UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Curso Operador Linux

Módulo 5

SHELLS DE LINUX

Presentación

En esta unidad se introducirá a los participantes en el funcionamiento del intérprete de comandos del sistema.

Objetivos

Los participantes al finalizar la Unidad:

- Conocerán los diferentes shells que se pueden encontrar en un sistema Linux.
- Comprenderán el funcionamiento de las entradas y salidas estándar del sistema y poder manipularlas.
- Comenzaran a entender el manejo de procesos y las entradas - salidas estándar del sistema.
- Comprenderán el funcionamiento de un script.

Temario

5.1 Comprensión de los shells
5.2 Metacaracteres
5.3 Entrada y salida de Comandos (redirecciones)
5.4 Alias de comandos de shell
5.5 Comprensión de los shell scripts

Actividad de aplicación del conocimiento

Los participantes encontraran la actividad de aplicación en un archivo por separado en caso de ser requerida.

Examen

Los participantes deberán rendir el examen online o presentar el material solicitado según corresponda.

5.1 Comprensión de los shells

En los sistemas operativos hay un programa especial llamado Shell (también conocido como intérprete de comandos) que acepta instrucciones o comandos (normalmente en ingles) que, si son validos, son pasados al Kernel.

El shell no es parte del kernel del sistema pero utiliza al mismo para ejecutar programas, crear archivos, etc. Es un intérprete que ejecuta los comandos leídos del dispositivo de entrada estándar (teclado por ejemplo) o bien un archivo.

En GNU/Linux hay disponibles varios shells, se pueden citar:

Nombre del Shell:	Desarrollado por:	Detalles
BASH (Bourne Again Shell)	Free Software Foundation	Es el shell mas comun en Linux
CSH (C Shell)	University of California	La sintaxis de este shell es muy similar a la del lenguaje de programación C
KSH (Korn Shell	AT&T BELL Labs	--
TCSH	--	TCSH es una versión mejorada y totalmente compatible del UNIX C Shell (CSH)

Cualquiera de los anteriores shells leen los comandos del usuario, a través del teclado o mouse, y le dicen al sistema operativo lo que el usuario desea hacer. Si los comandos se están ingresando mediante el teclado se dice que se ingresan mediante línea de comandos.

Sh y bash Shell

Debido al éxito he historia del Bourne Shell utilizado en Unix, utilizado para escribir infinidad de scripts, es que se han incluido – algunos de ellos – en Linux.

La extensión del shell, llamándolo ahora bash, incluye muchas extensiones y nuevas funcionalidades que no se encontraban en su versión original. Se ha mantenido al compatibilidad por lo que se pueden correr scripts sh en bash.

En Linux sh es un link hacia bash. Al ser llamado mediante sh Linux intentara emular lo mejor posible al Bourne Shell original.

Startup Files

Al arrancar un shell se ejecutan los archivos de inicialización. El shell ejecuta primero los comandos en /etc/profile

A continuación busca los archivos /.bash_profile, /.bash_login, y /.profile, en este orden, ejecutando los comandos que encuentre en el orden que encuentra los archivos. Podemos poner nuestros propios comandos de configuración para anular los que se encuentran en /etc/profile.

Al desloguearnos de sistema, bash ejecuta los comandos que se encuentran en ~/.bash_logout. En este normalmente se encuentran los comandos para limpiar la sesión como ser el borrado de archivos temporales.

Caracteres especiales

Carácter	Descripción
NEWLINE	Initiates execution of a command
;	;
( )	Groups commands for execution by a subshell or identifies a function
&	Executes a command in the background
|	Sends standard output of preceding command to standard input of following command (pipe)
>	Redirects standard output
>>	Appends standard output
<	Redirects standard input
<<	Here document
*	Any string of zero or more characters in an ambiguous file reference
?	Any single character in an ambiguous file reference
\	Quotes the following character
´	Quotes a string, preventing all substitution
´´	Quotes a string, allowing only variable and command substitution
´…´	Performs command substitution
[]	Character class in an ambiguous file reference

$	References a variable
.(dot builtin)	Executes a command
#	Begins a comment
{ }	Used to surround the contents of a function
: (null builtin)	Returns true
&& (Boolean AND)	Executes command on right only if command on left succeeds (returns a zero exit status)
| |	Executes command on right only if command on left fails (returns a nonzero exit status)
! (Boolean NOT)	Reverses exit status of a command
$()	Performs command substitution
[ ]	Evaluates an arithmetic expression

Sintaxis	**Descripción**
cmd &	Execute cmd in background
cmd1 ;cmd2	Command sequence
(cmd1 ; cmd2)	Execute commands in a subshell
cmd1 | cmd2	Use output from cmd1 as input to cmd2
cmd1 ´cmd2´	Use otput from cmd2 as arguments to cmd1
cmd1 && cmd2	Execute cmd2 only if cmd1 succeeds (AND)
cmd1 || cmd2	Execute cmd2 only if cmd1 fails (OR)

5.2 Metacaracteres

Cuando se trabaja en la línea de comando es muy necesario contar con una herramienta que nos permita integrar varios nombres de archivos, con el uso de uno o más caracteres especiales

Esta técnica es muy común en todos, o la mayoría de los shell existentes, pero a pesar de que en Linux los caracteres que se utilizan son muy parecidos a los utilizados en otros sistemas operativos, no siempre tienen el mismo significado y/o resultado. En algunos casos el mal uso de estos metacaracteres puede traernos consecuencias no deseadas

Se entiende por metacaracteres a los caracteres especiales utilizados para remplazar uno o más caracteres comunes.

El asterisco «*», metacaracter universal.

El asterisco * reemplaza uno o más caracteres alfanuméricos incluyendo caracteres especiales como «,», «.», «+», «-». etc. exceptuando el punto «.» al inicio de un nombre de archivo o directorio.

Ejemplo: En un directorio existen los archivos «.agenda», «a.g.e.n.d.a», «agenda» y «agenda.»

```
# ls *
a.g.e.n.d.a  agenda  agenda.
#  ls .*
.agenda

.:
a.g.e.n.d.a  agenda  agenda.
#
```

Como se puede observar, al ejecutar «ls *» solamente se pudo ver tres archivos: a.g.e.n.d.a, agenda y agenda.; al ejecutar «ls .* », se visualizó, primero, el archivo «.agenda», luego el contenido del directorio «.» y por último el contenido del directorio «..».

Como se habrá notado no es necesario utilizar «*.*» para reemplazar todos los archivos de un directorio. Es necesario destacar que en Linux no existe el concepto de extensión de archivo como en DOS o Windows. En Linux, como en todos los Unix, un archivo puede contener varios puntos, al igual que otros carácteres especiales como la coma «,», los dos puntos «:», etc.

Por lo tanto, si ejecutamos «ls *.*» solamente se verán los archivos que contienen un punto (después del primer carácter al menos) dentro de su nombre

El símbolo de pregunta «?», metacarácter de posición

El símbolo de pregunta «?» reemplaza un único carácter y en la posición donde se encuentra el símbolo. Este metacarácter reemplaza cualquier carácter, incluyendo el punto «.» a menos que este punto sea el primer carácter del nombre del archivo o directorio

Ejemplo: En un directorio tengo los archivos «agen», «Agenda», «Lgenda», «.agenda»

```
# ls ?gen*
agen  Agenda  Lgenda

 # ls .?gen*
.agenda
```

En este ejemplo notamos que el signo de pregunta «?» reemplazo la letra «A», la «a» y la «L», pero no el «.». Para poder visualizar el archivo «.agenda» hubo que poner el punto «.» expresamente

Es importante destacar que a diferencia del asterisco «*», el signo de pregunta es un metacarácter de posición, eso quiere decir que solo reemplazará cualquier carácter que se encuentre en esa posición y únicamente en esa posición

5.3 Entrada y salida de Comandos (redirecciones)

En Linux, así como en todos los Unix, los procesos manejan por separado sus vías de entrada, así como también dos tipos de salida, la estándar y la de error.

Se entiende por entrada estándar a lo que puede recibir un proceso como ingreso de información durante su ejecución y salida, todo lo que el proceso devuelve al usuario.

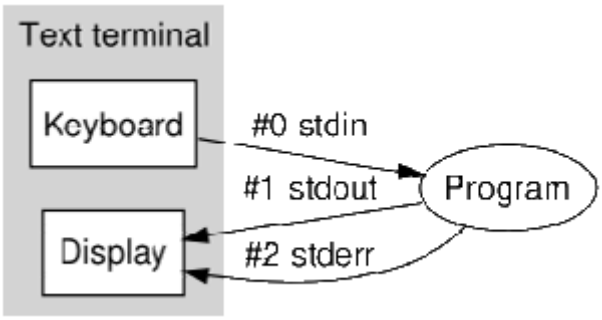
Por defecto, la entrada de un proceso es el teclado, en el caso de la salida sería el monitor.

Por ejemplo, la salida estándar del comando «ls» sería el listado de archivos y directorios que aparece en pantalla. En el caso de la entrada estándar de un comando lo podríamos ejemplificar con el comando «cat», utilizando como entrada el archivo a imprimir.

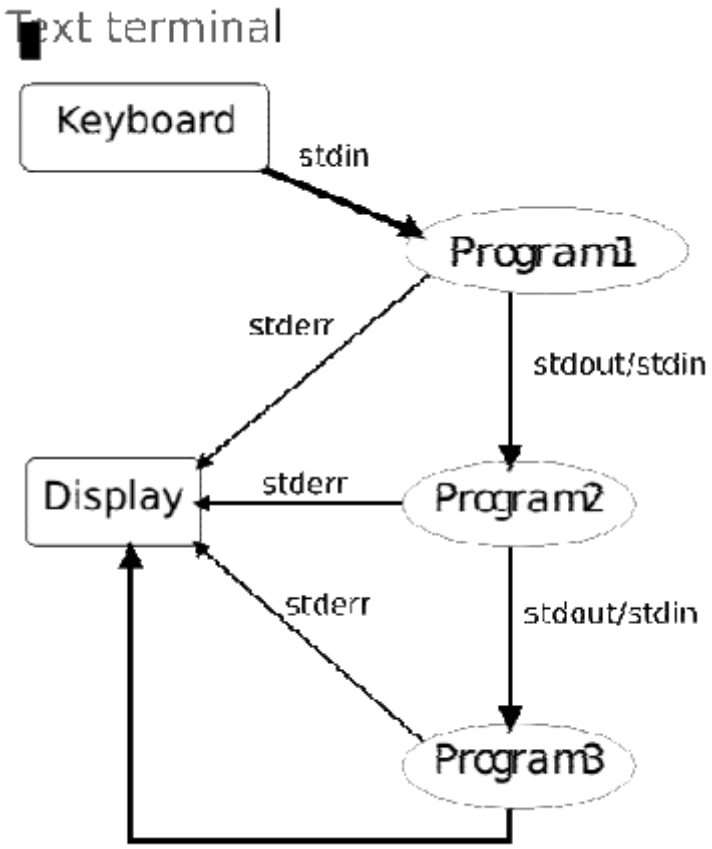
En el caso de la salida de error, serían todos los errores que devuelve un comando. Es posible en Linux, atrapar esta salida y manipularla en forma diferente.

El redireccionamiento y manipulación de las salidas de los procesos es una herramienta muy útil para la administración y monitoreo de un sistema Linux.

El siguiente grafico muestra un programa al que ingresa la entrada estándar (STDIN ó #0) y sus salidas son las salida estándar (STDOUT ó #1) y la salida de error (STDERR ó #2)



El siguiente ejemplo muestra un pipeline. En este caso se ingresa al programa mediante STDIN la salida de error del mismo se envía al display y la salida STDOUT se utiliza como entrada el segundo programa y así sucesivamente.



Redireccionando las salidas de los comandos

Uso del símbolo ">"

El símbolo mayor se puede utilizar para enviar la salida (estándar o de error) de un proceso a un archivo o dispositivo. Esto se utiliza para poder monitorear con más detenimiento la salida de un proceso o utilizar la misma para, luego de algún tipo de edición o manipulación del archivo creado, utilizarlo como entrada de algún otro proceso, o como simple reporte.

Es importante entender que cuando se redirecciona la salida de un proceso, utilizando esta técnica, esta salida, queda estática en algún archivo o dispositivo, y el archivo destino o el contenido del dispositivo es sobrescrito sin mayores avisos.

Las salidas de los procesos se diferencian en salida de error y salida estándar.

Salida de error (stderr)

Por la salida de error de un proceso se envían todos los mensajes que produce un comando cuando éste encuentra un problema.

Para atrapar la salida de error se utiliza el identificador «2». A continuación se pueden observar algunos ejemplos:

```
# ls /undirectorioinexistente
ls: cannot access /undirectorioinexistente: No such file or directory
#
```

Si quisiéramos atrapar el mensaje de error del comando anterior debería hacer lo siguiente:

```
# ls /undirectorioinexistente 2>/tmp/err.log
#  cat /tmp/err.log
ls: cannot access /undirectorioinexistente: No such file or directory
#
```

El mensaje de error ahora, quedo en el archivo al cual fue redireccionado (en este caso "/tmp/err.log").

Salida estándar (stdout)

Por la salida estándar de un proceso se envían todos los mensajes que no son considerados errores. Por ejemplo, la salida del comando ls - el listado de archivos y directorios - es la salida estándar.

Para atrapar la salida de estándar se utiliza el identificador «1» o ningún identificador, ya que es el default. A continuación se citan algunos ejemplos:

```
# ls -FaC /
./          .gnome/          boot/  home/   mnt/   root/  usr/
../         .gnome_private/  dev/   lib/    opt@   sbin/  var/
```

Si quisiera atrapar la salida del ejemplo, debería hacer lo siguiente:

```
# ls -FaC / > /tmp/stdout.txt
#
# cat /tmp/stdout.txt
./          .gnome/          boot/  home/   mnt/   root/  usr/
../         .gnome_private/  dev/   lib/    opt@   sbin/  var/
#
```

Como se puede observar, la salida del primer comando fue íntegramente atrapada al redireccionarla hacia el archivo /tmp/stdout.txt.

Uso del símbolo ">>"

Es posible que al redireccionar una salida (sea de error o estándar), no quiera sobrescribir el contenido del archivo donde se encuentra la salida. Para realizar esto podrá utilizar este símbolo.

Utilizando esta técnica, cuando redireccionemos una salida, el archivo destino no se sobrescribirá, toda la salida será agregada al final del archivo, dejando, como estaba, el contenido previo del archivo.

```
# ls -l / > /tmp/salida.txt

# cat /tmp/salida.txt
drwxr-xr-x      2       root    root    2048    Nov  4 01:06    bin
drwxr-xr-x      2       root    root    1024    Oct  26 10:35   boot
drwxr-xr-x      5       root    root    34816   Nov 6 15:58     dev
drwxr-xr-x      30      root    root    3072    Nov  6 15:53    etc
drwxr-xr-x      2       root    root    12288   Oct 25 19:18    lost+found
drwxr-xr-x      6       root    root    1024    Oct 26 14:51    mnt
```

```
ls -FaC /usr >> /tmp/salida.txt
#
# cat /tmp/salida.txt
drwxr-xr-x      2       root    root    2048    Nov  4 01:06    bin
drwxr-xr-x      2       root    root    1024    Oct 6 10:35     boot
drwxr-xr-x      5       root    root    34816   Nov 6 15:58     dev
drwxr-xr-x      30      root    root    3072    Nov  6 15:53    etc
drwxr-xr-x      6       root    root    1024    Oct 26 13:52    home
drwxr-xr-x      4       root    root    3072    Oct 25 19:26    lib
drwxr-xr-x      2       root    root    12288   Oct 25 19:18    lost+found
drwxr-xr-x      6       root    root    1024    Oct 26 14:51    mnt

./      bin/   etc/            include/  libexec/     man/    shared/
../     dict/  games/          info/     local/       sbin/   src/

X11R6/  doc/   i486-linux-libc5/  lib/     lost+found/  share/  tmp@
#
```

En el primer comando se redirecciona al archivo /tmp/salida.txt la salida del comando «ls -l /», luego, se agrega la salida del comando «ls -FaC /usr/» a este mismo archivo. De esta forma se obtuvo la salida de dos comandos en un mismo archivo, ya que no se sobrescribió.

Unificando la salida de error y la salida estándar

En algunos casos es necesario unificar la salida de error y la salida estándar para tener todo en un mismo archivo. Existen muchas maneras para realizar esto, en este manual sólo se describe una, ya que es la más tradicional y utilizada.

Uso del "&" como unificador

Para unificar las salidas de un comando, no solo utilizaremos el símbolo &, sino que también se utilizaran los números 1 y 2 para identificar la salida estándar y de error, respectivamente.

```
# ls -FaC /var /VariableS
ls: /VariableS: No such file or directory
./  cache/  db/  lib/  lock/  nis/     run/  tmp/
../  catman/  gdm/  local/  log/  preserve/  spool/  yp/

#
```

Como se puede observar el directorio /VariableS no existe; no es el caso del directorio /var.

Si tan solo redireccionamos la salida estándar, el mensaje de error no es atrapado y en consecuencia se despliega en la pantalla.

```
# ls -FaC /var /VariableS > /tmp/salida.txt
ls: /VariableS: No such file or directory

#
```

Lo contrario sucede cuando atrapamos la salida de error, el mensaje de error es atrapado y redireccionado al archivo /tmp/error.txt, pero la salida estándar se termina desplegando en la pantalla, sin ser atrapada.

```
# ls -FaC /var /VariableS 2> /tmp/error.txt
./  cache/  db/  lib/  lock/  nis/     run/  tmp/
../  catman/  gdm/  local/  log/  preserve/  spool/  yp/

#
```

Utilizando el símbolo &, podrá atrapar las dos salidas (estándar y error) y redireccionarlas a un mismo archivo.

```
# ls -FaC /var/VariableS > /tmp/Salida-y-Error.txt 2>&1

# cat /tmp/Salida-y-Error.txt
ls: /VariableS: No such file or directory
./   cache/   db/   lib/    lock/  nis/       run/    tmp/
../  catman/  gdm/  local/  log/   preserve/  spool/  yp/
```

Es importante tener en cuenta el orden que se utilizan en las unificaciones, el 2>&1 va al final

Es importante tener en cuenta el orden que se utiliza en las unificaciones, el 2>&1 va al final.

Redireccionando la salida de un proceso a otro (PIPE)

En la sección anterior se vio como hacer para enviar las salidas a un archivo, existe también la manera de utilizar la salida de un proceso como entrada de otro proceso. Para realizar esto utilizaremos pipes (cañerías en inglés) y se simbolizan con este carácter: «|»

```
# ls -l
total 65
drwxr-xr-x      2       root    root     2048    Nov  4 01:06    bin
drwxr-xr-x      2       root    root     1024    Oct  26 10:35   boot
drwxr-xr-x      5       root    root    34816    Nov  6 15:58    dev
drwxr-xr-x      30      root    root     3072    Nov  6 15:53    etc
drwxr-xr-x      6       root    root     1024    Oct 26 13:52    home
drwxr-xr-x      4       root    root     3072    Oct 25 19:26    lib
drwxr-xr-x      2       root    root     1228    Oct 25 19:18    lost+found
drwxr-xr-x      6       root    root     1024    Oct 26 14:51    mnt

#  ls -l | grep home
drwxr-xr-x   6 root    root       1024 Oct 26 13:52 home
drwxr-xr-x   8 root    root       1024 Oct 28 21:30 vhome
#
```

Acá se puede ver el uso de los pipes para filtrar una salida extensa. En este ejemplo se selecciono de la salida del <<ls –l>>, solamente aquellas líneas que contengan la palabra home.

Esta herramienta es muy poderosa y se puede utilizar en forma encadenada, teniendo varios procesos unidos por pipe.

Bifurcaciones (TEE)

Cuando la cadena de pipes es muy larga y necesitamos ver la salida de un comando que está en el medio de esa cadena, se puede utilizar la bifurcación de pipes, esta técnica nos permite guardar en un archivo todo los caracteres que están pasando en ese lugar del pipe, y también enviarlo como salida del proceso (o entrada del proceso siguiente). De esta forma obtendrá en un archivo la información requerida y no romperá la cadena de pipes.

Las bifurcaciones se hacen con el comando «tee».

```
# ls -l | tee /tmp/bif.txt | grep home
drwxr-xr-x   6 root     root        1024 Oct 26 13:52 home
drwxr-xr-x   8 root     root        1024 Oct 28 21:30 vhome

#  cat /tmp/bif.txt
total 65
drwxr-xr-x        2 root     root        2048 Nov  4 01:06 bin
drwxr-xr-x        2 root     root        1024 Oct 26 10:35 boot
drwxr-xr-x        5 root     root       34816 Nov  6 15:58 dev
drwxr-xr-x       30 root     root        3072 Nov  6 15:53 etc
drwxr-xr-x        6 root     root        1024 Oct 26 13:52 home
drwxr-xr-x        4 root     root        3072 Oct 25 19:26 lib
drwxr-xr-x        2 root     root       12288 Oct 25 19:18 lost+found
drwxr-xr-x        6 root     root        1024 Oct 26 14:51 mnt
lrwxrwxrwx        1 root     root          10 Oct 26 16:35 opt -> /usr/local
dr-xr-xr-x       74 root     root           0 Nov  4 19:44 proc
drwxr-x---       13 root     root        1024 Nov  6 16:17 root
drwxr-xr-x        3 root     root        2048 Nov  4 01:06 sbin
drwxrwxrwt       15 root     root        1024 Nov  6 17:21 tmp
drwxr-xr-x       20 root     root        1024 Nov  4 01:04 usr
drwxr-xr-x       16 root     root        1024 Oct 25 19:28 var
drwxr-xr-x        8 root     root        1024 Oct 28 21:30 vhome
```

Como podrá ver, el comando «tee /tmp/bif.txt», grabó en el archivo pasado como argumento lo que obtuvo como entrada estándar (la salida del comando ls), luego de escribir el archivo, envió eso mismo al otro comando «grep».

De esta forma, la cadena de pipes sigue respondiendo como antes, pero se obtuvo un archivo con lo que estaba transitando por la cadena.

Esto es muy útil cuando tenemos problemas dentro de una cadena larga y necesitamos ver el contenido de un paso, sin romper la cadena de pipes

5.4 Alias de comandos de shell

Alias es un comando que posibilita el reemplazo de una palabra por otra serie de palabras. Se utiliza para abreviar un comando o bien para agregar argumentos a un comando muy utilizado.

Los alias se creen en el momento mediante la sentencia <alias> y el par nombre argumento

```
# ls
agen  a.g.e.n.d.a  Agenda  archivo  error.log  Lgenda
#
# alias ls='ls -la'
#
# ls
total 16
drwxr-xr-x  9 root root  1024 2008-11-02 08:07 .
drwxr-xr-x 22 root root  1024 2008-10-25 21:06 ..
-rw-r--r--  1 root root     0 2008-11-01 13:19 agen
-rw-r--r--  1 root root     0 2008-11-01 13:20 .agenda
-rw-r--r--  1 root root     0 2008-11-01 13:19 a.g.e.n.d.a
-rw-r--r--  1 root root     0 2008-11-01 13:18 Agenda
drwx------  2 root root  1024 2008-10-25 18:53 .aptitude
-rw-r--r--  1 root user    0 2008-10-30 00:44 archivo
-rw-------  1 root root  2756 2008-11-02 08:06 .bash_history
-rw-r--r--  1 root root   429 2008-11-02 07:48 .bashrc
```

Un alias durara lo que dure la sesión. Para evitar perder los alias, y en caso de ser utilizados recurrentemente, deberemos configurarlos el los archivos de configuración del shell pudiendo hacerlo para la sesión de usuario o bien para todos los usuarios. Esto se realiza en el archivo ~/.bashrc del directorio home del usuario o bien en /etc/bashrc para todos los usuarios de bash del sistema.

Para ver los alias definidos en sistema utilizamos el mismo comando pero sin parámetros

```
# alias
alias l=´ls -CF´
alias la=´ls -A´
alias ll='ls -l'
alias ls='ls --color=auto'

# ll
total 52068
-rw-r--r-- 1 user user     1734 2008-10-26 01:08 61B8DB62.gpg
drwxr-xr-x 3 user user     4096 2008-11-02 07:51 Desktop
-rw-r--r-- 1 user  user  53236677 2008-05-16 02:28 VMwareTools.tar.gz
drwxr-xr-x 7 root  root      4096 2008-05-16 02:28 vmware-tools-distrib
#
```

En caso de ya existir un alias de un comando es posible imponerse a el por medio del uso de comillas.

Para remover un alias se utiliza el comando <unalias>

```
# unalias ls
#
#  ls
agen  a.g.e.n.d.a  Agenda  archivo  error.log  Lgenda

#
```

5.5 Comprensión de los shell scripts

¿Qué es un Shell Script?

Es una serie de comandos escritos en texto plano en un archivo. Este archivo se pasa a un intérprete de comandos que realizara las acciones programadas en el script.

Como escribir un Script

Editar un archivo (por ejemplo con vi) para escribir el script

```
#  vi prueba.sh
#!/bin/sh
#
#  Mi primer script
#
clear
echo "Knowledge is
Power"
:wq
```

Modificar los permisos del archivo creado agregándole permisos de ejecución

```
# chmod +x prueba.sh

ó

#  chmod 755 prueba.sh
```

Ejecutar el script

```
$ bash prueba.sh

ó

$  sh prueba.sh

ó

$  ./prueba.sh
```

La sintaxis ./ significa ejecutar en el directorio actual. La sentencia para ejecutar la da el punto <<.>>.

Llamado del intérprete

Para que el sistema llame al intérprete correcto nuestro script debe indicar al comienzo cual es el intérprete que utilizamos

```
#!/bin/bash

ó

#!/bin/sh
```

Comentarios

Los comentarios se realizan por medio del símbolo numeral #

```
#!/bin/bash
#  Este es un script de ejemplo para el curso Operador Linux
```

Variables

Para inicializar una variable hacemos lo siguiente

VAR1=”contenido”

Y para ver el contenido de la variable la debemos llamar mediante el símbolo $. Por ejemplo para imprimir en pantalla el contenido de la variable VAL1 haremos lo siguiente

echo $VAR1

Hay ciertas variables que pertenecen al sistema operativo y podrán ser utilizadas por nuestro script, podemos citar las siguientes

$SHELL Ubicación del shell del usuario
$HOME Directorio inicial del usuario
$PATH Ruta de los ejecutables
$PS1 Prompt actual
$USERNAME
$HOSTNAME

Hay también variables relacionadas con los argumentos que se pasan al script por línea de comandos, veamos los siguientes

$0	Contiene el nombre con el que se llama al script
$1 a $20	Contienen los valores de los argumentos con los que se llamo al script
$*	Es una cadena con los argumentos concatenados
$#	Cantidad de argumentos con los que se llamo al script

Estado de salida

Al finalizar la ejecución de un comando o script Linux devuelve dos tipos de valores utilizados para saber si la ejecución finalizo correctamente o no.

El valor devuelto será cero si el comando se ejecuto sin problemas.

El valor devuelto será distinto a cero si no s ejecuto correctamente o si se encontraron errores.

Estos valores son conocidos como Exit Status.

```
#!/bin/bash
# Este es un script de ejemplo para el curso Operador Linux
clear

echo “Primer parámetro: $1”
echo “Segundo parámetro:” $2

echo “Cantidad de parámetros:” $#
echo “Parámetros concatenados:” $*

echo “Suma de los parámetros:”
expr $1 $2
echo $? “ esta es la salida de error”
```

Ejecutamos el script con 2 argumentos numéricos, en este caso los números 5 y 3

```
#./prueba.sh 5 3
Primer parámetro: 5
Segundo parámetro: 3
Cantidad de parámetros: 2
Parámetros concatenados: 5 3
Suma de los parámetros:
8
0 esta es la salida de error
```

Ejecutamos el script con 2 argumentos, uno numérico y otro alfanumérico

```
#./prueba.sh 5 texto
Primer parámetro: 5
Segundo parámetro: texto
Cantidad de parámetros: 2
Parámetros concatenados: 5 texto
Suma de los parámetros:
expr: non-numeric argument
 3 esta es la salida de error
```

La sentencia read

Se utiliza para ingresar datos por medio del teclado y cargarlos en variables. Esta sentencia detiene el script hasta que se introduzcan datos por línea de comandos.

```
#!/bin/bash
# Este es un script de ejemplo para el curso Operador Linux
clear
echo “Ingrese su nombre”
read nombre
echo “Bienvenido al sistema $nombre, espero que disfrute su estadía!”
```

Comparaciones

Se utilizan para obtener un resultado y tomar acciones a partir del mismo.

-z	Verdadero si la longitud de una cadena es cero
-n	Verdadero si la longitud de una cadena es distinta a cero
string1 = string2	Verdadero si los strings son iguales
string1 != string2	Verdadero si los strings son distintos

```
#!/bin/bash
# Este es un script de ejemplo para el curso Especialista Linux
clear
echo “Ingrese su nombre”
read nombre
if [ -z $nombre ] ;then
echo “No ha ingresado si nombre”
else
echo “ Bienvenido al sistema $nombre, espero que disfrute su estadía!”
```

Comparaciones para operaciones aritméticas

Mathematical Operator in Shell	Meaning	Normal Arithmetical	Shell	
			For test statement with if command	For [expr] statement with if command
-eq	is equal to	5 == 6	if test 5 -eq 6	if [5 -eq 6]
-ne	is not equal to	5 != 6	if test 5 -ne 6	if [5 -ne 6]
-lt	is less than	5 < 6	if test 5 -lt 6	if [5 -lt 6]
-le	is less than or equal to	5 <= 6	if test 5 -le 6	if [5 -le 6]
-gt	is greater than	5 > 6	if test 5 -gt 6	if [5 -gt 6]
-ge	is greater than or equal to	5 >= 6	if test 5 -ge 6	if [5 -ge 6]

Comparaciones entre cadenas de caracteres (strings)

Operator	Meaning
string1 = string2	string1 is equal to string2
string1 != string2	string1 is NOT equal to string2
string1	string1 is NOT NULL or not defined
-n string1	string1 is NOT NULL and does exist
-z string1	string1 is NULL and does exist

Comparaciones para archivos y directorios

Test	Meaning
-s file	Non empty file
-f file	Is File exist or normal file and not a directory
-d dir	Is Directory exist and not a file
-w file	Is writeable file
-r file	Is read-only file
-x file	Is file is executable

Operadores Lógicos

Operator	Meaning
! expression	Logical NOT
expression1 -a expression2	Logical AND
expression1 -o expression2	Logical OR

La sentencia test o expr

Se utiliza para verificar si una expresión es verdadera o falsa. En caso que la evaluación sea verdadera regresara el valor cero y un valor distinto a cero en caso contrario.

```
#!/bin/bash
#  Este es un script de ejemplo para el curso Especialista Linux
clear
echo “Ingrese un numero”
read numero
if test $numero -gt 0
then
echo "$numero es un numero positivo"
fi
```

La sentencia if

Es utilizada para realizar toma de decisiones en el script. Si se cumple la condición evaluada se ejecuta el comando.

```
#!/bin/sh
# Este es un script de ejemplo para el curso Especialista Linux
#
if cat $1
then
echo -e "\n\nArchivo $1, archivo encontrado y mostrado en pantalla"
fi
```

```
#!/bin/sh
# Este es un script de ejemplo para el curso Especialista Linux

if [ $# -eq 0 ]
then
echo "$0 : Debe introducir un numero entero como argumento"
exit 1
fi

if test $1 -gt 0
then
echo "$1 es positivo"
else
echo "$1 es negativo"
fi
```

Sentencia for

Esta sentencia crea una variable que servirá para contabilizar las iteraciones. El proceso continuara hasta que se terminen las iteraciones.

```
#!/bin/sh
#
if [ $# -eq 0 ]
then
echo "Error - debe ingresar un argumento en la línea de comando"
echo "Sintaxis: $0 numero"
echo "Utilizado para imprimir la tabla de multiplicar del numero dado"
exit 1
fi
n=$1
for i in 1 2 3 4 5 6 7 8 9 10
do
echo "$n * $i = `expr $i \* $n`"
done
```

Sentencia while

La sentencia se ejecuta mientras una dada condición es verdadera.

```
#!/bin/sh
#
if [ $# -eq 0 ]
then
echo "Error - debe ingresar un argumento en la línea de comando"
echo "Sintaxis: $0 numero"
echo "Utilizado para imprimir la tabla de multiplicar del numero dado"
exit 1
fi
n=$1
i=1
while [ $i -le 10 ]
do
    echo "$n * $i = `expr $i \* $n`"
     i=`expr $i + 1`
done
```

Sentencia case

Posibilita la comparación de una variable contra múltiples valores.

```
#!/bin/sh
#
if [ -z $1 ]
then
        renta="*** Vehiculo desconocido ***"
elif [ -n $1 ]
then
                renta=$1
fi
case $renta in
                "auto") echo "El alquiler de $renta es de \$20 por km";;
                "camioneta") echo " El alquiler de $renta es de \$50 por km ";;
                "jeep") echo " El alquiler de $renta es de \$5 por km ";;
                "bici") echo " El alquiler de $renta es de \$2 por km ";;
                *) echo "Lo sentimos, no podemos alquiarle un $renta";;
Esac
```

Varios comandos en la misma línea

Para poner varios comandos en la mima línea se utiliza el separador <<;>>. La sintaxis sería la siguiente: comando1;comando2

```
#!/bin/bash
# Este es un script de ejemplo para el curso Operador Linux
clear
echo “Ingrese su nombre”
read nombre
if [ -z $nombre ] ;then
echo “No ha ingresado si nombre”
else
echo “ Bienvenido al sistema $nombre, espero que disfrute su estadía!”
```

Debugging y depuración de scripts

Cuando programamos y tenemos errores o necesitamos depurar el código utilizamos el debug. Para esto utilizaremos los tres argumentos que se citan a continuación:

-v Imprime las líneas tal como son leídas del script
-x Ejecuta el script expandiendo cada comando y cada variable
-n Verifica los errores de sintaxis del script sin ejecutarlo