Introducción. Scripts. Automatizando tareas

¿Qué es un script?

Supongamos que tenemos una serie de tareas que realizamos repetidamente en la computadora, como:

- Copiar archivos de una carpeta a otra.
- Renombrar un grupo de imágenes.
- Convertir archivos de un formato a otro.

Un script es un archivo, un conjunto de instrucciones escritas en un lenguaje especial (secuencia de comandos) que le indica al sistema operativo qué hacer paso a paso. En lugar de ejecutar cada comando individualmente, podemos escribirlos a todos en un script y ejecutarlos con un solo clic.

Los scripts son muy útiles para:

- Automatizar tareas repetitivas: Ahorra tiempo y esfuerzo al automatizar tareas comunes.
- Simplificar tareas complejas: Divide tareas grandes en pasos más pequeños y manejables.
- Personalizar nuestro entorno, configurando el sistema operativo a gusto.
- Aprender a programar: Los scripts son una excelente forma de iniciarse en el mundo de la programación.

Extensiones de archivo comunes:

Los scripts se guardan en archivos de texto con extensiones especiales que indican el tipo de lenguaje utilizado y el sistema operativo para el que están diseñados. Algunas extensiones comunes son:

- .bat y .cmd: Se utilizan para scripts en sistemas operativos Windows.
- .sh: Se utiliza para scripts en sistemas operativos Linux y macOS (Bash).

En este documento nos enfocaremos en los scripts de Bash (.sh) que utilizados en sistemas Linux.

Creación de Scripts en Bash

Estructura básica de un script:

Un script en Bash es simplemente un archivo de texto que contiene una serie de comandos. Estos comandos son los mismos que se utilizan en la terminal de Linux. Para crear un script, se puede utilizar cualquier editor de texto plano, como nano, vim o gedit.

Ejemplo:

```
#!/bin/bash
echo "Que tal?"
```

Este script, al ser ejecutado, mostrará el mensaje "Que tal?" en la terminal.

Shebang (#!/bin/bash):

La primera línea de un script en Bash debe ser *#!/bin/bash*.
Esta línea, conocida como "shebang", indica al sistema operativo que el script debe ser interpretado con Bash. Sin esta línea, el sistema no sabrá cómo ejecutar el script.

Comentarios (#):

Los comentarios son líneas de texto que son ignoradas por el intérprete de Bash. Se utilizan para documentar el código y hacerlo más legible. Cualquier línea que comience con el símbolo # es un comentario.

Ejemplo:

```
#!/bin/bash
# Este script muestra un saludo
echo "Que tal?" # Imprime "Que tal?" en la terminal
```

Comandos:

Un script en Bash puede contener cualquier comando válido que se pueda ejecutar en la terminal. Repasando, algunos comandos comunes son:

- **echo:** Muestra texto en la terminal.
- **ls:** Lista los archivos y directorios.
- **cd:** Cambia el directorio actual.
- **mkdir:** Crea nuevos directorios.
- **rm:** Elimina archivos y directorios.
- **cp:** Copia archivos y directorios.
- **mv:** Mueve o renombra archivos y directorios.
- **grep:** Busca patrones en archivos.
- **find:** Busca archivos y directorios.
- **date:** Muestra la fecha y hora actual.
- **whoami:** Muestra el nombre del usuario actual.

Ejemplo:

```
#!/bin/bash
# Este script lista los archivos del directorio actual
ls -l
```

Permisos de ejecución (chmod +x):

Para que un script pueda ser ejecutado, se le deben otorgar permisos de ejecución. Esto se puede hacer con el comando chmod:

chmod +x nombre_del_script.sh

Donde nombre_del_script.sh es el nombre del archivo del script.

Ejemplo:

```
chmod +x mi_script.sh
```

Ejecución de un script:

Una vez que el script tiene permisos de ejecución, se puede ejecutar de las siguientes maneras:

- **./nombre_del_script.sh:** Especificando la ruta completa al script.
- **bash nombre_del_script.sh:** Utilizando el comando bash seguido del nombre del script.

Variables

Las variables son elementos fundamentales en la programación, ya que permiten almacenar y manipular datos dentro de un script. En Bash, las variables se utilizan para guardar información como números, texto, nombres de archivos y otros valores. Este instructivo proporciona una guía detallada sobre el uso de variables en Bash, incluyendo su definición, asignación, tipos de datos, restricciones de nombres y variables de entorno.

Definición y Asignación:

Para definir una variable en Bash, se utiliza el siguiente formato:

nombre_variable=valor_variable

Es importante tener en cuenta que no se deben dejar espacios en blanco alrededor del signo =.

Ejemplo:

```
nombre="Juan"

edad=30
precio=19.99
```

En este ejemplo, se definen tres variables: nombre con el valor "Juan", edad con el valor 30 y precio con el valor 19.99.

Referencia:

Para acceder al valor almacenado en una variable, se utiliza el símbolo $ seguido del nombre de la variable.

Ejemplo:

```
echo $nombre  # Imprime "Juan"
echo $edad    # Imprime "30"
```

Para evitar ambigüedades, especialmente cuando se utiliza la variable dentro de una cadena de texto, se puede encerrar el nombre de la variable entre llaves {}.

Ejemplo:

```
echo "Hola, ${nombre}!"  # Imprime "Hola, Juan!"
```

Tipos de datos:

Bash no tiene una tipificación de datos estricta como otros lenguajes de programación. Todas las variables se tratan como cadenas de texto, pero también se pueden utilizar para almacenar números y realizar operaciones aritméticas.

Ejemplo:

```
numero1=10
numero2=5
suma=$((numero1 + numero2))
echo $suma  # Imprime "15"
```

Restricciones de nombres:

Al nombrar variables en Bash, se deben seguir las siguientes reglas:

- Deben comenzar con una letra o un guion bajo (_).
- Pueden contener letras, números y guiones bajos.
- No se permiten espacios en blanco ni caracteres especiales.

Ejemplos de nombres válidos:

```
nombre
edad_usuario
_contador
```

Ejemplos de nombres inválidos:

```
1nombre
nombre usuario
mi-variable
```

Variables de entorno:

Las variables de entorno son variables predefinidas por el sistema que contienen información sobre el entorno de ejecución. Algunas variables de entorno comunes son:

- HOME: Directorio de inicio del usuario.
- PATH: Lista de directorios donde se buscan los comandos ejecutables.
- USER: Nombre de usuario actual.
- PWD: Directorio de trabajo actual.

- SHELL: Shell actual que se está utilizando.
- TERM: Tipo de terminal.

Para acceder al valor de una variable de entorno, se utiliza el símbolo $ seguido del nombre de la variable, al igual que con las variables definidas por el usuario.

Ejemplo:

```
echo $HOME  # Imprime el directorio de inicio del usuario
echo $PATH  # Imprime la lista de directorios en PATH
```

Recomendaciones:

- Utilizar nombres descriptivos para las variables.
- Utilizar mayúsculas para las variables de entorno y minúsculas para las variables definidas por el usuario.
- Evitar el uso de nombres de variables que coincidan con comandos o palabras clave de Bash.

Operaciones Aritméticas

Bash, además de ser un potente intérprete de comandos, ofrece la capacidad de realizar operaciones aritméticas. Esto resulta útil para automatizar cálculos, manipular datos numéricos y realizar tareas que involucren operaciones matemáticas en scripts. Este instructivo proporciona una guía detallada sobre las operaciones aritméticas en Bash, incluyendo los operadores básicos, el incremento y decremento, y las diferentes formas de evaluar expresiones aritméticas.

1. Operadores básicos:

Bash soporta los operadores aritméticos básicos para realizar operaciones con números enteros:

- +: Suma
- -: Resta
- *: Multiplicación
- /: División entera (el resultado se trunca al entero más cercano)
- %: Módulo (resto de la división entera)
- **: Potencia

Ejemplos:

```
a=10
b=3
suma=$((a + b))    # suma = 13
resta=$((a - b))   # resta = 7
producto=$((a * b))  # producto = 30
division=$((a / b))  # division = 3
modulo=$((a % b))   # modulo = 1
potencia=$((a ** b)) # potencia = 1000
```

2. Incremento y decremento:

Los operadores de incremento (++) y decremento (--) se utilizan para aumentar o disminuir el valor de una variable en 1.

- ++: Incremento
- --: Decremento

Estos operadores pueden utilizarse en su forma prefijo (++variable o --variable) o sufijo (variable++ o variable--). En la forma prefijo, la variable se incrementa o decrementa antes de que se evalúe su valor, mientras que en la forma sufijo, la variable se incrementa o decrementa después de que se evalúe su valor.

Ejemplos:

```
a=5
b=$((++a))  # a = 6, b = 6
c=$((a++))  # a = 7, c = 6
d=$((--a))  # a = 6, d = 6
e=$((a--))  # a = 5, e = 6
```

3. Expresiones aritméticas:

Existen diferentes formas de evaluar expresiones aritméticas en Bash:

- expr: El comando expr permite evaluar expresiones aritméticas.

```
resultado=$(expr $a + $b)
```

- $((...)): Esta es la forma más común y recomendada para evaluar expresiones aritméticas.

```
resultado=$((a + b))
```

- let: El comando let permite realizar asignaciones y operaciones aritméticas en una sola línea.

```
let "resultado = a + b"
```

Recomendaciones:

- Utilizar paréntesis para agrupar operaciones y asegurar el orden de evaluación deseado.
- Utilizar la forma $((...)) para evaluar expresiones aritméticas, ya que es la más legible y eficiente.
- Tener en cuenta que la división entera trunca el resultado al entero más cercano.
- Utilizar el operador de módulo (%) para obtener el resto de una división entera.

Manipulación de Cadenas

Bash ofrece una variedad de herramientas para manipular cadenas de texto, lo que resulta esencial para procesar información, formatear datos y realizar tareas de administración de sistemas. Este instructivo proporciona una guía detallada sobre las técnicas de manipulación de cadenas en Bash, incluyendo la extracción de subcadenas, el borrado y reemplazo de subcadenas, la conversión a mayúsculas y minúsculas, y la concatenación.

1. Extracción de subcadenas:

Para extraer una porción de una cadena de texto, se utiliza la siguiente sintaxis:

${cadena:posicion:longitud}
Donde:

- cadena: es la variable que contiene la cadena de texto.
- posicion: es el índice del primer carácter a extraer (comenzando en 0).
- longitud: es el número de caracteres a extraer.

Ejemplo:

```
cadena="Hola Mundo"
subcadena=${cadena:5:5}  # subcadena = "Mundo"
```

2. Borrado de subcadenas:

Para eliminar una subcadena de una cadena de texto, se utilizan las siguientes sintaxis:

- ${cadena#subcadena}: Elimina la coincidencia más corta de subcadena desde el principio de cadena.
- ${cadena##subcadena}: Elimina la coincidencia más larga de subcadena desde el principio de cadena.

Ejemplos:

```
cadena="www.ejemplo.com"
nueva_cadena=${cadena#www.}    # nueva_cadena = "ejemplo.com"
nueva_cadena=${cadena##www.}   # nueva_cadena = "com"
```

3. Reemplazo de subcadenas:

Para reemplazar una subcadena por otra, se utilizan las siguientes sintaxis:

- ${cadena/buscar/reemplazar}: Reemplaza la primera coincidencia de buscar con reemplazar.
- ${cadena//buscar/reemplazar}: Reemplaza todas las coincidencias de buscar con reemplazar.

Ejemplos:

```
cadena="Hola Mundo Mundo"
nueva_cadena=${cadena/Mundo/Linux}   # nueva_cadena = "Hola Linux Mundo"
nueva_cadena=${cadena//Mundo/Linux}  # nueva_cadena = "Hola Linux Linux"
```

4. Conversión a mayúsculas/minúsculas:

Para convertir una cadena de texto a mayúsculas o minúsculas, se utilizan las siguientes sintaxis:

- ${cadena^^}: Convierte cadena a mayúsculas.
- ${cadena,,}: Convierte cadena a minúsculas.

Ejemplos:

```
cadena="Hola Mundo"
mayusculas=${cadena^^}  # mayusculas = "HOLA MUNDO"
minusculas=${cadena,,}  # minusculas = "hola mundo"
```

5. Concatenación:

Para concatenar (unir) cadenas de texto, simplemente se colocan una al lado de la otra.

Ejemplo:

```
cadena1="Hola"
cadena2="Mundo"
nueva_cadena="$cadena1 $cadena2"  # nueva_cadena = "Hola Mundo"
```

Recomendaciones:

- Utilizar comillas dobles para encerrar las cadenas de texto que contienen espacios en blanco o caracteres especiales.
- Recordar que la posición de los caracteres en una cadena comienza en 0.
- Experimentar con las diferentes técnicas de manipulación de cadenas para comprender su funcionamiento.

Entrada del Usuario

La interacción con el usuario es un componente esencial en muchos scripts de Bash. Permitir que el usuario ingrese datos por teclado hace que los scripts sean más flexibles y dinámicos. Este instructivo se centra en el comando read, la herramienta principal en Bash para obtener la entrada del usuario, y cómo utilizarla eficazmente en tus scripts.

1. El comando read:

El comando read permite que un script pause su ejecución y espere a que el usuario ingrese información a través del teclado. La información ingresada se almacena en una variable para su uso posterior.

Sintaxis básica:

read variable

Cuando se ejecuta read, el script se detiene y espera a que el usuario escriba algo y presione Enter. Lo que el usuario escribe se guarda en la variable especificada.

2. Mostrando un mensaje al usuario:

Para que la interacción sea más clara, puedes mostrar un mensaje al usuario indicándole qué información se espera que ingrese. Esto se logra con la opción -p:

Sintaxis:

read -p "Mensaje: " variable

Ejemplo:

read -p "Introduce tu nombre: " nombre_usuario
Este comando mostrará "Introduce tu nombre: " en la terminal y esperará a que el usuario ingrese su nombre. El nombre ingresado se almacenará en la variable nombre_usuario.

3. Ocultando la entrada del usuario:

Cuando se necesita que el usuario ingrese información sensible, como una contraseña, es importante que lo que escribe no se muestre en la pantalla. La opción -s permite ocultar la entrada del usuario:

Sintaxis:

read -s variable
Ejemplo:

read -s contraseña
En este caso, cualquier cosa que el usuario escriba no será visible en la terminal, lo que proporciona un nivel básico de seguridad para la entrada de contraseñas.

4. Asignación a variables:

La información que el usuario ingresa con read se almacena directamente en la variable especificada en el comando. Esta variable puede ser utilizada luego en el script para realizar diferentes acciones.

Ejemplo:

```
#!/bin/bash
echo "Hola, ¿cómo te llamas?"
read nombre
echo "Mucho gusto, $nombre. ¿Qué edad tienes?"
read edad

echo "Ok, $nombre, tienes $edad años."
```

En este script, se pide al usuario su nombre y edad, y luego se utiliza esa información para mostrar un mensaje personalizado.

Recomendaciones:

- Utilizar la opción -p para proporcionar instrucciones claras al usuario.
- Utilizar la opción -s para la entrada de datos sensibles.
- Validar la entrada del usuario para asegurarse de que cumple con los requisitos del script.
- Combinar read con estructuras condicionales (if, elif, else) para crear scripts interactivos.