

Introducción a los Sistemas Operativos

Práctica 3

**Objetivo**

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

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

 **Shell Scripting** es el proceso de escribir un conjunto de comandos en un archivo para que se ejecuten en un shell, que es la interfaz entre el usuario y el sistema operativo.

 **Tareas**: Los scripts están orientados a tareas repetitivas y automatizadas como la manipulación de archivos, procesamiento de texto, administración del sistema, y automatización de procesos.

 **Compilación**: No es necesario compilar los scripts porque el shell interpreta cada línea y la ejecuta de inmediato. Esto hace que sean más flexibles y fáciles de modificar.

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?

* **echo**: Imprime texto o el valor de variables en la pantalla.
  + Ejemplo: echo "Hola, Mundo" mostrará Hola, Mundo.
* **read**: Lee entrada del usuario y la guarda en una variable.
  + Ejemplo: read nombre espera que el usuario escriba algo y guarda el valor en la variable nombre.

**(a) Comentarios en un script**

* Para escribir comentarios, se usa el símbolo # al inicio de una línea.
  + Ejemplo: # Este es un comentario

**(b) Declaración y referencia de variables**

* Declaración: Solo debes escribir el nombre de la variable y asignarle un valor.
  + Ejemplo: nombre="Juan"
* Referencia: Para acceder al valor de una variable, antepones $ al nombre de la variable.
  + Ejemplo: echo $nombre imprimirá el valor de nombre, en este caso, Juan.

1. 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:"

* 1. Asignar al archivo creado los permisos necesarios de manera que pueda ejecutarlo
  2. Ejecutar el archivo creado de la siguiente manera: ./mostrar
  3. ¿Qué resultado visualiza?
  4. Las backquotes (`) entre el comando whoami ilustran el uso de la sustitución de comandos. ¿Qué significa esto?
  5. 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.

D) las backquotes permiten que el resultado de un comando se use como parte de una cadena o como un argumento en otro comando, facilitando la creación de scripts dinámicos que pueden interactuar con el entorno del sistema.

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?

Cuando ejecutas un script y le pasas valores al momento de invocarlo, estos valores se convierten en "parámetros posicionales" que puedes acceder de la siguiente manera:

* $1, $2, $3, ...: Representan los parámetros posicionales en el orden en que fueron proporcionados. Por ejemplo, si ejecutas un script como ./mi\_script.sh valor1 valor2 valor3, dentro del script:
  + $1 será valor1
  + $2 será valor2
  + $3 será valor3
* $0: Contiene el nombre del script. En nuestro ejemplo anterior, $0 sería ./mi\_script.sh.

**Variables Especiales**

El shell tiene varias variables especiales que puedes utilizar para obtener información adicional. Aquí están las que necesitas entender para este ejercicio:

* **$#**: Indica el **número de argumentos** pasados al script.
  + Ejemplo: Si ejecutas ./mi\_script.sh valor1 valor2, entonces $# será 2.
* **$\***: Contiene **todos los argumentos** pasados al script en una sola cadena de texto, separados por espacios.
  + Ejemplo: Si ejecutas ./mi\_script.sh valor1 valor2, entonces $\* será valor1 valor2.
* **$?**: Representa el **estado de salida** del último comando ejecutado, indicando si el comando anterior tuvo éxito (0 significa éxito, cualquier otro número indica un error).
  + Ejemplo: Si el último comando tuvo éxito, $? será 0. Si falló, $? será un número distinto de 0.
* **$HOME**: Es una variable del sistema que contiene la ruta del **directorio personal del usuario actual**.
  + Ejemplo: Si tu usuario es alumno, $HOME generalmente será /home/alumno.

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

Causar la terminación de un script, puede tomar valores entre 0 y 255, 0 indica que se ejecuto de forma exitosa, cualquier otro valor indica error, concretamente el 1 es error genérico mientras que los demás se le asigna a un problema especifico.

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.

El comando expr permite realizar evaluaciones de expresiones aritméticas, lógicas y de comparación en la línea de comandos de Unix/Linux. Aquí están las operaciones que se pueden utilizar:

**1. Operaciones Aritméticas**

* **Suma**: expr arg1 + arg2
* **Resta**: expr arg1 - arg2
* **Multiplicación**: expr arg1 \\* arg2 (se debe escapar el asterisco con \)
* **División**: expr arg1 / arg2
* **Módulo** (resto de la división): expr arg1 % arg2

**2. Operaciones de Comparación**

* **Mayor que**: expr arg1 \> arg2
* **Menor que**: expr arg1 \< arg2
* **Mayor o igual que**: expr arg1 \>= arg2
* **Menor o igual que**: expr arg1 \<= arg2
* **Igual a**: expr arg1 = arg2
* **Distinto de**: expr arg1 != arg2

**3. Operaciones Lógicas**

* **AND**: expr arg1 \&\& arg2 (dentro de un contexto de condiciones, no se usa directamente con expr).
* **OR**: expr arg1 \|\| arg2 (igual que el AND, se usa en condiciones).

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 manera [ 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.

El comando test (y su equivalente con corchetes [ expresión ]) se utiliza para evaluar expresiones y generar un valor de retorno: true (0) o false (1). A continuación se detallan los tipos de expresiones que se pueden usar con test.

**1. Evaluación de Archivos**

* **Existencia de archivo**:
  + test -e archivo o [ -e archivo ]: Verifica si el archivo existe.
* **Es un directorio**:
  + test -d archivo o [ -d archivo ]: Verifica si el archivo es un directorio.
* **Es un archivo regular**:
  + test -f archivo o [ -f archivo ]: Verifica si el archivo es un archivo regular.
* **Es legible**:
  + test -r archivo o [ -r archivo ]: Verifica si el archivo es legible.
* **Es escribible**:
  + test -w archivo o [ -w archivo ]: Verifica si el archivo es escribible.
* **Es ejecutable**:
  + test -x archivo o [ -x archivo ]: Verifica si el archivo es ejecutable.

**2. Evaluación de Cadenas de Caracteres**

* **Cadenas vacías**:
  + test -z cadena o [ -z cadena ]: Verifica si la cadena está vacía.
* **Cadenas no vacías**:
  + test -n cadena o [ -n cadena ]: Verifica si la cadena no está vacía.
* **Igualdad de cadenas**:
  + test "cadena1" = "cadena2" o [ "cadena1" = "cadena2" ]: Verifica si ambas cadenas son iguales.
* **Desigualdad de cadenas**:
  + test "cadena1" != "cadena2" o [ "cadena1" != "cadena2" ]: Verifica si las cadenas son diferentes.

**3. Evaluaciones Numéricas**

* **Comparaciones**:
  + **Mayor que**: test num1 -gt num2 o [ num1 -gt num2 ]
  + **Menor que**: test num1 -lt num2 o [ num1 -lt num2 ]
  + **Mayor o igual que**: test num1 -ge num2 o [ num1 -ge num2 ]
  + **Menor o igual que**: test num1 -le num2 o [ num1 -le num2 ]
  + **Igual a**: test num1 -eq num2 o [ num1 -eq num2 ]
  + **Distinto de**: test num1 -ne num2 o [ num1 -ne num2 ]

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

if

case while for select

MiraGitHub

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

**Sentencia break**

* **Acción**: La sentencia break se utiliza para salir de un bucle (como for, while o until) de manera inmediata, interrumpiendo su ejecución. Al encontrar un break, el control del programa se transfiere a la siguiente línea de código que sigue después del bucle.
* **Parámetros**: break no recibe parámetros. Sin embargo, puede recibir un número opcional que indica cuántos niveles de bucles se deben salir. Por defecto, break sale del bucle más interno.

**Sentencia continue**

* **Acción**: La sentencia continue se utiliza para saltar a la siguiente iteración de un bucle, omitiendo el resto del código en la iteración actual. El bucle continúa su ejecución con la siguiente iteración.
* **Parámetros**: Similar a break, continue no recibe parámetros, pero puede aceptar un número que indica cuántos niveles de bucles se deben omitir. Por defecto, continue afecta solo al bucle más interno.

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

**1. Tipos de Variables**

En shell scripting, principalmente se manejan dos tipos de variables:

* **Variables Escalares**: Son las variables más comunes y almacenan un solo valor, que puede ser un número, una cadena de texto o un valor booleano.

**Ejemplo**:

bash

Copiar código

nombre="Juan"

edad=25

es\_estudiante=true

* **Arreglos (Arrays)**: Se utilizan para almacenar múltiples valores bajo un mismo nombre de variable, permitiendo agrupar datos relacionados.

**Ejemplo**:

bash

Copiar código

frutas=("manzana" "naranja" "plátano")

**2. ¿Es Shell Script Fuertemente Tipado?**

No, shell scripting no es un lenguaje fuertemente tipado. Esto significa que no es necesario declarar el tipo de una variable al momento de crearla. Puedes asignar diferentes tipos de valores a la misma variable en diferentes momentos sin restricciones.

**Ejemplo**:

bash

Copiar código

var=10 # Variable numérica

var="texto" # Ahora es una cadena

**3. Definición de Arreglos**

Los arreglos en shell scripting se definen utilizando paréntesis y los elementos se separan por espacios.

**Sintaxis**:

bash

Copiar código

nombre\_arreglo=(elemento1 elemento2 elemento3)

**Acceso a Elementos**: Para acceder a un elemento de un arreglo, se utiliza la sintaxis ${nombre\_arreglo[indice]}, donde indice comienza en 0.

**Ejemplo**:

bash

Copiar código

# Definir un arreglo

colores=("rojo" "verde" "azul")

# Acceder al primer elemento

echo ${colores[0]} # Salida: rojo

# Modificar un elemento

colores[1]="amarillo"

echo ${colores[1]} # Salida: amarillo

# Iterar sobre un arreglo

for color in "${colores[@]}"; do

echo $color

done

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

Si, hay 2 maneras: function nombre {block} **o** nombre() {block}

Las funciones en shell scripting pueden recibir parámetros al momento de ser invocadas. Estos parámetros se pueden acceder dentro de la función utilizando las variables especiales:

* $1, $2, $3, etc., representan los parámetros que se pasan a la función.
* $@ se refiere a todos los parámetros pasados a la función.
* $# indica el número total de parámetros pasados.

**Ejemplo:**

bash

Copiar código

saludar() {

echo "Hola, $1. Tu edad es $2."

}

# Llamar a la función con parámetros

saludar "Juan" 25

**Salida**:

Copiar código

Hola, Juan. Tu edad es 25.

**3. Uso de Parámetros Dentro de Funciones**

Cuando se definen funciones que requieren varios parámetros, se pueden usar todos los parámetros disponibles para realizar diferentes operaciones.

**Ejemplo:**

Copiar código

sumar() {

resultado=$(( $1 + $2 ))

echo "La suma de $1 y $2 es: $resultado"

}

# Llamar a la función con dos parámetros

sumar 10 20

**Salida**:

Copiar código

La suma de 10 y 20 es: 30

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 cuál 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 calculadora debe funcionar recibiendo la operación y los números como parámetros
2. 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.

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

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.
2. 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
3. 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

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.
2. 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

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

1. 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.

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
2. 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.
3. 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

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

1. 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 q

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.

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>

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.

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 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

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.