

Bourne Again Shell

Parte I: Scripts Básicos

Dr. Mario M. Berón

Universidad Nacional de San Luis

Usando Múltiples Comandos

Uso de ";"

Uno de los factores claves del shell script es la posibilidad de ingresar múltiples comandos y procesar los resultados de cada comando posiblemente pasando los resultados de un comando a otro. El shell permite encadenar comandos de manera simple usando ";".

Ejecutando dos Comandos Secuencialmente

```
# date; who
```

Creando Archivos de Script

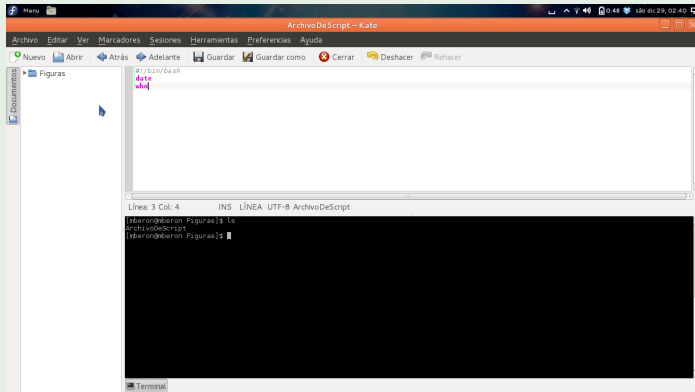
Pasos para Crear un Archivo de Script

Los comandos que un usuario quiere ejecutar pueden ser colocados en un archivo. Para realizar esta tarea es necesario:

- 1 Crear un archivo de texto con algún editor de textos.
- 2 Colocar en la primera línea del archivo creado el siguiente texto:
`#!/bin/bash`
dicha línea indica el intérprete de comandos que se utilizará para ejecutar el script.
- 3 Colocar los comandos uno debajo de otro.

Creando Archivos de Script

Ejemplo de Archivo de Script



```
#!/bin/bash
date
who
```

Línea: 3 Col: 4 INS LÍNEA UTF-8 ArchivoDeScript

```
reberon@reberon-Figuras]$ ls
ArchivoDeScript
reberon@reberon-Figuras]$
```

Importante

Es posible agregar comentarios a los shell scripts. Esta tarea se realiza utilizando el caracter #.

Creando Archivos de Script

Problemas

Una vez creado el archivo de script si el mismo se intenta ejecutar se obtendrá un mensaje de error. Esto se debe a que:

- 1 La variable PATH del sistema operativo no contiene el directorio donde se encuentra el script (Paso no Encontrado).
- 2 No se tiene permiso de ejecución (Permiso Denegado).

Creando Archivos de Script

Paso no Encontrado

La variable PATH contiene un conjunto de carpetas que el sistema operativo utilizará para buscar los comandos. Si la carpeta donde esta el comando no está especificada en la variable PATH entonces se obtendrá un error que indica que el comando no se puede encontrar.

Este problema se puede solucionar de la siguiente manera:

- Incorporar la carpeta donde el script se encuentra. De esta forma cada vez que se invoque dicho shell el sistema operativo lo podrá encontrar en la carpeta especificada.
- Usar un paso absoluto o relativo que indique donde se encuentra el script.

Creando Archivos de Script

Ejemplo

Asumiendo que:

- 1 El script está almacenado en la carpeta corriente.
- 2 El script tiene como nombre test1.
- 3 Se usa la segunda aproximación.

La invocación al script se realiza de la siguiente manera:

```
#./test1
```

Creando Archivos de Script

Permiso Denegado

También puede suceder que los pasos para ejecutar el script sean correctos pero el usuario no tenga permiso de ejecución. En esta situación cuando se ejecute el comando:

```
#./test1
```

se obtendrá como resultado:

```
Permission Denied
```


Creando Archivos de Script

Permiso Denegado

En estos casos la solución consiste en ejecutar el comando *chmod* para incorporar los permisos correspondientes.

Mensajes

Echo

Muchas veces es necesario que el script muestre mensajes que le informen al usuario la tarea que está realizando el script. Esta tarea se lleva a cabo utilizando el comando *echo*. Este comando recibe como entrada una cadena de caracteres y muestra por pantalla la cadena recibida como parámetro.

Ejemplo

```
# echo "Hola Mundo"  
Hola Mundo
```

Mensajes

Echo - Características

- 1 El comando echo puede usar comillas dobles o simples para delimitar la cadena de entrada.
- 2 Las comillas no son obligatorias para demarcar la cadena de entrada.

Ejemplo

Los siguientes comandos producen la misma salida:

```
# echo Hola Mundo
```

```
# echo 'Hola Mundo'
```

```
# echo "Hola Mundo"
```

Mensajes

Echo - Características

Es importante notar que si la cadena de entrada tiene comillas en su interior se deberán usar comillas de demarcación diferentes a las usadas dentro de la cadena. En otras palabras si la cadena contiene comillas simples entonces la cadena debe ser encerrada con comillas dobles y viceversa. En otro caso las comillas internas no se mostrarán por pantalla.

Ejemplo

```
# echo Hola 'Juan'
Hola Juan
# echo "Hola 'Juan'"
Hola 'Juan'
```

Variables

Variables de Ambiente

Las variables de ambiente proveen información específica del sistema, como por ejemplo:

- 1 El nombre del sistema.
- 2 El nombre del usuario registrado en el sistema.
- 3 La identificación del usuario en el sistema.
- 4 Etc.

Variables

Variables de Ambiente

Un listado completo de las variables de ambiente puede obtenerse por medio del uso del comando: *set*.

```
# set
BACKSPACE=Delete
BASH=/bin/bash
EUID=1000
HISTFILE=/home/rich/.bash history
HISTFILESIZE=1000
HISTSIZ=1000
HOME=/home/rich
HOSTNAME=testing
HOSTTYPE=i586
LANG=en
LANGUAGE=en US:en
LINES=24
LOGNAME=rich
...
```

Variables

Variables de Ambiente

Las variables de ambiente se pueden acceder anteponiendo al nombre de la variable el singo \$.

Ejemplo

- ❶ \$USER: Nombre del usuario.
- ❷ \$HOME: El directorio del usuario.
- ❸ \$UID: La identificación del usuario en el sistema.
- ❹ Etc.

Variables

Variables de Ambiente

Las variables también pueden ser referenciadas usando el formato `$variable`. Este tipo de notación es frecuentemente usada para identificar, sin inconvenientes, el nombre de la variable.

Variables

Variables de Usuario

El programador de un script también puede definir sus propias variables. Las variables le permiten almacenar datos que pueden ser usados en diferentes partes de un script. De esta manera un script se asemeja más a un programa en un lenguaje de programación común.

Variables

Características

- 1 Pueden ser cualquier cadena de caracteres de hasta 20 caracteres. Dentro de la cadena pueden haber: letras, dígitos, guiones bajos, etc.
- 2 Existen diferencias entre mayúsculas y minúsculas. Ej: *Var* es diferente de *var*.
- 3 Se asignan valores usando el signo `=`. Ej: `var=1`. Es importante mencionar que no pueden aparecer espacios en blanco entre la variable el signo igual y el valor que se desea asignar.
- 4 El script determina automáticamente el tipo de dato de la variable.
- 5 El tiempo de vida de una variable esta determinado por: el tiempo de creación y la finalización del script.
- 6 Las variables de usuario pueden ser referenciadas de la misma forma que son referenciadas las variables del sistema.

Backtick

Backtick

El caracter Backtick ` (comillas simples hacia atrás) permite asignar la salida de un comando a una variable.

Ejemplo

```
#!/bin/bash
# Uso del caracter backtick
test=`date`
echo "La fecha y el tiempo son: " ` $test
```

Redireccionamiento de la Entrada y la Salida

Redireccionamiento de la Salida

Comando > Archivo

Comando >> Archivo

Ejemplo

```
$ date > test
$ ls test
test
$ cat test
mar ene 29 15:07:57 ART 2013
$
```

Redireccionamiento de la Entrada y la Salida

Redireccionamiento de la Entrada

Comando < Archivo

Comando << Marcador Datos Marcador

Ejemplo

```
$ date > test
$ ls test
test
$ cat test
mar ene 29 15:07:57 ART 2013
$ wc < test
1  6 29 test
```

Redireccionamiento de la Entrada y la Salida

Redireccionamiento de la Entrada

Comando `<` Archivo

Comando `<<` Marcador Datos Marcador

Ejemplo

```
$ wc << Marcador  
> Hola que tal  
> Marcador  
1  3 13
```

Redireccionamiento de la Entrada y la Salida

Pipes

Son usados cuando la salida de un comando se necesita como entrada en otro comando.

Comando1 | Comando2

Comando1 y Comando2 son ejecutados al mismo tiempo. Conforme Comando1 produce una salida esta es enviada a Comando2 para que dicho comando la procese.

Importante

Es importante notar que no hay límites en la cantidad de pipes que se deseen usar. Se pueden escribir tantos pipes como se deseen siempre y cuando se respete la longitud de la línea de comandos que en general es de 255 caracteres.

Operaciones Matemáticas

El Comando Expr

Este comando es usado para procesar ecuaciones matemáticas desde la línea de comandos.

Ejemplo de Operadores Soportados por Expr

- $\text{ARG1} \mid \text{ARG2}$: Retorna ARG1 si ningún argumento es NULL o cero. En otro caso retorna ARG2.
- $\text{ARG1} \& \text{ARG2}$: Retorna ARG1 si ningún argumento es null o cero. En otro caso retorna 0.
- $\text{ARG1} < \text{ARG2}$: Retorna 1 si ARG1 es menor que ARG2. En otro caso retorna 0.
- $\text{ARG1} \leq \text{ARG2}$: Retorna 1 si ARG1 es menor o igual que ARG2. En otro caso retorna 0.
- $\text{ARG1} = \text{ARG2}$: Retorna 1 si ARG1 es igual a ARG2. En otro caso retorna 0.

Operaciones Matemáticas

El Comando Expr

El comando *expr* es un poco incómodo de usar porque algunos operadores tienen un significado diferente en el shell. Por otra parte el valor del comando debe ser capturado a través del uso del backtick.

Ejemplo de Uso de Backtic

```
#!/bin/bash
# An example of using the expr command
var1=10
var2=20
var3=`expr $var2 / $var1`
echo The result is $var3
```

Operaciones Matemáticas

Uso de Corchetes

Afortunadamente los problemas presentados por el comando *expr* se pueden evitar por medio del uso de *[operaciones]*.

Ejemplo de Uso de Backtick

```
#!/bin/bash
var1=100
var2=50
var3=45
var4=$[$var1 * ($var2 - $var3)]
echo The final result is $var4
```

Operaciones Matemáticas

Problema

Los operadores matemáticos del intérprete de comandos solo soportan aritmética entera.

```
#!/bin/bash
var1=100
var2=45
var3=$((var1 / var2))
echo The final result is $var3
```

Invocación del Script

```
$ chmod u+x test8
$ ./test8
The final result is 2
$
```

Operaciones Matemáticas

Operaciones de Punto Flotante en Shell Scripts

Una forma de solucionar la limitación del intérprete de comandos respecto de las operaciones matemáticas es usando la calculadora de bash: `bc`.

Características de `bc`

Permite ingresar expresiones en una línea de comandos, luego las interpreta, calcula y retorna el resultado. `bc` reconoce:

- Números enteros y de punto flotante.
- Variables simples y arreglos.
- Comentarios (`/* aquí va un comentario */`)
- Expresiones
- Sentencias de programación
- Funciones

Operaciones Matemáticas

Uso de bc

- Para invocar a bc se escribe lo siguiente en la línea de comandos:
`$ bc`
- Para salir de bc se debe escribir el comando *quit*
- Una vez invocado bc se puede comenzar a escribir las expresiones que el usuario quiere calcular.
- La aritmética de punto flotante esta controlada por la variable *scale*. Esta variable controla la cantidad de decimales que se utilizarán en las respuestas que requieren operaciones de punto flotante. El valor por defecto de esta variable es 0.
- También bc permite el uso de variables. Una vez que se define una variable la misma se puede usar durante toda la sesión de bc.

Operaciones Matemáticas

Uso de bc en Scripts

El formato general para usar bc en los scripts es el siguiente:

```
variable='echo "opciones; expresión" | bc'
```

- opciones: Se utiliza para inicializar variables. Cuando se desea utilizar más de una variable se las separa con ";".
- expresion: Define la expresión matemática que se quiere evaluar.

Operaciones Matemáticas

Ejemplo de Uso de bc en Scripts

```
#!/bin/bash
var1='echo " scale=4; 3.44 / 5" | bc'
echo The answer is $var1

$ chmod u+x test9
$ ./test9
The answer is .6880
$
```

Finalización de un Script

Finalización de un Script

- Todo comando que ejecuta en el intérprete de comandos usa un estado de salida que le indica al intérprete que el procesamiento está realizado.
- El estado de salida es un número entre 0 y 255 que el comando le pasa al intérprete cuando finaliza su ejecución.
- Este valor se puede capturar para utilizarlo dentro de los scripts.

Finalización de un Script

Verificación del Estado de Salida

- El estado de salida se reporta en la variable \$?.
- Esta variable se puede inspeccionar inmediatamente después del haber ejecutado el comando que se quiere analizar.
- Por convención los comandos que completan su ejecución exitosamente tienen como estado de salida 0. En caso contrario un entero positivo se coloca en la variable que mantiene el estado de salida.

Finalización de un Script

Ejemplo

```
$ date  
Sat Sep 29 10:01:30 EDT 2007  
$ echo $?  
0  
$
```

Finalización de un Script

Códigos de Estado

- 0 Successful completion of the command
- 1 General unknown error
- 2 Missuse of shell command
- 126 The command can't execute
- 127 Command not found
- 128 Invalid exit argument
- 128+x Fatal error with Linux signal x
- 130 Command terminated with Ctl-C
- 255 Exit status out of range

Finalización de un Script

El Comando Exit

Por defecto el script retorna como estado de salida el estado retornado por el último comando.

La característica mencionada en el párrafo precedente se puede cambiar, es decir el usuario puede especificar el estado de salida.

La tarea mencionada previamente se puede llevar a cabo usando el comando *exit*.

Finalización de un Script

Ejemplo

```
#!/bin/bash
# testing the exit status
var1=10
var2=30
var3=$(( $var1 + var2 ))
echo The answer is $var3
exit 5
```

```
$ chmod u+x test13
$ ./test13
The answer is 40
$ echo $?
5
$
```

Finalización de un Script

Comentarios

- Si el número retornado por un comando exit es superior a 255 se aplica aritmética modular para llevarlo a un número en el rango permitido.
- El comando exit puede recibir una variable como parámetro.