# ISO practica 3.

1) Shell scripting se refiere a la creación y ejecución de secuencias de comandos (scripts) en un shell de línea de comandos en sistemas operativos Unix y Unix-like, como GNU/Linux. Estos scripts son programas que están escritos en lenguajes de scripting de shell, como Bash, sh, zsh, entre otros, y se utilizan para automatizar tareas, ejecutar comandos en secuencia y realizar una variedad de funciones.

Tareas que realiza:

* Automatización de tareas repetitivas: Los scripts de shell pueden automatizar tareas que deben realizarse de manera repetitiva, como la copia de archivos, la organización de datos, la programación de tareas, etc.
* Administración del sistema: Se utilizan para administrar y configurar sistemas, realizar copias de seguridad, monitorear recursos, crear usuarios, instalar software, entre otros.
* Procesamiento de datos: Los scripts pueden procesar archivos de texto, filtrar datos, generar informes, analizar registros, etc.
* Interacción con programas y servicios: Pueden interactuar con otros programas y servicios a través de la línea de comandos, lo que les permite realizar tareas como la automatización de despliegues, la gestión de contenedores, la comunicación con bases de datos y servicios web, etc.

Los scripts de shell no necesitan compilarse. A diferencia de los lenguajes de programación compilados, como C o C++, los scripts de shell son interpretados por el shell o el intérprete de comandos directamente en tiempo de ejecución. Esto significa que puedes escribir un script de shell y ejecutarlo sin necesidad de compilarlo previamente.

Las principales razones por las que los scripts de shell no se compilan son:

* Rapidez y flexibilidad: La interpretación en tiempo de ejecución permite una edición rápida y una depuración más sencilla, ya que los cambios en el script se reflejan de inmediato en la siguiente ejecución.
* Portabilidad: Los scripts de shell son altamente portátiles, ya que se pueden ejecutar en cualquier sistema Unix-like que tenga el intérprete de comandos adecuado instalado. No es necesario compilarlos para una plataforma específica.
* Acceso a comandos del sistema: Los scripts de shell pueden aprovechar los comandos y utilidades del sistema operativo sin necesidad de vincularse con bibliotecas específicas.

2) los comentarios se especifican con #

La declaración de una variable se realiza poniendo el nombre de una variable seguido del signo “=” y luego se coloca el valor que se quiere dentro de la misma

Para acceder a ellas se usa el signo “$”.

4) En un script de shell, puedes acceder a los parámetros que se le pasan cuando se invoca utilizando variables especiales. Aquí están algunas de las variables más comunes que se utilizan para acceder a estos parámetros:

* $# (número de parámetros): Esta variable contiene el número de parámetros que se pasan al script. Por ejemplo, si ejecutas un script con tres parámetros, $# contendrá el valor 3.
* $ (todos los parámetros como una sola cadena):\* Esta variable contiene todos los parámetros pasados al script como una sola cadena. Los parámetros se separan por espacios. Puedes usar esta variable para acceder a todos los parámetros juntos.
* $? (código de salida del último comando ejecutado): Esta variable contiene el código de salida del último comando ejecutado. Un valor de 0 generalmente indica éxito, mientras que un valor distinto de 0 indica un error. Puedes utilizar esta variable para verificar si el último comando se ejecutó correctamente.
* $1, $2, $3, ... (parámetros individuales): Estas variables contienen los parámetros individuales pasados al script. $1 contiene el primer parámetro, $2 contiene el segundo, y así sucesivamente. Por ejemplo, si ejecutas un script como ./mi\_script.sh parametro1 parametro2, entonces $1 contendrá "parametro1" y $2 contendrá "parametro2".
* $0 (nombre del script): Esta variable contiene el nombre del script en sí mismo, es decir, cómo se llamó el script al invocarl

5) El comando exit en un script de shell se utiliza para finalizar la ejecución del script y salir de él. Puedes usar exit para indicar el estado de salida del script, es decir, si el script se ejecutó con éxito o si ocurrió algún error.

El comando exit puede recibir un valor como parámetro, que se conoce como "código de salida" o "estatus de salida". Este código de salida es un número entero que se utiliza para indicar el resultado de la ejecución del script. Los valores típicos para el código de salida son:

* 0: Generalmente se utiliza para indicar que el script se ejecutó con éxito, es decir, sin errores.
* Valores distintos de 0 (por ejemplo, 1, 2, -1, etc.): Se utilizan para indicar que ocurrió algún tipo de error durante la ejecución del script. El valor específico puede tener un significado particular dependiendo del script y de las convenciones utilizadas. Por ejemplo, algunos scripts pueden usar el código de salida 1 para indicar un error de entrada, 2 para un error de archivo no encontrado, etc.

El código de salida se almacena en una variable especial llamada $?, que puedes consultar después de la ejecución del comando exit para determinar el estado de salida del script.

6) El comando expr en sistemas Unix y Linux se utiliza para realizar operaciones de evaluación de expresiones. Puedes utilizar expr para llevar a cabo diversas operaciones, incluyendo las siguientes:

* Operaciones aritméticas: Puedes realizar operaciones matemáticas básicas, como suma, resta, multiplicación y división, con expr.
* Comparaciones numéricas: expr permite comparar números y evaluar expresiones numéricas.
* Expresiones booleanas: Puedes utilizar expr para evaluar expresiones booleanas, como AND (&&) y OR (||).
* Longitud de cadenas: expr puede utilizarse para calcular la longitud de una cadena de texto.
* Extracción de subcadenas: Puedes usar expr para extraer subcadenas de una cadena más grande.
* Concatenación de cadenas: Puedes concatenar cadenas de texto con expr. Por ejemplo:

Es importante tener en cuenta que expr tiene algunas limitaciones en cuanto a la sintaxis y las operaciones que puede realizar. No es tan versátil como otros lenguajes de programación o utilidades de scripting.

7) El comando test o [ ] en GNU/Linux se utiliza para evaluar expresiones condicionales y generar un valor de retorno, que generalmente es true o false. Estas expresiones pueden ser usadas para realizar una variedad de pruebas, incluyendo evaluación de archivos, evaluación de cadenas de caracteres y evaluaciones numéricas. A continuación, se describen algunos tipos de expresiones que pueden ser usadas con el comando test o [ ]:

1. Evaluación de archivos:

-e archivo: Verifica si un archivo o directorio existe.

-f archivo: Comprueba si un archivo regular existe.

-d directorio: Comprueba si un directorio existe.

-s archivo: Verifica si un archivo tiene un tamaño mayor que cero.

-r archivo: Comprueba si un archivo es legible.

-w archivo: Comprueba si un archivo es escribible.

-x archivo: Comprueba si un archivo es ejecutable.

2. Evaluación de cadenas de caracteres:

cadena1 = cadena2: Comprueba si las dos cadenas son iguales.

cadena1 != cadena2: Comprueba si las dos cadenas son diferentes.

-z cadena: Comprueba si una cadena está vacía (longitud cero).

-n cadena: Comprueba si una cadena no está vacía.

3. Evaluaciones numéricas:

n1 -eq n2: Comprueba si dos números son iguales.

n1 -ne n2: Comprueba si dos números son diferentes.

n1 -lt n2: Comprueba si n1 es menor que n2.

n1 -le n2: Comprueba si n1 es menor o igual a n2.

n1 -gt n2: Comprueba si n1 es mayor que n2.

n1 -ge n2: Comprueba si n1 es mayor o igual a n2.

8) if [ condición ]; then

# Código a ejecutar si la condición es verdadera

else

# Código a ejecutar si la condición es falsa

fi

case "$variable" in

valor1)

# Código a ejecutar si variable coincide con valor1

;;

valor2)

# Código a ejecutar si variable coincide con valor2

;;

\*)

# Código a ejecutar si no se cumple ninguna de las condiciones anteriores

;;

Esac

while [ condición ]; do

# Código a ejecutar mientras la condición sea verdadera

Done

for variable in elemento1 elemento2 elemento3; do

# Código a ejecutar para cada elemento en la lista

Done

select opcion in opcion1 opcion2 opcion3; do

case $opcion in

opcion1)

# Código a ejecutar para la opción 1

;;

opcion2)

# Código a ejecutar para la opción 2

;;

opcion3)

# Código a ejecutar para la opción 3

;;

\*)

echo "Opción no válida"

;;

esac

done

9) La sentencia break se utiliza para salir inmediatamente de un bucle (como un bucle for, while o until) antes de que se complete debido a una condición específica. Cuando se encuentra una sentencia break, el bucle se interrumpe y la ejecución continúa después del bucle.

La sentencia continue se utiliza para omitir la iteración actual de un bucle y continuar con la siguiente iteración. En lugar de salir completamente del bucle, como lo hace break, continue permite continuar con las siguientes iteraciones del bucle.

10) En los scripts de shell (como Bash), las variables pueden clasificarse en varios tipos y no se requiere una declaración explícita del tipo de variable, lo que significa que Bash no es un lenguaje de programación fuertemente tipado. Aquí están los tipos de variables comunes en Bash:

Variables de cadena (string variables): Estas variables almacenan cadenas de caracteres, como texto. No es necesario declarar explícitamente que una variable es una variable de cadena; simplemente le asignas un valor de texto.

Variables numéricas (numeric variables): Bash trata todas las variables como cadenas de caracteres de forma predeterminada, pero puedes realizar operaciones numéricas si utilizas la aritmética de shell adecuada.

Variables de arreglo (array variables): Sí, puedes definir arreglos en Bash. Los arreglos son una colección ordenada de elementos, y se pueden acceder y manipular individualmente o en grupo. Para definir un arreglo:

Variables booleanas (boolean variables): Bash no tiene tipos de datos booleanos nativos, pero puedes usar valores como 0 para "falso" y 1 para "verdadero" o usar cadenas como "true" y "false" para representar valores booleanos.

En resumen, en los scripts de shell como Bash, puedes trabajar con variables de cadena, variables numéricas, arreglos y simular variables booleanas utilizando valores numéricos o cadenas. La flexibilidad en el manejo de tipos de variables es una característica distintiva de los lenguajes de scripting de shell.

11) Si se pueden definir funciones con la sintaxis <nombredelafuncion>(){#codigo}

Tambien se pueden pasar parámetros en su llamado, <nombre de la función> #parametros.

Se accede a ellos a través de “$<numero>” para saber cuales son los parámetros utilizamos “$@”.

15) El comando **cut** en GNU/Linux se utiliza para procesar líneas de entrada y recortar columnas o campos específicos de esas líneas. El delimitador predeterminado es el carácter de tabulación (**\t**), pero puedes especificar un delimitador personalizado si lo deseas. Aquí están los parámetros más comunes que puede recibir el comando **cut** junto con ejemplos de uso:

**Sintaxis básica:**

bashCopy code

cut [OPCIONES] ARCHIVO

**Opciones más comunes:**

1. **-f, --fields**: Especifica qué campos o columnas deseas extraer de cada línea.

Ejemplo: Supongamos que tenemos un archivo llamado **datos.txt** con líneas de texto delimitadas por comas y queremos extraer el segundo campo de cada línea.

bashCopy code

cut -d ',' -f 2 datos.txt

1. **-d, --delimiter**: Especifica el delimitador utilizado en las líneas de entrada. Por defecto, el delimitador es el carácter de tabulación (**\t**).

Ejemplo: Si las líneas están delimitadas por el carácter **:** y queremos extraer el tercer campo de cada línea:

bashCopy code

cut -d ':' -f 3 archivo.txt

1. **--complement**: Invierte la selección, es decir, selecciona todas las columnas excepto las especificadas.

Ejemplo: Si queremos extraer todas las columnas excepto la primera de un archivo delimitado por comas:

bashCopy code

cut -d ',' --complement -f 1 archivo.txt

1. **-c, --characters**: En lugar de campos basados en delimitadores, puedes especificar un rango de caracteres que deseas extraer de cada línea.

Ejemplo: Si queremos extraer los caracteres del 1 al 5 de cada línea:

bashCopy code

cut -c 1-5 archivo.txt

1. **-s, --only-delimited**: Suprime las líneas que no contienen un delimitador.

Ejemplo: Si solo queremos extraer el tercer campo de líneas que contienen el delimitador **:**:

bashCopy code

cut -d ':' -f 3 -s archivo.txt

Estos son algunos de los usos más comunes del comando **cut** en GNU/Linux. Puedes combinar estas opciones para adaptar el comando a tus necesidades específicas y procesar líneas de entrada de acuerdo a tus requisitos.