
* **git checkout** crea ramas y te ayuda a navegar entre ellas. Por ejemplo, el siguiente comando crea una nueva y automáticamente se cambia a ella:
  + command git checkout -b <branch-name>
* Para cambiar de una rama a otra, sólo usa:
  + git checkout <branch-name>
* **git remote** te permite ver todos los repositorios remotos. El siguiente comando listará todas las conexiones junto con sus URLs:
  + git remote -v
* Para conectar el repositorio local a un servidor remoto, usa este comando:
  + git remote add origin <host-or-remoteURL>
* Por otro lado, el siguiente comando borrará una conexión a un repositorio remoto especificado:
  + git remote <nombre-del-repositorio>
* **git branch** se usa para listar, crear o borrar ramas. Por ejemplo, si quieres listar todas las ramas presentes en el repositorio, el comando debería verse así:
  + git branch
* Si quieres borrar una rama, usa:
  + git branch -d <branch-name>
* **git pull** fusiona todos los cambios que se han hecho en el repositorio remoto con el directorio de trabajo local.
  + git pull
* **git merge** se usa para fusionar una rama con otra rama activa:
  + git merge <branch-name>
* **git diff** se usa para hacer una lista de conflictos. Para poder ver conflictos con respecto al archivo base, usa:
  + git diff --base <file-name>
* El siguiente comando se usa para ver los conflictos que hay entre ramas antes de fusionarlas:
  + git diff <source-branch> <target-branch>
* Para ver una lista de todos los conflictos presentes usa:
  + git diff
* **git tag** marca commits específicos. Los desarrolladores lo usan para marcar puntos de lanzamiento como v1.0 y v2.0.
  + git tag 1.1.0 <instert-commitID-here>
* **git log** se usa para ver el historial del repositorio listando ciertos detalles de la confirmación. Al ejecutar el comando se obtiene una salida como ésta:
  + commit 15f4b6c44b3c8344caasdac9e4be13246e21sadw
  + Author: Alex Hunter <[alexh@gmail.com](mailto:alexh@gmail.com)>
  + Date: Mon Oct 1 12:56:29 2016 -0600
* **git reset** sirve para resetear el index y el directorio de trabajo al último estado de confirmación.
  + git reset - -hard HEAD
* **git rm** se puede usar para remover archivos del index y del directorio de trabajo.
  + git rm filename.txt
* **git stash** guardará momentáneamente los cambios que no están listos para ser confirmados. De esta manera, pudes volver al proyecto más tarde.
  + git stash
* **git show** se usa para mostrar información sobre cualquier objeto git.
  + git show
* **git fetch** le permite al usuario buscar todos los objetos de un repositorio remoto que actualmente no se encuentran en el directorio de trabajo local.
  + git fetch origin
* **git ls-tree** te permite ver un objeto de árbol junto con el nombre y modo de cada ítem, y el valor blob de SHA-1. Si quieres ver el HEAD, usa:
  + git ls-tree HEAD
* **git cat-file** se usa para ver la información de tipo y tamaño de un objeto del repositorio. Usa la opción -p junto con el valor SHA-1 del objeto para ver la información de un objeto específico, por ejemplo:
  + git cat-file –p d670460b4b4aece5915caf5c68d12f560a9fe3e4
* **git grep** le permite al usuario buscar frases y palabras específicas en los árboles de confirmación, el directorio de trabajo y en el área de preparación. Para buscar por www.hostinger.com en todos los archivos, usa:
  + git grep “www.hostinger.com”
* **gitk** muestra la interfaz gráfica para un repositorio local. Simplemente ejecuta:
  + gitk
* **git instaweb** te permite explorar tu repositorio local en la interfaz GitWeb. Por ejemplo:
  + git instaweb –http=webrick
* **git gc** limpiará archivos innecesarios y optimizará el repositorio local.
  + git gc
* **git archive** le permite al usuario crear archivos zip o tar que contengan los constituyentes de un solo árbol de repositorio. Por ejemplo:
  + git archive - -format=tar master
* **git prune** elimina los objetos que no tengan ningún apuntador entrante.
  + git prune
* **git fsck** realiza una comprobación de integridad del sistema de archivos git e identifica cualquier objeto corrupto
  + git fsck
* **git rebase** se usa para aplicar ciertos cambios de una rama en otra. Por ejemplo:
  + git rebase master