Curso Intensivo GNU/Linux

Febrero 27 2016

Daniel Mejía Raigosa
danielmejia55@gmail.com

Recapitulación sesión pasada...

- La línea de comandos también conocida como el prompt, shell, o consola
- Es el mecanismo de más bajo nivel para la ejecución de tareas en Linux
- La línea de comandos tiene la posibilidad de usar las siguientes intérpretes
  - Bash (Por defecto)
  - Sh
  - Csh
  - Dash

Recapitulación sesión pasada...

Las variables de entorno contienen información a la que se accede a través del nombre de la variable (al igual que ocurre en los lenguajes de programación),

Las variables de entorno más relevantes son

- SHELL=/bin/bash
- TERM=xterm
- USER=*usuario*
- PATH=/usr/local/bin:/usr/bin:/bin:/usr/local/sbin:/usr/sbin:/sbin

el comando *env* permite visualizar las variables de entorno para el usuario

Estructura de los comandos en Linux

La estructura básica de un comando en linux es

```
$ [comando] [opciones] [argumentos] ...
```

Por ejemplo,

```
$ ls -l /home/
```

- ls es el comando para listar (ls = list).
- -l es la opción para presentar la salida del comando como una lista (-l=-list).
- /home/ es un argumento.

Páginas de manual o man pages

- La gran mayoría de los comandos en linux vienen acompañados de una página de manual o *man page*.
- Las páginas de manual man proporcionan una descripción detallada y completa de cada comando indicando.

Por ejemplo,

```
$ man tar
```

Proporciona una descripción detallada del comando *tar*

NAME

tar — The GNU version of the tar archiving utility

SYNOPSIS

tar [**-**] **A --catenate --concatenate | c --create | d --diff --compare | --delete | r --append | t --list | --test-label | u --update | x --extract --get** [options] [pathname ...]

DESCRIPTION

Tar stores and extracts files from a tape or disk archive.

The first argument to tar should be a function; either one of the letters **Acdrtux**, or one of the long function names. A function letter need not be prefixed with ``-'', and may be combined with other single-letter options. A long function name must be prefixed with **--**. Some options take a parameter; with the single-letter form these must be given as separate arguments. With the long form, they may be given by appending **=**value to the option.

FUNCTION LETTERS

Main operation mode:

-A, --catenate, --concatenate
append tar files to an archive

-c, --create
create a new archive

-d, --diff, --compare
find differences between archive and file system

--delete
delete from the archive (not on mag tapes!)

-r, --append
append files to the end of an archive

-t, --list
list the contents of an archive

EXAMPLES

Create archive.tar from files foo and bar.

```
tar -cf archive.tar foo bar
```

List all files in archive.tar verbosely.

```
tar -tvf archive.tar
```

Extract all files from archive.tar.

```
tar -xf archive.tar
```

SEE ALSO

tar(5), symlink(7), rmt(8)

Comandos de uso frecuente

- *ls*
- *cat*
- *mv*
- *cp*
- *rm*
- *mkdir*
- *pwd*
- *less*
- *tree*
- *head*
- *tail*
- *echo*
- *grep*
- *locate*
- *find*
- *ps*
- *ssh*
- *zip*
- *unzip*
- *tar*
- *sleep*
- *editores de texto (nano, pico, emacs, vim)*

Operador de tubería o Pipe |

- El resultado de la ejecución de un comando puede ser redireccionado hacia otro comando mediante el operador de tubería o “pipe” |
- Se pueden concatenar cuantas tuberías se quiera

Por ejemplo

```
$ ls -l | less
```

Operador de redirección de stream >

- El resultado de la ejecución de un comando puede ser redireccionado hacia un archivo mediante el redireccionador de stream “>”

Por ejemplo

```
$ ls -l > mi_archivo.txt
```

Operador de redirección de stream <

- Se puede ejecutar un comando haciendo llamada a un archivo en disco mediante el redireccionador de stream “<” (reemplaza entrada standard)

Por ejemplo

```
$ cat < mi_archivo.txt
```

Ejecución condicional && y ||

- commando1 && commando2:
  comando2 se ejecuta si, y sólo si, el comando 1 se ejecuta satisfactoriamente (valor de salida 0)
  command2 is executed if, and only if, command1 returns an exit status of zero.

- command1 || command2:
  comando2 se ejecuta si, y sólo si, el comando 1 falla al ejecutarse (valor de salida 1)

$ rm myf && echo "Archivo borrado" || echo "Archivo no borrado”

Introducción al scripting

- Algunas características dependen del shell.
  Para ver shell disponibles ejecutar
  $ cat /etc/shells
- Los scripts shell se escriben en archivos de texto plano (.bash, .sh)
- Un script shell permite interacción con el usuario, con archivos en el disco, con variables en memoria, mediante procesos I/O.

- Útil para abreviar tareas (crear comandos nuevos) y ahorra tiempo

- Útil para la automatización

- Administración de sistema también es automatizable

Introducción al scripting

Requisitos:

- Editor de texto plano (vim, nano, emacs)
- Añadir permisos de ejecución al script usando
  $ chmod permiso archivo_script

Ejemplo:

$ chmod +x script_back-up-usuarios.sh

Estructura básica de un script

#!/bin/bash -> cabecera (indica el interprete a usar)

Los comentarios se escriben con # inicial -> comentarios

echo “Hola usuario $USER” -> Ejecución de comandos

...

exit 0 -> valor de retorno (opcional, para depuración)

Scripting y variables

Hay dos tipos de variables:

- Variables de sistema(entorno): Variables de entorno automáticas de GNU/Linux. Estas variables figuran con letras mayúsculas ($ env o $ set muestran las variables disponibles)
- Variables definidas por el usuario: Creadas y mantenidas por el usuario. Estas variables suelen usar letras minúsculas.

Ingreso de datos por teclado

```
#!/bin/bash
echo "Cuál es tu nombre?:"
read fname
echo "Hola $fname, seamos amigos!"
```

Ejecución de comandos en script

```
#!/bin/bash
clear
echo "Hola $USER"
echo "Hoy es \c ";date
echo "Número de logins de usuario : \c" ; who | wc -l
echo "Calendario del mes:"
cal
exit 0
```

Ejecución de comandos en script ($())

```
#!/bin/bash
logins=$(who | wc -l)
fecha=$(date)
clear
echo "Hola $USER"
echo "Hoy es $fecha "
echo "Número de logins de usuario $logins”
echo "Calendario del mes:"
cal
exit 0
```

Ejecución de comandos en script (``)

```
#!/bin/bash
who | wc -l
date
clear
echo "Hola $USER"
echo "Hoy es `date` "
echo "Número de logins de usuario `who | wc -l`"
echo "Calendario del mes:"
cal
exit 0
```

Ejecución de comandos en script

```
#!/bin/bash
who | wc -l
date
clear
echo "Hola $USER"
echo "Hoy es `date` "
echo "Número de logins de usuario $(who | wc -l)"
echo "Calendario del mes:"
cal
exit 0
```

Pasar argumentos a script

```
#!/bin/bash

echo “El argumento dado fue $1”
```

```
#!/bin/bash
argumento = $1
echo “El argumento dado fue $argumento”
```

Comando de ensayo (test) [...]

Expresión	Significado	Ejemplo
eq	Igual que	[5 -eq 6]
ne	No es igual que	[5 -ne 6]
lt	menor que	[5 -lt 6]
le	menor o igual que	[5 -le 6]
gt	mayor que	[5 -gt 6]
ge	mayor o igual que	[5 -ge 6]
a	existe archivo	[-a mi_archivo.txt]

Condicional if … then … fi

```
#!/bin/bash

if cat $1; then
        echo -e "\n\nEl archivo $1, fue encontrado y mostrado en pantalla"
fi
```

```
#!/bin/bash

if [ 5 -le 6 ] ; then
        echo “ 5 es menor o igual que 6 ”
fi
```

Bucle for … do … done

Sintaxis

for { variable } in { lista }
do
líneas de código indendas

la indentación es importante para el orden visual

Se repite hasta el done mientras se cumpla la condición de bucle
done

Bucle for … do … done

```
#!/bin/bash
for i in 1 2 3 4 5
do
        echo “Hola por $i vez”
done
```

```
#!/bin/bash
for i in $(seq 1 5)
do
        echo “Hola por $i vez”
done
```

Bucle while... do ... done

Sintaxis:

```
while [ condición ]
 do
      commando1
      commando2
      commando3
      ..
      ....
  done
```

Bucle while... do ... done

```
#!/bin/bash
n=$1
i=1
while [ $i -le 10 ]
do
  echo "$n * $i = `expr $i \* $n`"
  i=`expr $i + 1`
done
```

Administración de paquetes de software

- .deb (DEBian)
  - apt-get
  - aptitude
  - dpkg
- .rpm (Red hat Package Manager)
  - rpm
  - yum
  - apt-rpm
- Sources (.tar.gz / .tar.xz / .pkg / .lzma / comprimidos)

Programando tareas con CRON

- CRON es un demonio para la gestión de tareas con marcas de tiempo a un minuto
- Generalmente se encuentra instalado y activo en las distro
- Depende de la hora(hh:mm DD/MM/YY) establecida en el sistema
- Cualquier usuario puede usar cron para automatizar sus tareas
- Sólo root puede crear tareas con nivel de administración
- Se programan editando una tabla llamada “crontab”
  $ crontab -e

Fundamentos de redes en GNU/Linux

Archivos de configuración importantes:

- /etc/hosts
  Añadir nombres de host de la red que no son dados por un DNS.

- /etc/resolv.conf
  Especifica direcciones de DNS, entre otros detalles. Se modifica automáticamente con scripts de inicialización

- /etc/network/interfaces
  Especifica las directrices de configuración de las interfaces de red.

Fundamentos de redes en GNU/Linux

- nestat
- ip
- ifconfig (/sbin/ifconfig)
- iwconfig (wireless)
- nmap

Fundamentos de redes en GNU/Linux

- nestat

  $ netstat --interfaces

- ip

  $ ip link show

- nmap

  $ nmap localhost

  $ nmap -A 192.168.0-5.0-190

Conexión remota con SSH

- SSH es un cliente Secure SHell
- Se utiliza para acceder al shell de máquinas remotas y ejecutar comandos en ellas
- La estructura más simple del comando es

  $ ssh usuario@host.dominio

- ssh examina el archivo /etc/hosts para ubicar nombres de servidores o nodos

  $ ssh admin@main

Administración de procesos

- *who*
- *ps*
- *jobs*
- *htop*
- *top*
- *lslocks*
- *kill*
- *killall*

Administración de servicios y recursos

- *service*
- *systemctl*
- *journalctl*
- *dmesg*
- *lsblk*
- *lsof*
- *lspci*
- *lsusb*
- *lscpu*
- *du*
- *df*