BASH 03.03 - Shell script

Información sobre el documento

El objetivo de este documento es enseñar, compartir conocimientos para facilitar el aprendizaje. Este documento es mejorable, y será actualizado si es preciso.

1. - Introducción

El objetivo de este documento es practicar en la creación de shell scripts bash.

En este documento vamos a trabajar con:

1. Redirecciones de salida: estándar y error.
2. Obtener la salida de un comando para trabajar con ella dentro del script.
3. Trabajar con aritmética
4. Bucles FOR simples

Para sacar el máximo partido al documento, se recomienda leer el apartado donde se presentan los contenidos en el material didáctico proporcionado en el sitio web de educatica o de terceros autores.

2. - Actividades

No mostrar información no solicitada

Comprobar si existe un usuario

Copia el script llamado **userInfo03.sh** como **userInfo04.sh** y vamos a modificarlo para añadir nuevas características.

Este script deberá realizar las mismas tareas que el script userInfo03.sh solo que esta vez, **vamos a evitar que se muestre información de salida o de error por pantalla**. Como esa información no la vamos a utilizar, podemos redirigir la salida del comando al fichero del sistema que se utiliza para esto (fichero vacío).

1. Comprueba que se pasa como parámetro el nombre del usuario.
    1. Si no se pasa parámetro se deberá mostrar un mensaje de error y salir con un código de error -1.
2. Comprueba que el usuario existe. **Utiliza el comando id con las redirecciones de salida propuestas**.
    1. Si el usuario no existe se deberá mostrar un mensaje de error y salir con un código de error -2.
3. Crea una nueva variable llamada homeDir cuyo contenido será la ruta del directorio personal del usuario que se ha insertado por parámetro. Vamos a suponer que el directorio personal de cualquier usuario está en el directorio /home y tiene como nombre el del usuario.
4. Comprobar que el directorio personal del usuario existe.
    1. Si no existe, se deberá mostrar un mensaje de error y salir con un código de error -3.
5. Si las comprobaciones han sido correctas, entonces mostraremos la información del script
    1. Mostrará en pantalla el siguiente mensaje: “Nombre del usuario: “ seguido del nombre del usuario insertado por teclado.
    2. Mostrará en pantalla información de identificación del usuario cuyo nombre se ha insertado por teclado. Esta información de identificación estará compuesta por el nombre del usuario, el UID, el grupo principal al que pertenece con su GID y los grupos a los que pertenece el usuario. Hay un comando que nos da toda esta información.
    3. Muestre el siguiente mensaje en pantalla “Directorio personal del usuario “ seguido de la ruta del directorio personal del usuario insertado por teclado. Esta información está almacenada en la variable homeDir creada anteriormente.
    4. Muestre en pantalla información de permisos del directorio personal del usuario insertado por teclado, no de su contenido.
    5. Muestre en pantalla el contenido del directorio personal del usuario insertado por teclado, sin información extendida de permisos.

Una vez creado el script, deberás configurarlo para que pueda ser ejecutado y ejecutarlo desde la terminal.

Redirecciones de salida: estándar y error

En muchas ocasiones querremos almacenar la salida de un comando en un fichero para poderla utilizar más tarde o tenerla de forma persistente.

Conocemos las redirecciones simple y doble, sus similitudes y diferencias. Ambas crean el fichero si no existe, pero si el fichero ya existía la simple sobrescribe su contenido con los nuevos valores y la doble añade al final de fichero los datos, manteniendo los anteriores.

```
comando > ruta_fichero
comando >> ruta_fichero
```

Hasta ahora hemos redireccionado solo la salida estándar de un comando a un fichero. Sin embargo, los comandos y aplicaciones pueden utilizar otro dispositivo de salida para mostrar información: la salida de error.

La salida de error es utilizada por una aplicación cuando quiere mostrar un error y no una salida de datos propia de una ejecución correcta. Esta salida de errores también se puede redireccionar, pero para ello se debe especificar el descriptor de fichero de esta salida, cuyo identificador es el número 2.

```
comando 2> ruta_fichero
```

Esta redirección almacenará la salida de errores en el fichero ruta_fichero. Si el comando muestra algo por la estándar se mostrará en pantalla, no se almacenará en el fichero.

Habitualmente lo que se suele hacer en un script es evitar que un comando muestre información de error al usuario, controlando el error de forma interna. Por ejemplo con una sentencia condicional, como el if, se puede comprobar una condición de error y actuar en consecuencia.

Para estos casos, se utiliza un fichero especial que en realidad no almacena nada y que solo se suele utilizar para estos casos. Este fichero es /dev/null.

```
comando 2> /dev/null
```

La salida de errores del comando no se mostrará en pantalla, se almacenará en un fichero que en realidad no almacena nada. Es como un contenedor siempre vacío.

Se pueden realizar varias combinaciones en bash con estas redirecciones, que se comentan a continuación a modo de resumen.

```
Comando > ruta_fichero
```

Redirige la salida estándar al fichero ruta_fichero. La salida de error se mostrará en pantalla.

```
Comando 2> ruta_fichero
```

Redirige la salida de error al fichero ruta_fichero. La salida estándar se mostrará en pantalla.

```
Comando > ruta_fichero1 2> ruta_fichero2
```

Redirige la salida estándar al fichero ruta_fichero1 y la salida de error se redirige a otro fichero ruta_fichero2

```
Comando > ruta_fichero 2>&1
Comando &> ruta_fichero
```

Redirige la salida estándar y la de error al mismo fichero ruta_fichero

Obtener salida de un comando

Obtener la ruta del directorio personal de un usuario

Copia el script llamado **userInfo04.sh** como **userInfo05.sh** y vamos a modificarlo para añadir nuevas características.

Este script deberá realizar las mismas tareas que el script userInfo05.sh solo que esta vez, **vamos a obtener información del usuario del fichero de cuentas de usuario del sistema para evitar equivocarnos con el directorio personal**.

Hasta ahora, el **directorio personal** del usuario hemos supuesto que está en el directorio /home y que se llama como el nombre del usuario. Pero no podemos estar seguros al 100% de esto. La mejor forma es **obtener dicha información del fichero de cuentas de usuario** con la ayuda de tuberías y comandos de filtro.

- La variable homeDir contendrá la ruta del directorio personal del usuario que se ha insertado por parámetro. El valor de homeDir lo obtendremos del fichero de cuentas de usuario del sistema.

Obtener la salida de un comando

Para obtener la salida de un comando en bash y poder trabajar con ella, por ejemplo guardándola en una variable, se utiliza la siguiente sintaxis:

```
$(comando)
```

El interprete de comandos ejecutará el comando que se escribe entre los paréntesis y sustituye la expresión $(comando) por la salida obtenida. Este valor se puede utilizar como el valor que se obtiene de una variable, como por ejemplo $USER, $PWD, $1, etc.

```
echo “los permisos del fichero datos.txt son $(ls -l datos.txt | cut -d” “ -f1)
```

Por ejemplo, si queremos guardar la salida de un comando en una variable tan solo tendremos que utilizar una asignación, como hemos hecho hasta ahora para crear variables.

```
variable=$(comando)
echo La salida de comando ha sido $variable
```

Entre los paréntesis se pueden utilizar las herramientas de bash, como tuberías, que se necesiten. Eso si, lo que se almacenará en la variable será la salida que proporcione lo que se ejecute entre los paréntesis. A continuación se presentan algunos ejemplos.

```
echo $USER:$PWD:$(date +%H:%M)
```

Muestra el nombre del usuario, el directorio actual y la hora y minutos del sistema.

```
ficheros=$(ls)
```

Almacena en la variable fichero la salida del comando ls, el listado de ficheros que haya en el directorio actual.

```
grupos=$(cat /etc/group | cut -d”:” -f1)
```

Almacena en la variable grupos un listado con los nombres de los grupos del sistema.

```
usuarios=$(cat /etc/passwd | grep /bin/bash | cut -d”:” -f1 | sort)
```

Almacena en la variable usuarios un listado ordenado con los nombres de cuentas de usuarios del sistema del sistema que utilizan /bin/bash como interprete de comandos.

```
sysUsers=$(cat /etc/passwd | grep -v /bin/bash | cut -d”:” -f1 | sort)
```

*Almacena en la variable usuarios un listado ordenado con los nombres de cuentas de usuarios del sistema del sistema que **no** utilizan /bin/bash como interprete de comandos.*

El objetivo de almacenar información de salida de comandos en variables es poder trabajar con esa información en el script. A continuación se muestra un pequeño ejemplo.

```
grupo=contabilidad
gid=$(cat /etc/group | grep $grupo | cut -d":" -f3)
miembros=$(cat /etc/group | grep $grupo | cut -d":" -f4)
echo "El grupo $grupo tiene GID $gid y sus miembros son $miembros"
```

Usando funciones

Función de error

Copia el script llamado **userInfo05.sh** como **userInfo06.sh** y vamos a modificarlo para añadir nuevas características.

Este script deberá realizar las mismas tareas que el script userInfo05.sh solo que **vamos a usar una función para mostrar mensajes de error**.

1. Crea una función llamada error que recibirá por parámetro un par de valores.
    1. El primer parámetro será el código de error.
    2. El segundo parámetro de la función será el mensaje de error a mostrar.
2. Deberás utilizar la función error para mostrar mensajes de error de tu script.
    1. Si no se pasa parámetro se deberá mostrar un mensaje de error y salir con un código de error -1.
    2. Si el usuario no existe se deberá mostrar un mensaje de error y salir con un código de error -2.
    3. Si no existe el directorio personal del usuario, se deberá mostrar un mensaje de error y salir con un código de error -3.

Funciones

Las funciones son una herramienta muy útil y potente que nos permite reutilizar un conjunto de comandos (bloque de código) en cualquier parte de nuestro script.

Una función se define con la palabra reservada function, seguida de un identificador único y un cuerpo de función, que se delimita con llaves, compuesto por un bloque de comandos como se muestra a continuación.

Palabra reservada function, identificador y cuerpo de función

```
function identificador {
    En el cuerpo de la función, formado por un bloque de comandos, se realiza la acción de la función.
    Comando1
    …
    ComandoN
}
```

La llave de cierre indica que la función termina en este punto

Para invocar o ejecutar una función, basta con escribir el nombre de la función en el script. La función se debe de haber escrito en líneas anteriores a la línea del script en la que pretendamos utilizarla.

```
function log {
     echo -n $USER:$PWD:
     date +%H:%M
}

log
ls -l $HOME
log
```

Al ejecutar una función, se ejecutan el bloque de comandos entero de la función. De esta forma, se gana en eficiencia, legibilidad y mantenimiento de scripts, puesto que solo escribimos un bloque de comandos una vez y lo podemos reutilizar en varias partes del script.

Además, a las funciones también se les puede pasar parámetro. Dentro del bloque de la función se puede acceder al valor del parámetro igual que con los parámetros de los scripts, es decir con las variables especiales $1, $2, etc. Si dentro del cuerpo de una función

Para pasar parámetros al invocar la función, después de escribir el nombre de la función, habría que escribir los parámetros que se quiere pasar, similar a una aplicación con parámetros.

```
function log {
     echo $USER:$PWD:$(date +%H:%M)
}

function mensaje {
     echo [MSG] - $1
}

function salir {
     echo Salimos de la aplicación
     echo Hasta pronto ;)
     exit
}

echo Vamos a usar funciones
# El siguiente comando tiene un fallo semántico. Trata de averiguarlo mirando la invocación y la función.
mensaje Vamos a usar funciones
# El siguiente comando funcionará como esperas.
mensaje "Vamos a usar funciones"
log
salir
```

Operaciones aritméticas

Suma, resta y multiplicación

Crea un script llamado **aritmetica.sh** que realizará las siguientes tareas:

1. Recibirá por parámetro dos operandos. Si no se pasan los dos parámetros, deberá dar un error y salir con un código de error. Utiliza la función error creada anteriormente.
2. Almacenará el parámetro 1 en la variable operando1 y el parámetro 2 en la variable operando2.
3. Deberá calcular y mostrar:
    1. La suma de los operandos, mostrando un mensaje con toda la información.
    2. La resta de los operandos, mostrando un mensaje con toda la información.
    3. La multiplicación de los operandos, mostrando un mensaje con toda la información.

Aritmética

Desde un script podemos realizar cálculos aritméticos escribiendo la operación aritmética entre corchetes precedido por el símbolo $.

```
$[EXPRESION]
```

En este caso, a diferencia de los corchetes utilizados para obtener una condición (comando test), no se dejan espacios después y antes del corchete de apertura y cierre.

Se pueden utilizar variables dentro de la expresión así como asignar el valor de un cálculo a una variable.

```
$[10*5]
x=6
y=$[$x*5]
z=$[$y+$x]
echo La suma de $x y $y es $[$x+$y]
```

Recuerda que los comandos que se escriben en un shell script de bash son interpretados por bash. Al encontrar $[EXPRESION], bash sustituirá toda esa cadena, por el resultado de la operación aritmética en EXPRESION.

Bucle for. Primeros pasos

La tabla de multiplicar

Crea un script llamado **multiplica.sh** que realizará las siguientes tareas:

1. Recibirá por parámetro un número. Si no se pasa parámetro mostrará un mensaje de error y saldrá con un código de error. Utiliza una función de error (la creada anteriormente).
2. Calculará y mostrará la tabla de multiplicar, del 1 al 10, del número pasado por parámetro.

Bucles for

El bucle for en bash es un bucle que recorre los elementos de una lista y nos permite realizar operaciones sobre dichos elementos. Su sintaxis básica es la siguiente:

```
for variable in lista; do
     cuerpo del for – bloque de comandos
done
```

El cuerpo del bucle for se ejecutará tantas veces como elementos haya en la lista. Para cada iteración o ejecución del bucle, la variable tomará el valor que le corresponda (valor i-esimo) de la lista de elementos.

Este bucle es ideal para recorrer una serie de elementos que está en una lista y realizar alguna acción concreta con ellos.

Como lista, podemos escribir los elementos de la lista literalmente, podemos utilizar variables o ejecutar comandos y obtener su salida con **$(**CMD**)**.

```
for numero in 1 2 3 4; do
     echo El numero actual es $numero
     echo El cuadrado de $numero es $[$numero*$numero]
done

for numero in $(seq 1 4); do
     echo El numero actual es $numero
     echo El cuadrado de $numero es $[$numero*$numero]
done

for fichero in $(ls); do
     echo Fichero $fichero
     echo Permisos: $(ls -l $fichero | cut -d" " -f1)
done
```