Trabajo Práctico Nº 3

Shell Scripting

1.- ¿Qué es el Shell Scripting?

Es un archivo con una secuencia de comandos de determinado Shell.

¿A qué tipos de tareas están orientados los script?

La programación en shell-script es muy útil para resolver tareas repetitivas, típicas de los Administradores.

¿Los scripts deben compilarse? ¿Por qué?

No. Son interpretados comando por comando cada vez que se ejecutan.

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

echo:

El comando echo se utiliza para mostrar cadenas de texto (strings) en la pantalla de la terminal.

Parámetros

echo cuenta con dos parámetros principales, -n y -e, siendo este último para usar distintos “saltos de línea“.

-n –> No hace un salto de línea al final de la cadena de texto.

-e –> Activa los distintos saltos de línea, estos saltos de linea están precedidos por la barra invertida (\).

Distintos saltos de línea:

\a –> Alambra (timbre).

\b –> Espacio atrás.

\c –> Suprime “salto de linea” final.

\f –> Efectua un salto de página.

\n –> Efectua un salto de línea.

\r –> Retorno de carro.

\t –> Hace un tabulado horizontal.

\v –> Hace un tabulado vertical.

\\ –> Barra invertida.

\nnn –> Escribe el caracter ASCII cuya codigo es “nnn” (escrito en octal).

Sintaxis del comando echo:

echo [-n][-e | parámetros]

read:

Lee una línea de la entrada estándar y asigna las palabras de la línea a las variables indicadas, si el número de palabras es mayor que el número de variables se asigna a la última variable las palabras sobrantes. Si es a la inversa se restaura el valor de las variables no asignadas.

read -e inserta toda la línea en la primer variable

read -p frase muestra frase si se lee de un terminal

read -p "Escriba algo: " palabra1 palabra2 palabra3

echo $palabra1 $palabra2 $palabra3

¿Como se indican los comentarios dentro de un script?

Las líneas que empiezan por "#" son comentarios.

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

Las variables en Bash se definen como NOMBRE=valor (sin espacios antes o después del simbolo '=') y su valor se usa, poniendo el simbolo '$' delante del nombre de la variable, $NOMBRE.

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:

mkdir practica-shell-script

touch mostrar.sh

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

chmod 777 mostrar.sh

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

c) ¿Qué resultado visualiza?

Nada.

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

Para incluir la salida de un comando dentro de una línea de comandos de otro comando, encierra el comando cuya salida quieres incluir, entre backquotes (`)

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.

gedit mostrar.sh

echo "Directorio

personal:`pwd`"

echo “Contenido del directorio: `ls`”

echo “Espacio libre: `df`”

4.- Parametrización:

¿Cómo se acceden a los parámetros enviados al script al momento de su invocación?

Existen dos categorías de estos: los parámetros posiciónales y los parámetros especiales.

Los parámetros posiciónales

Estos son simplemente lo argumentos pasados al script, cuando un script es invocado.

Estos son almacenados en las variables reservadas 1,2,3,...9,10,11,... y pueden ser llamados con las expresiones $1,$2...${10},${11}...

Nota: El shell Bourne está limitado a los parámetros del 0 al 9.

Los parámetros especiales

Estas también son variables reservadas que permite en algunos casos efectuar operaciones sobre los mismos parámetros.

Estos parámetros son los siguientes:

$0 Contiene el nombre del script tal como es invocado

$\* El conjunto de todos los parámetros en un solo argumento

$@ El conjunto de argumentos, un argumento por parámetro

$# El número de parámetros pasados al script

$? El código de retorno del último comando

$$ El PID del shell que ejecuta el script

$! El PID del último proceso ejecutado en segundo plano

$HOME el nombre del directorio personal del usuario

5.- ¿Cual es la funcionalidad de comando exit?

Cierra las ventanas o las conexiones remotas establecidas o las shell abiertas.

¿Qué valores recibe como parámetro y cual es su significado?

exit 6 :termina el script devolviendo el número indicado, lo que puede usarse para determinar condiciones de error.

exit 0 :termina el script devolviendo 0, para indicar la finalización

exitosa de tareas. Escribir sólo exit también devuelve código de error 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.

+ Suma

- Resta

/ División

\\* Multiplicación

% Modulo

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 ].

#!/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:"

pwd

Investigar que 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.

Puede ser una evaluación: de cadenas de caracteres, números enteros, archivos (evaluamos permisos y existencia del archivo) u operadores lógicos que permiten realizar una combinación de varias evaluaciones.

8.- Estructuras de control. Investigue la sintaxis de las siguientes estructuras de control

incluidas en shell scripting:

o if

if [ condicion ]

then bloque de comandos

else

bloque de comandos

fi

o case

case variable in

patron1) comandos condicion1;;

patron2) comandos condicion2;;

...

patron n) comandos condicion n;;

\*) comandos si no se cumplen ninguna;;

esac

o while

While (condición)

do

comandos

done

o for

- for variable in [lista de palabras]; do comandos; done

- for ((variable=valor\_inicial; a reporte.txt

17.- Escribir un script que al ejecutarse imprima en pantalla los nombre 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

mARIA

Ayuda: Investigar el comando tr

tr es un filtro que nos permite cambiar una determinada información de un archivo por otra.

Cambia cada uno de los caracteres especificados en el conjunto inicial por los caracteres especificados en el conjunto final.

ls | tr [:lower:] [:upper:]

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.

Ayuda: Investigar la sentencia sleep

#!/bin/bash

#Recibe parametro un nombre de usuario

#Ejemplo de Ejecucion ejercicio18usuario

if [ $# -ne 1 ]

then

echo "La Cantidad de parametros es incorrecta"

exit 1

fi

#Se podria validar contra el /etc/passwd que el usuario exista

while true; do

seLogueo=`who | grep $1 | wc -l`

if [ $seLogueo != 0 ]

then

echo "usuario $1 logueado"

exit 0

fi

sleep 10

done

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 como 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 los elementos de la pila

#!/bin/bash

#Implementa una pila con arreglos

pila=()

function push

{

if [ $# -ne 1 ]

then

echo "La Cantidad de parametros es incorrecta"

return 1

fi

longitud=${#pila[\*]}

pila[$longitud]=$1

}

function pop

{

ultimoElem=`expr ${#pila[\*]} - 1`

echo ${pila[$ultimoElem]}

unset pila[$ultimoElem]

}

function length

{

echo ${#pila[\*]}

}

function print

{

echo ${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 y luego la totalidad

de los elementos que en ella se encuentran.

push primero

push segundo

push tercero

push cuarto

push quinto

push sexto

push septimo

push octavo

push noveno

push decimo

pop

pop

pop

echo "Longitud:"

length

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

num=(1 3 5 7 9 3 5 4)

function productoria

{

x=1

LONG=`expr ${#num[\*]} - 1`

for i in `seq 0 $LONG`

do

x=`expr $x \\* ${num[$i]}`

done

}

productoria

echo $x

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

arr=(1 2 3 4 5)

impares=0

for i in `seq 0 4`; do

resto=`expr ${arr[$i]} % 2`

if [ $resto = 0 ]; then

echo ${arr[$i]}

else impares=`expr $impares + 1`

fi

done

echo Total impares:$impares

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

array=($(cat /etc/group | grep -E ^users: | cut -d: -f4 | tr ',' ' '))

echo ${array[\*]}

arr=(1 2 3 4 5)

case $1 in

"-b")

long= `expr ${#arr[\*]} - 1`

if [ $2 < $long ]

then

echo ${arr[$2]}

fi;;

"-l")

echo ${#arr[\*]};;

"-i")

echo ${arr[\*]};;

\*)

echo "Parametro incorrecto"

esac

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.

if [ $# -eq 0 ]; then

echo Cantidad de parametros incorrecta.

exit

fi

INEX=0

x=1

for i in $\*; do

RESTO=`expr $x % 2`

if [ $RESTO -ne 0 ]; then

if [ -e $i ]; then

if [ -d $i ]; then

echo "$i es un directorio."

else echo "$i es un fichero."

fi

else INEX=`expr $INEX + 1`

fi

fi

x=`expr $x + 1`

done

echo Inexistentes:$INEX

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

o inicializar: Crea un arreglo llamado array vacío

o agregar\_elem : Agrega al final del arreglo el parámetro recibido

o eliminar\_elem : Elimina del arreglo el elemento que se encuentra en la posición recibida como parámetro. Debe validar que se reciba una posición válida

o longitud: Imprime la longitud del arreglo en pantalla

o imprimir: Imprime todos los elementos del arreglo en pantalla

o inicializarConValores : Crea un arreglo con longitud y en todas las posiciones asigna el valor

function inicializar

{

array=()

}

function agregar\_elem

{

if [ $# -ne 1 ]; then

echo "La Cantidad de parametros es incorrecta"

return 1

fi

longitud=${#array[\*]}

array[$longitud]=$1

}

function eliminar\_elem

{

if [ $# -ne 1 ]; then

echo "La Cantidad de parametros es incorrecta"

return 1

fi

ultimoElem=`expr ${#array[\*]} - 1`

if [ $1 -le $ultimoElem ];then

echo ${array[$1]}

unset array[$1]

else echo "La posición no es válida"

fi

}

function longitud

{

echo ${#array[\*]}

}

function imprimir

{

echo ${array[\*]}

}

function inicializarConValores

{

LONG=`expr $1 - 1`

for i in `seq 0 $LONG`; do

array[$i]=$2

done

}

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.

if [ $# -ne 1 ]; then

echo Cantidad de parametros incorrecta.

exit

fi

if [ -e $1 ]; then

CANT=0

for i in `ls $1`; do

if [ -f $i ]&&[ -r $i ]&&[ - w $i ]; then

CANT=`expr $CANT + 1`

fi

done

echo $CANT

else echo "No existe directorio."

exit 4

fi

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:

o verArchivo : 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

o cantidadArchivos: Imprime la cantidad de archivos del /home con terminación

.doc

o borrarArchivo : 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

if [ $# -ne 1 ]; then

echo "La Cantidad de parametros es incorrecta"

exit 1

fi

ARRAY=()

LISTA=`ls $1 |

grep ".doc$"`

for i in $LISTA; do

ARRAY[$pos]=$i

pos=`expr $pos + 1`

done

function verArchivo

{

if [ $# -ne 1 ]; then

echo "La Cantidad de parametros es incorrecta"

exit 1

fi

LONG=`expr ${#ARRAY[\*]} - 1`

for i in `seq 0 $LONG`; do

if [ ${ARRAY[$i]} = $1 ]; then

echo $1

return 0

fi

done

echo "Archivo no encontrado"

return 5

}

function cantidadArchivos

{

echo ${#ARRAY[\*]}

}

function borrarArchivo

{

if [ $# -ne 2 ]; then

echo "La Cantidad de parametros es incorrecta"

exit 1

fi

LONG=`expr ${#ARRAY[\*]} - 1`

i=0

x=0

while [ $i -le $LONG ] && [ $x -eq 0 ]; do

if [ ${ARRAY[$i]} = $2 ]; then

unset ARRAY[$i]

x=1

fi

i=`expr $i + 1`

done

if [ $x -eq 0 ]; then

echo "Archivo no encontrado"

return 10

else

echo "Confirmar borrado lógico. (si-no)"

read CONF

if [ $CONF = "no" ]; then

rm $1/$2

fi

fi

}

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

ACTUAL=`pwd`

TOTAL=0

if [ ! -d /home/marilyn/img ]; then

mkdir /home/marilyn/img

fi

for i in `ls $ACTUAL`; do

if [ -x $i ]&&[ ! -d $i ]; then

mv $i /home/marilyn/img

TOTAL=` expr $TOTAL + 1`

echo "Archivo $i movido"

fi

done

if [ $TOTAL -eq 0 ]; then

echo "No se movió ningun fichero"

fi

echo "Se movieron $TOTAL fichero"