Introducción a los Sistemas Operativos
Práctica 3

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?

El shell scripting es la escritura y el uso de archivos de scripts escritos en el lenguaje que provee la shell del sistema operativo y ejecutados por la misma. Son archivos de texto plano y agrupan una serie de comandos que bien se podrían ingresar uno a uno por consola.

Son prácticos para manejar y procesar archivos, para la generación de informes, la administración de recursos del sistema y la automatización de tareas repetitivas.
Con Shell Scripting es muy simple crear procesos y manipular sus salidas.

Los scripts no son compilados, sino que son interpretados comando por comando. Esto significa que necesitan un intérprete para codificar la información para que pueda ser procesada y ejecutada. Es independiente de la plataforma sobre la que se ejecuta.

2. Investigar la funcionalidad de los comandos echo y read

El comando **echo** se utiliza para mostrar una línea de texto o cadena que se pasa como argumento. Este comando es uno de los más utilizados y se utiliza en los scripts de Bash para mostrar el texto de estado en un archivo o en la terminal. El comando Echo de Linux repite lo que se le encarga que repita. La funcionalidad básica de echo es la misma en todos los lenguajes de programación: se introduce una entrada, que en la mayoría de los casos es en forma de cadena, y ésta se recibe y se muestra de nuevo sin cambios.

Por otro lado, el comando **read** en Bash se utiliza para leer la entrada del usuario o la entrada de un archivo. En un script, se podría usar un bucle while read para leer de un archivo, que en este caso es la salida del comando echo. Cuando read lee la entrada, guarda la primera palabra en la variable y la procesa en el bucle. La segunda palabra se guarda en el búfer del comando.

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

Los comentarios en bash se indican con un # al principio de la línea.

No admite comentarios de múltiples líneas.

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

Para crear una variable String:

NOMBRE="pepe" # SIN espacios alrededor del =

Para accederla se usa $:

echo $NOMBRE

3. 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:"
```

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

chmod 711 mostrar.sh

(b) Ejecutar el archivo creado

./mostrar

(c) ¿Qué resultado visualiza?

El script se ejecuta correctamente.

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

La sustitución de comandos en Bash es una técnica que permite ejecutar un comando y usar su salida en otro lugar. Esto se hace encerrando el comando entre backquotes

Por ejemplo:

echo "Hoy es `date`"

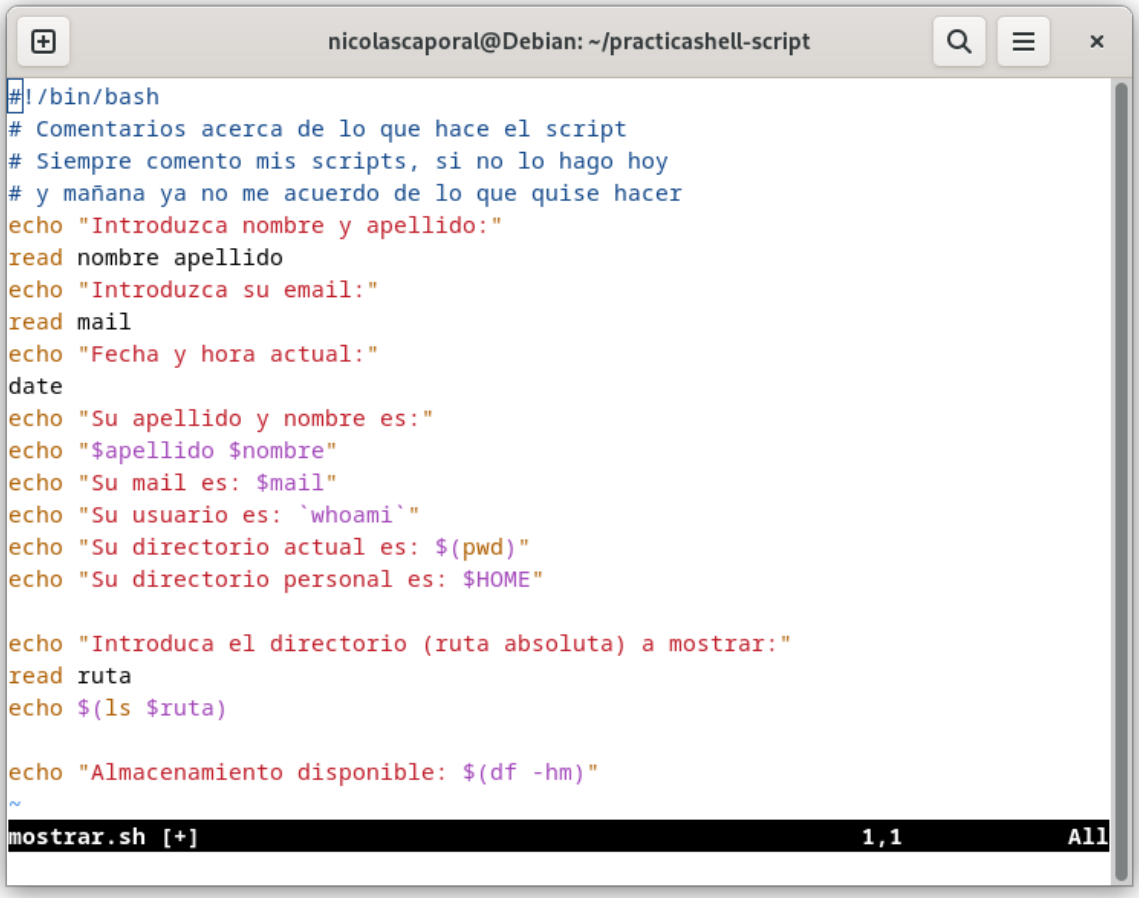
En este caso, el comando `date` se ejecuta y su salida se inserta en el lugar donde estaba el comando.

Además de los backquotes, también puedes usar la sintaxis $()

echo "Hoy es $(date)"

Ambas instrucciones son equivalentes, hacen exactamente lo mismo.

(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 scripts pueden recibir argumentos en su invocación.
Para acceder a esos parametros se utilizan las variables especiales $1, $2, $3, etc. Cada una de estas variables corresponde a un argumento pasado al script, en el orden en el que fueron pasados.

- **$0** contiene la invocación al script.
- **$1, $2, $3, ...** contienen cada uno de los argumentos.
- **$#** contiene la cantidad de argumentos recibidos.
- **$*** contiene la lista de todos los argumentos.
- **$?** contiene en todo momento el valor de retorno del último comando ejecutado
- **$HOME** contiene el directorio personal del usuario

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

Causa la terminación de un script

Recibe como parámetro cualquier valor entre 0 y 255, dicho número es el estado de salida que se devuelve al shell:

- El valor 0 indica que el script se ejecutó de forma exitosa
- Un valor distinto indica un código de error

Se puede consultar el *exit status* con el valor de $?

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.

Las operaciones que se pueden utilizar con el comando `expr` son las siguientes:

- **+** : Suma
- **-** : Resta
- **/** : División
- ***** : Multiplicación[3]
- % : Módulo

En la multiplicación se debe anteponer al signo asterisco (`*`) una barra invertida (`\`) para que Bash no realice sustitución de nombres de archivo.

Por ejemplo, para calcular el resultado de `1 + 2 * 8 / 3`, se usaría el comando `expr` de la siguiente manera:

*expr 1 + 2 * 8 / 3*

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 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 en Bash se utiliza para evaluar expresiones condicionales. Sirve para evaluar una expresión, condición, comprobar los atributos de ficheros y realizar comparaciones de cadenas y aritméticas. Devolviendo un valor de cero 0 (true, verdadero o éxito) ó uno 1 (false, falso o fracaso). Generalmente se usa en scripts.

Pongo 5 ejemplos en cada categoría, para que se entienda el concepto

1. Evaluación de archivos:

- **-e file:** Verdadero si el archivo existe.
- **-d file:** Verdadero si el archivo existe y es un directorio.
- **-f file:** Verdadero si el archivo existe y es un archivo regular.
- **-r file:** Verdadero si el archivo existe y es legible.
- **-w file:** Verdadero si el archivo existe y es escribible.

2. Evaluación de cadenas de caracteres:

- **-z string:** Verdadero si la longitud de la cadena es cero.
- **-n string:** Verdadero si la longitud de la cadena no es cero.
- **string1 = string2:** Verdadero si las cadenas son iguales.
- **string1 != string2:** Verdadero si las cadenas no son iguales.
- **string:** Verdadero si la cadena no está vacía.

3. Evaluaciones numéricas:

- **n1 -eq n2:** Verdadero si n1 es igual a n2.
- **n1 -ne n2:** Verdadero si n1 no es igual a n2.
- **n1 -gt n2:** Verdadero si n1 es mayor que n2.
- **n1 -ge n2:** Verdadero si n1 es mayor o igual a n2.

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

if:

```
if [ condición ]
then
   comando1
   comando2
   ...
fi
```

También se puede usar elif y else:

```
if [ condición ]
then
   comando1
elif [ otra_condición ]
then
   comando2
else
   comando3
fi
```

---

case:

```
case expresión in
   patrón1)
      comando1;;
   patrón2)
      comando2;;
   *)
      comando_por_defecto;;
esac
```

---

while:

```
while [ condición ]
do
   comando1
   comando2
   ...
done
```

---

for:

```
for variable in lista_de_elementos
do
   comando1
   comando2
   ...
done
```

También se puede usar una sintaxis similar a la de C:

```
for (( i=0; i<limite; i++ ))
do
   comando1
   comando2
   ...
done
```

---

select:

```
select variable in lista_de_elementos
do
   comando1  # generalmente un case para manejar la elección del usuario.
done
```

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

Las sentencias `break` y `continue` en Bash se utilizan para controlar el flujo de ejecución dentro de un bucle.

break:

Esta sentencia se utiliza para terminar la ejecución del bucle actual y pasar el control del programa al comando que sigue al bucle terminado. Se utiliza para salir de un bucle `for`, `while`, `until` o `select`. La sintaxis de la sentencia `break` es la siguiente:

break [n]

Donde `[n]` es un argumento opcional y debe ser mayor o igual que 1.
Cuando se proporciona `[n]`, se termina el n-ésimo ciclo de cierre.

continue:

Esta sentencia se utiliza para omitir los comandos restantes dentro del cuerpo del bucle que encierra la iteración actual y pasar el control del programa a la siguiente iteración del bucle. La sintaxis de la sentencia `continue` es la siguiente:

continue [n]

Donde `[n]` es un argumento opcional y puede ser mayor o igual a 1.
Cuando se proporciona `[n]`, se reanuda el n-ésimo ciclo circundante.

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

Tipos de variables:

Según el alcance:

Variables Globales: Por defecto en Bash, las variables son globales. Son visibles desde el shell que las creó o desde cualquier hijo de esa shell. Estas variables se pueden utilizar en cualquier parte del script.

Variables Locales: Se definen con la palabra clave '*local*' de manera explícita. Son visibles sólo desde el shell que las creó. Estas variables se utilizan para almacenar datos que sólo son relevantes para una sección específica del script, como dentro de una función.

Según el tipo de dato

1. Cadenas de caracteres (Strings): Las variables pueden almacenar cadenas de caracteres. Por ejemplo:

nombre="Juan"

2. Arreglos (Arrays): Bash soporta arreglos unidimensionales.

arreglo_vacio=()

materias=(DBD ING1 OO1 ISO)

nombres[0]="Pedro"

No se necesita declarar el tipo de dato de la variable cuando se crea.
El intérprete de Bash determinará el tipo de dato basándose en el valor asignado.

Shell Script no es fuertemente tipado. Aunque técnicamente puede comportarse como un lenguaje fuertemente tipado, generalmente se comporta como un lenguaje de scripting de tipo dinámico. Esto significa que no necesitas declarar el tipo de dato de una variable cuando la creas. El intérprete de Bash determinará el tipo de dato basándose en el valor que le asignes

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

Se pueden declarar de 2 formas, con y sin la palabra clave *function*:

- *function nombre { block }*
- *nombre() { block }*

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.

15. Comando cut:

El comando cut nos permite procesar las líneas de la entrada que reciba (archivo, entrada estándar, resultado de otro comando, etc) y cortar columnas o campos, siendo posible indicar cuál es el delimitador de las mismas. Investigue los parámetros que puede recibir este comando y cite ejemplos de uso

El comando `cut` en Bash se utiliza para procesar las líneas de la entrada que recibe y cortar columnas o campos.

Algunos de los parámetros que puede recibir este comando:

-b, --bytes=LIST: Esta opción enumera y selecciona únicamente los bytes de cada línea en base a los que indiquemos en LIST. LIST puede hacer referencia a un byte, un conjunto de bytes o un rango de bytes.

-c, --characters=LIST: Esta opción solo selecciona los caracteres de cada línea en base a LIST.

-d, --delimiter=DELIM: Es posible usar el carácter DELIM para usarse como delimitador de campo.

--output-delimiter=STRING: Es posible usar STRING como la cadena delimitadora de salida de resultados.

Algunos ejemplos de uso del comando `cut`:

```
# Extraer únicamente los nombres y apellidos de los alumnos

cut -d ":" -f 4,5 ejercicio-cut.txt


# Sustituir los dos puntos «:», por un espacio o un tabulador

cut -d ":" -f 4,5 --output-delimiter=$'\t' ejercicio-cut.txt


# Usar otro comando, al que le enviamos la salida standard (stdout)

cut -d ":" -f 4,5 ejercicio-cut.txt | tr -s ':' ' '
```
