Práctica 3 - ISO

1. ¿Qué es el Shell Scripting? ¿A qué tipos de tareas están orientados los scripts? ¿Los scripts deben compilarse? ¿Por qué?

Es la práctica de escribir secuencias de comandos o scripts que están destinados a ser ejecutados por un intérprete de comandos/Shell. Los scripts de Shell están compuestos de comandos que normalmente se ejecutarían desde una Shell de forma interactiva, pero se agrupan en un archivo para automatizar tareas o secuencias de tareas más complejas.

Tipos de tareas a las que están orientados los Scripts:

- Automatización de Tareas del Sistema, de Desarrollo, Tareas Repetitivas, de Seguridad.
- Procesamiento de Archivos y Datos.
- Gestión de Usuarios y Permisos.
- Interacción con Redes y Servidores, Bases de Datos, APIs y Servicios Web.
- Generación de Informes y Logs.
- Configuración de Ambientes de Desarrollo.

Los scripts no deben compilarse, estos son interpretados por la Shell en tiempo de ejecución, es decir, el código es leído línea por línea y ejecutado de inmediato por la Shell, las razones por las cuales no necesitan ser compilados son:

- Interpretación en tiempo real: El Shell los interpreta en tiempo real permitiendo una rápida iteración y prueba de código.
- Portabilidad: Los scripts suelen ser independientes del hardware y el sistema operativo ya que el Shell se encarga de adaptarlos a cada entorno.
- Menor sobrecarga de Desarrollo: Al no ser necesaria la compilación y enlazamiento del código antes de ejecutarlo, el proceso de desarrollo y prueba se vuelve más ágil.

2. Investigar la funcionalidad de los comandos echo y read.

"echo": Su función principal es imprimir texto o variables en la salida estándar del sistema. Parámetros:

- "-n": Evita que se añada un salto de línea automático al final del texto.
- "-e": Habilita la interpretación de escape o saltos de línea usando "\n (salto de línea) ó \t (tabulación)".
- "-E": Deshabilita la interpretación de escape o saltos de línea.

"read": Se utiliza para leer entradas del usuario desde la línea de comandos directamente desde el teclado mientras se ejecuta un script, la entrada se almacena en una variable. Parámetros:

- "-p [prompt]": Muestra un mensaje antes de solicitar la entrada.
- "-s": Habilita el modo silencioso, es decir, la entrada del usuario no se va a mostrar en la pantalla.
- "-[n]": Especifica el número máximo de caracteres, ese número se tiene que especificar en n.
- "-t [n]": Especifica un tiempo límite de espera en segundos para ingresar la entrada. Se debe especificar la cantidad de segundos en n.

(a) ¿Como se indican los comentarios dentro de un script?

- Comentarios de una sola línea: #Esto es un comentario
- Comentarios de varias líneas: <<COMMENT
  Esto es un comentario de varias líneas
  COMMENT

(b) ¿Cómo se declaran y se hace referencia a variables dentro de un script?

Formas de nombrar Variables:

- Deben comenzar con una letra o un guión bajo.
- Pueden contener letras, números y guiones bajos.
- Los nombres de las variables son case sensitive.

Declaración: para la declaración se utilizan las normas de nombramiento anteriores y la forma de darles valor es la siguiente: mi_variable=valor. No se deben dejar espacios entre el signo "=".

Referencia: para hacer referencia a una variable y obtener su valor se debe utilizar el símbolo "$" seguido del nombre de la variable, por ejemplo, $mi_variable.

3. Crear dentro del directorio personal del usuario logueado un directorio llamado practicashell-script y dentro de él un archivo llamado mostrar.sh cuyo contenido sea el siguiente:

```
#!/bin/bash
# Comentarios acerca de lo que hace el script
# Siempre comento mis scripts, si no hoy lo hago
# y 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:"
```

(a) Asignar al archivo creado los permisos necesarios de manera que pueda ejecutarlo

(b) Ejecutar el archivo creado de la siguiente manera: ./mostrar

(c) ¿Qué resultado visualiza?

```
root@c04685c6d964:/ISO/Practica-3# mkdir practica-shell-script
root@c04685c6d964:/ISO/Practica-3# cd practica-shell-script
root@c04685c6d964:/ISO/Practica-3/practica-shell-script# ls -la
total 0
drwxr-xr-x 1 root root 512 Sep 22 18:31 .
drwxrwxrwx 1 root root 512 Sep 22 18:31 ..
root@c04685c6d964:/ISO/Practica-3/practica-shell-script# touch mostrar.sh
root@c04685c6d964:/ISO/Practica-3/practica-shell-script# ./mostrar.sh
bash: ./mostrar.sh: Permission denied
root@c04685c6d964:/ISO/Practica-3/practica-shell-script# chmod 766 mostrar.sh
root@c04685c6d964:/ISO/Practica-3/practica-shell-script# ./mostrar.sh
Introduzca su nombre y apellido:
Matias Guaymas
Fecha y hora actual:
Sun Sep 22 18:32:40 UTC 2024
./mostrar.sh: line 12: unexpected EOF while looking for matching `"'
root@c04685c6d964:/ISO/Practica-3/practica-shell-script# ./mostrar.sh
Introduzca su nombre y apellido:
Matias Guaymas
Fecha y hora actual:
Sun Sep 22 18:33:53 UTC 2024
Su apellido y nombre es:
Guaymas Matias
Su usuario es: root
Su directorio actual es:
/ISO/Practica-3/practica-shell-script
root@c04685c6d964:/ISO/Practica-3/practica-shell-script#
```

(d) Las backquotes (`) entre el comando whoami ilustran el uso de la sustitución de comandos. ¿Qué significa esto?

En el contexto del comando "whoami" las backquotes se utilizan para ejecutar un comando dentro de otro comando y luego sustituir la salida del comando interno en el comando externo.

(e) Realizar modificaciones al script anteriormente creado de manera de poder mostrar distintos resultados (cuál es su directorio personal, el contenido de un directorio en particular, el espacio libre en disco, etc.). Pida que se introduzcan por teclado (entrada estándar) otros datos.

4. Parametrización: ¿Cómo se acceden a los parámetros enviados al script al momento de su invocación? ¿Qué información contienen las variables $#, $, $? Y $HOME dentro de un script?*

Cuando se acceden a los parámetros enviados al script al momento de su invocación se hace a través de variables especiales, estas suelen estar predefinidas y contienen la información de los argumentos proporcionados al script, estos parámetros se almacenan en las variables "$1, $2, $3, …" al ejecutar el script con argumentos se debe hacer de la siguiente manera "./script.sh parámetro_1 parámetro_2".

Variables:

- "$#": Contiene el número de argumentos que se pasaron al script.
- "$*": Representa todos los argumentos en una sola cadena de texto, esta variable no preserva los espacios en blanco entre los argumentos.
- "$?": Contiene el código de salida del último comando ejecutado. Si el comando se ejecuta con éxito, "$? = 0". Si hay algún error, será un valor diferente de 0.
- "$HOME": Contiene la ruta al directorio raíz del usuario actual.

5. ¿Cuál es la funcionalidad de comando exit? ¿Qué valores recibe como parámetro y cuál es su significado?

Para terminar un script usualmente se utiliza la función exit. Recibe como parámetro un valor entre 0 y 255. Si no tiene argumentos, el código de salida predeterminado es el 0. El valor 0 indica que el script se ejecutó de forma exitosa y un valor distinto indica un código.

6. El comando expr permite la evaluación de expresiones. Su sintaxis es: expr arg1 op arg2, donde arg1 y arg2 representan argumentos y op la operación de la expresión. Investigar qué tipo de operaciones se pueden utilizar.

"expr": Herramienta de Shell que permite la avaluación de expresiones en scripts de Shell. Las expresiones pueden incluir:

Operadores Aritméticos:

- Suma (+): Suma dos números "expr 5 + 3".
- Resta (-): Resta dos números "expr 8 – 2".
- Multiplicación (*): Multiplica dos números "expr 4 * 3"
- División (/): Divide un número por otro "expr 10 / 2".
- Módulo (%): Devuelve el resto de la división de dos números "expr 7 % 3".

Operadores de Comparación:

- Igual (=): Compara si dos cadenas son iguales "expr "hola" = "hola"".
- Diferente (!=): Compara si dos cadenas son diferentes "expr "hola" != "chau"".

Operadores Lógicos:

- AND (&&): Operación lógica "y"; "expr 1 && 0".
- OR (||): Operación lógica "o"; "expr 1 || 0".

Operadores Relacionales:

- Menor que (<): Compara si el primer argumento es menor que el segundo "expr 5 < 10".

- Mayor que (>): Compara si el primer argumento es mayor que el segundo “expr 8 > 3”.
- Menor o igual que (<=): Compara si el primer argumento es menor o igual que el segundo “expr 5 <= 5”.
- Mayor o igual que (>=): Compara si el primer argumento es mayor o igual al segundo “expr 10 >= 8”.

Otros operadores:

- Longitud de Cadena (length): Devuelve la longitud de una cadena “expr length hola”

7. El comando “test expresión” permite evaluar expresiones y generar un valor de retorno, true o false. Este comando puede ser reemplazado por el uso de corchetes de la siguiente manera [expresión]. Investigar qué tipo de expresiones pueden ser usadas con el comando test. Tenga en cuenta operaciones para: evaluación de archivos, evaluación de cadenas de caracteres y evaluaciones numéricas.

“test”: Herramienta utilizada para evaluar expresiones y devolver un valor de éxito o fracaso (true o false). Normalmente se utiliza como [expresión].

Evaluación de Archivos:

- “-e archivo”: Comprueba si el archivo existe.
- “-f archivo”: Comprueba si es un archivo regular y no un directorio o dispositivo.
- “-d directorio”: Comprueba si el directorio existe.
- “-s archivo”: Comprueba si el archivo tiene un tamaño mayor que cero.
- “-r archivo”: Comprueba si el archivo tiene permisos de lectura.
- “-w archivo”: Comprueba si el archivo tiene permisos de escritura.
- “-x archivo”: Comprueba si el archivo tiene permisos de ejecución.

Evaluación de Cadenas de Caracteres:

- “-z cadena”: Comprueba si la cadena es vacía (length = 0).
- “-n cadena”: Comprueba si la cadena no es vacía (length > 0).
- “cadena1 = cadena2”: Comprueba si las cadenas son iguales.
- “cadena1 != cadena2”: Comprueba si las cadenas son distintas.

Evaluaciones Numéricas:

- “num1 –eq num2”: Comprueba si los números son iguales.
- “num1 –ne num2”: Comprueba si los números son distintos.
- “num1 –lt num2”: Comprueba si el primer número es menor que el segundo.
- “num1 –le num2”: Comprueba si el primer número es menor igual que el segundo.
- “num1 –gt num2”: Comprueba si el primer número es mayor que el segundo.
- “num1 –ge num2”: Comprueba si el primer número es mayor igual que el segundo.

Otros operadores:

- “! [Expresión]”: Negación.
- “expresión1 –a expresión2”: AND lógico.
- “expresión1 -0 expresión2”: OR lógico

8. Estructuras de control. Investigue la sintaxis de las siguientes estructuras de control incluidas en shell scripting:

```
if [[ condición ]]; then
    # Código a ejecutar si la condición es verdadera
elif [[ otra_condición ]]; then
    # Código a ejecutar si otra_condición es verdadera
else
    # Código a ejecutar si ninguna condición es verdadera
fi
```

```
case $variable in
    valor1)
        # Código a ejecutar si variable coincide con valor1
        ;;
    valor2)
        # Código a ejecutar si variable coincide con valor2
        ;;
    *)
        # Código a ejecutar si no se cumple ninguna de las anteriores
        ;;
esac
```

```
select opcion in opcion1 opcion2; do
    case $opcion in
        opcion1)
            # Código a ejecutar para la opción 1
            ;;
        opcion2)
            # Código a ejecutar para la opción 2
            ;;
        *)
            # Código a ejecutar si se selecciona una opción no válida
            ;;
    esac
done
```

```
while [[ condición ]]; do
    # Código a ejecutar mientras la condición sea verdadera
done
```

```
until [[ condición ]]; do
    # Código a ejecutar mientras la condición NO sea verdadera
done
```

```bash
# 1. for con una lista de elementos
for variable in ${arreglo[@]}; do
    # Código a ejecutar para cada elemento en la lista
done

# 2. For con una secuencia numérica
for ((i=1; i<=5; i++)); do
    # Código a ejecutar para cada valor de i en el rango 1-5
done

# 3. for con una expansión de llaves (brace expansion):
for letra in {a..z}; do
    # Código a ejecutar para cada letra en el rango de a a z
done

# 4. for con la expansión de archivos (globbing):
for archivo in *.txt; do
    # Código a ejecutar para cada archivo que coincida con *.txt
done

# 5. for con el resultado de un comando:
for elemento in $(comando); do
    # Código a ejecutar para cada elemento generado por el comando
done

# 6. for con un array
for elemento in "${mi_array[@]}"; do
    # Código a ejecutar para cada elemento en el array
done

# 7. For en un rango numérico usando seq
for i in $(seq 1 5); do
    # Código a ejecutar para cada valor en el rango 1-5
done
```

9. ¿Qué acciones realizan las sentencias break y continue dentro de un bucle? ¿Qué parámetros reciben?

break termina el bucle actual y pasa el control del programa al comando que sigue al bucle terminado. Se utiliza para salir de un bucle for, while, until o select.

break n;

n es un argumento opcional y debe ser mayor o igual a 1. break 1 es equivalente a break.

continue omite los comandos restantes dentro del cuerpo del bucle que lo encierra para la iteración actual y pasa el control del programa a la siguiente iteración del bucle.

continue n;

n es opcional y puede ser mayor o igual a 1. Cuando se da n, se reanuda el bucle n-ésimo. continue 1 es equivalente a continue.

10. ¿Qué tipo de variables existen? ¿Es shell script fuertemente tipado? ¿Se pueden definir arreglos? ¿Cómo?

Bash soporta strings y arrays. Bash es un lenguaje de tipado débil y dinámico. No es necesario declarar explícitamente el tipo de una variable y las variables pueden cambiar de tipo en tiempo de ejecución.

Los nombres de las variables son case-sensitive y pueden contener mayúsculas, minúsculas, numeros y el símbolo “_”, pero no pueden empezar con un número.

Sobre los arreglos:

- Creación:

```
arreglo_a=() # Se crea vacío
arreglo_b=(1 2 3 5 8 13 21) # Inicializado
```

- Asignación de un valor en una posición concreta:

```
arreglo_b[2]=spam
```

- Acceso a un valor del arreglo (En este caso las llaves no son opcionales):

```
echo ${arreglo_b[2]}
copia=${arreglo_b[2]}
```

- Acceso a todos los valores del arreglo:

```
echo ${arreglo[@]} # o bien ${arreglo[*]}
```

- Tamaño del arreglo:

```
${#arreglo[@]} # o bien ${#arreglo[*]}
```

- Borrado de un elemento (reduce el tamaño del arreglo pero no elimina la posición, solamente la deja vacía):

```
unset arreglo[2]
```

- Los índices en los arreglos comienzan en 0

11. ¿Pueden definirse funciones dentro de un script? ¿Cómo? ¿Cómo se maneja el pasaje de parámetros de una función a la otra?

Definiendo funciones

```
unaFuncion() {
    echo "Hola, $1!"
}

# Sintaxis alternativa 1
function alternativa1() {
    echo "Hola, $1!"
}

#  Sintaxis alternativa 2
function alternativa2 {
    echo "Hola, $1!"
}
```

```
unaFuncion "Mate"
alternativa1 "Mati"
alternativa2 "Chino"
```

return - levantando errores

return finaliza la ejecución de una función y puede retornar un valor entre 0 y 225 (por defecto devuelve 0). El valor 0 se considera como un valor de retorno exitoso.

```
myfunc() {
  return 1
}
```

```
if myfunc; then
  echo "success"
else
  echo "failure"
fi
```

Devolviendo valores

```
myfunc() {
    local myresult='some value'
    echo "$myresult"
}
```

```
result=$(myfunc)
```

12. Evaluación de expresiones:

(a) Realizar un script que le solicite al usuario 2 números, los lea de la entrada Standard e imprima la multiplicación, suma, resta y cuál es el mayor de los números leídos.

(b) Modificar el script creado en el inciso anterior para que los números sean recibidos como parámetros. El script debe controlar que los dos parámetros sean enviados.

(c) Realizar una calculadora que ejecute las 4 operaciones básicas: +, - ,*, %. Esta calculadora debe funcionar recibiendo la operación y los números como parámetros

13. Uso de las estructuras de control:

(a) Realizar un script que visualice por pantalla los números del 1 al 100 así como sus cuadrados.

(b) Crear un script que muestre 3 opciones al usuario: Listar, DondeEstoy y QuienEsta. Según la opción elegida se le debe mostrar:

- Listar: lista el contenido del directoria actual.
- DondeEstoy: muestra el directorio donde me encuentro ubicado.
- QuienEsta: muestra los usuarios conectados al sistema.

(c) Crear un script que reciba como parámetro el nombre de un archivo e informe si el mismo existe o no, y en caso afirmativo indique si es un directorio o un archivo. En caso de que no exista el archivo/directorio cree un directorio con el nombre recibido como parámetro.

14. Renombrando Archivos: haga un script que renombre solo archivos de un directorio pasado como parámetro agregándole una CADENA, contemplando las opciones:

"-a CADENA": renombra el fichero concatenando CADENA al final del nombre del archivo

"-b CADENA": renombra el fichero concantenado CADENA al principio del nombre del archivo

Ejemplo:

Si tengo los siguientes archivos: /tmp/a /tmp/b

Al ejecutar: ./renombra /tmp/ -a EJ

Obtendré como resultado: /tmp/aEJ /tmp/bEJ

Y si ejecuto: ./renombra /tmp/ -b EJ

El resultado será: /tmp/EJa /tmp/EJb

15. Comando cut. El comando cut nos permite procesar las líneas de la entrada que reciba (archivo, entrada estándar, resultado de otro comando, etc) y cortar columnas o campos, siendo posible indicar cual es el delimitador de las mismas. Investigue los parámetros que puede recibir este comando y cite ejemplos de uso.

Parámetros:

- "-d [delimitador]": Especifica el delimitador, por defecto, el tabulador.
- "-f [num_campo] [archivo]": Especifica los campos que se quieren extraer.
- "-c [rango] [archivo]": Extraer un rango de caracteres específicos de una línea de texto en lugar de columnas basadas en un delimitador.
- "-b [rango] [archivo]": Trabaja igual que el parámetro "-c" pero en vez de caracteres usa bytes.

16. Realizar un script que reciba como parámetro una extensión y haga un reporte con 2 columnas, el nombre de usuario y la cantidad de archivos que posee con esa extensión. Se debe guardar el resultado en un archivo llamado reporte.txt

17. Escribir un script que al ejecutarse imprima en pantalla los nombres de los archivos que se encuentran en el directorio actual, intercambiando minúsculas por mayúsculas, además de eliminar la letra a (mayúscula o minúscula). Ejemplo, directorio actual:

IsO

pepE

Maria

Si ejecuto: ./ejercicio17

Obtendré como resultado:

iSo

PEPe

mRI

Ayuda: Investigar el comando tr

18. Crear un script que verifique cada 10 segundos si un usuario se ha loqueado en el sistema (el nombre del usuario será pasado por parámetro). Cuando el usuario finalmente se loguee, el programa deberá mostrar el mensaje "Usuario XXX logueado en el sistema" y salir.

19. Escribir un Programa de "Menu de Comandos Amigable con el Usuario" llamado menu, el cual, al ser invocado, mostrará un menú con la selección para cada uno de los scripts creados en esta práctica. Las instrucciones de cómo proceder deben mostrarse junto con el menú. El menú deberá iniciarse y permanecer activo hasta que se seleccione Salir. Por ejemplo:

MENU DE COMANDOS

03. Ejercicio 3

12. Evaluar Expresiones

13. Probar estructuras de control

...

Ingrese la opción a ejecutar: 03

20. Realice un script que simule el comportamiento de una estructura de PILA e implemente las siguientes funciones aplicables sobre una estructura global definida en el script:

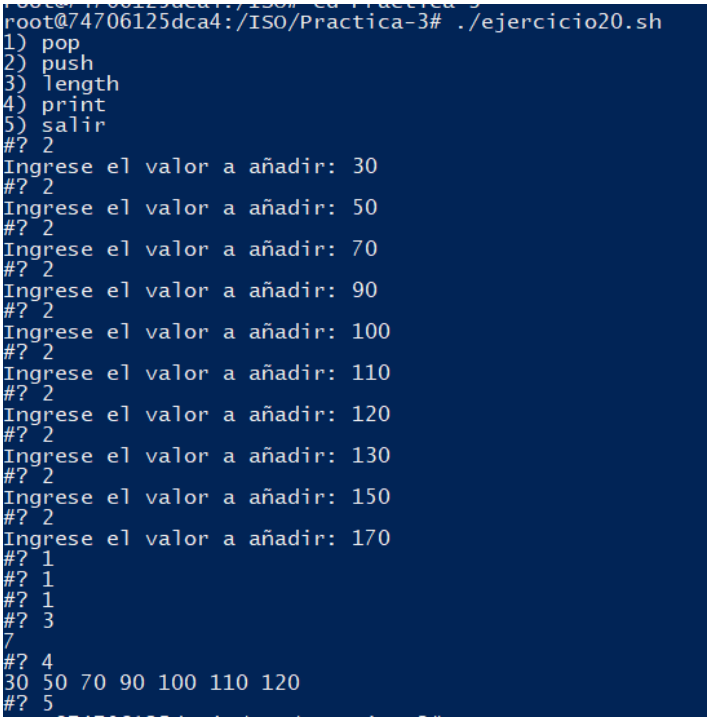
push: Recibe un parámetro y lo agrega en la pila pop: Saca un elemento de la pila

length: Devuelve la longitud de la pila print: Imprime todos elementos de la pila

21. Dentro del mismo script y utilizando las funciones implementadas:

- Agregue 10 elementos a la pila
- Saque 3 de ellos
- Imprima la longitud de la cola

- Luego imprima la totalidad de los elementos que en ella se encuentran.



22. Dada la siguiente declaración al comienzo de un script: num=(10 3 5 7 9 3 5 4) (la cantidad de elementos del arreglo puede variar). Implemente la función productoria dentro de este script, cuya tarea sea multiplicar todos los números del arreglo

23. Implemente un script que recorra un arreglo compuesto por números e imprima en pantalla sólo los números pares y que cuente sólo los números impares y los informe en pantalla al finalizar el recorrido.

24. Dada la definición de 2 vectores del mismo tamaño y cuyas longitudes no se conocen.

vector1=(1 .. N)

vector2=(7 .. N)

Por ejemplo:

vector1=(1 80 65 35 2)

y

vector2=(5 98 3 41 8).

Complete este script de manera tal de implementar la suma elemento a elemento entre ambos vectores y que la misma sea impresa en pantalla de la siguiente manera:

La suma de los elementos de la posición 0 de los vectores es 6

La suma de los elementos de la posición 1 de los vectores es 178

...

La suma de los elementos de la posición 4 de los vectores es 10

25. Realice un script que agregue en un arreglo todos los nombres de los usuarios del sistema pertenecientes al grupo “users”. Adicionalmente el script puede recibir como parametro:

- *"-b n": Retorna el elemento de la posición n del arreglo si el mismo existe. Caso contrario, un mensaje de error.*
- *"-l": Devuelve la longitud del arreglo*
- *"-i": Imprime todos los elementos del arreglo en pantalla*

26. Escriba un script que reciba una cantidad desconocida de parámetros al momento de su invocación (debe validar que al menos se reciba uno). Cada parámetro representa la ruta absoluta de un archivo o directorio en el sistema. El script deberá iterar por todos los parámetros recibidos, y solo para aquellos parámetros que se encuentren en posiciones impares (el primero, el tercero, el quinto, etc.) verificar si el archivo o directorio existen en el sistema, imprimiendo en pantalla que tipo de objeto es (archivo o directorio). Además, deberá informar la cantidad de archivos o directorios inexistentes en el sistema.

27. Realice un script que implemente a través de la utilización de funciones las operaciones básicas sobre arreglos:

- *inicializar: Crea un arreglo llamado array vacío*
- *agregar_elem <parametro1>: Agrega al final del arreglo el parámetro recibido*
- *eliminar_elem <parametro1>: Elimina del arreglo el elemento que se encuentra en la*
- *posición recibida como parámetro. Debe validar que se reciba una posición válida*
- *longitud: Imprime la longitud del arreglo en pantalla*
- *imprimir: Imprime todos los elementos del arreglo en pantalla*
- *inicializar_Con_Valores <parametro1><parametro2>: Crea un arreglo con longitud <parametro1>y en todas las posiciones asigna el valor <parametro2>*

28. Realice un script que reciba como parámetro el nombre de un directorio. Deberá validar que el mismo exista y de no existir causar la terminación del script con código de error 4. Si el directorio existe deberá contar por separado la cantidad de archivos que en él se encuentran para los cuales el usuario que ejecuta el script tiene permiso de lectura y escritura, e informar dichos valores en pantalla. En caso de encontrar subdirectorios, no deberán procesarse, y tampoco deberán ser tenidos en cuenta para la suma a informar.

29. Implemente un script que agregue a un arreglo todos los archivos del directorio /home cuya terminación sea .doc. Adicionalmente, implemente las siguientes funciones que le permitan acceder a la estructura creada:

- *verArchivo <nombre_de_archivo>: Imprime el archivo en pantalla si el mismo se encuentra en el arreglo. Caso contrario imprime el mensaje de error "Archivo no encontrado" y devuelve como valor de retorno 5*
- *cantidadArchivos: Imprime la cantidad de archivos del /home con terminación .doc*
- *borrarArchivo <nombre_de_archivo>: Consulta al usuario si quiere eliminar el archivo lógicamente. Si el usuario responde Si, elimina el elemento solo del arreglo. Si el usuario responde No, elimina el archivo del arreglo y también del FileSystem. Debe validar que el archivo exista en el arreglo. En caso de no existir, imprime el mensaje de error "Archivo no encontrado" y devuelve como valor de retorno 10*

30. Realice un script que mueva todos los programas del directorio actual (archivos ejecutables) hacia el subdirectorio "bin" del directorio HOME del usuario actualmente logueado. El script debe imprimir en pantalla los nombres de los que mueve, e indicar cuántos ha movido, o que no ha movido ninguno. Si el directorio "bin" no existe,deberá ser creado