

[www.ac.upc.es/homes/cristina/problemas.pdf](http://www.ac.upc.es/homes/cristina/problemas.pdf)

# Iniciación al Unix: el shell

*Arquitectura de Computadores y  
Sistemas Operativos  
(Ingeniería Electrónica)*

Septiembre 2000

Cristina Barrado Muxí (D6-111)

## Capítulo 1: El intérprete de comandos

- 1- Definición
- 2- Login / identificación
- 3- Algunos comandos y el *help*
- 4- Sistema de ficheros
- 5- Procesos
- 6- Protección
- 7- Redirección
- 8- Pipes
- 9- Unix vs. Linux

### Objetivos

- Conocer y ser capaces de usar los comandos básicos
- Introducción a conceptos básicos

## **1- Definición de intérprete de comandos**

Primera definición: es el interfaz entre el usuario y el S.O.

En Windows tenemos dos:

- interfaz gráfico: escritorio
- interfaz texto: consola MS-DOS

En Unix tenemos un interfaz de texto llamado **shell**. También encontramos entornos de ventanas (X-windows).

Segunda definición: es un programa que interpreta y ejecuta órdenes del usuario.

Es un programa es potente, basado en primitivas sencillas pero programables. En Unix los comandos son algo crípticos, pues en muchos casos evitan las vocales.

Cualquier usuario puede escribir un intérprete de comandos propio, ya que no necesita privilegios. Se ejecuta en espacio de usuario.

## 2- Login / identificación

Cada usuario interactivo puede establecer una o mas sesiones con el sistema. Para ello usa un terminal que consta de pantalla y teclado. Opcionalmente puede tener otros dispositivos como ratón, micrófono, cámara,...

Al encender un terminal, el S.O. lo detecta y escribe en la pantalla el nombre de la máquina y la versión de S.O. A continuación queda a la espera de que algún usuario se identifique:

```
username: cristina  
password:*****
```

Verificado el usuario, se inicia una sesión. Una sesión finaliza con el comando **logout**.

En pantalla aparece un carácter especial, el *prompt*, por ejemplo un '\$' o un '>'. Su aparición indica al usuario que el intérprete esta esperando una orden.

### 3- Algunos comandos y el *help*

#### lista de algunos comandos de *shell* (en negrita están los más básicos):

<b>ar</b>	- creación de librerías	<b>make</b>	- generación y mantenimiento de ficheros
<b>at, batch-</b>	- ejecución por lotes (no interactiva)	<b>man</b>	- manual on-line
<b>awk</b>	- filtro	<b>mkdir</b>	- crea un directorio
<b>cat</b>	- muestra en contenido de un fichero	<b>more</b>	- muestra ficheros en pantalla
<b>cc, gcc</b>	- compilador de C	<b>mv</b>	- mover ficheros (cambio de nombre)
<b>cd</b>	- cambiar de directorio	<b>passwd</b>	- cambia password
<b>chgrp</b>	- cambio de grupo	<b>ps</b>	- estado de los procesos
<b>chmod</b>	- cambio de permisos de acceso	<b>pwd</b>	- directorio de trabajo (path working directory)
<b>chown</b>	- cambio de propietario	<b>read, line-</b>	- leer línea de la entrada estándar
<b>compress, uncompress</b>	- comprimir	<b>rm</b>	- borrar fichero (remove)
<b>cp</b>	- copiar	<b>rmdir</b>	- borrar directorio
<b>cut</b>	- corte vertical de un fichero	<b>sleep</b>	- bloqueo temporal
<b>date</b>	- fecha y hora	<b>sort</b>	- ordenar
<b>dbx</b>	- debugger	<b>tail</b>	- final de un fichero
<b>diff</b>	- compara ficheros de texto	<b>talk</b>	- conexión interactiva con otro usuario
<b>echo</b>	- escribir en la salida estándar	<b>tar</b>	- encapsular (tape archives)
<b>elm</b>	- lector de correo electrónico	<b>tee</b>	- duplicar salida estándar a otro canal
<b>env</b>	- variables de entorno	<b>telnet</b>	- conexión a otro host vía telnet
<b>expr</b>	- evalúa expresiones	<b>test</b>	- evalúa condiciones
<b>f77</b>	- compilador de Fortran	<b>time</b>	- tiempo de duración de una ejecución
<b>file</b>	- tipo de fichero	<b>touch</b>	- actualiza el tiempo de modificación de un fichero
<b>find</b>	- búsqueda de ficheros	<b>tr</b>	- traductor
<b>ftp</b>	- protocolo de transferencia de ficheros	<b>tty</b>	- nombre del terminal
<b>grep</b>	- búsqueda de patrones	<b>uniq</b>	- lista líneas repetidas
<b>head</b>	- principio de un fichero	<b>vi</b>	- editor de pantalla básico
<b>hostname</b>	- nombre de la máquina	<b>wc</b>	- contar palabras
<b>kill</b>	- interrumpir proceso	<b>whatis</b>	- describe la función de un comando
<b>ld</b>	- linker	<b>whereis</b>	- busca ficheros
<b>login</b>	- entrada al sistema	<b>which</b>	- devuelve el path de un ejecutable
<b>logout</b>	- salir del sistema	<b>who</b>	- usuarios conectados
<b>lp, cancel</b>	- acceso a la impresora	<b>whoami</b>	- quien soy yo? - devuelve el login_name propio
<b>ls</b>	- listar ficheros		
<b>mail</b>	- correo electrónico		

Por ejemplo:

```
$ date
Tue Sep 19 08:43:22 MET DST 2000
```

```

$
$ who
sniar pts/0 Sep 15 19:20 (perpinya.ac.upc.es)
jmanel pts/1 Sep 19 07:54 (busquets-i-bautravers.ac.upc.es)
sniar pts/2 Sep 15 19:23 (perpinya.ac.upc.es)
cristina pts/7 Sep 19 08:23 (parera.ac.upc.es)
yolandab pts/8 Sep 18 08:48 (xt509.eupvg.upc.es)
bgoeman pts/16 Sep 18 10:15 (ros.ac.upc.es)
yolandab pts/17 Sep 18 14:13 (xt509.eupvg.upc.es)
jzalamea pts/19 Sep 14 11:19 (serra.ac.upc.es)
$
$ finger
jzalamea Javier Zalamea Leo 12:15 34 fonoll (serra.ac.upc.es)
larri Josep Lluís Larrib sh 2 week *r7 pons (ros.ac.upc.es)
llorenc Llorenc Cerda 14:27 17 fonoll (giralt.ac.upc.es)
marcoa Marco Antonio Peny 15:04 1 llobet (pcrigel.ac.upc.es)
marisa Marisa Gil 6 days 12 goday (piquer.ac.upc.es)
marta Marta Jimenez 0:51 *1* florensa (martino.ac.upc.es)
(...)
$

```

La estructura general de un comando es:

```
$ comando [ - opciones ] [ argumentos ]
```

donde los corchetes ([ ]) indican opcionalidad. Es decir, las opciones y los argumentos pueden aparecer o no en el comando.

Las opciones suelen ser una o mas letras precedidas por un guión. Las opciones de un comando pueden variar de un Unix a otro.

Por ejemplo, el comando **who** tiene entre otras una opción **-u** que añade información útil y otra opción **-H** que incluye una fila con las cabeceras:

```

$ who -u
sniar pts/0 Sep 15 19:20 old 14570 (perpinya.ac.upc.es)
(...)
$ who -uH
NAME LINE TIME IDLE PID COMMENTS
sniar pts/0 Sep 15 19:20 old 14570 (perpinya.ac.upc.es)
(...)

```

Los argumentos dependen del comando. Algunos comandos, como **man**, requieren obligatoriamente un argumento.

Muchos comandos del shell se interpretan como nombres de dispositivos que contiene los datos de entrada sobre los que aplicar una orden determinada. Por ejemplo:

```
$ cat fichero
```

indica que se quiere mostrar (concatenar) el contenido del fichero. En ausencia de argumentos, estos comandos asumen que los datos a procesa llegan por el canal de entrada estándar (ver Apartado 7-Redirección).

### 3.1 El help

Para ayudas interactivas tenemos habitualmente tres opciones: usar el comando **man**, usar la opción **-h** o usar la utilidad **info** de *gnu*.

- man

Es el comando más importante y estándar de ayuda *on-line*. Accede a las páginas del manual y las formatea para su salida en pantalla. En particular puede pedir ayuda sobre sí mismo introduciendo:

```
$ man man
```

obtenemos:

```
Reformatting page. Wait... done
```

```
MAN(1)          USER COMMANDS          MAN(1)
```

#### NAME

```
man - display reference manual pages; find reference pages
by keyword
```

#### SYNOPSIS

```
man [-] [-t] [-M path] [-T macro-package] [[section] title
...] ...
man [-M path] -k keyword ...
man [-M path] -f filename ...
```

#### DESCRIPTION

```
man displays information from the reference manuals. It can
display complete manual pages that you select by title, or
one-line summaries selected either by keyword (-k), or by
the name of an associated file (-f).
```

```
A section, when given, applies to the titles that follow it
on the command line (up to the next section, if any). man
looks in the indicated section of the manual for those
--More--(10%)
```

Lo que se visualiza es el resultado de una cadena de comandos enlazados mediante *pipes*. El último comando de la cadena es el comando

**more** que encamina la información que recibe hacia el terminal, pero bloqueándose en cada nueva pantalla. Si se quiere ver la página siguiente se pulsa la barra espaciadora, y si se quiere abandonar la consulta se pulsa la tecla ‘q’ (de *quit*).

Al final de la última página del manual suele incluirse un conjunto de referencias a otras páginas del manual relacionadas con la que se está consultando.

Las páginas del manual se organizan en 8 secciones según se traten de comandos de *shell* (sección 1), llamadas a sistema (sección 2), librerías de lenguaje (sección 3), etc.

En el caso de consultar comandos de *shell* (sección 1), en el apartado de *synopsis* aparecen entre corchetes las partes del comando consultado que son opcionales. Por ejemplo, el comando **man** tiene la opción **-k** que permite buscar en las páginas de ayuda usando una clave.

- **passwd**

Éste es el comando que necesitamos para cambiar el *password*. Para conocer qué comando hemos de aplicar consultamos los manuales con la clave *password* y obtenemos:

```
$ man -k password

getpass( ) (3C) - read a password
getpwent( ), getpwuid( ), getpwnam( ), setpwent( ), endpwent( ),
    fgetpwent( ) (3C) - get password file entry
getspwent( ), getspwuid( ), getspwaid( ), getspwnam( ), setspwent( ),
    endspwent( ), fgetspwent( ) (3C) - get secure password
    file entry
passwd(1) - change login password
passwd(4) - password file, pwd.h
putpwent( ) (3C) - write password file entry
putspwent( ) (3C) - write secure password file entry
pwck, grpck(1M) - password/group file checkers
pwget, grget(1) - get password and group information
vipw(1M) - edit the password file
yppasswd( ) (3N) - update user password in Network Information
    Service
yppasswd(1) - change login password in Network Information System
```

---

De la sección 1 vemos que hay varios comandos relacionados. En este caso el que buscamos es el comando *passwd*. Para conocer su forma de utilización y sus opciones podéis consultar su página de manual:

```
$ man passwd
```

o usar la opción de ayuda rápida del propio comando que casi todos suelen tener:

```
$ passwd -h
```

## 4- El sistema de ficheros

El sistema de ficheros (S.F.) organiza la información no volátil (discos, CDs, DVDs, Cintas,...) de forma jerárquica, usando un árbol invertido. En Unix existe un sólo árbol que abarca todos los dispositivos mientras que en Windows hay un árbol para cada dispositivo.

La raíz del árbol se representa con un carácter especial: `/` en Unix y `\` en Windows. Los nodos del árbol pueden ser terminales (hojas) o intermedios. Los nodos intermedios son directorios (o carpetas). Las hojas son los ficheros.

Para identificar un nodo del sistema de ficheros podemos usar el nombre (*pathname*) **absoluto**. Consiste en una cadena de caracteres que comienza con el símbolo del directorio raíz y continúa con los nombres de los nodos (directorios) atravesados hasta llegar al nodo a identificar, separados los nombres de los directorios con el carácter especial de la raíz. Por ejemplo:

```
/usr/alumnos/alopez/prac3.c
```

Cuando un usuario se identifica en el login el S.O. lo posiciona dentro del árbol en su directorio de entrada (*homedir*). Con el comando `cd` el usuario puede moverse por el árbol, modificando el directorio actual de trabajo (`cwd` - *current working directory*). Cualquier nodo del árbol puede identificarse a partir del camino desde el directorio actual de trabajo. Es lo que se conoce como nombre (*pathname*) **relativo**. Por ejemplo:

```
prac3.c          - desde /usr/alumnos/alopez
```

./prac3.c	- desde /usr/alumnos/alopez
alopez/prac3.c	- desde /usr/alumnos
../alopez/prac3.c	- desde /usr/alumnos/vperez

El “.” y el “..” son dos nombres de directorios que existen en cualquier directorio de Unix. El directorio “.” hace identifica al propio directorio y el “..” identifica al directorio padre (o antecesor en el árbol).

Los nodos de un S.F. Unix pueden ser de otros tipos además de directorios y ficheros. Por ejemplo los nodos /dev/tty\* identifican los terminales, los /dev/dsk\* identifican discos, los /proc/\* procesos activos. EL carácter especial “\*” o comodín equivale a cualquier cadena de caracteres. En cualquier lugar del árbol podemos encontrar o crear nodos de tipo enlace (*link*) y nodos para comunicación (*fifos*).

## 5- Procesos

Primera definición: un proceso es un programa en ejecución

Un programa es un conjunto de ‘ordenes (sentencias) escritas en un lenguaje de programación. La estructura típica de un programa es: leer datos, procesarlos y generar un resultado. Un programa es en ente estático y esta almacenado en el sistema de ficheros.

Segunda definición: un proceso es un **entorno** de ejecución

Cuando se activa un programa se precisan recursos del computador: memoria física, procesador, acceso a dispositivos,... y estas necesidades varían durante la ejecución. El proceso es un ente dinámico: nace, crece, muere, hereda, se reproduce, descansa, duerme, espera, mata, se alimenta, mata, se comunica,...

Para crear un proceso se suele parar por las siguientes fases:

- Edición: Se genera un programa en algún lenguaje de alto nivel (fichero fuente)
- Compilación: Se traduce el programa a lenguaje ensamblador (fichero objeto)
- Linkado/montaje: Se enlaza el programa con las librerías del lenguaje (fichero ejecutable).
- Ejecución/carga en memoria: Se activa el programa creando un proceso.

Si se usa un lenguaje de programación interpretado no es necesario compilar y linkar el fichero para obtener el ejecutable. El fichero fuente es también ejecutable.

Los comandos del intérprete de comandos son, en su mayoría, programas ejecutables disponibles por todos los usuarios en `/bin/`. Por lo tanto, cada vez que ejecutamos uno de ellos, estamos creando un proceso. Es más, el propio intérprete de comandos que atiende a un usuario es un proceso creado al inicio de la sesión y que muere al finalizarla.

Observad que, en un S.O. multiusuario, el intérprete de comandos está activo varias veces. Es decir, tenemos varios procesos intérprete de comandos ejecutándose a la vez. Todos se han generado a partir de un mismo programa, el `/bin/sh`. Pero cada ejecución es un entorno diferente: las variables del programa son las mismas, pero sus valores no; se hacen las mismas operaciones para leer datos o escribir resultados, pero en terminales diferentes, se tiene un directorio actual de trabajo pero en cada proceso fluctúa independientemente....

La creación de procesos hijos puede ser síncrona (lo más habitual desde shell) o asíncrona. Un proceso hijo es síncrono si el padre espera su finalización para continuar y es asíncrono si ambos procesos proceden su ejecución en paralelo.

Desde shell se indica que un comando se debe ejecutar en paralelo con el intérprete de comandos usando el carácter especial `'&'`. Por ejemplo:

```
$ cp fuente destino &  
$
```

el shell puede ejecutar nuevas órdenes sin esperar a que acabe la copia de los dos ficheros.

En una misma línea de comandos se pueden indicar varias órdenes y su relación entre ellas usando los caracteres especiales ‘;’ y ‘&’. Comandos separados por ‘;’ se ejecutan secuencialmente mientras que los separados por ‘&’ se ejecutan en paralelo. Por ejemplo:

```
$ cp fuente destino ; more destino  
$ cp fuente destino & cat fuente
```

## 6- Protección

Los procesos son elementos activos que el S.O. debe controlar.

El hardware y los ficheros son los elementos pasivos que el S.O. debe proteger.

Para ello es necesario que el S.O. guarde la información necesaria en unos y otros. Básicamente se guarda la información de propietario y de accesos permitidos (protecciones) de la siguiente forma:

- Elementos pasivos (p.e. un fichero): propietario + protecciones
- Elementos activos (procesos): propietario

Cuando un proceso intenta un acceso a un objeto el S.O. verifica los propietarios de ambos elementos. Establecida la relación de propiedad, la información de protección correspondiente establece si el acceso es correcto.

En Unix las protecciones distinguen tres tipos de acceso:

- r - acceso de lectura
- w- acceso de escritura/modificación
- x - acceso de ejecución

y la relación de propiedad se determina según 3 dominios:

- u - usuario/propietario
- g - group / grupo al que pertenece el propietario
- o - others / resto de usuarios que no son ni el propietario ni de su grupo

Las protecciones son, por lo tanto, una matriz 3x3 de accesos y dominios (rwx) x (ugo). La forma de mostrarlas es usando una lista que concatena las filas (ugo) de la matriz. Por ejemplo:

```
rwX rw- r--
```

indica que el propietario tiene todos los permisos de acceso, los usuarios del grupo del propietario tienen los accesos de lectura y escritura, pero no de ejecución y el resto de usuarios sólo pueden leer.

La información para identificar el usuario son:

- uid - identificador del usuario
- gid - identificador del grupo

Por lo tanto, el S.O. guarda infracción del (uid+gid) para cada proceso y la información (uid+gid) + ( (ugo) x (rwx) ) para cada fichero. Contrastando los uid y gid establece si el proceso es del propietario o del grupo o de otros y aplica la fila de accesos correspondiente.

La información sobre el propietario de los procesos se hereda de padres a hijos. Es decir, si un proceso pertenece a un usuario, sus procesos hijo también.

## 7- Redirección

Una de las funciones habituales de todo proceso es comunicarse con el exterior. Por ejemplo, al leer los datos de entrada o a generar los resultados. Para ello el sistema le ofrece un mecanismo genérico que denominamos canales de entrada/salida (también conocidos como *file descriptors*, *streams*, *handlers*, ...).

Un canal de E/S es una vía de comunicación del proceso con un dispositivo del sistema operativo. Un dispositivo de E/S es por ejemplo un fichero o un periférico. En Unix un canal se identifica con un entero positivo y pequeño: 0, 3, ...

A los canales 0, 1 y 2 se les denomina también canales de entrada estándar (**stdin**), salida estándar (**stdout**) y error estándar (**stderr**) respectivamente. Por ejemplo, el intérprete de comandos que atiende a un usuario interactivo tiene sus stdin, stdout y stderr asociados al terminal.

Los canales de E/S se **heredan** de padres a hijos. Ésto quiere decir que si un proceso padre posee los canales 0,3 y 4, sus procesos hijos tendrán también esos mismos canales y accederán con ellos a los mismos dispositivos que el proceso padre. Es decir, heredar equivale además a compartir.

Por eso, todo proceso creado por el intérprete de comandos hereda sus tres canales de E/S estándar y su salida aparece en el terminal asociado al shell padre.

La **redirección** de E/S es un mecanismo que ofrece el S.O. que permita modificar los canales heredados de un proceso. Desde shell se pueden usar los caracteres especiales ‘<’ y ‘>’ para redireccionar el stdin y stdout respectivamente. Por ejemplo:

```
$ ls f*
fmconsole.log  fmdictionary  fmfilesvisited
$
$ ls f* > salida
$ more salida
fmconsole.log  fmdictionary  fmfilesvisited
$
```

## 8- Pipes

Las pipes son un dispositivos de E/S que ofrece el sistema a los procesos que les permite comunicarse con otros procesos.

Desde shell se pueden crear dos procesos comunicados por pipe usando el carácter especial '|'. Los dos procesos se ejecutan en paralelo y los datos van en un único sentido. Por ejemplo:

```
$ cat fich_largo | more
```

activa dos nuevos procesos en paralelo, **cat** y **more**, de forma que los datos que salen por el canal stdout del **cat** se redireccionan a la pipe de donde toma sus datos de entrada el proceso **more**.

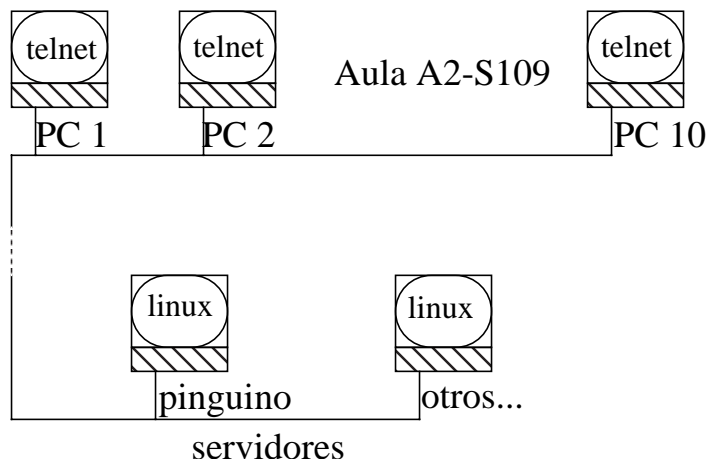
## 9- Unix vs. Linux

Unix es un sistema operativo multiusuario y multitarea. Esto quiere decir que múltiples usuarios y múltiples tareas pueden compartir los recursos del sistema a la vez.

Linux es una implementación concreta del Unix para PCs. El intérprete de comandos propio de Linux se denomina *bash* (*Bourne Again shell*) y es una evolución del (*Bourne*) *shell* de Unix.

El entorno de trabajo habitual en Unix es el mismo que se ofrece en la asignatura de LabARISO, es decir, el acceso a un mismo servidor Unix desde diferentes estaciones de trabajo. Los usuarios se conectan a una máquina (un PC con Windows en el caso de LabARISO) y desde allí establecen una conexión de sesión usando, por ejemplo, `telnet`. El ordenador remoto (un PC con Linux en el caso de LabARISO) es capaz de atender a todos los usuarios en paralelo. Es más, cada usuario

puede establecer más de una conexión remota al mismo. Los ordenadores locales hacen de clientes y el ordenador remoto de servidor.



Cuando un usuario (por ejemplo, en su casa) instala Linux en su PC el entorno de trabajo es diferente. En un mismo ordenador pueden estar instalados varios S.O., pero sólo uno de ellos está activo. Al arrancar el PC se elige el S.O. y se accede a él sin necesidad de usar ningún protocolo de conexión. Aunque el S.O. sea multiusuario, ningún otro usuario estará compartiendo el ordenador. En este entorno, si un usuario quiere, por ejemplo, copiar un fichero desde el Sistema de Ficheros de Unix a la disquetera puede usar el comando linux **mcopy** (copia de formato Unix a formato MS-DOS y viceversa). El contenido del disquete será visible después desde Windows.

En el entorno remoto, es decir, usando una conexión de sesión al servidor Unix, el comando **mcopy** no permite hacer una copia directa a `a:` porque el sistema operativo (su administrador) tendrá protegido ese dispositivo para el resto de usuarios. Es más, la disquetera es un dispositivo físico del servidor remoto, por lo cual, aún suponiendo que no estuviera protegida, no conseguiríamos una copia sobre el disquete de nuestra estación de trabajo.

La solución para poder hacer esa copia es usar un servicio de transferencia de ficheros, como por ejemplo el **ftp** (*file tranfer protocol*). Desde el

ordenador cliente se establece una nueva conexión al servidor a través del servicio **ftp** y se copian los ficheros necesarios (**put**: para transferir del cliente al servidor, **get** para transferir del servidor al cliente).