2 EL SISTEMA OPERATIVO LINUX

2.1 INTRODUCCION

EL ORIGEN DE UNIX Y LINUX

El sistema operativo UNIX surgió hace más de tres décadas. Desde entonces se han desarrollado otros sistemas operativos basados en UNIX y denominados *tipo UNIX*. Existen diferencias entre estos sistemas, pero son mucho más las coincidencias que las diferencias.

El sop UNIX tuvo un enorme aumento de popularidad a partir de la década de los noventas (y particularmente en los últimos años), gracias al surgimiento del sop LINUX.

LINUX es un sistema operativo "tipo UNIX"

EL sistema operativo Linux fue creado por Linus Torvald en la Universidad de Helsinki en Finlandia. Linus estaba interesado en crear y expandir las capacidades de ciertos sistemas UNIX. Comenzó su trabajo en 1991 y la versión 1.0 fue liberada en 1994.

Los sistemas operativos tipo Unix son los más utilizados en la investigación científica.

KERNEL Y DISTRIBUIONES

Linux utiliza la licencia GPL (General Public License) de GNU y su código fuente es libre y está disponible para cualquiera. Estrictamente hablando Linux es el **núcleo** (Kernel) del sop.

EL Kernel es el corazón del sistema operativo Linux. Maneja los recursos del sistema, los más importates son:

- Administración de E/S
- Administración de procesos
- Administración de dispositivos
- Administración del Sistema de Archivos
- Administración de la Memoria Central

El sop es el kernel Linux que viene con una *distribución* de software (Debian, Suse, RedHat, Ubuntu, etc.). Una de las principales ventajas de Linux (y todos los sistemas tipo UNIX) es que viene con una larga colección de programas estándar. Estos programas realizan una variedad muy amplia de tareas, desde el listado de archivos hasta leer el correo y todos ellos son accesibles a los usuarios de una forma acoplada. La

utilización de estos constituyen un verdadero lenguaje de programación en un entorno UNIX.

TERMINALES Y LOGINS

Para iniciar una sesión es necesario poder acceder a una *terminal*. Pueden destacarse dos tipos de terminales:

- Terminales de texto: consta solo de una pantalla en la que se imprimen caracteres. La única entrada es un teclado.
- Terminales gráficas: consta de una pantalla gráfica. En este modo se pueden emplear ventanas que emulan el comportamiento de una Terminal de texto (xterm o gnome-terminal)

Los sistemas Linux vienen ya con entornos gráficos para comenzar una sesión en un ambiente de trabajo en base a ventanas. El sistema *X Windows* es el sistema estándar de ventanas en las estaciones de trabajo. Es corriente que el sistema sea arrancado automáticamente cuando la máquina inicia (pero es opcional). Actualmente existen varios entornos de trabajo populares como GNOME y KDE.

> Para comenzar una sesión en Linux, debe uno tener un usuario registrado en el sistema. Dicho usuario se identifica con un *nombre de usuario* y una *clave* (password).

Para iniciar una sesión abrimos una consola de comandos o *shell*. Inmediatamente nos encontramos con el *login prompt*:

```
Debian GNU/Linux (etch/x86_64) hostname tty2
Hostname login:
```

Luego de ingresar el usuario y su password, la consola está lista para aceptar *comandos* por parte del usuario:

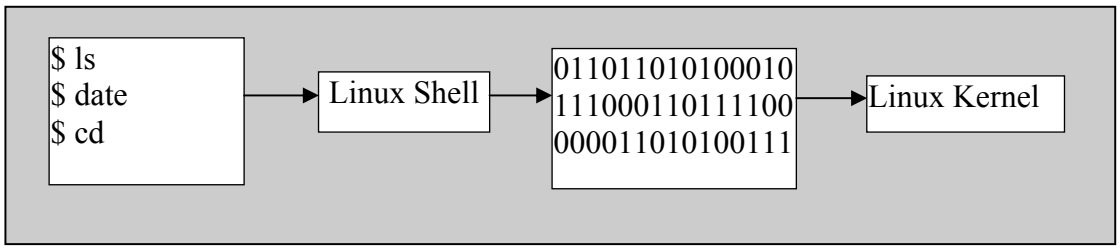
```
Debian GNU/Linux (etch/x86_64) hostname tty2
Hostname login: user
Password: *********

user@hostname:~$__
```

En cada línea de la consola a la espera de que ingresemos algún comando tenemos siempre la referencia de donde estamos trabajando: el usuario que inició la sesión seguido del nombre de la computadora y luego del ":" el directorio donde estamos

actualmente (~ en este caso). El símbolo $ es el *prompt* indica que la consola está lista para aceptar comandos.

LA SHELL Y LOS COMANDOS

Una computadora entiende el lenguaje de 0's y 1's, el lenguaje binario. En los sistemas UNIX existe un programa especial especial llamado *Shell*. La Shell acepta las instrucciones o comandos y los traduce al lenguaje binario nativo de las computadoras.



Existen varios tipos de shell. Algunas tienen sus ventajas y desventajas. Entre las más conocidas están: *tcsh., ksh, bash*. La más potente de todas es la bash (Bourne Again Shell) y es la que vamos a utilizar en este curso.

En Linux un *comando* es un programa que uno puede ejecutar. Para ejecutar un comando en la shell, se escribe su nombre, sus *argumentos* (de tenerlos) y se presiona Enter. Los comandos también pueden aceptar varias *opciones*. El resultado del comando del comando diferirá de acuerdo a las opciones utilizadas. La forma general de un comando es :

comando {opciones} {argumentos}

A continuación mostramos algunos comandos comunes en Linux. El comando **date** es un comando simple:

```
user@hostname:~$ date

 Wed Mar 26 17:15:50 UYT 2008
```

También acepta *opciones* :

```
user@hostname:~$ date -u
Thu Mar 27 01:11:09 UTC 2008
```

Un ejemplo de un comandos con argumentos y opciones es **cd** (Change Directory) y **ls** (list) , el argumento para **cd** es el camino al directorio donde queremos posicionarnos. El comando **ls** puede llevar la opción –l que permite listar los archivos del directorio actual con más información:

```
cesarv@ravnos:~$ cd /
cesarv@ravnos:/$ ls
bin   dev  etc   initrd  lost+found  mnt  proc  sbin  tmp  var
boot  dvd  home  lib     media       opt  root  sys   usr
```

ESTRUCTURA DE DIRECTORIOS

Aunque haya diferentes distribuciones y cada una traiga sus programas, la estructura básica de directorios y archivos es más o menos la misma en todas:

```
/-|--> bin
  |--> boot
  |--> cdrom
  |--> dev
  |--> etc
  |--> floppy
  |--> home
  |--> lib
  |--> mnt
  |--> proc
  |--> root
  |--> sbin
  |--> tmp
  |--> usr -|--> X11
  |         |--> bin
  |         |--> include
  |         |--> lib
  |         |--> local -|--> bin
  |         |           |--> lib
  |         |--> man
  |         |--> src --> linux
  |         |--> doc
  |--> var
```

El árbol que observamos muestra un típico árbol de directorios en Linux. Pueden variar, sin embargo, algunos de los nombres dependiendo de la distribución o versión de Linux que se esté usando. Algunos directorios destacados son:

/home - Espacio reservado para las cuentas de los usuarios.

/bin, **/usr/bin** - Binarios (ejecutables) básicos de unix.

/etc, aquí se encuentran los archivos de configuración de todo el software de la máquina.

/proc, es un sistema de archivo virtual. Contiene archivos que residen en memoria pero no en el disco duro. Hace referencia a los programas que se están corriendo en el momento en el sistema.

/dev (device) (dispositivo). Aquí se guardan los controladores de dispositivos. Se usan para acceder a los dispositivos físicos del sistema y recursos como discos duros, modems, memoria, mouse, etc. Algunos dispositivos:
hd: hda1 será el disco duro IDE, primario (a), y la primera partición (1).
fd: Así también, los archivos que empiecen con las letras fd se referirán a los controladores de las disketteras: fd0 será la primera diskettera, fd1 la segunda y así sucesivamente.
sd: son los dispositivos SCSI. Su uso es muy similar al del hd.
lp: son los puertos paralelos. lp0 es el puerto conocido como LPT1.
null: este es usado como un agujero negro, ya que todo lo que se dirige allí desaparece.
*tty***:** hacen referencia a cada una de las consolas virtuales. Como es de suponerse, tty1será la primera consola virtual, tty2 la segunda, etc.

/usr/local - Zona con las aplicaciones no comunes a todos los sistemas unix, pero no por ello menos utilizadas. En /usr/doc se puede encontrar información relacionada con dicha aplicación (en forma de páginas de manual, texto, html o bien archivos dvi, Postscript o pdf). También encontramos archivos de ejemplo, tutoriales, HOWTO, etc.

2.2 COMANDOS DE LINUX

2.1 TRABAJANDO CON ARCHIVOS Y DIRECTORIOS

La estructura de directorios en UNIX es jerárquica o arborescente, debido a que todos los directorios nacen en un mismo punto (denominado directorio raíz). Un archivo se encuentra situado siempre en un directorio y su acceso se realiza empleando el camino que conduce a el en el árbol de directorios del sistema. Este camino es conocido como el *path*.

El acceso a un archivo se puede realizar empleando:

- Path Absoluto, aquel que empieza con /
  Por ejemplo : /etc/printcap

- Path Relativo, aquel que no empieza con /
  Por ejemplo : ../estudiantes/archivo1

El signo * puede sustituir cualquier conjunto de caracteres y el signo ? cualquier caracter individual. Por ejemplo:

```
user@hostname:~$ ls /bin/*ou??
/bin/mount  /bin/touch  /bin/umount
```

Los archivos cuyos nombre que comiencen por "." se denominan ocultos, así por ejemplo en el directorio de partida de un usuario:

```
user@hostname:~$ ls -a /root/
.           .fonts.cache-1      .mc
..          .gconf              .mcop
```

Exiten algunos directories especiales: el “.” (directorio actual), el “..” (directorio padre) y el “~” (directorio home del usuario).

Ahora vamos a ver algunos comandos para trabajar con directorios y archivos.

NAVEGANDO, CREANDO, VISUALIZANDO DIRECTORIOS Y ARCHIVOS

A continuación se dan algunos comandos para navegar y alterar directorios.

ls (LiSt)

Este comando permite listar los archivos de un determinado directorio. Si no se le suministra argumento, lista los archivos y directorios en el directorio actual. Si se añade el nombre de un directorio el listado es del directorio suministrado. Existen varias opciones que modican su funcionamiento entre las que destacan:

-l (Long listing) proporciona un listado extenso, que consta de los permisos de cada archivo, el usuario el tamaño del archivo, etc.

-a (list All) lista también los archivos ocultos.

-R (Recursive) lista recursivamente el contenido de todos los directorios que se encuentre.

-t ordena los archivos por tiempo de modificación.

-S ordena los archivos por tamaño.

-r invierte el sentido de un orden.

pwd (Print Working Directory)

Este comando proporciona el nombre del directorio actual.

cd (Change Directory

Permite moverse a través de la estructura de directorios. Si no se le proporciona argumento se provoca un salto al directorio $home. El argumento puede ser un nombre absoluto o relativo de un directorio.

mkdir (MaKe DIRectory)

Crea un directorio con el nombre (absoluto o relativo) proporcionado.

rmdir (ReMove DIRectory)

Elimina un directorio con el nombre (absoluto o relativo) suministrado. Dicho directorio debe de estar vacío.

Algunos comandos para alterar archivos son:

(CoPy) cp

Copia un archivo(s) con otro nombre y/o a otro directorio. Veamos algunas opciones:

-i (interactive), impide que la copia provoque una pérdida del archivo destino si éste existe.
-R (recursive), copia un directorio y toda la estructura que cuelga de él.

mv (MoVe)

Mover un archivo(s) a otro nombre y/o a otro directorio. Dispone de opciones análogas al caso anterior.

rm (ReMove)

Borrar un archivo(s). En caso de que el argumento sea un directorio y se haya sumnistrado la opción -r, es posible borrar el directorio y todo su contenido. La opción -i pregunta antes de borrar.

LOCALIZANDO ARCHIVOS

Podemos ubicar ciertos archivos o palabras de interés en el sistema con:

whereis

Recibe como argumento un nombre de archivo y localiza binarios, fuentes y archivos de man pages.

apropos

Recibe como argumento una palabra clave y la busca entre las páginas del manual y las descripciones.

locate

Lista os archivos en una base de datos del sistema que coinciden con un patrón de argumento.

find

Permite la búsqueda de un archivo en la estructura de directorios.
find dir -name file –print
Comenzando por el directorio dir busca en forma recursiva el archivo de nombre file y luego imprime el path si lo encuentra

VISUALIZACION DE ARCHIVOS

cat

Muestra por pantalla el contenido de un archivo que se suministra como argumento.

less

Permite moverse en ambas direcciones. Otra ventaja es que no lee el archivo entero antes de arrancar.

AGRUPANDO Y COMPACTANDO ARCHIVOS

Hay una serie de comandos que permiten agrupar y compactar un conjunto de archivos en un único archivo. También posibilitan la operación inversa: descomprimir un archivo compuesto de varios en sus componentes.

tar

Este comando permite la creación/extracción de archivos contenidos en un único archivo denominado tarfile (o tarball). Este tarfile suele ser luego comprimido con *gzip* la versión de compresión gnu o bien con *bzip2*.
La acción a realizar viene controlada por el primer argumento:

- c (Create) creación
- x (eXtract) extracción
- t (lisT) mostrar contenido
- r añadir al final
- u (Update) añadir aquellos archivos que no se hallen en el tarfile o que hayan sido modificados con posterioridad a la versión que aparece.

Algunas opciones para tar son:

- v (Verbose) : muestra en la consola el proceso de la operación.
- z : comprimir o descomprimir (según que la primera opción sea c o x) el contenido con el comando *gzip*.
- I : comprimir o descomprimir (según que la primera opción sea c o x) el contenido con el comando *bzip2*.
- f file: permite especificar el nombre del archivo para el tarfile.

FILTROS

Existe un conjunto de órdenes en unix que permiten el procesamiento de archivos de texto. Se denominan filtros (Unix Filters) porque recibien datos por su stdin y retornándolos modificados por su stdout. Se suelen utilizar junto con la redirección. Para facilitar la comprensión de los ejemplos siguientes supondremos que existen dos archivos llamado file1.txt file2.txt, que tienen en su interior:

File1.txt	File2.txt
primero , hola	primero ,hola
segundo , que-tal	segundo , como-estas
tercero , chau	tercero , chau

cat

El filtro más básico, copia la entrada a la salida.

```
user@hostname:~$ cat file1.txt
primero  hola
segundo  quetal
tercero  chau
```

cut

Para un archivo compuesto por columnas de datos, permite escribir sobre la salida cierto intervalo de columnas; o más general aún: por cada fila del archivo se puede seleccionar cierto intervalo de caracteres. La opción "-b N-M" permite indicar el intervalo en bytes que se escribirán en la salida.

```
cut -d , -f 1 file1.txt
primero
segundo
tercero
```

-d es la opción de delimitador (en este caso la ,) que se usa para separar las cadenas de caracteres en cada fila. La opción f 1 especifica que se devuelva el primer "campo" (field) separado por el delimitador.

Para separar los carateres del 2 al 5 de cada fila:

```
user@hostname:~$ cut -b 2-5 file1.txt
rime
egun
erce
```

diff

Permite comparar el contenido de dos archivos.

```
user@hostname:~$ diff file1.txt file2.txt
2c2
< segundo,quetal
---
> segundo,universo
```

head

Muestra las primeras diez líneas de un archivo.

```
user@hostname:~$ head -2 file1.txt
primero,hola
segundo,quetal
```

tail

Muestra las últimas filas de un archivo.

```
user@hostname:~$ tail -2 file1.txt
segundo,quetal
tercero,chau
```

grep (Global Regular Expresión Print)

Permite la búsqueda de una cadena de caracteres (*expresiones regulares*) en uno o varios archivos, imprimiendo el nombre del archivo y la línea en que encuentra la cadena.

```
cvoulgaris@introcomp:~$ grep  primero file1.txt
primero,hola
cvoulgaris@introcomp:~$ grep ho file1.txt
primero,hola
cvoulgaris@introcomp:~$ grep segundo *.txt
file1.txt:segundo,quetal
file2.txt:segundo,universo
```

Algunas opciones útiles:

- c : Elimina la salida normal y sólo cuenta el número de apariciones de la cadena en cada archivo.
- i : Ignora en la comparación de la cadena y el archivo, las mayúsculas y minúsculas.
- r : La búsqueda la hace recursiva.
- v: Invierte la búsqueda mostrando todas las líneas donde no aparece la cadena pedida.

wc (Word Count)

Contabiliza el número de líneas, palabras y caracteres de un archivo.

```
user@hostname:~$ wc file1.txt
3    3   41 file1.txt
```

awk

Es un procesador de archivos de texto que permite la manipulación de las líneas de forma tal que tome decisiones en función del contenido de la misma. Es un comando complejo y muy poderoso, de hecho es un lenguaje de programación de por si. Ejemplo, supongamos que tenemos nuestro archivo file1.txt con sus dos columnas separadas por la , :

```
user@hostname:~$  awk  -F  ,  '{print  $2  "  separo  "  $1}'
file1.txt
hola separo primero
quetal separo segundo
chau separo tercero
```

COMANDOS DE ACCESO REMOTO A REDES

Desde el comienzo de su desarrollo, los sistemas UNIX fueron diseñados para interactuar en forma remota. Hoy en día existen varios protocolos que posibilitan eso. Introducimos entonces algunos conceptos.

Para identificarse en una red, una computadora lleva asignada una IP (Internet Protocol), esto es un número del tipo 192.168.1.5. Estos cuatros números separados por un "." identifican unívocamente a una computadora en una red cualquiera, ya sea una LAN o Internet.

Pero es difícil intentar de comunicarse con una computadora recordando su IP y es por esto que existe el concepto de *domino*:

Un nombre de dominio es aquel que identifica a una computadora o grupo de computadoras en una red.

Para identificar a una computadora en Internet tenemos dominios como:

www.google.com
www.fisica.edu.uy

Cuando queremos acceder a una computadora conociendo su nombre de dominio y que requiera la autenticación por parte del usuario, es estándar en varios protocolos la notación *user@domain*. Vemos esto al logearnos a un sistema UNIX, en los correos electrónicos y otros protocolos de acceso por redes.

A su luego que identificamos a una computadora por su nombre y nos autenticamos como usuario, debemos especificar con que protocolo queremos "hablarle" (ftp,ssh,https) y la forma de comunicarnos es:

protocolo://usuario@dominio/recurso

El protocolo puede ser ssh, ftp, etc., el recurso es alguna carpeta u archivo a la que accedemos.

Los comandos de conexión remota que vamos a ver son *ftp* y *ssh*.

ftp (File Transfer Protocol)

Aplicación para copiar archivos entre máquinas de una red. ftp exige un nombre de cuenta y password para la máquina remota. Algunas de las opciones más empleadas (una vez establecida la conexión) son:

- cd : cambia directorio en la máquina remota.
- lcd : Cambia directorio en la máquina local.
- ls : Lista el directorio remoto.
- !ls : Lista el directorio local.
- get rfile [lfile] : transfiere el archivo rfile de la máquina remota a la máquina local llamandolo lfile. En caso de no suministrarse el segundo argumento supone igual nombre en ambas máquinas.
- put lfile [rfile] : transfiere el archivo lfile de la máquina local a la máquina remota, denominándola rfile. En caso de no suministrarse el segundo argumento ,se supone igual nombre en ambas máquinas.

Por ejemplo para conectarnos a sitio de la nasa de dominio ftp.hq.nasa.gov:

```
user@hostname:~$ ftp ftp.hq.nasa.gov
Connected to ftp.hq.nasa.gov.
220 FTP Server Ready
Name (ftp.hq.nasa.gov:user): anonymous
331 Anonymous login ok, send your complete email address as
your password.
Password:
230-Warning: This system is owned and operated by the US
Federal Government.
          Unauthorized access to this system is a violation
of US Federal
          law and could lead to prosecution.

 This is NASA HQ ANONYMOUS FTP SERVER.
```

Un ftp de usuario: *anonymous* , es aquel que permite un acceso a cualquier usuario. La clave debe ser algún correo electrónico o nula (es indistinto cual, el acceso es libre).

ssh (Secure Shell)

ssh permite acceder a una shell en una máquina remota. El usuario luego de logearse, trabaja en dicha máquina al igual que un usuario local. Ssh es "seguro", esto es la información de intercambio entre la máquina local y la remota, está encriptada.

```
user@hostname:~$ ssh remoteuser@remotehost
Password:
remoteuser@remotehostname:~$
```

scp (Secure CoPy)

scp permite realizar copias de archivos entre dos máquinas distintas. Es una combinación de cp (copy) y ssh. La sintaxis es scp origen destino:

```
user@hostname:~$ scp localfile remoteuser@remotehost:/path
Password:
lx-optra en horus (Horus).lnk            100%  393     0.4KB/s   00:00
```

ping

Es un comando simple. Permite saber si existe conexión a una máquina remota. No todas las máquinas aceptan que se les envíen un ping.

```
cesarv@ravnos:~$ ping www.fisica.edu.uy
PING www.fisica.edu.uy (164.73.83.126) 56(84) bytes of data.
64 bytes from horus.fisica.edu.uy (164.73.83.126): icmp_seq=1 ttl=64 time=0.103 ms
64 bytes from horus.fisica.edu.uy (164.73.83.126): icmp_seq=2 ttl=64 time=0.145 ms

--- www.fisica.edu.uy ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1000ms
rtt min/avg/max/mdev = 0.103/0.124/0.145/0.021 ms
```

wget (Web GET)

Este comando permite bajar de la web, archivos en forma no interactiva. Soporta los protocolos http, https, ftp.

```
user@hostname:~$ wget http://www.fisica.edu.uy
--14:14:03--  http://www.fisica.edu.uy/
           => `index.html'
Resolviendo www.fisica.edu.uy... 164.73.83.126
Conectando con www.fisica.edu.uy[164.73.83.126]:80... conectado.
Petición HTTP enviada, esperando respuesta... 200 OK
Longitud: 20,301 [text/html]

100%[====================================>] 20,301        --.--K/s

14:14:03 (20.23 MB/s) - `index.html' guardado [20301/20301]
```

COMANDOS DEL SISTEMA

Protección de archivos en Linux

Dado que el sistema de archivos UNIX es compartido por un conjunto de usuarios, surge el problema de la necesidad de privacidad. Sin embargo, dado que existen conjuntos de personas que trabajan en común, es necesaria la posibilidad de que un conjunto de usuarios puedan tener acceso a una serie de archivos (que puede estar limitado para el resto de los usuarios).

Cada archivo y directorio del sistema dispone de un propietario, un grupo al que pertenece y unos permisos.

Existen tres tipos fundamentales de permisos:

lectura (r-Read): en el caso de un archivo significa poder examinar el contenido del mismo; en el caso de un directorio significa poder entrar en dicho directorio.

escritura (w-Write): en el caso de un archivo significa poder modificar su contenido; en el caso de un directorio es crear un archivo o directorio en su interior.

ejecución (x-eXecute): en el caso de un archivo significa que ese archivo se pueda ejecutar (binario o archivo de procedimientos); en el caso de un directorio es poder ejecutar alguna orden dentro de él.

Se distinguen tres grupos de personas sobre las que especificar permisos:

user: el usuario propietario del archivo.

group: el grupo propietario del archivo (excepto el usuario). Cada usuario puede pertenecer a uno o varios grupos y el archivo generado pertenece a uno de los mismos.

other: el resto de los usuarios (excepto el usuario y los usuarios que pertenezcan al grupo)

Para visualizar los permisos, ejecutamos el comando ls -l:

```
user@hostname:~$ ls -l
total 4
-rw-r--r-- 1 user user 41 Apr  3 03:29 file1.txt
```

Los diez primeros caracteres muestran la protección del archivo.

El primer carácter, indica que tipo de archivo es:

- - : archivo
- d : directorio
- l : enlace
- b : dispositivo de bloque

Los siguientes 3 caracteres (2,3,4) son los permisos del usuario. Los caracteres 5,6,7 son los permisos del grupo y los 3 últimos (8,9,10) los de otros usuarios.

Estas ternas de caracteres pueden tomar los valores de permisos vistos: **r** para el de la izquierda, **w** para el del medio y **x** para el de la derecha.

Los comandos más relevantes para nuestro interés son

Chmod (Change MODe)

El comando permite cambiar los permisos rwx para el usuario propietario, el grupo y los demás usuarios. Algunos ejemplos son:

```
user@hostname:~$ ls -l
total 0
-rw-r--r-- 1 user user 0 Apr  4 04:40 file.txt
user@hostname:~$ chmod u+x,g+wx,o+x file.txt
user@hostname:~$ ls -l
total 0
-rwxrwxr-x 1 user user 0 Apr  4 04:40 file.txt
```

Cuando con chmod eliminamos el permiso de lectura a todo un directorio ya sea para *group* o *other*, denegamos la posibilidad de lectura para los miembros de grupo o los demás usuarios. La opción –R en este caso permite aplicar el cambio hecho a los permisos del directorio en forma recursiva a todos sus subdirectorios y archivos.

Otros comandos útiles son:

id

Muestra la identificación del usuario, así como el conjunto de grupos a los que el usuario pertenece.

```
user@hostname:~$ id
uid=1000(user) gid=1000(user) groups=1000(user)
```

Visualización y control de procesos

En una máquina existen una multitud de procesos que pueden estar ejecutándose simultáneamente.

ps (ProceSs)

Reporta el estado actual de los procesos en memoria
Las opciones más comunes son:
x : Lista todos los procesos que pertenezcan al usuario, incluyendo los que no están asociados a una terminal.
u : Lista los usuario propietarios de los procesos
a : Lista procesos de todos los usuarios.

```
user@introcomp:~$ ps aux
USER     PID    %CPU  %MEM  VSZ   RSS TTY      STAT START   TIME COMMAND
root     14219   0.0  0.0   5844  664 ?        Ss   02:06   0:00 /sbin/syslogd
root     14292   0.0  0.0   5092  724 ?        Ss   02:07   0:00 /usr/sbin/dhcdb
user     25961  0.0  0.1  50024  1776 ?        S    05:10   0:00 sshd: user@pts/
user     25962  0.0  0.3  18924  3108 pts/1    Ss   05:10   0:00 -bash
user     27321  0.0  0.0  13624   952 pts/1    R+   05:32   0:00 ps aux
```

kill

Envía una señal a un proceso en ejecución. La señal más KILL, termina el proceso (es la señal por defecto y no hace falta especificarla).

Otros comandos del sistema

date

Muestra el día y la hora actual.

du (Disk Usage)

Permite ver el espacio de disco ocupado por el archivo o directorio suministrado como argumento. La opción -s impide que cuando se aplique recursividad en un directorio se muestren los subtotales. La opción -h imprime los tamaños en un formato fácil de leer.

df (Disk Free)

Muestra los sistemas de archivos que están montados en el sistema, con las cantidades totales, usadas y disponibles para cada uno.

echo

Imprime en la salida estándar una cadena de caracteres.

```
user@introcomp:~$ echo Hola que tal
Hola que tal
```

Alias

Permite definir un alias para uno o varios comandos.

```
user@introcomp:~$ alias lh='ls -lh'
user@introcomp:~$ lh
total 4.0K
-rwxrwxr-x 1 user user 11 Apr  4 06:05 file.txt
user@introcomp:~$ alias ..='cd ..'
user@introcomp:~$ ..
user@introcomp:/home$ pwd
/home
```

REDIRECCION Y ENCADENAMIENTO DE COMANDOS

Cuando un programa espera que se teclee algo, aquello que el usuario teclea se conoce como el *Standard Input: stdin*. Los caracteres que el programa retorna por pantalla es lo que se conoce como *Standard Output: stdout* (o *Standard Error: stderr*).

Cuando un comando se ejecuta con error, el mensaje que se retorna por pantalla es uno de error, o sea la salida estándar de error (stderr).

La salida o la entrada de un comando hacia/desde la consola, se puede *redireccionar* a archivos u otros comandos.

Las formas de redirección son las siguientes:

- ***comando {args} > file*** : La salida del comando en vez de devolverse por la stdout, se redirecciona al archivo file.
- ***comando < file*** : La entrada al comando cuyo argumento es un archivo, es leída del archivo file
- ***comando1 {args} | comando2*** : La salida del comando1 es la entrada del comando2. Esta redirección se llama *tubería (pipe)*.

También existen variantes a las anteriores:

- ***comando {args} >> file*** : La salida del comando, se agrega al final del archivo file si este existe, sino se crea.

- ***comando {args} >& file***: Redireccionar el stdout y el stderr.

Algunos ejemplos:

```
user@hostname:~$ date > date_file
user@hostname:~$ cat date_file
Fri Apr 11 09:08:48 UTC 2008
```

```
user@hostname:~$ cat > letras
a
g
z
e
d
j
v
user@hostname:~$ sort < letras
a
d
e
g
j
v
z
```

```
user@hostname:~$ cat > dirs
/bin /home /usr
user@hostname:~$ cat dirs| cut -f 1 -d " "
/bin
user@hostname:~$ cat dirs| cut -f 2,3 -d " "
/home /usr
```

En Linux también podemos encadenar comandos con el separador ";". Los comandos se ejecutan uno tras otro como si se ejecutasen de a uno seguido por ENTER.

Ejemplos:

```
cvoulgaris@introcomp:~$ cd /home; ls ; cd ; pwd ; echo OPA\!\!
abergengruen  cvoulgaris  iavellanal  jventurini  mmartinez  sbecco
amaciel       dcarrasco   jchadikov   lcolombo    nibanez    user
aschmidt      ftp         jrama       mfrederico  pnunez     vfeldman
/home/cvoulgaris
OPA!!
```

Otro ejemplo más complicado:

```
cvoulgaris@introcomp:~$ echo LEO EL ARCHIVO DE USUARIOS Y
SEPARO A LA DESCRIPCION DE USER ; less /etc/passwd | grep
cvoulgaris | cut -f 5 -d :
LEO EL ARCHIVO DE USUARIOS Y SEPARO A LA DESCRIPCION DE
USER
Cesar Voulgaris,,,,Docente del curso intro-comp
```

2-3 PROGRAMANDO EN LA SHELL

Un programa o script de Linux es un archivo de tipo ejecutable constituido por una secuencia de comandos de Linux.

VARIABLES

A veces para procesar nuestros datos, debemos guardarlos en la memoria RAM. La RAM esta indexada o dividida en muchas localizaciones, cada una de ellas tiene lleva asociada un número llamado *dirección de memoria* en donde están nuestros datos.

Los programadores pueden dar nombres a estas localizaciones de memoria llamados *variables*.

En Linux hay dos tipos de variables:

Variables del sistema: son creadas y mantenidas por Linux. Este tipo de variables está definida en mayúsculas.

Variables definidas por el usuario: Creadas y mantenidas por el usuario.

Al utilizar el caracter $ antes del nombre de una variable, se *sustituye* (salvo que esté dentro de una cadena delimitada por las comillas verticales ") el contenido de la variable. De esta forma podemos evaluar la variable. Algunas variables importantes del sistema:

```
user@hostname:~$ echo $BASH
/bin/bash
user@hostname:~$ echo $USER
cvoulgaris
user@hostname:~$ echo $HOSTNAME
introcomp
user@hostname:~$ echo $PATH
/usr/local/bin:/usr/bin:/bin:/usr/games
user@hostname:~$ echo $HOME
```

Las variables del usuario se definen por: *nombre=valor* (sin espacios!). Para evaluarlas: *$nombre*.

Ejemplos:

```
user@hostname:~$ curso="introduccion a la omputacion"
user@hostname:~$ echo $curso
introduccion a la computacion
user@hostname:~$ curso=introduccion-a-la-computacion
user@hostname:~$ echo $curso
introduccion-a-la-computacion
```

Se pueden definir variables de tipo *array*. Los arrays proveen un método de agrupar un conjunto de valores en un único nombre de variable: *nombre[indice]=valor*.

```
user@hostname:~$ topico[1]="linux"
user@hostname:~$ topico[2]="C"
user@hostname:~$ topico[3]="Fortran"
user@hostname:~$ echo ${topico[1]}
linux
user@hostname:~$ echo ${topico[*]}
linux C Fortran
```

El caracter $ también se puede usar para *sustitución de una expresión aritmética*:

```
user@hostname:/home/user$ echo $((5+6))
11
```

EL USO DE COMILLAS EN LINUX

Existen varias formas de interpretar el contenido de adentro de algunos de estos 3 tipos de comillas:

`lista de comandos` : Todos los comandos dentro de estas comillas se sustituyen por su resultado final.

'contenido' : el contenido dentro de las comillas verticales, se reproducen literalmente, no se sustituyen variables, ni comandos.

“contenido”: el contenido se reproduce pero se sustituyen las variables y los comandos (siempre que se anteceda a la variable el símbolo $ y los comandos estén dentro de ``).

Ejemplos:

```
cvoulgaris@introcomp:~$ var="variable de caracteres"
cvoulgaris@introcomp:~$ echo $var
variable de caracteres
cvoulgaris@introcomp:~$ var1='Imprimo la variable $var y ejecuto el comando `pwd`'
cvoulgaris@introcomp:~$ echo $var1
Imprimo la variable $var y ejecuto el comando `pwd`
cvoulgaris@introcomp:~$ var2="Imprimo la variable $var y ejecuto el comando `pwd`"
cvoulgaris@introcomp:~$ echo $var2
Imprimo la variable variable de caracteres y ejecuto el comando /home/cvoulgaris
```

PRIMEROS EJEMPLOS DE SHELL SCRIPTS

Un script es un archivo de texto que comienza siempre con la línea: **#!/bin/bash** y es seguido por uno o más comandos de linux y comentarios que no se ejecutan. Escribir y ejecutar y analizar los siguientes scripts (siempre deben cambiar el permiso del archivo script a "ejecución" para correrlo):

```
#!/bin/bash

# Esto es un comentario: mi primer script de linux!

echo "hola mundo!";
echo "soy $USER";
echo "Hoy es: `date`"
```

```
#!/bin/bash

# Se puede utilizar el comando expr para evaluar expresiones aritméticas
echo 'El resulatdo de `expr 6.3` es:'
resultado=`expr 6 + 3`
echo $resultado

# Lo anterior es equivalente a utilizar "sustitución de expresiones"
echo 'El resultado de $((expresion)), siendo expresión una relación aritmética es:'
resultado=$((6+3))
echo $resultado
```

ARGUMENTOS DE UN SCRIPT

Un script no es otra cosa que un comando de Linux definido por el usuario en base a comandos ya existentes. Por lo tanto puede aceptar argumentos de entrada.

Linux almacena en variables reservadas los argumentos que se le ingresan a un script:

$0 : referencia al nombre del script mismo.
$1 : referencia al primer argumento.
$2….$N : son los contenidos de los argumentos 2…N respectivamente.
$* : Son todos argumentos juntos.
$# : contiene la cantidad de argumentos que se le pasan al script.

Estas variables son necesarias para acceder a los argumentos dentro del script.

```
#!/bin/bash

#Este script muestra el uso de argumentos en la línea de comandos
#Correr el script con dos argumentos argumentos cualquiera

echo "La cantidad de argumentos de este script es: $#"
echo "El nombre del script es: $0"
echo "El primer argumento es: $1"
echo "El segundo argumento es: $2"
echo "Todos los argumentos son: $*"
```

CONTROL DE FLUJO

El comando if.
El comando **if** ejecuta acciones dependiendo de si una condición es verdadera (El resultado que devuelve a la shell es 0) o falsa (devuelve un número mayor que cero).

La sintaxis es:

if *list1*
then
list2
else
list3
fi

Las listas pueden ser muy generales (sucesión de comandos, etc.), pero en general la condición que se le pasa al comando if es de la forma: **if [expresión]**. Se debe dejar espacios entre la expresión y los corchetes rectos.
Las expresiones son de tres tipos:

- Test de Archivos:
  La expresión es de la forma **[opción archivo/directorio]**
  Las opciones más comunes son:

  If [-d directorio] : devuelve verdadero si existe el directorio.
  If [-f archivo] : " " si existe el archivo y es regular.
  If [-x archivo] : " " si el archivo es ejecutable.
  If [-w archivo] : " " si el archivo es escribible.
  If [-r archivo] : " " si el archivo es de lectura.

  Ejemplo:

```
#!/bin/bash
if [ -f $HOME/archivo ]
then
    echo "$HOME/archive existe"
fi
```

- Comparación de string:
  La expresión es de la forma: **[opcion string] , [string1 = = string2] , [string1 != = string2] :**

  opción –z : devuelve verdadero si el string tiene tamaño nulo.
  opción –n : " " si el string tiene tamaño no nulo.
  s1 == s2 : " " si el string s1 es idéntico a s2.
  s1!== s2 : " " si el string s1 es distinto a s2.

```
#!/bin/bash
if [ "hola" == "chau " ]
then
   echo "son iguales"
else
   echo "son distintos"
fi
```

- Comparaciones numéricas.
  La expresión es de la forma: **[int1 operador int2]** con i1 e i2 enteros.

  Los operadores pueden ser:

  i1 -eq i2 : Verdadero si i1 e i2 son iguales.
  i1 -ne i2 : Verdadero si i1 e i2 son distintos.
  i1 -lt i2 : Verdadero si i1 es menor que i2.
  i1 -le i2 : Verdadero si i1 es menor o igual que i2.
  i1 -gt i2 : Verdadero si i1 es mayor a i2.
  i1 -ge i2 : Verdadero si i1 es mayor o igual a i2.

Para ver ejemplos de como utilizar el comando if, abrir y ejecutar el script condicionales.sh en el home de "user".

El comando for.
El comando **for** implementa un *bucle (loop* en inglés*)* o repetición de una serie de comandos para cada uno de los elementos de una lista que se específica.

La sintaxis de la lista puede darse de varias formas:

for var in *cadena1 cadena2 ... cadenaN*
do
  comandos
done

o para expresiones aritméticas:

for *((var=int1 ; var <int2 ; var=var+incr))*
do
  commandos
done

```
#!/bin/bash
for i in “primero” “segundo” “tercero”
do
    echo $i
done
```

El resultado de este commando (dentro del script es):

```
primero
segundo
tercero
```

Otro ejemplo para especificar una lista numérica con expresiones aritméticas:

```
#!/bin/bash
for ((i=1;i<=4;i=i+2))
do
    echo $i
done
```

El resultado es:

```
1
2
4
```

Si se omite la expresión aritmética para el incremento (la tercera), entonces *incr* es 1.

El comando while.
Este comando al igual que for implementa un bucle, pero la repetición está determinada por una condición explícita que se evalúa con cierta o falsa y no por un rango de una lista.

while comando
do
  comandos
done

El comando en general se da por una expresión como en el comando if:

```
x=0
while [ $x -lt 4 ]
do
   x=$(($x+1))
   echo $x
done
```

El resultado es:

```
0
1
2
3
```