UNIVERSIDAD PRIVADA DEL NORTE

FACULTAD DE INGENIERÍA

GRUPO 6 – CAMPO SEMANA 1

CURSO: TÉCNICAS DE PROGRAMACIÓN ORIENTADA A OBJETOS

INTEGRANTES:

- **ARCA PÉREZ, JOSE ROBERTO**
- **TENA ALVARADO, JHOSTYN ANDRES**
- **BENANCIO MARIANO, WILIAN SNAYDER**
- **NARCISO CHECASACA, KAREN**
- **CALLE CORDOVA, DIEGO ALONSO**

DOCENTE:

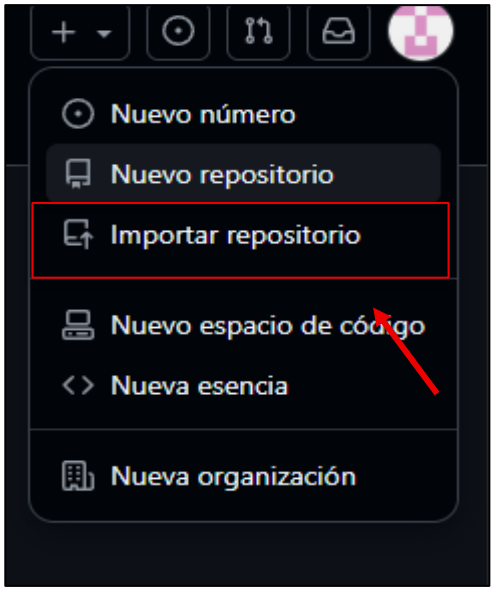
• TORRES RODRIGUEZ, MARTIN EDUARDO

Los Olivos de Pro, 2025

GUÍA DE PRIMEROS PASOS CON GIT

En este documento se detalla los pasos básicos para empezar a usar Git y GitHub. Aquí muestro cómo crear un repositorio local, clonar un repositorio remoto, navegar dentro del proyecto, hacer commits, trabajar con ramas (branches) y realizar la fusión de ramas (merge).

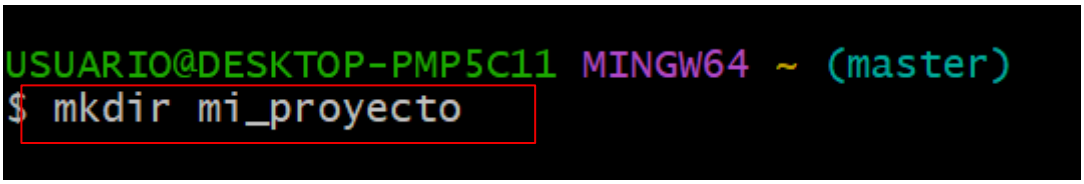
1.- CREAMOS UN NUEVO REPOSITORIO EN GITHUB



2.- UNA VEZ CREADO EL REPOSITORIO, COPIAMOS EL SIGUIENTE ENLACE.

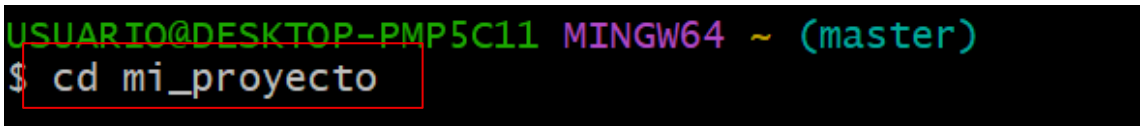
3.- **CREAMOS UN REPOSITORIO LOCAL EN GIT.**

Con el comando ***mkdir mi_proyecto***, que crea una carpeta llamada **mi_proyecto** en la ubicación actual. Esta carpeta servirá como directorio del repositorio Git que se inicializará posteriormente.



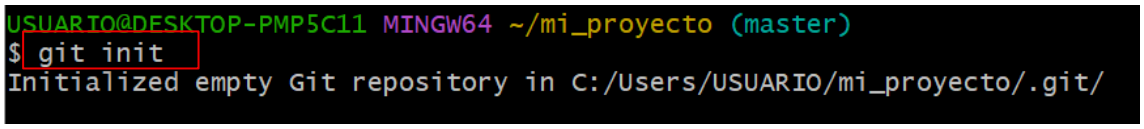
4.- ENTRAMOS EN LA CARPETA DEL PROYECTO

Posteriormente, con el comando ***cd mi_proyecto*** entramos en la carpeta del proyecto. En este directorio es donde luego inicializaré el repositorio Git usando git init



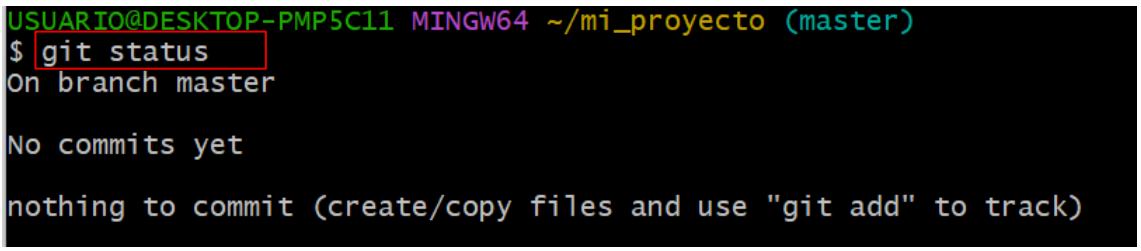
5.- **INICIALIZAMOS UN REPOSITORIO GIT EN LA CARPETA ACTUAL**

se observa que se ejecutó el comando ***git init***., Git indica que se ha creado un repositorio vacío en la carpeta ***C:/Users/USUARIO/mi_proyecto/.git/,*** confirmando que la inicialización fue exitosa.



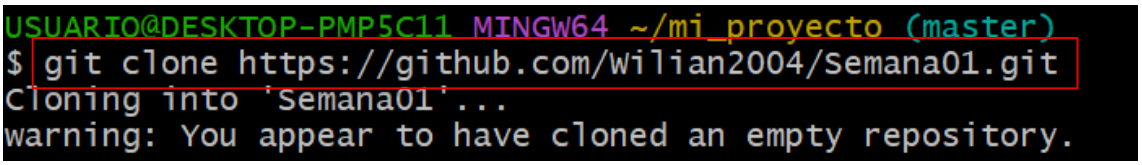
6**.- VER EL ESTADO ACTUAL DEL REPOSITORIO GIT**

Con el comando ***git status*** verifico el estado del repositorio. En este caso, Git me indica que estamos en la rama master, que todavía no hay commits y que no hay archivos para seguir, es decir, el repositorio está vacío.



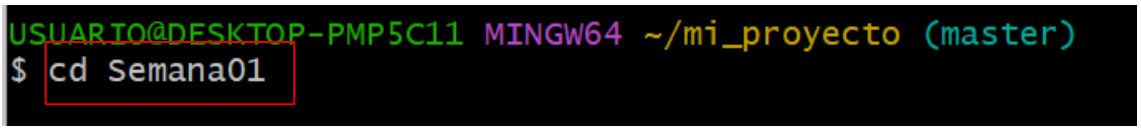
7.- CLONAMOS UN REPOSITORIO DE GITHUB A LA MAQUINA LOCAL

Con el comando ***git clone https://github.com/wilian2004/Semana01.git*** traigo el repositorio remoto a mi máquina. Git crea una carpeta llamada Semana01 dentro de mi_proyecto. En este caso, Git me avisa con una advertencia que el repositorio está vacío, porque aún no tiene archivos ni commits.



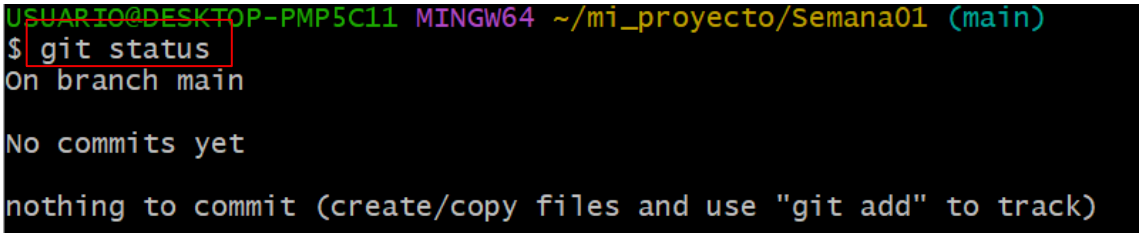
8.- INGRESAMOS AL REPOSITORIO QUE CLONAMOS

Con el comando ***cd Semana01*** entro en la carpeta del repositorio que acabo de clonar desde GitHub. A partir de aquí, puedo trabajar directamente con los archivos del repositorio y realizar commits, crear ramas o cualquier otra acción de Git.



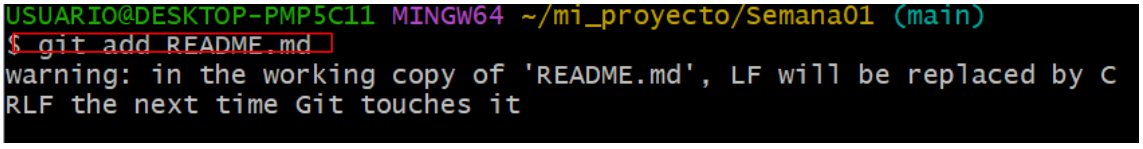
9.- VER EL ESTADO DEL REPOSITORIO

Con el comando ***git status*** reviso el estado del repositorio clonado. Git indica que estamos en la rama ***main***, que no hay commits y que no hay archivos para seguir, es decir, el repositorio sigue vacío.



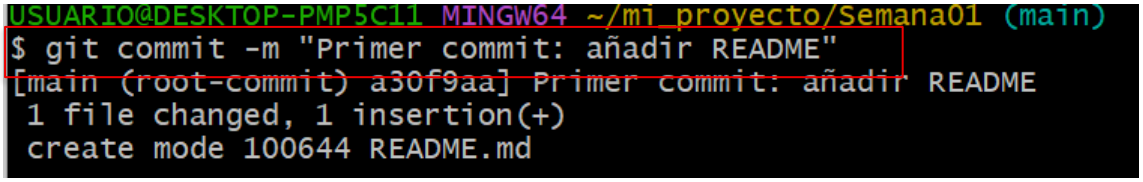
10.- AGREGAMOS UN ARCHIVO USANDO EL COMANDO ADD.

Con ***git add README.md*** preparo el archivo README.md para guardarlo en el próximo commit

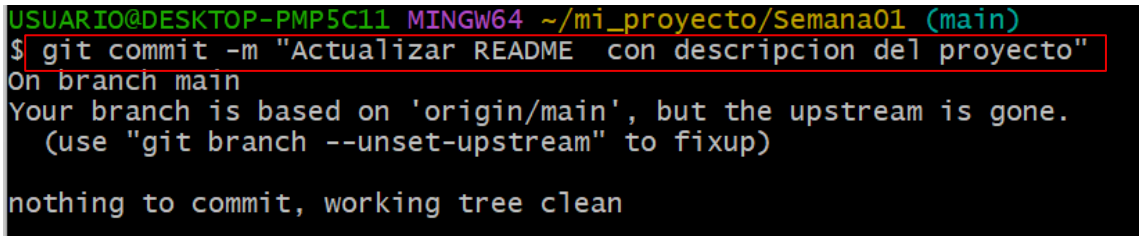


11.- HACEMOS NUESTRO PRIMER COMMIT

Con el comando ***git commit -m "Primer commit: añadir README"*** guardo los cambios del archivo README.md en el repositorio. Git confirma que se creó el commit y que el archivo fue agregado correctamente.

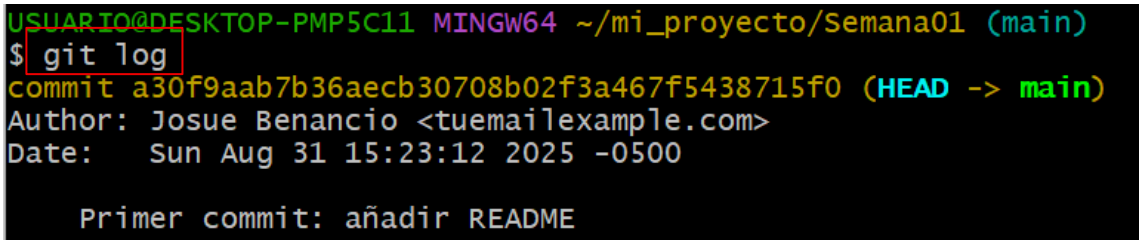


12.- CAMBIAMOS LA DESCRIPCIÓN USANDO UN NUEVO COMMIT.



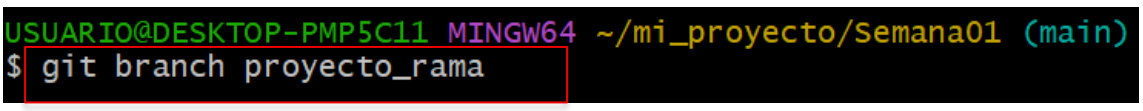
13.- VEMOS EL HISTORIAL DE COMMITS EN NUESTRO REPOSITORIO GIT.

Con el comando ***git log*** puedo visualizar el historial de commits del repositorio. Esto me permite ver los mensajes de cada commit, quién los hizo y cuándo, ayudándome a llevar un registro de los cambios realizados.



14.- CREAMOS UNA RAMA NUEVA LLAMADA "PROYECTO_RAMA."

Con el comando ***git branch proyecto_rama*** creo una nueva rama en el repositorio llamada "proyecto_rama". Esto me permite trabajar en cambios separados sin afectar la rama principal (main).



15.- CAMBIAMOS A LA RAMA QUE ACABAMOS DE CREAR.

Con el comando ***git checkout proyecto_rama*** me muevo a la rama que acabamos de crear. A partir de aquí, cualquier cambio que haga se registrará en esta rama sin afectar la rama main

```
USUARIO@DESKTOP-PMP5C11 MINGW64 ~/mi_proyecto/Semana01 (main)
$ git checkout proyecto_rama
Switched to branch 'proyecto_rama'
```

16.- VER LAS RAMAS EXISTENTES

Con el comando ***git branch*** puedo listar todas las ramas que existen en el repositorio. La rama en la que estoy actualmente, por ejemplo la que creé proyecto_rama, aparecerá marcada con un asterisco (*).

```
USUARIO@DESKTOP-PMP5C11 MINGW64 ~/mi_proyecto/Semana01 (proyecto_rama)
$ git branch
  main
* proyecto_rama
```

17.- FUSIONAMOS LOS CAMBIOS A LA RAMA PRINCIPAL

Con el comando ***git merge proyecto_rama*** puedo integrar los cambios realizados en la rama "proyecto_rama" dentro de la rama principal (main). Esto nos permite que las modificaciones hechas en la rama de trabajo se unan al proyecto principal sin perder historial de cambios.

```
USUARIO@DESKTOP-PMP5C11 MINGW64 ~/mi_proyecto/Semana01 (main)
$ git merge proyecto_rama
Already up to date.
```

18.-Crear repositorio local

Primero ingresé a la carpeta de mi proyecto, luego la inicialicé como un repositorio de Git para que empiece a rastrear los cambios. Finalmente, cambié el nombre de la rama principal de master a main, siguiendo la convención actual recomendada.

```
USUARIO@DESKTOP-PMP5C11 MINGW64 ~/eclipse-workspace/POOGitCasos (master)
$ cd ~/eclipse-workspace/POOGitCasos

USUARIO@DESKTOP-PMP5C11 MINGW64 ~/eclipse-workspace/POOGitCasos (master)
$ git init
Initialized empty Git repository in C:/Users/USUARIO/eclipse-workspace/PooGitCas
os/.git/

USUARIO@DESKTOP-PMP5C11 MINGW64 ~/eclipse-workspace/POOGitCasos (master)
$ git branch -M main
```

19.-Crear archivos iniciales

Se creó el archivo README.md con información básica del proyecto y el archivo guia_git.md para documentar el uso de Git en la práctica.

```
USUARIO@DESKTOP-PMP5C11 MINGW64 ~/eclipse-workspace/POOGitCasos (main)
$ echo "# Proyecto POO + Git" > README.md
```

```
USUARIO@DESKTOP-PMP5C11 MINGW64 ~/eclipse-workspace/POOGitCasos (main)
$ echo "##Guia de Git para la practica" > guia_git.md
```

20.-Primer commit

Se agregaron los archivos al área de preparación (git add .) y se registró el primer commit (git commit -m "Versión inicial del proyecto con casos"). Con esto se guardó el estado inicial del proyecto en el historial de Git.

```
USUARIO@DESKTOP-PMP5C11 MINGW64 ~/eclipse-workspace/POOGitCasos (main)
$ git add .
```

```
USUARIO@DESKTOP-PMP5C11 MINGW64 ~/eclipse-workspace/POOGitCasos (main)
$ git commit -m "Version inicial del proyecto con casos"
```

21.-Conectar con GitHub

El repositorio local se enlazó con un repositorio remoto en GitHub y luego se subieron los cambios iniciales con git push -u origin main.

```
USUARIO@DESKTOP-PMP5C11 MINGW64 ~/eclipse-workspace/POOGitCasos (main)
$ git remote set-url origin https://github.com/Loky214/PracticaCasos.git

USUARIO@DESKTOP-PMP5C11 MINGW64 ~/eclipse-workspace/POOGitCasos (main)
$ git push -u origin main
```

22.-Ver estado del repositorio

Se verificó el estado de los archivos con git status

```
USUARIO@DESKTOP-PMP5C11 MINGW64 ~/eclipse-workspace/POOGitCasos (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
```

23.-Subir cambios de los casos prácticos

Cada caso de programación (Caso 1, Caso 2 y Caso 3) se fue desarrollando en su respectiva carpeta. Después de implementar cada uno, se registraron cambios en Git mediante git add, git commit y git push, con mensajes descriptivos que indicaban qué caso se había implementado.

```
USUARIO@DESKTOP-PMP5C11 MINGW64 ~/eclipse-workspace/POOGitCasos (main)
$ git add .

USUARIO@DESKTOP-PMP5C11 MINGW64 ~/eclipse-workspace/POOGitCasos (main)
$ git commit -m "Implementando Caso 1: Lectura de datos simples con Scanner"
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean

USUARIO@DESKTOP-PMP5C11 MINGW64 ~/eclipse-workspace/POOGitCasos (main)
$ git push
Everything up-to-date
```

```
USUARIO@DESKTOP-PMP5C11 MINGW64 ~/eclipse-workspace/POOGitCasos (main)
$ git add .

USUARIO@DESKTOP-PMP5C11 MINGW64 ~/eclipse-workspace/POOGitCasos (main)
$ git commit -m "Implementando Caso 2: Clase Estudiante con atributos privados y
 scanner"
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean

USUARIO@DESKTOP-PMP5C11 MINGW64 ~/eclipse-workspace/POOGitCasos (main)
$ git push
Everything up-to-date
```

```
USUARIO@DESKTOP-PMP5C11 MINGW64 ~/eclipse-workspace/POOGitCasos (main)
$ git add .

USUARIO@DESKTOP-PMP5C11 MINGW64 ~/eclipse-workspace/POOGitCasos (main)
$ git commit -m
error: switch `m' requires a value

USUARIO@DESKTOP-PMP5C11 MINGW64 ~/eclipse-workspace/POOGitCasos (main)
$ git commit -m "Implementando Caso 3: Clase CuentaBancaria con validacion de sa
ldo"
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean

USUARIO@DESKTOP-PMP5C11 MINGW64 ~/eclipse-workspace/POOGitCasos (main)
$ git push
Everything up-to-date
```

24.-Cambiando entre ramas

Se creó una nueva rama llamada desarrollo con git branch desarrollo y se cambió a ella con git checkout desarrollo. En esa rama se realizaron cambios de prueba y luego se fusionó con la rama principal (main) usando git merge.

```
USUARIO@DESKTOP-PMP5C11 MINGW64 ~/eclipse-workspace/POOGitCasos (main)
$ git branch desarrollo

USUARIO@DESKTOP-PMP5C11 MINGW64 ~/eclipse-workspace/POOGitCasos (main)
$ git checkout desarrollo
Switched to branch 'desarrollo'

USUARIO@DESKTOP-PMP5C11 MINGW64 ~/eclipse-workspace/POOGitCasos (desarrollo)
$ git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.

USUARIO@DESKTOP-PMP5C11 MINGW64 ~/eclipse-workspace/POOGitCasos (main)
$ git merge desarrollo
Already up to date.
```

Proyectos en java:

Caso1:

```java
package caso1;

import java.util.Scanner;

public class UsuarioSimple {
    private String nombre;
    private int edad;

    public UsuarioSimple(String nombre, int edad) {
        this.nombre = nombre;
        this.edad = edad;
    }

    public String getNombre() { return nombre; }
    public int getEdad() { return edad; }

    public String presentacion() {
        return "Hola, soy " + nombre + " y tengo " + edad + " años.";
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Nombre: ");
        String n = sc.nextLine();
        System.out.print("Edad: ");
        int e = sc.nextInt();
        UsuarioSimple u = new UsuarioSimple(n, e);
        System.out.println(u.presentacion());
        sc.close();
    }
}
```

```
Nombre: Wilian
Edad: 21
Hola, soy Wilian y tengo 21 años.
```

Caso 2:

```java
package caso2;

import java.util.Scanner;

public class EstudianteInteractivo {
    private String nombre;
    private int edad;
    private String carrera;

    public EstudianteInteractivo(String nombre, int edad, String carrera) {
        this.nombre = nombre;
        this.edad = edad;
        this.carrera = carrera;
    }

    public String resumen() {
        return "Soy " + nombre + ", tengo " + edad + " años y estudio " + carrera + ".";
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Nombre: ");
        String n = sc.nextLine();
        System.out.print("Edad: ");
        int e = sc.nextInt();
        sc.nextLine(); // limpiar buffer
        System.out.print("Carrera: ");
        String c = sc.nextLine();
        EstudianteInteractivo est = new EstudianteInteractivo(n, e, c);
        System.out.println(est.resumen());
        sc.close();
    }
}
```

```
Nombre: Wilian
Edad: 21
Carrera: Ingenieria de sistemas
Soy Wilian, tengo 21 años y estudio Ingenieria de sistemas.
```

Caso 3:

```java
package caso3;

import java.util.Scanner;

public class CuentaBancaria {
    private String titular;
    private double saldo;

    public CuentaBancaria(String titular, double saldoInicial) {
        this.titular = titular;
        this.saldo = saldoInicial;
    }

    public void depositar(double monto) {
        saldo += monto;
    }

    public void retirar(double monto) {
        if (monto <= saldo) {
            saldo -= monto;
        } else {
            System.out.println("Fondos insuficientes.");
        }
    }

    public void mostrarSaldo() {
        System.out.println("Titular: " + titular + " | Saldo: $" + saldo);
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Nombre del titular: ");
        String t = sc.nextLine();
        System.out.print("Saldo inicial: ");
        double s = sc.nextDouble();

        CuentaBancaria cuenta = new CuentaBancaria(t, s);
        cuenta.mostrarSaldo();

        System.out.print("Monto a depositar: ");
        cuenta.depositar(sc.nextDouble());
        cuenta.mostrarSaldo();

        System.out.print("Monto a retirar: ");
        cuenta.retirar(sc.nextDouble());
        cuenta.mostrarSaldo();

        sc.close();
    }
}
```

```
Nombre del titular: Wilian
Saldo inicial: 2000
Titular: Wilian | Saldo: $2000.0
Monto a depositar: 1000
Titular: Wilian | Saldo: $3000.0
Monto a retirar: 2000
Titular: Wilian | Saldo: $1000.0
```