INTRODUCCIÓN AL SISTEMA OPERATIVO UNIX

CAPITULO 1. INTRODUCCION

1.1 Antecedentes históricos

El S.O. Unix fue creado a finales de la década de los 60 sobre la base de varios trabajos realizados conjuntamente por el MIT y Laboratorios BELL. Dichos trabajos (proyecto MULTICS) iban encaminados a la creación de un macrosistema de computación que diese servicio a miles de usuarios. Si bien el proyecto fracasó, posiblemente por intentar abarcar demasiado contando con unos elementos hardware limitados en esa época, influyó decisivamente sobre la evolución de los sistemas informáticos posteriores.

Un antiguo componente de dicho proyecto (Ken Thompson) desarrolló por su cuenta un sistema operativo monousuario con la característica principal de un sistema de archivos jerárquico.

El sistema encontró muchos entusiastas y se hizo portable al reescribirse casi íntegramente en lenguaje "C", y se suministró en código fuente a las universidades como objeto de formación. Así, la universidad de California en Berkeley retocó dicho sistema (fundamentalmente, comunicaciones y diversas utilidades como el editor "vi"), y liberó lo que luego sería el BSD, uno de los dos "dialectos" principales del UNIX.

Actualmente, existen dos corrientes las cuales cada vez poseen más elementos comunes: el BSD 4.2 y el System V R4.

1.2 Generalidades

El S.O. Unix se encarga de controlar y asignar los recursos físicos del ordenador (hardware) y de planificar tareas. Podemos establecer tres elementos principales dentro de este S.O.

- El núcleo del sistema operativo (kernel), el escalón más bajo que realiza tareas tales como el acceso a los dispositivos (terminales, discos, cintas, ...).
- El intérprete de comandos (shell) es el interfase básico que ofrece UNIX de cara al usuario. Además de ejecutar otros programas, posee un lenguaje propio así como numerosas características adicionales que se estudiarán en un capítulo posterior.
- Utilidades "de fabrica"; normalmente se trata de programas ejecutables que vienen junto con el Sistema Operativo, algunas de ellas son:
  - Compiladores: C, assembler y en algunos casos Fortran 77 y C++.
  - Herramientas de edición: Editores (vi, ex), formateadores (troff), filtros, ...
  - Soporte de comunicaciones: Herramientas basadas en TCP/IP (telnet, ftp, ...).
  - Programas de administración del sistema (sysadm , sa , va, ...).
  - Utilidades diversas.

CAPITULO 2. ORDENES BASICAS

2.1 Conexión y desconexión

Para acceder al sistema, este presenta el mensaje de login, con el que quiere significar algo así como "introduce el usuario con el que quieres abrir una sesión".

```
UNIX(r) System V Release 4.2
login:
```

Una vez tecleado el usuario que se quiere y haber pulsado *RETURN*, solicita una palabra de paso (password), la cual, como es natural, no se verá en pantalla aunque se escriba.

```
UNIX(r) System V Release 4.2
login:antonio
Password:
$
```

Tanto el nombre del usuario como la palabra de paso han de ser escritas "de golpe", es decir, no se pueden dar a los cursores para modificar ningún carácter y mucho menos la tecla de *Backspace*, *Ins*, *Del*, ... Esto es debido a que, tanto este carácter como los aplicados a los cursores son caracteres válidos en nombres de usuario y en palabras de paso.

El sistema, una vez aceptado el nombre del usuario (el cual como es obvio habrá sido asignado por el Administrador, así como la palabra de), lanza por pantalla unos mensajes de bienvenida y termina con el símbolo denominado "prompt", símbolo configurable (como casi todo en UNIX) y que suele ser un '$' ó un '#'.

Existe en todos los sistemas UNIX un superusuario llamado "root", que puede hacer absolutamente lo que quiera en el sistema. Además, hay algunos usuarios especiales, dependiendo del sistema que se trate con más privilegios de los normales (admin ó sa ó sysadm , usuario de administración del equipo, uucp como usuario de comunicaciones) y el resto, que corresponden a usuarios normales.

El programa que está en este momento mostrando el prompt es la shell ó intérprete de comandos. Con este prompt indica algo así como "preparado para que me escribas el comando que quieres ejecutar".

Cada comando debe finalizar en un *RETURN*, el cual funcionalmente se asemeja a la orden "AR" en la mili (El sargento dice "firmes", pero nadie se mueve hasta que da el *RETURN*, es decir, "AR").

También es significativa la diferencia entre mayúsculas y minúsculas; no es lo mismo "cal" que "CAL". El primero es un comando de calendario; el segundo no existe y la shell se queja de que no lo encuentra:

```
$ cal
January 1995
 S  M Tu  W Th  F  S
 1  2  3  4  5  6  7
 8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30 31
$ CAL
UX:sh:ERROR:CAL: Not found (no existe ningún comando "CAL"!)
```

Para terminar la sesión, se tecleará el comando "exit", lo que provocará la aparición del mensaje de login:

```
$ exit
UNIX(r) System V Release 4.2
login:
```

efecto que puede también conseguirse apagando y encendiendo la terminal (en terminales "civilizadas") ó pulsando *Ctrl D* (Tecla de "*Ctrl*" y tecla "*D*" a la vez). Puede que en su terminal alguna de estas dos cosas no funcione: piense que en UNIX casi todo es configurable.

2.2 Conceptos : grupo,directorio de trabajo , shell , tipo de terminal

Cada usuario, aparte de la palabra de paso, posee unos determinados atributos los cuales básicamente son:

Grupo: A cada usuario, en el momento de su creación, se le asigna uno (ó varios, si la versión del sistema es moderno). La misión de esto radica en agrupar usuarios en "grupos" para permitir y denegar accesos en el sistema. Para ver a que grupo pertenecemos, se puede ejecutar el comando "id", el cual producirá un resultado similar al siguiente:

```
$ id
uid=129(jose) gid=1(other)
```

Como era de esperar, el comando nos contesta con siglas en inglés; "uid" (UserID), identificador de usuario, un número y un nombre (129 y jose); "gid" (GroupID), identificador de grupo, un número y un nombre (1 y other).

Directorio de trabajo: Al iniciar una sesión (tras hacer "login"), cada usuario "cae" en un determinado directorio prefijado. Este habrá sido fijado

por el administrador en el momento de la creación, y normalmente estará situado en "/home/jose", "/usr/people/jose" ó similar. No existe un estándar sobre ésto y puede ser cualquier subdirectorio del sistema, incluída la raíz. Para visualizar cual es nuestro directorio, podemos escribir el comando "pwd" (Print Working Directory) inmediatamente después de haber hecho "login".

Shell: Si bien nos hemos referido de forma genérica a la shell existen diversas versiones asignables a usuarios, de las cuales podemos citar algunas:

- Bourne shell ("sh"): Una de las más antiguas y, por tanto, más seguras.
- Korn shell ("ksh"): Basada en la de Bourne pero más moderna y con más funcionalidades.
- C-shell ("csh"): Preferida en entornos BSD y bastante potente aunque más criptica que las dos anteriores.

Veremos más adelante las características de cada una de ellas con más detalle.

Características del Terminal: Dependiendo de como se tenga configurada la terminal, la sesión se comporta de una manera determinada frente los siguientes caracteres especiales:

- *Ctrl-C* (INTR) Interrumpe el programa que se está ejecutando en ese momento.
- *Ctrl-H* (ERASE) Borra el último caracter escrito.
- *Ctrl-D* (EOF) Termina la sesión ; posee un efecto lar a escribir "exit".

2.3 Comandos (who/date/man/who am i/logname)

En UNIX, un comando se invoca escribiendo su nombre y, separados por blancos, los argumentos opcionales. Como antes, se debe pulsar *RETURN* una vez se ha escrito la orden correspondiente.

El comando "who" informa de los usuarios que se hallan presentes en el sistema.

```
$ who
jose ttyp05 Ene 27 10:45
conta1 ttyp15 Ene 27 11:05
carmen syscons Ene 27 10:48
```

Los campos corresponden al usuario, terminal asociado y fecha de conexión.

Si deseamos saber quienes somos (cara al sistema, claro - el unix y la filosofía son cosas distintas), usaremos el comando "who am i".

A veces, no coinciden el usuario con el que uno se ha conectado (con login) y el usuario efectivo. Ya se verá más adelante algo más sobre este tema. En este caso, el comando que muestra con que usuario se ha hecho la conexión será "logname".

El comando "date" permite ver la fecha y hora del sistema y también fijarla; si escribimos "date" a secas:

```
$ date
Mon Jan 4 18:46:12 GMT 1995
```

Y si intentamos cambiarla, por motivos obvios, sólamente podremos hacerlo como el usuario "root" (Las implicaciones que el cambio de fecha trae consigo en relación con los backups incrementales, así como cualquier proceso dependiente de la hora, pueden ser importantes).

No obstante, hay veces que se debe hacer; en estos casos, todos los menús de administración de cualquier equipo UNIX traen el comando "Change Date/Time". No obstante, podemos dar desde "root" el siguiente comando:

```
date -u 1202120094
```

para cambiar a día 02 del mes 12 a las 12.00 del año 1994. (El formato para date es mmddhhmmaa).

2.4 Archivos y directorios

Uno de los primeros "manuales" de UNIX (La definición del BSD), establecía que en Unix "todo son archivos". Tenemos varios tipos:

Archivo "normal" con sus variantes que más tarde veremos. Es algo que, referido a un nombre, contiene una secuencia determinada de caracteres (bytes).

El sistema operativo no impone ningún tipo de formato ni de registro. Ejemplos de archivos "normales" puede ser el que nos creamos con algún editor, conteniendo un documento. Por ejemplo, el archivo "/etc/motd" es un archivo cuyo contenido es el mensaje del día. Si ejecutamos el comando "cat" (ver el contenido de un archivo), tendríamos una salida como ésta:

```
$ cat /etc/motd
************************************************
© 1981-95 AT&T Unix System Laboratories
Bienvenido al equipo XXXXXXXXXX
************************************************
```

El mismo comando "cat" es un archivo, lo que ocurre es que, en vez de contener "letras" legibles, contiene código binario que sólo el equipo es capaz de comprender. Podemos probar ésto haciendo:

```
$ cat /bin/cat
kjsdñkqbehls wqej jsdf sdcaj
dfkjsadf df asldfj af ^C ^C ^B aasd ( .... BASURILLA .... )
```

Pues también son archivos normales aquellos que contienen programas compilados (ejecutables).

Directorios, que son archivos cuyo contenido son nombres de archivos. Funcionalmente, se comportan de la misma manera que en sistemas tipo MS-DOS.

Archivos especiales.Ya que es el propio sistema operativo el que debe ocuparse de los detalles "sucios" de acceso a dispositivos, la apuesta de los creadores y continuadores de UNIX fue el de homogeneizar todos estos accesos vía archivo. De tal forma, si tenemos un archivo correspondiente a una impresora, bastará escribir en él para que salga en papel; de la misma manera, si deseamos ver lo que una cinta Exabyte, DAT ó Streamer contiene debe ser suficiente con "leer" lo que el archivo de acceso a esos dispositivos contiene. Si bien la nomenclatura varía entre sistemas, normalmente todos estos tipos residen en el directorio "/dev". Podemos citar algunos de ellos:

/dev/lp0, /dev/lp1	Primera y segunda impresora en paralelo.
/dev/tty00, /dev/tty01	Terminales ó modems en conexión serie.
/dev/eth0	Primera placa ethernet
/dev/dsk/c0d0s0	Primera partición del disco "0" en la controladora "0"
/dev/rmt/0	Unidad de cinta Stremer con rebobinado automático.

Podemos crear un archivo de múltiples formas; utilizando un editor, mediante un programa ó utilizando algún comando. Unicamente se tendrá en cuenta, a la hora de escoger nombres para los archivos, las siguientes reglas:

- El nombre no podrá exceder de 255 letras (14 en versiones UNIX anteriores a la R2.1)
- Utilizar caracteres "normales" (Si bien se puede usar cualquier carácter, desechar aquellos que no sean A-Z /a-z / 0-9). Especial atención merecen aquellos archivos cuyo primer caracter es el punto ".", ya que se consideran "ocultos" en la medida en que determinados comandos, al detectar este hecho, los ignoran.

Vamos a crear un archivo cuyo contenido sea "hola":

```
$ echo "hola" > tonto
$
```

(parece que no ha ocurrido nada)

```
$ cat tonto
hola
$
```

(efectivamente- ahora el comando "ls" nos da un listado de los archivos)

```
$ ls
tonto
$ ls -a
```

```
.
..
tonto
.profile
$
```

Ahora parece que el archivo "tonto" ha creado! Lo que ocurre es que estamos en nuestro directorio de trabajo, y ahí existe desde el principio un archivo "oculto" (el .profile) el cual sólo podemos listar utilizando la opción "-a" (all) del ls. Además de esto, aparecen el "punto" (.) y el "punto-punto", sinónimos de "directorio actual" y "directorio anterior".

Primero, vamos a ver en que directorio estamos, para lo cual emplearemos el comando "pwd" (Print Working Directory).

```
$ pwd
/usr/jose
$ ls
tonto
```

Nuestro directorio de trabajo es "/usr/jose" y tenemos dentro del mismo el archivo "tonto". Utilizando los conceptos de "." y "..", podemos ejecutar el comando "ls" pasando los mismos como argumentos:

```
$ ls .
tonto
```

(listar el contenido del directorio "actual"; funciona igual que "ls" a secas)

```
$ ls ..
jose
paco
antonio
bin
(....)
```

Estamos diciendo que nos liste el "directorio anterior", es decir, "/usr".

Podemos hacer lo mismo cambiando al directorio anterior y ejecutando "ls": nos cambiamos de directorio usando el comando "cd" (Change Directory), con direccionamiento absoluto (indicando todo el camino de subdirectorios desde el directorio "raiz" (/):

```
$ cd /usr
$ pwd
/usr
$ ls
jose
paco
antonio
bin
(...)
```

O bien, cambiando con direccionamiento relativo, y se dice así por ser relativo al directorio donde estamos trabajando:

$ cd ..

(nos cambiamos al directorio anterior, es decir, a "/usr")

```
$ pwd
/usr
ls
jose
paco
antonio
bin
(...)
```

Para volver a nuestro directorio "origen" (home directory), bastaría con ejecutar el comando "cd" a secas; él vuelve siempre al mismo sitio, en este caso, a "/usr/jose", estemos donde estemos.

Si queremos crear un nuevo directorio emplearemos en camino absoluto ó relativo el comando "mkdir" con el argumento "nombre_del_directorio_que_queremos_crear".

Para eliminarlo, ejecutaremos "rmdir" con el mismo argumento:

```
$ mkdir nuevo
$ ls
tonto
nuevo
```

Tenemos ahora el archivo de siempre y un directorio que se llama "nuevo". Pero escribiendo "ls" no vemos a simple vista ninguna diferencia; hay que dar la opción "-F" ó "-l":

```
$ ls -F
tonto
nuevo/
$ ls -l
-rw-r--r-- 1 jose sys 4 Mar 1 11:07 tonto
drw-r--r-- 1 jose sys 84 Mar 1 11:07 nuevo
```

la letra "d" al principio del listado largo nos indica que se trata de un directorio

```
$ cd nuevo
$ pwd
/usr/jose/nuevo
```

nos hemos cambiado al directorio nuevo

```
$ cd ..
$ pwd
/usr/jose
```

hemos vuelto

```
$ rmdir nuevo
$ ls
tonto
```

ya no está!

2.5 Tipos de archivos

Si bien UNIX no impone una estructura a ningún archivo, éstos tendrán características comunes dependiendo para lo que sirvan; podemos agrupar éstos en varios tipos:

- **Ejecutables.** Nornalmente se trata de programas compilados y contienen código binario ininteligible para la mayoría de los humanos pero no así para la máquina. Ejemplos de estos archivos pueden ser los archivos "/bin/ls", "/bin/cat/", ... Todos ellos deben tener activados los permisos de ejecución que más tarde veremos.
- **Binarios**, englobando dentro de esta categoría aquellos que son empleados por programas capaces de entender su contenido (un archivo indexado accesible desde COBOL, por ejemplo) pero no legibles.
- **Texto**, correspondientes a aquellos archivos que contienen registros de caracteres terminados en "nueva línea" y normalmente son legibles. Ejemplo de archivo de texto puede ser el "/etc/motd", "/etc/passwd" y cualquiera que haya sido confeccionado con el editor.
- **Dispositivo.** Cualquier archivo asignado a un dispositivo físico; normalmente residen a partir del subdirectorio "/dev" y son archivos de terminales, de particiones de discos, de puertos de impresora, ... Dentro de esta categoría podemos incluir el archivo especial "/dev/null", el cual tiene como misión el que cualquier cosa que se le mande es desechada.

Cualquier archivo de tipo dispositivo puede tener dos categorías: bien de tipo "caracter", ó "raw mode" (lee y escribe "de golpe") ó de tipo "bloque" ó "cooked mode" (lee y escribe a trozos según el tamaño de buffer que tenga asignado).

Si se tienen dudas acerca del tipo de archivo, podemos ejecutar el comando: "file <nombre_del_archivo>".

```
$ file /bin/ls
/bin/ls: ELF 32-bit MSB executable M88000 Version 1
```

(Ejecutable dependiente del procesador 88000 de Motorola)

```
$ file /dev/null
character special
```

2.6 Tipos de acceso/permisos

En cualquier sistema multiusuario es preciso que existan métodos que impidan que un determinado usuario pueda modificar ó borrar un archivo confidencial, ó incluso leer su contenido. Asimismo, determinados comandos (apagar la máquina, por ejemplo) deben estar permitidos exclusivamente a determinados usuarios, quedando inoperantes para los demás.

En UNIX, estos métodos radican en que cada archivo tiene un propietario, que es el usuario que creó el archivo. Además, los usuarios están divididos en grupos, asignación que normalmente se lleva a cabo por el Administrador del sistemas, dependiendo de la afinidad de las tareas que realizan. El archivo anterior tiene también un grupo, que es el grupo del usuario que lo ha creado. UNIX distingue tres tipos de acceso- lectura, escritura y ejecución- sobre el usuario que lo ha creado, los usuarios del mismo grupo que el que lo creó y todos los demás usuarios.

Por todo lo anteriormente dicho, un archivo puede tener cualquier combinación de los tres tipos de acceso sobre tres tipos de usuarios: el creador, los de su grupo y todos los demás (otros cualquiera que no cumplan ninguna de las dos condiciones anteriores). Para ver los permisos de un archivo cualquiera, empleamos el comando "ls -l" (formato largo):

```
$ ls -l
-rw-r--r-- 1 jose sys 4 Mar 1 11:07 tonto
drw-r--r-- 1 jose sys 84 Mar 1 11:07 nuevo
```

Los campos que lo componen son:

`-rw-r--r--`	tipo de archivo (la primera raya; en el caso de "nuevo", una "d" significa que es un directorio, y una raya "-" significa archivo normal) y 3 grupos de tres correspondientes, los tres primeros, a permisos del usuario; los tres siguientes, a los del grupo, y los tres últimos, a todos los demás.
`jose`	Usuario que creó el archivo.
`sys`	Grupo del creador.
`4`	Tamaño en bytes.
`Mar 1 11:07`	Fecha de creación ó de última modificación.
`tonto`	Nombre del archivo.

Cada grupo de tres elementos pueden ser "rwx", que son: permiso para leer, permiso para escribir y permiso para ejecutar. Una raya significa "carece de permiso".

Por tanto , el archivo "tonto" puede ser leído y escrito (y, por tanto, borrado), por el usuario "jose". Cualquier otro usuario, sea del grupo "sys" ó no, tiene permisos sólo de lectura.

En el caso de directorios, todo igual salvo que en este caso la "x" de ejecutar no tendría sentido; por ello, aquí este carácter significa "el usuario X puede cambiarse a este directorio".

Para darle ó quitarle permisos a un archivo ó directorio, se emplea el comando "chmod <máscara> <nombre_de_archivo>". La máscara es un número octal de hasta un máximo de cuatro cifras correspondiente a sumar los siguientes números:

1 Permiso de ejecución.

2 Permiso de escritura.

4 Permiso de lectura.

Por tanto, para darle máximos permisos al archivo "tonto", ejecutaremos el comando:

```
$ chmod 777 tonto
$ ls tonto
-rwxrwx rwx 1 jose sys 4 Mar 1 11:07 tonto
```

al darle tres sietes (1 + 2 + 4 = 7), el archivo se queda con la máscara rwx. Si queremos sólo lectura para creador, grupo y otros, el comando sería "chmod 444 tonto" y así sucesivamente.

En determinadas ocasiones puede ser necesario cambiar de propietario a un archivo ó directorio; para ello utilizamos el comando "chown <nombre_del_propietario> <archivo>". Igual pasa con los grupos; el comando es "chgrp <nombre_del_grupo> <archivo>"

```
$ chown juan tonto
$ ls tonto
-rwxrwxrwx 1 juan sys 4 Mar 1 11:07 tonto
$ chgrp conta tonto
$ ls tonto
-rwxrwxrwx 1 jose conta 4 Mar 1 11:07 tonto
```

Pero, cuando creamos un archivo, que permisos toma por defecto? El valor de "umask" que tenemos asignado en la sesión complementado con 666 (rw-rw-rw). El comando "umask" a secas nos devuelve dicho valor:

```
$ umask
022
```

En este caso, 666 - 022 = 644, es decir, cualquier archivo que creemos con un editor ó con otros comandos serán creados con permisos 644 (rw-r--r--).

Para cambiar la máscara, usamos el comando umask con la nueva máscara que le quieremos dar:

```
$ umask 000
```

En cuyo caso, todos los archivos a partir del ese momento, y hasta que finalice la sesión, serán creados con "barra libre" para todo el mundo.

2.7 Manipulación de archivos

Podemos examinar una primera tanda de comandos hermanos, que son los siguientes:

```
cp <archivo a copiar> <archivo nuevo>
mv <archivo a mover> <archivo nuevo>
ln <archivo a lincar> <archivo nuevo>
```

"cp" copia el primer argumento al segundo. Valen caminos relativos, es decir

```
$ cp tonto /tmp
$ cp tonto /tmp/nuevo
$ cp /home/jose/tonto /tmp/tonto
```

producirían el mismo resultado; copian el archivo desde el directorio actual al /tmp.

El comando "mv" se comporta igual salvo que el archivo original desaparece; es similar al "RENAME" de MS-DOS.

El comando "ln" hace que pueda existir un contenido con varios nombres:

```
$ ln tonto tonto1
$ ls -l tonto*
-rw-r--r-- 2 jose sys 4 Mar 1 11:07 tonto
-rw-r--r-- 2 jose sys 4 Mar 1 11:07 tonto1
```

ahora "tonto" y "tonto1" hacen referencia al mismo contenido; por tanto, cualquier cambio que realicemos en uno se verá en el otro, y si borramos ahora el archivo "tonto", quedará el " tonto1" con el mismo contenido, que sólo será borrado al borrar el último de los archivos lincados a su contenido.

Debido a la naturaleza del propio comando, no se pueden hacer enlaces (links) entre archivos ó directorios situados en sistemas de archivos distintos.

Para este último caso, existe una opción del comando "ln", el "ln -s" (link simbólico); crea un enlace simbólico entre dos entidades pero sólo como una referencia; en este caso si el archivo original se borra, el link queda suelto y el contenido irrecuperable.

El comando "rm" borra archivos y directorios; para borrar el tonto1 que ya no necesitamos, podemos escribir:

```
$ rm tonto1
```

Valen metacaracteres. Si escribimos

```
$ rm *
```

nos borramos TODO lo que haya en el "directorio actual" SIN pedir confirmación.

Una opción de rm lo hace más seguro al pedirla:

```
$ rm -i tonto1
tonto1 ?
```

La opción más peligrosa puede ser la "-r" (recursivo) que borra archivos y directorios a partir del directorio de arranque. Por tanto, ojo con poner

```
$ rm -r .
```

si estamos en el directorio "raíz" y tenemos privilegios, podemos borrar el contenido de TODOS LOS DISCOS

Por último, veremos un comando de utilidad simililar a "ls" pero con más opciones; el comando "find" muestra en pantalla directorios y/o archivos atendiendo a opciones determinadas. Si bien la sintaxis es demasiado extensa para estudiarla en su totalidad, conviene ver algunos ejemplos:

El comando toma como argumentos declaraciones en formato "find <directorio base> <opciones> -print". Por tanto, para sacar en pantalla todos los archivos y directorios del disco, valdría con:

```
find / -print
```

Si se desean sólo archivos ó sólo directorios:

```
find / -type f -print
```

ó

```
find / -type d -print
```

respectivamente.

Buscar todos los archivos y directorios que tengan más de 30 días de antigüedad:

```
find / -atime +30 -print
```

El resto de las opciones sirve para buscar archivos con determinados permisos, mayores ó menores que un determinado tamaño.

2.8 Directorios del sistema

Si bien la estructura de los directorios varía entre versiones y dialectos de UNIX, podemos mencionar algunos de los más representativos:

/bin, /usr/bin, /usr/sbin, /usr/ucb	Directorios de programas ejecutables. Es aquí donde se mantienen la mayoría de los comandos (/bin/ls , /bin/vi , /usr/bin/chmod, ...)
/etc	Programas y archivos diversos: aquí residen la mayoría de los archivos de configuración y las shell-scripts de arranque y parada del equipo.
/tmp, /usr/tmp, /var/tmp	Directorios "temporales"; el propio sistema operativo los usa como almacenamiento intermedio en muchas de sus tareas; normalmente, todo el contenido de estos directorios se borra al apagar el equipo.
/home, /usr/acct, /var/acct	Directorios de usuarios.
/var/adm, /usr/adm	Directorios de administración: programas de administrador, archivos con informaciones de la paquetería instalada y demás.
/usr/log, /var/log	Directorios para archivos de "log" de sistemas y de programas. Normalmente, existirá un archivo "console" con toda la información que se desvía a la consola del sistema, y varios archivos informativos, de advertencia y de errores del sistema.
/lib, /usr/lib	Directorios de archivos de librería.
/usr/spool	Directorios de spool de archivos de impresión (/usr/spool/lp), de comunicaciones (/usr/spool/uucp) y demás.
/usr/mail	En este directorio se halla el correo de todos los usuarios.
/usr/preserve, /var/preserve	Aquí se guardan todas las copias de archivos incompletamente terminados por los editores de UNIX.
/dev	Directorio de archivos de dispositivos.

2.9 Resumen del editor de pantalla "vi"

Vi es un editor de pantalla completa creado en el UNIX de Berkeley y extendido más tarde a todos los demás dialectos. La ventaja de este editor radica en su compatibilidad con el resto de herramientas de edición UNIX.

Vi contempla, una vez iniciada la sesión con él, dos modos:

Modo comando: es el modo en el que arranca por defecto. Vale para dar comandos tales como: leer un archivo, escribir un archivo, búsqueda ó sustitución de caracteres dentro del texto actual. Este modo es al que se vuelve siempre, pulsando la tecla *Esc*.

Modo insercion/reemplazo: es el que se usa para escribir caracteres. Se entra a este modo desde el modo comando pulsando la letra " i ". Desde

el modo comando, podemos pasar también a modo inserción escribiendo las siguientes letras:

i	Pasar a modo inserción, delante de la posición del cursor.
a	Igual, pero detrás de la posición del cursor.
I	Pasar a modo inserción ,pero empezando a insertar al principio de la línea donde se esté.
A	Igual, pero empezando al final de la línea donde se esté.
o	Pasar a modo inserción, pero abriendo una línea nueva debajo de donde se esté.
O	Igual, pero la línea nueva se abre arriba.
Esc	Pasar a modo comando.
Bkspc	Borra la última letra escrita.
Ctrl-v	(^v), Identifica el carácter que vamos a escibir a continuación como un caracter especial, es decir, un escape, ó salto de hoja (^L), ó cualquier carácter ascii entre el 1 y el 31. (1=^A, 2=^B, ...).

Pasando a modo comando, podemos emplear las siguientes secuencias para movernos por el texto:

^	Ir al principio de la línea.
$	Ir al final de la línea.
l,h	Izquierda / derecha.
j,k	Abajo / Arriba.
^F	Una pantalla adelante.
^B	Una pantalla atrás.
^G	Enseña el número de línea donde estamos posicionados.
1G	Al principio del todo.
G	Al final del todo.
/<cadena>	Busca <cadena> desde la línea actual hasta el final del texto.
/	Sigue buscando más ocurrencias de la cadena.
?<cadena>	Busca <cadena> desde la línea actual para atrás.
?	Sigue buscando más ocurrencias , para atrás.
ZZ	Grabar y salir.
dd	Borrar la línea donde se esté.
J	Juntar la línea donde se esté y la de abajo.
r	Reemplaza una sola letra.
R	Reemplaza todo hasta que se pulse *Esc*.
yy	yank/yank: Marca la línea actual.
p	Copia la línea marcada despues del cursor.
P	Copia la línea marcada antes del cursor.
.	Repite el último cambio.
u	Undo (anula el último cambio)
U	Undo, pero en la línea actual.

Ordenes de "dos puntos": Al pulsar en modo comando el carácter ":", el cursor se baja a la última linea y el editor saca los dos puntos en espera de que le demos algún comando de estos. Salimos de aquí pulsando *Esc*. Algunos son:

:wq!	(write/quit) Grabar y salir.
:w! <nombre_archivo>	(write) Graba el texto en <nombre_archivo>.
:r <nombre_archivo>	(read) Incluye el archivo como parte del texto en edición.
:!<comando>	Ejecutar <comando>. Vuelve al terminar el comando.

:<número>	Ir a la línea <número>.
:q!	(quit) Salir sin grabar.
:se nu	Numera las líneas. (set number, set nonumber - pone y quita).
:1,5de	Borra desde la 1 a la 5.
:1,5co20	Copia desde la 1 a la 5 a partir de la línea 20.
:1,5mo20	Igual, pero mueve. (desaparece desde la línea 1 a la 5).
:g/XXX/s//YYY/g	Cambia en todo el texto XXX por YYY.

Hay muchísimos más comandos, pero con estos es suficiente.

CAPITULO 3. SHELL

3.1 Introducción

La shell es el programa más importante para la mayoría de los usuarios y administradores de UNIX; con la excepción del editor de textos y del menú de administración, posiblemente es con el que más tiempo se trabaja. La shell es el lenguaje de comandos de UNIX; es un programa que lee los caracteres tecleados por los usuarios, los interpreta y los ejecuta.

A diferencia de otros intérpretes más estáticos en otros sistemas operativos, aquí existe además un completo lenguaje de programación que permite adaptar de manera relativamente sencilla el sistema operativo a las necesidades de la instalación.

Una vez que un usuario común se ha registrado cara al sistema con su login y su password, se le cede el control a la shell, la cual normalmente ejecuta dos archivos de configuración, el general (/etc/profile) y el particular (<directorio_del_usuario>/.profile). Una vez aquí, la propia shell despliega el literal "inductor de comandos", que normalmente será:

```
$
```

ó

```
#
```

Y el cursor en espera de que alguien escriba algún comando para ejecutar.

Tipos y propiedades

Ya que, como hemos explicado anteriormente, la shell es un programa, existen varios, cada uno con sus características particulares. Veamos algunas de ellas:

Bourne shell (/bin/sh): Creada por Steven Bourne de la AT&T. Es la más antigua de todas y, por tanto, la más fiable y compatible entre plataformas. Esta es en la que se basan las explicaciones posteriores.

Korn shell (/bin/ksh): Creada por David G. Korn de los laboratorios Bell de la AT&T. Más moderna, toma todos los comandos de la Bourne y le añade varios más así como varias características de reedición interactiva de comandos, control de trabajos y mejor rendimiento en términos de velocidad que la anterior. Existen dos versiones, una primera "maldita" y la actual, de 16/11/1988. Podemos averiguar qué versión tiene la nuestra ejecutando el comando "what /bin/ksh". Pero no es oro todo lo que reluce; cuando se trata de situaciones extremas ó complicadas donde la ksh falla, la de Bourne normalmente está más "blindada".

C Shell, desarrollada por Bill Joy en la Universidad de California y, por tanto, más extendida entre UNIX BSD. Bastante más críptica que la de Bourne, incorpora no obstante sustanciales mejoras.

Estructura de las órdenes - Metacaracteres

Cuando escribimos cualquier comando y pulsamos *INTRO*, es la shell y no UNIX quien se encarga de interpretar lo que estamos escribiendo y ordenando que se ejecute dicho comando. Aparte de los caracteres normales, la shell inter preta otros caracteres de modo especial: un grupo de caracteres se utiliza para generar nombres de archivos sin necesidad de teclearlos explícitamente.

Cuando la shell está interpretando un nombre, los caracteres * ? [ ] se utilizan para generar patrones. El nombre que contenga alguno de estos

caracteres es reemplazado por una lista de los archivos del directorio actual cuyo nombre se ajuste al patrón generado.

Las reglas de generación de patrones son:

*	Vale cualquier cadena de caracteres.
?	Vale un carácter cualquiera.
[..]	Vale cualquiera de los caracteres que coincida con los que estén entre corchetes.

Ejemplo. Supongamos que en nuestro directorio actual tenemos los siguientes archivos:

```
$ ls
tonto
tonta
diario
mayor
```

Veamos cada una de las salidas correspondientes a las reglas anteriores:

(valen todos)

```
$ ls *
tonto
tonta
diario
mayor
```

(todos, pero tienen que acabar en "o")

```
$ ls *o
tonto
diario
```

(que empiecen por tont y cualquier otro carácter)

```
$ ls tont?
tonto
tonta
```

(que empiecen por tont y el siguiente sea "o" ó "a")

```
$ ls tont[oa]
```

3.2 Concepto de comando / proceso

Para comprender la manera en la cual la shell ejecuta los comandos hay que tener en cuenta las circunstancias siguientes:

Tras sacar en pantalla el indicador $, espera a que se le introduzca algo, lo cual será interpretado y ejecutado en el momento de pulsar *INTRO*.

La shell evalúa lo que hemos escrito buscando primero si contiene un carácter "/" al principio. En caso que sea así, lo toma como un programa y lo ejecuta.

Si no, examina si se trata de una función (o un alias, en el caso de la ksh). Una función es una secuencia de comandos identificada por un nombre unívoco, y se verá más adelante. En caso de no encontrar ninguna con ese nombre, busca a ver si se trata de un comando interno (exit, exec, trap, etc.) ó palabra reservada (case, do, done, if, for , ..., etc.), para ejecutarlo ó pedir más entrada. Si ninguna de estas condiciones es cierta, la shell piensa que lo que hemos escrito es un comando, y lo busca dentro de los directorios contenidos en la variable de entorno PATH. Si no está, saca un mensaje del tipo "XXXX: not found", siendo XXXX lo que hemos escrito.

Mejor verlo con un ejemplo: supongamos que escribimos alguna aberración del tipo:

```
$ hola
```

Suponiendo que la variable PATH contenga los directorios /bin, /usr/bin y /etc, la shell busca el comando "/bin/hola", "/usr/bin/hola" y "/etc/hola". Ya que obviamente no existe, la contestación será:

```
sh : hola : not found.
```

La shell utiliza UNIX para la ejecución de procesos, los cuales quedan bajo su control. Podemos definir un proceso aquí como un programa en ejecución. Ya que UNIX es multitarea, utiliza una serie de métodos de "tiempo compartido" en los cuales parece que hay varios programas ejecutándose a la vez, cuando en realidad lo que hay son intervalos de tiempo cedidos a cada uno de ellos según un complejo esquema de prioridades.

Cuando la shell lanza un programa, se crea un nuevo proceso en UNIX y se le asigna un número entero (PID) entre el 1 y el 30,000, del cual se tiene la seguridad que va a ser unívoco mientras dure la sesión. Lo podemos ver ejecutando el comando "ps", el cual nos da los procesos activos que tenemos asociados a nuestro terminal.

Un proceso que crea otro se le denomina proceso padre. El nuevo proceso, en éste ámbito, se le denomina proceso hijo. Este hereda casi la totalidad del entorno de su padre (variables, etc.), pero sólo puede modificar su entorno, y no el del padre.

La mayoría de las veces, un proceso padre se queda en espera de que el hijo termine; esto es lo que sucede cuando lanzamos un comando; el proceso padre es la shell, que lanza un proceso hijo (el comando). Cuando este comando acaba, el padre vuelve a tomar el control, y recibe un número entero donde recoge el código de retorno del hijo (0=terminación sin errores, otro_valor=aquí ha pasado algo).

Cada proceso posee también un número de "grupo de procesos". Procesos con el mismo número forman un solo grupo, y cada terminal conectado en el sistema posee un solo grupo de procesos.

Comando ps -j. Si uno de nuestros procesos no se halla en el grupo asociado al terminal, recibe el nombre de proceso en background (segundo plano).

Podemos utilizar algunas variantes del comando "ps" para ver que procesos tenemos en el equipo:

ps	muestra el número de proceso (PID), el terminal, el tiempo en ejecución y el comando. Sólo informa de nuestra sesión.
ps -e	de todas las sesiones.
ps -f	full listing: da los números del PID, del PPID (padre), uso del procesador y tiempo de comienzo.
ps -j	da el PGID (número de grupo de los procesos- coincide normalmente con el padre de todos ellos)

Este comando puede servirnos para matar ó anular procesos indeseados. Se debe tener en cuenta que cada proceso lleva su usuario y por tanto sólo el (ó el superusuario) pueden matarlo.

Normalmente, si los programas que componen el grupo de procesos son civilizados, al morir el padre mueren todos ellos siempre y cuando el padre haya sido "señalizado" adecuadamente. Para ello, empleamos el comando "kill -<número de señal> PID", siendo PID el número del proceso ó del grupo de procesos.

Los números de señal son:

-15	TERM ó terminación. Se manda para que el proceso cancele ordenadamente todos sus recursos y termine.
-1	corte
-2	interrupción.
-3	quit
-5	hangup
-9	la más enérgica de todas pero no permite que los procesos mueran ordenadamente.

3.3 Entrada y salida. Interconexión

Para cada sesión, UNIX abre tres archivos predeterminados, la entrada estándar, la salida estándar y el error estándar, como archivos con número 0,1 y 2. La entrada es de donde un comando obtiene su información de entrada; por defecto se halla asociada al teclado del terminal. La salida estándar es donde un comando envía su resultado; por defecto se halla asociada a la pantalla del terminal; el error estándar coincide con la salida estándar, es decir,con la pantalla.

A efectos de modificar este comportamiento para cualquier fin que nos convenga, la shell emplea 4 tipos de redireccionamiento:

<	Acepta la entrada desde un archivo.
>	Envía la salida estándar a un archivo.

>> Añade la salida a un archivo existente. Si no existe, se crea.

| Conecta la salida estándar de un programa con la entrada estándar de otro.

Intentar comprender estas explicaciones crípticas puede resultar cuanto menos confuso; es mejor pensar en términos de "letras":

< En vez de coger líneas del teclado, las toma de un archivo. Mejor, adquirir unos cuantos conceptos más para entender un ejemplo.

> Las letras que salen de un comando van a un archivo.

Supongamos que ejecutamos el comando ls usando esto:

```
$ ls > cosa
$
```

(parece que no ha pasado nada)

```
$ ls
tonto
tonta
diario
mayor
cosa
$
```

(se ha creado un archivo que antes no estaba, cosa)

```
$ cat cosa
tonto
tonta
diario
mayor
cosa
$
```

(y contiene la salida del comando "ls")

>> En vez de crear siempre de nuevo el archivo, añade las letras al final del mismo.

| El comando a la derecha toma su entrada del comando de la izquierda. El comando de la derecha debe estar programado para leer de su entrada; no valen todos.

Por ejemplo, ejecutamos el programa "cal" que saca unos calendarios muy aparentes. Para imprimir la salida del cal, y tener a mano un calendario sobre papel, podemos ejecutar los siguientes comandos:

```
$ cal 1996 > /tmp/cosa
$ lp /tmp/cosa
```

Hemos usado un archivo intermedio. Pero, ya que el comando "lp" está programado para leer su entrada, podemos hacer (más cómodo, y ahorramos un archivo intermedio):

```
$ cal 1996 | lp
```

El uso de la interconexión (pipe) exige que el comando de la izquierda lance información a la salida estándar. Por tanto, podríamos usar muchos comandos, tales como cat, echo, cal, banner, ls, find, who, ... Pero, el comando de la derecha debe estar preparado para recibir información por su entrada; no podemos usar cualquier cosa. Por ejemplo, sería erróneo un comando como:

```
ls | vi
```

(si bien ls si saca caracteres por su salida estándar, el comando "vi" no está preparado para recibirlos en su entrada).

Y la interconexión no está limitada a dos comandos; se pueden poner varios, tal que así:

```
cat /etc/passwd | grep -v root | cut -d":" -f2- | fold -80 |
lp
```

(del archivo /etc/passwd sacar por impresora aquellas líneas que no contengan root desde el segundo campo hasta el final con un ancho máximo de 80 columnas).

Normalmente, cualquier comando civilizado de UNIX devuelve un valor de retorno (tipo ERRORLEVEL del DOS) indicando si ha terminado correctamente ó bien si ha habido algún error; en el caso de comandos compuestos como éste, el valor global de retorno lo da el último comando de la interconexión; en el último ejemplo, el retorno de toda la línea sería el del comando "lp".

3.4 Variables de entorno

Una variable de entorno en la shell es una referencia a un valor. Se distinguen dos tipos: locales y globales.

Una variable local es aquella que se define en el shell actual y sólo se conocerá en ese shell durante la sesión de conexión vigente.

Una variable global es aquella que se exporta desde un proceso activo a todos los procesos hijos.

Para crear una variable local:

```
# cosa="ANTONIO ROMERO"
```

Para hacerla global

```
# export cosa
```

Para ver que variables tenemos:

```
# set
LOGNAME=root
TERM=vt220
PS1=#
SHELL=/bin/sh
(salen mas)
```

Una variable se inicializa con la expresión <variable>=<valor>. Es imprescindible que el carácter de igual '=' vaya SIN espacios. Son lícitas las siguientes declaraciones:

```
# TERM=vt220
# TERM="vt220"
```

(TERM toma el mismo valor en ambos casos)

```
# contador=0
```

Una variable se referencia anteponiendo a su nombre el signo dólar '$'.

Utilizando el comando 'echo', que imprime por la salida estándar el valor que se le indique, podemos ver el contenido de algunas de estas variables:

```
# echo TERM
TERM
```

(!! MUY MAL!!- Hemos dicho que la variable se referencia por $).

```
# echo $TERM
vt220
```

(AHORA SI)

Otra metedura de pata que comete el neófito trabajando con variables:

```
# $TERM=vt220
```

Y el señor se queda tan ancho. Investigando que es esto, la shell interpreta que le queremos poner algo así como "vt220=vt220", lo cual es erróneo.

Para borrar una variable, empleamos el comando "unset" junto con el nombre de la variable que queremos quitar, como:

```
# echo $cosa
ANTONIO ROMERO
# unset cosa
# echo $cosa
#
```

Cuidadito con borrar variables empleadas por programas nativos de UNIX, tales como TERM! Si borráis ésta variable, el editor "vi" automáticamente dejará de funcionar.

Otro problema que es susceptible de suceder es el siguiente: supongamos una variable denominada COSA y otra denominada COSAS. La shell, en el momento de evaluar la expresión "$COSAS", se encuentra ante la siguiente disyuntiva:

- Evaluar $COSA y pegar su contenido a la "S" (<contenido de COSA> + "S")
- Evaluar $COSAS, empleando intuición.

Cara a la galería, ambas evaluaciones por parte de la shell serían correctas, pero dependiendo de lo que nosotros queramos hacer puede producir efectos indeseados. A tal fin, en conveniente utilizar los caracteres "llave" -{}- para encerrar la variable que queremos expandir. De tal forma, para reflejar "COSA", escribiríamos:

```
${COSA}
```

Y para reflejar "COSAS",

```
${COSAS}
```

Con lo que tenemos la seguridad de que las variables siempre son bien interpretadas. Las llaves se utilizan SIEMPRE en el momento de evaluar la variable, no de asignarle valores. No tiene sentido hacer cosas como

```
{COSAS}=tontería.
```

Algunas de las variables usadas por el sistema ó por programas del mismo son:

HOME	Directorio personal. Usado por el comando "cd", se cambia aquí al ser llamado sin argumentos.
LOGNAME	Nombre del usuario con el que se ha comenzado la sesión.
PATH	Lista de rutas de acceso, separadas por dos puntos ‘:’ y donde una entrada con un sólo punto identifica el "direcorio actual". Son válidas asignaciones como: `# PATH=$PATH:/home/pepe:/home/antonio`
PS1	Símbolo principal del indicador de "preparado" del sistema. Normalmente, su valor será ‘#’ -o ‘$’.
TERM	Tipo de terminal.

Podemos ver como se inicializan las variables consultando los archivos de inicialización. Estos archivos son:

/etc/profile	Archivo de inicialización global. Significa que, tras hacer login, todos los usuarios pasan a través del mismo. Inicializa variables como PATH, TERM, ...
<directorio usuario>/.profile	Archivo particular, reside en el "home directory" del usuario en cuestión. Es, por panto, particular para cada uno de ellos y es aquí donde podemos configurar cosas tales como que les salga un menú al entrar, mostrarles el correo, ...

3.5 Otros programas para archivos

Comando: **wc** <opciones> <nombre-archivo>

Misión: cuenta lineas, palabras o caracteres de un archivo.

Opciones:

-l	lineas
-c	letras
-w	palabras

Ejemplo:

```
wc -l /etc/passwd
```

El número de líneas del archivo /etc/passwd.

Comando: **sort** <opciones> <indicador de campo> <archivo>

Misión: ordena un archivo de acuerdo a una secuencia de odenacion determinada.

Ejemplo: Dado un archivo con:

codigo nombre ciudad telefono

```
$ sort archivo # ordena por todos los campos
$ sort +1 archivo # ordena a partir del campo 1 (nombre ;
codigo es el 0)
$ sort +2n archivo # ordena por nombre , en secuencia
invertida.
$ sort +2n -t":" archivo # igual que antes , pero suponiendo
que el separador es ":".
```

Comando: **grep** <cadena a buscar> <archivo>

Misión: buscar apariciones de palabras ó textos en archivos.

Ejemplo:

Busca las líneas que contienen "root" en el archivo.

```
$ grep root /etc/passwd
root:Ajiou42s:0:1:/:/bin/sh
```

Comando: **diff** <archivo1> <archivo2>

Misión: Encuentra las diferencias entre dos archivos.

Ejemplo:

```
$ cat nombres1
jose
juan
roberto
$ cat nombres2
jose
roberto
$ diff nombres1 nombres2
2d1
< juan
```