

Puesta en Producción Segura

T1A2

Scripts básicos



Cursos 23/24 – IES La Marisma

Trabajo Práctico 2

Pedro Manuel García Álvarez

ÍNDICE

1.- Haz un script llamado <code>multiplo10.sh</code> que pida un número por teclado e indique si es un múltiplo de 10 o no.	6
1.1.- Abre la terminal (CTRL+ALT+T) y navega hasta el directorio donde se guarda el archivo.	6
1.2.- Abre el editor de texto nano y escribe el siguiente ejercicio en un archivo llamado <code>multiplo10.sh</code>	6
1.3.- Guarda el archivo con el nombre " <code>multiplo10.sh</code> ".	6
1.4.- Escribe los siguientes comandos:.....	6
1.5.- Explicación del código línea a línea :	7
2.- Haz un script llamado <code>altura_mayor.sh</code>, que pida por teclado la altura en centímetros de tres personas y nos diga la mayor de las 3 en metros.	8
2.1.- Abre la terminal (CTRL+ALT+T) y navega hasta el directorio donde se guarda el archivo.	8
2.2.- Abre el editor de texto nano y escribe el siguiente ejercicio en un archivo llamado <code>altura_mayor.sh</code>	8
2.3.- Guarda el archivo con el nombre " <code>altura_mayor.sh</code> ".	8
2.4.- Escribe los siguientes comandos:.....	8
2.5.- Explicación del código línea a línea :	9
3.- Haz un script llamado <code>cuenta10ficheros.sh</code>, que nos diga si en el directorio pasado como parámetro hay más de 10 ficheros o no, es decir, los directorios no deben ser contados.	10
3.1.- Abre la terminal (TRL+ALT+T) y navega hasta el directorio donde se guarda el archivo.	10
3.2.- Abre el editor de texto nano y escribe el siguiente ejercicio en un archivo llamado <code>cuenta10ficheros.sh</code>	10
3.3.- Guarda el archivo con el nombre " <code>cuenta10ficheros.sh</code> ".	10
3.4.- Escribe los siguientes comandos:.....	11
3.5.- Explicación del código línea a línea :	11

4.- Haz un script llamado `decada_edad.sh`, que dada una edad que introducimos por pantalla, nos diga como resultado la década en la que nacimos (ejemplo 70, 80, 90 ...). 13

4.1.- Abre la terminal (CTRL+ALT+T) y navega hasta el directorio donde se guarda el archivo. 13

4.2.- Abre el editor de texto nano y escribe el siguiente ejercicio en un archivo llamado **`década_edad.sh`**. 13

4.3.- Guarda el archivo con el nombre "**`decada_edad.sh`**". 13

4.4.- Escribe los siguientes comandos:..... 14

4.5.- Explicación del código línea a línea : 14

5.- Haz un script llamado `diadelmes.sh`, que nos diga cuántos días tiene el mes actual en el momento de su ejecución (se considera que febrero tiene 28 días). 16

5.1.- Abre la terminal (CTRL+ALT+T) y navega hasta el directorio donde se guarda el archivo. 16

5.2.- Abre el editor de texto nano y escribe el siguiente ejercicio en un archivo llamado **`diadelmes.sh`**. 16

5.3.- Guarda el archivo con el nombre "**`diadelmes.sh`**". 16

5.4.- Escribe los siguientes comandos:..... 17

5.5.- Explicación del código línea a línea : 17

6.- Haz un script llamado `horoscopochino.sh`, que pida el año en que nacimos (4 cifras) y nos diga por pantalla qué animal nos corresponden según el horóscopo chino. 19

6.1.- Abre la terminal (CTRL+ALT+T) y navega hasta el directorio donde se guarda el archivo. 19

6.2.- Abre el editor de texto nano y escribe el siguiente ejercicio en un archivo llamado **`horoscopochino.sh`**. 20

6.3.- Guarda el archivo con el nombre "**`horoscopochino.sh`**". 20

6.4.- Escribe los siguientes comandos:..... 20

6.5.- Explicación línea a línea el código : 21

7.- Haz un script llamado sumarango.sh, que pida dos números por teclado y calcule la suma de los números que conforman ese rango. El resultado tiene que estar mostrado de dos formas, obtenido mediante el comando seq, y mediante la nomenclatura usada en lenguaje C..... 22

7.1.- Abre la terminal (CTRL+ALT+T) y navega hasta el directorio donde se guarda el archivo. 22

7.2.- Abre el editor de texto nano y escribe el siguiente ejercicio en un archivo llamado **sumarango.sh**. 22

7.3.- Guarda el archivo con el nombre "**sumarango.sh**". 23

7.4.- Escribe los siguientes comandos:..... 23

7.5.- Explicación línea a línea el código : 23

8.- Implementar un script llamado calculadora.sh que muestre el siguiente menú: 25

8.1.- Abre la terminal (CTRL+ALT+T) y navega hasta el directorio donde se guarda el archivo. 25

8.2.- Abre el editor de texto nano y escribe el siguiente ejercicio en un archivo llamado **calculadora.sh**. 26

8.3.- Guarda el archivo con el nombre "**calculadora.sh**". 27

8.4.- Escribe los siguientes comandos:..... 27

8.5.- Explicación línea a línea el código : 29

9.- Realizar un script que se llame usuarios.sh muestre la lista de información de un usuario del sistema pasado por parámetro (nombre de usuario, UID, GID y directorio de trabajo)..... 32

9.1.- Abre la terminal (CTRL+ALT+T) y navega hasta el directorio donde se guarda el archivo. 32

9.2.- Abre el editor de texto nano y escribe el siguiente ejercicio en un archivo llamado **usuarios.sh**. 32

9.3.- Guarda el archivo con el nombre "**usuarios.sh**". 32

9.4.- Escribe los siguientes comandos:..... 33

9.5.- Explicación línea a línea el código : 33

10.- Hacer un script llamado copia_scripts.sh, que haga un copiado de los scripts creados en la carpeta pasada como parámetro, deben ser empaquetados con el comando tar y el nombre del archivo debe tener el siguiente formato: "copia_scripts_ddmmaaaa.tar" siendo dd el día, mm el mes y aaaa el año en el que se produce la copia.35

10.1.- Abre la terminal (CTRL+ALT+T) y navega hasta el directorio donde se guarda el archivo..... 35

10.2.- Abre el editor de texto nano y escribe el siguiente ejercicio en un archivo llamado **copia_scripts.sh**. 35

10.3.- Guarda el archivo con el nombre "**copia_scripts.sh**". 35

10.4.- Escribe los siguientes comandos: 36

10.5.- Explicación línea a línea el código : 36

11.- OPCIONAL. Realiza un script que suba a tu cuenta de GitHub todos los ejercicios de la actividad pidiendo por parámetro el nombre de usuario correspondiente de la plataforma.....38

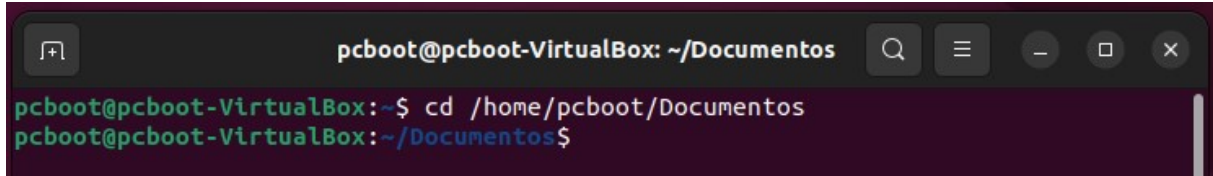
11.1.- Primero crear un token..... 38

11.2.- Explicación línea a línea para subir repositorio al GitHub:..... 41

11.3.- Repositorio GitHub 44

1.- Haz un script llamado multiplo10.sh que pida un número por teclado e indique si es un múltiplo de 10 o no.

1.1.- Abre la terminal (CTRL+ALT+T) y navega hasta el directorio donde se guarda el archivo.



```
pcboot@pcboot-VirtualBox: ~/Documentos
pcboot@pcboot-VirtualBox:~$ cd /home/pcboot/Documentos
pcboot@pcboot-VirtualBox:~/Documentos$
```

1.2.- Abre el editor de texto nano y escribe el siguiente ejercicio en un archivo llamado multiplo10.sh.



```
GNU nano 6.2 multiplo10.sh
#!/bin/bash

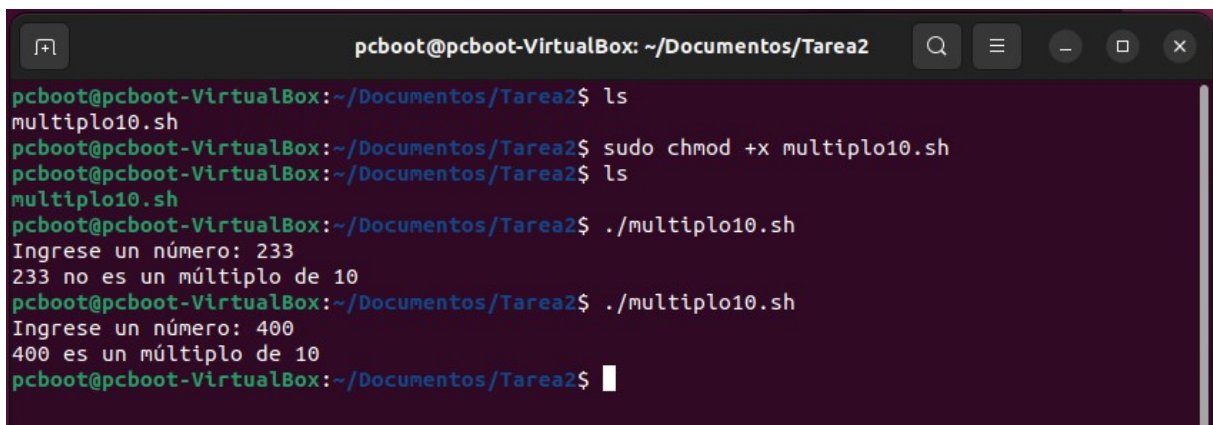
# Solicitar al usuario que introduzca un número
read -p "Ingrese un número: " numero

# usamos la instrucción if para verificar si el número es multiplo de 10
if [ $($numero % 10) -eq 0 ]; then
    # Si el número es un múltiplo de 10, imprimimos un mensaje
    echo "$numero es un múltiplo de 10"
else
    # Si el número no es un múltiplo de 10, imprimimos un mensaje diferente
    echo "$numero no es un múltiplo de 10"
fi
```

1.3.- Guarda el archivo con el nombre "multiplo10.sh".

Guarda el archivo CTRL+O y salir del editor CTRL+X.

1.4.- Escribe los siguientes comandos:



```
pcboot@pcboot-VirtualBox: ~/Documentos/Tarea2
pcboot@pcboot-VirtualBox:~/Documentos/Tarea2$ ls
multiplo10.sh
pcboot@pcboot-VirtualBox:~/Documentos/Tarea2$ sudo chmod +x multiplo10.sh
pcboot@pcboot-VirtualBox:~/Documentos/Tarea2$ ls
multiplo10.sh
pcboot@pcboot-VirtualBox:~/Documentos/Tarea2$ ./multiplo10.sh
Ingrese un número: 233
233 no es un múltiplo de 10
pcboot@pcboot-VirtualBox:~/Documentos/Tarea2$ ./multiplo10.sh
Ingrese un número: 400
400 es un múltiplo de 10
pcboot@pcboot-VirtualBox:~/Documentos/Tarea2$
```

1.5.- Explicación del código línea a línea :

```
#!/bin/bash
```

Esta línea indica que el script debe ser interpretado por el intérprete de comandos bash.

```
# Solicitar al usuario que introduzca un número  
read -p "Ingrese un número: " numero
```

Esta línea muestra el mensaje "Ingrese un número: " y espera a que el usuario ingrese un número, que se almacena en la variable "numero".

```
# Usamos la instrucción if para verificar si el número es múltiplo de 10  
if [ $((numero % 10)) -eq 0 ]; then
```

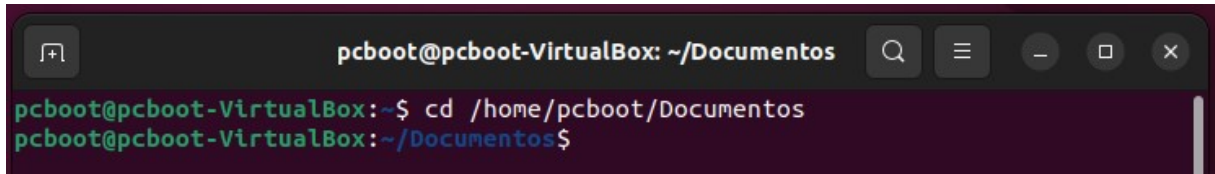
Aquí se inicia una estructura condicional "if". Se verifica si el residuo de la división del número ingresado por 10 es igual a 0, lo que indica que el número es múltiplo de 10.

```
    # Si el número es un múltiplo de 10, imprimimos un mensaje  
    echo "$numero es un múltiplo de 10"  
else  
    # Si el número no es un múltiplo de 10, imprimimos un mensaje diferente  
    echo "$numero no es un múltiplo de 10"  
fi
```

Si el número es múltiplo de 10, se imprime el mensaje "\$numero es un múltiplo de 10". De lo contrario, se imprime el mensaje "\$numero no es un múltiplo de 10". Esto es un script simple que ilustra el uso de la entrada del usuario, la estructura condicional "if" y la aritmética básica en bash.

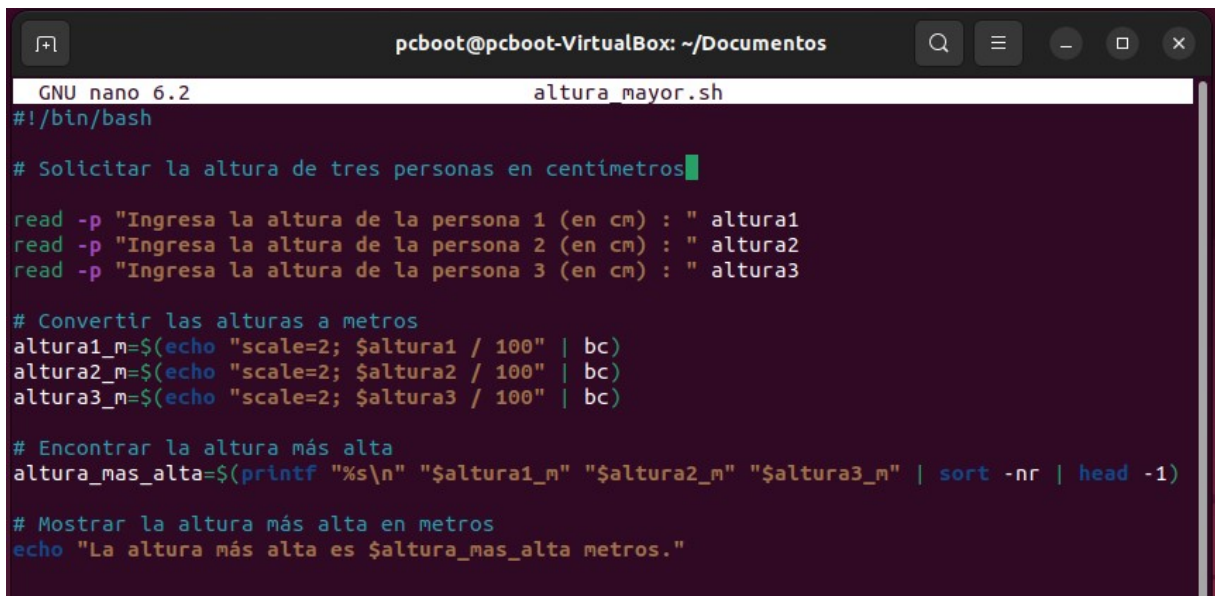
2.- Haz un script llamado `altura_mayor.sh`, que pida por teclado la altura en centímetros de tres personas y nos diga la mayor de las 3 en metros.

2.1.- Abre la terminal (CTRL+ALT+T) y navega hasta el directorio donde se guarda el archivo.



```
pcboot@pcboot-VirtualBox: ~/Documentos
pcboot@pcboot-VirtualBox:~$ cd /home/pcboot/Documentos
pcboot@pcboot-VirtualBox:~/Documentos$
```

2.2.- Abre el editor de texto nano y escribe el siguiente ejercicio en un archivo llamado `altura_mayor.sh`.



```
GNU nano 6.2          altura_mayor.sh
#!/bin/bash

# Solicitar la altura de tres personas en centímetros

read -p "Ingresa la altura de la persona 1 (en cm) : " altura1
read -p "Ingresa la altura de la persona 2 (en cm) : " altura2
read -p "Ingresa la altura de la persona 3 (en cm) : " altura3

# Convertir las alturas a metros
altura1_m=$(echo "scale=2; $altura1 / 100" | bc)
altura2_m=$(echo "scale=2; $altura2 / 100" | bc)
altura3_m=$(echo "scale=2; $altura3 / 100" | bc)

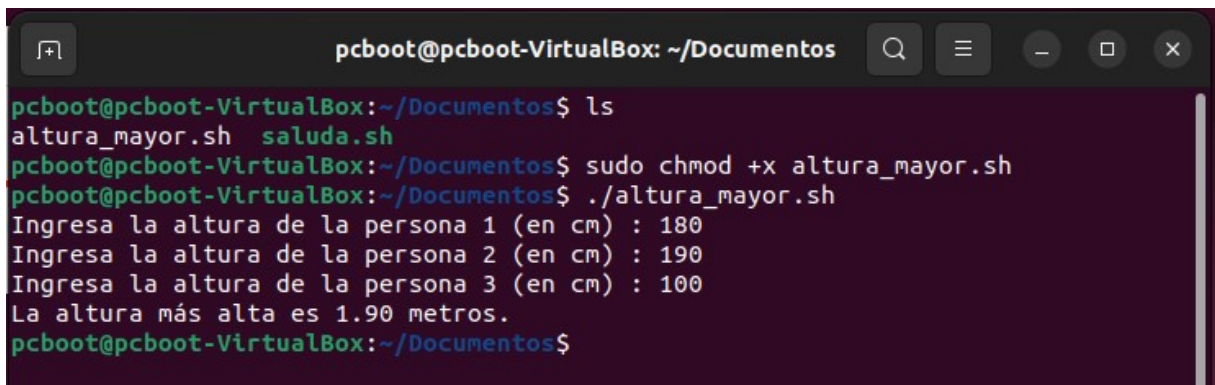
# Encontrar la altura más alta
altura_mas_alta=$(printf "%s\n" "$altura1_m" "$altura2_m" "$altura3_m" | sort -nr | head -1)

# Mostrar la altura más alta en metros
echo "La altura más alta es $altura_mas_alta metros."
```

2.3.- Guarda el archivo con el nombre "`altura_mayor.sh`".

Guarda el archivo CTRL+O y salir del editor CTRL+X.

2.4.- Escribe los siguientes comandos:



```
pcboot@pcboot-VirtualBox: ~/Documentos
pcboot@pcboot-VirtualBox:~/Documentos$ ls
altura_mayor.sh  saluda.sh
pcboot@pcboot-VirtualBox:~/Documentos$ sudo chmod +x altura_mayor.sh
pcboot@pcboot-VirtualBox:~/Documentos$ ./altura_mayor.sh
Ingresa la altura de la persona 1 (en cm) : 180
Ingresa la altura de la persona 2 (en cm) : 190
Ingresa la altura de la persona 3 (en cm) : 100
La altura más alta es 1.90 metros.
pcboot@pcboot-VirtualBox:~/Documentos$
```


2.5.- Explicación del código línea a línea :

```
#!/bin/bash
```

Esta línea indica que el script debe ser interpretado por el intérprete de comandos bash.

```
# Solicitar la altura de tres personas en centímetros
read -p "Ingresa la altura de la persona 1 (en cm): " altura1
read -p "Ingresa la altura de la persona 2 (en cm): " altura2
read -p "Ingresa la altura de la persona 3 (en cm): " altura3
```

Estas líneas solicitan al usuario que ingrese la altura de tres personas en centímetros y almacenan estos valores en las variables "altura1", "altura2" y "altura3".

```
# Convertir las alturas a metros
altura1_m=$(echo "scale=2; $altura1 / 100" | bc)
altura2_m=$(echo "scale=2; $altura2 / 100" | bc)
altura3_m=$(echo "scale=2; $altura3 / 100" | bc)
```

Estas líneas convierten las alturas ingresadas de centímetros a metros utilizando el comando "bc" para realizar cálculos aritméticos.

```
# Encontrar la altura más alta
altura_mas_alta=$(printf "%s\n" "$altura1_m" "$altura2_m" "$altura3_m" | sort -nr | head -1)
```

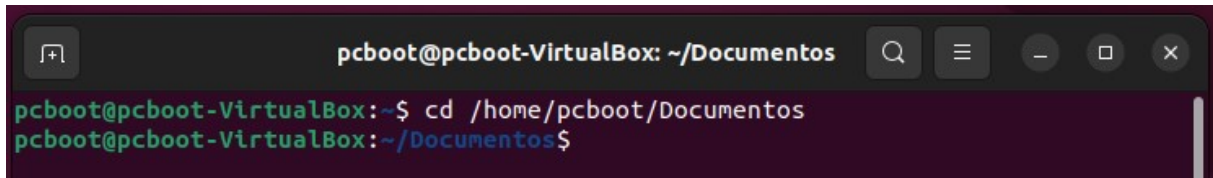
Esta línea encuentra la altura más alta entre las tres alturas convertidas a metros utilizando el comando "sort" para ordenar los valores de forma descendente y "head -1" para obtener el valor más alto.

```
# Mostrar la altura más alta en metros
echo "La altura más alta es $altura_mas_alta metros."
```

Finalmente, esta línea imprime el mensaje que indica la altura más alta encontrada en metros. En resumen, este script solicita las alturas de tres personas en centímetros, las convierte a metros, encuentra la altura más alta y luego muestra este valor en metros.

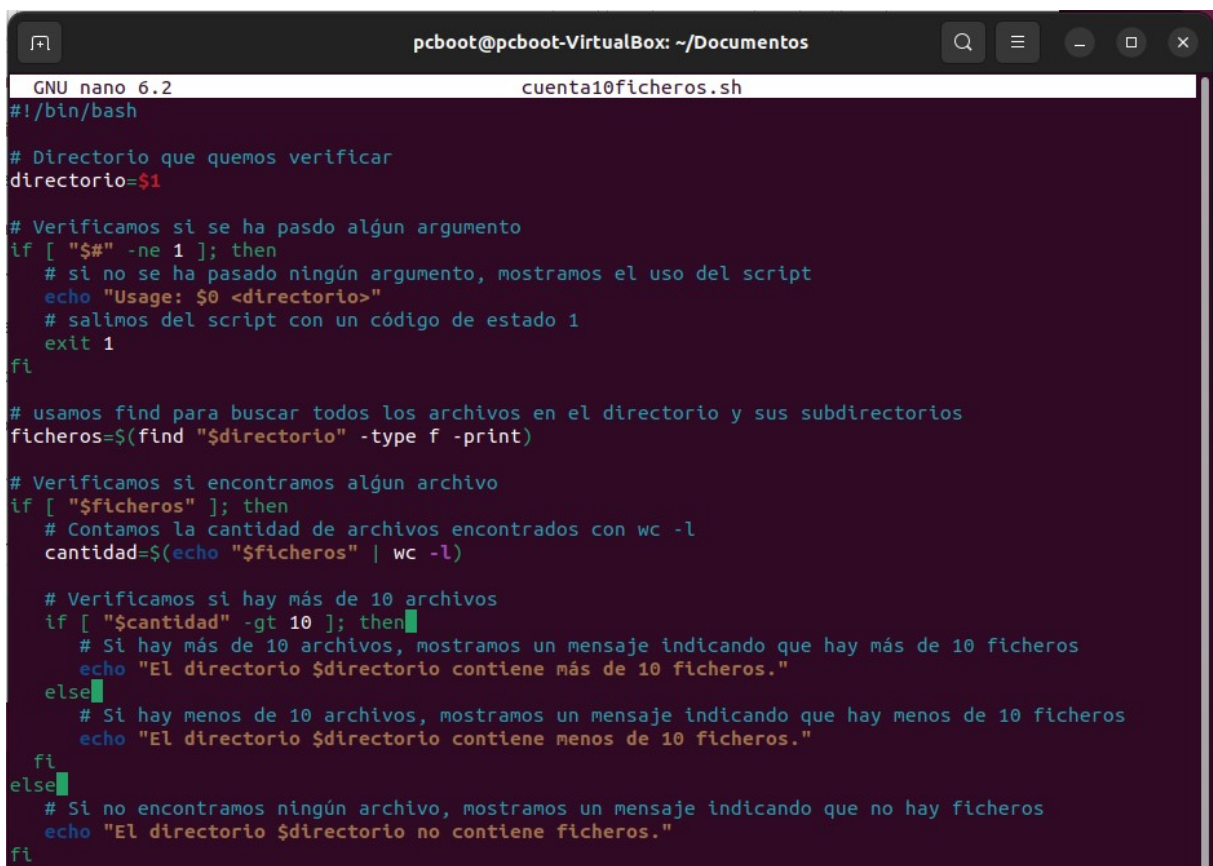
3.- Haz un script llamado cuenta10ficheros.sh, que nos diga si en el directorio pasado como parámetro hay más de 10 ficheros o no, es decir, los directorios no deben ser contados.

3.1.- Abre la terminal (TRL+ALT+T) y navega hasta el directorio donde se guarda el archivo.



```
pcboot@pcboot-VirtualBox: ~/Documentos
pcboot@pcboot-VirtualBox:~$ cd /home/pcboot/Documentos
pcboot@pcboot-VirtualBox:~/Documentos$
```

3.2.- Abre el editor de texto nano y escribe el siguiente ejercicio en un archivo llamado cuenta10ficheros.sh.



```
GNU nano 6.2 cuenta10ficheros.sh
#!/bin/bash

# Directorio que quemos verificar
directorio=$1

# Verificamos si se ha pasado algún argumento
if [ "$#" -ne 1 ]; then
    # si no se ha pasado ningún argumento, mostramos el uso del script
    echo "Usage: $0 <directorio>"
    # salimos del script con un código de estado 1
    exit 1
fi

# usamos find para buscar todos los archivos en el directorio y sus subdirectorios
ficheros=$(find "$directorio" -type f -print)

# Verificamos si encontramos algún archivo
if [ "$ficheros" ]; then
    # Contamos la cantidad de archivos encontrados con wc -l
    cantidad=$(echo "$ficheros" | wc -l)

    # Verificamos si hay más de 10 archivos
    if [ "$cantidad" -gt 10 ]; then
        # Si hay más de 10 archivos, mostramos un mensaje indicando que hay más de 10 ficheros
        echo "El directorio $directorio contiene más de 10 ficheros."
    else
        # Si hay menos de 10 archivos, mostramos un mensaje indicando que hay menos de 10 ficheros
        echo "El directorio $directorio contiene menos de 10 ficheros."
    fi
else
    # Si no encontramos ningún archivo, mostramos un mensaje indicando que no hay ficheros
    echo "El directorio $directorio no contiene ficheros."
fi
```

3.3.- Guarda el archivo con el nombre "cuenta10ficheros.sh".

Guarda el archivo CTRL+O y salir del editor CTRL+X.

3.4.- Escribe los siguientes comandos:

```
pcboot@pcboot-VirtualBox: ~/Documentos
pcboot@pcboot-VirtualBox:~/Documentos$ ls
almacen altura_mayor.sh cuenta10ficheros.sh saluda.sh
pcboot@pcboot-VirtualBox:~/Documentos$ sudo chmod +x cuenta10ficheros.sh
pcboot@pcboot-VirtualBox:~/Documentos$ ls
almacen altura_mayor.sh cuenta10ficheros.sh saluda.sh
pcboot@pcboot-VirtualBox:~/Documentos$ ./cuenta10ficheros.sh almacen
El directorio almacen contiene menos de 10 ficheros.
pcboot@pcboot-VirtualBox:~/Documentos$ ls
almacen altura_mayor.sh cuenta10ficheros.sh saluda.sh
pcboot@pcboot-VirtualBox:~/Documentos$ ls almacen
asdfsdf 'saluda (4.ª copia).sh' 'saluda (7.ª copia).sh' saluda.sh
'saluda (10.ª copia).sh' 'saluda (5.ª copia).sh' 'saluda (8.ª copia).sh' sfa
'saluda (3.ª copia).sh' 'saluda (6.ª copia).sh' 'saluda (9.ª copia).sh'
pcboot@pcboot-VirtualBox:~/Documentos$ ./cuenta10ficheros.sh almacen
El directorio almacen contiene más de 10 ficheros.
pcboot@pcboot-VirtualBox:~/Documentos$ ls almacen
asdfsdf 'saluda (13.ª copia).sh' 'saluda (6.ª copia).sh' saluda.sh
'saluda (10.ª copia).sh' 'saluda (3.ª copia).sh' 'saluda (7.ª copia).sh' sfa
'saluda (11.ª copia).sh' 'saluda (4.ª copia).sh' 'saluda (8.ª copia).sh'
'saluda (12.ª copia).sh' 'saluda (5.ª copia).sh' 'saluda (9.ª copia).sh'
pcboot@pcboot-VirtualBox:~/Documentos$
```

3.5.- Explicación del código línea a línea :

```
#!/bin/bash
```

Esta línea indica que el script debe ser interpretado por el intérprete de comandos bash.

```
# Directorio que queremos verificar
directorio=$1
```

Esta línea asigna el primer argumento pasado al script a la variable "directorio", que representa el directorio que se desea verificar.

```
# Verificamos si se ha pasado algún argumento
if [ "$#" -ne 1 ]; then
    # Si no se ha pasado ningún argumento, mostramos el uso del script
    echo "Usage: $0 <directorio>"
    # Salimos del script con un código de estado 1
    exit 1
fi
```

Estas líneas verifican si se ha pasado un argumento al script. Si no se ha pasado ningún argumento, muestra un mensaje de uso y sale del script con un código de estado 1.

```
# Usamos find para buscar todos los archivos en el directorio y sus subdirectorios
ficheros=$(find "$directorio" -type f -print)
```

Esta línea utiliza el comando "find" para buscar todos los archivos en el directorio y sus subdirectorios, y almacena los resultados en la variable "ficheros".

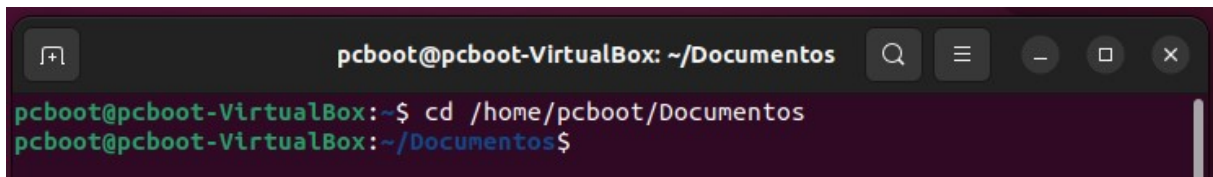
```
# Verificamos si encontramos algún archivo
if [ "$ficheros" ]; then
    # Contamos la cantidad de archivos encontrados con wc -l
    cantidad=$(echo "$ficheros" | wc -l)

    # Verificamos si hay más de 10 archivos
    if [ "$cantidad" -gt 10 ]; then
        # Si hay más de 10 archivos, mostramos un mensaje
        # indicando que hay más de 10 ficheros
        echo "El directorio $directorio contiene más de 10 ficheros."
    else
        # Si hay menos de 10 archivos, mostramos un mensaje indicando
        # que hay menos de 10 ficheros
        echo "El directorio $directorio contiene menos de 10 ficheros."
    fi
else
    # Si no encontramos ningún archivo, mostramos un mensaje
    # indicando que no hay ficheros
    echo "El directorio $directorio no contiene ficheros."
fi
```

Estas líneas verifican si se encontraron archivos en el directorio. Si se encontraron archivos, cuenta la cantidad de archivos y muestra un mensaje dependiendo de la cantidad encontrada. Si no se encontraron archivos, muestra un mensaje indicando que no hay archivos en el directorio.

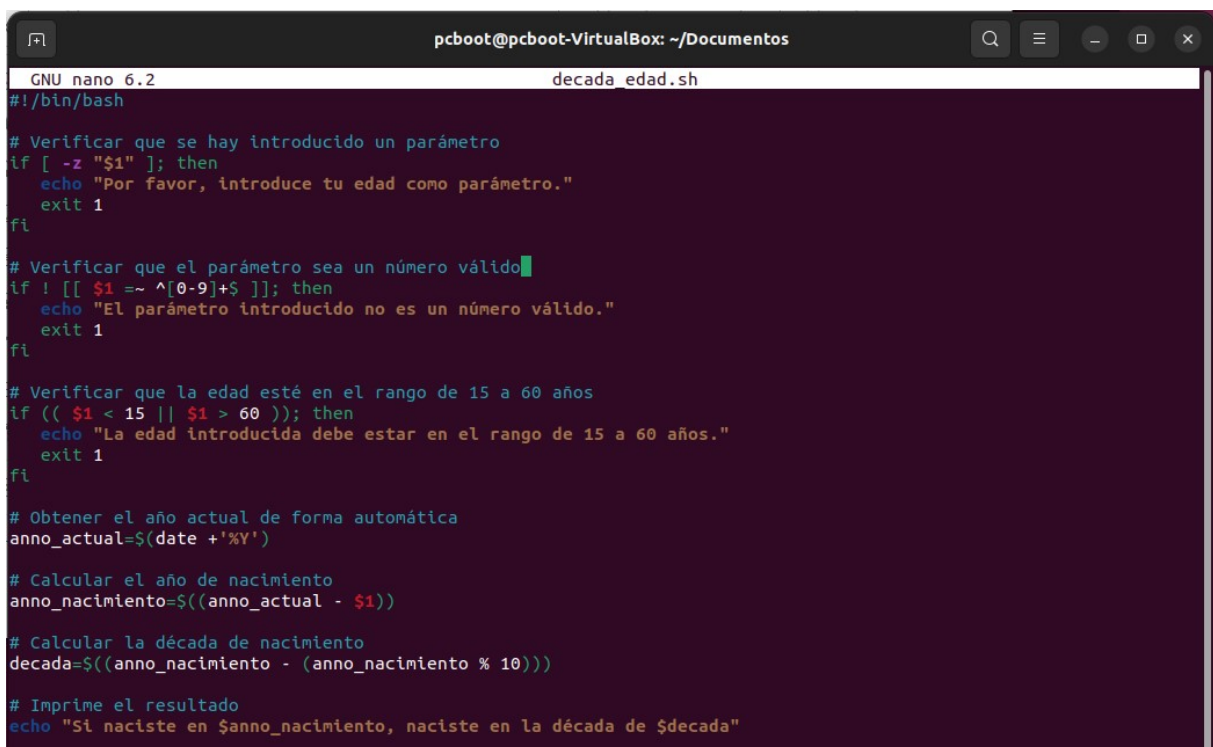
4.- Haz un script llamado `decada_edad.sh`, que dada una edad que introducimos por pantalla, nos diga como resultado la década en la que nacimos (ejemplo 70, 80, 90 ...). Suponemos que la edad introducida está entre 15 y 60 años. El script debe coger el año actual de forma automática. La salida debe producirse en el siguiente formato: "Si naciste en 1983, naciste en la década de 1980".

4.1.- Abre la terminal (CTRL+ALT+T) y navega hasta el directorio donde se guarda el archivo.



```
pcboot@pcboot-VirtualBox: ~/.Documents
pcboot@pcboot-VirtualBox:~$ cd /home/pcboot/Documents
pcboot@pcboot-VirtualBox:~/.Documents$
```

4.2.- Abre el editor de texto nano y escribe el siguiente ejercicio en un archivo llamado `decada_edad.sh`.



```
GNU nano 6.2          decada_edad.sh
#!/bin/bash

# Verificar que se hay introducido un parámetro
if [ -z "$1" ]; then
    echo "Por favor, introduce tu edad como parámetro."
    exit 1
fi

# Verificar que el parámetro sea un número válido
if ! [[ $1 =~ ^[0-9]+$ ]]; then
    echo "El parámetro introducido no es un número válido."
    exit 1
fi

# Verificar que la edad esté en el rango de 15 a 60 años
if (( $1 < 15 || $1 > 60 )); then
    echo "La edad introducida debe estar en el rango de 15 a 60 años."
    exit 1
fi

# Obtener el año actual de forma automática
anno_actual=$(date +%Y)

# Calcular el año de nacimiento
anno_nacimiento=$((anno_actual - $1))

# Calcular la década de nacimiento
decada=$((anno_nacimiento - (anno_nacimiento % 10)))

# Imprime el resultado
echo "Si naciste en $anno_nacimiento, naciste en la década de $decada"
```

4.3.- Guarda el archivo con el nombre "`decada_edad.sh`".

Guarda el archivo CTRL+O y salir del editor CTRL+X.

4.4.- Escribe los siguientes comandos:

```
pcboot@pcboot-VirtualBox: ~/Documentos
pcboot@pcboot-VirtualBox:~/Documentos$ ls
almacen altura_mayor.sh cuenta10ficheros.sh decada_edad.sh saluda.sh
pcboot@pcboot-VirtualBox:~/Documentos$ sudo chmod +x decada_edad.sh
pcboot@pcboot-VirtualBox:~/Documentos$ ./decada_edad.sh 51
Si naciste en 1972, naciste en la década de 1970
pcboot@pcboot-VirtualBox:~/Documentos$ ./decada_edad.sh
Por favor, introduce tu edad como parámetro.
pcboot@pcboot-VirtualBox:~/Documentos$ ./decada_edad.sh 80
La edad introducida debe estar en el rango de 15 a 60 años.
pcboot@pcboot-VirtualBox:~/Documentos$
```

4.5.- Explicación del código línea a línea :

```
#!/bin/bash
```

Esta línea indica que el script debe ser interpretado por el intérprete de comandos bash.

```
# Verificar que se haya introducido un parámetro
if [ -z "$1" ]; then
    echo "Por favor, introduce tu edad como parámetro."
    exit 1
fi
```

Estas líneas verifican si se ha introducido un parámetro al script. Si no se ha introducido, muestra un mensaje y sale del script con un código de estado 1.

```
# Verificar que el parámetro sea un número válido
if ! [[ $1 =~ ^[0-9]+$ ]]; then
    echo "El parámetro introducido no es un número válido."
    exit 1
fi
```

Estas líneas verifican si el parámetro introducido es un número válido. Si no es un número válido, muestra un mensaje y sale del script con un código de estado 1.

```
# Verificar que la edad esté en el rango de 15 a 60 años
if (( $1 < 15 || $1 > 60 )); then
    echo "La edad introducida debe estar en el rango de 15 a 60 años."
    exit 1
fi
```

Estas líneas verifican si la edad está en el rango de 15 a 60 años. Si no está en ese rango, muestra un mensaje y sale del script con un código de estado 1.

```
# Obtener el año actual de forma automática
anno_actual=$(date +%Y)

# Calcular el año de nacimiento
anno_nacimiento=$((anno_actual - $1))

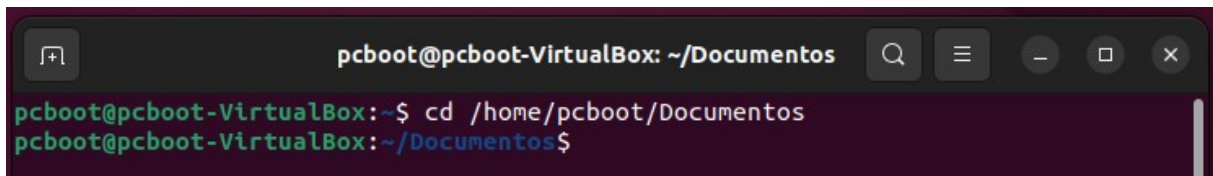
# Calcular la década de nacimiento
decade=$((anno_nacimiento - (anno_nacimiento % 10)))

# Imprimir el resultado
echo "Si naciste en $anno_nacimiento, naciste en la década de $decade"
```

Estas líneas obtienen el año actual, calculan el año de nacimiento restando la edad al año actual, calculan la década de nacimiento y finalmente imprimen el resultado.

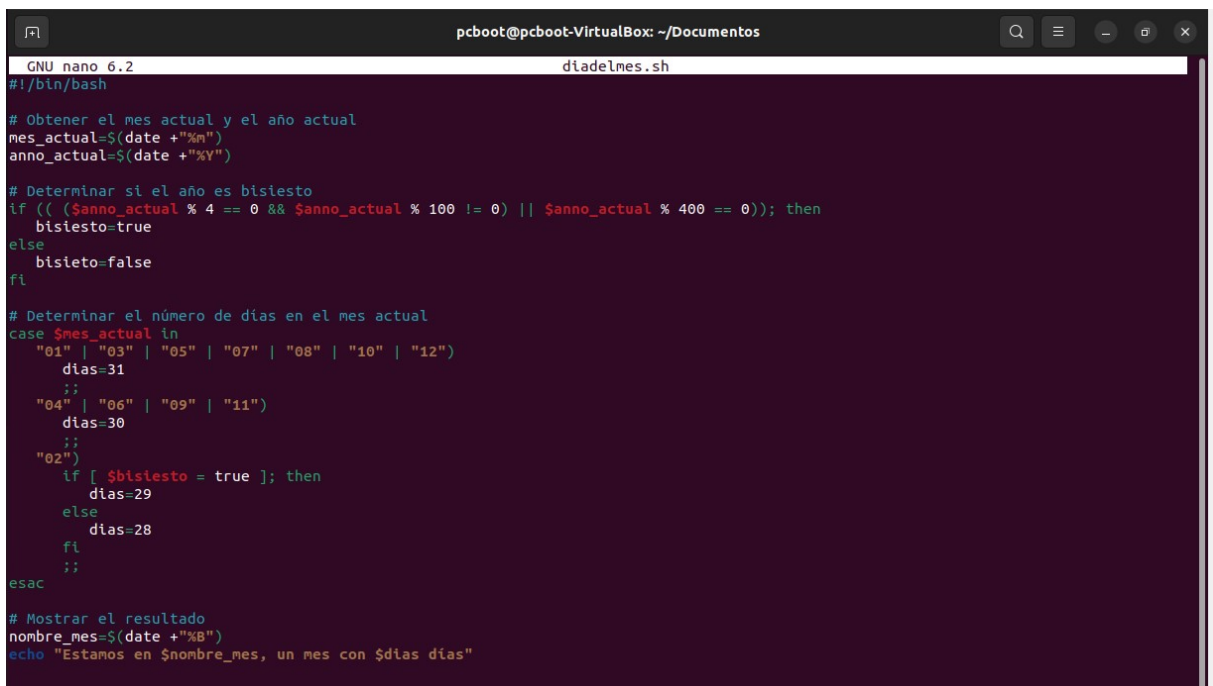
5.- Haz un script llamado `diadelmes.sh`, que nos diga cuántos días tiene el mes actual en el momento de su ejecución (se considera que febrero tiene 28 días). La salida debe producirse en el siguiente formato: "Estamos en noviembre, un mes con 30 días".

5.1.- Abre la terminal (CTRL+ALT+T) y navega hasta el directorio donde se guarda el archivo.



```
pcboot@pcboot-VirtualBox: ~/.Documents
pcboot@pcboot-VirtualBox:~$ cd /home/pcboot/Documentos
pcboot@pcboot-VirtualBox:~/.Documents$
```

5.2.- Abre el editor de texto nano y escribe el siguiente ejercicio en un archivo llamado `diadelmes.sh`.



```
GNU nano 6.2 diadelmes.sh
#!/bin/bash

# Obtener el mes actual y el año actual
mes_actual=$(date +%m)
año_actual=$(date +%Y)

# Determinar si el año es bisiesto
if (( ($año_actual % 4 == 0 && $año_actual % 100 != 0) || $año_actual % 400 == 0 )); then
    bisiesto=true
else
    bisiesto=false
fi

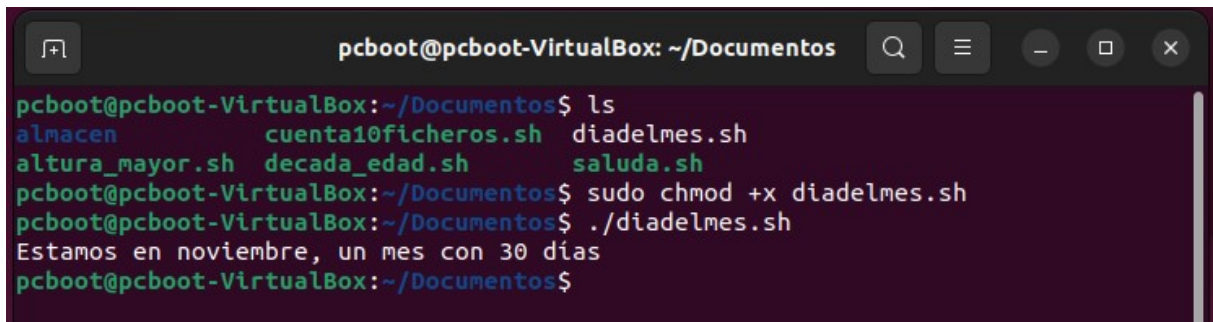
# Determinar el número de días en el mes actual
case $mes_actual in
    "01" | "03" | "05" | "07" | "08" | "10" | "12")
        dias=31
        ;;
    "04" | "06" | "09" | "11")
        dias=30
        ;;
    "02")
        if [ $bisiesto = true ]; then
            dias=29
        else
            dias=28
        fi
        ;;
esac

# Mostrar el resultado
nombre_mes=$(date +%B)
echo "Estamos en $nombre_mes, un mes con $dias días"
```

5.3.- Guarda el archivo con el nombre "`diadelmes.sh`".

Guarda el archivo CTRL+O y salir del editor CTRL+X.

5.4.- Escribe los siguientes comandos:



```
pcboot@pcboot-VirtualBox: ~/Documentos
pcboot@pcboot-VirtualBox:~/Documentos$ ls
almacen          cuenta10ficheros.sh  diadelmes.sh
altura_mayor.sh  decada_edad.sh       saluda.sh
pcboot@pcboot-VirtualBox:~/Documentos$ sudo chmod +x diadelmes.sh
pcboot@pcboot-VirtualBox:~/Documentos$ ./diadelmes.sh
Estamos en noviembre, un mes con 30 días
pcboot@pcboot-VirtualBox:~/Documentos$
```

5.5.- Explicación del código línea a línea :

```
#!/bin/bash
```

Esta línea indica que el script debe ser interpretado por el intérprete de comandos bash.

```
# Obtener el mes y el año actual
mes_actual=$(date +%m)
año_actual=$(date +%Y)
```

Estas líneas utilizan el comando date para obtener el mes y el año actuales y los almacenan en las variables mes_actual y año_actual, respectivamente.

```
# Determinar si el año es bisiesto
if (( ($año_actual % 4 == 0 && $año_actual % 100 != 0) || $año_actual % 400 == 0 )); then
    bisiesto=true
else
    bisiesto=false
fi
```

En este bloque, se determina si el año actual es bisiesto. La expresión condicional verifica si el año es divisible por 4 pero no por 100, o si es divisible por 400. Dependiendo del resultado, se establece la variable bisiesto en true o false.

```
# Determinar el número de días en el mes actual, considerando si el año es bisiesto
case $mes_actual in
    "01" | "03" | "05" | "07" | "08" | "10" | "12")
        dias=31
        ;;
    "04" | "06" | "09" | "11")
        dias=30
        ;;
    "02")
        if [ $bisiesto = true ]; then
            dias=29
        else
            dias=28
        fi
        ;;
esac
```

En este bloque, se utiliza una declaración case para determinar el número de días en el mes actual, considerando si el año es bisiesto. Dependiendo del mes y si el año es bisiesto, se asigna el número de días a la variable días.

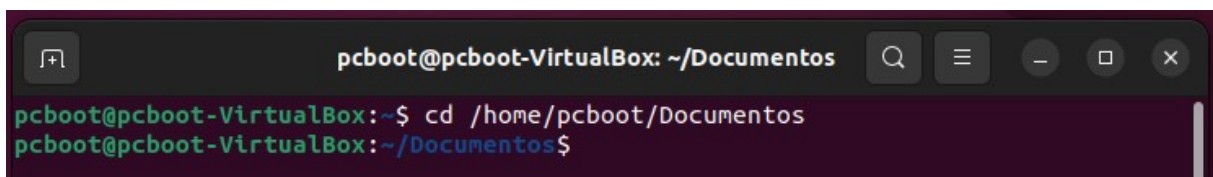
```
# Mostrar el resultado
nombre_mes=$(date +%B)
echo "Estamos en $nombre_mes, un mes con $dias días"
```

Finalmente, se muestra el resultado con el nombre del mes y el número de días. Este script permite determinar dinámicamente el número de días en el mes actual, considerando si el año es bisiesto, y mostrar el resultado correspondiente. Las referencias proporcionadas contienen información adicional sobre el cálculo de años bisiestos y el número de días en febrero, así como detalles sobre el 29 de febrero y su relevancia en el calendario gregoriano.

6.- Haz un script llamado horoscopochino.sh, que pida el año en que nacimos (4 cifras) y nos diga por pantalla qué animal nos corresponden según el horóscopo chino. Para calcularlo debemos dividir el año entre 12 y el resto nos indicará el animal según la siguiente tabla:

8	El mono	0	La rata	4	El dragón
9	El gallo	1	El buey	5	La serpiente
10	El perro	2	El tigre	6	El caballo
11	El cerdo	3	El conejo	7	La cabra

6.1.- Abre la terminal (CTRL+ALT+T) y navega hasta el directorio donde se guarda el archivo.



```
pcboot@pcboot-VirtualBox: ~/Documentos
pcboot@pcboot-VirtualBox:~$ cd /home/pcboot/Documentos
pcboot@pcboot-VirtualBox:~/Documentos$
```

6.2.- Abre el editor de texto nano y escribe el siguiente ejercicio en un archivo llamado horoscopochino.sh.



```
GNU nano 6.2 horoscopochino.sh
#!/bin/bash

# Solicitar el año de nacimiento al usuario y validar la entrada
while true; do
    echo "Por favor, ingresa el año de tu nacimiento (4 cifras):"
    read anno
    if [[ $anno =~ ^[0-9]{4}$ ]]; then
        break
    else
        echo "El año ingresado no es válido. Por favor, ingresa un año de 4 cifras."
    fi
done

# Calcular el animal correspondiente al año de nacimiento
animal=""
case $((anno % 12)) in
    0) animal="El mono" ;;
    1) animal="La rata" ;;
    2) animal="El buey" ;;
    3) animal="El tigre" ;;
    4) animal="El conejo" ;;
    5) animal="El dragón" ;;
    6) animal="La serpiente" ;;
    7) animal="El caballo" ;;
    8) animal="La cabra" ;;
    9) animal="El mono" ;;
    10) animal="El gallo" ;;
    11) animal="El perro" ;;
esac

# Mostrar el resultado
echo "Según el horóscopo chino, naciste en el año del $animal"
```

6.3.- Guarda el archivo con el nombre "horoscopochino.sh".

Guarda el archivo CTRL+O y salir del editor CTRL+X.

6.4.- Escribe los siguientes comandos:



```
pcboot@pcboot-VirtualBox: ~/Documentos
pcboot@pcboot-VirtualBox:~/Documentos$ ls
almacen          cuenta10ficheros.sh  diadelmes.sh        saluda.sh
altura_mayor.sh  decada_edad.sh       horoscopochino.sh
pcboot@pcboot-VirtualBox:~/Documentos$ sudo chmod +x horoscopochino.sh
pcboot@pcboot-VirtualBox:~/Documentos$ ls
almacen          cuenta10ficheros.sh  diadelmes.sh        saluda.sh
altura_mayor.sh  decada_edad.sh       horoscopochino.sh
pcboot@pcboot-VirtualBox:~/Documentos$ ./horoscopochino.sh
Por favor, ingresa el año de tu nacimiento (4 cifras):
1972
Según el horóscopo chino, naciste en el año del El conejo
pcboot@pcboot-VirtualBox:~/Documentos$
```

6.5.- Explicación línea a línea el código :

```
#!/bin/bash
```

Esta línea indica que el script debe ser interpretado por el intérprete de comandos bash.

```
# Solicitar el año de nacimiento al usuario y validar la entrada
while true; do
    echo "Por favor, ingresa el año de tu nacimiento (4 cifras):"
    read año
    if [[ $año =~ ^[0-9]{4}$ ]]; then
        break
    else
        echo "El año ingresado no es válido. Por favor, ingresa un año de 4 cifras."
    fi
done
```

Estas líneas utilizan un bucle "while" para solicitar al usuario que ingrese su año de nacimiento. Luego, valida la entrada para asegurarse de que el año ingresado consta de 4 cifras.

```
# Calcular el animal correspondiente al año de nacimiento
animal=""
case $((año % 12)) in
    0) animal="El mono" ;;
    1) animal="La rata" ;;
    2) animal="El buey" ;;
    3) animal="El tigre" ;;
    4) animal="El conejo" ;;
    5) animal="El dragón" ;;
    6) animal="La serpiente" ;;
    7) animal="El caballo" ;;
    8) animal="La cabra" ;;
    9) animal="El mono" ;;
    10) animal="El gallo" ;;
    11) animal="El perro" ;;
esac
```

Estas líneas utilizan una declaración "case" para calcular el animal correspondiente al año de nacimiento basado en el horóscopo chino.

```
# Mostrar el resultado
echo "Según el horóscopo chino, naciste en el año del $animal"
```

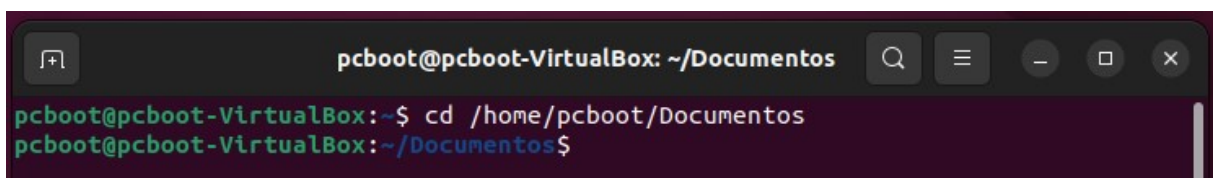
Finalmente, esta línea imprime el resultado que indica el animal correspondiente al año de nacimiento según el horóscopo chino.

7.- Haz un script llamado sumarango.sh, que pida dos números por teclado y calcule la suma de los números que conforman ese rango. El resultado tiene que estar mostrado de dos formas, obtenido mediante el comando seq, y mediante la nomenclatura usada en lenguaje C.

Por ejemplo: Si introducimos el 5 y el 8 debemos mostrar el 26 por pantalla que es el resultado de sumar 5+6+7+8.

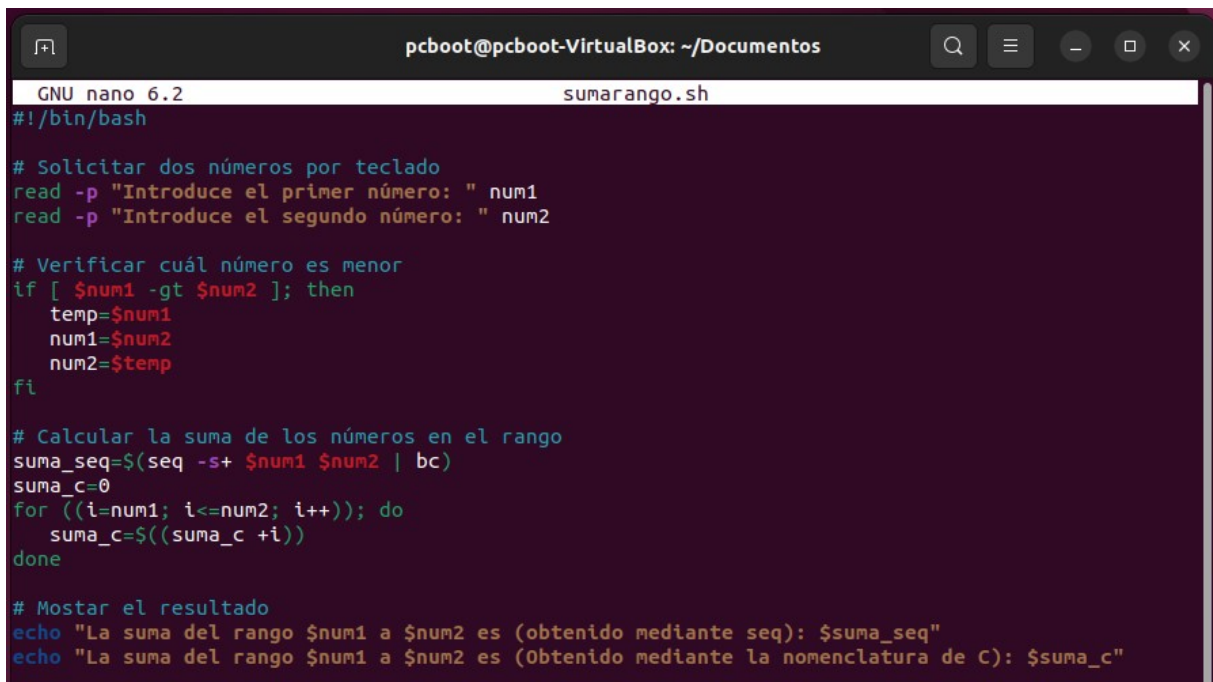
OPCIONAL: El programa comprueba qué número es menor antes de calcular la suma del rango para poder invertir el orden en caso necesario, por si introdujéramos primero el mayor evitar entrar en un bucle erróneo.

7.1.- Abre la terminal (CTRL+ALT+T) y navega hasta el directorio donde se guarda el archivo.



```
pcboot@pcboot-VirtualBox: ~/Documentos
pcboot@pcboot-VirtualBox:~$ cd /home/pcboot/Documentos
pcboot@pcboot-VirtualBox:~/Documentos$
```

7.2.- Abre el editor de texto nano y escribe el siguiente ejercicio en un archivo llamado sumarango.sh.



```
GNU nano 6.2 sumarango.sh
#!/bin/bash

# Solicitar dos números por teclado
read -p "Introduce el primer número: " num1
read -p "Introduce el segundo número: " num2

# Verificar cuál número es menor
if [ $num1 -gt $num2 ]; then
    temp=$num1
    num1=$num2
    num2=$temp
fi

# Calcular la suma de los números en el rango
suma_seq=$(seq -s+ $num1 $num2 | bc)
suma_c=0
for ((i=num1; i<=num2; i++)); do
    suma_c=$((suma_c + i))
done

# Mostar el resultado
echo "La suma del rango $num1 a $num2 es (obtenido mediante seq): $suma_seq"
echo "La suma del rango $num1 a $num2 es (Obtenido mediante la nomenclatura de C): $suma_c"
```


7.3.- Guarda el archivo con el nombre "sumarango.sh".

Guarda el archivo CTRL+O y salir del editor CTRL+X.

7.4.- Escribe los siguientes comandos:

```
pcboot@pcboot-VirtualBox: ~/Documentos
pcboot@pcboot-VirtualBox:~/Documentos$ ls
almacen          cuenta10ficheros.sh  diadelmes.sh       saluda.sh
altura_mayor.sh  decada_edad.sh       horoscopochino.sh  sumarango.sh
pcboot@pcboot-VirtualBox:~/Documentos$ sudo chmod +x sumarango.sh
pcboot@pcboot-VirtualBox:~/Documentos$ ls
almacen          cuenta10ficheros.sh  diadelmes.sh       saluda.sh
altura_mayor.sh  decada_edad.sh       horoscopochino.sh  sumarango.sh
pcboot@pcboot-VirtualBox:~/Documentos$ ./sumarango.sh
Introduce el primer número: 8
Introduce el segundo número: 5
La suma del rango 5 a 8 es (obtenido mediante seq): 26
La suma del rango 5 a 8 es (Obtenido mediante la nomenclatura de C): 26
pcboot@pcboot-VirtualBox:~/Documentos$
```

7.5.- Explicación línea a línea el código :

```
#!/bin/bash
```

Esta línea indica que el script debe ser interpretado por el intérprete de comandos bash.

```
# Solicitar dos números por teclado
read -p "Introduce el primer número: " num1
read -p "Introduce el segundo número: " num2
```

Estas líneas solicitan al usuario que introduzca dos números por teclado.

```
# Verificar cuál número es menor
if [ $num1 -gt $num2 ]; then
    temp=$num1
    num1=$num2
    num2=$temp
fi
```

Esta sección verifica cuál de los dos números es menor y los ordena en consecuencia.

```
# Calcular la suma de los números en el rango
suma_seq=$(seq -s+ $num1 $num2 | bc)
suma_c=0
for ((i=num1; i<=num2; i++)); do
    suma_c=$((suma_c + i))
done
```

Aquí se calcula la suma de los números en el rango utilizando el comando seq y la nomenclatura de C.

```
# Mostrar el resultado
echo "La suma del rango $num1 a $num2 es (obtenido mediante seq): $suma_seq"
echo "La suma del rango $num1 a $num2 es (obtenido mediante la nomenclatura de C): $suma_c"
```

Finalmente, se muestra el resultado de la suma obtenida mediante seq y la nomenclatura de C. Este script puede ser ejecutado en un terminal de bash para calcular la suma de un rango de números ingresados por el usuario.

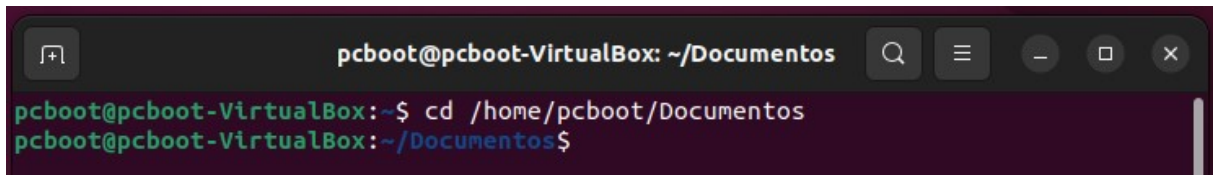
8.- Implementar un script llamado calculadora.sh que muestre el siguiente menú:

- 1) Sumar
- 2) Restar
- 3) Dividir
- 4) Multiplicar
- 5) Salir

Después de mostrar el menú, se pedirá que se elija una opción. Si la opción elegida no está entre el 1 y el 4, se mostrará un mensaje de error. En caso de que la opción sea válida, se pedirán dos números por teclado y en función de la operación elegida, se devolverá el resultado por pantalla, manejando posibles errores al introducir caracteres.

- Cada operación será implementada haciendo uso de funciones
- Si la opción elegida no es válida, se volverá a mostrar el menú
- Si el script arroja algún error debe almacenarse en un fichero con log de errores creado para la ocasión (con contenido independiente cada vez que se ejecuta el script).
- El programa terminará, cuando pulse 5.

8.1.- Abre la terminal (CTRL+ALT+T) y navega hasta el directorio donde se guarda el archivo.



```
pcboot@pcboot-VirtualBox: ~/Documentos
pcboot@pcboot-VirtualBox:~$ cd /home/pcboot/Documentos
pcboot@pcboot-VirtualBox:~/Documentos$
```

8.2.- Abre el editor de texto nano y escribe el siguiente ejercicio en un archivo llamado calculadora.sh.




```
GNU nano 6.2 calculadora.sh
#!/bin/bash

# Función para pedir un número
pedir_numero() {
    read -p "Introduce un número: " num
    echo $num
}

# Función para sumar
sumar() {
    num1=$(pedir_numero)
    num2=$(pedir_numero)
    echo "El resultado de la suma es $((num1 + num2))"
    read -n 1 -s -r -p "Presiona cualquier tecla para continuar"
}

# Función para restar
restar() {
    clear
    num1=$(pedir_numero)
    num2=$(pedir_numero)
    echo "El resultado de la resta es: $((num1 - num2))"
    read -n 1 -s -r -p "Presiona cualquier tecla para continuar"
}
```



```
GNU nano 6.2 calculadora.sh *
# Función para dividir
dividir() {
    clear
    num1=$(pedir_numero)
    num2=$(pedir_numero)
    if [ $num2 -eq 0 ]; then
        echo "No se puede dividir por cero."
    else
        echo "El resultado de la división es: $((num1 / num2))"
    fi
    read -n 1 -s -r -p "Presiona cualquier tecla para continuar"
}

# función para multiplicar
multiplicar() {
    clear
    num1=$(pedir_numero)
    num2=$(pedir_numero)
    echo "El resultado de la multiplicación es: $((num1 * num2))"
    read -n 1 -s -r -p "Presiona cualquier tecla para continuar"
}
```

```
pcboot@pcboot-VirtualBox: ~/Documentos
GNU nano 6.2 calculadora.sh *
# Función principal

main() {
    while true; do
        clear # Borra la pantalla
        echo "Menú:"
        echo "1) Sumar"
        echo "2) Restar"
        echo "3) Dividir"
        echo "4) Multiplicar"
        echo "5) Salir"
        read -p "Elige una opción: " opcion

        case $opcion in
            1) sumar ;;
            2) restar ;;
            3) dividir ;;
            4) multiplicar ;;
            5) break ;;
            *) echo "Opción no válida." ;;
        esac
    done
}

# Llamada a la función principal
main
```

8.3.- Guarda el archivo con el nombre "calculadora.sh".

Guarda el archivo CTRL+O y salir del editor CTRL+X.

8.4.- Escribe los siguientes comandos:

```
pcboot@pcboot-VirtualBox: ~/Documentos
pcboot@pcboot-VirtualBox:~/Documentos$ ls
almacen          calculadora.sh      decada_edad.sh     horoscopochino.sh  sumarango.sh
altura_mayor.sh  cuenta10ficheros.sh diadelmes.sh       saluda.sh
pcboot@pcboot-VirtualBox:~/Documentos$ sudo chmod +x calculadora.sh
pcboot@pcboot-VirtualBox:~/Documentos$ ./calculadora.sh
```

```
pcboot@pcboot-VirtualBox: ~/Documentos
Menú:
1) Sumar
2) Restar
3) Dividir
4) Multiplicar
5) Salir
Elige una opción:
```

```
pcboot@pcboot-VirtualBox: ~/Documentos
Menú:
1) Sumar
2) Restar
3) Dividir
4) Multiplicar
5) Salir
Elige una opción: 1
Introduce un número: 10
Introduce un número: 20
El resultado de la suma es 30
Presiona cualquier tecla para continuar
```

```
pcboot@pcboot-VirtualBox: ~/Documentos
Introduce un número: 10
Introduce un número: 20
El resultado de la resta es: -10
Presiona cualquier tecla para continuar
```

```
pcboot@pcboot-VirtualBox: ~/Documentos
Introduce un número: 20
Introduce un número: 10
El resultado de la división es: 2
Presiona cualquier tecla para continuar
```

```
pcboot@pcboot-VirtualBox: ~/Documentos
Introduce un número: 100
Introduce un número: 0
No se puede dividir por cero.
Presiona cualquier tecla para continuar
```

```
pcboot@pcboot-VirtualBox: ~/Documentos
Introduce un número: 20
Introduce un número: 30
El resultado de la multiplicación es: 600
Presiona cualquier tecla para continuar
```

```
pcboot@pcboot-VirtualBox: ~/Documentos
Menú:
1) Sumar
2) Restar
3) Dividir
4) Multiplicar
5) Salir
Elige una opción: 5
pcboot@pcboot-VirtualBox:~/Documentos$
```

8.5.- Explicación línea a línea el código :

```
#!/bin/bash
```

Esta línea indica que el script debe ser interpretado por el intérprete de comandos bash.

```
# Función para pedir un número
pedir_numero() {
    read -p "Introduce un número: " num
    echo $num
}
```

La función pedir_numero solicita al usuario un número y lo devuelve.

```
# Función para sumar
sumar() {
    num1=$(pedir_numero)
    num2=$(pedir_numero)
    echo "El resultado de la suma es: $((num1 + num2))"
    read -n 1 -s -r -p "Presiona cualquier tecla para continuar"
}
```

La función sumar obtiene dos números utilizando la función pedir_numero, realiza la suma y muestra el resultado. Luego, pausa la ejecución hasta que se presione una tecla.

```
# Función para restar
restar() {
    num1=$(pedir_numero)
    num2=$(pedir_numero)
    echo "El resultado de la resta es: $((num1 - num2))"
    read -n 1 -s -r -p "Presiona cualquier tecla para continuar"
}
```

La función restar obtiene dos números utilizando la función pedir_numero, realiza la resta y muestra el resultado. Luego, pausa la ejecución hasta que se presione una tecla.

```
# Función para dividir
dividir() {
    num1=$(pedir_numero)
    num2=$(pedir_numero)
    if [ $num2 -eq 0 ]; then
        echo "No se puede dividir por cero."
    else
        echo "El resultado de la división es: $(( $num1 / $num2 ))"
    fi
    read -n 1 -s -r -p "Presiona cualquier tecla para continuar"
}
```

La función dividir obtiene dos números utilizando la función pedir_numero, verifica si el segundo número es cero y muestra un mensaje de error si es así. Si no es cero, realiza la división y muestra el resultado. Luego, pausa la ejecución hasta que se presione una tecla.

```
# Función para multiplicar
multiplicar() {
    num1=$(pedir_numero)
    num2=$(pedir_numero)
    echo "El resultado de la multiplicación es: $(( $num1 * $num2 ))"
    read -n 1 -s -r -p "Presiona cualquier tecla para continuar"
}
```

La función multiplicar obtiene dos números utilizando la función pedir_numero, realiza la multiplicación y muestra el resultado. Luego, pausa la ejecución hasta que se presione una tecla.


```
# Función principal
main() {
    while true; do
        clear # Borra la pantalla
        echo "Menú:"
        echo "1) Sumar"
        echo "2) Restar"
        echo "3) Dividir"
        echo "4) Multiplicar"
        echo "5) Salir"
        read -p "Elige una opción: " opcion

        case $opcion in
            1) sumar ;;
            2) restar ;;
            3) dividir ;;
            4) multiplicar ;;
            5) break ;;
            *) echo "Opción no válida." ;;
        esac
    done
}
```

La función main muestra un menú con las opciones de operación, lee la opción seleccionada por el usuario y llama a la función correspondiente. Utiliza un bucle while para mantener el menú en pantalla hasta que el usuario elija la opción para salir. Utiliza el comando clear para borrar la pantalla antes de mostrar el menú.

```
# Llamada a la función principal
main
```

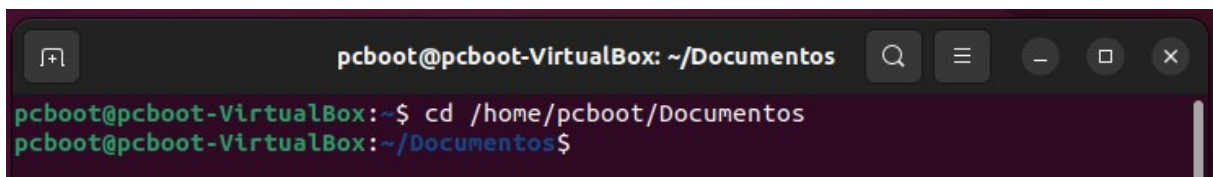
Finalmente, se llama a la función main para iniciar la ejecución del script. Este script implementa una calculadora simple en shell que permite al usuario realizar operaciones matemáticas básicas y pausar la ejecución hasta que se presione una tecla para continuar.

9.- Realizar un script que se llame usuarios.sh muestre la lista de información de un usuario del sistema pasado por parámetro (nombre de usuario, UID, GID y directorio de trabajo).

Cuando se muestre la información el script debe preguntar si quiere introducir otro usuario para mostrar su info o si se quiere salir del programa.

Se debe verificar que ese usuario está dado de alta en el sistema y por supuesto deberá mostrarse por pantalla el mensaje oportuno de darse tal circunstancia. Los errores arrojados por los comandos empleados serán enviados a un log destinado a ello, aunque el script avisará por pantalla si el usuario no existe.

9.1.- Abre la terminal (CTRL+ALT+T) y navega hasta el directorio donde se guarda el archivo.



```
pcboot@pcboot-VirtualBox: ~/Documentos
pcboot@pcboot-VirtualBox:~$ cd /home/pcboot/Documentos
pcboot@pcboot-VirtualBox:~/Documentos$
```

9.2.- Abre el editor de texto nano y escribe el siguiente ejercicio en un archivo llamado usuarios.sh.



```
GNU nano 6.2 usuarios.sh
#!/bin/bash

# Función para mostrar la información de un usuario

mostrar_informacion_usuario() {
    local username=$1
    if id "$username" &>/dev/null; then
        echo "Nombre de usuario: $username"
        echo "UID: $(id -u $username)"
        echo "GID: $(id -g $username)"
        echo "Directorio de trabajo: $(eval echo ~$username)"
    else
        echo "El usuario $username no existe."
    fi
}

# Función principal
main() {
    while true; do
        read -p "Introduce el nombre de usuario (o 'salir' para terminar): " username
        if [ "$username" = "salir" ]; then
            break
        fi
        mostrar_informacion_usuario "$username"
    done
}

# Llamada a la función principal
main
```

9.3.- Guarda el archivo con el nombre "usuarios.sh".

Guarda el archivo CTRL+O y salir del editor CTRL+X

9.4.- Escribe los siguientes comandos:

```
pcboot@pcboot-VirtualBox: ~/Documentos
pcboot@pcboot-VirtualBox:~/Documentos$ ls
almacen          calculadora.sh    decada_edad.sh    horoscopochino.sh  sumarango.sh
altura_mayor.sh  cuenta10ficheros.sh diadelmes.sh      saluda.sh           usuarios.sh
pcboot@pcboot-VirtualBox:~/Documentos$ sudo chmod +x usuarios.sh
pcboot@pcboot-VirtualBox:~/Documentos$ ls
almacen          calculadora.sh    decada_edad.sh    horoscopochino.sh  sumarango.sh
altura_mayor.sh  cuenta10ficheros.sh diadelmes.sh      saluda.sh           usuarios.sh
pcboot@pcboot-VirtualBox:~/Documentos$ ./usuarios.sh
Introduce el nombre de usuario (o 'salir' para terminar): pcboot
Nombre de usuario: pcboot
UID: 1000
GID: 1000
Directorio de trabajo: /home/pcboot
Introduce el nombre de usuario (o 'salir' para terminar): root
Nombre de usuario: root
UID: 0
GID: 0
Directorio de trabajo: /root
Introduce el nombre de usuario (o 'salir' para terminar): salir
pcboot@pcboot-VirtualBox:~/Documentos$
```

9.5.- Explicación línea a línea el código :

```
#!/bin/bash
```

Esta línea indica que el script debe ser interpretado por el intérprete de comandos bash.

```
# Función para mostrar la información de un usuario
mostrar_informacion_usuario() {
    local username=$1
    if id "$username" &>/dev/null; then
        echo "Nombre de usuario: $username"
        echo "UID: $(id -u $username)"
        echo "GID: $(id -g $username)"
        echo "Directorio de trabajo: $(eval echo ~$username)"
    else
        echo "El usuario $username no existe."
    fi
}
```

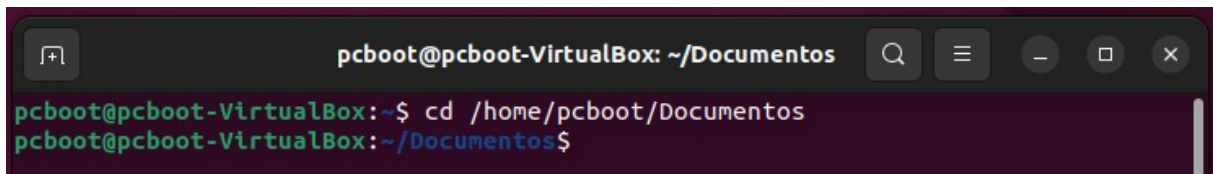
La función `mostrar_informacion_usuario` toma un nombre de usuario como parámetro, verifica si el usuario existe en el sistema utilizando el comando `id`, y muestra su nombre de usuario, UID, GID y directorio de trabajo si el usuario existe. Si el usuario no existe, muestra un mensaje indicando que el usuario no existe.

```
# Función principal
main() {
    while true; do
        read -p "Introduce el nombre de usuario (o 'salir' para terminar): " username
        if [ "$username" = "salir" ]; then
            break
        fi
        mostrar_informacion_usuario "$username"
    done
}
```

La función main utiliza un bucle while para solicitar al usuario que introduzca un nombre de usuario. Si el usuario introduce "salir", el bucle se interrumpe y el programa termina. De lo contrario, se llama a la función mostrar_informacion_usuario con el nombre de usuario introducido.

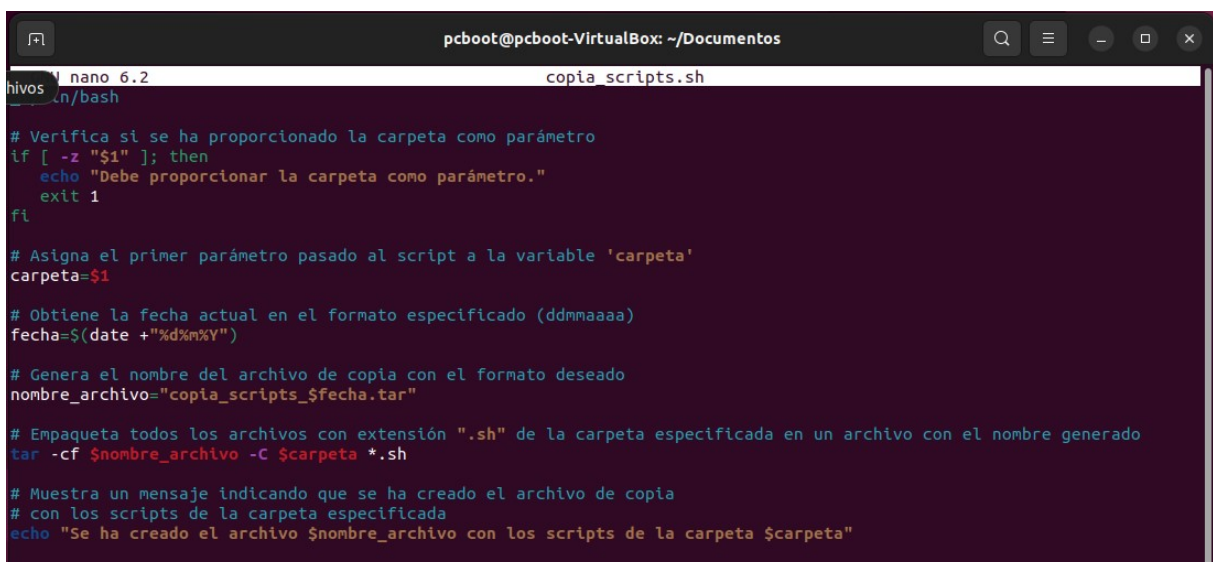
10.- Hacer un script llamado copia_scripts.sh, que haga un copiado de los scripts creados en la carpeta pasada como parámetro, deben ser empaquetados con el comando tar y el nombre del archivo debe tener el siguiente formato: "copia_scripts_ddmmaaaa.tar" siendo dd el día, mm el mes y aaaa el año en el que se produce la copia.

10.1.- Abre la terminal (CTRL+ALT+T) y navega hasta el directorio donde se guarda el archivo.



```
pcboot@pcboot-VirtualBox: ~/Documentos
pcboot@pcboot-VirtualBox:~$ cd /home/pcboot/Documentos
pcboot@pcboot-VirtualBox:~/Documentos$
```

10.2.- Abre el editor de texto nano y escribe el siguiente ejercicio en un archivo llamado copia_scripts.sh.



```
nano 6.2
copia_scripts.sh
# Verifica si se ha proporcionado la carpeta como parámetro
if [ -z "$1" ]; then
    echo "Debe proporcionar la carpeta como parámetro."
    exit 1
fi

# Asigna el primer parámetro pasado al script a la variable 'carpeta'
carpeta=$1

# Obtiene la fecha actual en el formato especificado (ddmmaaaa)
fecha=$(date +"%d%m%Y")

# Genera el nombre del archivo de copia con el formato deseado
nombre_archivo="copia_scripts_$fecha.tar"

# Empaqueta todos los archivos con extensión ".sh" de la carpeta especificada en un archivo con el nombre generado
tar -cf $nombre_archivo -C $carpeta *.sh

# Muestra un mensaje indicando que se ha creado el archivo de copia
# con los scripts de la carpeta especificada
echo "Se ha creado el archivo $nombre_archivo con los scripts de la carpeta $carpeta"
```

10.3.- Guarda el archivo con el nombre "copia_scripts.sh".

Guarda el archivo CTRL+O y salir del editor CTRL+X.

10.4.- Escribe los siguientes comandos:

```
pcboot@pcboot-VirtualBox: ~/Documentos
pcboot@pcboot-VirtualBox:~/Documentos$ ls
almacen      calculadora.sh  cuenta10ficheros.sh  diadelmes.sh  saluda.sh  usuarios.sh
altura_mayor.sh  copia_scripts.sh  decada_edad.sh  horoscopochino.sh  sumarango.sh
pcboot@pcboot-VirtualBox:~/Documentos$ sudo chmod +x copia_scripts.sh
pcboot@pcboot-VirtualBox:~/Documentos$ pwd
/home/pcboot/Documentos
pcboot@pcboot-VirtualBox:~/Documentos$ ./copia_scripts.sh /home/pcboot/Documentos
Se ha creado el archivo copia_scripts_19112023.tar con los scripts de la carpeta /home/pcboot/Documentos
pcboot@pcboot-VirtualBox:~/Documentos$ ls
almacen      calculadora.sh  copia_scripts.sh  decada_edad.sh  horoscopochino.sh  sumarango.sh
altura_mayor.sh  copia_scripts_19112023.tar  cuenta10ficheros.sh  diadelmes.sh  saluda.sh  usuarios.sh
pcboot@pcboot-VirtualBox:~/Documentos$
```

10.5.- Explicación línea a línea el código :

```
#!/bin/bash
```

Esta línea indica que el script debe ser interpretado por el intérprete de comandos bash.

```
# Verifica si se ha proporcionado la carpeta como parámetro
if [ -z "$1" ]; then
    echo "Debe proporcionar la carpeta como parámetro."
    exit 1
fi
```

En esta sección, se verifica si se ha proporcionado la carpeta como parámetro al script. Si no se ha proporcionado, se muestra un mensaje de error y se sale del script con el código de salida 1.

```
# Asigna el primer parámetro pasado al script a la variable 'carpeta'
carpeta=$1
```

Aquí se asigna el primer parámetro pasado al script a la variable carpeta.

```
# Obtiene la fecha actual en el formato especificado (ddmmaaaa)
fecha=$(date +"%d%m%Y")
```

Esta línea obtiene la fecha actual en el formato deseado (ddmmaaaa) y la almacena en la variable fecha.

```
# Genera el nombre del archivo de copia con el formato deseado
nombre_archivo="copia_scripts_$fecha.tar"
```

Aquí se genera el nombre del archivo de copia con el formato especificado.

```
# Empaqueta todos los archivos con extensión ".sh" de la carpeta  
# especificada en un archivo con el nombre generado anteriormente  
tar -cf $nombre_archivo -C $carpeta/*.sh
```

Esta línea utiliza el comando tar para empaquetar todos los archivos con extensión ".sh" de la carpeta especificada en un archivo con el nombre generado anteriormente.

```
# Muestra un mensaje indicando que se ha creado el archivo de copia  
# con los scripts de la carpeta especificada  
echo "Se ha creado el archivo $nombre_archivo con los scripts de la carpeta $carpeta"
```

Finalmente, se muestra un mensaje indicando que se ha creado el archivo de copia con los scripts de la carpeta especificada.


11.- OPCIONAL. Realiza un script que suba a tu cuenta de GitHub todos los ejercicios de la actividad pidiendo por parámetro el nombre de usuario correspondiente de la plataforma.

Si encuentras problema con la autenticación en GitHub mediante token, hazlo de manera local en tu repositorio o introduce el token al conectar con GitHub antes de lanzar el script.

11.1.- Primero crear un token.

Para crear un token de acceso personal en GitHub, sigue estos pasos:

1.- Accede a la configuración de tu cuenta: Inicia sesión en tu cuenta de GitHub y haz clic en tu foto de perfil en la esquina superior derecha. Selecciona "Settings" en el menú desplegable.



Sign in to GitHub

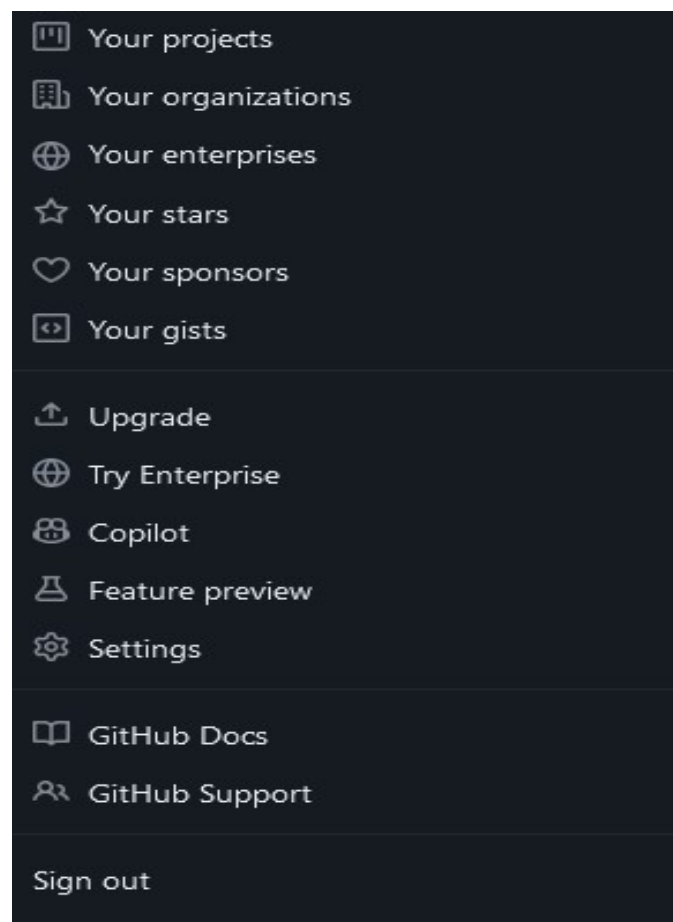
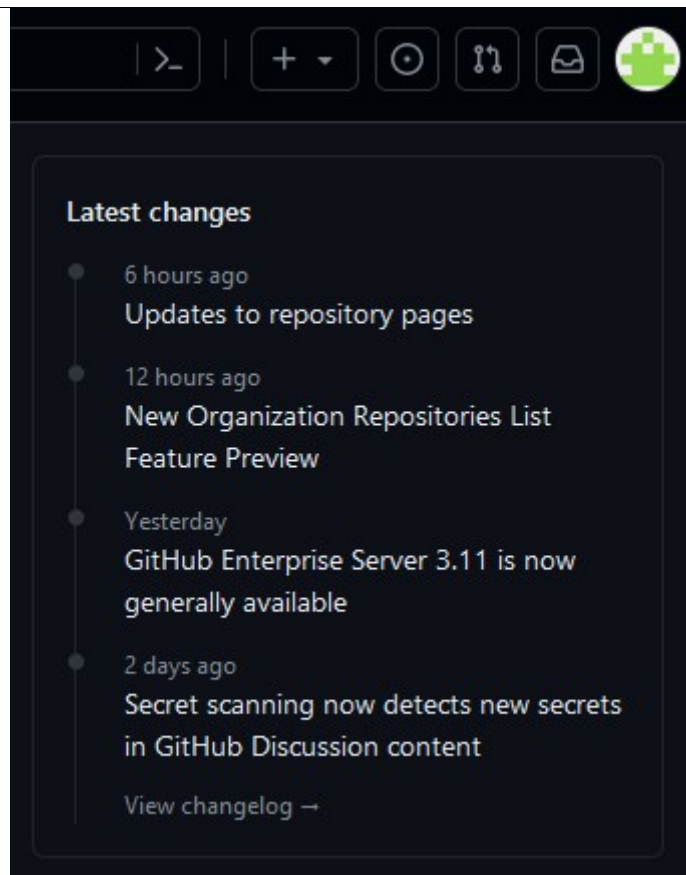
Username or email address

Password [Forgot password?](#)

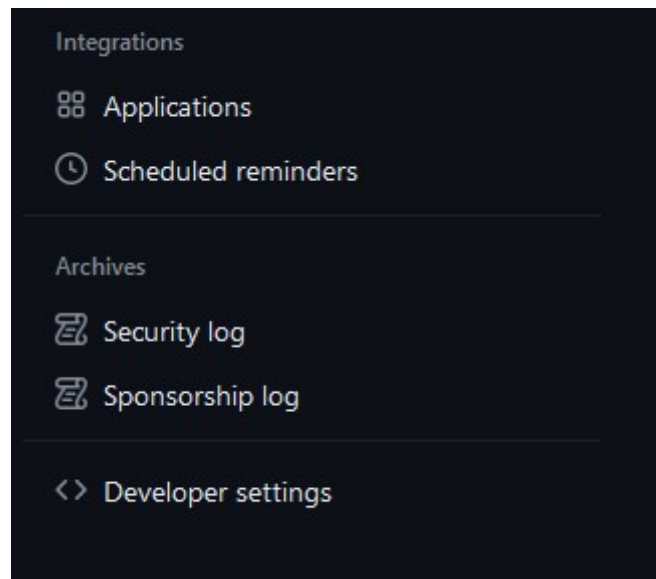
Sign in

[Sign in with a passkey](#)

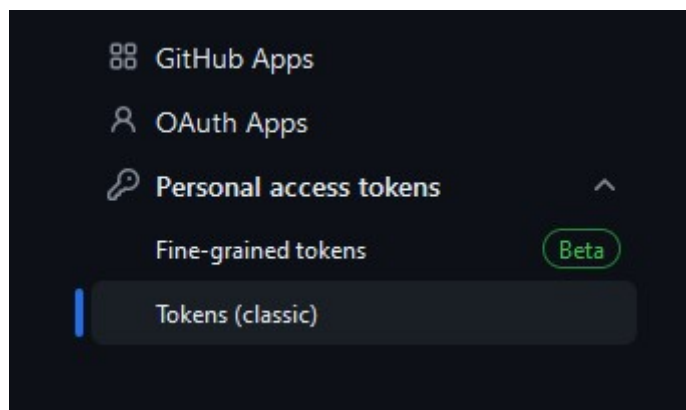
New to GitHub? [Create an account](#)



2.- Accede a los tokens de acceso personal: En el panel izquierdo, haz clic en "Developer settings" y luego en "Personal access tokens".



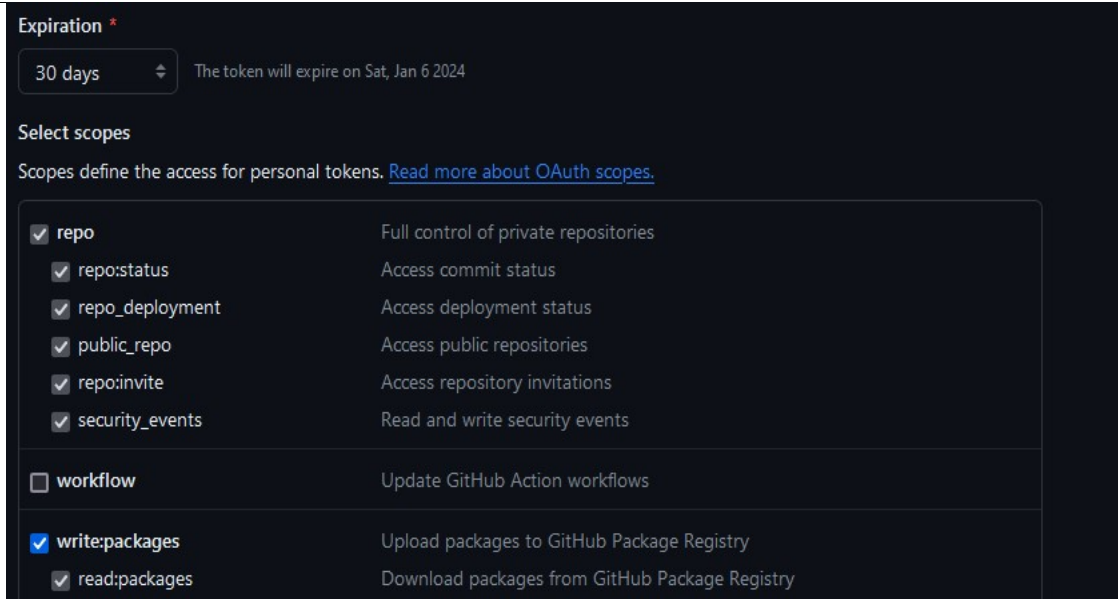
3.- Personal Access tokens: Haz clic en "Tokens (classic)".



4.- Proporciona la información del token: Después se te pedirá que proporciones información sobre el token. Primero, ingresa un nombre descriptivo para el token. Luego, selecciona los alcances (o permisos) que deseas otorgar al token.

Note

What's this token for?



Expiration *
30 days The token will expire on Sat, Jan 6 2024

Select scopes
Scopes define the access for personal tokens. [Read more about OAuth scopes.](#)

<input checked="" type="checkbox"/> repo	Full control of private repositories
<input checked="" type="checkbox"/> repo:status	Access commit status
<input checked="" type="checkbox"/> repo_deployment	Access deployment status
<input checked="" type="checkbox"/> public_repo	Access public repositories
<input checked="" type="checkbox"/> repo:invite	Access repository invitations
<input checked="" type="checkbox"/> security_events	Read and write security events
<input type="checkbox"/> workflow	Update GitHub Action workflows
<input checked="" type="checkbox"/> write:packages	Upload packages to GitHub Package Registry
<input checked="" type="checkbox"/> read:packages	Download packages from GitHub Package Registry

5.- Obtén el token: Desplázate hacia abajo y haz clic en "Generate token". Se generará un token de acceso personal. Asegúrate de copiar este token y guardarlo en un lugar seguro. Una vez que salgas de esta página, no podrás ver el token nuevamente.



Generate token Cancel

11.2.- Explicación línea a línea para subir repositorio al GitHub:

Esta línea de comando se utiliza para configurar globalmente la dirección de correo electrónico del usuario en Git. Cuando se utiliza la opción "--global", la configuración se aplica a todos los repositorios de Git en el sistema. El correo electrónico especificado se asocia con las confirmaciones que realice el usuario. Por ejemplo, si quieres configurar el correo electrónico "ejemplo@correo.com", la línea de comando completa sería:

```
pcboot@pcboot-VirtualBox:~/Documentos/prueba$ git config --global user.email "int.marisma@gmail.com"
```

La línea de comando "git config --global user.name" se utiliza para configurar globalmente el nombre de usuario en Git. Al usar la opción "--global", la configuración se aplica a todos los repositorios de Git en el sistema. Por ejemplo, si quieres configurar el nombre de usuario como "intmarisma", la línea de comando completa sería:

```
pcboot@pcboot-VirtualBox:~/Documentos/prueba$ git config --global user.name "intmarisma"
```

La línea de comando "git init" se utiliza para crear un nuevo repositorio de Git. Al ejecutar este comando, se crea un directorio oculto

llamado `.git` en el directorio actual, el cual contiene toda la metadata necesaria para el nuevo repositorio, incluyendo subdirectorios para objetos, referencias y archivos de plantilla. También se crea un archivo `HEAD` que apunta al commit actual. Si se desea crear un repositorio sin ningún commit inicial, se puede utilizar la opción `--bare`. Aquí tienes un resumen de lo que hace el comando `"git init"`:

Crea un nuevo repositorio de Git en el directorio actual.

Inicializa un directorio como un repositorio de Git, creando un subdirectorio `.git` que contiene toda la metadata del repositorio.

Puede usarse para convertir un proyecto existente, no versionado, en un repositorio de Git.

Por ejemplo, para inicializar un nuevo repositorio, simplemente ejecuta `"git init"` en el directorio del proyecto. Si deseas crear un repositorio sin ningún commit inicial, puedes usar `"git init --bare"`, la línea de comando completa sería:

```
pcboot@pcboot-VirtualBox: ~/Documentos/prueba2$ git init
```

La instrucción `"git add"` se utiliza para agregar archivos nuevos o modificados en tu directorio de trabajo al área de preparación de Git. Esto significa que los archivos agregados con `"git add"` están listos para ser incluidos en el próximo commit. Es una parte importante del flujo de trabajo de Git, ya que te permite seleccionar los cambios específicos que deseas incluir en tu próxima confirmación. Por ejemplo, puedes usar `"git add archivo.txt"` para agregar un archivo específico, o `"git add ."` para agregar todos los archivos modificados en tu directorio de trabajo. Es importante notar que `"git add"` no afecta directamente el repositorio en sí, solo prepara los cambios para ser confirmados con `"git commit"`, la línea de comando completa sería:

```
pcboot@pcboot-VirtualBox: ~/Documentos/prueba2$ git add .
```

La instrucción `"git commit -m 'Poner comentario'"` se utiliza para confirmar los cambios en el área de preparación de Git. Al ejecutar esta instrucción, se crea un nuevo commit con los cambios que han sido previamente agregados con `"git add"`. El mensaje entre comillas después de la bandera `"-m"` se utiliza para proporcionar una descripción corta y significativa de los cambios realizados en el commit. Por ejemplo, si has realizado cambios en uno o varios archivos y quieres confirmar esos cambios, puedes usar `"git add ."` para agregar los cambios al área de

preparación, seguido de "git commit -m 'Mensaje descriptivo'" para crear el commit. Esto ayuda a mantener un historial claro y comprensible de los cambios en el repositorio, la línea de comando completa sería:

```
pcboot@pcboot-VirtualBox:~/Documentos/prueba2$ git commit -m "subir los archivo"
[master (commit-raiz) dc6e6d0] subir los archivo
9 files changed, 301 insertions(+)
create mode 100755 altura_mayor.sh
create mode 100755 calculadora.sh
create mode 100755 copia_scripts.sh
create mode 100755 cuenta10ficheros.sh
create mode 100755 decada_edad.sh
create mode 100755 diadelmes.sh
create mode 100755 horoscopochino.sh
create mode 100755 sumarango.sh
create mode 100755 usuarios.sh
```

La instrucción "git branch -M main" se utiliza para renombrar la rama actual a "main". Anteriormente, la rama principal en Git solía llamarse "master", pero en un esfuerzo por hacer el lenguaje más inclusivo y eliminar referencias a la esclavitud, se ha optado por cambiar el nombre predeterminado a "main". Por lo tanto, al ejecutar "git branch -M main", se está renombrando la rama actual a "main" en lugar de "master", la línea de comando completa sería:

```
pcboot@pcboot-VirtualBox:~/Documentos/prueba2$ git branch -M main
```

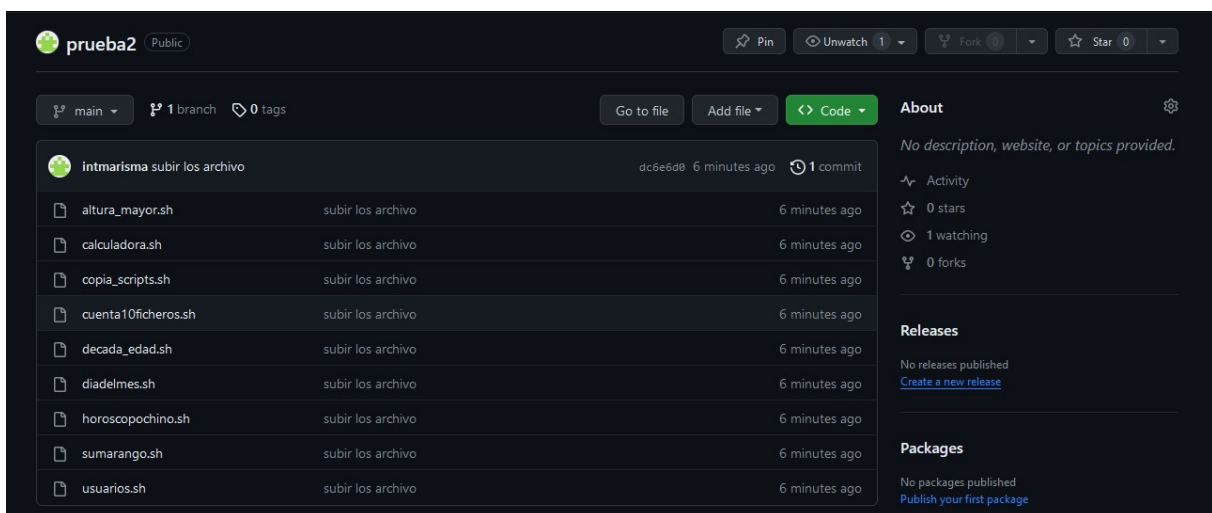
La instrucción "git remote add origin <URL>" se utiliza para agregar un nuevo control remoto a tu repositorio de Git. El término "origin" es el nombre convencionalmente utilizado para el control remoto principal, pero se puede elegir otro nombre si se prefiere. El "<URL>" es la dirección del repositorio remoto al que deseas enlazar. Por ejemplo, si deseas enlazar tu repositorio local a un repositorio remoto en GitHub, puedes usar "git remote add origin <URL>". Una vez que has agregado el control remoto, puedes usar el nombre del control remoto (en este caso, "origin") en otras instrucciones de Git, como "git push" y "git pull", en lugar de tener que escribir la URL completa cada vez, la línea de comando completo sería:

```
pcboot@pcboot-VirtualBox:~/Documentos/prueba2$ git remote add origin https://github.com/intmarisna/prueba2.git
```

La instrucción "git push -u origin main" se utiliza para subir la rama local "main" a un repositorio remoto llamado "origin" y establecer el seguimiento entre la rama local y la rama remota. La opción "-u" es para configurar la rama remota como rama de seguimiento, lo que significa que, en el futuro, puedes simplemente usar "git push" o "git pull" sin especificar el nombre de la rama y Git sabrá a qué rama remota hacer referencia. Esto es útil para simplificar el flujo de trabajo. En resumen, esta instrucción sube la rama local "main" al repositorio remoto "origin" y establece el seguimiento entre la rama local y la rama remota, la línea de comando completo sería:

```
pcboot@pcboot-VirtualBox: ~/Documentos/prueba2
pcboot@pcboot-VirtualBox:~/Documentos/prueba2$ git push -u origin main
Username for 'https://github.com': intmarisma
Password for 'https://intmarisma@github.com':
Enumerando objetos: 11, listo.
Contando objetos: 100% (11/11), listo.
Comprimiendo objetos: 100% (11/11), listo.
Escribiendo objetos: 100% (11/11), 3.93 KiB | 309.00 KiB/s, listo.
Total 11 (delta 0), reusados 0 (delta 0), pack-reusados 0
To https://github.com/intmarisma/prueba2.git
 * [new branch]      main -> main
Rama 'main' configurada para hacer seguimiento a la rama remota 'main' de 'origin'.
pcboot@pcboot-VirtualBox:~/Documentos/prueba2$
```

El resultado de todo el proceso descrito es la subida de la rama local "main" al repositorio remoto "origin" y el establecimiento de un seguimiento entre la rama local y la rama remota. Esto significa que los cambios realizados en la rama local ahora están disponibles en el repositorio remoto, y cualquier cambio realizado en la rama remota se reflejará en la rama local cuando se realice un "git pull". Este flujo de trabajo es fundamental en el uso de Git para colaborar en proyectos de desarrollo de software.



11.3.- Repositorio GitHub

Todo mis ejercicios están disponibles en el repositorio de GitHub en el siguiente enlace : <https://github.com/intmarisma/PSS-practicas-2>

Índice Alfabético

actual	13, 15, 16, 17, 18, 36	intérprete.....	7, 9, 11, 14, 17, 21, 23, 29, 33, 36
almacena.....	7, 12, 36	introducimos.....	13, 22
altura.....	8, 9	mensaje.....	7, 9, 11, 12, 14, 15, 25, 30, 32, 33, 36, 37
altura_mayor.sh	8	metros	8, 9
año.....	13, 15, 17, 18, 19, 21, 35	multiplo10.sh	6
aritmética	7	nacimos	13, 19
automática	13	nano	6, 13, 22, 26, 32
bash.....	7, 9, 11, 14, 17, 21, 23, 24, 29, 33, 36	navega.....	6, 8, 10, 13, 16, 19, 22, 25, 32, 35
básica	7	pantalla.....	13, 19, 22, 25, 31, 32
bc.....	9	personas	8, 9
centímetros	8, 9	resultado.....	13, 15, 17, 18, 21, 22, 24, 25, 29, 30
código.....	6, 11, 13, 14, 15, 22, 26, 32, 36	resultados	12
comandos.....	6, 7, 8, 9, 11, 14, 17, 20, 21, 23, 27, 29, 32, 33, 36	resumen.....	9
década	13, 15	salida.....	13, 16, 36
decada_edad.sh	13	script.....	6, 7, 8, 9, 10, 11, 13, 14, 15, 16, 17, 18, 19, 21, 22, 23, 24, 25, 29, 31, 32, 33, 35, 36, 38, 46
directorio.....	6, 8, 10, 11, 12, 13, 16, 19, 22, 25, 32, 33, 35	sort	9
directorios	10	terminal.....	6, 8, 10, 13, 16, 19, 22, 24, 25, 32, 35
edad	13, 15	variables	9, 17
editor.....	6, 8, 10, 13, 16, 20, 22, 23, 26, 27, 32, 35	verifican	11, 12, 14, 15
estado.....	11, 14, 15	verificar.....	11, 32
ficheros	10, 12		
find.....	12		
Guarda.....	6, 8, 10, 13, 16, 20, 23, 27, 32, 35		