Desarrollo web integral.

Unidad I. Control de versiones.

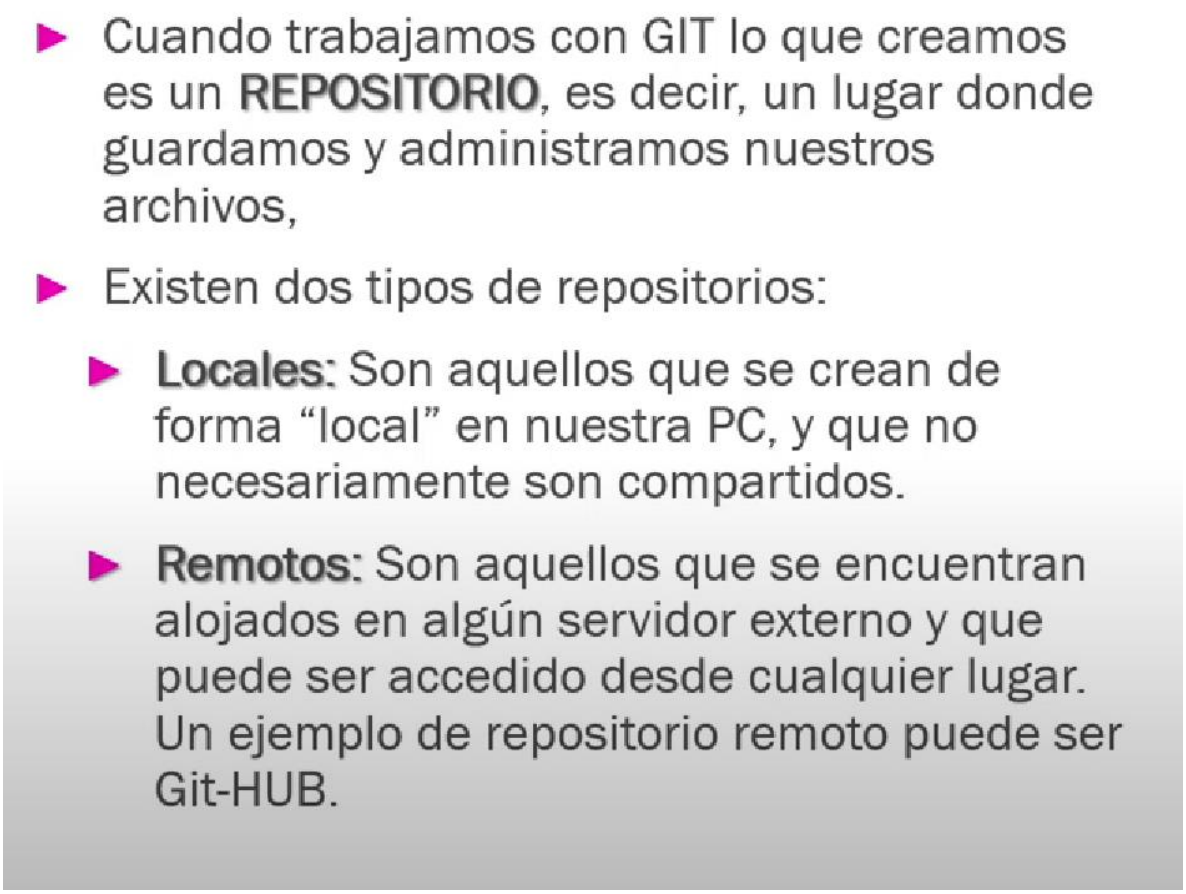
Practica 01; Introducción a git.

Instrucciones; A continuación, encontraras un breve repaso sobre GIT y una practica inicial para conocer su entorno de trabajo.

Entregable; Reporte de la practica en PDF.

Git es un sistema de control de versiones que nos permite manipular las distintas versiones que va a tener nuestro desarrollo de un determinado Software sobre todo cuando trabajamos en equipo y tenemos que manipular nuevas funcionalidades, etc.

¿Qué es un repositorio?



Comandos de GIT básicos;

- 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.

```
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”
```

Consejo profesional

Ten en cuenta que los cambios confirmados no llegarán al repositorio remoto.

- 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
```

- 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>
```

Consejo profesional

Reemplaza <master> con la rama en la que quieres enviar los cambios cuando no quieras enviarlos a la rama maestra.

- 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>
```

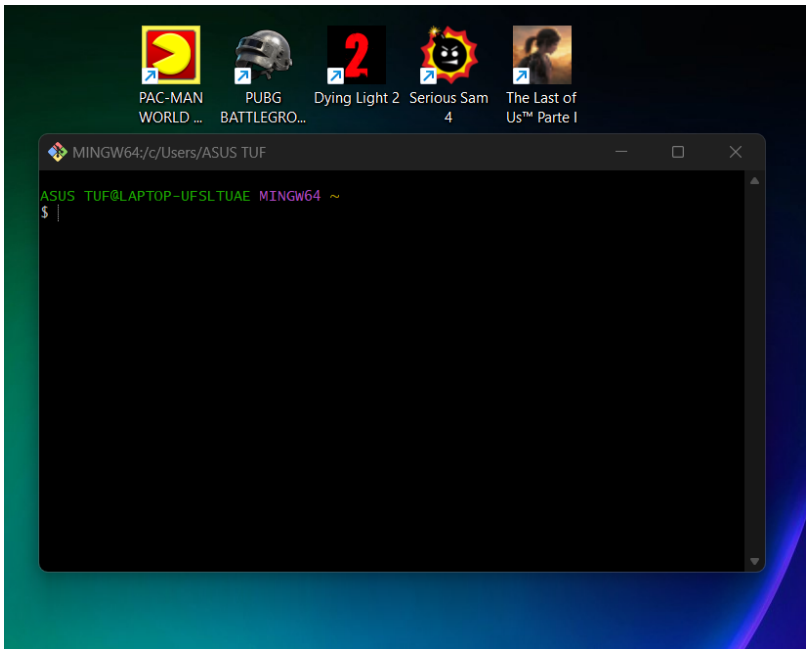
- Git diff; se usa para hacer una lista de conflictos. Para poder ver conflictos con respecto al archivo base, usa:

```
git diff --base <file-name>
```

Instalación de GIT;

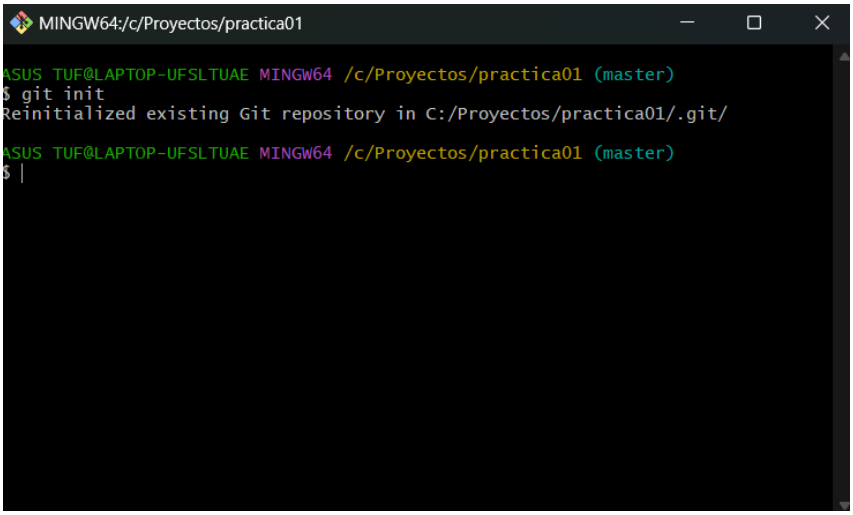
En el siguiente link; https://git-scm.com/downloads

Vamos a descargar la versión más actual de GIT.



Cuando ejecutamos el instalador solo vamos a dar clic en la casilla para inicializar GIT BASH y se habrá instalado correctamente, lo sabremos si abre correctamente GIT. Para confirmar la confirmación vamos a ingresar al directorio de cualquier carpeta (Preferentemente una carpeta vacía)

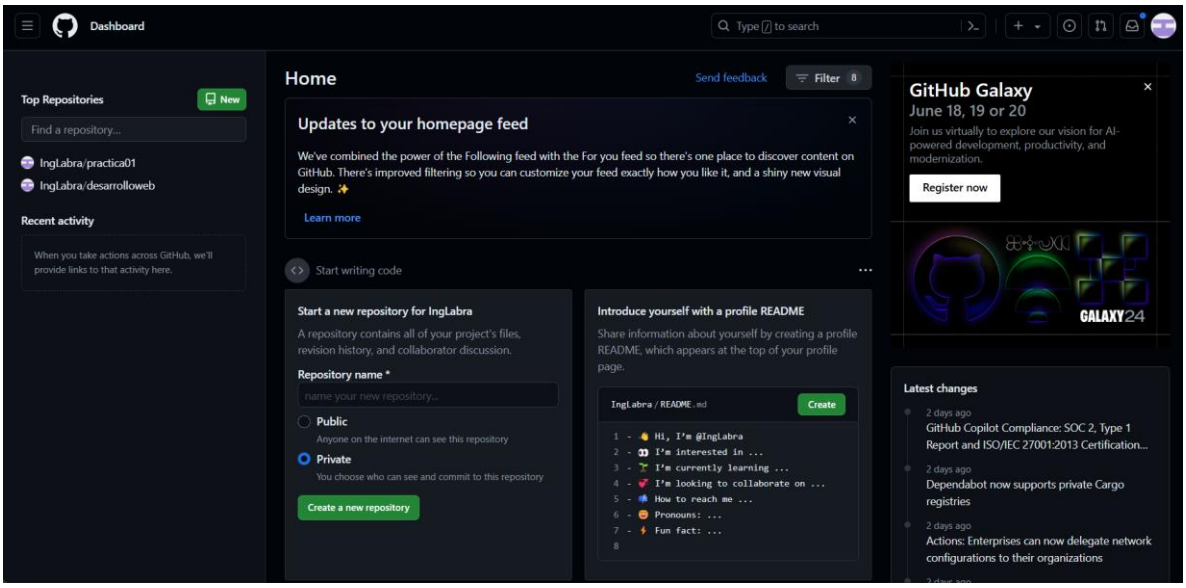
Estando en la carpeta, aplicamos el comando GIT INIT;



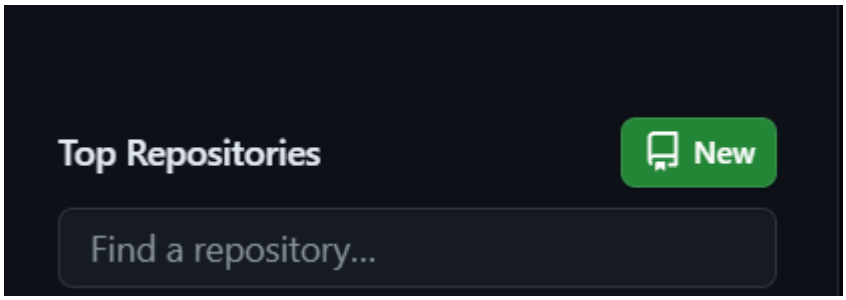
Si todo sale bien, tendremos un mensaje de que se ha inicializado un repositorio en GIT.

Con ello, hemos creado un repositorio local, sin embargo, recordemos que existen también los repositorios remotos, para ello;

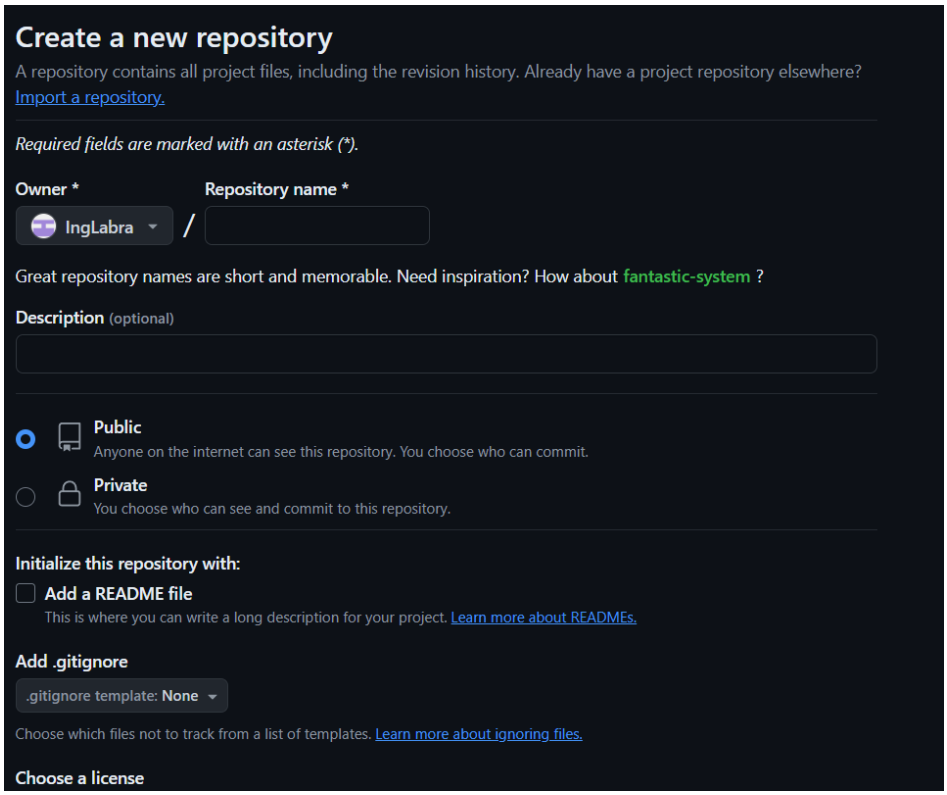
Nos dirigimos a la página oficial de GITHUB;



Estando aquí, nos dirigimos a la opción de;



Nos redirige al formulario para crear un nuevo repositorio;



Dejamos todo tal cual, solo agregamos un nombre y damos clic en crear.

Una vez que se ha creado el repositorio, ahora lo vamos a conectar con nuestro proyecto local.

Primero debemos decir de quien es el repositorio, escribimos;

```
ASUS TUF@LAPTOP-UFSLTUAE MINGW64 /c/Proyectos/practica01 (master)
$ git config user.name "Ing_Labra"
```

Debemos configurar el nombre y el correo;

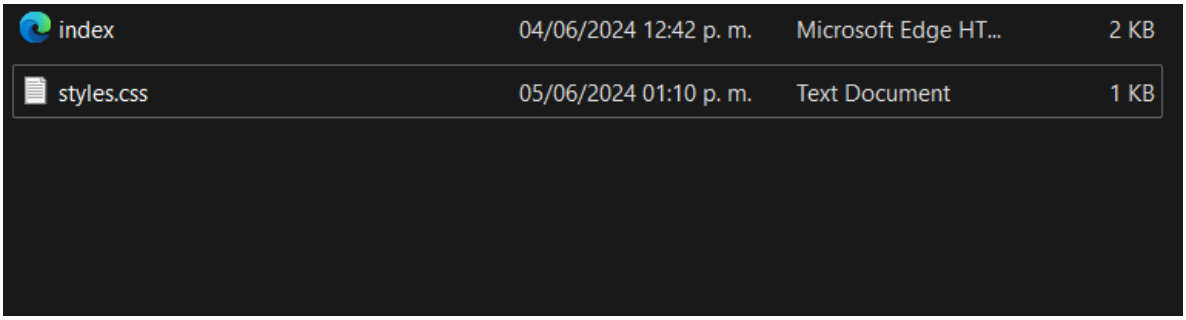
```
ASUS TUF@LAPTOP-UFSLTUAE MINGW64 /c/Proyectos/practica01 (master)
$ git config user.email "labrasantiagoh@gmail.com"
```

Para hacer la conexión con GIT HUB, escribimos;

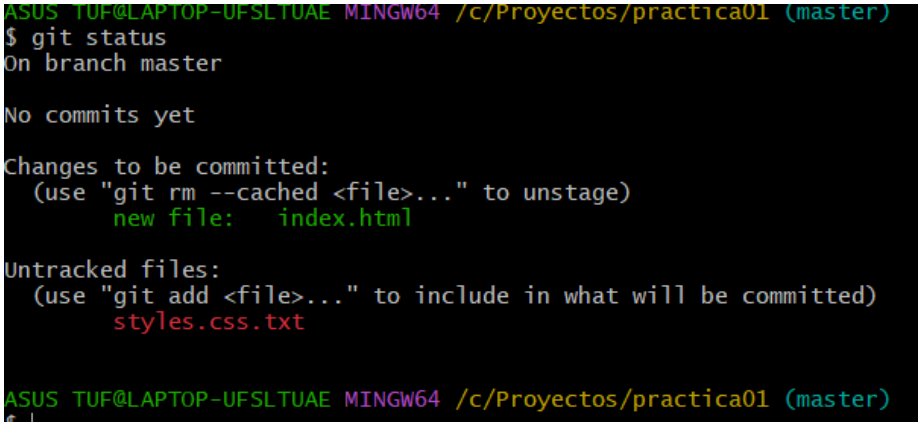
```
ASUS TUF@LAPTOP-UFSLTUAE MINGW64 /c/Proyectos/practica01 (master)
$ git remote add origin https://github.com/IngLabra/practica01.git
```

MANEJO DE ADD, COMMIT Y PUSH.

Vamos a crear un nuevo archivo en la carpeta que creamos;

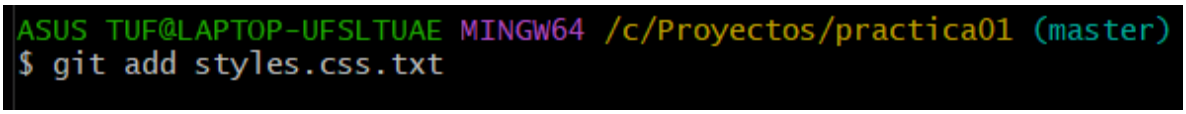


Una vez que hacemos eso, escribimos ahora GIT STATUS para saber que archivos de los que hemos creado no han sido subidos a GITHUB;



En rojo aparecerán los archivos que no han sido subidos y dentro del mismo esta el comando que debemos ejecutar para poder subirlos, sin embargo, cuando tenemos muchos archivos que no han sido subidos podemos usar GIT ADD . para que todos los archivos en rojo, sean subidos al mismo tiempo.

Como yo solo tengo un archivo, lo subiré separado;



Si vuelvo a escribir GIT STATUS, podrán ver que ya esta en verdecito.

Con add solo lo estamos agregando de forma provisoria, pero una vez que ya estamos seguros de que queremos guardar el archivo ahora escribimos;

```
ASUS TUF@LAPTOP-UFSLTUAE MINGW64 /c/Proyectos/practica01 (master)
$ git commit -m "Estoy agregando un nuevo archivo css"
[master (root-commit) 39797ef] Estoy agregando un nuevo archivo css
 2 files changed, 46 insertions(+)
 create mode 100644 index.html
 create mode 100644 styles.css.txt

ASUS TUF@LAPTOP-UFSLTUAE MINGW64 /c/Proyectos/practica01 (master)
$
```

Para que sea guardado escribimos GIT COMMIT – M “EL MENSAJE QUE QUIERAN AGREGAR”

Ahora vamos a pasar este archivo a nuestro repositorio remoto haciendo uso de PUSH;

DATO IMPORTANTE; Una rama es una línea de trabajo distinta a la principal que tenemos, cada línea de trabajo distinta que tengamos es una rama, Ej. Cada versión del proyecto (V 1.0.1, 1.0.2) es una rama.

La sintaxis para el PUSH es la siguiente;

```
ASUS TUF@LAPTOP-UFSLTUAE MINGW64 /c/Proyectos/practica01 (master)
$ git push -u desarrollo web master
```

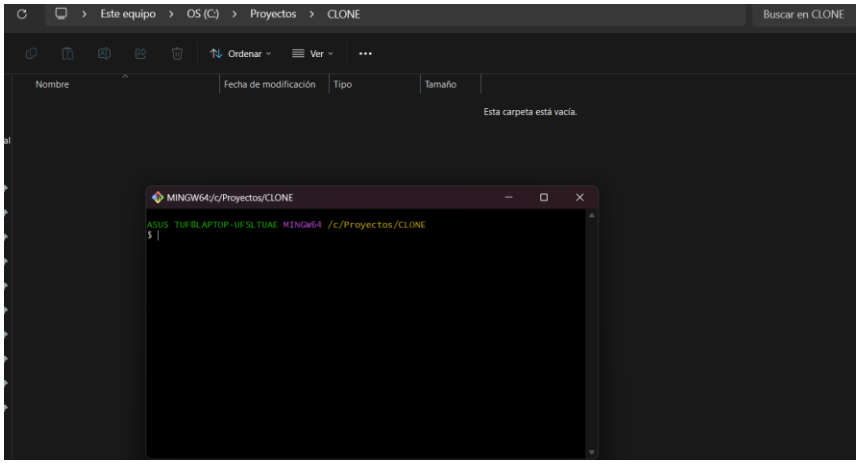
Si todo sale bien, debería aparecer este mensaje;

```
Excepci▒n no controlada: System.ComponentModel.Win32Exception: El identi
ficador de la ventana no es v▒lido
   en MS.Win32.ManagedWndProcTracker.HookUpDefWindowProc(IntPtr hwnd)
   en MS.Win32.ManagedWndProcTracker.OnAppDomainProcessExit()
   en MS.Internal.ShutDownListener.HandleShutDown(Object sender, EventAr
gs e)
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 254 bytes | 254.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/todocodeacademy/pruebagit.git
 * [new branch]      master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.
```

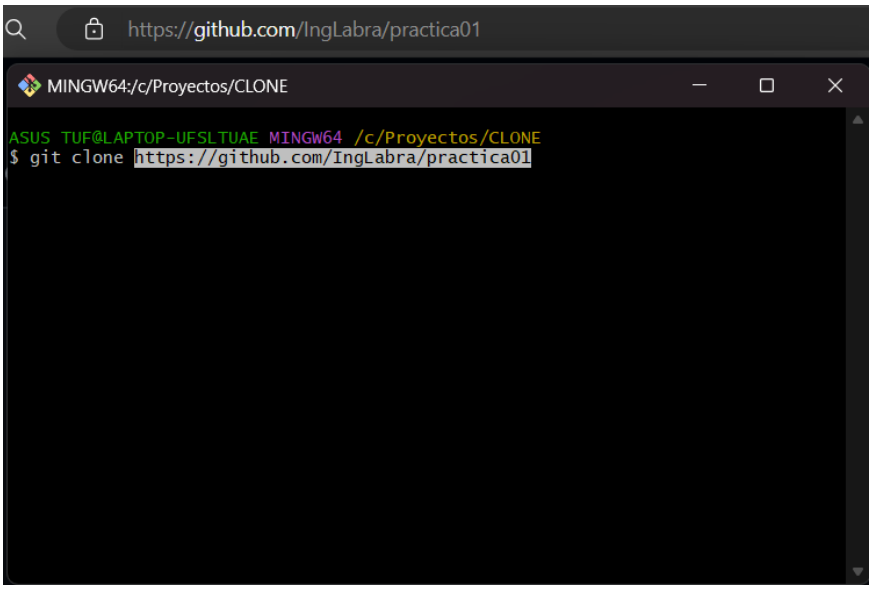
GIT CLONE y GIT PULL;

GIT clone se utiliza para descargar el repositorio completo desde una ubicación remota y luego almacenarlo localmente en el directorio.git.

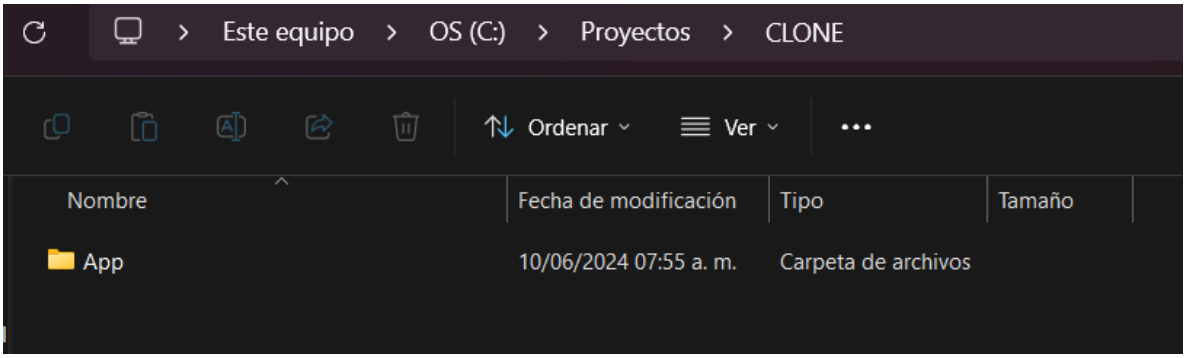
Primero, vamos a crear una carpeta donde vamos a clonar un proyecto de nuestro repositorio, esta carpeta debe estar vacía.



Abrimos GIT BASH en esta carpeta y escribimos GIT CLONE, junto a la URL del repositorio que quiero copiar, es decir que primero debemos entrar al repositorio en git hub, copiamos el link y lo pegamos junto al comando.



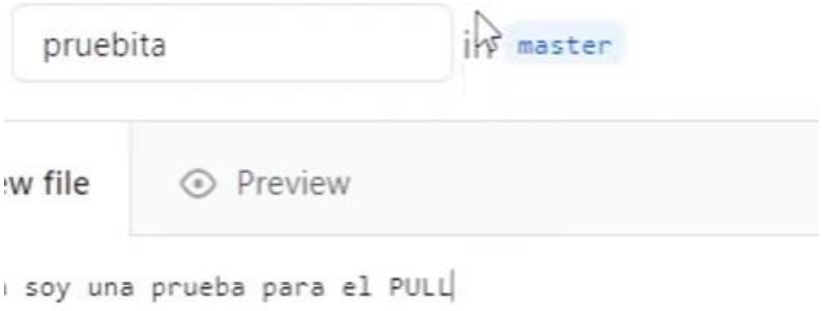
Una vez teniéndolo, damos enter y si miramos la carpeta que creamos, la carpeta del repositorio se habrá creado con todo y sus archivos;



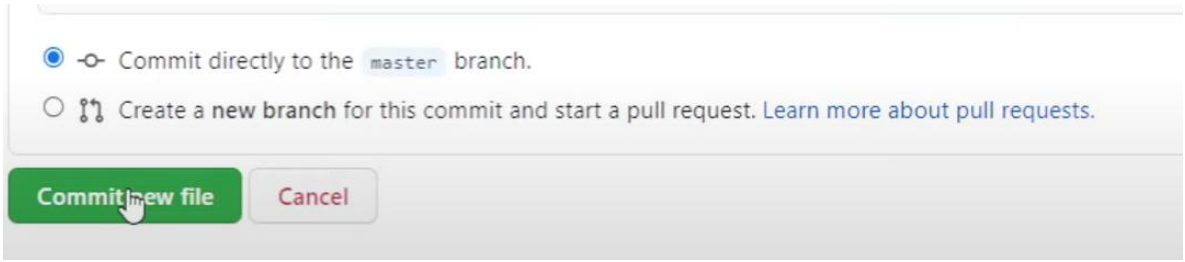
El comando git pull se emplea para extraer y descargar contenido desde un repositorio remoto y actualizar al instante el repositorio local para reflejar ese contenido.

Supongamos que estamos trabajando en conjunto con otro compañero y este compañero hizo un cambio, agrego un nuevo archivo y debes tenerlo en tu computadora también y para ello hacemos uso del PULL.

En nuestro repositorio vamos a crear un nuevo archivo con el nombre que quieran y un mensaje “ESTA ES UNA PRUEBA CON PULL”.

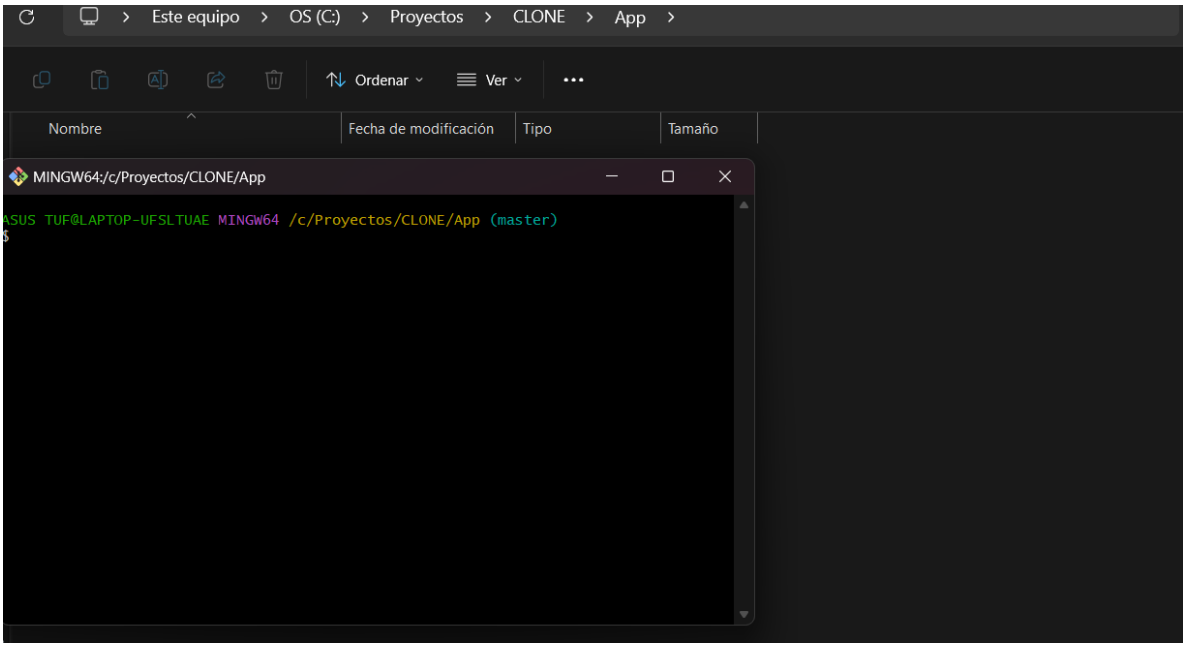


Es importante verificar la rama donde se esta creando el archivo. (En mi caso es en Master)

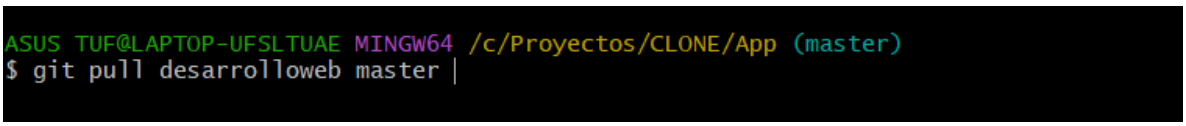


Ahora vamos hacer un commit, como lo hacemos localmente pero ahora desde GIT HUB, una vez hecho eso, en nuestro repositorio habrá aparecido el archivo, pero no localmente.

Nos dirigimos a la carpeta de nuestros proyectos clonados y estando aquí entramos a la carpeta del repositorio donde está el nuevo archivo de la misma forma en GIT BASH.



Estando aquí escribimos el comando;



Primero escribimos GIT PULL seguido del repositorio donde cree el archivo y finalmente la rama donde se encuentra el archivo.

```
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), 686 bytes | 68.00 KiB/s, done.
From https://github.com/todocodeacademy/pruebagit
 * branch            master     -> FETCH_HEAD
   a142d36..bd86bad  master     -> origin/master
Updating a142d36..bd86bad
Fast-forward
 pruebita | 1 +
 1 file changed, 1 insertion(+)
```

Al dar enter nos mostrara el o los archivos que se han añadido al repositorio y por automático los enviara a la carpeta.

Y solo verificamos en la carpeta si se ha creado.

Creación de ramas;

Supongamos que queremos crear una nueva versión de nuestros archivos para verificar si estos generan conflictos o si funcionan correctamente, para ello abrimos git bash en el proyecto que queremos crearle la nueva rama.

```
$ git branch
* master
```

Primero vamos a escribir el comando git Branch para saber en qué rama estamos actualmente.

Para crear una nueva rama escribimos el comando;

```
ASUS TUF@LAPTOP-UFSLTUAE MINGW64 /c/Proyectos/CLONE/App (master)
$ git branch rama01
```

Con esto, se crea una nueva rama y solo verificamos si se ha creado volviendo a escribir el comando;

```
$ git branch
* master
  rama1
```

Si de casualidad nos equivocamos al momento de escribir el nombre de la rama y quiero cambiarle el nombre, solo escribimos;

```
$ git branch -m rama1 ramita1
```

Primero escribiendo -m, seguido del nombre anterior de la rama y después el nuevo nombre;

```
$ git branch
* master
  ramita1
```

Ahora si quiero cambiar a la otra rama, lo único que tengo que hacer es escribir el comando;

```
$ git checkout ramita1
Switched to branch 'ramita1'
```

Con git checkout escribo el nombre de la rama donde quiero posicionarme y ahí nos permitirá ingresar a dicha rama.

Si vuelvo a cargar git Branch podremos verificar que ya estamos en otra rama;

```
$ git branch
  master
* ramita1
```

Ahora si quiero eliminar una rama o ya no la voy a usar, lo primero que debo hacer es salirme de la rama que quiero eliminar, en este caso ramita1, asi que vuelvo a la rama master y estando aquí aplico el siguiente comando;

```
$ git branch -d ramita1
```

El -d es de delete :D.

Seguido de la rama que quiero eliminar. (Recordando que primero debo salirme de esa rama)

Para crear archivos dentro de la rama, primero debemos seleccionarla y después escribimos;

```
$ touch "texto.txt"
```

Con esto, un nuevo archivo ha sido creado.

Debemos también confirmar si es un cambio exclusivo de la rama 1 o si se va hacia la rama Master.

Si volvemos a la rama Master, podremos notar que el archivo creado esta en Master.

```
$ git checkout master
Switched to branch 'master'
Your branch is ahead of 'origin/master' by 2 commits.
  (use "git push" to publish your local commits)
```

Para que estos cambios se queden en la rama que queremos vamos aplicar los mismos comandos de add (para que agregue los archivos que no se han confirmado) y git commit para confirmar.

```
$ git add .
```

```
$ git commit -m "estoy commiteando los arch"
```

Estos dos comandos deben ser ejecutados en la rama donde estamos aplicando los cambios.

SI VUELVO A LA RAMA MASTER, EN EL EXPLORADOR DE ARCHIVOS PODRAN NOTAR QUE LOS ARCHIVOS QUE CREAMOS EN LA RAMA1 SE HAN BORRADO. SI VOLVEMOS A LA RAMA1 LOS ARCHIVOS VUELVEN APARECER.

GIT DIFF Y MERGE;

Vamos a posicionarnos en la rama 01, recordando que actualmente contamos con dos ramas, la principal y la que creamos (RAMA1)

Cuando solemos tener mas de una rama, por ejemplo, 5, no estaremos cambiando rama por rama para ver las diferencias, para ello, usaremos el comando DIFF, el cual nos ayuda a conocer las diferencias que pueden existir entre ramas.

El comando funciona asi, escribo primero git diff seguido de las ramas que quiero comparar, en este caso quiero comparar master con rama1, al dar clic me arroga que archivos son diferentes entre ramas.

En mi caso me dice que existen dos archivos diferentes entre la rama principal y la rama1 (Aquellos que dicen new file)

```
$ git diff master rama1
diff --git a/texto.txt b/texto.txt
new file mode 100644
index 0000000..e69de29
diff --git a/texto2.txt b/texto2.txt
new file mode 100644
index 0000000..e69de29
```

Si cambio el orden, es decir, primero escribo rama1 y después master, me arroja esto;

```
$ git diff rama1 master
diff --git a/texto.txt b/texto.txt
deleted file mode 100644
index e69de29..0000000
diff --git a/texto2.txt b/texto2.txt
deleted file mode 100644
index e69de29..0000000
```

Me dice que master tiene un archivo menos, es decir, un archivo eliminado llamado texto y texto 2.

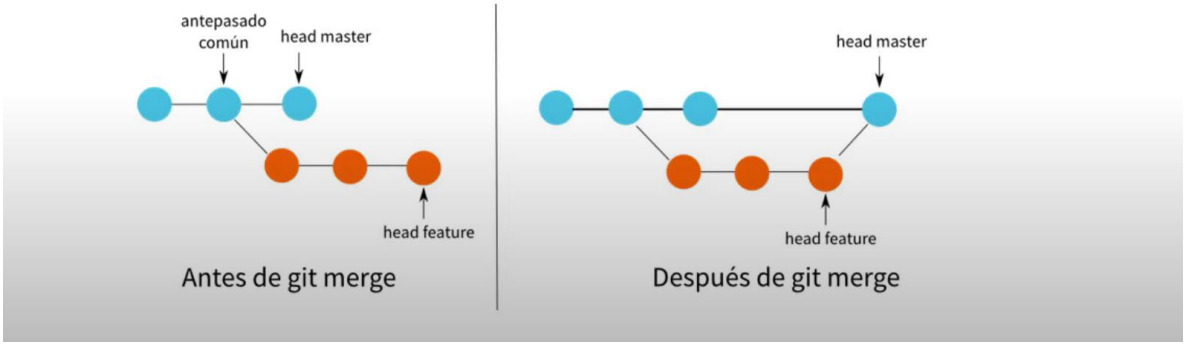
Una vez que hemos asegurado que las ramas son correctas, es decir, que los archivos los podemos agregar a la ramificación principal, aplicaremos MERGE.

El comando merge me permite unificar ramas en una sola para que si tengo diferencias en una rama, unificándolas podremos tener los mismos archivos.

```
> git diff rama1 rama2
```
Muestra las diferencias entre una rama y otra

```
> git checkout ramaAActualizar
```
Cambiar a la rama a actualizar

```
> git merge ramaOrigen ramaDestino
```
Cambiar la rama a actualizar



Si aplico el comando merge (Primero escribo git merge seguido de las ramas que quiero unificar) me enviara un mensaje de que ya estamos al día.

```
$ git merge rama1 master
Already up to date.
```

Esto pasa porque nos encontramos en la rama que esta más actualizada (Rama 1)

Cuando quiera hacer un merge debo posicionarme sobre la rama que va a recibir los cambios en ese momento, en este caso, si yo quiero recibir los cambios en master;

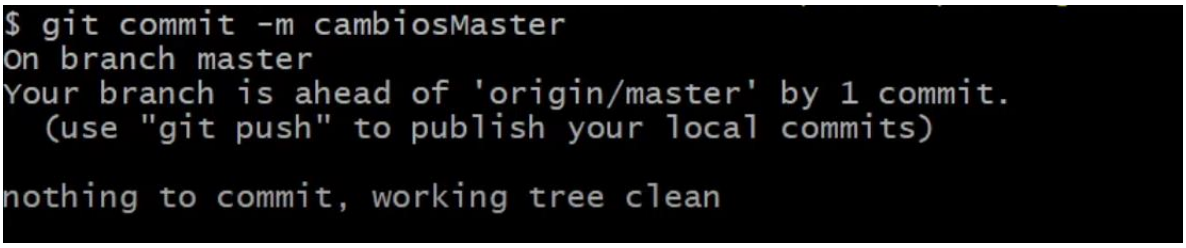
```
$ git checkout master
Switched to branch 'master'
```

Una vez que estoy en la rama que quiero que reciba los cambios, ahora si;

```
$ git merge rama1 master
Updating 1410358..bdf4f1c
Fast-forward
 texto.txt  | 0
 texto2.txt | 0
 2 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 texto.txt
 create mode 100644 texto2.txt
```

Con este mensaje confirmamos que efectivamente se hizo un cambio y ahora solo queda confirmarlo;

Con git add y git commit.



Si hacemos un checkout a master podremos ver que en efecto, es cine.