Practica 3 (Shell Scriping) - Resolucion

Trabajo Práctico 3.pdf

---

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

    -Son secuencias de comandos destinados a ejecutarse en un Shell. Estan orientados en automatizar tareas o realizar varias tareas diviendolas en subtaeras mas sencillas. No debn compilarse porque son interpretados en tiempo de ejecucion por el interprete de comandos

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

-echo → imprime una cadena de texto o variables en la salida estandar

-read → lee de teclado las entradas delu suario hasta que se encuentra un salto de linea

a. ¿Cómo se indican los comentarios dentro de un script?

    -Comentario simple # ; Comentario de varias lineas <<COMMENT

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

```
NOMBRE="pepe" # SIN ESPACIOS AL REDEDOR DEL =
echo $NOMBRE # → pepe
echo ${NOMBRE}todo_esto_no_es_parte # CONCATENA

arreglo_a=() # ARRAY VACIO
```

```
arreglo_b=(1 2 3 4) # INICIALIZADO
arreglo_b[2]=spam
# ARRANCA EN 1, IMPRIME SPAM
echo ${arreglo2[2]}
# DEVUELVE TODOS LOS ELEMENTOS DEL ARREGLO CON '@' o '*'
echo ${arreglo[@]}
# TAMAÑO DEL ARREGLO
echo ${#arreglo[@]}
# BORRA CONTENIDO DE LA POS 2, QUEDA VACIA
unset arreglo[2]

# GUARDA EN MIS ARCHIVOS EL CONTENIDO DEL HOME
# DEL USUARIO WHOAMI
mis_archivos = "$(ls/home/$(whoami))"
```

3. Crear dentro del directorio personal del usuario logueado un directorio llamado *practica-shell-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 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:"

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

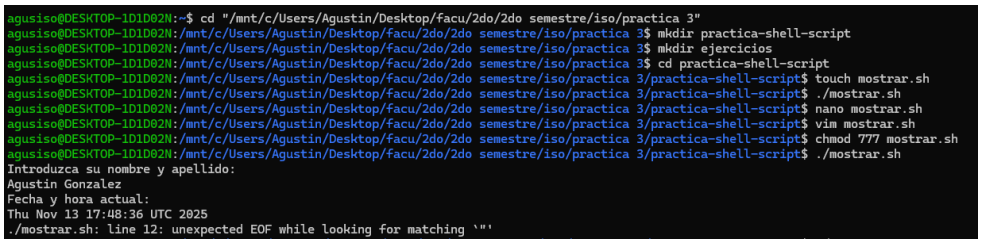
b. Ejecutar el archivo creado de la siguiente manera: ./mostrar.sh

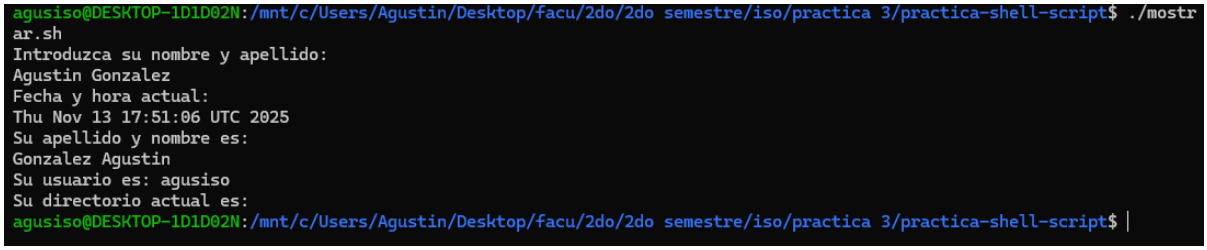
c. ¿Qué resultado visualiza?

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

-Se utilizan para ejecutar un comando dentro de otro. En este caso ejecuta el comando que muestra el nombre del usuario y sustituye 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?

-Se acceden a traves de variables, las cuales estan predefinidas y contienen la informacion de los argumentos proporcionados al script. Los parametros se almacenan en variables $1, $2, ... $N.

-Cuando ejecutas el script con X cantidad de parametros le pasas los parametros justo despues del llamado

→ ./script.sh param1 param2

-$# → Contiene numero de argumentos que se pasaron al script (en el caso de arriba seria 2)

-$* → Representa todos los argumentos en una cadena de texto sin espacios en blanco entre ellos

-$? → Contiene el codigo de salida del ultimo comando ejecutado. En caso de exito → 0, en caso de fallo → <> 0

-$HOME → Ruta al directorio raiz del usuario actual

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

-Es una funcion que se utiliza para terminar un script. Recibe como parametro un valor entre 0 y 255 y si significado es el siguiente:

-Si el script termino de manera exitosa → Exit = 0

-Cualquier otro tipo de error o estado particular→ Exit <> 0

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 que tipo de operaciones se pueden utilizar.

-Evalua la expresion y mostrar el resultado en la salida estandar

-Tipos:

-Aritmeticas:

Operación	Ejemplo	Resultado
Suma	`expr 5 + 3`	8
Resta	`expr 10 - 4`	6
Multiplicación	`expr 4 \* 2`	8
División	`expr 9 / 3`	3
Módulo (resto)	`expr 10 % 3`	1

-Comparacion:

Operador	Ejemplo	Resultado
=	expr 5 = 5	1
!=	expr 5 != 3	1
>	expr 6 \> 3	1
<	expr 2 \< 5	1
>=	expr 5 \>= 5	1
<=	expr 4 \<= 2	0

-Logicas:

Operador	Significado	Ejemplo	Resultado
`	`	OR lógico	expr 0 | 1
&	AND lógico	expr 1 \& 0	0

-Con cadenas:

Operación	Descripción	Ejemplo	Resultado
length	Longitud de una cadena	expr length "hola"	4
index	Posición del primer carácter de una subcadena	expr index "ubuntu" "b"	2
substr	Extrae una subcadena	expr substr "sistemas" 3 4	stem

7. El comando *testexpresion* 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.

-Evaluacion de archivos:

Expresión	Significado	Ejemplo
-e archivo	Existe	[-e /home/user/archivo.txt]
-f archivo	Es un archivo regular	[-f archivo.txt]

Expresión	Significado	Ejemplo
`-d directorio`	Es un directorio	`[ -d /home/user/docs ]`
`-r archivo`	Tiene permiso de lectura	`[ -r archivo.txt ]`
`-w archivo`	Tiene permiso de escritura	`[ -w archivo.txt ]`
`-x archivo`	Tiene permiso de ejecución	`[ -x script.sh ]`
`-s archivo`	El archivo no está vacío	`[ -s archivo.txt ]`

-Evaluacion de cadenas de carecteres:

Expresión	Significado	Ejemplo
`-z cadena`	La cadena está vacía	`[ -z "$var" ]`
`-n cadena`	La cadena **no** está vacía	`[ -n "$var" ]`
`cadena1 = cadena2`	Son iguales	`[ "$a" = "$b" ]`
`cadena1 != cadena2`	Son distintas	`[ "$a" != "$b" ]`

-Evaluacion numerica:

Operador	Significado	Ejemplo
`-eq`	Igual	`[ $a -eq $b ]`
`-ne`	Distinto	`[ $a -ne $b ]`
`-gt`	Mayor que	`[ $a -gt $b ]`
`-ge`	Mayor o igual	`[ $a -ge $b ]`
`-lt`	Menor que	`[ $a -lt $b ]`
`-le`	Menor o igual	`[ $a -le $b ]`

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

➢ if:

```
if [ condition ]
then
  bloque
elif [ condition ]
then
  bloque
else
  bloque
fi
```

➢ case:

Selección:

```
case $variable in
"valor 1")
  bloque
  ;;
"valor 2")
  bloque
  ;;
*)
  bloque
  ;;
esac
```

➢ while:

while

```
while [ condicion ] # Mientras se cumpla la condición
do
  bloque
done
```

➢ for:

- C-*style*:

```
for ((i=0; i < 10; i++))
do
  bloque
done
```

- Con lista de valores (foreach):

```
for i in valor1 valor2 valor3 valorN
do
  bloque
done
```

➢ select:

```
select accion in Nuevo Salir
do
  case $accion in
    "Nuevo")
      echo "Seleccionado: Nuevo"
      ;;
    "Salir")
      exit 0
      ;;
  esac
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 program al comando que sigue al bucle terminado. Se usa para salir de un bucle for, while, until o select en determinado momento y pasar la ejecucion a la siguiente iteracion. break n, siendo n un argumento opcional mayor o igual a 1

-Continue: omite los comandos restantes dentro del cuerpo del bucle que lo encierra para la iteracion actual. Pasa el cnotrol del programa a la siguiente iteracion. continue n, siendo n mayor o igual a 1 e indica que se reanuda el bucle n-esimo

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

- `bash` soporta *strings* y *arrays*
- Los nombres son *case sensitive*
- Para crear una variable:

```
NOMBRE="pepe" # SIN espacios alrededor del =
```

- Para accederla se usa `$`:

```
echo $NOMBRE
```

- Para evitar ambigüedades se pueden usar llaves:

```
# Esto no accede a $NOMBRE
echo $NOMBREesto_no_es_parte_de_la_variable
# Esto sí
echo ${NOMBRE}esto_no_es_parte_de_la_variable
```

- Creación:

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

- Asignación de un valor en una posición concreta:

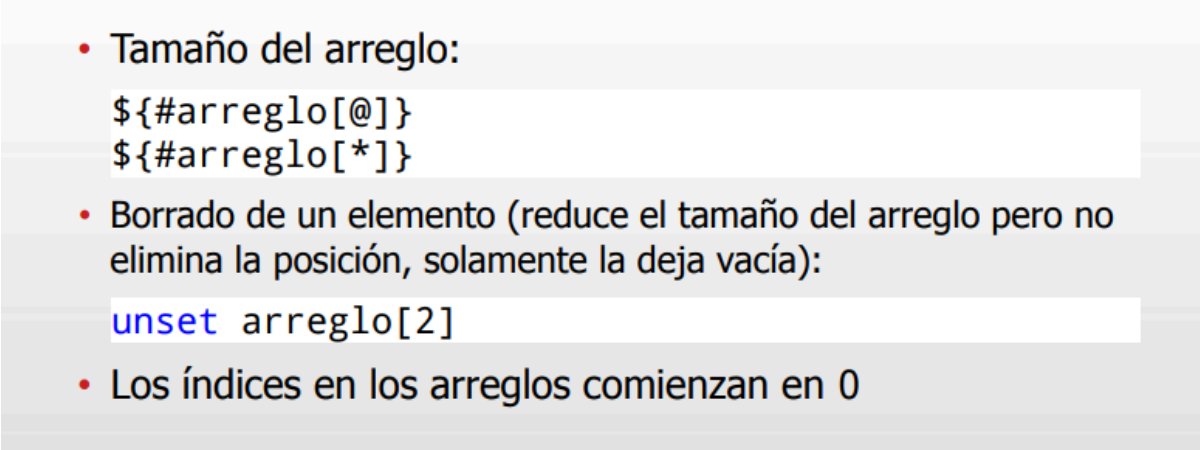
```
arreglo_b[2]=spam
```

- 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[*]}
```



(en el punto 2 tengo mas o menos lo mismo explicado)

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

-Las funciones permiten modularizar el comportamiento de los scripts. Se pueden declarar de 2 formas:

-function nombre { bloque }

-nombre() { bloque }

-Con la sentencia return se retorna un valor entre 0 y 255

-El valor de retorno se puede evaluar mediante la variable $?

-Reciben argumentos en las variables $1, $2, etc.

-Pasaje de parametros:

```
# Recibe 2 argumentos y devuelve:
# 1 si el primero es el mayor
# 0 en caso contrario

mayor() {
  echo "Se van a comparar los valores: $*"
  if [ $1 -gt $2 ]; then
    echo "$1 es el mayor"
    return 1
  fi
  echo "$2 es el mayor"
  return 0
}

mayor 5 6     # Invocación
echo $?       # Imprime el exit status de la función
```

12 - 30) HECHO EN SHELL EN CARPETA EJERCICIOS