

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

El Shell Scripting es una forma de programación que se realiza utilizando comandos de Shell en sistemas operativos Unix-like o en el símbolo de sistema en Windows.

Un script de Shell es un archivo de texto que contiene una serie de comandos que se ejecutan en secuencia de manera automática, lo que permite automatizar tareas y procesos en el SO.

Se orientan a:

- **Automatización de tareas repetitivas:** Los scripts pueden utilizarse para automatizar tareas que se realizan con frecuencia, como la copia de archivos, generación de informes, administración de usuarios, descarga de archivos, etc.
- **Administración del sistema:** Los administradores de sistemas utilizan scripts para configurar, mantener y monitorear sistemas, redes y servidores. Por ejemplo, se pueden escribir scripts para realizar copias de seguridad, configurar Firewalls, administrar registros de registro y realizar actualizaciones del sistema.
- **Procesamiento de datos:** Los scripts pueden utilizarse para procesar y manipular datos en archivos. Esto es útil en aplicaciones como el análisis de registros, la conversión de formatos de archivos y la extracción de información relevante de conjuntos de datos.
- **Automatización de tareas de desarrollo:** Los desarrolladores pueden utilizar scripts para automatizar tareas relacionadas con el desarrollo de Software, como la compilación de código, la ejecución de pruebas, la implementación y la gestión de repositorios de código fuente.
- **Interacción con aplicaciones y servicios:** Los scripts pueden interactuar con aplicaciones y servicios a través de la línea de comandos, lo que permite automatizar flujos de trabajo que involucran múltiples programas o servicios.

En cuanto a la compilación de scripts, en el contexto de Shell Scripting, los scripts no necesitan ser compiladores como lo haría un lenguaje de programación compilado. En cambio, los scripts de Shell son interpretados por el intérprete de Shell, lo que significa que el código se ejecuta directamente línea por línea sin necesidad de una compilación previa.

La razón principal por la que los scripts de Shell no se compilan es la flexibilidad y la simplicidad que ofrecen. Esto permite a los usuarios y administradores de sistemas crear, modificar y ejecutar rápidamente scripts sin la necesidad de herramientas de desarrollo adicionales. Además, dado que los scripts suelen consistir en una secuencia de comandos que ya están disponibles en el sistema operativo, no es necesario un proceso de compilación para generar un ejecutable separado. Esto hace que los scripts de Shell sean accesibles y fáciles de usar para una amplia gama de usuarios.

2) Investigar la funcionalidad de los comandos echo y read.

Echo: Imprime en pantalla lo que se le adjunte.

Read: Lee una línea mediante la entrada estándar y almacenarla en una variable.

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

Los comentarios se indican con #.

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

Solo existen los Strings y Arrays.

Para declarar una variable se utiliza **nombre = "valor"**

Para hacer referencia a una variable se utiliza **\$nombre**

Se puede utilizar **{}** para evitar ambigüedades.

```
nombre = "Leo"
```

```
echo ${nombre}
```

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:

```
#!/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 ser ejecutado.

```
chmod 555 mostrar.sh
```

- b) Ejecutar el archivo creado de la siguiente manera: ./mostrar.sh
- c) ¿Qué resultado visualiza?

```
root@6c8b359891ce:/ISO/Practica 3/practica-shell-script# ./mostrar.sh
Introduzca su nombre y apellido:
Leo Luna
Fecha y hora actual:
Mon Sep 23 21:40:13 UTC 2024
Su apellido y nombre es:
Luna Leo
Su usuario es: root
Su directorio actual es:
```

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

Permiten adjuntar la salida del comando al string donde se utiliza.

- 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.

```
#!/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 $nombre"
usuario=$(whoami)
echo "Su usuario es: ${usuario}"
echo "Su directorio actual es: `pwd`"
echo "Su espacio libre en disco es: `df --total | grep total`"
echo "Su directorio personal es `cat /etc/passwd | grep ${usuario} | cut -f6 -d:`"
echo "Ingrese la ruta absoluta de un directorio para ver su contenido:"
read ruta
echo "`ls ${ruta}`"
```

Código completo bajo la carpeta /Prácticas/Docker/Práctica 3

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?

Variable	Descripción
<code>\$0</code>	Contiene el nombre del script en ejecución.
<code>\$1</code> , <code>\$2</code> , ..., <code>\$N</code>	Contiene los argumentos pasados al script. <code>\$1</code> es el primer argumento, <code>\$2</code> es el segundo, y así sucesivamente. Por ejemplo, si ejecutas <code>script.sh arg1 arg2</code> , <code>\$1</code> será <code>arg1</code> y <code>\$2</code> será <code>arg2</code> .
<code>\$#</code>	Contiene el número de argumentos pasados al script.
<code>\$*</code>	Contiene todos los argumentos pasados al script como una sola cadena. Si se utilizan con comillas dobles (<code>"\$*"</code>), se consideran como un solo argumento.
<code>\$@</code>	Similar a <code>\$*</code> , pero cuando se usa con comillas dobles (<code>"\$@"</code>), cada argumento se mantiene como una palabra separada.
<code>\$?</code>	Contiene el código de salida del último comando ejecutado. Un valor de <code>0</code> generalmente indica que el comando se ejecutó correctamente, mientras que cualquier otro valor indica un error.
<code>\$HOME</code>	Contiene la ruta del directorio personal del usuario actual. Este es el directorio que se asigna a un usuario cuando inicia sesión.

5) ¿Cuál es la funcionalidad del comando `exit`? ¿Qué valores recibe como parámetro y cuál es su significado?

Exit se utiliza para terminar un script. Al ejecutarse, el script finaliza y devuelve un valor entre 0 y 255. El valor 0 indica que el script se ejecutó de forma exitosa, el resto de los valores indican un error.

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 expresión. Investigar qué tipo de operaciones se pueden utilizar.

exp EXPRESION imprime el valor de **EXPRESION** en la salida estándar. Una línea en blanco debajo separa grupos de precedencia creciente.

Tenga en cuenta que muchos operadores deben escaparse o “encomillarse” para las Shells. Las comparaciones son aritméticas si ambos argumentos son números, de lo contrario, son lexicográficas. Las coincidencias de patrones devuelven la cadena coincidente entre `\(y\)` o `null`; si no se utilizan `\(y\)`, devuelven el número de caracteres coincidentes o 0.

El estado de salida es 0 si **EXPRESION** no es nula ni 0, 1 si **EXPRESION** es nula o 0, 2 si **EXPRESION** es sintácticamente inválida y 3 si se ha producido un error.

Expresión	Descripción
ARG1 ARG2	ARG1 si no es nulo ni 0, de lo contrario ARG2.
ARG1 & ARG2	ARG1 si ninguno de los argumentos es nulo o 0, de lo contrario, 0.
ARG1 < ARG2	ARG1 es menor que ARG2.
ARG1 <= ARG2	ARG1 es menor o igual a ARG2.
ARG1 = ARG2	ARG1 es igual a ARG2.
ARG1 != ARG2	ARG1 es diferente de ARG2.
ARG1 >= ARG2	ARG1 es mayor o igual a ARG2.
ARG1 > ARG2	ARG1 es mayor que ARG2.
ARG1 + ARG2	Suma aritmética de ARG1 y ARG2.
ARG1 - ARG2	Resta aritmética de ARG1 y ARG2.
ARG1 * ARG2	Producto aritmético de ARG1 y ARG2.
ARG1 / ARG2	Cociente aritmético de ARG1 dividido por ARG2.
ARG1 % ARG2	Resto aritmético de ARG1 dividido por ARG2.
STRING : REGEXP	Coincidencia de patrón anclada de REGEXP en STRING.
match STRING REGEXP	Igual que STRING : REGEXP.
substr STRING POS LENGTH	Subcadena de STRING, comenzando en la posición POS contada desde 1, con longitud LENGTH.
index STRING CHARS	Índice en STRING donde se encuentra cualquier carácter de CHARS, o 0 si no se encuentra.
length STRING	Longitud de STRING.
+ TOKEN	Interpretar TOKEN como una cadena, incluso si es una palabra clave como 'match' o un operador como '/'.
(EXPRESSION)	Valor de la EXPRESSION.

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.

Expresión		Descripción
Expresiones		
(EXPR)		EXPR es verdadera
! EXPR		EXPR es falsa
Strings		
-z STR		La longitud de STR es cero
-n STR	STR	La longitud de STR es distinta de cero
STR1 =, !=, <, > STR2		Comparar strings
Enteros		
INT1 -eq INT2		INT1 es igual a INT2
INT1 -ne INT2		INT1 no es igual a INT2
INT1 -lt INT2		INT1 es menor estricto que INT2
INT1 -le INT2		INT1 es menor o igual que INT2
INT1 -gt INT2		INT1 es mayor estricto que INT2
INT1 -ge INT2		INT1 es mayor o igual que INT2
Archivos		
-e FILE		FILE existe
-d FILE		FILE existe y es un directorio
-f FILE		FILE existe y es un archivo regular
-s FILE		FILE existe y tiene un tamaño mayor que 0.

8) Estructuras de control. Investigue la sintaxis de las siguientes estructuras de control incluidas en Shell Scripting:

- If
- Case
- While
- For
- Select