

Introducción a los Sistemas Operativos

Práctica 3

**Objetivo**

El objetivo de esta práctica es que el alumno desarrolle habilidades concernientes a *Shell Scrip- ting*.

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 o programas en lenguajes de shell para automatizar tareas en sistemas operativos. Estos scripts se utilizan para una variedad de tareas, como automatización de tareas repetitivas, administración del sistema, procesamiento de datos y texto, entre otros. Los scripts de shell no se compilan, ya que son interpretados directamente por el intérprete de shell, lo que permite una rápida iteración y modificación del código, aunque puede tener como consecuencia una ejecución más lenta en comparación con programas compilados.

1. Investigar la funcionalidad de los comandos echo y read
   1. ¿Como se indican los comentarios dentro de un script?
   2. ¿Cómo se declaran y se hace referencia a variables dentro de un script?

En un script de shell, los comentarios se indican utilizando el símbolo "#" (almohadilla)

* Declaración de variables: Las variables se declaran sin la necesidad de especificar un tipo de dato y se asignan valores utilizando el operador "=" (igual). Por ejemplo, **mi\_variable="Hola, mundo"** declara una variable llamada "mi\_variable" y le asigna el valor "Hola, mundo".
* Referencia a variables: Para hacer referencia al valor de una variable, se coloca el signo "$" seguido del nombre de la variable. Por ejemplo, **echo $mi\_variable** imprimirá el valor de la variable "mi\_variable".
* También se puede hacer un read mi variable

1. 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 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:"

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

Chmod 700 Miarchivo.ch

* 1. Ejecutar el archivo creado de la siguiente manera: ./mostrar
  2. ¿Qué resultado visualiza?

Un programa

* 1. Las backquotes (`) entre el comando whoami ilustran el uso de la sustitución de co- mandos. ¿Qué significa esto?

En tu ejemplo, al usar backquotes alrededor del comando **whoami**, estás capturando la salida del comando **whoami**, que normalmente mostraría el nombre del usuario actual, y utilizando ese resultado como parte de otro comando o en la línea de comandos en general

* 1. 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.

1. 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?
   1. **$0**: Representa el nombre del script en sí mismo.
   2. **$1**, **$2**, **$3**, ...: Representan los argumentos o parámetros posicionales que se pasan al script. **$1** es el primer argumento, **$2** el segundo, y así sucesivamente.
   3. **$#**: Indica la cantidad total de argumentos pasados al script.
   4. **$\*** o **$@**: Representan todos los argumentos como una lista (espaciados). **$\*** trata todos los argumentos como una sola cadena, mientras que **$@** los trata como una lista de elementos individuales.
   5. **$?**: Contiene el código de retorno (exit status) del último comando ejecutado. Un valor de 0 generalmente indica que el comando se ejecutó con éxito, mientras que un valor distinto de 0 indica un error.
   6. **$HOME**: Contiene la ruta al directorio principal del usuario actual. Por ejemplo, **/home/usuario** en sistemas Unix/Linux.
2. ¿Cual es la funcionalidad de comando exit? ¿Qué valores recibe como parámetro y cual es su significado?

Los valores que recibe como parámetro van de 0 a 255

El comando **exit** se utiliza para finalizar la ejecución de un script de shell y especificar un código de retorno. **exit** sin argumentos devuelve un código de retorno predeterminado de 0 (éxito), mientras que **exit** con un valor numérico como argumento permite especificar un código de retorno personalizado para indicar si ocurrió algún error durante la ejecución del script.

1. 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.

**Operadores aritméticos**:

* 1. **+**: Suma.
  2. **-**: Resta.
  3. **\***: Multiplicación.
  4. **/**: División entera.
  5. **%**: Módulo (resto de la división).

**Operadores de comparación**:

* 1. **=**: Igual a.
  2. **!=**: Diferente de.
  3. **<**: Menor que.
  4. **<=**: Menor o igual que.
  5. **>**: Mayor que.
  6. **>=**: Mayor o igual que.

**Operadores lógicos**:

* 1. **:** (dos puntos) o **|**: OR lógico.
  2. **&**: AND lógico.
  3. **!**: NOT lógico.

**Operadores de asignación**:

* 1. **=**: Asignación.

**Operadores de concatenación de cadenas**:

* 1. **:** (dos puntos) o **.**: Concatenación de cadenas.

Ej :

if [ $(expr 10 > 5) -eq 1 ]; then

echo "10 es mayor que 5"

fi

cadena1="Hola, "

cadena2="mundo!"

concatenada=$(expr "$cadena1" : '\(.\*\)' : "$cadena2")

echo $concatenada # Esto imprimirá "Hola, mundo!"

1. 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 ma- nera [ expresión ]. 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.

**Evaluación de archivos**:

* 1. **-e archivo**: True si el archivo existe.
  2. **-f archivo**: True si el archivo existe y es un archivo regular (no un directorio ni un enlace simbólico).
  3. **-d directorio**: True si el directorio existe.
  4. **-r archivo**: True si el archivo es legible.
  5. **-w archivo**: True si el archivo es escribible.
  6. **-x archivo**: True si el archivo es ejecutable.
  7. **-s archivo**: True si el archivo tiene un tamaño mayor que cero.
  8. **-z cadena**: True si la cadena de caracteres está vacía.
  9. **-h archivo**: True si el archivo es un enlace simbólico.

**Evaluación de cadenas de caracteres**:

* 1. **-n cadena**: True si la cadena de caracteres no está vacía.
  2. **-z cadena**: True si la cadena de caracteres está vacía.
  3. **cadena1 = cadena2**: True si las dos cadenas son iguales.
  4. **cadena1 != cadena2**: True si las dos cadenas son diferentes.

**Evaluaciones numéricas**:

* 1. **n1 -eq n2**: True si n1 es igual a n2 (números iguales).
  2. **n1 -ne n2**: True si n1 no es igual a n2 (números diferentes).
  3. **n1 -lt n2**: True si n1 es menor que n2 (número 1 es menor que número 2).
  4. **n1 -le n2**: True si n1 es menor o igual que n2 (número 1 es menor o igual que número 2).
  5. **n1 -gt n2**: True si n1 es mayor que n2 (número 1 es mayor que número 2).
  6. **n1 -ge n2**: True si n1 es mayor o igual que n2 (número 1 es mayor o igual que número 2).

1. Estructuras de control. Investigue la sintaxis de las siguientes estructuras de control inclui- das en shell scripting:

if case while for select

if [ "$edad" -ge 18 ]

then

# Código a ejecutar si la condición es verdadera (true).

elif [ "$edad" -ge 19 ]

then

# Código a ejecutar si la otra condición es verdadera (true).

else

# Código a ejecutar si ninguna condición es verdadera (opcional).

Fi

case $dia\_semana in

lunes)

echo "Hoy es el primer día de la semana."

;;

martes)

echo "Hoy es el segundo día de la semana."

;;

\*)

echo "Hoy es un día desconocido."

;;

Esac

while [ $contador -le 5 ]; do

echo "Iteración $contador"

contador=$((contador + 1))

done

frutas=("manzana" "pera" "naranja" "uva")

for fruta in "${frutas[@]}"; do

echo "Me gusta la $fruta."

Done

PS3="Selecciona una opción: "

opciones=("Opción 1" "Opción 2" "Salir")

select opcion in "${opciones[@]}"; do

case $opcion in

"Opción 1")

echo "Has seleccionado la Opción 1."

;;

"Opción 2")

echo "Has seleccionado la Opción 2."

;;

"Salir")

echo "Saliendo del menú."

break

;;

\*)

echo "Opción no válida."

;;

Esac

1. ¿Qué acciones realizan las sentencias break y continue dentro de un bucle?

**break** se utiliza para salir completamente de un bucle, mientras que **continue** se utiliza para saltar la iteración actual y continuar con la siguiente iteración en el bucle

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

Strings y vectores de strings, El Shell no es fuertemente tipado

Ej de Arreglos mi\_array=("valor1" "valor2" "valor3")

primer\_elemento="${mi\_array[0]}"

segundo\_elemento="${mi\_array[1]}"

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

En shell scripting, es posible definir funciones para encapsular un conjunto de comandos y mejorar la modularidad del código. Las funciones se declaran con un nombre seguido de paréntesis y se llaman utilizando ese nombre. Los parámetros se pasan como argumentos posicionales y se acceden dentro de la función utilizando **$1**, **$2**, **"$@"**, etc. Esto permite reutilizar código y mejorar la organización de scripts en shell.

Ej:

mi\_funcion() {

for parametro in "$@"; do

echo "Parámetro: $parametro"

done

}

mi\_funcion "Manzana" "Banana" "Cereza"

1. Evaluación de expresiones:
   1. 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.
   2. 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.
   3. Realizar una calculadora que ejecute las 4 operaciones básicas: +, - ,\*, %. Esta calcu- ladora debe funcionar recibiendo la operación y los números como parámetros

Ver ARCHIVO11

#!/bin/bash

echo -n "Ingrese 2 numeros "

read x y

if [ "$#" -ne 3 ]

then echo "Error parametros"

exit 3

fi

echo "Mayor"

if [ "$x" -lt "$y" ]

then echo "$y"

else echo "$x"

fi

echo "Suma: $((x + y))"

echo "Resta: $((x - y))"

echo "Multiplicación: $((x \* y))"

if [ "$y" -eq 0 ]

then

echo "No se puede dividir por cero."

else

echo "División: $((x / y))"

fi

if [ "$1" -lt "$2" ]

then echo "$2"

else echo "$1"

fi

echo "Suma: $(($1 + $2))"

echo "Resta: $(($1 - 42))"

echo "Multiplicación: $(($1 \* $2))"

if [ "$2" -eq 0 ]

then

echo "No se puede dividir por cero."

else

echo "División: $(($1 / $2))"

fi

echo "Calculadora"

case "$3" in

+)echo "Res: $(($1 + $2))";;

-)echo "Res: $(($1 - $2))";;

x)echo "Res: $(($1 \* $2))";;

%)echo "Res: $(($1 % $2))";;

.\*)echo "Invalido"

esac

1. Uso de las estructuras de control:
   1. Realizar un script que visualice por pantalla los números del 1 al 100 así como sus cuadrados.
   2. 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.

* 1. 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.

#!/bin/bash

#uncomentario

for ((i = 1; i <= 10; i++)); do

cuadrado=$((i \* i))

echo "Número: $i - Cuadrado: $cuadrado"

done

numero=1

while [ $numero -le 10 ]; do

cuadrado=$((numero \* numero))

echo "El numero es $numero y su cuadrado es $cuadrado"

numero=$((numero + 1))

done

case "$1" in

listar) echo "El directorio contiene $(ls)";;

donde) echo "El directorio esta en $(pwd)";;

quien) echo "Esta conectado $(who)";;

\*) echo "Error";;

Esac

if [ "$#" -ne 2 ];

then echo "Error"

echo exit 77

fi

nombre="$2"

if [ -e "$nombre" ]; then

if [ -d "$nombre" ]; then

echo "El directorio $nombre existe."

elif [ -f "$nombre" ]; then

echo "El archivo $nombre existe."

else

echo "El objeto $nombre existe, pero no es ni un ar> fi

else

echo "El objeto $nombre no existe. Creando directorio..> mkdir "$nombre"

echo "Directorio $nombre creado."

Fi

EJERCICIO13.sh

1. 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

EJERCICIO14.sh

1. 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** en sistemas Unix y Linux se utiliza para extraer secciones específicas de líneas de texto de una entrada. Los parámetros principales son:

* **-c LISTA\_DE\_CAMPOS** o **-f LISTA\_DE\_CAMPOS**: Permite especificar los caracteres o campos que se desean extraer de cada línea.
* **-d DELIMITADOR**: Define un delimitador personalizado en lugar del valor por defecto (tabulación).
* **-s**: Evita que se muestren líneas que no contienen el delimitador cuando se utiliza **-f**.

El comando es útil para trabajar con archivos de texto o la salida de otros comandos y puede ser personalizado para satisfacer tus necesidades de extracción de datos.

Ejemplo 1: Para extraer los primeros tres caracteres de cada línea de un archivo llamado **archivo.txt**: cut -c 1-3 archivo.txt

Ejemplo 2: Supongamos que tienes un archivo CSV (valores separados por comas) llamado **datos.csv** y deseas extraer el segundo campo de cada línea (columna 2)

cut -d ',' -f 2 datos.csv

Ejemplo 3: Para extraer el tercer campo de un archivo TSV (valores separados por tabulaciones) llamado **datos.tsv**:

cut -d $'\t' -f 3 datos.tsv

Ejemplo 4: Supongamos que tienes un archivo CSV llamado **datos.csv** con algunas líneas que no tienen comas. Para extraer el segundo campo solo de las líneas que contienen comas:

cut -d ',' -f 2 -s datos.csv

1. 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

EJERCICIO16.sh

1. 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

#!/bin/bash

cat "$1" | tr -d 'A' | tr -d 'a' | tr 'a-zA-Z' 'A-Za-z'

cat "$1" | tr -d 'A' | tr -d 'a' | tr 'a-zA-Z' 'A-Za-z' > "$1"

1. 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.

Ejercicio18 REVISAR

1. 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

1. Evaluar Expresiones
2. Probar estructuras de control

...

Ingrese la opción a ejecutar: 03

EJERCICIO19.sh

#!/bin/bash Importanteeeeee

#Hola

echo " 12- Calculadora (Ingresa 1 2 +)"

echo " 13-Cuadrados (Ingresa Arch)"

echo " 14-Renombre (Fichero, -b, Cadena)"

echo " 16-Tabla (.txt)"

echo " 17-Cambio (Arch) "

echo "Muestre la variable"

read x

var="/home/ignacio/ISO2023/EJERCICIO"

fin=".sh"

while [ "$x" -ne 0 ]; do

x=${var}${x}${fin}

echo $x

echo "Ingrese las vars"

read A B C

"$x" "A" "B" "C"

echo "Muestre la variable"

read x

done

1. 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

EJERCICIO20

1. Dentro del mismo script y utilizando las funciones implementadas: Agregue 10 elementos a la pila luego saque 3 de ellos

Imprima la longitud de la cola

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

1. 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 (EJERCICIO22.sh)

#!/bin/bash

num=(10 3 5 7 9 3 5 4)

producto=1

productoria(){

for i in "${num[@]}";do

producto=$(( i \* producto ))

done

}

productoria

echo "$producto"

1. 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. (EJERCICIO22.sh)
2. 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

#!/bin/bash

vector1=(1 80 65 35 2)

vector2=(5 98 3 41 8)

if [ ${#vector1[@]} -ne ${#vector2[@]} ]; then

echo "Los vectores no tienen la misma longitud"

exit 1

fi

for ((i = 0; i < ${#vector1[@]}; i++)); do

aux=$((vector1[$i] + vector2[$i]))

echo "La suma de los elementos de la posición $i d>

done

1. 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

EJERCICIO25.sh (ChatGPT) Revisar

1. Escriba un script que reciba una cantidad desconocida de parámetros al momento de su invo- cación (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 reci-

bidos, y solo para aquellos parámetros que se encuentren en posiciones impares (el primero, el tercero, el 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. EJERCICIO26.sh

1. 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>

EJERCICIO27.sh

1. 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.

EJERCICIO28.sh

1. 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 en- contrado” 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 archi- vo 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 EJERCICIO29.sh

1. 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.

EJERCICIO30.sh