Shell scripting en Linux

Roberto Gómez Cárdenas
rogomez@itesm.mx
http://homepage.cem.itesm.mx/rogomez

Introducción

- Shell
  - Interfaz con el usuario
  - Interprete de comandos
  - Aspectos de programación
- Shell script
  - Ejecución automática de comandos
  - Procesamiento en batch de comandos
  - Tareas repetitivas

Shells en Linux

- Existen varios disponibles
- Ejemplos
  - Bourne shell: sh
  - Korn shell: ksh
  - C shell: csh
  - Bash: bsh
- El shell bash es el más popular.

El Bourne Again Shell

- Abreviado shell
- Es el shell por default en la mayoría de las distribuciones Linux
- También usado en todas las plataformas Unix
- Contiene características de
  - ksh, csj, sh, etc

Los scripts

- Archivos que contienen comandos a ser ejecutados por el shell.
- Puede ser cualquier comando que pueda teclearse a partir del prompt:
  - comando que invoque una utilidad Unix, (vi, netscape, etc)
  - un programa compilado
  - otro script
- Aparte de estos comandos existe un grupo de comandos, (los *comandos de control de flujo)*, que fueron diseñados para ser usados en scripts.

Características programación bash

- Soporta varías características de programación
  - Variables, arreglos, ciclos, operadores de decisión, funciones, parámetro posicionales
- Pipes, redirección entrada/salida
- Características varias
  - Expansiones, control de trabajos (jobs)
- Comandos construidos a su interior
  - read, echo, source, alias

Un script shell muy simple

```
#!/bin/bash
# Archivo: p1
# Uses the cat command to display a file

echo Ejemplo de script
echo –n Fecha:
date
echo Usuarios conectados
who
echo  Contenido directorio trabajo
ls -l
```

Ejecutando un script shell

- Primer método

```
$ chmod u+x catfile.sh
$ ./catfile.sh file1
```

- Segundo método

```
$ bash catfile.sh file1
```

Variables

- Posible usar variables como en cualquier lenguaje de programación.
- Valores siempre almacenados como strings
  - Existen operadores matemáticas en el lenguaje shell que convertirá variables a número para cálculos
- No es necesario declarar una variable
  - Con solo asignar un valor a su referencia, esta será creada
- Ejemplo, nombre archivo: sc1

```
#!/bin/bash
STR="Hello World!"
echo $STR
```

Carácterísticas variables

- Las variables no cuentan con un tipo.
- Variables pueden tomar un número o un carácter.
  - cont = 0
  - cont = domingo
- El carácter \ es el carácter de escape y preserva el valor literario del carácter que le sigue

```
$ ls \*
ls *: No such file or directory
$
```

Apostrofes y comillas

- Cuando se asignan cadenas de caracteres que contiene espacios o caracteres especiales, la cadena debe estar encerrada entre apostrofes o comillas
- El uso de comillas (partial quoting) dentro de una cadena de caracteres permitira que cualquier variable dentro de las comillas sea interpretada

```
$ var="test string"
$ newvar="Value of var is $var"
$ echo $newvar
Value of var is test string
$
```

Apostrofes y comillas

- El uso de apostrofes (full quoing) dentro de una cadena de caracteres no permitirá una interpretación de variables

```
$ var=’prueba 1’
$ newvar=‘El valor de var es: $var’
$ echo $newvar
El valor de var es: $var
$
```

Ejemplos

```
$ pippo=  pluto
$ pippo  =pluto
```

error

```
$ ls [Pp]*
$ ls "[Pp]*"
$ ls '[Pp]*'
```

no resuelto

```
$ var="'(]\\{}\$\""
$ echo $var
$ echo "$var"
$ echo '$var'

$ echo \z       # z
$ echo \\z        # \z
$ echo '\\z'    # \\z
```

Más ejemplos

```
$ pippo=  cat

$ echo "comando = \" $pippo \" "
comando =" cat "

$ echo 'comando = \" $pippo \" '
comando =\" $pippo \"

$ echo 'comando = " $pippo " '
comando =" $pippo "
$
```

Variables de ambiente

- Existen dos tipos de variables
  - Variables locales
  - Variables de ambiente
- Variables ambiente
  - Son inicializadas por el sistema y se pueden listar con el comando env
  - Almacenan valores especiales

Ejemplo salida comando env

```
$ env
HOSTNAME=localhost
PVM_RSH=/usr/bin/rsh
SHELL=/bin/bash
TERM=xterm
HISTSIZE=1000
USER=root
LS_COLORS=no=00:fi=00:di=00;34: ln=00;36:pi=40;33:so=00;35
PVM_ROOT=/usr/share/pvm3
USERNAME=root
MAIL=/var/spool/mail/root
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:
INPUTRC=/etc/inputrc
PWD=/root
LANG=en_US.UTF-8
        :
        :
        :
```

Ejemplo salida comando env

```
        :
        :
        :
SSH_ASKPASS=/usr/libexec/openssh/gnome-ssh-askpass
SHLVL=1
HOME=/root
BASH_ENV=/root/.bashrc
LOGNAME=root
LESSOPEN=|/usr/bin/lesspipe.sh %s
DISPLAY=:0.0
G_BROKEN_FILENAMES=1
XAUTHORITY=/root/.xauthauuuYC
_=/bin/env
$
```

Algunas variables de ambiente

- LOGNAME: contiene el nombre del usuario
- HOSTNAME: contiene el nombre de la computadora
- MACHTYPE: hardware del sistema
- PS1: secuencia caracteres mostrados antes del prompt
- UID: uid del usuario
- SHLVL: el nivel del shell

Variables predefinidas para lectura de parámetros en la línea de comandos

Variable	Significado
$?	Valor de salida del último comando, 0 si todo salió bien
$0	nombre del script
$1 a $9	argumentos que se pasaron al script
$#	numero argumentos pasados al script
$*	lista de argumentos a partir de $1
$$	numero pid del proceso actual
$!	número pid del proceso hijo

recordar que para desplegar su valor es necesario usar el comando *echo*

Ejemplo de uso variables parámetros

```
$ more p2
echo Nombre del script: $0
echo Numero argumentos: $#
echo Lista de argumentos: $*
echo pid del proceso actual: $$
echo pid del proceso hijo: $!
$ chmod u+x p2
$ ./p2 uno dos tres
Nombre del script: ./s1
Numero de argumentos: 3
Lista de argumentos: uno dos tres
pid del proceso actual: 3818
pid del proceso hijo:
$
```

El prompt

- Símbolo que indica que el shell esta listo para recibir instrucciones.
- Existen prompts por defeault, dependiendo del shell y tipo de unix utilizado
  - $ bourne shell o korn shell
  - % c shell o tc shell
  - # representa que el usuario es root
- Es posible que un usuarios defina su propio prompt a través de la variable de ambiente correspondiente
  - En el caso de Linux es PS1, por ejemplo el valor

    PS1 = '\[\u@\h \W]$ '
  - produce el siguiente prompt

    [toto@localhost bin]$

Opciones para configurar el prompt

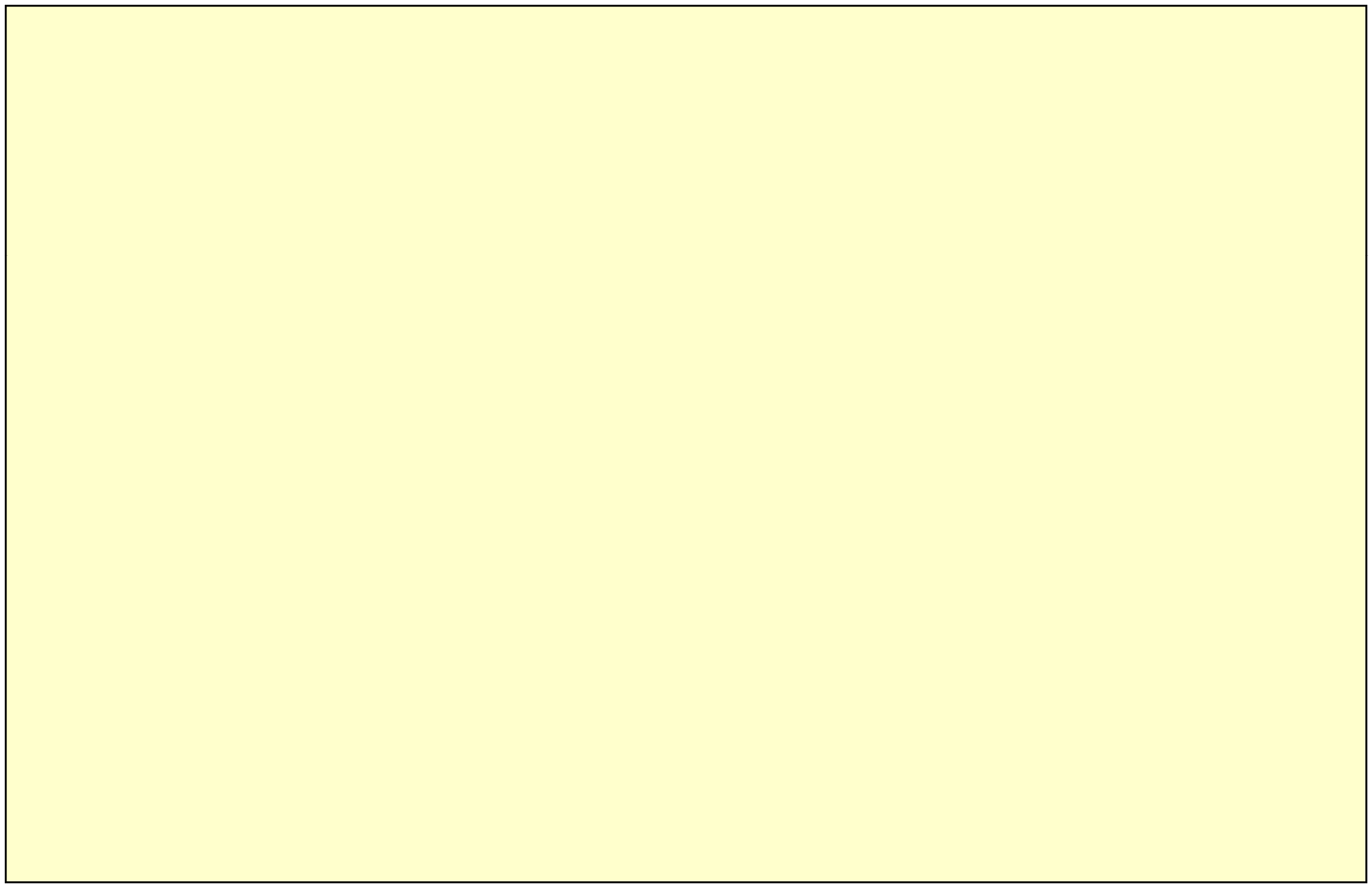
Carácteres	Significado
\u	Nombre usuario
\W	Directorio trabajo
\w	Ruta completa de trabajo
\t	La hora actual
\d	La fecha actual
\s	El nombre del shell
\h	El nombre de la máquina actual
\#	El número de comando
\!	La posición en el history
\$	Carácter de prompt según el shell
\nnn	Carácter nnn (en octal)

Ejercicio

- Modifique su variable de ambiente prompt de tal forma que despliegue un prompt con las siguientes características:
  - Muestre el nombre del usuario
  - La hora entre parantesis, precedida y seguida de un blanco
  - El carácter dos puntos “:”
  - El numero de instrucción
  - El carácter mayor que “>”
- Un ejemplo de lo anterior se presenta a continuación:

rogomez (13:34) :23>

Solución ejercicio



Comando exit

- Puede ser usado para terminar un script
- También puede regresar un valor, el cual esta disponible al padre del proceso
- Sintaxis

```
exit nnn
```

  - donde nnn es el status de salida
  - puede tomar un valor entre 0 y 255
  - es el programador el que decide que número usar

Comando exit

- Cuando un script termina con exit que no cuenta con parámetro el status regresado es el status con el que terminó el último comando ejecutado en el script.

```
#!/bin/bash

COMMAND_1

. . .

# exit with status of last
command.
COMMAND_LAST

exit
```

```
#!/bin/bash

COMMAND_1

. . .

# exit with status of last
command.
COMMAND_LAST

exit $?
```

Comando read

- Permite solicitar datos de entrada y almacenarlos en una variable.
- Ejemplo, nombre archivo: leer

```
#!/bin/bash
#Nombre archivo: renombra
echo -n "De el nombre del archivo: "
read original
echo –n "De el nuevo nombre:"
read nuevo
mv $original $nuevo
echo "El archivo $original ahora se llama $nuevo"
```

Opciones comando read

- read –s
  - No hace un eco de la entrada
- read –nN
  - Solo acepta n caracteres como entrada
- read –p “mensaje”
  - Despliega mensaje
- read –tT
  - Acepta entrada por T segundos

Ejercicio read

- Crear programa r1

```
#!/bin/bash
echo De una frase
read frase
echo $frase
```

- Ejecutar programa
- Substituir la línea read frase por
  - read –s *(guardar, ejecutar y conclusiones)*
  - read –n5 *(guardar, ejecutar y conclusiones)*
  - read –p “PRUEBA: ” *(guardar, ejecutar y conclusiones)*
  - read –t2 *(guardar, ejecutar, no teclear nada en 3 segs y conclusiones)*

Operadores aritméticos

Operador	Significado
+	suma
-	resta
*	multiplicación
/	división
**	exponenciación
%	modulo

Enunciado let

- Usado para llevar a cabo funciones matemáticas

```
$ let X=10+2*7
$ echo $X
24
$ let Y=X+2*4
$ echo $Y
32
```

- Expresión puede ser evaluada con:

$[expression] or $((expression))

```
$ echo $((123+20))
      143
$ VALORE=$[123+20]
$ echo $[123*$VALORE]
      1430
$ echo $[2**3]
$ echo $[8%3]
```

Enunciados condicionales

- Decidir si una acción se lleva a cabo o no.
- Esta decisión se toma evaluando una expresión.
- Sintaxis básica:

```
if [ expression ];
  then
        statements
  elif [ expression ];
        then
          statements
  else
          statements
fi
```

```
if [ expression ];
  then
        statements
  else
          statements
fi
```

```
if [ expression ];
  then
        statements
fi
```

Expresiones

- Una expresión puede ser
  - Comparación de strings
  - Comparación numérica
  - Operadores archivos
  - Operadores lógicos
- Las expresiones deben ir entre corchetes, separadas por un espacio

```
[ expresion ]
```

- Los operadores también deben ir separados por un espacio

```
[ expresion1 operador expresion2 ]
```

Comparativos strings

Operador	Significado
=	Comparar si dos strings son iguales
!=	Comparar si dos strings no son iguales
-n	Evaluar si longitud del string es mayor que cero
-z	Evaluar si longitud del string es igual a cero

Ejemplo	Significado
[s1 = s2]	Verdad si s1 es igual a s2, sino falso
[s1 != s2]	Verdad si es diferente a s2 sino falso
[-n s1]	Verdad si s1 no esta vacío, sino falso
[-z s1]	Verdad si la longitud de s1 es igual a cero, sino falso

Ejemplo expresiones strings

```
#!/bin/bash
# Archivo: checa

echo -n “Introduzca su nombre: "
read name
if [ “$name” = “$USER” ];
then
        echo “Hola, $name. Como se encuentra hoy?”
else
        echo “Usted no es $USER, quien es usted?”
fi
```

Comparativos numéricos

Operador	Significado
-eq	Comparar si dos números son iguales
-ge	Comparar si un número es mayor que o igual a otro
-le	Comparar si un número es menor que o igual a otro
-ne	Comparar si dos números no son iguales
-gt	Comparar si un número es mayor que otro
-lt	Comparar si un número es menor que otro

Ejemplos comparativos

Tomando en cuenta que n1 y n2 son dos números enteros

Operador	Significado
[n1 -eq n2]	Verdad si n1 es igual a n2, falso en caso contrario
[n1 -ge n2]	Verdad si n1 es mayor o igual a n2, falso en caso contrario
[n1 -le n2]	Verdad si n1 es menor o igual a n2, falso en caso contrario
[n1 -ne n2]	Verdad si n1 no es igual a n2, falso en caso contrario
[n1 -gt n2]	Verdad si n1 es mayor que n2, falso en caso contrario
[n1 -lt n2]	Verdad si n1 es menor que n2, falso en caso contrario

Ejemplo

```
#!/bin/bash
#  Archivo: numeros
echo -n “Introduzca un numero 1 < x < 10: "
read num
if [ “$num” -lt 10 ];    then
      if [ “$num” -gt 1 ]; then
          echo “$num*$num=$(($num*$num))”
      else
          echo “Mala insercion, es menor a 1 !”
      fi
else
    echo “Mala insercion, es mayor a 10 !”
fi
```

Comparativos archivos

Los más usados son:

Operador	Significado
-d	Verifica si el path es un directorio
-f	Verifica si el path es un archivo
-e	Verifica si el nombre del archivo existe
-s	Verifica si el archivo tiene una longitud mayor a cero
-r	Verifica si el archivo tiene permiso de lectura
-w	Verifica si el archivo tiene permiso de escritura
-x	Verifica si el archivo tiene permiso de ejecución

Ejemplo

Tomando en cuenta que fd es el nombre de un archivo

Operador	Significado
[-d fd]	Verdad si fd es un directorio, falso en caso contrario
[-f fd]	Verdad si fd es un archivo, falso en caso contrario
[-e fd]	Verdad si fd existe, falso en caso contrario
[-s fd]	Verdad si la longitud del archivo fd es mayor a 0, falso en caso contrario
[-r fd]	Verdad si fd tiene permisos de lectura, falso en caso contrario
[-w fd]	Verdad si fd tiene permisos de escritura, falso en caso contrario
[-x fd]	Verdad si fd tiene permisos de ejecución, falso en caso contrario

Ejemplo

```
#!/bin/bash
# Nombre archivo: copia
# Script que verifica que /etc/fstab existe, si es asi
# lo copia, sino indica que no existe

if [ -f /etc/fstab ];
        then
        cp /etc/fstab .
                echo “Hecho.”
        else
        echo “El archivo no existe.”
                exit 1
        fi
```

Ejercicio

- Escriba un script, de nombre *respaldo* , que pregunte por el nombre de un archivo
  - El script debe verificar si el archivo existe
  - En caso de que exista debe copiar dicho archivo a un directorio de nombre Backup, añadiendo la extensión .back al nombre del archivo.
    - Si el archivo se llama toto, lo copiara como toto.back
  - Si el directorio Backup no existe lo debe crear.
  - Si el archivo .back ya existe en el directorio Backup, debe preguntar si desea reemplazarlo.
  - Si el archivo no existe, entonces debe salirse del script con el mensaje: “El archivo <arch> no existe !!! “

Solución ejercicio

Operadores lógicos

Operador	Significado
!	Negación (NOT) de una expresión lógica
-a &&	AND lógico entre dos expresiones lógicas
-o ||	OR lógico entre dos expresiones lógicas

```
#!/bin/bash
# Nombre archivo: if3.sh
echo -n “Enter a number 1 < x < 10:”
read num
if [ “$num” -gt 1 –a “$num” -lt 10 ];
then
        echo “$num*$num=$(($num*$num))”
else
        echo “Wrong insertion !”
fi
```

Parámetros shell

- Parámetros posicionales
  - Asignados de los argumentos del shell cuando el script es invocado
  - El parámetro posicional “N” puede ser referenciado como “${N}”, o como “$N” donde N consiste de un simple dígito
  - $0: el nombre del script corriendo
  - $1: el primer parámetro
  - $2: el segundo parámetro
  - etc.

Parámetros shell

- Parámetros especiales

Variable	Significado
$?	Valor de salida del último comando, 0 si todo salió bien
$0	nombre del script
$@	Arreglo de palabras conteniendo todos los parámetros pasados al script
$#	numero argumentos pasados al script
$*	lista de argumentos a partir de $1
$$	numero pid del proceso actual
$!	número pid del proceso hijo

Enunciado case

- Usado para ejecutar enunciados basado en valores específicos.
- Sintaxis

```
case $var in
val1)
        statements;;
val2)
        statements;;
*)
        statements;;
esac
```

Ejemplo case

```
#!/bin/bash
# Nombre archivo: case.sh
echo -n “Introduzca un numero entre 1 y 10: ”
 read x
case $x in
        1) echo “El valor de x es 1.”;;
        2) echo “El valor de x es 2.”;;
        3) echo “El valor de x es 3.”;;
        4) echo “El valor de x es 4.”;;
        5) echo “El valor de x es 5.”;;
        6) echo “El valor de x es 6.”;;
        7) echo “El valor de x es 7.”;;
        8) echo “El valor de x es 8.”;;
        9) echo “El valor de x es 9.”;;
        0 | 10) echo “Numero equivocado.”;;
        *) echo “Valor no reconocido.”;;
esac
```

Enunciados iteración

- Estructura for es usada para ciclar a través de un rango de variables.
- Sintaxis

```
for var in list
do
        statements
done
```

Ejemplo iteración

- Un clásico

```
#!/bin/bash
#Nombre archivo: itera
let sum=0
for num in 1 2 3 4 5
do
        let “sum = $sum + $num”
done
echo $sum
```

- Listado todos los archivos del directorio actual

```
#!/bin/bash
#Nombre archivo: lista
for x in *
do
    ls -l “$x”
    sleep 1
done
```

Enunciado while

- Estructura ciclo
- Usada para ejecutar un conjunto de comandos mientras una condición especificada es verdad
- Sintaxis

```
while expression
do
        statements
done
```

Ejemplo uso enunciado while

```
#!/bin/bash
# Nombre archivo: while.sh
echo –n “Enter a number: ”; read x
let sum=0; let i=1
while [ $i –le $x ];
do
    let sum=$sum+$i
    let i=$i+1
done
echo “the sum of the first $x numbers is: $sum”
```

Enunciado continue

- Provoca un salto a la siguiente iteración del ciclo, saltando los comandos restantes.

```
#!/bin/bash
# Nombre archivo: cont.sh
LIMIT=19
echo
echo “Printing Numbers 1 through 20 (but not 3 and 11)”
a=0
while [ $a -le “$LIMIT” ]; do
   a=$(($a+1))
   if [ “$a” -eq 3 ] || [ “$a” -eq 11 ]
   then
            continue
   fi
   echo -n “$a -”
done
```

Enunciado break

- Termina el cliclo (se sale de él)

```
#!/bin/bash
# Nombre archivo: break.sh
LIMIT=19
echo “Printing Numbers 1 through 20, but something happens  after 2 … ”
a=0
while [ $a -le “$LIMIT” ]; do
   a=$(($a+1))
   if [ “$a” -gt 2 ]
   then
      break
   fi
   echo -n “$a -”
done
echo; echo; echo
exit 0
```

Enunciado until

- Similar a la estructura while
- Cicla hasta que la condición es verdad
- Sintaxis

```
until [expression]
do
        statements
done
```

Ejemplo enunciado until

```
# !/bin/bash
# Nombre archivo: countdown.sh
echo “Enter a number: ”; read x
echo ; echo Count Down
until [ “$x” -le 0 ]; do
    echo $x
    x=$(($x –1))
    sleep 1
  done
echo ; echo GO !
```

Ejercicio

- Escribir un programa copiabin.sh que mueva todos los programas del directorio actual (archivos ejecutables) hacia el subdirectorio bin del directorio hogar del usuario, muestre los nombres de los que mueve e indique cuántos ha movido o que no ha movido ninguno. Si el directorio bin no existe, deberá ser creado.

Solución ejercicio