CURSO DE GIT

Git es un sistema de control de versiones distribuido creado por Linus Torvalds en 2005. Está diseñado para gestionar y seguir los cambios en proyectos de software, permitiendo a los desarrolladores colaborar eficientemente. Git guarda un historial detallado de modificaciones, identificando quién y cuándo se realizaron los cambios. Facilita la creación de ramas, la fusión de cambios, y la reversión a versiones anteriores, funcionando de manera descentralizada, lo que permite trabajar offline y sincronizar los cambios posteriormente.

GitHub, fundado en 2008, es una plataforma web que se basa en Git y añade servicios para la gestión y colaboración en proyectos de software. Ofrece un entorno centralizado para almacenar, compartir y colaborar en repositorios Git. Sus características incluyen la gestión de problemas, solicitudes de extracción, seguimiento de errores y tareas, y herramientas de integración continua para automatizar pruebas y despliegues. Permite la creación de repositorios públicos y privados, y facilita la revisión de código y la comunicación dentro del equipo.

Usos y aplicaciones:

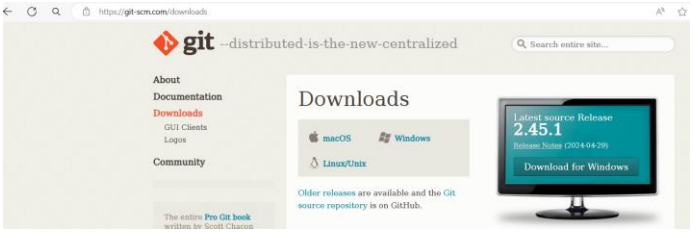
- **Desarrollo de software**: Ambos son esenciales para proyectos de código abierto y empresas privadas, facilitando la colaboración en equipos grandes y la implementación de prácticas de desarrollo ágil.
- **Otros campos**: También se usan en la gestión de configuraciones de sistemas, documentación técnica y otros ámbitos donde el control de versiones y la colaboración son críticos.

En resumen, Git es una herramienta para el seguimiento y control de versiones en proyectos de software, mientras que GitHub es una plataforma que amplía las capacidades de Git, mejorando la colaboración y gestión de proyectos con herramientas adicionales.

INSTALACION DE GIT

1. **Descargar el instalador**:

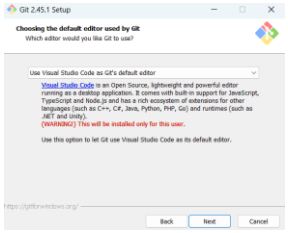
- Visita la página oficial de Git: git-scm.com
- Haz clic en "Download" y selecciona la versión adecuada para Windows.



2. **Ejecutar el instalador**:

- Abre el archivo descargado.

- Sigue las instrucciones del instalador:
    - **Seleccionar componentes**
    - **Editor predeterminado de Git**: Elegir el editor de texto que utilizamos



    - **Otras configuraciones**: Dejar las configuraciones por defecto a menos que tengas necesidades específicas.

3. **Finalizar la instalación:**

- Completa el proceso de instalación y cierra el instalador.

4. **Verificar la instalación:**

```
C:\Users\aguil>git -v
git version 2.45.1.windows.1
```

CONFIGURACION INICIAL

Modificar el nombre del usuario a nivel global en la configuración de git

```
aguil@David_Aguilar MINGW64 ~
$ git config --global user.name "David Aguilar"
```

Modificar el email del usuario a nivel global en la configuración de git

```
aguil@David_Aguilar MINGW64 ~
$ git config --global user.email "david.aguilar@ies-systems.com"
```

Configura el editor de texto predeterminado que Git utilizará cuando necesite que el usuario introduzca un mensaje de confirmación. En este caso, está configurando el editor para que sea Visual Studio Code

```
aguil@David_Aguilar MINGW64 ~
$ git config --global core.editor "code --wait"
```

Cuando core.autocrlf está establecido en true, Git convertirá automáticamente los saltos de línea a la convención apropiada para el sistema operativo en el que se esté trabajando. Esto es útil cuando se trabaja en un equipo con diferentes sistemas operativos, como Windows, macOS y Linux, para mantener la consistencia en los archivos.

```
aguil@David_Aguilar MINGW64 ~
$ git config --global core.autocrlf true
```

AREAS DE TRABAJO

1. **Directorio de trabajo (Working Directory)**: Es el directorio en tu sistema de archivos donde trabajas con tus archivos. Aquí es donde modificas, creas y eliminas archivos como parte de tu proceso de desarrollo.
2. **Área de preparación (Staging Area o Index)**: También conocida como "índice", es una especie de zona intermedia entre tu directorio de trabajo y tu repositorio Git. Aquí agregas los cambios que deseas incluir en tu próxima confirmación. Puedes seleccionar qué cambios quieres incluir en la próxima confirmación y organizarlos antes de comprometerlos en tu repositorio.
3. **Repositorio Git (Git Repository o Local Repository)**: Es donde Git almacena la historia de todos tus cambios confirmados. Este es el lugar donde los cambios se vuelven permanentes y se guardan en la historia del proyecto. Aquí se encuentran todas las confirmaciones (commits) y ramas (branches).

El flujo típico de trabajo en Git implica mover los cambios de tu directorio de trabajo al área de preparación y luego confirmar esos cambios en el repositorio Git.

Ubicación donde crearemos el repositorio

```
aguil@David_Aguilar MINGW64 /c
$ mkdir repo

aguil@David_Aguilar MINGW64 /c
$ cd repo
```

El comando **git init** se utiliza para iniciar un nuevo repositorio Git en un directorio existente. Cuando ejecutas este comando en un directorio, Git crea un nuevo subdirectorio oculto llamado ".git"

```
aguil@David_Aguilar MINGW64 /c/repo
$ git init
Initialized empty Git repository in C:/repo/.git/
```

Agregar el archivo "archivo.txt" al área de preparación utilizando el comando **git add**. Ahora, este archivo está listo para ser incluido en la próxima confirmación que realice en el repositorio.

```
aguil@David_Aguilar MINGW64 /c/repo (master)
$ git add archivo.txt
```

Cuando ejecuto **git status**, veo que hay cambios listos para ser confirmados, pero el mensaje está incompleto y no muestra la lista de archivos específicos que están preparado

La sección "Changes to be committed" indica que hay cambios en el área de preparación (staging area) que están listos para ser confirmados en el repositorio.

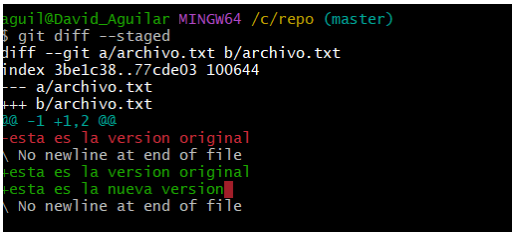
```
aguil@David_Aguilar MINGW64 /c/repo (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   archivo.txt
```

Agregamos un nuevo commit el cual nos agrego el archivo que habíamos preparado anteriormente

```
aguil@David_Aguilar MINGW64 /c/repo (master)
$ git commit
[master (root-commit) 75114e3] Nuevo commit
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 archivo.txt
```

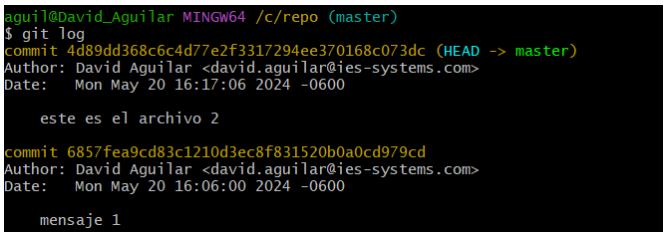
GIT DIFF

Para conocer cual es la diferencia entre lo que esta en el área de preparación, la línea roja es lo que esta en el commit y la verde esta en el área de preparación



El comando **git log** muestra el historial de confirmaciones (commits) en el repositorio Git.

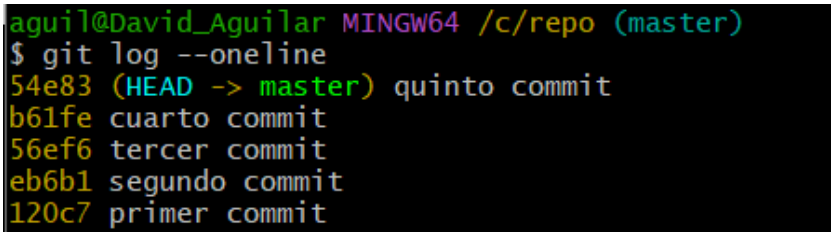
- Se han realizado dos commits en el repositorio.



- Cada commit en el registr o muestra su identificador único (hash), el autor, la fecha y hora en que se realizó el commit, y el mensaje asociado con ese commit. Esto te permite rastrear los cambios realizados en el repositorio a lo largo del tiempo.

MODIFICAR Y DESHACER COMMIT

El primer paso será crear 5 commits



Modificar un commit en el cual le cambiaremos el nombre remplazando “Quinto Commit” por “Ultimo commit “

```
aguil@David_Aguilar MINGW64 /c/repo (master)
$ git commit --amend
[master 2d772] ultimo commit
 Date: Mon May 20 16:41:13 2024 -0600
 1 file changed, 1 insertion(+), 1 deletion(-)

aguil@David_Aguilar MINGW64 /c/repo (master)
$ git log --oneline
2d772 (HEAD -> master) ultimo commit
b61fe cuarto commit
56ef6 tercer commit
eb6b1 segundo commit
120c7 primer commit
```

RAMAS(BRANCHES)

Consultar las ramas

```
aguil@David_Aguilar MINGW64 /c/repo (master)
$ git branch
* master
```

Creación de la rama (el nombre de la rama debe describir la función a realizar)

```
aguil@David_Aguilar MINGW64 /c/repo (master)
$ git branch modificar-dev
```

Crear e ir a la rama

```
aguil@David_Aguilar MINGW64 /c/repo (master)
$ git switch -c rama-nueva-temporal
Switched to a new branch 'rama-nueva-temporal'
```

Eliminar una rama, para ello debemos ubicarnos en una rama diferente ya que no podemos estar dentro de la rama a eliminar

```
aguil@David_Aguilar MINGW64 /c/repo (master)
$ git branch -d rama-nueva-temporal
Deleted branch rama-nueva-temporal (was cb214).

aguil@David_Aguilar MINGW64 /c/repo (master)
$ git branch
* master
  modificar-dev
```

Modificar el nombre de una rama, si nos encontraos en una rama diferente debemos utilizar la siguiente estructura en la cual indicaremos el nombre de la rama a modificar y el nuevo nombre que le asignaremos

```
aguil@David_Aguilar MINGW64 /c/repo (master)
$ git branch -m modificar-dev modificar-texto

aguil@David_Aguilar MINGW64 /c/repo (master)
$ git branch
* master
  modificar-texto
```

Caso contrario si nos encontramos en la rama se realizara lo siguiente

```
aguil@David_Aguilar MINGW64 /c/repo (master)
$ git switch modificar-texto
Switched to branch 'modificar-texto'

aguil@David_Aguilar MINGW64 /c/repo (modificar-texto)
$ git branch -m cambiar-texto
```

FUCIONAR RAMAS

El comando **git merge optimize-function** indica que se está realizando una fusión de la rama **optimize-function** en la rama actual. El resultado del comando muestra que se ha completado un "fast-forward merge," lo que significa que la rama actual se ha movido hacia adelante para incluir los cambios de **optimize-function** sin crear un nuevo commit de fusión.

```
aguil@David_Aguilar MINGW64 /c/repo (master)
$ git merge optimize-function
Updating 903c4..beabd
Fast-forward
 func.py | 11 +++--------
 1 file changed, 3 insertions(+), 8 deletions(-)
```

Para eliminar una fusión se utilizara el commit de la rama que estaba fusionada

```
aguil@David_Aguilar MINGW64 /c/repo (master)
$ git reset --hard 903c4
HEAD is now at 903c4 Funciones
```

MERGE CONFLICTS

El comando **git merge conflict-branch** indica que intentaste fusionar la rama **conflict-branch** en la rama actual, pero se ha producido un conflicto de fusión en el archivo

```
aguil@David_Aguilar MINGW64 /c/repo (master)
$ git merge conflict-branch
Auto-merging codigo.py
CONFLICT (content): Merge conflict in codigo.py
Automatic merge failed; fix conflicts and then commit the result.
```

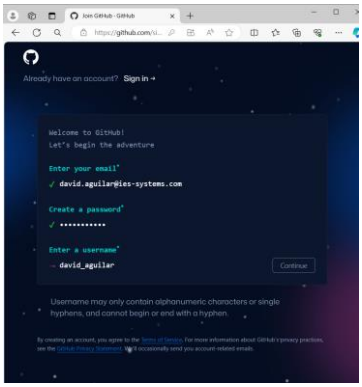
```
Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
<<<<<<< HEAD (Current Change)
print("Hola "+nombre+" "+apellido+ "Como estas?")
=======
saludo ="Hola "+nombre+" "+apellido
print(saludo)
>>>>>>> conflict-branch (Incoming Change)
```

- El bloque entre <<<<<<< **HEAD** y ======= es el código en la rama actual.
- El bloque entre ======= y >>>>>>> **conflict-branch** es el código en la rama **conflict-branch**.

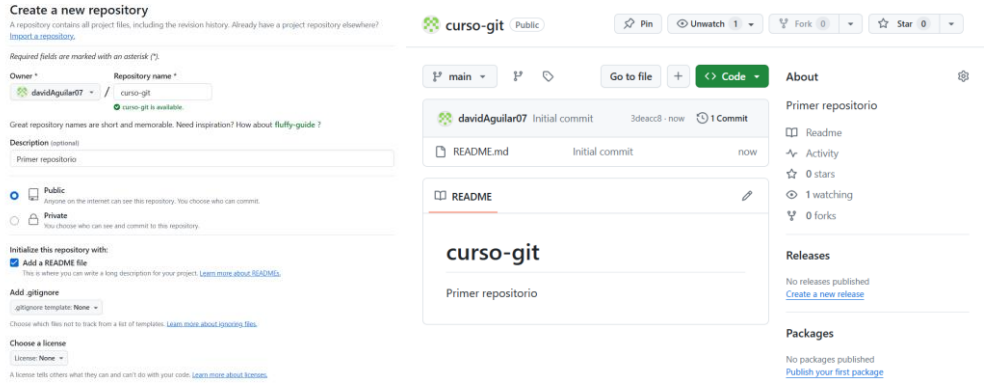
GIT HUB

GitHub es una plataforma basada en la web que utiliza Git para el control de versiones. Proporciona una variedad de características para facilitar la colaboración en proyectos de software, incluyendo repositorios, issues, pull requests y más.

Creación de cuenta en github

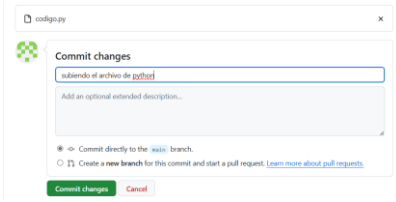


Crear y configurar el repositorio remoto



Agregar un nuevo archivo

- Dar al botón de agregar
- Dar click en upload files
- Posteriormente arrastrar el archivo y se cargara el archivo
- Le daremos el titulo
- Damos en commit changes



GIT CLONE

1. **Obtener la URL del Repositorio**:
    - Ve al repositorio en GitHub que deseas clonar.
    - Haz clic en el botón "Code" en la página principal del repositorio.
    - Copia la URL del repositorio. Puede ser una URL HTTPS o SSH. Ejemplo de URL HTTPS: **https://github.com/usuario/repo.git**.
2. **Abrir una Terminal**:
    - Abre una terminal o línea de comandos en tu sistema operativo.
3. **Clonar el Repositorio**:
    - Navega a la ubicación donde deseas clonar el repositorio usando el comando **cd**.
    - Ejecuta el comando **git clone** seguido de la URL del repositorio.

```
aguil@David_Aguilar MINGW64 /c/repo-remoto
$ git clone https://github.com/davidAguilar07/repo-uno.git
```

GIT PUSH

Primero, realiza cambios en tu repositorio local y agrégalos al área de preparación

```
aguil@David_Aguilar MINGW64 /c/repo-remoto/repo-uno (master)
$ git add codigo.py

aguil@David_Aguilar MINGW64 /c/repo-remoto/repo-uno (master)
$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   codigo.py
```

Luego, haz un commit de los cambios

```
aguil@David_Aguilar MINGW64 /c/repo-remoto/repo-uno (master)
$ git commit -m "cambiando el nombre"
[master bd81d] cambiando el nombre
 1 file changed, 1 insertion(+), 1 deletion(-)
```

Para enviar tus commits al repositorio remoto, usa

```
aguil@David_Aguilar MINGW64 /c/repo-remoto/repo-uno (master)
$ git push origin master
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 350 bytes | 350.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/davidAguilar07/repo-uno.git
   2fb38..bd81d  master -> master
```

GIT PULL Y GIT FETCH

El comando **git pull** se utiliza para obtener y fusionar los cambios del repositorio remoto a tu repositorio local. Básicamente, **git pull** es una combinación de dos comandos: **git fetch** y **git merge**. Primero, descarga los cambios del repositorio remoto y luego los fusiona con tu rama actual.

```
aguil@David_Aguilar MINGW64 /c/repo-remoto/repo-uno (master)
$ git pull
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 999 bytes | 49.00 KiB/s, done.
From https://github.com/davidAguilar07/repo-uno
   bd81d..d7e03  master     -> origin/master
Updating bd81d..d7e03
Fast-forward
 codigo2.py | 2 ++
 1 file changed, 2 insertions(+)
 create mode 100644 codigo2.py
```

MIGRAR UN REPOSITORIO

- Desarrollar el repositorio local

```
aguil@David_Aguilar MINGW64 /c/nuevo-repo (master)
$ git log --oneline
83c77 (HEAD -> master) ultimo commit
1f0b8 2do commit
175b0 primer commit
```

- Crear un Nuevo Repositorio en GitHub
- Configurar el Repositorio Local
- Copiamos la ruta de GitHub u pegamos en GitBash

```
aguil@David_Aguilar MINGW64 /c/nuevo-repo (master)
$ git remote add origin https://github.com/davidAguilar07/nuevo-repo.git

aguil@David_Aguilar MINGW64 /c/nuevo-repo (master)
$ git remote -v
origin  https://github.com/davidAguilar07/nuevo-repo.git (fetch)
origin  https://github.com/davidAguilar07/nuevo-repo.git (push)
```

Una ves que ejecutamos el **git remote -v** nos indica que podemos realizar el **fetch** y el **push**