Linux para principiantes
INTRODUCCIÓN A LA PROGRAMACIÓN DE SCRIPT EN BASH

Vladimir Zúñiga C.

19 de julio 2006

Índice general

0.1. Lo que no encontrará en este documento

El objetivo de este documento es servir de referencia general para que el usuario de sus primeros pasos en la programación de script utilizando el interprete de comandos Bash, en este documento **NO** encontrará:

- Una guia de comandos de sistema comunes utilizados en la programación bash, para información referente a este particular dirijase a la guia sobre comandos utilizada en clases anteriores-.

0.2. Introducción.

Uno de los pilares en los que se apoya la flexibilidad y potencia del sistema operativo GNU/LINUX es su transparencia, la cual nos permite interactuar a un nivel mucho más profundo

con nuestro sistema, teniendo la posibildad de optimizar recursos o solucionar ptoblemas *a mano*, esto es a traves de comandos de consola, edición de archivos de configuración y, si contamos con los conocimientos, modificación de codigo fuente. Si bien no es un requisito el saber programar para utilizar GNU/LINUX, lo cierto es que saber aunque sea lo básico nos servirá para poder aprovechar todo el potencial de nuestro sistema operativo. Una buena forma de empezar en estas lides es la programación de script de BASH

0.3. ¿Qué es BASH?

Bash significa *Bourne again shell*, y es el nombre la versión libre desarrollada por el proyecto GNU de la antigua bourne shell de UNIX. Bash es el interprete de comandos (shell) por defecto de los sistemas operativos basados en el kernel Linux y su función es proporcionar una interfaz en la cual el usuario introduce comandos que la shell interpreta y envia al núcleo (kernel) para que este ejecute las operaciones. Cada vez utilizamos comandos de consola (cp, ls, cd, mkdir, cat, etc.) estamos haciendo uso de un intreprete de comandos, el cual por lo general es Bash. Programas como aterm, Eterm, xterm y similares no son más que interfaces que permiten a Bash ejecutarse desde una ventana.

0.4. ¿Qué es un Script?

Se le suele llamar script a una pieza de software que no necesita ser compilado para ser ejecutado. A diferencia de otros lenguajes de programación como C, C++ y similares que necesitan pasar por un proceso de compilación para traducir su codigo fuente a codigo de maquina, en los lenguajes interpretados como Perl, TCL y Bash, el codigo fuente siempre se mantiene como tal, pero para ejecutar los scripts escritos en estos lenguajes es necesario contar con un interprete. La mayor ventaja de los lenguajes interpretados es que pueden ser modificados en cualquier momento sin tener que pasar por procesos de compilación para probar los cambios, lo que nos permite testear nuestros programas rapidamente, facilitando la experimentación y el aprendizaje a traves de una metodologia de ensayo y error. Además la distribución de los programas hechos en lenguajes de script se acomoda perfectamente al software libre, pues no es necesario distribuir el codigo fuente de forma separada.

0.5. Comandos e instrucciones.

Otra ventaja de utilizar Bash como primer lenguaje para aprender a programar es que si conocemos los comandos de consola conoceremos los comandos de Bash. Todos los comandos que empleamos cuando trabajamos desde una terminal nos servirán para escribir nuestros scripts.

A esta altura, si hemos seguido las guías, deberiamos conocer y utilizar de forma natural los comandos más comunes. Ante cualquier duda recurra a la guia de comandos básicos.

Siempre que comencemos un script debemos poner en la primera linea del mismo cual es y donde esta el interprete que utilizaremos[1]. Bash esta por defecto en la carpeta /bin del sistema, por lo quecuando programemos en bash la primera linea de nuestros script debe ser siempre #!/bin/bash.

ejemplo:

Veamos el ejemplo más clasico, un script que muestra en pantalla el texto *Hola Mundo*

```
#!/bin/bash
echo "Hola Mundo"
```

Este script utiliza el comando *echo*, el cual muestra un mensaje en la salida estandar(a menos que nosostros redireccionemos la salida). La salida estandar es hacia donde se direccionan por defecto los mensajes y resultados que nos brindan los comandos. Como la salida estandar es la pantalla, este script nos mostrara en pantalla el mensaje que nosotros le indicamos , en este caso *Hola Mundo* utilizamos comillas cuando utilizamos mensajes, rutas o nombres de archivo que tengan espacios en blanco.

Para hacer ejecutable este script, lo grabaremos en alguna carpeta en la que tengamos privilegios (nuestra carpeta de usuario por ejemplo) y le cambiaremos los atributos otorgandole permisos de ejecución. Al nombre con el que elijamos guardar nuestro script le agragamos la extensión ***.sh*** para que sea facil reconocer de que tipo de archivo se trata.

chmod +x nombre_del_script.sh

Ahora podremos ejecutarlo desde el directorio en donde lo grabamos con el siguiente comando:

./nombre_del_script.sh

Tambien podemos ejecutarlo, nuevamente desde dentro del directorio donde lo grabamos, anteponiendo el nombre del interprete de comandos que queremos que lo ejecute: *sh, bash, ksh, csh*, etc.

ejemplo:

sh nombre_del_script.sh

Para ejecutarlo desde una carpeta distinta a aquella en donde fue guardado debemos simplemente agregar la ruta de nuestro script.

[1]Esto es común a todos los lenguajes interpretados

0.6. Redireccionamiento.

En los sistemas basados en UNIX en general y en aquellos que utilizan el kernel LINUX en particular todo lo que este contenido en el directorio raiz y sea accesible desde el es considerado un archivo, por tanto van a ser tratados como archivos tanto un documento de texto como un dispositivo de almacenamiento, una impresora e incluso la pantalla. La salida de un comando es, como se indica un poco más arriba, hacia donde se envia la informción obtenida como resultado de la ejecución de un comando, La entrada es, obviamente, desde donde obtiene información ese comando. El redireccionamiento consite en cambiar la entrada o salida estandar de un comando hacia un archivo o dispositivo distinto al señalado como predeterminado o estandar.

0.6.1. Operadores de redirección.

Para indicar el sentido de redireccionamiento (es decir desde donde, o hacia donde circula la información) se utiliza un sintaxis basada en simbolos bastante facil de comprender.

> redirecciona la salida hacia un archivo, sobreescribiendo la información anterior si la hubiese.

>> redirecciona la salida hacia un archivo, añadiendo la información a la existente si la hubiese, sin borrar esta

< redirecciona la entrada desde un archivo, sobreescribiendo la información anterior si la hubiese.

Veamos un ejemplo, utilzando el script anterior. Supongamos que ahora queremos que el mensaje *Hola Mundo* no sea mostrado en pantalla, sino que queremos qeu se almacene en un archivo, al cual, creativamente, le daremos el nombre de *foo*. Para poder realizar esto simplemente redireccionamos la salida del comando echo de la siguiente forma:

```
#!/bin/bash
echo "Hola Mundo" > foo
```

Ejecutamos el script, listamos el directorio y veremos un archivo llamado foo, cuyo unico contenido es nuestro mensaje. Si volvemos a ejecutar el script, volveremos a tener el mismo resultado, es decir el contenido del archivo no será que nuestro mensaje escrito una vez. Sin embargo si cambiamos el operador de dirección por >> nuestro mensaje aparecerá repetido tantas veces como ejecuciones del script realicemos.

```
#!/bin/bash
echo "Hola Mundo" >> foo
```

0.7. Variables

Una variable es, como su nombre lo dice, un dato cuyo valor puede variar. Se utilizan para menciomar de forma relativa un dato cuando este, por ser susceptible de cambiar, no puede ser denominado de forma absoluta. Es, en resumidas cuentas, un nombre simbolico que representa un valor. En bash, una vez definida la variable debemos anteponer el signo $ seguido de su nombre para invocarlas. Para definirlas basta escribir su nombre y otorgarles un valor (numerico o alfanumerico)

Otra forma de definirlas es a traves del comando read, cuyo uso veremos más adelante.

ejemplo

#!/bin/bash
MENSAJE="Hola Mundo"
echo $MENSAJE

En este ejemplo definimos la variable MENSAJE, dandole como valor o contenido el mensaje "Hola Mundo", luego al comando echo le dijimos que mostrará el contenido de la variable invocandola con el signo $. Como no le dimos ninguna redirección, envio el contenido de la variable a la salida estandar, es decir, la pantalla.

Las variables numericas son suceptibles de ser operadas matematicamente, es decir, podemos realizar calculos utilizando las variables.

Ejemplo:

```
#!/bin/bash
    echo "ingrese un numero:"
    read dato1
    echo "ingrese otro numero:"
    read dato2
    multipli=$[$dato1*$dato2]
    echo "$dato1 multiplicado por $dato2 da como resulatdo $multipli"
```

En este ejemplo hacemos uso del comando read, el cual nos permite ingresar un valor a una variable durante la ejecución del script, lo cual le da mucho más dinamismo y utilidad a nuestras creaciones. Es importante recordar la sintaxis utilizada para la definición de la variable *multipli*, pues notaran que presenta algunas variaciones respevto a la declaración de las demás variables que hemos visto, pues esta variable encierra una operación matematica.

Nota: No se debe usar una palabra reservada del sistema como un comando como nombre de una variable. Una forma de evitarlo es emplear mayúsculas para los nombres de las variables.

Ejemplo:

```
#!/bin/bash
    CMD=uname
    $CMD
```

0.8. Insertar Comentarios.

Un comentario es una linea de texto incluida en un script bash que el interprete de comandos pasa por alto. Su función es servir de guia al programador al momento de escribir sus scripts y son una excelente referencia para hacer modificaciones a posteriori. Tambien sirven para anular una linea de codigo si es que su uso es optativo, o para ir habilitando y dehabilitando lineas en procesos de depuración de errores.

Para insertar un comentario basta insertar el signo # al comienzo de la linea.

Ejemplo:

```
#!/bin/bash
    #esta linea es un comentario, no afecta en nada a la ejecución de este script
    MENSAJE="Hola Mundo"
    #otro comentario
    echo $MENSAJE
```

0.9. Estructuras condicionales

0.9.1. if / then

La estructura condicional es una herramienta muy util al momento de escribir nuestros script, pues nos permiten determinar que si se cumple determinada condición (determinado valor de variable, por ejemplo) entonces se ejecute determinado comando. La sintaxis de una estructura condicional es la siguiente:

if <condición>;
then <comandos>

fi

Esta estructura puede contener más de un bloque, lo que le otorga a nuestros scripts la capacidad de analizar más de una condición:

if <condición 1>;
then
<bloque 1>
elif <condición 2>;
then
<bloque 2>
elif <condición 3>;
then
<bloque 3>
fi

Tambien puede definirse una acción a realizar si se cumple la condición y otra que se realizará si no se cumple la condición, para eso se agrega la instrucción e*lse* de la siguiente manera:

if <condición>;
then <comandos>
else <comandos>
fi

Ejemplo:

```
#!/bin/bash
    echo elija su opcion
    echo 1 - Verdadero echo
    2 - Falso
    read OP
    if [ "$OP" = "1" ];
    then
    echo Esto se vera si la condición es VERDAD
    else
    echo Esto se vera si la condición es FALSA
    fi
    echo FIN
```

0.9.2. case

El comportamiento de la estructura condicional case es bastante similar a lo visto en if/then, es decir, se basa en condiciones, variables y comandos a realizar frente a determinado valor de dichas variables, la diferencia con la estructura anterior es que no podemos definir un comando a ejecutarse si la condición no se cumple (como *else*) sin embargo crear menus de opciones es bastante más facil con esta estructura:

Ejemplo:

```
#!/bin/bash
echo elija su opcion
    echo 1 - Fecha
    echo 2 - Hora
    read op
    case $op
    in
    "1") date +" %A %e de %B de %G"
    ;;
    "2") date + %T
    ;;
    *) echo Opción incorrecta
    ;;
    esac
```

0.9.3. Condiciones.

Para poder utilizar estructuras condicionales es impresincdible conocer los operadores de condición, a contiinuación va un listado de los distintos operadores usados dependiendo del tipo de datos a manejar.

0.9.3.1. Cadenas de texto (strings)

["$a" = "$b"] Igualdad
["$a" == "$b"] Igualdad
["$a" != "$b"] Desigualdad
["$a" \< "$b"] Orden alfabético ascendente.
["$b" \> "$b"] Orden alfabético descendente.
[-z "$a"] es verdadero si la variable esta vacía
[-n "$a"] es verdadero si la variable no esta vacia

0.9.3.2. Condiciones sobre números enteros (integer)

["$a" -eq "$b"] Igualdad
["$a" -ne "$b"] Desigualdad
["$a" -gt "$b"] Mayor que
["$a" -lt "$b"] Menor que
["$a" -ge "$b"] Mayor o igual que
["$a" -le "$b"] Menor o igual que

0.9.3.3. Condiciones en Archivos

[-e nombre fichero] Existencia
[-f nombre fichero] es fichero ordinario
[-s nombre fichero] es fichero vacío (0 bytes)
[-d nombre fichero] es directorio
[-b nombre fichero] es dispositivo de bloques
[-c nombre fichero] es dispositivo de caracteres
[-L nombre fichero] es enlace simbólico
[-r nombre fichero] tiene permiso de lectura
[-x nombre fichero] tiene permiso de ejecución
[-w nombre fichero] tiene permiso de escritura
[-O nombre fichero] somos propietarios del fichero (UID)
[-G nombre fichero] somos propietarios del fichero (GID)
[-N nombre fichero] el fichero se modifico desde la ultima vez que fue leído
[-h nombre fichero] es enlace duro

0.9.3.4. Negación

Se usa el carácter !
[! -e fichero]
[! -f fichero]

0.10. Licencia.