

# Temas Selectos de Ingeniería en Computación II

## FUNCIONES

### Equipo

Los Hijos de Pablito



### Grito de Batalla

¡Yibambe Yibambe UAMKANDA FOREVER!

### Integrantes

Flores Machuca Ernesto

2193041595

## **1. Requerimientos**

1. Servidor Linux/UNIX.
2. Cuenta en el servidor.

## **2. Descripción**

Los administradores, programadores, y usuarios en general requieren usar las herramientas y utilerías, que proporciona el sistema operativo Linux, para hacer más fácil sus labores diarias.

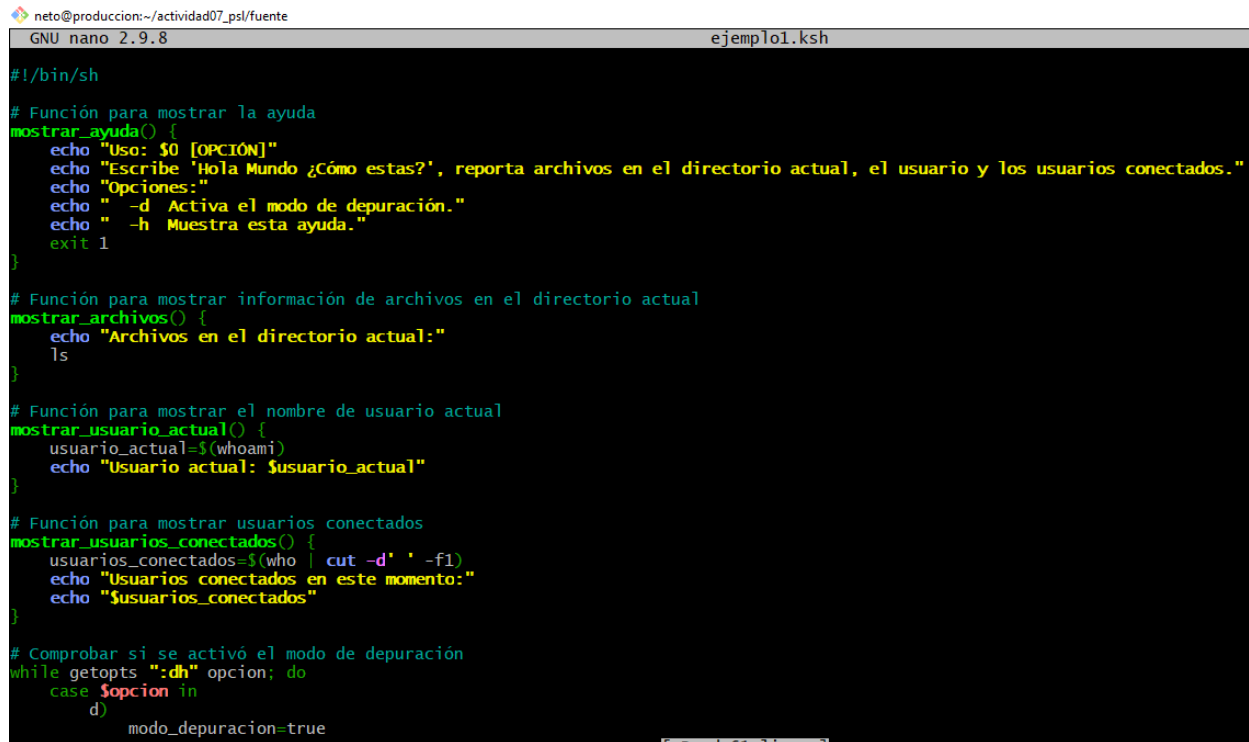
Para administrar eficientemente se usa la programación en Shell. Esta permite automatizar las tareas que se deben de hacer frecuentemente, como levantar servicios, administrar cuentas de usuarios, respaldar los datos, etc.

Los tipos de datos permiten restringir el conjunto de valores de una variable. Permiten un mejor control de los datos que maneja el programa.

Elaboraras varios programas para ilustrar esto

### 3. Desarrollo

a) **Crea un Shell script que escriba Hola Mundo ¿Como estas ?, reporte los archivos en el directorio actual, el usuario con el que estas conectado, los usuarios conectados en ese momento. Agregar la opción -d que active el modo de depuración. Usar funciones.**



```
neto@produccion:~/actividad07_psl/fuente
GNU nano 2.9.8                                ejemplo1.ksh

#!/bin/sh

# Función para mostrar la ayuda
mostrar_ayuda() {
    echo "Uso: $0 [OPCIÓN]"
    echo "Escribe 'Hola Mundo ¿Cómo estas?', reporta archivos en el directorio actual, el usuario y los usuarios conectados."
    echo "Opciones:"
    echo "  -d Activa el modo de depuración."
    echo "  -h Muestra esta ayuda."
    exit 1
}

# Función para mostrar información de archivos en el directorio actual
mostrar_archivos() {
    echo "Archivos en el directorio actual:"
    ls
}

# Función para mostrar el nombre de usuario actual
mostrar_usuario_actual() {
    usuario_actual=$(whoami)
    echo "Usuario actual: $usuario_actual"
}

# Función para mostrar usuarios conectados
mostrar_usuarios_conectados() {
    usuarios_conectados=$(who | cut -d' ' -f1)
    echo "Usuarios conectados en este momento:"
    echo "$usuarios_conectados"
}

# Comprobar si se activó el modo de depuración
while getopts ":dh" opcion; do
    case $opcion in
        d)
            modo_depuracion=true
    esac
done
```

Ilustración 1. Script shell Ejercicio 1 parte 1

```

neto@produccion:~/actividad07_psl/fuente
GNU nano 2.9.8

# Función para mostrar el nombre de usuario actual
mostrar_usuario_actual() {
    usuario_actual=$(whoami)
    echo "Usuario actual: $usuario_actual"
}

# Función para mostrar usuarios conectados
mostrar_usuarios_conectados() {
    usuarios_conectados=$(who | cut -d' ' -f1)
    echo "Usuarios conectados en este momento:"
    echo "$usuarios_conectados"
}

# Comprobar si se activó el modo de depuración
while getopts ":dh" opcion; do
    case $opcion in
        d)
            modo_depuracion=true
            ;;
        h)
            mostrar_ayuda
            ;;
        \?)
            echo "Opción no válida: -$OPTARG"
            mostrar_ayuda
            ;;
    esac
done

# Mensaje principal
if [ "$modo_depuracion" = true ]; then
    echo "Modo de depuración activado."
fi

echo "Hola Mundo ¿Cómo estás?"

```

Ilustración 2. Script shell ejercicio 1 parte 2

```

neto@produccion:~/actividad07_psl/fuente
GNU nano 2.9.8

        modo_depuracion=true
        ;;
    h)
        mostrar_ayuda
        ;;
    \?)
        echo "Opción no válida: -$OPTARG"
        mostrar_ayuda
        ;;
    esac
done

# Mensaje principal
if [ "$modo_depuracion" = true ]; then
    echo "Modo de depuración activado."
fi

echo "Hola Mundo ¿Cómo estás?"

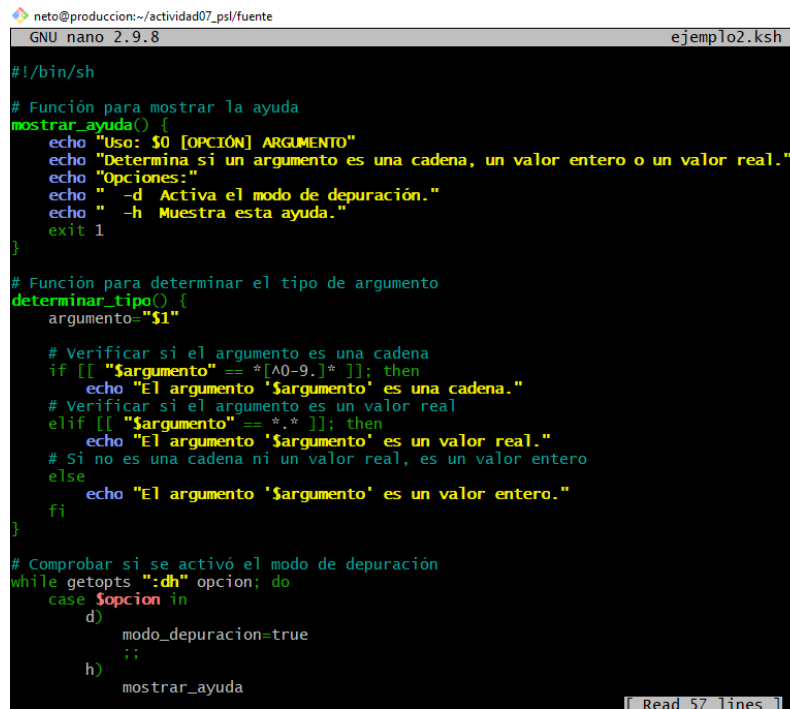
if [ "$modo_depuracion" = true ]; then
    mostrar_archivos
    mostrar_usuario_actual
fi

mostrar_usuarios_conectados

```

Ilustración 3. Script shell ejercicio1 parte 3

**b) Crea un Shell script que determine si un argumento es una cadena, un valor entero o un valor real. Agregar la opción -d que active el modo de depuración. Usar funciones.**



```
neto@produccion:~/actividad07_psl/fuente
GNU nano 2.9.8 ejemplo2.ksh

#!/bin/sh

# Función para mostrar la ayuda
mostrar_ayuda() {
    echo "Uso: $0 [OPCIÓN] ARGUMENTO"
    echo "Determina si un argumento es una cadena, un valor entero o un valor real."
    echo "Opciones:"
    echo "  -d Activa el modo de depuración."
    echo "  -h Muestra esta ayuda."
    exit 1
}

# Función para determinar el tipo de argumento
determinar_tipo() {
    argumento="$1"

    # Verificar si el argumento es una cadena
    if [[ "$argumento" == *[A0-9.]* ]]; then
        echo "El argumento '$argumento' es una cadena."
    # Verificar si el argumento es un valor real
    elif [[ "$argumento" == *.* ]]; then
        echo "El argumento '$argumento' es un valor real."
    # Si no es una cadena ni un valor real, es un valor entero
    else
        echo "El argumento '$argumento' es un valor entero."
    fi
}

# Comprobar si se activó el modo de depuración
while getopts ":dh" opcion; do
    case $opcion in
        d)
            modo_depuracion=true
            ;;
        h)
            mostrar_ayuda
    esac
done
```

Read 57 lines

Ilustración 4. Script shell ejercicio 2 parte 1

```

neto@produccion:~/actividad07_psl/fuente
GNU nano 2.9.8

if [[ "$argumento" == *[^0-9.]* ]]; then
    echo "El argumento '$argumento' es una cadena."
# Verificar si el argumento es un valor real
elif [[ "$argumento" == *.* ]]; then
    echo "El argumento '$argumento' es un valor real."
# Si no es una cadena ni un valor real, es un valor entero
else
    echo "El argumento '$argumento' es un valor entero."
fi
}

# Comprobar si se activó el modo de depuración
while getopts ":dh" opcion; do
    case $opcion in
        d)
            modo_depuracion=true
            ;;
        h)
            mostrar_ayuda
            ;;
        \?)
            echo "Opción no válida: -$OPTARG"
            mostrar_ayuda
            ;;
    esac
done

# Verificar que se proporcionó un argumento
if [ $# -eq 0 ]; then
    mostrar_ayuda
fi

# Mostrar el tipo del argumento
if [ "$modo_depuracion" = true ]; then
    echo "Modo de depuración activado."
fi

```

Ilustración 5. Script shell ejercicio2 parte 2

```

        mostrar_ayuda
        ;;
    \?)
        echo "Opción no válida: -$OPTARG"
        mostrar_ayuda
        ;;
    esac
done

# Verificar que se proporcionó un argumento
if [ $# -eq 0 ]; then
    mostrar_ayuda
fi

# Mostrar el tipo del argumento
if [ "$modo_depuracion" = true ]; then
    echo "Modo de depuración activado."
fi

argumento="$1"
determinar_tipo "$argumento"

```

Ilustración 6. Script shell ejercicio2 parte 3

c) Crea un Shell script que reciba como argumentos varios archivos. Agregar la opción -p que reporte los permisos del archivo, la opción -t que reporte el tipo del archivo, -u que reporte el usuario dueño del archivo, -g que reporte el grupo del archivo, -T que reporte el tamaño del archivo, -f que reporte la fecha de la actualización más reciente del archivo. Usar funciones.

```
neto@produccion:~/actividad07_ps/fuente
GNU nano 2.9.8 ejemplo3.ks

#!/bin/sh

# Función para mostrar la ayuda
mostrar_ayuda() {
    echo "Uso: $0 [OPCIÓN] ARCHIVO [ARCHIVO...]"
    echo "Reporta información sobre los archivos especificados."
    echo "Opciones:"
    echo "  -p Reporta los permisos del archivo."
    echo "  -t Reporta el tipo del archivo."
    echo "  -u Reporta el usuario dueño del archivo."
    echo "  -g Reporta el grupo del archivo."
    echo "  -T Reporta el tamaño del archivo."
    echo "  -f Reporta la fecha de la actualización más reciente del archivo."
    echo "  -h Muestra esta ayuda."
    exit 1
}

# Función para mostrar los permisos del archivo
mostrar_permisos() {
    archivo="$1"
    permisos=$(stat -c "%A" "$archivo")
    echo "Permisos de $archivo: $permisos"
}

# Función para mostrar el tipo del archivo
mostrar_tipo() {
    archivo="$1"
    tipo=$(file -b --mime-type "$archivo")
    echo "Tipo de $archivo: $tipo"
}

# Función para mostrar el usuario dueño del archivo
mostrar_dueno() {
    archivo="$1"
    dueno=$(stat -c "%U" "$archivo")
    echo "Dueño de $archivo: $dueno"
}
```

[ Read 68 lines ]

Ilustración 7. Script shell ejercicio3 parte 1

```
neto@produccion:~/actividad07_ps/fuente
GNU nano 2.9.8 ejemplo3.ks

}

# Función para mostrar el grupo del archivo
mostrar_grupo() {
    archivo="$1"
    grupo=$(stat -c "%G" "$archivo")
    echo "Grupo de $archivo: $grupo"
}

# Función para mostrar el tamaño del archivo
mostrar_tamano() {
    archivo="$1"
    tamano=$(stat -c "%s" "$archivo")
    echo "Tamaño de $archivo: $tamano bytes"
}

# Función para mostrar la fecha de la actualización más reciente del archivo
mostrar_fecha_actualizacion() {
    archivo="$1"
    fecha_actualizacion=$(stat -c "%y" "$archivo")
    echo "Fecha de actualización de $archivo: $fecha_actualizacion"
}

# Comprobar si se proporcionaron argumentos
if [ $# -eq 0 ]; then
    mostrar_ayuda
fi

# Comprobar opciones y procesar argumentos
while getopts ":ptugTfh" opcion; do
    case $
```

Ilustración 8. Script shell ejercicio3 parte 2

**d) Crea un Shell script que lea una frase en una cadena. Las palabras de la frase almacénalas separadas en un arreglo. Imprime las palabras de la frase en orden inverso. Usar funciones.**

```
neto@produccion:~/actividad07_psl/fuente
GNU nano 2.9.8 ejemplo4.ksh

#!/bin/sh

# Función para mostrar la ayuda
mostrar_ayuda() {
    echo "Uso: $0 [OPCIÓN] FRASE"
    echo "Lee una frase en una cadena, almacena las palabras en un arreglo y las imprime en orden inverso."
    echo "Opciones:"
    echo "  -h Muestra esta ayuda."
    exit 1
}

# Comprobar si se proporcionaron argumentos
if [ $# -eq 0 ]; then
    mostrar_ayuda
fi

# Comprobar opciones y procesar argumentos
while getopts ":h" opcion; do
    case $opcion in
        h)
            mostrar_ayuda
            ;;
        \?)
            echo "Opción no válida: -$OPTARG"
            mostrar_ayuda
            ;;
        esac
    done

# Eliminar las opciones procesadas
shift $((OPTIND - 1))

# Leer la frase como una cadena
frase="$1"

# Función para separar la frase en palabras y almacenarlas en un arreglo
```

Ilustración 9. Script shell ejercicio 4 parte 1

```
neto@produccion:~/actividad07_psl/fuente
GNU nano 2.9.8 ejemp

while getopts ":h" opcion; do
    case $opcion in
        h)
            mostrar_ayuda
            ;;
        \?)
            echo "Opción no válida: -$OPTARG"
            mostrar_ayuda
            ;;
        esac
    done

# Eliminar las opciones procesadas
shift $((OPTIND - 1))

# Leer la frase como una cadena
frase="$1"

# Función para separar la frase en palabras y almacenarlas en un arreglo
separar_palabras() {
    frase="$1"
    # Usar el espacio como delimitador para separar las palabras
    IFS=" " read -ra palabras <<< "$frase"
}

# Función para imprimir las palabras en orden inverso
imprimir_en_orden_inverso() {
    arreglo=("$@")
    # Recorrer el arreglo en orden inverso e imprimir cada palabra
    for ((i=${#arreglo[@]}-1; i>=0; i--)); do
        echo "${arreglo[i]}"
    done
}

# Llamar a la función para separar las palabras
separar_palabras "$frase"
```

Ilustración 10. Script shell ejercicio4 parte2



e) Crea un Shell script que le solicite al usuario el tamaño de un arreglo. Este se llenará con números aleatorios. En un ciclo infinito se le pide al usuario que adivine un número que este en el arreglo. El programa termina cuando se adivine un número que este en el arreglo o se llegue a un máximo de intentos. Usar funciones.

```
neto@produccion:~/actividad07_psl/fuente
GNU nano 2.9.8 ejemplo5.

#!/bin/sh

# Función para mostrar la ayuda
mostrar_ayuda() {
    echo "Uso: $0 [OPCIÓN]"
    echo "Juego: Adivina el número en un arreglo."
    echo "Opciones:"
    echo "  -h Muestra esta ayuda."
    exit 1
}

# Función para generar un arreglo de números aleatorios
generar_arreglo() {
    local tamaño="$1"
    arreglo=()
    for ((i = 0; i < tamaño; i++)); do
        numero=$((RANDOM % 100)) # Genera números aleatorios entre 0 y 99
        arreglo+=("$numero")
    done
}

# Función para jugar a adivinar el número
jugar() {
    local arreglo=("$@")
    local tamaño=${#arreglo[@]}
    local intentos_realizados=0
    local max_intentos=5 # Puedes ajustar el número máximo de intentos aquí

    echo "Bienvenido al juego 'Adivina el número en un arreglo'."
    echo "Tienes un máximo de $max_intentos intentos para adivinar."

    # Número aleatorio para adivinar
    numero_a_adivinar=${arreglo[$((RANDOM % tamaño))]}
}
```

Ilustración 11. Script shell ejercicio 5 parte 1

```
neto@produccion:~/actividad07_psl/fuente
GNU nano 2.9.8 ejemplo5.ksh

read -p "Adivina un número en el arreglo (0-99): " adivinanza
((intentos_realizados++))

if [[ $adivinanza -eq $numero_a_adivinar ]]; then
    echo "¡Felicidades! Adivinaste el número $numero_a_adivinar en $intentos_realizados intentos."
    break
elif [[ $intentos_realizados -ge $max_intentos ]]; then
    echo "Lo siento, has alcanzado el máximo de intentos. El número era $numero_a_adivinar."
    break
else
    echo "Intento incorrecto. Intenta de nuevo."
fi
done

# Comprobar opciones y procesar argumentos
while getopts ":h" opcion; do
    case $opcion in
        h)
            mostrar_ayuda
            ;;
        \?)
            echo "Opción no válida: -$OPTARG"
            mostrar_ayuda
            ;;
    esac
done

# Eliminar las opciones procesadas
shift $((OPTIND - 1))

# Comprobar si se proporcionó el tamaño del arreglo como argumento
if [ $# -eq 0 ]; then
    echo "Debes proporcionar el tamaño del arreglo como argumento."
fi
```

Ilustración 12. Script shell ejercicio 5 parte 2

```
neto@produccion:~/actividad07_psl/fuente
GNU nano 2.9.8

    h)
    mostrar_ayuda
    ;;
    \?)
    echo "Opción no válida: -$OPTARG"
    mostrar_ayuda
    ;;
    esac
done

# Eliminar las opciones procesadas
shift $((OPTIND - 1))

# Comprobar si se proporcionó el tamaño del arreglo como argumento
if [ $# -eq 0 ]; then
    echo "Debes proporcionar el tamaño del arreglo como argumento."
    mostrar_ayuda
fi

# Obtener el tamaño del arreglo desde el argumento
tamano_arreglo="$1"

# Llamar a la función para generar el arreglo
generar_arreglo "$tamano_arreglo"

# Llamar a la función para jugar
jugar "${arreglo[@]}"
```

Ilustración 13. Script shell ejercicio 5 parte 3

f) **Crea un Shell script que ordene los elementos en un arreglo por el método de la burbuja. Los elementos y el tamaño del arreglo se generan aleatoriamente. Usar funciones.**

```

neto@produccion:~/actividad07_psl/fuente
GNU nano 2.9.8 ejemplo6.ksh

#!/bin/sh

# Función para mostrar la ayuda
mostrar_ayuda() {
    echo "Uso: $0 [OPCIÓN] TAMAÑO_ARREGLO"
    echo "Genera un arreglo de elementos aleatorios y lo ordena utilizando el método de la burbuja."
    echo "Opciones:"
    echo "  -h Muestra esta ayuda."
    exit 1
}

# Función para generar un arreglo de elementos aleatorios
generar_arreglo() {
    local tamaño="$1"
    arreglo=()
    for ((i = 0; i < tamaño; i++)); do
        numero=$((RANDOM % 100)) # Genera números aleatorios entre 0 y 99
        arreglo+=("$numero")
    done
}

# Función para ordenar el arreglo utilizando el método de la burbuja
ordenar_arreglo() {
    local arreglo=("$@")
    local tamaño=${#arreglo[@]}
    local intercambio

    for ((i = 0; i < tamaño - 1; i++)); do
        intercambio=false
        for ((j = 0; j < tamaño - 1 - i; j++)); do
            if [ ${arreglo[j]} -gt ${arreglo[j+1]} ]; then
                # Realizar un intercambio si el elemento actual es mayor que el siguiente
                temp=${arreglo[j]}
                arreglo[j]=${arreglo[j+1]}
                arreglo[j+1]=$temp
                intercambio=true
            fi
        done
    done
}

# Si no se realizó ningún intercambio en esta pasada, el arreglo ya está ordenado
if [ "$intercambio" = false ]; then
    break
fi
done
}

# Comprobar opciones y procesar argumentos
while getopts ":h" option; do
    case $option in
        h)
            mostrar_ayuda
            ;;
        \?)
            echo "Opción no válida: -$OPTARG"
            mostrar_ayuda
            ;;
    esac
done

# Eliminar las opciones procesadas
shift $((OPTIND - 1))

# Comprobar si se proporcionó el tamaño del arreglo como argumento
if [ $# -eq 0 ]; then
    echo "Debes proporcionar el tamaño del arreglo como argumento."
    mostrar_ayuda
fi

# Obtener el tamaño del arreglo desde el argumento

```

Ilustración 14. Script shell ejercicio 6 parte 1

```

neto@produccion:~/actividad07_psl/fuente
GNU nano 2.9.8 ejemplo6.ksh

        fi
    done

    # Si no se realizó ningún intercambio en esta pasada, el arreglo ya está ordenado
    if [ "$intercambio" = false ]; then
        break
    fi
done
}

# Comprobar opciones y procesar argumentos
while getopts ":h" option; do
    case $option in
        h)
            mostrar_ayuda
            ;;
        \?)
            echo "Opción no válida: -$OPTARG"
            mostrar_ayuda
            ;;
    esac
done

# Eliminar las opciones procesadas
shift $((OPTIND - 1))

# Comprobar si se proporcionó el tamaño del arreglo como argumento
if [ $# -eq 0 ]; then
    echo "Debes proporcionar el tamaño del arreglo como argumento."
    mostrar_ayuda
fi

# Obtener el tamaño del arreglo desde el argumento

```

Ilustración 15. Script shell ejercicio 6 parte 2

**g) Crea un Shell script que a partir de problema anterior implemente el juego del segundo problema usando búsqueda binaria para buscar el número. Usar funciones.**

```
neto@produccion:~/actividad07_psl/fuente
GNU nano 2.9.8

    echo "Opción no válida: -$OPTARG"
    mostrar_ayuda
    ;;
esac
done

# Eliminar las opciones procesadas
shift $((OPTIND - 1))

# Comprobar si se proporcionó el tamaño del arreglo como argumento
if [ $# -eq 0 ]; then
    echo "Debes proporcionar el tamaño del arreglo como argumento."
    mostrar_ayuda
fi

# Obtener el tamaño del arreglo desde el argumento
tamano_arreglo="$1"

# Llamar a la función para generar el arreglo
generar_arreglo "$tamano_arreglo"

# Mostrar el arreglo antes de ordenar
echo "Arreglo antes de ordenar: ${arreglo[@]}"

# Llamar a la función para ordenar el arreglo
ordenar_arreglo "${arreglo[@]}"

# Mostrar el arreglo ordenado
echo "Arreglo ordenado: ${arreglo[@]}"
```

Ilustración 16. Script shell ejercicio 7

**Finalmente comprimimos la carpeta que contiene los Shell script realizados anteriormente**



Ilustración 17. Compresión de la carpeta de los shell script