Shell Scripting

Shell Script

Un shell script es un **grupo de comandos**, funciones y variables que en lugar de ejecutarlo seguidos dentro de nuestra línea de comandos, podremos colocarlos todos dentro de un archivo. La ventaja es que los shells scripts tienen la **misma sintaxis de varios lenguajes de programación** y son capaces de manejar las **mismas tareas** siguiendo la misma lógica al momento de crearlos**.**
Como todo programa, debemos de escribirlos en un archivo de texto y le otorgamos luego **permisos de ejecución**

Hecho esto, para ejecutarlos debemos escribir en la terminal el nombre del archivo **empezando con "./"** y éste se ejecutará como un comando en el sistema a menos que tengamos la carpeta la cual pertenece el archivo dentro de nuestra variable **PATH**
Para escribir nuestros programas en shell usaremos cualquier clase de editor de texto, por ejemplo: Vi, nano o cualquier otro con el cuál estemos familiarizados.

Introducción

La primera lina de nuestro script se la conoce como sheabang:
`#!/bin/bash`

Los caracteres **#!** le indican al sistema que el primer argumento que sigue en la línea es el **interprete de comandos** a utilizar para ejecutar este script dado que este lenguaje de programación es un ipo de lenguaje interpretado y no compilado. Para el ejemplo se **invocará al shell bash** para interpretar estos comandos. Otro ejemplo puede ser **python.**

Para volver nuestro archivo ejecutable, recuerden que lo podíamos hacer con el siguiente comando en nuestra terminal:

```
# chmod 755 script.sh
```

Ejemplo 1, sintaxis:

```
#!/bin/bash
clear
echo "En que directorio estoy_..."
pwd
echo "Gracias!!"
```

¿Qué hicimos?

1.- Limpiar la pantalla con el comando "clear".
2.- Usar el comando "wcho".
3.- Ejecutar el comando "pwd".
4.- Ejecutar el comando "echo".

Comando "echo"

Este comando permite **imprimir caracteres en la terminal**.

Si ejecutamos el siguiente comando; echo "hola", él imprime en pantalla hola.

```
# echo hola
hola
```

Variables

La asignación de variables se realiza simplemente indicando su nombre, un signo =, y el valor, como en el siguiente ejemplo:

```
NOMBRE="Hugo"
```

Si queremos imprimir en pantalla el valor de la variable nombre, tenemos que escribir en la terminal:

```
echo $NOMBRE
```

También podemos imprimir el valor de la variable limitándola con los caracteres "{" y "}", esto nos da mayor libertad cuando la variable es incluida dentro de una cadena de caracteres.

```
echo ${NOMBRE}
```

Echo es el comando que nos permite **enviar a la salida por pantalla el valor de la variable** pasada como argumento.

Este comando suele utilizarse si queremos testear en pantalla el progreso de un script o si deseamos que el usuario pueda ir obteniendo información mientras el script se ejecuta.

Manejo de entrada y salida

La entrada/salida puede manejarse con los comandos **echo y read**. Con ellos el script podrá interactuar con el usuario, recibiendo datos de él y mostrando en pantalla resultados.

Un ejemplo del uso de estos comandos sería el siguiente:

```
#!/bin/bash
clear
echo -n "Por favor introduzca su nombre: "
read NOMBRE
echo -n "Introduzca ahora su apellido: "
read APELLIDO
echo -e "Bienvenido a Instituto Linux\\033[0;32m" ${NOMBRE} ${APELLIDO}
"\\033[0;38m"
```

¿Qué hicimos?

1.- Borramos la pantalla
2.- Preguntamos el nombre del usuario
3.- Preguntamos el apellido
4.- Imprimimos el nombre y el apellido en color verde

También podríamos haber usado:

```
read -p "Ingrese su apellido: " APELLIDO
```

Variables predefinidas

Veamos las variables ya definidas por el shell que podremos usar en nuestros scripts.

Descripción	Variable
Número de argumentos	$#
Todos los argumentos del Shell	$*
Opciones suministradas al Shell	$-
Valor de retorno del último comando ejecutado	$?
Identificación del PID (número de proceso)	$$

Argumentos

Los argumentos sirven para pasarle a un programa o una función valores desde la línea de comando.

Como veremos en la tabla siguiente, si queremos hacer mención a **todos los argumentos** introducidos por pantalla podríamos hacer uso de la variable $*.
Si queremos indicar que debe utilizarse el primer argumento pasado tendremos que usar la variable **$1**, y así sucesivamente según el argumento que deseemos utilizar en nuestro script.

Significado	Variable
Todos los argumentos	$*
Cantidad de argumentos	$#
Nombre del script	$0
Primer argumento	$1
Segundo argumento	$2
Argumento 'enésimo'	$n

En la líneas de impresión (echo), **para imprimir el símbolo "$" (pesos) se tiene que anteponer el símbolo "\"** (contra barra), de lo contrario el shell lo va a interpretar como una variable, y si ésta existe imprimirá su contenido.

Los argumentos que se pasan en el momento de invocar el script se asignan a las variables automáticas $1, $2, $3, etc.Si creamos el script.

```
#!/bin/bash
echo "\$1 vale $1, \$2 vale $2, \$3 vale $3
```

Lo podemos mostrar de la siguiente manera.

```
# ./script perro gato loro
$1 vale perro, $2 vale gato, $3 vale loro.
```

En ocasiones se desea **asignar a una variable el valor obtenido mediante la ejecución de otro comando**. Esto se logra **encerrando el comando entre $(comando)** como podemos ver en este nuevo ejemplo:

```
horaydia=$(date)
```

Otras variables automáticas son **$0** que contiene el nombre del script, y **$#** que contiene la cantidad de argumentos ingresados durante la ejecución del script.

Expresiones

Como en cualquier lenguaje de programación, podemos evaluar expresiones y hacer depender de dicha evaluación la ejecución de un comando o secuencias de comandos

El comando test devuelve verdadero o falso como resultado (usualmente se utiliza con el comando if):

```
test "$nombre" = "Hugo" (chequea igualdad)
```

Si deseamos chequear desigualdad debemos usar el operador != como en C:

```
test "$nombre" != "Paco" (chequea desigualdad)
```

Existen otras opciones:

- **`test $variable`**: chequea si la variable contiene algún dato, en cuyo caso devuelve verdadero.
- **`test -z $variable`**: chequea si la variable no contiene datos (longitud cero), en cuyo caso devuelve verdadero.
- **`test $VARIABLE1 -eq $VARIABLE`**2: chequea igualdad entre variables numéricas enteras, si son iguales devuelve verdadero.
- **`test $VARIABLE1 -ge $VARIABLE2`**: chequea si el primer valor es mayor o igual que el segundo, en cuyo caso devuelve verdadero
- **`test $VARIABLE1 -gt $VARIABLE2`**: chequea si el primer valor es mayor que el segundo, en cuyo caso devuelve verdadero.
- **`test $VARIABLE1 -le $VARIABLE2`**: chequea si el primer valor es menor o igual que el segundo, en cuyo caso devuelve verdadero.
- **`test $VARIABLE1 -lt $VARIABLE`**2: chequea si el primer valor es menor que el segundo, en cuyo caso devuelve verdadero.
- **`test $VARIABLE1 -ne $VARIABLE`**2: chequea que los dos valores no sean iguales, en cuyo caso devuelve verdadero.
- **`test $archivo1 -nt $archivo`**2: chequea las fechas de modificación de dos archivos, devolviendo verdadero si el archivo1 es más nuevo que el archivo2.
- **`test -f $archiv`**o: chequea si el archivo existe, en caso afirmativo devuelve verdadero.

- **`test -d $archivo`**: chequea si el archivo en cuestión es un directorio y si existe, en cuyo caso devuelve verdadero.
- **`test -r $archivo`**: chequea si tenemos permiso de lectura sobre el archivo, devolviendo verdadero en caso afirmativo.
- **`test -w $archivo`**: chequea si tenemos permiso de escritura sobre el archivo, devolviendo verdadero en caso afirmativo.
- **`test -x $archivo`**: chequea si tenemos permiso de ejecución sobre el archivo, devolviendo verdadero en caso afirmativo.
- **`test -b $archivo`**: chequea si el archivo es un block device, devolviendo verdadero en caso afirmativo.
- **`test -c $archivo`**: chequea si el archivo es un character device, devolviendo verdadero en caso afirmativo.

Estructuras de control

Dentro de las estructuras de control podemos dividirlas en dos categorías principales como en cualquier lenguaje de programación:

- Basadas en valor de verdad
- Repetitivas

Basadas en valor de verdad

Este tipo de estructuras permiten tomar una decisión entre diferentes caminos basados en un valor de verdad. Si es **verdadero en el momento de ser evaluada** toma un camino que es un bloque, sino ejecutará otro bloque se conjunto de sentencias.

Nos basaremos en dos **if** y **case:**

If

La sintaxis general es común a varios lenguajes de programación:

```
if test; then
      comandos
else
      comandos
fi
```

Ejemplo:

```
#!/bin/bash
echo -n "Cuantas hojas tiene el trébol?":
read hojas
if [ $hojas = "cuatro" ]
then
      echo "Mucha suerte!!"
else
      echo " Siga participando"
fi
```

Case

Esta estructura evalúa el valor de una variable una única vez, y con esa única vez es suficiente para seguir **uno de varios bloques** o **uno por defecto.** Típicamente usado para **reemplazar if anidados.**

```
#!/bin/bash
echo -n "Ingrese numero del dado":
read NUMERO
case $NUMERO in
      2,4,6)
      echo "Par";;
      1,3,5)
      echo "Impar";;
      *)
      echo "Eso no es un dado";;
esac
```

Estructura de bucle

Estas estructuras se utilizan para ejecutar un bloque de instrucciones un número dado ó un número de veces hasta que la condición se cumpla. Llamaremos bucle o ciclo a todo proceso que se repite un cierto número de veces dentro de un script ó programa.

Dentro de estas estructuras encontramos **while** la cual ejecutara un número infinito de veces hasta que se cumpla una condición:

```
while sleep 3
do
      echo hola mundo.
done
```

O tendremos for, la cual se ejecutará un número finito de veces. Este for es muy parecido al de python ya que se ejecutara un número “infinito” de veces; pero cada iteración se guardará en una variable.

```
for file_name in `ls /`
do
      file ${file_name}
done
```