Alejandro Costich Sandoval
Jessica Abril Quintero Castillo
Fernanda Nicole García Melendrez

PRIMERA PARTE GIT

1. Cree un nuevo repositorio Git con el nombre de PracticasGit
2. Inicialice y configure el repositorio
3. Cree un archivo llamado Laberinto que contenga el siguiente código:

```
import random
FILAS = 5
COLUMNAS = 5
META_FILA = FILAS - 1
META_COLUMNA = COLUMNAS - 1
def main():
  laberinto = [[False] * COLUMNAS for _ in range(FILAS)]
  fila_jugador, columna_jugador = 0, 0
  # Inicializar el laberinto
  laberinto[META_FILA][META_COLUMNA] = True  # Meta
  # Bucle principal del juego
  while True:
    # Imprimir el laberinto
    for i in range(FILAS):
      for j in range(COLUMNAS):
        if i == fila_jugador and j == columna_jugador:
          print("X ", end='')  # Jugador
        elif laberinto[i][j]:
          print("O ", end='')  # Meta
        else:
          print(". ", end='')  # Espacio vacío
      print()
    # Verificar si el jugador alcanzó la meta
    if fila_jugador == META_FILA and columna_jugador == META_COLUMNA:
      print("\n¡Has alcanzado la meta! ¡Felicidades!")
      break
    # Solicitar al jugador el siguiente movimiento
    movimiento = input("\nIngrese su siguiente movimiento (w: arriba, s: abajo, a: izquierda, d: derecha): ")

    # Mover al jugador según la entrada del usuario
    if movimiento == 'w' and fila_jugador > 0:
      fila_jugador -= 1
    elif movimiento == 's' and fila_jugador < FILAS - 1:
      fila_jugador += 1
    elif movimiento == 'a' and columna_jugador > 0:
      columna_jugador -= 1
    elif movimiento == 'd' and columna_jugador < COLUMNAS - 1:
      columna_jugador += 1
    else:
      print("Movimiento no válido.")

if __name__ == "__main__":
  main()
```

5. Cree un segundo archivo nómbralo como LaberintoFunciones, optimizando el código realizando las siguientes funciones inicializar_laberinto, imprimir_laberinto, verificar_meta, obtener_movimiento, mover_jugador y su función principal.

Alejandro Costich Sandoval
Jessica Abril Quintero Castillo
Fernanda Nicole García Melendrez

school tmp > PracticasGit > LaberintoFunciones.py > ...

```python
import random

FILAS = 5
COLUMNAS = 5
META_FILA = FILAS - 1
META_COLUMNA = COLUMNAS - 1

def inicializar_laberinto():
    laberinto = [[False] * COLUMNAS for _ in range(FILAS)]
    laberinto[META_FILA][META_COLUMNA] = True  # Meta
    return laberinto

def imprimir_laberinto(laberinto, fila_jugador, columna_jugador):
    for i in range(FILAS):
        for j in range(COLUMNAS):
            if i == fila_jugador and j == columna_jugador:
                print("X ", end='')  # Jugador
            elif laberinto[i][j]:
                print("O ", end='')  # Meta
            else:
                print(". ", end='')  # Espacio vacío
        print()

def verificar_meta(fila_jugador, columna_jugador):
    return fila_jugador == META_FILA and columna_jugador == META_COLUMNA

def obtener_movimiento():
    return input("\nIngrese su siguiente movimiento (w: arriba, s: abajo, a: izquierda, d: derecha): ")

def mover_jugador(fila_jugador, columna_jugador, movimiento):
    if movimiento == 'w' and fila_jugador > 0:
        fila_jugador -= 1
    elif movimiento == 's' and fila_jugador < FILAS - 1:
        fila_jugador += 1
    elif movimiento == 'a' and columna_jugador > 0:
        columna_jugador -= 1
    elif movimiento == 'd' and columna_jugador < COLUMNAS - 1:
        columna_jugador += 1
    else:
        print("Movimiento no válido.")
    return fila_jugador, columna_jugador

def main():
    laberinto = inicializar_laberinto()
    fila_jugador, columna_jugador = 0, 0

    while True:
        imprimir_laberinto(laberinto, fila_jugador, columna_jugador)
        if verificar_meta(fila_jugador, columna_jugador):
            print("\n¡Has alcanzado la meta! ¡Felicidades!")
            break

        movimiento = obtener_movimiento()
        fila_jugador, columna_jugador = mover_jugador(fila_jugador, columna_jugador, movimiento)

if __name__ == "__main__":
    main()
```

6. Deja ambos archivos en el estado 2.

```
C:\Users\FNico\OneDrive\Desktop\school tmp\PracticasGit>git add LaberintoFunciones.py

C:\Users\FNico\OneDrive\Desktop\school tmp\PracticasGit>git add Laberinto.py
```

7. Realiza un cambio en LaberintoFunciones que te permita cambiar las teclas de movimiento del laberinto. Dale clic en guardad y a continuación revisa con el comando Git Status. ¿Qusucedió?
   Notifica que ha habido un cambio en el archivo, pero que no se ha hecho un commit todavía.

Alejandro Costich Sandoval
Jessica Abril Quintero Castillo
Fernanda Nicole García Melendrez

```
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   Laberinto.py
        new file:   LaberintoFunciones.py

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   LaberintoFunciones.py
```

8. Vuelve a dejar LaberintoFunciones en el 2do. estado y escribe tus conclusiones.
   Una vez hecho esto, solo se verá en la lista de cambios que necesitan commit.

```
C:\Users\FNico\OneDrive\Desktop\school tmp\PracticasGit>git status
On branch master

No commits yet

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

9. Agrega la fecha actual a ambos archivos y guardarlos. Utiliza el comando git add – all y describe lo sucedido.
   Todos los archivos pasaron a estado 2.
10. Aplica el comando git commit -m “El setup inicial del proyecto” y aplica el comando git status. Comenta lo sucedido:
    Los archivos pasaron al estado 3 (repo local), por lo que las branches están limpias, no se necesitan adds ni commits.

```
C:\Users\FNico\OneDrive\Desktop\school tmp\PracticasGit>git status
On branch master
nothing to commit, working tree clean
```

11. Realiza un cambio en ambos archivos agregando el nombre de los participantes del equipo. Deja LaberintoFunciones en el 2do. Estado. Revisa con Git Status y que se encuentre en el estado indicado(verde: LaberintoFunciones) y (rojo: Laberinto) y aplica el comando git commit ¿Qué sucedió?

Alejandro Costich Sandoval
Jessica Abril Quintero Castillo
Fernanda Nicole García Melendrez

```
nothing to commit, working tree clean

C:\Users\FNico\OneDrive\Desktop\school tmp\PracticasGit>git add LaberintoFunciones.

C:\Users\FNico\OneDrive\Desktop\school tmp\PracticasGit>git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   LaberintoFunciones.py

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   Laberinto.py
```

```
C:\Users\FNico\OneDrive\Desktop\school tmp\PracticasGit>git commit -m "caca"
[master 94643c8] caca
 1 file changed, 6 insertions(+)

C:\Users\FNico\OneDrive\Desktop\school tmp\PracticasGit>git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   Laberinto.py
```

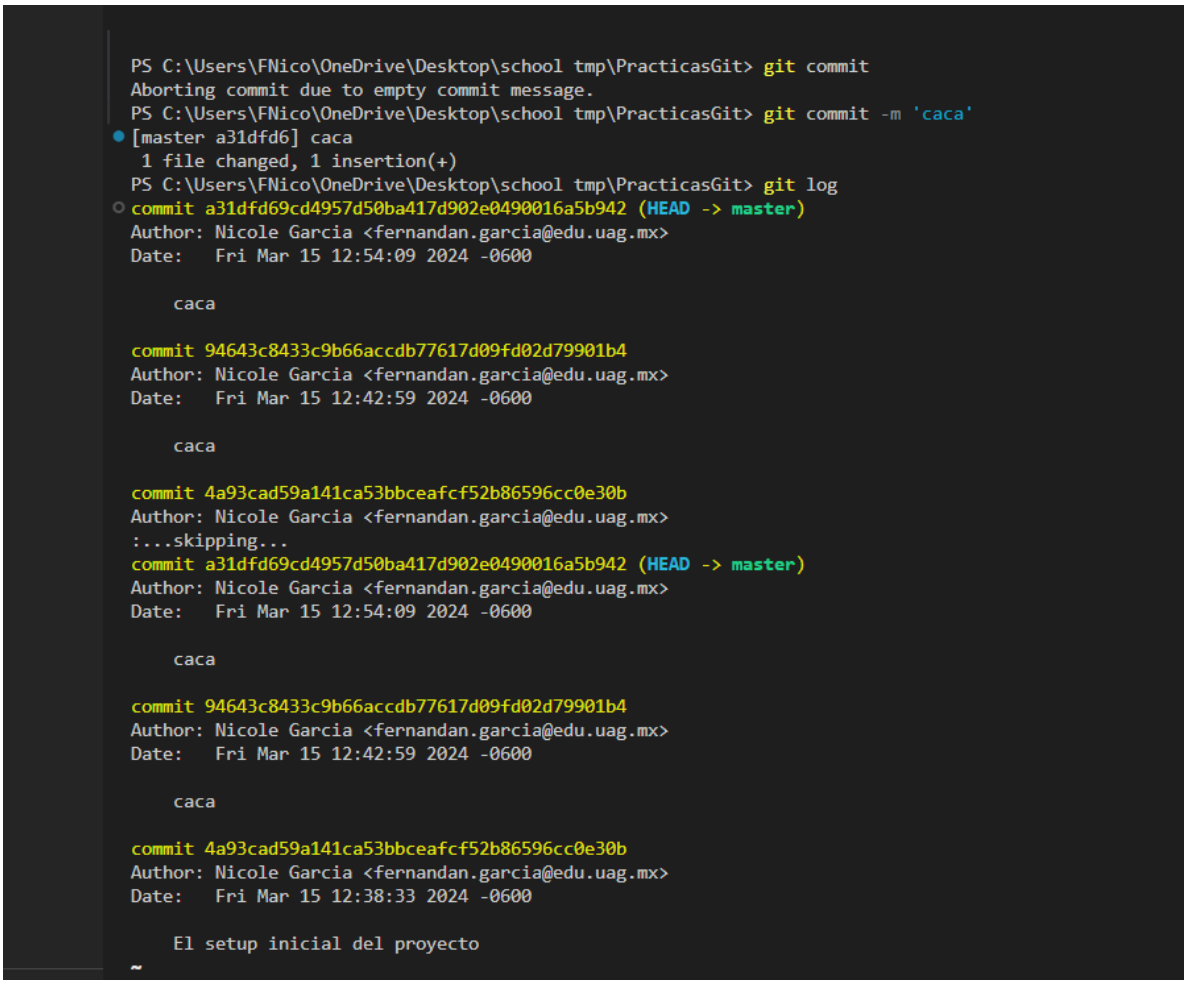
Laberinto necesita add y commit.

12. En el esapacio del editor escribe “Modificando nuestro archivo LaberintoFunciones” posteriormente presiona la tecla **ESC + :wq**

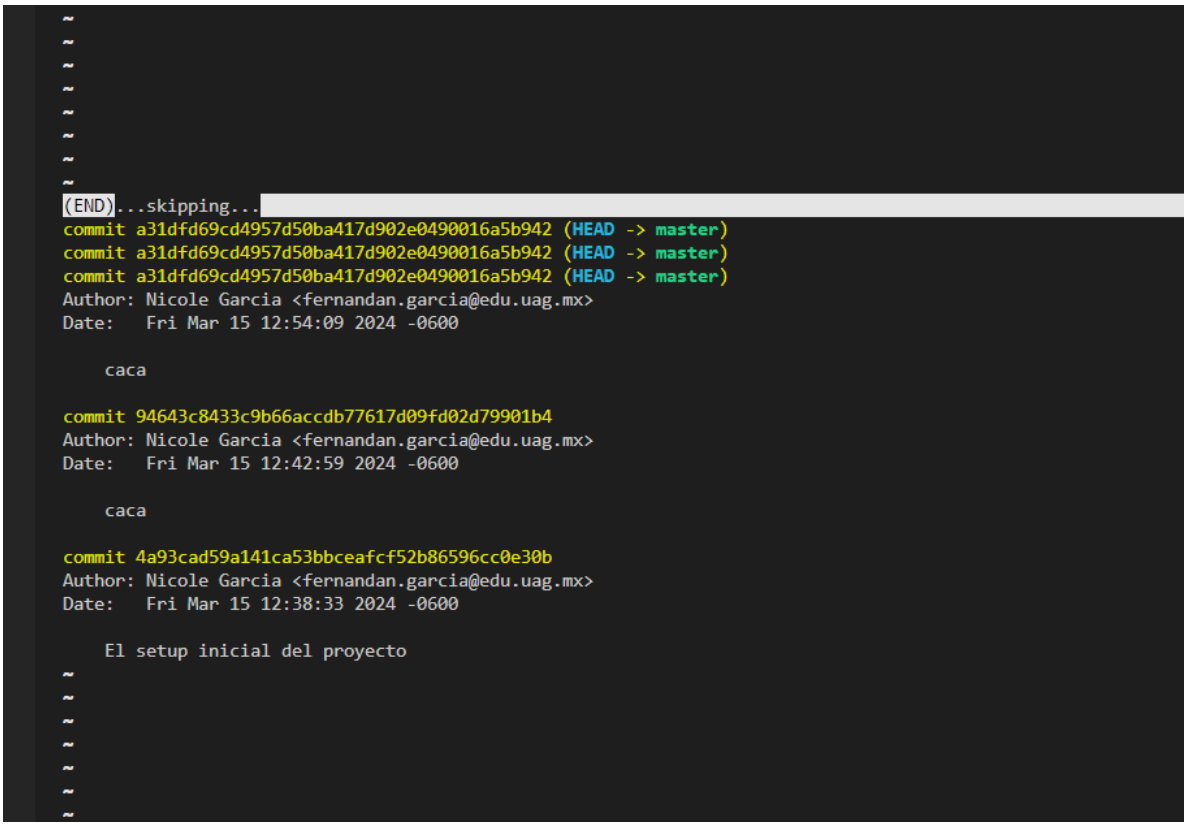
¿Cómo ver el estado de los Commits?
git log.

Escribe en la línea de comando **git log,** realiza una captura de pantalla y agrega

Alejandro Costich Sandoval
Jessica Abril Quintero Castillo
Fernanda Nicole García Melendrez



Alejandro Costich Sandoval
Jessica Abril Quintero Castillo
Fernanda Nicole García Melendrez



Investiga que es el hash del commit:

En el contexto de Git, el "hash del commit" se refiere a un valor único que se asigna a cada commit en un repositorio Git. Este hash es una cadena de 40 caracteres hexadecimal generada utilizando el algoritmo de hash SHA-1. Este hash se calcula a partir de la información del commit, incluyendo el contenido de los archivos en ese punto en el tiempo, el mensaje del commit, el autor, la marca de tiempo, y el hash del commit padre (para commits que no son el primero en una rama).
El hash del commit es esencialmente una "huella digital" única que identifica de manera única ese commit en particular dentro del historial del repositorio. Esto significa que cada commit tiene su propio hash único y ningún otro commit en el repositorio debe tener el mismo hash.
Algunos aspectos importantes sobre el hash del commit en Git incluyen:
Unicidad: Cada commit tiene un hash único que lo identifica de manera única en el historial del repositorio.
Integridad: El hash del commit se calcula utilizando la información del commit, lo que significa que incluso un pequeño cambio en los archivos o en la información del commit resultará en un hash completamente diferente. Esto ayuda a garantizar la integridad de los datos en el repositorio.
Referencia: Los hashes de commit se utilizan para referenciar commits en varios contextos dentro de Git, como al hacer referencia a un commit específico al realizar operaciones como fusiones, ramificaciones, reversiones, etc.

Alejandro Costich Sandoval
Jessica Abril Quintero Castillo
Fernanda Nicole García Melendrez

Corto y largo: Aunque los hashes de commit completos son de 40 caracteres, es común que se usen solo los primeros caracteres (generalmente 7 o 8) para referirse a un commit de manera abreviada. Git suele ser lo suficientemente inteligente como para reconocer un hash único incluso con solo unos pocos caracteres.

En resumen, el hash del commit en Git es una parte fundamental del sistema de control de versiones que proporciona una identificación única y una garantía de integridad para cada commit en el historial del repositorio.