# Trabajo Práctico 3

## Introducción a los Sistemas Operativos

1. El Shell Scripting es la técnica de escribir un **guion** o programa de ordenador que contiene una serie de comandos que serían normalmente ejecutados línea por línea en la interfaz de línea de comandos (shell) del sistema operativo.
   Los scripts de shell están orientados principalmente a la **automatización** de tareas rutinarias, repetitivas, largas o complejas. Su objetivo principal es reducir la intervención manual y minimizar los errores humanos.
   Los scripts de shell son **interpretados**, no compilados. Esto significa que el intérprete de shell (por ejemplo, Bash) lee y ejecuta las instrucciones del script línea por línea en tiempo de ejecución, directamente desde el archivo de texto.
2. echo sirve para imprimir texto, mientras que read lee una línea desde entrada estándar en una variable.
   a. Se indican con #
   b. Bash soporta variables de tipo string y array, además son case-sensitive.
      `NOMBRE="pepe"`
3. ```
   #!/bin/bash
   # Comentarios acerca de lo que hace el script
   # Siempre comento mis scripts, si no lo hago hoy,
   # mañana ya no me acuerdo de lo que quise hacer
   echo "Introduzca su nombre y apellido:"
   read nombre apellido
   echo "Fecha y hora actual:"
   date
   echo "Su apellido y nombre es:"
   echo "$apellido $nombre"
   echo "Su usuario es: `whoami`"
   echo "Su directorio actual es: `pwd`"
   echo -e "y su contenido es: \n`ls`"
   echo "Espacio libre en disco: `df -h`"
   echo "Ahora introduzca su edad:"
   read edad
   echo "$nombre $apellido de edad $edad:"
   echo "Borrando directorio /home..."
   ```
4. Se accede a los parámetros (o argumentos) enviados a un script al momento de su invocación utilizando variables posicionales: $1, $2, $3,..., $n.

| Variable | Nombre | Información que Contiene |
|---|---|---|
| $# | Número de parámetros | El número total de argumentos posicionales que se pasaron al script (excluyendo el nombre del script, $0). Se usa para validar si se recibieron suficientes argumentos. |
   ```

| | | |
|---|---|---|
| $* | Todos los parámetros (como una sola cadena) | Todos los argumentos posicionales (a partir de $1) tratados como una única palabra/cadena. Si la variable se encierra entre comillas dobles ("$*"), se mantiene como una única cadena. |
| $@ | Todos los parámetros (como lista) | Todos los argumentos posicionales (a partir de $1) tratados como una lista separada de palabras. Si se encierra entre comillas dobles ("@$"), cada argumento se mantiene como una cadena separada (es el uso más común). |
| $? | Código de salida | El código de salida (o estado de finalización) del último comando ejecutado en primer plano. Un valor de 0 generalmente indica que el comando se ejecutó exitosamente. Cualquier valor distinto de cero (e.g., 1,2) indica un error. |
| $HOME | Directorio personal | Contiene la ruta absoluta del directorio de inicio (o directorio personal) del usuario que está ejecutando el script. Es una variable de entorno estándar. |
| "$*" | - | Se expande a una única palabra que contiene todos los argumentos separados por el primer carácter de la variable $IFS (Internal Field Separator, por defecto un espacio). |
| "$@" | - | Se expande a una única palabra que contiene todos los argumentos separados por el primer carácter de la variable $IFS (Internal Field Separator, por defecto un espacio). |

5. Para terminar un script usualmente se utiliza el comando exit:
    - Causa la terminación de un script.
    - Puede devolver cualquier valor entre 0 y 255:
        - El valor 0 indica que el script se ejecutó de forma exitosa
        - Un valor distinto indica un código de error
        - Se puede consultar el exit status imprimiendo la variable $?
6. El comando expr en Shell Scripting permite realizar operaciones aritméticas, relacionales (comparación) y lógicas, además de manipulación de cadenas de texto.

**Operaciones aritméticas:**

| Operador | Descripción |
|---|---|
| + | Suma |
| - | Resta |
| * | Multiplicación (debe escaparse con * o citarse: '*') |
| / | División |
| % | Módulo (resto de la división) |

**Operaciones relacionales:**

| Operador | Descripción | Resultado (Código de Salida) |
|---|---|---|
| = o == | Igual a | 1 (verdadero) o 0 (falso) |
| != | No igual a | 1 (verdadero) o 0 (falso) |
| > | Mayor que | 1 (verdadero) o 0 (falso) |
| >= | Mayor o igual que | 1 (verdadero) o 0 (falso) |
| < | Menor que | 1 (verdadero) o 0 (falso) |
| <= | Menor o igual que | 1 (verdadero) o 0 (falso) |

**Operaciones lógicas:**

| Operador | Descripción |
|---|---|
| \|\| | Se evalúa como verdadero si al menos una de las dos expresiones es verdadera. Es útil para ejecutar un comando alternativo si el primero falla. |
| & | Se evalúa como verdadero solo si ambas expresiones a su alrededor son verdaderas. Se utiliza para encadenar comandos que deben ejecutarse en secuencia, donde el segundo comando solo se ejecuta si el primero tiene éxito. |
| ! | Invierte el resultado de una condición. Si la condición es verdadera, ! la convierte en falsa, y viceversa. |

7. El comando test permite evaluar tres tipos principales de expresiones. Para archivos, puedes usar -d para saber si es un directorio, -f para si es un archivo regular, -r para si tiene permiso de lectura, -w para escritura, -x para ejecución, -s si el tamaño es mayor a cero y -e si el archivo existe. Para cadenas de caracteres, se usa = para igualdad, != para desigualdad, -z si la longitud es cero y -n si la longitud es no nula. Para evaluaciones numéricas, se utilizan operadores como -eq para igual, -ne para distinto, -gt para mayor que, -lt para menor que, -ge para mayor o igual y -le para menor o igual.
8. Las estructuras de control tienen sintaxis específicas:
    a. La sentencia if se estructura como: if [ condicion ]; then comandos; elif [ condicion ]; then comandos; else comandos; fi.
    b. La sentencia case es: case $variable in patron1) comandos ;; patron2) comandos ;; *) comandos_por_defecto ;; esac.
    c. La sentencia while es: while [ condicion ]; do comandos; done.

   d. La sentencia for tiene varias formas, una común es: for variable in lista; do comandos; done.
   e. La sentencia select es: select variable in lista; do comandos; done (se usa para menús interactivos).

9. Las sentencias break y continue modifican el flujo de los bucles. Break termina la ejecución del bucle actual inmediatamente y salta a la instrucción siguiente al done. Continue detiene la iteración actual y salta al inicio de la siguiente evaluación de la condición del bucle. Ambas pueden recibir un parámetro numérico opcional n, que indica cuántos niveles de bucles anidados deben romperse o continuarse (por defecto es 1).
10. En Shell Script existen variables locales, globales y de entorno. No es un lenguaje fuertemente tipado; por defecto, las variables se tratan como cadenas de caracteres, aunque se pueden evaluar aritméticamente en contextos específicos. Sí se pueden definir arreglos. Se definen encerrando los elementos entre paréntesis separados por espacios, por ejemplo: arreglo=(elemento1 elemento2 elemento3), o asignando índices individualmente como arreglo[0]=valor.
11. Se pueden definir funciones dentro de un script. La sintaxis básica es: nombre_funcion() { comandos; } o también function nombre_funcion { comandos; }. El pasaje de parámetros no se declara en la definición de la función (entre los paréntesis); en su lugar, se pasan argumentos al llamar a la función (igual que a un script) y dentro de la función se accede a ellos usando las variables especiales $1, $2, $3, etc., correspondientes al orden de los argumentos.
12.
    a.

```
echo "Introduzca el primer nro:"
read n1
echo "Introduzca el segundo nro:"
read n2
echo "Ahora se mostrarán las siguientes operaciones entre los dos números
ingresados:"
echo "Multiplicación: `expr $n1 \* $n2`"
echo "Suma: `expr $n1 + $n2`"
echo "Resta: `expr $n1 - $n2`"
mayor=$(($n1 > $n2 ? $n1 : $n2))
echo "Nro mayor: $mayor"
```

    b.

```
if [[ $# == 2 ]]; then
       echo "Se mostrarán las siguientes operaciones entre los dos números
ingresados por parámetro:"
       echo "Multiplicación: `expr $1 \* $2`"
       echo "Suma: `expr $1 + $2`"
       echo "Resta: `expr $1 - $2`"
       mayor=$(($1 > $2 ? $1 : $2))
       echo "Nro mayor: $mayor"
else
```

```
       echo "No se ingresó la cantidad de parámetros necesaria"
       echo "Cantidad necesaria: 2"
       echo "Se ingresaron: $#"; fi
```

**c.**

```
if [[ $# == 3 ]]; then
       if [[ $2 == "+" ]]; then
              echo $(( $1 + $3 ));
       elif [[ $2 == "-" ]]; then
              echo $(( $1 - $3 ));
       elif [[ $2 == "*" ]]; then
              echo $(( $1 * $3 ));
       elif [[ $2 == "%" ]]; then
              echo $(( $1 % $3 )); fi
else
       echo "No se ingresó la cantidad de parámetros necesaria"
       echo "Cantidad necesaria: 3"
       echo "Se ingresaron: $#"; fi
```

**13.**

**a.**

```
for (( i=1; i<=100; i++ )); do
       echo "Nro: $i"
       echo "Cuadrado: $(( $i * $i ))"
       done
```

**b.**

```
PS3="Seleccione una opción (1-4): "
select opcion in Listar DondeEstoy QuienEsta Salir
do
    case $opcion in
        Listar)
            echo "--- Listado del contenido (ls -l) ---"
            ls -l
            ;;
        DondeEstoy)
            echo "--- Ruta actual (pwd) ---"
            pwd
            ;;
        QuienEsta)
            echo "--- Usuarios conectados (who) ---"
            who
            ;;
        Salir)
            echo "Saliendo del menú."
            break
            ;;
        *)
            echo "Opción inválida: $REPLY. Por favor, ingrese el número de la
opción deseada."
            ;;
```

```
    esac
done
```

**c.**

```
if [[ $# != 1 ]]; then
    echo "Error: Debe ingresar exactamente un parámetro (nombre de
archivo/directorio)."
    echo "Uso: $0 <nombre>"
    exit 1
fi
NOMBRE="$1"
if [ -e "$NOMBRE" ]; then
    if [ -d "$NOMBRE" ]; then
        echo "El elemento '$NOMBRE' existe y es un directorio."
    elif [ -f "$NOMBRE" ]; then
        echo "El elemento '$NOMBRE' existe y es un archivo regular."
    else
        echo "El elemento '$NOMBRE' existe, pero no es un archivo regular ni un
directorio (ej. enlace simbólico)."
    fi
else
    echo "El elemento '$NOMBRE' no existe. Procediendo a crear un directorio."
    mkdir "$NOMBRE"
    if [[ $? != 0 ]]; then
        echo "Directorio '$NOMBRE' creado exitosamente."
    else
        echo "Error: No se pudo crear el directorio '$NOMBRE'."
    fi
fi
```

**14.**

```
if [[ $# != 3 ]]; then
       echo "Se necesitan exactamente tres parámetros";
       exit 1;
fi
OPCION=$1
DIR=$2
CADENA=$3
if [[ $OPCION == "-a" ]]; then
       for archivo in "$DIR"*; do
              mv "$archivo" "$archivo$CADENA"
       done
elif [[ $OPCION == "-b" ]]; then
       for full_path in "$DIR"*; do
              filename=$(basename "$full_path")
              directory=$(dirname "$full_path")
              mv "$full_path" "$directory/$CADENA$filename"
       done
fi;
```

**15.**

El comando `cut` permite extraer partes específicas de una línea de texto de tres maneras principales: por campos delimitados (`-f`), por posición de caracteres (`-c`) o por posición de bytes (`-b`).

**Parámetros Clave**

El parámetro más usado en `cut` es **`-f`** (fields), que especifica los campos que deseas extraer. Este parámetro generalmente se usa junto con **`-d`** (delimiter), que le indica al comando qué carácter está separando los campos. Si se omite `-d`, `cut` asume que el delimitador es el carácter de tabulación.

Los valores que puede recibir **`-f`** son: un número único (ej. `-f3`), un rango (ej. `-f1-5` para campos del 1 al 5), o una combinación de ambos (ej. `-f1,3,5-`).

El parámetro **`-c`** (characters) se usa para cortar la línea basándose en la posición del carácter, contando desde el inicio. Su sintaxis es idéntica a la de `-f`, pudiendo usar números únicos, rangos o combinaciones (ej. `-c1-10` para los primeros diez caracteres).

También existe el parámetro **`-b`** (bytes), que funciona de forma similar a `-c`, pero corta por posición de byte.

El modificador **`--complement`** invierte la selección, imprimiendo todo excepto los campos, caracteres o bytes especificados.

**Ejemplos de Uso**

**Ejemplo 1: Extraer Usuario y Shell de `/etc/passwd`**

El archivo `/etc/passwd` usa el carácter dos puntos (`:`) como delimitador. Para extraer el primer campo (nombre de usuario) y el último (shell), se utiliza:

```
cut -d: -f1,7 /etc/passwd
```

**Ejemplo 2: Extraer los primeros 10 caracteres de la salida de un comando**

Para obtener los diez primeros caracteres de cada línea que devuelve el comando `ls -l`:

```
ls -l | cut -c1-10
```

**Ejemplo 3: Extraer todo excepto una columna específica**

Para imprimir todos los campos del archivo `datos.csv` (delimitado por coma) excepto la columna 3, se usa:

```
cut -d, -f3 --complement datos.csv
```

**Ejemplo 4: Ignorar líneas sin el delimitador**

Para asegurarte de que solo se procesen las líneas que tienen el delimitador, se utiliza el parámetro **-s**:

```
cut -d, -f1 -s datos.csv
```

**16.**

```
if [[ $# != 1 ]]; then
        echo "Error: se requiere exactamente 1 parametro"
        exit 1
        fi

extension="$1"
salida="reporte.txt"
find /home -type f -name "*.$extension" -printf "%u\n" |
sort |
uniq -c |
awk '{print $2 "\t" $1}' > "$salida"

echo "Se ha guardado la informacion en $salida."
```

**17.**

```
for archivo in *; do
      if [[ -f "$archivo" ]]; then
            echo "$archivo" | tr '[:lower:][:upper:]' '[:upper:][:lower:]' | tr
-d 'aA'
      fi
done
```

**18.**

```
if [[ $# != 1 ]]; then
    echo "Debe ingresar el nombre del usuario a monitorear."
    exit 1
fi

USUARIO_A_BUSCAR="$1"
echo "Buscando al usuario '$USUARIO_A_BUSCAR'. Ctrl+C para detener."
while true; do
    who | grep -w "$USUARIO_A_BUSCAR" > /dev/null
    if [[ $? != 0 ]]; then
        echo "Usuario $USUARIO_A_BUSCAR logueado en el sistema."
        break
    fi
    echo "$(date +%H:%M:%S) - Usuario $USUARIO_A_BUSCAR no encontrado. Esperando
10 segundos..."
    sleep 10
done
```

**19.**

```
PS3="Ingrese el número de la opción a ejecutar (o '11' para Salir): "
```

```
echo "          MENÚ DE COMANDOS AMIGABLE CON EL USUARIO        "

select OPCION in "mostrar.sh" "Cálculo por entrada" "Cálculo por parámetros 1"
"Cálculo por parámetros 2" "Números del 1 al 100 con sus cuadrados" "Listar,
DondeEstoy y QuienEsta" "Renombrando archivos" "Reporte de extensiones"
"Intercambio y supresión de caracteres con tr" "Verificación con timer" "Salir"
do
    case $OPCION in
      "mostrar.sh")
           echo "-> Ejecutando mostrar.sh..."
           ./mostrar.sh
           ;;
      "Cálculo por entrada")
           echo "-> Ejecutando Cálculo por entrada..."
           ./12a.sh
           ;;
      "Cálculo por parámetros 1")
           echo "-> Ejecutando Cálculo por parámetros 1..."
           read -p "Ingrese el primer nro: " NRO1
           read -p "Ingrese el segundo nro: " NRO2
           ./12b.sh "$NRO1" "$NRO2"
           ;;
      "Cálculo por parámetros 2")
           echo "-> Ejecutando Cálculo por parámetros 2..."
           read -p "Ingrese el primer nro: " NRO1
           read -p "Ingrese la operación (+,-,*,%): " OP
           read -p "Ingrese el segundo nro: " NRO2
           ./12c.sh "$NRO1" "$OP" "$NRO2"
           ;;
      "Números del 1 al 100 con sus cuadrados")
          echo "-> Ejecutando Números del 1 al 100 con sus cuadrados..."
          ./13a.sh
          ;;
      "Listar, DondeEstoy y QuienEsta")
          echo "-> Ejecutando Listar, DondeEstoy y QuienEsta..."
          ./13b.sh
          ;;
      "Renombrando archivos")
          echo "-> Ejecutando Renombrando archivos..."
          read -p "Ingrese la opción: " OPCION
          read -p "Ingrese el directorio: " DIR
          read -p "Ingrese la cadena: " CADENA
          ./14.sh "$OPCION" "$DIR" "$CADENA"
          ;;
      "Reporte de extensiones")
          echo "-> Ejecutando Reporte de extensiones..."
          read -p "Ingrese la extensión: " EXTENSION
          ./16.sh "$EXTENSION"
          ;;
      "Intercambio y supresión de caracteres con tr")
          echo "-> Ejecutando Intercambio y supresión de caracteres con tr..."
```

```
            ./17.sh
            ;;
        "Verificación con timer")
            echo "-> Ejecutando Verificación con timer..."
            read -p "Ingrese el usuario: " USER
            ./18.sh "$USER"
            ;;
         "Salir")
             echo "Saliendo del Menú de Comandos."
             break
             ;;
         *)
             echo "Opción inválida. Por favor, ingrese el número asociado a una
de las opciones."
             ;;
    esac
done
```

**20.**

```
#!/bin/bash

PILA=()

push() {
    local elemento="$1"
    PILA+=("$elemento")
}

pop() {
    local longitud=${#PILA[@]}
    if [[ $longitud -eq 0 ]]; then
        echo "Error: Pila vacía."
        return 1
    fi

    local elemento_sacado=${PILA[longitud-1]}

    unset PILA[longitud-1]

    PILA=("${PILA[@]}")

    echo "POP: $elemento_sacado"
}

length() {
    echo "Largo: ${#PILA[@]}"
}

print() {
    local longitud=${#PILA[@]}
    if [[ $longitud -eq 0 ]]; then
```

```
        echo "Pila vacía."
        return
    fi

    echo "Elementos de la PILA:"
    for elemento in "${PILA[@]}"; do
        echo "$elemento"
    done
}


if [[ $# -ne 1 ]]; then
       echo "Debe ingresar un parámetro (primer elemento a agregar a la pila)"
       exit 1
fi

echo "Agregando 10 elementos..."
push "$1"
for i in {1..9}; do
    push "E$i"
done

echo "Sacando 3 elementos:"
pop
pop
pop

echo "Largo actual:"
length

echo "Elementos restantes:"
print
```

**21.**

```
productoria() {
       local aux=1;
       for n in "${num[@]}"; do
              echo "$n"
              aux=$(( aux*n ))
       done
       echo "Productoria de (${num[@]}) = $aux"
}
num=(10 3 5 7 9 3 5 4)
productoria
```

**22.**

```
analizarArreglo() {
       local cantImpares=0
       echo "Números pares:"
       for num in "${arreglo[@]}"; do
              if [[ $(( $num % 2 )) == 0 ]]; then
```

```
                echo "$num"
            else
                let cantImpares++
            fi
        done
        echo "Cantidad de números impares: $cantImpares"
}

arreglo=(2 3 5 7 11 13 17 19 23)
analizarArreglo
```

**23.**

```
sumarPosicionalmente() {
        if [[ ${#arreglo1[@]} != ${#arreglo2[@]} ]]; then
                echo "Los arreglos deben ser del mismo tamaño";
                return 1;
        fi
        TAMANIO=${#arreglo1[@]}
        for (( i = 0; i < TAMANIO; i++ )); do
                echo "La suma de los elementos de la posición $i de los vectores es
$(( arreglo1[i] + arreglo2[i] ))"
        done
}

arreglo1=(1 80 65 35 2)
arreglo2=(5 98 3 41 8)
sumarPosicionalmente
SALIDA=$?
if [[ $SALIDA != 0 ]]; then
        echo "Error de salida $SALIDA"
fi
```

**24.**

```
imprimirLongitudArreglo() {
        echo "Longitud del arreglo: ${#ARREGLO[@]}"
}
imprimirArreglo() {
        echo "Impresión del arreglo: (${ARREGLO[@]})"
}
retornarElementoEnN() {
        if [[ ${#ARREGLO[@]} < $2 ]]; then
                echo "Error: la longitud del arreglo es menor a N"
                return 1
        fi
        echo "Elemento en la posición $2: ${ARREGLO[$2]}"
}

GRUPO="users"
LISTA_USUARIOS_TEXTO=$(grep "^$GRUPO:" /etc/group | cut -d: -f4 | tr ',' ' ')
ARREGLO=( $LISTA_USUARIOS_TEXTO )
```

```
case $# in
	0) echo "Error: debe ingresar al menos un parámetro..."
		echo "- "-b n": Retorna el elemento de la posición n del arreglo si
el mismo existe"
		echo "- "-l": Devuelve la longitud del arreglo"
		echo "- "-i": Imprime todos los elementos del arreglo en pantalla"
		;;
	1) if [[ $1 == "-l" ]]; then
		imprimirLongitudArreglo
		elif [[ $1 == "-i" ]]; then
			imprimirArreglo
		else
			echo "Error: argumento desconocido"
		fi
		;;
	2) if [[ $1 == "-b" ]]; then
		retornarElementoEnN
		fi
		;;
	*) echo "Error: cantidad de argumentos incorrecta" ;;
esac
```

**25.**

```
CANT_PARAMETROS=$#
if [ $CANT_PARAMETROS -eq 0 ]; then
	echo "Se requiere al menos un parámetro"
	exit 1
fi
CANT_INEXISTENTES=0
CONTADOR=0
for RUTA in "$@"; do
	POS=$((CONTADOR + 1))
	if [ -e "$RUTA" ]; then
		if [ -d "$RUTA" ]; then
			echo "$RUTA corresponde a un DIRECTORIO"
		elif [ -f "$RUTA" ]; then
			echo "$RUTA corresponde a un ARCHIVO"
		fi
	else
		let CANT_INEXISTENTES++
	fi

	let CONTADOR++
done

echo "Cantidad de archivos/directorios inexistentes en el sistema:
$CANT_INEXISTENTES"
```

**26.**

```
inicializar() {
	ARRAY_VACIO=()
```

```
}

agregar_elem() {
	if [ $# -ne 1 ]; then
		echo "Error: 'agregar_elem' requiere un parámetro! (Elemento a
agregar)"
		return 1
	else
		ARRAY_VACIO+=("$1")
		echo "Agregado: '$1'"
	fi
}

eliminar_elem() {
	if [[ $# -ne 1 ]]; then
		echo "Error: 'eliminar_elem' requiere un parámetro! (La posición a
eliminar)."
		return 1
	fi

	local POSICION=$1
	local INDICE=$((POSICION - 1))

	if [[ $INDICE -lt 0 || $INDICE -ge ${#ARRAY_VACIO[@]} ]]; then
		echo "ERROR: Posición '$POSICION' inválida. Rango válido: 1 a
${#ARRAY_VACIO[@]}."
		return 1
	fi

	local elemento_eliminado="${ARRAY_VACIO[INDICE]}"
	unset ARRAY_VACIO[INDICE]
	ARRAY_VACIO=("${ARRAY_VACIO[@]}")

	echo "Eliminado: '$elemento_eliminado' de la posición $POSICION."
}

longitud() {
	echo "Longitud del array: ${#ARRAY_VACIO[@]}"
}

imprimir() {
	echo "Impresión del array: (${ARRAY_VACIO[@]})"
}

inicializar_Con_Valores() {
	if [[ $# -ne 2 ]]; then
		echo "Error: 'inicializar_Con_Valores' requiere dos parámetros! (La
longitud y el valor a rellenar)."
		return 1
	fi
```

```
        ARRAY_NUEVO=()
        for (( i=0; i<$1; i++ )); do
                ARRAY_NUEVO+=("$2")
        done

        echo "ARRAY_NUEVO inicializado con los siguientes valores:
(${ARRAY_NUEVO[@]})"
}

inicializar
agregar_elem "A"
agregar_elem "B"
agregar_elem "C"
eliminar_elem 2
longitud
imprimir
inicializar_Con_Valores 5 "Z"
```

**27.**

```
if [ $# -eq 0 ] || [ ! -d "$1" ]; then
        echo "Error: se requiere un parámetro que corresponda a un directorio."
        exit 4
fi

CANT=0
for archivo in "$1"/*; do
        if [ -f "$archivo" ]; then
                if [ -r "$archivo" ] && [ -w "$archivo" ]; then
                        let CANT++
                fi
        fi
done

echo "Cantidad de archivos para los cuales el usuario que ejecuta el script
tiene permiso de lectura y escritura: $CANT"
```

**28.**

```
agregarArchivos() {
        ARCHIVOS_ARRAY=()
        ARCHIVOS_ARRAY+=( $DIR$EXTENSION )
        if [[ ${ARCHIVOS_ARRAY[0]} == "$DIR$EXTENSION" ]]; then
                if [[ ! -e "$DIR$EXTENSION" ]]; then
                    ARCHIVOS_ARRAY=()
                    echo "Error: No se encontraron archivos $EXTENSION en $DIR."
                fi
        fi
}

verArchivo() {
        if [ $# -eq 0 ]; then
                echo "Debe ingresar un parámetro"
```

```
                return 1
        elif [ -e "$1" ]; then
                cat "$1"
        else
                echo "Error: archivo no encontrado"
                return 5
        fi
}

cantidadArchivos() {
        echo "Cantidad de archivos $EXTENSION en $DIR: ${#ARCHIVOS_ARRAY[@]}"
}

borrarArchivo() {
        if [ $# -eq 0 ]; then
                echo "Debe ingresar un parámetro"
                return 1
        else
                local POS=-1
                local ENCONTRADO=false
                for i in "${!ARCHIVOS_ARRAY[@]}"; do
                        if [ $ARCHIVOS_ARRAY[$i] == "$1" ]; then
                                let ENCONTRADO=true
                                let POS=$i
                        fi
                done
                if [ "$ENCONTRADO" == "false" ]; then
                        echo "Error: archivo no encontrado"
                        return 10
                fi

                read -p "¿Desea eliminar lógicamente el archivo $1? (S/N)" OP
                unset ARCHIVOS_ARRAY[$POS]
                if [ $OP == "N" ]; then
                        echo "Necesitamos su contraseña sudo para poder borrar en
$DIR"
                        sudo rm "$1"
                        echo "Archivo borrado del disco"
                fi
                echo "Archivo borrado del arreglo"
        fi
}

DIR="/home/*"
EXTENSION=".doc"

agregarArchivos
echo "${ARCHIVOS_ARRAY[@]}"
verArchivo "/home/aa.doc"
cantidadArchivos
borrarArchivo "/home/aa.doc"
```

**29.**

```
if [ ! -e "/home/bin" ]; then
	sudo mkdir "/home/bin"
fi

CANT_MOVIDOS=0
for archivo in *; do
	if [ -x "$archivo" ]; then
		echo "Moviendo el archivo '$archivo' a /home/bin..."
		sudo mv "$(readlink -f "$archivo")" "/home/bin"
		let CANT_MOVIDOS++
	fi
done

if [ $CANT_MOVIDOS -eq 0 ]; then
	echo "No se movió ningún archivo"
else
	echo "Cantidad de archivos movidos: $CANT_MOVIDOS"
fi
```

**30.**

## Set.sh

```
initialize() {
	SET=()
}

buscar_elemento() {
	if [ $# -ne 1 ]; then
		echo "Error: debe pasar exactamente 1 parámetro para buscar en el
Set"
		return 1
	fi
	for elem in "${SET[@]}"; do
		if [ "$elem" == "$1" ]; then
			return 0
		fi
	done
	return 1
}

initialize_with() {
	if [ $# -eq 0 ]; then
		echo "Error: debe pasar al menos 1 parámetro para inicializar el
Set"
		return 1
	fi

	for param in "$@"; do
		buscar_elemento "$param"
```

```
                RESULTADO=$?
                if [ $RESULTADO -eq 0 ]; then
                        echo "Error: el elemento $param ya existe en el conjunto"
                else
                        SET+=( "$param" )
                        echo "Se ha agregado el elemento '$param' al conjunto"
                fi
        done
}

add() {
        if [ $# -ne 1 ]; then
                echo "Error: debe pasar exactamente 1 parámetro para agregar al
Set"
                return 1
        fi

        buscar_elemento "$1"
        RESULTADO=$?
        if [ $RESULTADO -eq 0 ]; then
                echo "Error: el elemento $1 ya existe en el conjunto"
                return 1
        fi

        SET+=( "$1" )
        echo "Se ha agregado el elemento '$1' al conjunto"
}

remove() {
        if [ $# -eq 0 ]; then
                echo "Error: debe pasar al menos 1 parámetro para eliminar del Set"
                return 1
        fi

        local CANT_ELIMINADOS=0
        for param in "$@"; do
                local ENCONTRADO="false"
                local i=0
                while [ "$ENCONTRADO" == "false" ] && [ $i -lt ${#SET[@]} ]; do
                        if [ "$param" == "${SET[$i]}" ]; then
                                unset SET[$i]
                                echo "Se ha eliminado el elemento '$param'"
                                let ENCONTRADO="true"
                                let CANT_ELIMINADOS++
                        fi
                        let i++
                done
                if [ "$ENCONTRADO" == "false" ]; then
                        echo "Error: no se ha encontrado el elemento '$param'"
                fi
        done
```

```
        if [ $CANT_ELIMINADOS -eq 0 ]; then
                echo "Error: no se ha podido eliminar ningun elemento"
                return 1
        fi
}

contains() {
        if [ $# -ne 1 ]; then
                echo "Error: debe pasar exactamente 1 parámetro para buscar en el
Set"
                return 1
        fi

        buscar_elemento "$1"
        RESULTADO=$?
        if [ $RESULTADO -ne 0 ]; then
                echo "Error: el elemento $param no existe en el conjunto"
                return 1
        fi
        echo "El elemento $param ya existe en el conjunto"
}

print() {
        for elem in "${SET[@]}"; do
                echo "$elem"
        done
}

print_sorted() {
        echo "$(print | sort)"
}

initialize_with "a" "b" "b" "c" "a" "d" "z" "g" "h" "h" "5" "5" "3"
add
add "a"
add "j"
remove "a" "b" "k" "c"
echo "Impresión del Set:"
print
echo "Impresión ordenada del Set:"
print_sorted
```

## Bingo.sh

```
source ./Set.sh

if [ $# -gt 1 ]; then
        echo "Error: se puede pasar un parámetro como máximo (un valor máximo
para el rango de nros)"
```

```
        return 1
fi

VALOR_MAX=99
if [ $# -eq 1 ]; then
        let VALOR_MAX=$1
fi


initialize
echo "JUEGO DE BINGO INICIADO"

JUEGO_ACTIVO="true"
TOTAL_NUMEROS=$((VALOR_MAX + 1))
CANTIDAD_CANTADOS=0

while [ "$JUEGO_ACTIVO" == "true" ]; do
    if [ $CANTIDAD_CANTADOS -eq $TOTAL_NUMEROS ]; then
        echo "Se han cantado todos los números posibles ($TOTAL_NUMEROS)"
        echo "Finalizando partida..."
        break
    fi

    NUMERO_VALIDO="false"
    while [ "$NUMERO_VALIDO" == "false" ]; do
        RAND=$(( RANDOM % (VALOR_MAX + 1) ))

        buscar_elemento "$RAND"

        if [ $? -ne 0 ]; then
            NUMERO_VALIDO="true"
            NUMERO_A_CANTAR=$RAND
        fi
    done

    add "$NUMERO_A_CANTAR" > /dev/null
    let CANTIDAD_CANTADOS++

    echo "NÚMERO: $NUMERO_A_CANTAR"

    read -p "Presione ENTER para el siguiente número o escriba 'BINGO' para
terminar: " OPCION

    if [[ "$OPCION" == "BINGO" ]]; then
        echo "¡¡¡ B I N G O !!! Partida finalizada."
        JUEGO_ACTIVO="false"
    fi
done

echo "RESUMEN DE LA PARTIDA (Ordenados):"
print_sorted
```