Introducción a los Sistemas Operativos

# Práctica 3

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

El Shell Scripting es la creación de scripts (programas pequeños escritos en un lenguaje de programación interpretado) que se ejecutan en un shell o intérprete de comandos. El objetivo de un script es automatizar tareas repetitivas o complejas que normalmente se realizan manualmente en la línea de comandos.

Los scripts pueden estar orientados a una amplia gama de tareas, desde la administración del sistema hasta el procesamiento de datos y la automatización de tareas de desarrollo. Por ejemplo, un script puede automatizar la copia de archivos, la eliminación de archivos antiguos, la instalación de paquetes, el procesamiento de datos, la manipulación de archivos de configuración y mucho más.

A diferencia de los lenguajes de programación compilados, los scripts no necesitan ser compilados antes de su ejecución. En cambio, se ejecutan directamente por el intérprete de comandos. Esto ofrece varias ventajas, como la capacidad de depurar rápidamente y modificar el script sobre la marcha. Además, los scripts son portables y se pueden ejecutar en diferentes sistemas operativos sin necesidad de realizar cambios de código.

2. Investigar la funcionalidad de los comandos echo y read

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

El comando "echo" es un comando utilizado en Shell Scripting que se utiliza para imprimir texto en la pantalla o en un archivo de salida. Puede imprimir una cadena de texto, variables, comodines y cualquier otro tipo de contenido. Por ejemplo:

#!/bin/bash

echo "Hola, este es un ejemplo de texto impreso en pantalla"

El comando "read" se utiliza en Shell Scripting para leer una entrada de texto del usuario y asignarla a una variable. Esto es útil cuando se necesita que el usuario proporcione información al script. Por ejemplo:

#!/bin/bash

echo "¿Cuál es tu nombre?"

read nombre

echo "Hola, $nombre. Bienvenido!"

Para indicar un comentario dentro de un script de Shell, se utiliza el carácter "#". Todo lo que sigue después del carácter "#" en una línea se considera un comentario y se ignora durante la ejecución del script. Por ejemplo:

#!/bin/bash

# Este es un comentario

echo "Hola, mundo" # Este es otro comentario

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

En Shell Scripting, se pueden declarar variables utilizando un nombre de variable y un valor asignado a esa variable. No es necesario declarar el tipo de variable, ya que las variables son de tipo dinámico. Se pueden asignar valores de texto o numéricos a las variables.

Para hacer referencia a una variable en un script, se puede utilizar el nombre de la variable precedido por el signo de dólar "$". Por ejemplo:

#!/bin/bash

nombre="Juan"

echo "Hola, $nombre"

También es posible utilizar el valor de una variable en una operación matemática. Para hacerlo, es necesario utilizar el signo de dólar y paréntesis alrededor del nombre de la variable. Por ejemplo:

#!/bin/bash

numero1=10

numero2=5

resultado=$(($numero1 + $numero2))

echo "El resultado es $resultado"

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:

Texto

Descripción generada automáticamente

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

chmod +x archivo.sh

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

(c) ¿Qué resultado visualiza?

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

(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?

Los parámetros enviados al script al momento de su invocación se pueden acceder a través de las variables especiales $1, $2, $3, ..., $n, donde "n" es el número de parámetros que se pasan al script. La variable $1 contiene el primer parámetro, $2 contiene el segundo, y así sucesivamente.

La variable $# contiene el número total de parámetros pasados al script. La variable $\* contiene todos los parámetros pasados al script como una única cadena de texto. Es decir, $\* incluye todos los parámetros separados por el primer caracter en la variable de entorno IFS (Internal Field Separator), que por defecto es un espacio en blanco. La variable $? contiene el estado de salida del último comando ejecutado. Si el comando se ejecutó correctamente, su valor será 0. De lo contrario, contendrá un valor diferente de 0 que indica el código de error del comando.

La variable $HOME contiene la ruta del directorio home del usuario actual. Es importante destacar que estas variables sólo están disponibles dentro del script y no afectan el entorno general del sistema.

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

El comando "exit" en un script de shell se utiliza para salir del script y devolver un estado de salida específico. El estado de salida se utiliza para indicar si el script se ejecutó correctamente o no, y puede ser utilizado por otros scripts o programas que llamen al script para determinar si se debe tomar alguna acción adicional.

El comando "exit" puede recibir un valor numérico como parámetro, que es el estado de salida que se devuelve al sistema operativo. El valor por defecto es 0, que indica que el script se ejecutó correctamente sin errores. Si se produce algún error o excepción durante la ejecución del script, se puede devolver un valor diferente de 0 para indicar que algo salió mal.

Por ejemplo, si se tiene un script que realiza una operación y se desea devolver un estado de salida de 1 si la operación falla, se puede utilizar el siguiente comando:

if [ $resultado -ne 0 ]

then

echo "La operación falló"

exit 1

fi

En este caso, si la variable $resultado es diferente de 0, se imprime un mensaje de error y se sale del script con un estado de salida de 1. El valor de 1 se puede utilizar más adelante por otros scripts o programas que llamen a este script para tomar una acción adicional si es necesario.

Es importante destacar que el estado de salida debe ser un número entero entre 0 y 255, ya que estos son los valores permitidos por el sistema operativo.

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.

El comando expr en un script de shell se utiliza para realizar operaciones aritméticas y de cadena de texto.

A continuación se muestran algunos de los operadores que se pueden utilizar con expr:

* Operadores aritméticos:
  + Suma (+): devuelve la suma de dos números.
  + Resta (-): devuelve la resta de dos números.
  + Multiplicación (\*): devuelve el producto de dos números.
  + División (/): devuelve la división entera entre dos números.
  + Módulo (%): devuelve el resto de la división entera entre dos números.
* Operadores de comparación:
  + Igualdad (=): devuelve 1 si los dos argumentos son iguales, de lo contrario devuelve 0.
  + Diferencia (!=): devuelve 1 si los dos argumentos son diferentes, de lo contrario devuelve 0.
  + Mayor que (>): devuelve 1 si el primer argumento es mayor que el segundo, de lo contrario devuelve 0.
  + Menor que (<): devuelve 1 si el primer argumento es menor que el segundo, de lo contrario devuelve 0.
  + Mayor o igual que (>=): devuelve 1 si el primer argumento es mayor o igual que el segundo, de lo contrario devuelve 0.
  + Menor o igual que (<=): devuelve 1 si el primer argumento es menor o igual que el segundo, de lo contrario devuelve 0.
* Operadores de cadena de texto:
  + Longitud de cadena: se utiliza el operador length para obtener la longitud de una cadena de texto.
  + Subcadena: se utiliza el operador substr para obtener una subcadena de una cadena de texto.

# Operadores aritméticos

expr 2 + 2

expr 10 - 5

expr 4 \\* 3

expr 10 / 2

expr 11 % 3

# Operadores de comparación

expr 5 = 5

expr 5 != 5

expr 10 \> 5

expr 10 \< 5

expr 5 \>= 5

expr 5 \<= 5

# Operadores de cadena de texto

expr length "Hola" #Devuelve 4

expr substr "Hola mundo" 6 5 #Devuelve mundo

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

A continuación, se presentan algunos ejemplos de las expresiones que se pueden utilizar con el comando test o [ expresión ]:

1. Evaluación de archivos:

* -e archivo: devuelve true si el archivo existe.
* -f archivo: devuelve true si el archivo es un archivo regular (no un directorio).
* -d archivo: devuelve true si el archivo es un directorio.
* -s archivo: devuelve true si el archivo tiene un tamaño mayor que cero.
* -r archivo: devuelve true si el archivo tiene permisos de lectura para el usuario actual.
* -w archivo: devuelve true si el archivo tiene permisos de escritura para el usuario actual.
* -x archivo: devuelve true si el archivo tiene permisos de ejecución para el usuario actual.

1. Evaluación de cadenas de caracteres:

* -z cadena: devuelve true si la cadena está vacía.
* -n cadena: devuelve true si la cadena no está vacía.
* cadena1 = cadena2: devuelve true si las cadenas son iguales.
* cadena1 != cadena2: devuelve true si las cadenas son diferentes.

1. Evaluaciones numéricas:

* num1 -eq num2: devuelve true si los números son iguales.
* num1 -ne num2: devuelve true si los números son diferentes.
* num1 -lt num2: devuelve true si el número1 es menor que el número2.
* num1 -le num2: devuelve true si el número1 es menor o igual que el número2.
* num1 -gt num2: devuelve true si el número1 es mayor que el número2.
* num1 -ge num2: devuelve true si el número1 es mayor o igual que el número2.

Es importante tener en cuenta que la mayoría de estas expresiones también se pueden usar con el operador de comparación de corchetes dobles, como [[ expresión ]]

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

* If

if [ condición ]

then

# Código a ejecutar si la condición es verdadera

else

# Código a ejecutar si la condición es falsa

fi

* Case

case $variable in

valor1)

# Código a ejecutar si la variable es igual a valor1

;;

valor2)

# Código a ejecutar si la variable es igual a valor2

;;

\*)

# Código a ejecutar si la variable no coincide con ningún valor anterior

;;

esac

* While

while [ condición ]

do

# Código a ejecutar mientras la condición sea verdadera

done

* For

for variable in lista

do

# Código a ejecutar para cada elemento de la lista

Done

donde lista puede ser una lista de elementos separados por espacios o un rango de números como {1..10}.

* Select

select variable in lista

do

# Código a ejecutar para el elemento seleccionado

done

donde lista es una lista de elementos separados por espacios que se presentarán al usuario para que seleccione uno. El valor seleccionado se almacenará en la variable especificada.

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

La sentencia **break** se utiliza para salir de un bucle antes de que se haya completado. Si se encuentra una instrucción break, se salta al final del bucle y se continúa la ejecución a partir de la siguiente instrucción después del bucle. El parámetro opcional de break indica el número de niveles de bucles que se deben romper. Por ejemplo, break 2 rompe dos niveles de bucles anidados.

La sentencia **continue** se utiliza para saltar a la siguiente iteración del bucle. Si se encuentra una instrucción continue, se salta el resto del código dentro de la iteración actual del bucle y se procede con la siguiente iteración. El parámetro opcional de continue indica el número de niveles de bucles que se deben saltar. Por ejemplo, continue 2 salta a la siguiente iteración de los dos bucles anidados más cercanos.

i=0

while [ $i -lt 10 ]

do

i=$((i+1))

if [ $i -eq 3 ]

then

continue # Salta la iteración si i es igual a 3

fi

if [ $i -eq 7 ]

then

break # Sale del bucle si i es igual a 7

fi

echo $i

done

Este código imprime los números del 1 al 6, y luego sale del bucle cuando i es igual a 7. La instrucción **continue** salta la iteración donde i es igual a 3, y la instrucción **break** sale del bucle cuando i es igual a 7.

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

En Shell Script existen dos tipos de variables: las variables de entorno y las variables locales.

Las variables de entorno son variables que se establecen para todo el sistema y se pueden acceder desde cualquier parte del sistema. Por ejemplo, la variable de entorno $HOME contiene la ruta al directorio de inicio del usuario.

Las variables locales son variables definidas en un script o una función y solo se pueden acceder dentro de ese ámbito. Estas variables se pueden definir y modificar en cualquier parte del script o función.

Shell Script no es un lenguaje fuertemente tipado, lo que significa que no se requiere declarar explícitamente el tipo de variable al definirla. El tipo de una variable se determina automáticamente por el contenido que se le asigna.

Para declarar un arreglo, se usa la sintaxis **nombre\_del\_arreglo=(valor1 valor2 valor3 ...)**. Los elementos en el arreglo se separan por espacios en blanco. Por ejemplo, para definir un arreglo de nombres, se puede escribir:

nombres=(Juan Maria Pedro Ana)

Para acceder a los elementos de un arreglo, se usa la sintaxis **${nombre\_del\_arreglo[indice]}**. Los índices de los elementos del arreglo comienzan en 0. Por ejemplo, para acceder al segundo elemento del arreglo de nombres, se puede escribir:

echo ${nombres[1]} # Imprime "Maria"

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

Sí, es posible definir funciones dentro de un script de Shell. Para definir una función, se utiliza la siguiente sintaxis:

nombre\_funcion() {

# Código de la función

}

Por ejemplo, la siguiente función llamada **saludar** simplemente imprime un saludo en la pantalla:

saludar() {

echo "¡Hola, mundo!"

}

Para llamar a una función, simplemente se escribe su nombre seguido de paréntesis:

saludar # Llama a la función "saludar"

Las funciones también pueden recibir parámetros. Los parámetros se pasan a la función dentro de los paréntesis, separados por espacios. Dentro de la función, se hace referencia a los parámetros utilizando **$1**, **$2**, **$3**, etc., para el primer, segundo, tercer, etc. parámetro, respectivamente. Por ejemplo, la siguiente función llamada **saludar\_persona** acepta un parámetro, el nombre de una persona, y lo utiliza para imprimir un mensaje personalizado:

saludar\_persona() {

echo "¡Hola, $1!"

}

saludar\_persona "Juan" # Llama a la función "saludar\_persona" con el parámetro "Juan"

También es posible que una función devuelva un valor utilizando el comando **return**. Por ejemplo, la siguiente función llamada **sumar** toma dos parámetros y devuelve su suma:

sumar() {

local resultado=$(( $1 + $2 )) # Se almacena el resultado en una variable local

return $resultado # Se devuelve el resultado

}

suma=$(sumar 5 3) # Llama a la función "sumar" y almacena su resultado en una variable

echo $suma # Imprime "8"

Es posible llamar a una función desde otra función y pasar parámetros de una a otra utilizando los mismos principios explicados anteriormente.

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 cual 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 parámetro.

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 parametro agregandole 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 la 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

El comando cut tiene varios parámetros que se pueden utilizar para personalizar su comportamiento. Algunos de los parámetros más comunes son:

* -c: Corta caracteres específicos de cada línea.
* -d: Especifica el delimitador que se utilizará para separar los campos.
* -f: Especifica qué campos se deben cortar.

Aquí hay algunos ejemplos de cómo usar el comando cut:

* Cortar los primeros 10 caracteres de cada línea en un archivo:

cut -c 1-10 archivo.txt

* Cortar el tercer campo de cada línea en un archivo delimitado por comas:

cut -d ',' -f 3 archivo.csv

* Cortar los campos 2 y 4 de cada línea en un archivo delimitado por espacios:

cut -d ' ' -f 2,4 archivo.txt

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 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

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 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 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 )

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 parámetro:

* “-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 : 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 : 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.