Comandos básicos de Unix

Dr. Roberto Gómez Cárdenas
ITESM-CEM Dpto. Tecnologías Información

November 4, 2008

Resumen:

La siguiente es una lista que describe la sintaxis y funcionamiento de los principales comandos del sistema operativo Unix. La mayor parte de la información presentada en este documento fue obtenida a partir del comando `man` *de Unix. Este documento sólo describe parte de lo desplegado por* `man`*. Si el lector desea información más profunda se recomienda utilizar dicho comando.*

1 Historia de Unix

Este sistema operativo fue diseñado originalmente a finales de los años sesenta y principio de los años setenta por un grupo de investigadores que trabajaba en AT&T. Su sencillez y elegancia llamaron la atención de investigadores de las universidades y la industria. Unix ha alcanzado una posición de extraordinaria importancia, siendo el único sistema operativo que las compañías están dispuestas a aceptar como estándar preferido de sistema operativo abierto. Unix es el único sistema operativo que se ha instalado en todo tipo de computadoras, desde las microcomputadoras hasta las supercomputadoras, y es el único sistema operativo que implantan casi todos los fabricantes importantes de computadoras.

El sistema operativo Unix encuentra sus orígenes en el sistema operativo MULTICS.

Unix es el niño prodigio de Ken Thompson y Dennis Ritchie, dos investigadores de los laboratorios Bell. Al mismo tiempo, Ken Thompson trabajó en un programa de simulación del movimiento de los planetas en el sistema solar llamado *Space Travel*. El programa estaba bajo un sistema operativo llamado Multics, uno de los primeros sistemas operativos que proporcionaba un entorno multiusuario, y se ejecutaba en una computadora General Electric de la serie 6000. Multics era grande, lento y requería recursos esenciales de la computadora. Thompson encontró una computadora más pequeña a la cual transfirió el programa Space Travel para ejecutarlo. La computadora era una máquina poco utilizada, la PDP-7, construida por Digital Equipment Corporation (DEC). En dicha computadora Thompson creó un sistema operativo que llamó Unix, y a ese sistema operativo adaptó algunos de los conceptos avanzados de Multics. Existían ya otros sistemas operativos distintos de Multics que tenían más o menos las mismas capacidades y Unix se aprovechó del trabajo realizado en aquellos sistemas operativos, al combinar algunos de los aspectos más deseables de cada uno de ellos.

Unix se transfirió en 1970 a una computadora PDP-11/20 y posteriormente a la PDP-11/40, PDP-11/45 y finalmente a la PDP-11/70. Cada una de estas máquinas tenían características que gradualmente se añadían a la complejidad del hardware que Unix podía soportar. Dennis Ritchie y otros en los Labortarios Bell continuaron el proceso de desarrollo de Unix incorporando utilidades (tales como un procesador de texto).

Como la mayoría de los sistemas operativos, Unix fue originalmente escrito en lenguaje ensamblador, lo cual lo hacía dependiente de la computadora donde se ejecutaba. Por lo que transferir Unix de una computadora a otra requería una reescritura importante de los programas.

Thompson y Ritchie eran usuarios experimentados de Multics, que fue escrito en un lenguaje de alto nivel llamado PL/1 y eran conocedores de las ventajas de utilizar un lenguaje de alto nivel para escribir sistemas operativos. Es por esto que decidieron reescribir Unix en un lenguaje de alto

nivel. El lenguaje que eligieron era C y en 1973 Ken y Dennis reescribieron satisfactoriamente Unix en C[1].

Las universidades y colegios han jugado un papel importante en la popularidad del sistema operativo Unix. En 1975 los Laboratorios Bell ofrecieron, a un costo mínimo, el sistema operativo Unix a las instituciones educativas.

El objetivo original no era producir un sistema operativo, sino crear un ambiente de trabajo en el cual pudieran proseguir con su objetivo principal: la investigación en un área determinada. La palabra Unix viene de una deformación a través del tiempo de lo que es la palabra Unics. Esta última es una parodia del nombre del sistema operativo MULTICS que significaba MULtiprocessing Computer System, (Unics vendría a significar UNIprocessing Computer System).

2 Los diferentes sistemas Unix

La estandarización de Unix se ha convertido en un tema cada vez más debatido. Parece poco probable que en el futuro surja una norma Unix única. AT&T continua promoviendo su versión llamada Unix System V, muy utilizada en la industria. Por otro lado, las universidades siguen promoviendo la versión Unix de Berkeley, el cual es un derivado de la versión de AT&T. La comunidad Unix ha cooperado en el desarrollo de una especificación estandarizada del sistema denominada POSIX, que consiste de un subconjunto común de los principales sistemas Unix. La fundación de software abierto se constituyó para producir una versión de Unix basada, en gran medida, en la versión AIX de IBM. Pasarán muchos años antes de que aparezca un solo Unix estandarizado, si es que se consigue alguna vez. Tal vez no exista un diseño de sistemas operativos capaz de satisfacer las diversas necesidades de la comunidad informática mundial.

El origen de los diferentes sistemas Unix tiene su raíz en el nacimiento, en 1975, de la versión 6 de los laboratorios Bell de AT&T. Después de la presentación de esta versión surgen dos líneas diferentes conocidas como Sistema V y BSD.

Los desarrolladores de la Universidad de California en Berkeley (de ahí el nombre de BSD) han agrandado Unix de diferentes formas añadiendo un mecanismo de memoria virtual, el shell C, el control de tareas, la red TCP/IP, por nombrar solo un pequeño número. Algunos de estos nuevos mecanismos fueron introducidos en las líneas de código de AT&T.

El sistema V versión 4 es presentado como la fusión del Sistema V y de BSD, pero eso no es completamente exacto. El sistema V Versión 4 resulta de la incorporación de las funciones más importantes de BSD y de SunOS en el seno de Sistema V. Esta unión puede ser vista como una unión más que como una fusión, en la cual algunas características de cada uno son heredadas (a las cuales se debe añadir características cuyo origen es incierto).

La proliferación de constructores informáticos en el curso de los años 80's provocó la aparición en el mercado de decenas de nuevos sistemas Unix. Unix fue escogido por su bajo costo y por sus características técnicas, pero también a causa de la ausencia de otras opciones. Estos proveedores se basaron en versiones de BSD o sistema V aportando modificaciones menores o más importantes. La mayor parte de las versiones de Unix que aún subsisten provienen del sistema V versión 3 (en general versión 3.2), sistema V versión 4 y algunas veces de BSD 4.2 o 4.3 (SunOS es una excepción ya que tiene su origen en una versión más antigua de BSD). Para complicar las cosas, varios proveedores han mezclado características de BSD y del Sistema V en el corazón de un solo sistema operativo.

2.1 El sistema SCO Unix

En 1983 SCO, Santa Cruz Operations, lanza un Unix bajo el nombre de SCO XENIX System V para PCs basadas en procesadores Inter 8086 y 8088. En 1995 SCO adquieren la división de Unix Systems de la compañía Novell, (que a su vez la adquirió de AT&T). SCO comerciliza Unix System V bajo un producto denomindao UnixWare, que por algún tiempo se llamó OpenUnix. La última version de SCO UnixWare es 7.1.4, la cual sale en el 2006 y en junio del 2008 se libera el último paquete de mantenimiento.

[1] Aproximandamente un 95% de Unix está escrito en C, una parte muy pequeña está todavía escrita en lenguaje ensamblador, esa parte se encuentra concentrada en el núcleo, la parte que interacciona directamente con el hardware.

2.2 El sistema SunOS

Desarollo por Sun Microsystems para sus estaciones de trabajo y servidores. El nombre hace referencia a las versiones 1.0 a 4.1.4, que se basan en BSD. Las versiones 5.0 y supersiones se basan en System V Release C. Bajo SunOS se han incorporado funcionalidades importantes a Unix, entre la más importante esta NFS y NIS. La versión 4.1.2 soportaba la primera arquitectura Sun multiprocesador, (las series SPARC Sever 600MP). Las últimas versiones basadas en BSD, 4.1.3 y 4.1.4, fueron embarcadas en diciembre 1998 y se les dio soporte hasta septiembre del 2003.

2.3 El sistema Solaris

Es una implementación del sistema V.4 propuesto por Sun Microsystems en 1992 para reemplazar a SunnOS. Su desarrollo ha ido de la mano con el hardware de Sun SPARC, (incluyendo el soporte para aplicaciones SPARC de 64 bits en Solaris 7). Sun continua proporcionando los dos sistemas operativos. Hay que mencionar que las versiones de Solaris cuantan con un equivalente en las versiones de SunOS, por ejemplo la versión 10 de Solaris es la 5.10 de SunOS. La versión más actual (enero 2005) de Solaris es la 10.

2.4 El sistema HP-UX

Es la versión de Unix desarrollada y mantenida de Hewlett-Packard desde 1983. Sigue las características del Sistema V incorporando varias características de OSF/1[2]. HP-UX ha sido considerablemente modificado entre las versiones 9 y 10. Desde el punto de vista de la administración, HP-UX 9 se parece al sistema V.3 con algunas extensiones, por otro lado HP-UX 10 se asemeja a un sistema operativo del tipo V.4. En la actualidad la última versin de este sistema operativo es la 11.23, también conocido como 11iv3 (2006). Apartir de la versin 11.11 (2000) se usa un sistema de numeración doble, así la 11.11 es también conocida como 11i, la 11.20 es 11iv1.5 y así sucesivamente

2.5 El sistema IRIX

Es la versión de Unix creada por SGI (Silicon Graphics) para su plataforma MIPS de 64 bits en 1988. Las primeras versiones de IRIX incorporan numerosas características de BSD pero estas han desaparecido en el transcurso del tiempo a favor de una conformidad al sistema V.4. El 6 de septiembre de 2006, SGI anunció el fin de los productos IRIX/MIPS.

2.6 El sistema AIX

El sistema operativo de IBM de tipo Sistema V. En un principio Inicialmente significaba "Advanced IBM Unix" pero fue cambiado a "Advanced Interactive eXecutive". Han existido distintas versiones de AIX, algunas de las cuales ya no son soportadas. AIX V1, que corra en la IBM RT/PC (AIX/RT) apareci en 1986, y estaba basada en un System V Release 3. Desde 1989, AIX ha sido el sistema operativo para las estaciones de trabajo y servidores RS/6000 (AIX/6000). Durante el desarrollo de AIX, se integraron caractersticas del 4.2BSD y el 4.3BSD por parte de IBM y el Interactive Systems Corporation (bajo contrato con IBM). La version más actual (noviembre 2007) es la AIX 6.1.

2.7 El sistema OSF/1

En 1988, Sun y AT&T se pusieron de acuerdo para desarrollar juntos las futuras versiones del sistema V, decidieron integrar las características principales de BSD y SunOS dentro del sistema V para crear System V, release 4 (SVR4). En respuesta, IBM, DEC, Hewlett-Packard así como otros constructores y sociedades informáticas fundaron la OSF (*Open Software Foundation*) cuyo objetivo era la concepción de otro sistema operativo compatible con Unix y, sobre todo, independiente de AT&T. OSF/1 es el resultado de este esfuerzo, aunque OSF/1 constituye más una definición de estandares que una implementación real.

[2]OSF: Open Software Foundation.

Entre los estándares más importantes se encuentran POSIX (definido por IEEE/ANSI), el AT&T System V Interface Definition (SVID), la Application Environment Specification (AES) de la OSF y el X/Open Portability Guide de la X/Open, un consorcio fundado en Gran Bretaña en 1984.

2.8 El sistema DEC OSF/1

Con el objetivo de reemplazar su sistema operativo ULTRIX, la gente de DEC (Digital Equipment Corporation) desarrolla una nueva implementación de Unix basado en la especificación OSF/1. Inicialmente se mercadeo como DEC OSF/1. pero después DEC renombro al sistema operativo como Digital UNIX. Cuando Compaq adquirió a DEC, el sistema volvió a cambiar de nombre: Tru64 UNIX.

Se trata de un sistema operativo de 64 bits para estaciones de trabajo y servidores equipados con un procesador Alpha. Se comporta en como un sistema BSD desde el punto de vista de la administración del sistema, aunque en el fondo se trata de un Sistema V. HP-UX y DEC OSF/1 claman su conformidad a un conjunto de estandares practicamente identicos pero estas versiones deben ser administradas de forma diferente.

La última versión (2008) es la 5.1B-3. Se planea llegar hasta la 5.1B-6 en el 2011 y el soporte se dara hasta el 2012.

2.9 El sistema XENIX

Xenix es la primera versión de Unix diseñada para microcomputadoras, aún es utilizada. En 1979 Microsoft compró una licencia de la empresa AT&T con el objetivo de adaptar Unix a procesadores de 16 bits. Microsoft cedio XENIX a SCO, quien lo adapto para microprocesadores 80286 en 1985 y después para el 80386 cambiandole el nombre a SCO UNIX.

Microsoft no vendia XENIX directamente al usuario, sino que vendía licencias a los fabricantes de computadoras que deseaban usarlo en sus equipos. Entre las compañıas que adquirieron XENIX estan Intel, Tandy, Altos y SCO.

Esta versión proviene de la versión 7 y ha sido convertido progresivamente en un sistema V versión 2. XENIX influenció Sistema V versión 3, la mayor parte de sus funciones fueron incorparados en el Sistema V versión 3.2

Es posible encontrar versiones de este sistema operativo, pero solo se puede instalar sobre maquinas 288

2.10 El sistema Linux

Linux es un clon de Unix en el dominio público destinado a los procesadores Intel. Linux ha ganado en popularidad regularmente y es muy útil en varias situaciones: es un sistema Unix poco costoso que puede constituir un ambiente de investigación para los colegios y universidades, una solución económica para contar con una conexión Internet para las empresas pequeñas, un sistema Unix doméstico para los profesionales y una terminal X barata para los sitios Unix con presupuesto reducido.

El núcleo fue desarrollado por Linus Torvalds, (Linux es el Unix de Linus, Linus Unix) aunque otras personas han contribuido (y contribuyen) a su desarrollo. Linux es globalmente de tipo BSD. Técnicamente, el nombre de Linux hace referencia al corazón del sistema operativo (el núcleo y algunos controladores de periféricos) pero el nombre también se aplica al software de dominio público, donde las fuentes son de origen variado, que constituyen una *distribución*. Por otro lado, Linux es el núcleo del sistema operativo desarrollado por la gente de GNU. Desde esta perpesctiva, el nombre correcto de Linux es GNU Linux.

Hay que considerar que Linux no cuenta con una sola línea de código del sistema Unix original. Los comandos son los mismos que en Unix y de ahi que mucha gente lo considere como un Unix.

2.11 El sistema Minix

Es un sistema operativo desarrollado en 1987 por Andrew Tanenbaum con fines pedagógicos. Pensado en un principio para ser ejecutado a partir de discos flexibles, en una PC compatible. El sistema estaba incluido como parte del libro de Operating Systems: Design and Implementation. En el libro se dedicaba la mitad del espacio al código del sistema operativo.

La última versión, mayo 2006, es la 3.12 y soporta sólo arquitecturas derivadas de IA-32, y está disponible en LiveCD y en versiones compatibles con máquinas virtuales como BOCHS, Qemu, VMware y VirtualPC. Minix 3 esta disponible de forma gratuita y libre en sus página oficial www.minix3.org.

Minix fue la fuente de inspiración de Linus para desarrollar el sistema operativo Linux. De acuerdo a Tanenbaum, el constante rechazo para añadir nuevas propiedades al sistema operativo, fue lo que motivó a Linus a crear Linux. Tabenbaum quería mantener a Linux lo suficientemente pequeño para que sus alumnos lo pudieran entender en un semestre.

2.12 El sistema FreeBSD

FreeBSD es un sistema operativo Unix BSD avanzado para arquitecturas Intel (x86), DEC Alpha y PC-98. El soporte y desarrollo es proporcionado por un gran equipo de personas repartidas en todo el mundo. FrereeBSD es un derivado de BSD, la versin de UNIX desarrollada en la Universidad de California, Berkeley.

Se puede instalar FreeBSD desde una gran variedad de soportes, incluyendo CD-ROM, DVD-ROM, cintas magnticas, una particin MS-DOS, o si se dispone de conexin de red, se puede instalar directamente mediante FTP annimo o NFS.

FreeBSD es libre y gratuito. Está disponible completamente gratis incluyendo el código fuente en la página www.freebsd.org. La versión actual, febrero 2008, es la Free BSD 7.0.

2.13 El sistema OpenBSD

El proyecto OpenBSD produce una multiplataforma libre del sistema operativo Unix 4.4 BSD. Los esfuerzos de los integrantes del proyecto van dirigidos a reforzar la portabilidad, estandarización, seguridad, exactitud e integración de criptografía. OpenBSD soporta emulación binaria de la mayoría de los programas de Solaris SVR4, FreeBSD, Linux, BSD/OS, SunOS y HP-UX.

El sistema se puede se puede obtener sin cargo alguno desde nuestros servidores de FTP, y también se puede adquirir en un juego de 3 CD de bajo coste. La versión actual es OpenBSD 3.6, que fue liberado el 29 de octubre del 2004. Se puede obtener de la página www.openbsd.org

El desarrollo de OpenBSD est a cargo de voluntarios. Los fondos para el desarrollo del sistema y para el lanzamiento de nuevas versiones provienen de la venta de los CD y camisetas, as como de donaciones.

2.14 El sistema BSD/OS

El núcleo de BSD/OS está inspirado en el núcleo del sistema operativo 4.4 BSD de la Universidad de California Berkeley, con mejorías de BSD. Es una plataforma de red cliente/servidor rápida, escalable y que soporta multitareas. Cuenta con una pequeña huella, memoria virtual (opcional) y memoria de protección, con soporte para 768 Mbytes de RAM hasta 3.75 Gbytes de memoria virtual para el usuario. BSD/OS tiene un buen rendimiento en sistemas equipados con un poco más de 2 Mbytes de RAM.

3 Características principales del sistema Unix

El sistema operativo Unix es un sistema que presenta un par de características conocidas como *multiprogramación* y *tiempo compartido*. La primera de ellas permite que varios trabajos se efectuen al mismo tiempo y gracias a la segunda varias personas pueden estar dentro del sistema al mismo tiempo realizando actividades diferentes.

El sistema está constituido por tres partes, el núcleo, el shell y los programas.

El núcleo es la parte medular de Unix. Es el encargado de asignar tiempo y memoria a los programas y manejar las comunicaciones para responder a las peticiones que realice el usuario.

Un diagrama que presenta los principales componentes del núcleo de Unix se presenta en la figura 1

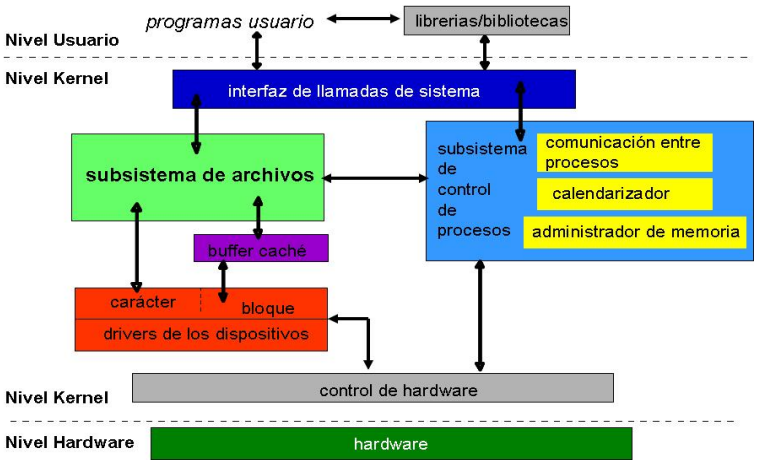

Figure 1: *Diagrama del núcleo de Unix*

El shell se compone de una línea de comandos y un prompt. El shell es el encargado de interpretar las instrucciones del usuario y, si es posible, llevarlos a cabo. En caso de no ser posible despliega, un mensaje de error.

Existem diferentes tipos de shell, los más importantes se encuentran descritos en la tabla de abajo

Shell	**Nombre**	**Descripción**
`sh`	Bourne Shell	Escripto por Bourne Shell en Bell
`ash`	Almquist shell	Reemplazo con licencia BSD del Bourne Shell
`bash`	Bourne-Again Shell	El shell de GNU Linux
`dash`	Debian Almquist Shell	Reemplazo ash en Debian
`ksh`	Korn Shell	Escrito por David Korn en Bell
`zsh`	Z shell	Considerado el shall más completo
`csh`	C shell	Escrito por Bill Joy en Berkeley
`tcsh`	TENEX C Shell	Extensión de CShell

Los programas constituyen lo que se conoce como comandos. Es a través de estos comandos que el usuario le indica al sistema lo que desea realizar.

Una forma de ilustrar como trabajan los diferentes componentes del sistema operativo es a través del siguiente ejemplo. Supongamos que un usuario desea borrar el archivo `toto`, dicho usuario sabe que el programa (comando) `rm` permite borrar archivos. Usando el shell, el usuario introduce el comando (`rm toto`). El shell busca el lugar donde se encuentra el archivo `rm` que contiene el código para borrar un archivo. Una vez que lo encuentra lo ejecuta. A través de funciones especiales dentro del código (conocidas como llamadas de sistema) se le transmiten peticiones al núcleo. El núcleo es el encargado de borrar el archivo `toto`. Cuando el programa `rm` termina de correr, el shell se pone en un estado de escucha esperando que el usuario teclee más comandos.

El presente documento está enfocado a los dos últimos componentes. Es decir, por un lado se explica todo lo relacionado con el shell y por el otro se da una lista de los comandos más importantes del sistema operativo Unix.

3.1 Entrando al sistema

Para que una persona (conocida como usuario) pueda tener acceso al sistema es necesario que se identifique con él. Esta identificación se realiza proporcionando al sistema un nombre (conocido como cuenta o login) y una contraseña (conocida como password). Este proceso se conoce con el nombre de firmarse con el sistema.

El nombre de la cuenta debe contar con ocho caracteres como máximo y es creado por el administrador del sistema. Este puede consistir en el apellido, nombre, o una clave asociada con el usuario (p.e. apellido del usario). La contraseña o password también es creada por el administrador del sistema y puede llegar a ser cambiada por el usuario, aunque muchos sistemas no lo permiten por razones de seguridad. La contraseña esta formada por al menos seis caracteres, (de los cuales al menos dos caracteres deben de ser diferentes a letras)

El sistema pregunta al usuario su cuenta a través del mensaje `login:` y, una vez tecleada esta, pregunta la contraseña desplegando `password:`. Cuando el usuario teclea su password no se distingue ninguno de los caracteres tecleados en la pantalla. Un ejemplo de esto es:

```
login: rogomez
password:
```

Si hay algun error, ya sea que hubo un error al introducir la cuenta o el password (o que el password fue cambiado y no coincide con la cuenta) se imprime un mensaje de error. Por ejemplo:

```
login: rogomez
password: *****
login incorrect
login:
```

Es importante remarcar que Unix no indica si el error estuvo al introducir la cuenta, el password o los dos, simplemente despliega un mensaje de error y el usuario debe intentar introducir sus datos de nuevo. En algunos sistemas, si al tercer intento el sistema sigue negando el acceso al usuario la máquina se apaga o el sistema se desactiva.

Si todo pasa bien, aparece el *prompt*, el cual indica el principio de la línea de comandos. Es a través de los comandos introducidos en esta línea que el usuario le va a indicar al sistema lo que desea hacer.

La mayor parte de los sistemas unix de hoy en día cuentan con un sistema de ventanas, por lo que una vez que el usuario se firma con el sistema este no presentara un prompt si no una pantalla al estilo un sistema Microsoft windows. Es posible interactuar con el sistema con el raton, al estilo Windows. Sin embargo si uno desea aprovechar al máximo el sistema, se aconseja interactuar con él a través de un terminal virtual la cual presentará el prompt. Un ejemplo de terminal virtual se presenta en la figura 2.

3.2 La línea de comandos

La línea de comandos empieza en el prompt y termina en el momento en que el usurio presiona la tecla `<RETURN>`. La primera palabra que se introduce en la línea de comandos es el nombre de un archivo ejecutable, o de un comando del sistema.

La línea de comandos forma parte de lo que se conoce como *shell*. El shell es el encargado de leer el comando y ejecutarlo. Existen una gran variedad de shells, entre los más comunes encontramos el bourne-shell (el primer shell), el c-shell, el tc-shell, el korn-shell y bash (ver tabla de arriba).

La línea de comandos empieza en el *prompt*. Por defecto el prompt es representado por un caracter aunque esto puede ser modificado por el usuario. Por ejemplo el prompt por defecto del bourne-shell es el caracter `$` y del c-shell es el caracter `%`. En este documento se utiliza el prompt del shell del autor el cual es de la forma: `rogomez@armagnac:89>` donde se despliega la cuenta, la máquina y el número de instrucción.

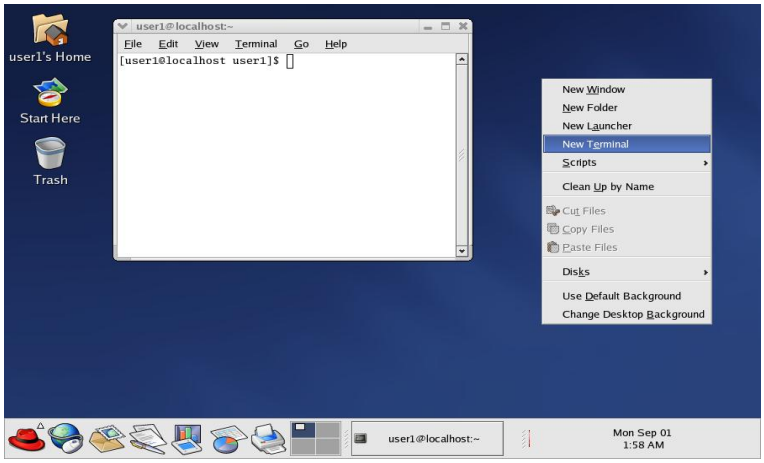

Figure 2: *Ejemplo entrada linux*

El comando puede ser seguido por una o más opciones, y/o uno o más argumentos, (separados por espacios o tabulaciones). El comando junto con sus opciones y/o argumentos, no es ejecutado antes del `<RETURN>`. Una vez que el comando termina su ejecución aparece el prompt de nuevo para indicar que el sistema está listo para ejecutar otro comando.

Para poder capturar una línea de comando muy larga, es posible insertar el carácter `\` al final de la primera línea, después de teclear `<RETURN>`, para poder teclear el resto de la línea de comando en una segunda línea de la pantalla.

Ejemplo:

```
rogomez@armagnac:2>~/bin/xvile articulo.tex -display \
  rogomez:0.0
rogomez@armagnac:3>
```

3.3 Sintaxis de los comandos UNIX

Como se dijo en la sección anterior, varios comandos cuentan con opciones y/o argumentos, para utilizarlos es necesario dejar un espacio:

- entre el nombre del comando y las opciones y/o los argumentos
- entre las opciones y los argumentos
- entre los argumentos

Un comando tiene opciones por defecto, si se quieren utilizar estas opciones se debe teclear:

```
rogomez@armagnac:4>nombre-comando <RETURN>
```

En caso contrario, los siguientes formatos son posibles:

1. `nombre-comando  argumento(s) <RETURN>`
2. `nombre-comando  opcion(es) <RETURN>`
3. `nombre-comando  opcion(es) argumento(s)  <RETURN>`

En general se puede decir que el comando le indica al sistema qué hacer, las opciones cómo hacerlo y los argumentos sobre quién hacerlo.

3.3.1 El comando: ¿qué hacer?

El comando es la primera palabra de la línea de comandos y siempre corresponde al nombre de un archivo ejecutable.

Por ejemplo:

```
rogomez@armagnac:4> ls
rogomez@armagnac:5> who
rogomez@armagnac:6> ps
```

3.3.2 Las opciones: ¿cómo hacerlo?

Un comando puede realizar diferentes tareas, o presentar resultados en diferentes formatos de acuerdo a sus opciones. Las opciones siguen al comando (separadas por un espacio) y le indican al sistema con cual opción se debe ejecutar el comando. En caso de no proporcinar opció se toma aquella por defecto. Generalmente las opciones están precedidas de un carácter `-` (o a veces de un caracter `+`).

Por ejemplo:

```
rogomez@armagnac:7> ls - l
rogomez@armagnac:8> date +%d%m%y
```

3.3.3 Los argumentos: ¿sobre quién actuar?

Generalmente se refieren a uno o varios nombres de archivo sobre los cuales el comando será ejecutado.

```
rogomez@armagnac:9> cat capitulo
rogomez@armagnac:10> cp archivo nuevo
rogomez@armagnac:11> ls -l tarea*
```

3.4 Comandos en minúsculas y MAYUSCULAS

Es muy importante remarcar que Unix, a diferencia de otros sistemas operativos, hace diferencia entre letras MAYÚSCULAS y minúsculas en los nombres de los comandos. Un comando constuido exclusivamente de letras minusculas no sera reconocido si alguna de estas letras es mayuscula. Es decir, no es lo mismo:

```
rogomez@armagnac:12>cd /bin
```

que:

```
rogomez@armagnac:13>CD /BIN
CD: Command not found
```

En el primer caso se hará lo que el comando indique, mientras que el segundo no será reconocido por el sistema y desplegará el mensaje de error correspondiente a este hecho: `Comando no encontrado`.

El mismo mensaje es desplegado si el comando no existe, o si se introducen carateres al azar sin significado alguno para el sistema.

3.5 Variantes en la ejecución de un comando

Se define ejecución de un comando a todo el trabajo que realiza dicho comando para satisfacer lo solicitado por el usuario. Existen varias formas en que esta ejecución puede llevarse a cabo. A continuación se explicarán algunas de las más comunes.

Cuando el usuario introduce un comando, el shell lo ejecuta sin indicar nada. Una de las características de Unix es que el sistema no indica como salió todo, o si ya terminó. El usuario se percata

que el comando terminó de ejcutarse porque el prompt aparece de nuevo preguntandole al usuario por un nuevo comando. Si existe algún error en la ejecución del comando, el shell desplegará un mensaje de error. La mayor parte de los comandos cuentan con una opción (verb—-v—, verbose) para que el shell indique lo que esta haciendo durante su ejecución.

3.5.1 Redirección de las entradas/salidas estándares

El resultado de la ejecución de un comando aparece en la salida estándar (la pantalla), mientras que los datos (y el comando mismo) son leídos de la entrada estándar (el teclado). Unix permite redireccionar las entradas/salidas estándar a partir de delimitadores angulares:

- `<` redirección de la entrada estándar.
- `>` redirección de la salida estándar (creación)
- `>>` redirección de la salida estándar (añadir)

Por redirección de salida estándar se entiende que en lugar de desplegar los resultados en pantalla, el sistema los envía a un archivo. La redirección de entrada estándar provoca que, en lugar de obtener los datos del teclado, se lean de un archivo.

Un ejemplo de redirección de la entrada estándar es:

```
rogomez@aramagnac:14>mail profesor <  tarea.txt
```

En este caso la entrada estándar del comando `mail` es substituida por el contenido del archivo `tarea.txt`.

Un ejemplo de redirección de la salida estándar (creación) se presenta a continuación:

```
rogomez@armagnac:15>cat arch1 arch2 > final.txt
```

La salida estándar del comando `cat` es redirigida al archivo `final.txt`. Esto trae como consecuencia que los contenidos de los archivos `arch1` y `arch2` sean copiados uno después del otro en el archivo `final.txt`. En la mayoría de los sistemas si este archivo ya existe, el sistema desplegará un mensaje de error; por ejemplo:

```
rogomez@armagnac:16>ls > sal
sal: File exists.+.
rogomez@armagnac:17>
```

El siguiente es un ejemplo de redirección de salida estándar utilizando los caracteres `>>`:

```
rogomez@armagnac:17>echo  ERRORES DE COPIA >> log
```

La salida estándar del comando `echo` será el archivo `log`. Dependiendo del tipo de sistema Unix y shell utilizado, si el archivo no existe, éste será creado. Si el archivo ya existe, se añadirá el resultado del comando echo al final del archivo.

3.5.2 Ejecución en background

Para los comandos lentos en su ejecución, resulta interesante poder disponer de la terminal de tal forma que se puedan ejecutar otros comandos.

Poniendo un `&` después del comando y de sus opciones y/o argumentos, el sistema ejecutará el comando en *background*, desplegando el prompt de nuevo y dejando al sistema listo para leer otro comando.

Por ejemplo:

```
rogomez@armagnac:18> netscape tareas.html -display walhalla: 0.0  &
[1] 712
rogomez@armagnac:19>
```

ejecutará el comando `netscape` con todas sus opciones y argumentos en background. El número 1 dentro de los corchetes es el número de trabajo (o job) asignado por el sistema y el 712 es el identificador del proceso que se encarga de dicho trabajo.

Es importante remarcar que el resultado de la ejecución de estos comandos será desplegado en la misma pantalla donde se ejecutó el comando.

3.5.3 Agrupación de comandos

Si se agrupan varios comandos entre paréntesis `( )`, estos seran considerados como una sola unidad.

Por ejemplo, los siguientes comandos:

```
rogomez@armagnac:19> echo El dia de hoy:   > log
rogomez@armagnac:20> date >> log
rogomez@armagnac:21> echo las personas siguientes >> log
rogomez@armagnac:22> who >> log
rogomez@armagnac:23> echo se encuentran conectadas >> log
rogomez@armagnac:24>
```

pueden agruparse en uno solo:

```
rogomez@armagnac:24> ( echo El dia de hoy;  date; echo las personas \
  siguientes;  who;  echo se encuentran conectadas ) > log
rogomez@armagnac:25>
```

3.5.4 Ejecutando comandos secuenciales

Es posible teclear diferentes comandos sobre la misma línea de comandos, separandolos por punto y comas (`;`).

En este caso los comandos son ejecutados secuencialmente, es decir que el segundo comando es ejecutado después de que el primero terminó su ejecución. Por ejemplo la siguiente secuencia de instrucciones:

```
rogomez@armagnac:25> date
Wed Oct 12 10:44:16 MET 1986
rogomez@armagnac:26> ls -C
prueba  archivo
rogomez@armagnac:27> who
rogomez console Oct 12 09:09
rogomez ttyp0 Oct 12 10:38
toto ttyp1 Oct 12 11:08
rogomez@armagnac:28>
```

se pudo haber tecleado como:

```
rogomez@armagnac:28> date;   ls -C;    who
Wed Oct 12 10:44:16 MET 1986
prueba  archivo
rogomez console Oct 12 09:09
rogomez ttyp0 Oct 12 10:38
toto ttyp1 Oct 12 11:08
rogomez@armagnac:29>
```

3.5.5 Pipelines

En algunas ocasiones es importante que el resultado de la ejecución de un comando sea la entrada de otro. Una opción para resolver lo anterior es utilizar redirecciones, la salida del comando se envia a un archivo y la entrada del otro comando redireccionarla con respecto a dicho archivo.

La salida estándar de un comando puede ser conectada a la entrada estándar de otro comando a través de lo que se conoce como pipelines. Un pipeline es un puente de comunicación entre la salida de un proceso y la entrada de otro. Es representado por una línea vertical `|`. La sintaxis del pipeline es:

```
comando [ ] [ ]  | comando [ ] [ ]  |  comando  [ ] [ ]
```

Un ejemplo de uso del pipeline es el siguiente:

```
rogomez@armagnac:28>cat numeros
uno un
dos deux
tres trois
cuatro quatre
cinco cinq
rogomez@armagnac:29>cat numeros  |  grep dos  |  more
dos deux
rogomez@armagnac:30>
```

En este caso la salida del comando `cat` es la entrada del comando `grep` y la salida de este es la entrada del comando `more` el cual al final lo imprime en pantalla.

En realidad una de las ventajas de los pipelines es el evitar la creación de archivos temporales para dejar resultados parciales en ellos. En efecto, a través de redirecciones y con archivos temporales es posible obtener el mismo resultado. Esto se le deja como ejercicio al lector.

A continuación se numeran algunos de los comandos básicos de Unix que se deben conocer. La mayor parte de los comandos cuenta con varias opciones, y tan solo de enumeran algunas de ellas.

4 El sistema de archivos de Unix

Una de las partes fundamentales del sistema operativo Unix son los archivos. Todo se hace a través de ellos. Los archivos se encuentran agrupados en como *directorios*. Estos directorias se encuentran organizados en una jerarquía de árbol, donde la raíz está representada por el caracter \ (ver figura 3).

La información almacenada dentro de cada directorio esta organizada de acuerdo al FHS (Filesystem Hierarchy Standard) que define la forma oficial de organizar los archivos en directorios Linux. Los directorios organizan archivos usuarios, núcleos, bitácoras, programas, utilerías y demas información dentro de diferentes categoras. La descripción de algunos del tipo de archivos que se encuentran en algunos directorios se encuentra en la tabla de abajo. Es importante tomar en cuenta que un usuario y/o administrador puede colocar los archivos donde el desee, el sistema no se lo impedirá. Sin embargo, algunas aplicaciones y/o utilerías pueden no funcionar correctamente si los archiviso o directorios no se encuentran ubicados donde el estándar FHS lo indica.

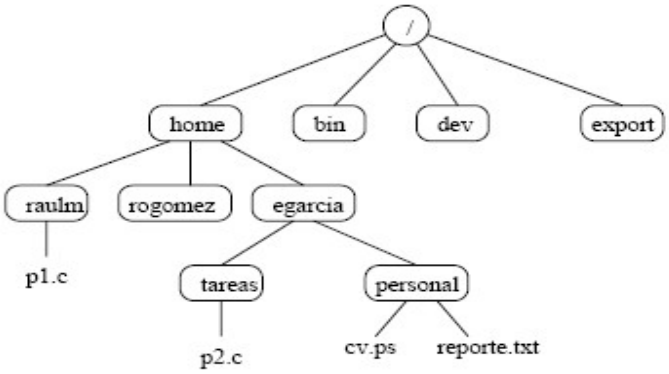

Figure 3: *Ejemplo arborencia linux*

Directorio	Descripción
`/`	directorio raíz
`/bin`	utilerías esenciales a nivel comando
`/boot`	archivos de arranque
`/dev`	drivers de dispositivos
`/etc`	la mayoría de los archivos de configuración
`/home`	directorios hogar para la mayor parte de los usuarios
`/lib`	librerías/bibliotecas del núcleo y varios comandos de línea
`/mnt`	punto de montaje para dispositivos almacenamiento removibles
`/opt`	aplicaciones como WordPerfect, OpenOffice
`/proc`	información sobre status máquina y procesos ejecutandosé
`/root`	directorio hogar para root
`/sbin`	comandos del administrador de sistemas
`/tmp`	archivos temporales
`/usr`	programas pequeños accesibles a todos los usuarios
`/var`	spools de la impresora y bitácoras

El directorio hogar (home) es el directorio donde se situa al usuario cuando este se firma con el sistema. Generalmente se encuentra debajo del directorio `/home/` y tiene el nombre del usuario. Si el usario tiene la cuenta `rogomez`, el directorio hogar es `/home/rogomez`

Los comandos que se describen en esta sección permiten ver el contenido de los archivos, borrarlos, moverlos, renombrarlos, listar el contenido de un directorio y cambiar de directorio.

4.1 El comando *pwd*, print working directory

Descripción: despliega el camino/ruta de acceso (path) del directorio actual, i.e. donde se encuentra dentro del sistema de archivos. Este comando despliega el nombre de un directorio nunca el de un archivo.

Sintaxis:

```
pwd
```

Ejemplo:

```
rogomez@armagnac:50>pwd
/home/dic/rogomez/Articulos
rogomez@armagnac:51>
```

Nota: existen dos tipos de rutas de acceso

- ABSOLUTA es necesario especificar todos los directorios para referenciar un archivo, por ejemplo: `/home/toto/perso/datos.txt`
- RELATIVA dependiendo de donde se encuentre el archivo, tan solo se dan a conocer algunos directorios, por ejemplo si el usuario se encuentra en el directorio `perso` solo necesita el archivo, en este caso `datos.txt`.

4.2 El comando *cd*

Descripción: permite cambiar de directorio. Una vez realizado el cambio despliega el directorio a donde se cambio. Sin parámetro alguno lo posiciona en el directorio donde inicio el usuario (directorio hogar), cuando se entro al sistema. Dando como parámetro `..` remonta en el directorio del padre.

Sintaxis:

```
cd [ nombre-directorio ]
```

Parámetros especiales:

`.` directorio actual

`..` directorio padre

`~` directorio hogar

sin argumentos regresa al directorio hogar

Ejemplo:

```
rogomez@armagnac:53>cd Cursos/
/home/rogomez/Cursos
rogomez@armagnac:54>cd ..
/home/rogomez
rogomez@armagnac:55>
```

4.3 El comando *ls*

Descripción: despliega los nombres de los archivos que se encuentran dentro del directorio actual

Sintaxis:

```
ls [opciones]
```

Algunas opciones:

`-a` lista los archivos *ocultos*, es decir aquellos que comienzan con un "."

`-l` listado en formato largo (ver adelante)

`-d` si el argumento es un directorio lista el nombre del archivo directorio y no su contenido

`-F` despliega un caracter al lado del archivo para identificar el tipo de archivo (`*` ejecutable, `/` directorio, `@` liga simbólica, y `=` sockets)

`-r` lista los archivos en el orden inverso

`-t` despliega los archivos en función de la hora de la última modificación

Ejemplo opciones comandos ls:

```
rogomez@armagnac:56>ls
a1  a2  colores  D1  hola  numeros  recibe.c
rogomez@armagnac:57>ls -r
recibe.c  numeros  hola  D1  colores  a2  a1
rogomez@armagnac:58>ls -a
.  ..  a1  a2  colores  D1  .hidden  hola  numeros  .oculto  recibe.c
```

```
rogomez@armagnac:59>ls -t
D1  a2  a1  numeros  colores  hola  recibe.c
rogomez@armagnac:60>ls -l
total 36
-rw-rw-r--    1 rogomez  academicos         8 Aug 14 13:01 a1
-rw-rw-r--    1 rogomez  academicos         8 Aug 14 13:01 a2
-rw-rw-r--    1 rogomez  academicos        42 Aug 14 13:00 colores
drwxrwxr-x    2 rogomez  academicos      4096 Aug 14 13:02 D1
-rwxr-xr-x    1 rogomez  academicos     11541 Aug 14 13:00 hola
-rw-r--r--    1 rogomez  academicos       442 Aug 14 13:00 numeros
-rw-r--r--    1 rogomez  academicos      1213 Aug 14 13:00 recibe.c
rogomez@armagnac:61>ls -lt
total 36
drwxrwxr-x    2 rogomez  academicos      4096 Aug 14 13:02 D1
-rw-rw-r--    1 rogomez  academicos         8 Aug 14 13:01 a2
-rw-rw-r--    1 rogomez  academicos         8 Aug 14 13:01 a1
-rw-r--r--    1 rogomez  academicos       442 Aug 14 13:00 numeros
-rw-rw-r--    1 rogomez  academicos        42 Aug 14 13:00 colores
-rwxr-xr-x    1 rogomez  academicos     11541 Aug 14 13:00 hola
-rw-r--r--    1 rogomez  academicos      1213 Aug 14 13:00 recibe.c
rogomez@armagnac:62>ls -F
a1  a2  colores  D1/  hola*  numeros  recibe.c
rogomez@armagnac:63>ls -d
.
rogomez@armagnac:64>
```

La salida en formato largo // Un ejemplo de salida en formato largo es el siguiente:

```
total 24
-rwxr-xr--  1 A00556677 alumnos   6531   Sep 15  18:13 cachafas
```

El número al lado de la palabra `total` representa el número de bloques (mínimo espacio físico asignable) que ocupan los archivos de dicho directorio. Tomando en cuenta la salida de arriba, los campos del formato largo son los siguientes:

- Primer caracter representa el tipo de archivo de acuerdo a la siguiente convención:
  - `-` lista los archivos que comienzan con un "."
  - `d` directorio
  - `d` socket
  - `d` pipe
  - `b` archivo especial dispositivos de bloque
  - `c` archivo especial dispositivos de caracteres

  En el ejemplo presentado caso se trata de un archivo común.
- Los siguientes nueve caracteres representan los permisos con que cuenta el propietario, los usuarios que pertenecen al mismo grupo que este y los tres últimos los permisos de todos los usuarios. Las letras representan el tipo de permiso de acuerdo a la siguiente nomenclatura
  - `r` permiso de lectura
  - `w` permiso de escritura
  - `x` permiso de ejecución
  - `-` no cuenta con dicho permiso

  Tomando en cuenta el ejemplo, el propietario cuenta con permisos de lectura, escritura y ejecución, el grupo con permisos de lectura y ejecución y el resto del mundo solo con permisos de ejecución.

- El número que sigue representa la cantidad de ligas sobre el archivo, en el ejemplo solo cuenta con una
- El propietario del archivo se despliega en la siguiente columna, `A00556677` es el propietario en el ejemplo
- Enseguida se muestra el grupo al que pertenece el propietario del archivo, `alumnos` en el ejemplo
- El tamaño del archivo, 6531 bytes en el ejemplo
- La fecha y hora de la última modificación, el 15 septiembre del año en curso, a las 18:13 fue la última vez que se modificó el archivo
- El nombre del archivo es `cachafas`

Metacaracteres
Son utilizados para hacer referencia a un conjunto de archivos cuyos nombres cuentan con caracteres en común. Los más utilizados son los siguientes:

`*` representa cualquier secuencia de caracteres

`?` sustituye un único carácter

`[ ]` cualquier caracter que se encuentre dentro de los corchetes, es posible definir un rango poniendo el primer y el último separados por un guión

Ejemplo de uso de metacaracteres:

```
rogomez@armagnac:65> ls
alliens.jpg  creasocket    hola     pgpcrack99.tar  recibe.c      socket
animales     creasocket.c  hola.c   pipe            revista       ws_ftp.log
colores      D1            liga     prueba.ppt      seciones.xls
correo1.gif  dico.tex      numeros  recibe          sesiones.pdf
rogomez@armagnac:66> ls [a-d]
ls: [a-d]: No such file or directory
rogomez@armagnac:67> ls [a-d]*
alliens.jpg  animales  colores  correo1.gif  creasocket  creasocket.c  dico.tex
rogomez@armagnac:68> ls [ad]
ls: [ad]: No such file or directory
rogomez@armagnac:69> ls [ad]*
alliens.jpg  animales  dico.tex
rogomez@armagnac:70> ls re*
recibe  recibe.c  revista
rogomez@armagnac:71> ls se?iones.*
seciones.xls  sesiones.pdf
rogomez@armagnac:72>
```

4.4 El comando *touch*

Descripción: Actualiza los tiempo de acceso y modificacin de cada archivo pasado como argumento al tiempo actual En caso de que el archivo no exista, crea un archivo vacío.

Sintaxis:

```
touch [opcion...] archivo...
```

Algunas opciones:

`-a` solo cambia el tiempo de acceso

`d, --date=STRING` utiliza STRING en lugar del tiempo actual

`-m` solo cambia el tiempo de modificación

`r, --reference=FILE` utiliza tiempo del archivo en lugar del tiempo actual

Ejemplos:

```
rogomez@armagnac:73>ls -l numeros
-rw-rw-r--    1 toto     toto          189 Jul 21 19:24 numeros
rogomez@armagnac:74> date
Wed Jul 21 19:26:31 CDT 2004
rogomez@armagnac:75> touch numeros
rogomez@armagnac:76> ls -l numeros
-rw-rw-r--    1 toto     toto          189 Jul 21 19:26 numeros
rogomez@armagnac:77>
```

4.5 El comando *file*

Descripción: Lleva a cabo una serie de comprobaciones en un archivo para tratar de clasificarlo. Tras su ejecución muestra el tipo de archivo e información al respecto del mismo.

Sintaxis:

```
file archivo...
```

Ejemplos:

```
rogomez@armagnac:78> ls
ComandosUnix.pdf       fig-s1.gif   hello
cve_sans.gif           fig-s2.gif   hello.c
dollarlogo_20x30.gif   fig-s3.gif   linuxpenguinlogo_30x30.gif
fig1.gif               fig-s4.gif   openbsdheadlogo_30x30.gif
fig2.gif               fig-s5.gif   son4.txt
fig3.gif               fig-s6.gif   winlogo_30x30.gif
rogomez@armagnac:79> file ComandosUnix.pdf
ComandosUnix.pdf: PDF document, version 1.2
rogomez@armagnac:80> file hello
hello: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), for
 GNU/Linux 2.2.5, dynamically linked (uses shared libs), not stripped
rogomez@armagnac:81> file hello.c
hello.c: ASCII text
rogomez@armagnac:82> file fig1.gif
fig1.gif: GIF image data, version 87a, 444 x 391
rogomez@armagnac:83>
```

4.6 El comando *echo*

Descripción: imprime sus argumentos sobre la salida estándar (la pantalla por default)

Sintaxis:

```
echo [ argumentos ]
```

Es posible combinar el comando echo con la redireccion de salida estandar para crear un archivo con un determinado contenido.

Ejemplo:

```
rogomez@armagnac:84>echo esto es una prueba
esto es una prueba
rogomez@armagnac:85>echo probando 1,2,3 > salida
rogomez@armagnac:86>more salida
probando 1,2,3
rogomez@armagnac:87>
```

4.7 El comando *clear*

Descripción: limpia la terminal si esto es posible. La forma de limpiarla varia de acuerdo al tipo de terminal.

Sintaxis:

```
clear
```

Nota: Es por demas decir que este comando no esta relacionado con el manejo de archivos.

4.8 El comando *more*

Descripción: despliegan el contenido de un archivo parándose cada vez que termina la pantalla.

Sintaxis:

```
more nombre-archivo
```

Ejemplo:

```
rogomez@armagnac:88>more numeros.txt
uno         un
dos         deux
tres        trois
cuatro      quatre
cinco       cinc
--More--(53%)
seis        six
siete       sept
ocho        huit
nueve       neuf
diez        dix
rogomez@armagnac:89>
```

Opciones de desplazamiento del comando more El comando more permite al usuario moverse en el archivo tal y como si estuviera usando un editor en este.

Tecla	**Propósito**
`SPACEBAR`	se visualiza pantalla por pantalla
`RETURN`	se visualiza una línea a la vez
`b`	se mueve una pantalla hacia atras
`f`	se mueve una pantalla hacia adelante
`h`	despliega un menu de ayuda de las opciones
`q`	se sale y regresa al shell
`/string`	busca hacia adelante el `string`
`n`	encuentra la siguiente ocurrencia del `string`

4.9 El comando *cat*

Descripción: su principal uso es el de concatenar archivos, pero también es utilizado para ver el contenido de un archivo. Lo que hace es copiar uno o varios archivos en la salida estándar (la pantalla por default). A diferencia del anterior este no se detiene entre pantalla y pantalla.

Sintaxis:

```
cat nombre-archivo
```

Ejemplo:

```
rogomez@armagnac:90>cat numeros.txt
uno         un
dos         deux
```

```
tres       trois
cuatro     quatre
cinco      cinc
seis       six
siete      sept
ocho       huit
nueve      neuf
diez       dix
rogomez@armagnac:91>
```

4.10 El comando *strings*

Descripción: para cada archivo pasado como argumento, el comando despliega las secuencias de caracteres imprimibles de hasta cuatro caracteres de largo (o el número especificado) y que son seguidos de caracteres no imprimibles. El comando es utilizado principalmente para determinar el contenido de archivos que no contienen texto.

Sintaxis:

```
strings [-a] [-n min-long]
```

Opciones:

- `-f` imprime el nombre del archivo antes de cada string
- `-n` imprime la cadena de longitud de caracteres que son al menos de longitud `min-len`

Ejemplo

```
rogomez@armagnac:92>ls -l hola
-rwxrwxr-x    1 rogomez  rogomez     11541 Jul 30 17:00 hola
rogomez@armagnac:7>file hola
hola: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), for GNU/Linux 2.2.5, dynamical
rogomez@armagnac:93>./hola
Hola Mundo
rogomez@armagnac:94>more hola

******** hola: Not a text file ********

rogomez@armagnac:95>strings hola
/lib/ld-linux.so.2
libc.so.6
printf
_IO_stdin_used
__libc_start_main
__gmon_start__
GLIBC_2.0
PTRh|
QVh(
Hola Mundo
rogomez@armagnac:96>
```

4.11 El comando *od*

Descripción: despliega una representación no ambigua, bytes octal por default, de un archivo a la salida estándar. Con más de un archivo como argumento, concatena los archivos en el orden proporcionado para la entrada. Cuando no se proporciona ningún archivo como argumento, o un caracter `-`, lee la entrada estándar. La primera columna representa la dirección de cada l ınea desplegada.

Sintaxis:

```
od [opciones] ... archivo ...
```

Opciones:

`-A` selecciona la base en la que el offset es desplegado
- `d` decimal
- `o` octal
- `x` hexadecimal

`-j bytes` se salta los primeros `bytes` y despliega a partir de ellos

`-N bytes` solo despliega los bytes indicados

`-t` define el formato de salida, entre los más usados encontramos:
- `d` decimal
- `o` octal
- `x` hexadecimal

`-w bytes` solo despliega los bytes indicados por línea

Ejemplo

```
rogomez@armagnac:52> more hola

******** hola: Not a text file ********

rogomez@armagnac:97> od hola | more
0000000 042577 043114 000401 000001 000000 000000 000000 000000
0000020 000002 000003 000001 000000 101170 004004 000064 000000
0000040 016424 000000 000000 000000 000064 000040 000006 000050
--More--
rogomez@armagnac:98>

rogomez@armagnac:98> od  -N 30 hola
0000000 042577 043114 000401 000001 000000 000000 000000 000000
0000020 000002 000003 000001 000000 101170 004004 000064
0000036
rogomez@armagnac:99> od  -Ad -N 30 hola
0000000 042577 043114 000401 000001 000000 000000 000000 000000
0000016 000002 000003 000001 000000 101170 004004 000064
0000030
rogomez@armagnac:100> od  -Ao -N 30 hola
0000000 042577 043114 000401 000001 000000 000000 000000 000000
0000020 000002 000003 000001 000000 101170 004004 000064
0000036
rogomez@armagnac:101> od  -Ax -N 30 hola
000000 042577 043114 000401 000001 000000 000000 000000 000000
000010 000002 000003 000001 000000 101170 004004 000064
00001e
rogomez@armagnac:102> od  -Ax -to -N 30 hola
000000 10623042577 00000200401 00000000000 00000000000
000010 00000600002 00000000001 01001101170 00000000064
00001e
rogomez@armagnac:103> od  -Ax -tx -N 30 hola
000000 464c457f 00010101 00000000 00000000
000010 00030002 00000001 08048278 00000034
00001e
rogomez@armagnac:104>
```

4.12 El comando *tar*

Descripción: programa diseñado para almacenar y extraer archivos de un archivo tar. El archivo tar puede construirse en una cinta, sin embargo es posible crearlo como un archivo común.

Sintaxis:

```
tar [ opcion ] archivo(s)
```

Opciones:

- `c` crear archivo
- `r` reemplazar
- `u` actualizar, los archivos son añadidos al archivo tar si no exsiten.
- `x` extraer / restablecer
- `v` da información de lo que el comando tar esta haciendo

Ejemplos:

```
rogomez@armagnac:105> ls -l
-rw-r-r--  1  rogomez      academico  96  Jul   1 17:08 arch1
-rw-r-r--  1 rogomez      academico 456  May 1 18:09 arch2
-rw-r-r--  1 rogomez      academico  56  May 5   2:69 arch3
rogomez@armagnac:106> tar -cvf nuevo arch1 arch2
rogomez@armagnac:107> ls -l
-rw-r-r--  1  rogomez      academico   96  Jul 1 17:08 arch1
-rw-r-r--  1 rogomez      academico 456  May 1 18:09 arch2
-rw-r-r--  1 rogomez      academico  56  May 5   2:69 arch3
-rw-r-r--  1 rogomez      academico 680   Jul   1  17:10 nuevo.tar
rogomez@armagnac:108> rm arch1
rogomez@armagnac:109> rm arch2
rogomez@armagnac:110> ls -l
-rw-r-r--  1 rogomez      academico   56  May 5   2:69 arch3
-rw-r-r--  1 rogomez      academico 680   Jul 1  17:10 nuevo.tar
rogomez@armagnac:111> tar -xfv nuevo
rogomez@armagnac:112> ls -l
-rw-r-r--  1  rogomez      academico  96  Jul 1  17:08 arch1
-rw-r-r--  1 rogomez      academico  456  May 1  18:09 arch2
-rw-r-r--  1 rogomez      academico   56  May 5   2:69 arch3
-rw-r-r--  1 rogomez      academico  680   Jul 1  17:10 nuevo.tar
rogomez@armagnac:113>
```

4.13 Los comando *gzip* y *gunzip*

Descripción: el comando *gzip* reduce el tamao de un archivo utilizando codificación Lempel-Ziv. Cuando es posible cada archivo pasado como argumento es reemplezado con otro de extensión `.gz`, manteniendo los permisos y estampillas de tiempo (accesos y modificación). El comando *gunzip* puede descomprimir archivos creados con los comandos *gzip zip compress* o *pack*. La detección del formato de entrada es automática.

Sintaxis:

```
gzip archivo
gunzip archivo
```

Ejemplos:

```
rogomez@armagnac:114>ls -l
total 1080
-rw-r--r--   1 rogomez profes   394751 Jan 26  2001 arch1.pdf
```

```
-rw-r--r--   1 rogomez profes   372670 Jan 26  2001 arch2.pdf
-rw-r--r--   1 rogomez profes   300325 Jan 26  2001 arch3.pdf
rogomez@armagnac:115>gzip arch1.pdf
rogomez@armagnac:116>ls -l
total 1064
-rw-r--r--   1 rogomez    382986 Jan 26  2001 arch1.pdf.gz
-rw-r--r--   1 rogomez    372670 Jan 26  2001 arch2.pdf
-rw-r--r--   1 rogomez    300325 Jan 26  2001 arch3.pdf
rogomez@armagnac:117>gunzip arch1.pdf.gz
rogomez@armagnac:118>ls -l
total 1080
-rw-r--r--   1 rogomez    394751 Jan 26  2001 arch1.pdf
-rw-r--r--   1 rogomez    372670 Jan 26  2001 arch2.pdf
-rw-r--r--   1 rogomez    300325 Jan 26  2001 arch3.pdf
rogomez@armagnac:119>
```

Nota: Existe otro tipo de comandos relacionados con comprensión de archivos. En la tabla de abajo se resumen estos

Extensión archivo	Comandos comprimir/descomprimir	Comentario
.Z	compress / uncompress	codigo Lempel-Ziv
.z	pack / unpack	cóigo Huffman
.zip	zip / unzip	código Lempel-Ziv
.gz	gzip / gunzip	versi ón GNU de zip
.rar	rar / unrar	formato propietario Roshal ARchiver
.bzip	bzip / bunzip	solo comprime datos, no archiva
.7z	7za , 7zr	formato abierto LZMA

4.14 El comando *ln*, link

Descripción: crea una liga para un archivo. Existen dos tipos de ligas, suave y dura (opción `-s` del comando). La liga dura permite que dos, o más, nombres de archivo hagan referencia al mismo espacio físico. No se puede crear una liga dura a un directorio y la liga y el archivo original son completamente idénticos, cualquier modificación se ve reflejado en el otro y una liga no ocupa espaci físico. Por otro lado, la liga suave (o liga simbólica) crea dos archivos, un archivo contiene los datos reales, el otro archivo sólo contiene el nombre del primero y sirve como apuntador al otro. La liga suave hace referencia la nombre del archivo y no al archivo en si. En este tipo de ligas es posible hacer referencia a archivos y directorios. Una liga suave ocupa una pequeña porción de espacio en disco.

Sintaxis:

```
ln [opciones]  archivo-a-ligar  liga
```

Opciones:

- `-f` forza la creación de una liga
- `-s` crea una liga simbólica

Ejemplo liga dura (ln):

```
rogomez@armagnac:89> ls
pln1
rogomez@armagnac:120> more pln1
Esto
```

```
es
una
prueba
rogomez@armagnac:121> ls -l pln1
-rw-rw-r--    1 toto     toto           21 Sep 18 10:29 pln1
rogomez@armagnac:122>ln pln1 pln2
rogomez@armagnac:123> ls -l pln2
-rw-rw-r--    2 toto     toto           21 Sep 18 10:29 pln2
rogomez@armagnac:124> more pln2
Esto
es
una
prueba
rogomez@armagnac:125> echo this is a test >> pln2
rogomez@armagnac:126> more pln2
Esto
es
una
prueba
this is a test
rogomez@armagnac:127> more pln1
Esto
es
una
prueba
this is a test
rogomez@armagnac:128> rm pln2
rogomez@armagnac:129> ll pln1
-rw-rw-r--    1 toto     toto           36 Sep 18 10:31 pln1
rogomez@armagnac:130>
```

Ejemplo liga suave (ln -s):

```
rogomez@armagnac:131> ls
a1
rogomez@armagnac:132> more a1
Esto es una prueba
rogomez@armagnac:133> ln -s a1 a1.liga
rogomez@armagnac:134> ls -l
Total 4
-rw-r--r--    1 rogomez gomez     9 Feb 12 10:09   a1
lrwxrwxrwx 1 rogomez gomez     4 Feb 23 17:24   a1.liga -> a1
rogomez@armagnac:135> echo This is a test >> a1.liga
rogomez@armagnac:136> more a1.liga
Esto es una prueba
This is a test
rogomez@armagnac:137> more a1
Esto es una prueba
This is a test
rogomez@armagnac:138>ls
a1 a1.liga
rogomez@armagnac:139> rm a1
rogomez@armagnac:140> ls -l
Total 0
lrwxrwxrwx   1   rogomez gomez     4 Feb 23 17:24   a1.liga -> a1
rogomez@armagnac:141> more a1.liga
```

```
a1.liga: Not such file or directory

rogomez@armagnac:142> ln -s /home/erick/grades  /tmp/grades.old
rogomez@armagnac:143> cd /tmp/grades.old
rogomez@armagnac:144> pwd
/home/erick/grades
rogomez@armagnac:145>
```

4.15 El comando *mkdir*

Descripción: utilizado en la creación de directorios.

Sintaxis:

```
mkdir [ opcion ] directorio...
```

Opciones:

- `-m` asigna permisos
- `-v` crea mensaje por cada directorio creado
- `-p` no error si existen, crea directorios padres si es necesario

Ejemplos:

```
rogomez@armagnac:146> ls -F
file1*    file2*   file3*   file4*  logfile  practica/
rogomez@armagnac:147> mkdir zoo
rogomez@armagnac:148>ls
file1*    file2*   file3*   file4*  logfile  practica/ zoo/
rogomez@armagnac:149> mkdir -p practica2/dir1/admin
rogomez@armagnac:150> ls -F
file1*    file2*   file3*   file4*  logfile  practica/ practica2/ zoo/
rogomez@armagnac:151> ls -F practica2
dir1/
rogomez@armagnac:152> cd practica2
rogomez@armagnac:153> ls -F dir1
admin/
rogomez@armagnac:154> mkdir -v dir1
mkdir: cannot create directory ‘dir1’: File exists
rogomez@armagnac:155> mkdir -v dir2
mkdir: created directory ’dir2’
rogomez@armagnac:156> ls -F practica2
admin/  dir2/
rogomez@armagnac:157>
```

4.16 El comando *rm* (remove)

Descripción: borra el nombre de un archivo o, si ese nombre fuera el último (el número de ligas es 1), el archivo será "físicamente" suprimido

Sintaxis:

```
rm archivo [ archivos ]
```

Opciones:

- `-r` recursivamente, si directorio contiene otro, borra contenido de este
- `-f` forza (no despliega errores, ni hace preguntas) ignora archivos no existentes y nunca previene

`-i` interactivo, (pregunta)

`-v` imprime nombre archivo antes de borrarlo

Ejemplo:

```
rogomez@armagnac:158>ls
prog.c    Tareas     toto.txt
rogomez@armagnac:159>ls Tareas
t1  t2
rogomez@armagnac:160>rm -i toto.txt
rm: remove toto.txt (yes/no)? y
rogomez@armagnac:161>ls
prog.c    Tareas
rogomez@armagnac:162>rm prog.c
rogomez@armagnac:163>ls
Tareas
rogomez@armagnac:164>rm -i Tareas/
rm: remove directory 'Tareas'? y
rm: cannot remove directory 'Tareas/': Is a directory
rogomez@armagnac:165>rm -r Tareas
rogomez@armagnac:20>ls
rogomez@armagnac:166>
```

4.17 El comando *rmdir*

Descripción: borra directorios, sin embargo este comando no borrará el directorio si este no se encuentra vacío

Sintaxis:

```
rmdir directorio  [ directorio ]
```

Ejemplo:

```
rogomez@armagnac:167> rmdir Tareas
rogomez@armagnac:168> rmdir Proyectos
rogomez@armagnac:169>rmdir Temporal
rmdir: directory "Temporal": Directory not empty
rogomez@armagnac:170>rm Temporal/*
rogomez@armagnac:171>
```

Nota:
Otra forma de borrar un directorio, la opción `-r` de `rm`.

4.18 El comando *chmod*

Descripción: sirve para cambiar los permisos de escritura, lectura y ejecución de una archivo o directorio. Solo el creador del archivo o directorio puede cambiar dichos permisos.

Sintaxis:

```
chmod nuevo-modo [ archivos  ]  [ directorios ]
```

Opciones:
Existen dos formas de especificar el nuevo modo:

1. en octal: `chmod ooo archivo`
2. en modo simbólico: `chmod [ ugoa ] [` = -] [rwx] + donde
   - `u` permisos del usuario
   - `g` permisos del grupo
   - `o` permisos de los otros

`a` todos los permisos

Ejemplo

```
rogomez@armagnac:172> ls -l e1
-rw-rw-rw-  1    toto       daemon       0 Oct 12 18:20  e1
rogomez@armagnac:173> chmod 755 e1
rogomez@armagnac:174> ls -l e1
-rwxr-xr-x  1    toto       daemon       0 Oct 12 18:20  e1
rogomez@armagnac:175> chmod a-x e1
rogomez@armagnac:176> ls -l e1
-rw-r--r--  1    toto       daemon       0 Oct 12 18:20  e1
rogomez@armagnac:177> chmod g+x  e1
rogomez@armagnac:178> chmod o-r  e1
rogomez@armagnac:179> ls -l e1
-rw-r-x---  1    toto       daemon       0 Oct 12 18:20  e1
rogomez@armagnac:180>
```

4.19 El comando *cp*

Descripción: copia un archivo ordinario

Sintaxis:

```
cp archivo1 archivo2
cp archivo  [archivos ]  directorio
```

Opciones:

- `-f` si archivo destino existente no puede ser abierto, lo borra e intenta de nuevo
- `-i` pregunta antes de copiarlo
- `-r` copia directorios recursivamente

Ejemplos:

```
rogomez@armagnac:181>ls
a1  dir1
rogomez@armagnac:182>cp a1 a2
rogomez@armagnac:183> ls
a1  a2  dir1
rogomez@armagnac:184> cp a1 a2 dir1
rogomez@armagnac:185> ls
a1  a2  dir1
rogomez@armagnac:186> ls dir1
a1  a2
rogomez@armagnac:187>
```

Notas:

- `cp` no modifica los archivos originales, tan solo los duplica
- la opción -r es copia recursiva, si el archivo a copiar es un directorio copia el contenido de este

4.20 El comando *mv*, move

Descripción: desplaza un archivo o lo renombra

Sintaxis:

```
mv antiguo-nombre  nuevo-nombre
mv archivo [ archivos ] directorio
```

Ejemplos:

```
rogomez@armagnac:188> ls
a1  dir1
rogomez@armagnac:189> mv a1 a2
rogomez@armagnac:190> ls
a2  dir1
rogomez@armagnac:191>ls dir1
rogomez@armagnac:192> mv a2 dir1/
rogomez@armagnac:193> ls dir1
a2
rogomez@armagnac:194> ls
dir1
rogomez@armagnac:195>
```

4.21 El comando *which*

Descripción: localiza un comando desplegando su pathname o alias. Toma una lista de nombres y busca por los archivos que serían ejecutados al escribir estos nombres como comandos. Cada argumento es expandido y buscado dentro del path del usuario. Tanto los alias como los paths son tomados del archivo `.cshrc`.

Sintaxis:

```
which [ nombre_archivo ]
```

Ejemplo:

```
rogomez@armagnac:196>which xeyes
/home/dic/rogomez/xeyes
rogomez@armagnac:197>which opnet
opnet: Command not found
rogomez@armagnac:198>which ls
alias ls='ls --color=tty'
     /bin/ls
rogomez@armagnac:199>
```

4.22 El comando *whereis*

Descripción: localiza el archivo binario, fuente y los archivos de los manuales de un comando

Sintaxis:

```
whereis  [ -bms ] archivo
```

Opciones:

- `-b` solo busca por binarios
- `-m` solo busca por secciones del manual
- `-s` solo busca por fuentes

Ejemplo:

```
rogomez@armagnac:200>whereis ls
ls: /bin/ls /usr/share/man/man1/ls.1.gz /usr/share/man/man1p/ls.1p.gz
rogomez@armagnac:201>whereis -b ls
ls: /bin/ls
rogomez@armagnac:202>whereis -m ls
ls: /usr/share/man/man1/ls.1.gz /usr/share/man/man1p/ls.1p.gz
rogomez@armagnac:203>whereis -s ls
```

```
ls:
rogomez@armagnac:204>
```

4.23 El comando *chown*

Descripción: cambia el propietario de un archivo

Sintaxis:

```
    chown propietario archivo
 chown --reference=ARCHIVO archivo
```

Opciones:

`-R` actua sobre directorios y archivos de forma recursiva

`--reference` cambio el propietario y grupo de cada archivo al del archivo pasado como referencia

Ejemplos:

```
rogomez@armagnac:205>ls -l
-rw-r-r-- 1     emata 19 May 1 18:09 arch1
rogomez@armagnac:206>chown root  arch1
rogomez@armagnac:207>ls -l
-rw-r-r-- 1     root   19 May 1 18:09 arch1
rogomez@armagnac:208>
```

4.24 El comando *chgrp*

Descripción: cambia el grupo propietario de un archivo

Sintaxis:

```
  chgrp propietario archivo
  chgrp --reference=ARCHIVO archivo
```

Opciones:

`-R` actua sobre directorios y archivos de forma recursiva

`--reference` cambio el grupo propietario de cada archivo al del archivo pasado como referencia

Ejemplos:

```
rogomez@armagnac:209>ls -lg
                -rw-r-r-- 1     emata       40      19 May 1 18:09 arch1
                rogomez@armagnac:210>chgrp 22 arch1
                rogomez@armagnac:211>ls -lg
                -rw-r-r-- 1     emata       22      19 May 1 18:09 arch1
                rogomez@armagnac:212>
```

4.25 El comando *umask*

Descripción: permite definir los permisos por default que los archivos y directorios tendran cuando se creen.

Sintaxis:

```
    umask [-p] [-S] [mode]
```

Opciones:

`-S` provoca que la mascara (permisos) se impriman en forma simbolica

`-p` si esta opción es proporcionada y `mode` es omitida, la salida es de tal forma que puede ser usuada como entrada.

Ejemplos:

```
rogomez@armagnac:242> umask
0002
rogomez@armagnac:243> touch a1
rogomez@armagnac:244> ls -l a1
-rw-rw-r--    1 rogomez  rogomez         0 Oct 17 22:09 a1
rogomez@armagnac:245> umask 066
rogomez@armagnac:246> touch a2
rogomez@armagnac:247> ls -l a2
-rw-------    1 rogomez  rogomez         0 Oct 17 22:09 a2
rogomez@armagnac:248>
```

4.26 El comando *find*

Descripción: permite encontrar archivos de acuerdo a varios criterios

Sintaxis:

```
find [path...] [expresion]
```

Parámetros:

`[path]` ruta del directorio donde empezará la búsqueda

`[expresion]` define el criterio de búsqueda y en caso de que sea verdad se lleva a cabo la acción especificada. Entre las posibles expresiones estan las siguientes:

Expresión	**Busca archivos que**
`-name filename`	concuerden con el nombre
`-size [+|-]`	mayores que +n, menores n o iguales a n
`-atime`	accedidos maś de +n días, menores n días y exactamente n días
`-mtime`	modificados mas de +n días, menores n días y exactamente n días
`-user loginID`	tengan propietario a loginID
`-type`	concuerden con un tipo archivo (f,d,s)
`-perm`	cuenten con ciertos permisos

Una vez que el comando encontro un archivo es posible llevar a cabo una acción sobre este. Entre las posibles acciones se encuentran las siguientes:

Acción	**Definición**
`-exec command {}\;`	ejecuta command a cada archivo encontrado. Los corchetes `{ }`, delimita donde se pasa el archivo como argumento. Espacio, backslash y punto y coma `(\;)` delimita el final del
`-ok command {}\;`	comando especifica la forma interactiva de `exec`. Requiere entrada antes que find aplique el command al archivo
`-print`	imprime el path completo en la salida estándar, es el default
`-ls`	imprime el pathname con todas sus características

Ejemplos:
Se dan ejemplos de búsqueda de archivos que deben cumplir con ciertas características y el lugar a partir de donde empieza la búsqueda.

- Archivos llamado `core`, desde directorio raíz

```
rogomez@armagnac:213> find / -name core
```

- Archivos llamados `core`, desde directorio hogar y borrarlos cuando se encuentran

```
rogomez@armagnac:214> find ~ -name core  - exec rm {} \;
```

- Archivos, desde directorio trabajo, que no han sido modificados en los últimos 90 das

```
rogomez@armagnac:215> find . -mtime +90
```

- Archivos mayores que 57 bloques (512-byte blocks) a partir del directorio hogar

```
rogomez@armagnac:216> find ~  -size +57
```

- Archivos cuyo nombre termina con `tif`, a partir del directorio `/usr`.

```
rogomez@armagnac:217> find /usr -name ‘*tif
```

- Posible utilizar caracteres "comodines" para buscar archivos cuyos nombre tienen caracteres en común. Necesario anteponer el caracter `\`. Por ejmplo si se desea buscar archivos con extension `.jpg`:

```
rogomez@armagnac:217> find / -name \*.jpg
```

5 Los comandos relacionados con usuarios

Este tipo de comandos nos permite obtener información acerca de los usuarios que estan utilizando el sistema.

Toda la información de los usuarios se encuentra almacenada en el archivo `/etc/password`. Este archivo se encuentra dividido en siete campos separados por el caracter de dos puntos (`:`). La sintaxis del archivo es la siguiente

```
usuario:password:uid:gid:gecos:home:shell
```

Donde

`usario`	el nombre de la cuenta del usuario
`passwd`	la contraseña cifrada del usuario
`uid`	el identificador del usuario
`gid`	el identificador del grupo del usuario
`gecos`	información sobre el usuario
`home`	directorio hogar del usuario
`shell`	shell de arranque del usuario

5.1 El comando *id*

Descripción: imprime los identificadores del usuario y del grupo

Sintaxis:

```
id [opciones]
```

Opciones:

`-y` despliega el calendario del año en curso.

`-m` imprime un calendario donde el lunes es el primer dia de la semana, en lugar del domingo

Ejemplo:

```
rogomez@armagnac:250>id
uid=501(rogomez) gid=501(rogomez) groups=501(rogomez),10(wheel),502(jesus)
rogomez@armagnac:251>
```

5.2 El comando *ulimit*

Descripción: Proporciona control sobre los recursos disponibles al shell y los procesos lanzados por él, en los sistemas que permiten tal control.

Sintaxis:

```
ulimit [ -SHa ]
```

Opciones:

`-S` especifica el limite suave

`-H` especifica el limite duro

`-a` despliega toda la información.

Ejemplo:

```
rogomez@armagnac:253> ulimit
unlimited
rogomez@armagnac:254> ulimit -a
core file size        (blocks, -c) 0
data seg size         (kbytes, -d) unlimited
file size             (blocks, -f) unlimited
max locked memory     (kbytes, -l) unlimited
max memory size       (kbytes, -m) unlimited
open files                    (-n) 1024
pipe size          (512 bytes, -p) 8
stack size            (kbytes, -s) 8192
cpu time             (seconds, -t) unlimited
max user processes            (-u) 2048
virtual memory        (kbytes, -v) unlimited
rogomez@armagnac:255>
```

5.3 El comando *who*

Descripción: despliega los usuarios conectados.

Sintaxis:

```
who
```

Ejemplo:

```
rogomez@armagnac:254>who
rogomez console Oct 10 09:48
rogomez ttyp0 Oct 10 11:18
mimoso  ttyp1 Oct 10 12:54
rogomez@armagnac:255>
```

5.4 El comando *whoami*

Descripción: muestra el nombre del usuario que tecleó el comando

Sintaxis:

```
whoami
```

Ejemplo:

```
rogomez@armagnac:256>whoami
rogomez
rogomez@armagnac:257>
```

5.5 El comando *su*

Descripción: permite cambiar de usuario, sin argumentos asume que se desea cambiar a root. Si es root no solicitara contrase

Sintaxis:

```
su [-] [usuario]
```

Opciones:

- cambia las variables de ambiente del antiguo usuario a las del nuevo usuario, en caso de no usar esta opción se queda con las variables de ambiente del antiguo usuario.

Ejemplo:

```
root@armagnac:53> id
uid=0(root) gid=0(root)
root@armagnac:54>su user1
user1@armagnac:55>id
uid=501(user1) gid=501(user1) groups=501(rogomez)
user1@armagnac:56>exit

root@armagnac:57>id
uid=0(root) gid=0(root)
root@armagnac:58>echo $PATH
/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin
root@armagnac:59>su user1
user1@armagnac:60>echo $PATH
/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin
user1@armagnac:61>exit

root@armagnac:62>id
uid=0(root) gid=0(root)
rooot@armagnac:63>echo $PATH
/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin
root@armagnac:64>su - user1
user1@armagnac:65>echo $PATH
/usr/local/bin:/usr/bin/:/home/user1/bin:
user1@armagnac:66>

user1@armagnac:66>su user2
Password:
user2@armagnac:67>id
uid=502(user2) gid=502(user2) groups=502(user2)
user2@armagnac:68>
```

6 Comandos relacionados con el tiempo

Los siguientes comandos proporcionan información acerca de la fecha y hora.

6.1 El comando *date*

Descripción: Despliega el tiempo actual del sistema en un determinado formato. Tambien permite definir la fecha y hora del sistema. Es posible definir el formato de salida.

Sintaxis:

```
date [ -u | --utc | --universal] [MMDDhhmm[[CC]YY][.ss]]
```

Opciones:

`-u` despliega en modo GMT (Greenwich Mean Time) saltandose el formato local.

`d, --date=STRING` despliega tiempo descrito por STRING

`r, --reference=FILE` despliega tiempo ltima

`s, --set reference=STRING` asigna tiempo descrito por STRING

Ejemplo:

```
rogomez@armagnac:280> date
Fri Mar 12 19:59:08 CST 1999
rogomez@armagnac:281> date '+DATE: %d-%n-10%y%nHEURE: %H:%M:%S'
DATE: 10-10-1988
HEURE: 16:01:47
rogomez@armagnac:282> date
Fri Jun 29  12:00:44  CDT 2007
rogomez@armagnac:283> date -u
Fri Jun 29  17:00:50  UTC 2007
rogomez@armaganc:284> date -r a1
Fri Jun 25  11:09:50  UTC 2007
rogomez@armaganc:285> date -d '11/20/2003 12:08:01'
Fri Jun 25  11:09:50  UTC 2007
rogomez@armagnac:286> date
Fri Jun 29 12:23:57 CDT 2007
rogomez@armagnac:287> date '+DATE: %m-%d-%y%nTIME: %H:%M:%S'
DATE: 06-29-07
TIME: 12:24:00
rogomez@armagnac:288> date '+FECHA: %d.%m.%y%nHORA: %H:%M:%S'
FECHA: 29.06.07
HORA: 12:24:03
rogomez@armagnac:289>
```

6.2 El comando *cal*

Descripción: Despliega el calendario del mes y año en curso.

Sintaxis:

```
cal [[mes]ao]
```

Opciones:

`-y` despliega el calendario del año en curso.

`-m` imprime un calendario donde el lunes es el primer día de la semana, en lugar del domingo

Ejemplo:

```
rogomez@armagnac:290>cal
      July 2008
Su Mo Tu We Th Fr Sa
       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
rogomez@armagnac:291> cal -m
      July 2008
Mo Tu We Th Fr Sa Su
    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
rogomez@armagnac:292> cal 2008
                           2008
       January               February              March
Su Mo Tu We Th Fr Sa   Su Mo Tu We Th Fr Sa   Su Mo Tu We Th Fr Sa
       1  2  3  4  5                   1  2                   1
 6  7  8  9 10 11 12    3  4  5  6  7  8  9    2  3  4  5  6  7  8
13 14 15 16 17 18 19   10 11 12 13 14 15 16    9 10 11 12 13 14 15
20 21 22 23 24 25 26   17 18 19 20 21 22 23   16 17 18 19 20 21 22
27 28 29 30 31         24 25 26 27 28 29      23 24 25 26 27 28 29
                                 30 31
       April                 May                   June
Su Mo Tu We Th Fr Sa   Su Mo Tu We Th Fr Sa   Su Mo Tu We Th Fr Sa
       1  2  3  4  5                1  2  3    1  2  3  4  5  6  7
 6  7  8  9 10 11 12    4  5  6  7  8  9 10    8  9 10 11 12 13 14
13 14 15 16 17 18 19   11 12 13 14 15 16 17   15 16 17 18 19 20 21
20 21 22 23 24 25 26   18 19 20 21 22 23 24   22 23 24 25 26 27 28
27 28 29 30            25 26 27 28 29 30 31   29 30

       July                  August                September
Su Mo Tu We Th Fr Sa   Su Mo Tu We Th Fr Sa   Su Mo Tu We Th Fr Sa
       1  2  3  4  5                   1  2       1  2  3  4  5  6
 6  7  8  9 10 11 12    3  4  5  6  7  8  9    7  8  9 10 11 12 13
13 14 15 16 17 18 19   10 11 12 13 14 15 16   14 15 16 17 18 19 20
20 21 22 23 24 25 26   17 18 19 20 21 22 23   21 22 23 24 25 26 27
27 28 29 30 31         24 25 26 27 28 29 30   28 29 30
                       31
       October               November              December
Su Mo Tu We Th Fr Sa   Su Mo Tu We Th Fr Sa   Su Mo Tu We Th Fr Sa
          1  2  3  4                      1       1  2  3  4  5  6
 5  6  7  8  9 10 11    2  3  4  5  6  7  8    7  8  9 10 11 12 13
12 13 14 15 16 17 18    9 10 11 12 13 14 15   14 15 16 17 18 19 20
19 20 21 22 23 24 25   16 17 18 19 20 21 22   21 22 23 24 25 26 27
26 27 28 29 30 31      23 24 25 26 27 28 29   28 29 30 31
                       30
rogomez@armagnac:293>
```

7 Otros comandos

Los comandos descritos en esta sección no pudieron ser agrupados o pertenecen a un grupo muy reducido. En esta sección el lector encontrará información sobre el manual, manejo de terminales y otros.

7.1 El comando *man*, (manual de Unix)

Descripción: permite conocer todo lo referente a un comando, llamada de sistema o dispositivo relacionado con Unix

Sintaxis:

```
man  [ opcion ]  [ seccion ]  titulo(s)
```

Opciones:

`-k` busca todas las secciones del manual que contengan información concerniente al comando.

`-s` busca en una sección en específico información sobre el comando.

Ejemplo:

```
rogomez@armagnac:300>man man
Reformatting page.  Wait... done

User Commands                                                   man(1)

NAME
    man - find and display reference manual pages

SYNOPSIS
     man [ - ] [ -adFlrt ] [ -M path ] [ -T macro-package ]
         [-s section ] name ...
     man [ -M path ] -k keyword ...
     man [ -M path ] -f file ...

DESCRIPTION
    The man command  displays  information  from  the  reference
    manuals.   It displays complete manual pages that you select
    by name, or one-line summaries selected  either  by  keyword
                         :
                         :
rogomez@armagnac:301>
```

Opción `-k`

Es posible invocar el comando `man` con la opción `-k <keyword>`, para enlistar los comandos relevantes y relacionados con el keyword. Esta opción no esta activada por default. El administrador debe activarla a través del comando `catman`.

El administrador debe activar la opción:

```
rogomez@armagnac:211> catman -w
rogomez@armagnac:212>
```

La salida del comando da lo siguiente:

```
rogomez@armagnac:302> man -k calendar
cal     cal (1)   - display a calendar
calendar   calendar (1) - reminder service
```

```
difftim     difftime (3c)  - computes the difference between two calendar times
mktime     mktime (3c) - converts a tm structure to a calendar time
        rogomez@armagnac:303>
```

Nota:
Si no se especifica ninguna sección, la página a imprimir es buscada en todas las secciones

7.2 El comando *history*

Descripción: Despliega un historial de lo tecleado por el usuario. Es posible asociar lo tecleado

Sintaxis:

```
history [OPCIONES]
```

Opciones:

`-c` limpia la lista de history borrando todas las entradas

`-d offset` borra la lista de history a partir de la posición `offset`

Comentario Es posible repetir alguno de los comandos listados utilizando el caracter `!` y el número de línea a repetir o un string. En este último caso ejecutará la primera línea que coincida con el string pasado como argumento.

Ejemplo:

```
rogomez@armagnac:1> uname
Linux
rogomez@armagnac:2> who
rogomez  :0           Jul 30 11:33
rogomez  pts/0        Jul 30 11:34 (:0.0)
rogomez@armagnac:3> date
Wed Jul 30 13:41:56 CDT 2008
rogomez@armagnac:4> toto
bash: toto: command not found
rogomez@armagnac:5> whoami
rogomez
rogomez@armagnac:6> history
    1  uname
    2  who
    3  date
    4  toto
    5  whoami
    6  history
rogomez@armagnac:7> !3
date
Wed Jul 30 13:42:07 CDT 2008
rogomez@armagnac:8> !who
whoami
rogomez
rogomez@armagnac:9>
```

7.3 El comando *alias*

Descripción:

Permite asignar un equivalente, o alias, de un comando, de acuerdo al formato `nombre=valor`, donde `nombre` es el nombre del equivalente del comando definido por `valor`.

Sin argumento, o con la opción `-p`, imprime la lista de alias de la forma `nombre=valor` en la salida estándar.

Los cambios no serán permanentes si no se aade al archivo `.bashrc`.

Sintaxis:

```
alias [-p] [nombre[=valor] ...]
```

Opciones:

`-p` despliega la lista de alias

Ejemplo

```
rogomez@armagnac:311> date
Thu Jul 10 18:14:02 CDT 2008
rogomez@armagnac:312> alias
alias l.='ls -d .* --color=tty'
alias ll='ls -l --color=tty'
alias ls='ls --color=tty'
alias vi='vim'
rogomez@armagnac:313> fecha
bash: fecha: command not found
rogomez@armagnac:314> alias fecha=date
rogomez@armagnac:315> fecha
Thu Jul 10 18:14:16 CDT 2008
rogomez@armagnac:316> alias
alias fecha='date'
alias l.='ls -d .* --color=tty'
alias ll='ls -l --color=tty'
alias ls='ls --color=tty'
alias vi='vim'
rogomez@armagnac:317>
```

7.4 El comando *tee*

Descripción: lee de la entrada estándar y escribe a la salida estándar y archivos

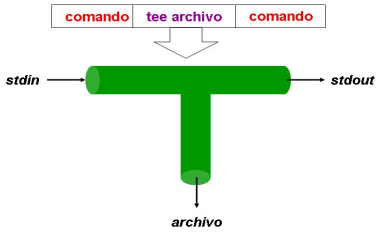

Figure 4: *El comando tee*

Sintaxis:

```
tee [OPCIONES]...   [ARCHIVO]...
```

Opciones:

`-a` añade los datos a los archivos, no sobreescribe

`-i` ignora las señales de interrupción

Ejemplo:

```
rogomez@armagnac:318> ls | tee salida
dante dante1 dir1 dir2 dir3 dir4 file.1   file.2    file.3
file1 file2 file3 file4 fruit fruit2 practice tutor.vi
rogomez@armagnac:319> more salida
dante dante1 dir1 dir2 dir3 dir4 file.1   file.2    file.3
file1 file2 file3 file4 fruit fruit2 practice tutor.vi
rogomez@armagnac:320> cal | tee a resultado
      August 2000
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
rogomez@armagnac:321>
```

7.5 El comando *tty*

Descripción: permite la identificación de la terminal. Regresa el nombre de la terminal del usuario.

Sintaxis:

```
tty [OPCION] ...
```

Opciones:

`-s` no imprime nada, solo regresa un status de salida

Ejemplo

```
rogomez@armagnac:322>tty
/dev/pts/6
rogomez@armagnac:323>tty -s
rogomez@armaganc:324>
```

Notas:
Nombre usuado es el equivalente al regresado por la función `ttyname()`

7.6 El comando *uname*

Descripción: sirve para la identificación del sistema. Despliega información acerca del sistema sobre el cual se esta trabajando. Si no se especifica ninguna opción, imprime el nombre del sistema

Sintaxis:

```
uname [ -mnrsva]
```

Opciones:

`-m` imprime el nombre de la máquina

`-n` imprime el nombre del nodo, el cual es utilizado para comunicaciones a través de una red

`-r` imprime la referencia de liberación, (release) del sistema operativo

`-s` imprime el nombre del sistema

`-v` imprime la versión del sistema operativo

`-a` imprime toda la información anterior

Ejemplo:

```
rogomez@armagnac:325>uname -a
SunOS mexico 4.1.3_U1 2 sun4c
rogomez@armagnac:326>
```

8 Comandos relacionados con procesos

Los procesos son una parte fundamental en todo sistema operativo.

8.1 El comando *ps*

Descripción: proporciona una lista de todos los procesos del sistema. Cada vez que se está ejecutando un comando o un programa se le asocia un número de proceso. El comando `ps` permite ver los números asociados a los procesos

Sintaxis:

```
ps [ [ -ef ]
```

Algunas opciones:

`-e` imprime información de cada proceso en el sistema, incluyendo PID, TTY, TIME y CMD

`-l` genera un listado completo, que añade los campos UID, PPID y STIME

Campos salida completa:

Campos	**Significado**
`UID`	nombre usuario propietario del proceso
`PID`	identificador del proceso
`PPID`	identificador del pariente del proceso
`C`	uso CPU para calendarización (obsoleto)
`STIME`	tiempo proceso empezó (hh:mm:ss)
`TTY`	terminal en la que proceso empezó
`TIME`	tiempo ejecución acumulativo del proceso
`CMD`	nombre del mcomando que creo el proceso

Ejemplo:

```
rogomez@armagnac:350>more ps.sal
  PID TTY          TIME CMD
 2005 pts/0    00:00:00 bash
 2265 pts/0    00:00:00 ps
rogomez@armagnac:351>ps
  PID TTY          TIME CMD
 2005 pts/0    00:00:00 bash
 2267 pts/0    00:00:00 ps
rogomez@armagnac:352>ps -f
UID        PID  PPID  C STIME TTY          TIME CMD
rogomez   2005  2003  0 15:30 pts/0    00:00:00 bash
rogomez   2268  2005  0 16:03 pts/0    00:00:00 ps -f
rogomez@armagnac:353>ps -e
rogomez@armagnac:354>ps -e
  PID TTY          TIME CMD
    1 ?        00:00:05 init
    2 ?        00:00:00 keventd
```

```
    3 ?        00:00:00 kapmd
    4 ?        00:00:00 ksoftirqd_CPU0
     :
     :
     :

 2003 ?        00:00:09 gnome-terminal
 2004 ?        00:00:00 gnome-pty-helpe
 2005 pts/0    00:00:00 bash
 2271 pts/1    00:00:00 bash
 2300 pts/1    00:00:00 vim
 2301 pts/0    00:00:00 ps
rogomez@armagnac:355>
```

8.2 El comando *pgrep*

Descripción: busca entre los procesos ejecutándose, y despliega en salida estándar, los identificadores de aquellos que concuerden con el crierio de selección. En el caso de que sean varios criterios, todos tienen que concordar para desplegar el identificador.

Sintaxis:

```
pgrep [-lnv] [patron busqueda]
```

Algunas opciones:

- `-l` lista el nombre del proceso y el identificador
- `-n` selecciona el proceso mas recientemente creado
- `-v` niega el match

Ejemplo:

```
rogomez@armagnac:352> ps
  PID TTY          TIME CMD
 2271 pts/1    00:00:00 bash
 2316 pts/1    00:00:00 vim
 2329 pts/1    00:00:00 ps
rogomez@armagnac:352> pgrep vi
2316
rogomez@armagnac:352> pgrep -l vi
2316 vim
rogomez@armagnac:352> pgrep -v vi
1
2
3
  :
  :
  :
1998
2003
2004
2005
2271
rogomez@armagnac:352>
```

8.3 El comando *top*

Descripción: proporciona un vista de la actividad del procesador a tiempo real; a diferencia del comando `ps` que toma una fotografía de los procesos en el momento en que se ejecuta el comando.

Sintaxis:

```
top
```

Ejemplo:

```
rogomez@armagnac:347> top
```

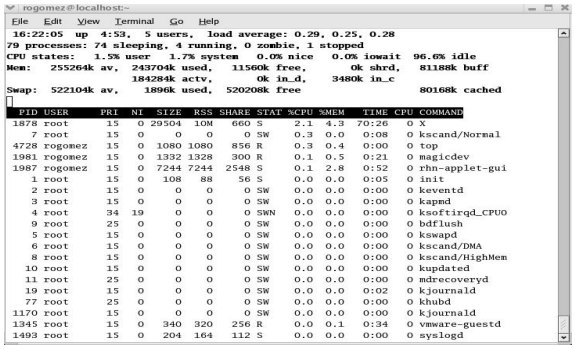

Figure 5: *Ejemplo salida comando top*

8.4 El comando *nice*

Descripción: ejecuta un comando con una determinada priorodad de calendarización Si no se especifica ningun comando, despliega la prioridad de calendarización actual. El rango varía entre -20 (máxima prioridad) y 19 (baja prioridad). Un usuario sin privilegios no puede aumentar su prioridad, tan solo puede disminuirla, solo el superusuario puede aumentar prioridades.

Sintaxis:

```
nice [opcion] [comando]
```

Opciones:

`-n` incrementa la prioridad por

Ejemplo:

```
rogomez@armagnac:357>nice prueba
rogomez@armagnac:358>
```

8.5 El comando *nohup*

Descripción: permite que el programa continúe ejecutandose aun cuando el usuario haya terminado su sesión. Si no se utilizan redirecciones, todas las salidas del programa se dirigen a un archivo de nombre `nohup.out`. Cuando se utiliza este comando el sisyema disminuye la prioridad de la ejecución del proceso.

Sintaxis:

```
nohup comando
```

Ejemplo:

```
rogomez@armagnac:359>nohup prueba
```

8.6 El comando *time*

Descripción: el comando time ejecuta el programa/comando especificado con los argumentos proporcionados. Cuando termina la ejecución, se despliega un mensaje a la salida estándar proporcionando estadísticas sobre el tiempo de ejecución, las cuales consisten en el tiempo real que paso entre la invocación y terminación (real), el tiempo CPU del usuario y el tiempo del CPU del sistema.

Sintaxis:

```
time comando [argumentos]
```

Ejemplo:

```
rogomez@armagnac:360>time prueba
real 0m3.623s
user 0m0.010s
sys 0m0.040s
rogomez@armagnac:361>
```

8.7 El comando *jobs*

Descripción: imprime una lista de los trabajos ejecutandose y su status

Sintaxis:

```
jobs [ OPCIONES ] [PID]
```

Algunas opciones:

- `-l` lista los identificadores de procesos aparte de la información de costumbre
- `-r` restringe la salida a los procesos que están "corriendo"
- `-s` restringe la salida a los procesos detenidos

Acciones relacionadas

Comando	Valor
`jobs`	despliega los trabajos (jobs) que se encuentran actualmente corriendo
`bg %n`	pone el trabajo n en background
`fg %n`	pone el trabajo n al frente (foreground)
`^Z`	detiene el trabajo del frente (foreground)
`stop %n`	detiene el trabajo n en background

8.8 El comando *kill*

Descripción: Envia una señal a un proceso. Si no se especifica ninguna señal se envia la señal SIGTERM. Esta señal provoca que el proceso que la reciba deje de ejecutar, siempre y cuando este proceso no la capture. Para otros procesos sera necesario utilizar la señal SIGKILL (9) para matar al proceso.

Sintaxis:

```
kill  [ -s signal | -p ] [ -a ] [ -- ] pid ...
```

Argumentos y algunas opciones:

`pid` lista de procesos a los que se les enviará la señal.

`-s signal` especifica la señal a enviar, es posible omitir el caracter s para definir la se

`-l` imprime la lista de los nombres de las señales

Ejemplo:

```
rogomez@armagnac:65> vi toto &
[1] 4285
rogomez@armagnac:65> ps
  PID TTY          TIME CMD
 3454 pts/3    00:00:00 bash
 4285 pts/3    00:00:00 vim
 4297 pts/3    00:00:00 ps
rogomez@armagnac:65> kill 4265
bash: kill: (4265) - No such process
rogomez@armagnac:65> kill 4285
rogomez@armagnac:65> ps
  PID TTY          TIME CMD
 3454 pts/3    00:00:00 bash
 4285 pts/3    00:00:00 vim
 4298 pts/3    00:00:00 ps
rogomez@armagnac:65> kill -9 4585
bash: kill: (4585) - No such process
rogomez@armagnac:65> ps
  PID TTY          TIME CMD
 3454 pts/3    00:00:00 bash
 4285 pts/3    00:00:00 vim
 4302 pts/3    00:00:00 ps
rogomez@armagnac:65> kill -9 4285
rogomez@armagnac:65> ps
  PID TTY          TIME CMD
 3454 pts/3    00:00:00 bash
 4303 pts/3    00:00:00 ps
[2]+  Killed                  vim toto
rogomez@localhost rogomez]$ kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL
 5) SIGTRAP      6) SIGABRT      7) SIGBUS       8) SIGFPE
 9) SIGKILL     10) SIGUSR1     11) SIGSEGV     12) SIGUSR2
13) SIGPIPE     14) SIGALRM     15) SIGTERM     17) SIGCHLD
18) SIGCONT     19) SIGSTOP     20) SIGTSTP     21) SIGTTIN
22) SIGTTOU     23) SIGURG      24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM   27) SIGPROF     28) SIGWINCH    29) SIGIO
30) SIGPWR      31) SIGSYS      33) SIGRTMIN    34) SIGRTMIN+1
35) SIGRTMIN+2  36) SIGRTMIN+3  37) SIGRTMIN+4  38) SIGRTMIN+5
39) SIGRTMIN+6  40) SIGRTMIN+7  41) SIGRTMIN+8  42) SIGRTMIN+9
43) SIGRTMIN+10 44) SIGRTMIN+11 45) SIGRTMIN+12 46) SIGRTMIN+13
47) SIGRTMIN+14 48) SIGRTMIN+15 49) SIGRTMAX-14 50) SIGRTMAX-13
51) SIGRTMAX-12 52) SIGRTMAX-11 53) SIGRTMAX-10 54) SIGRTMAX-9
55) SIGRTMAX-8  56) SIGRTMAX-7  57) SIGRTMAX-6  58) SIGRTMAX-5
59) SIGRTMAX-4  60) SIGRTMAX-3  61) SIGRTMAX-2  62) SIGRTMAX-1
63) SIGRTMAX
rogomez@armagnac:65>
```

8.9 El comando *pkill*

Descripción: envía la señal especificada (por defecto SIGTERM) a cada proceso que coincida con el criterio de busqueda

Sintaxis:

```
pkill [-signal] [-lnv] [patron busqueda]
```

Algunas opciones:

`-n` selecciona el proceso más recientemente creado

`-v` niega el criterio de búsqueda

Ejemplo:

```
rogomez@armagnac:360> ps
  PID TTY          TIME CMD
 2271 pts/1    00:00:00 bash
 2316 pts/1    00:00:00 vim
 2329 pts/1    00:00:00 ps
rogomez@armagnac:361> pkill vi
2316
rogomez@armagnac:362>
```

9 Los comandos tipo filtro

Este tipo de comandos fueron diseñados para actuar sobre archivos tipo texto, es decir archivos que solo contienen caracteres imprimibles. Los comandos reciben un archivo como argumento de entrada, lo procesan y el resultado lo envían a salida estandar. Si se requiere que el resultado se almacene en un archivo es necesario redireccionar la salida estandar al archivo.

9.1 El comando *grep*

Descripción: sirve para encontrar dentro de un conjunto de archivos, todas las líneas que contienen una cadena de caracteres especificada por una expresión regular

Sintaxis:

```
grep [ opciones  ] expr-reg [ archivos ]
```

Opciones:

`-v` despliega las líneas que no contienen la expresión

`-c` imprime solo el número de líneas que contienen la expresión

`-i` no hace diferencia entre mayúsculas y minúsculas

`-n` despliega el número de línea

`-l` solo lista los nombres de los archivos que coincidad con lo buscado

`-w` realiza la búsqueda como una palabra, ignora aquellas concordancias que son substrings de palabras más grandes

Ejemplos:

```
rogomez@armagnac:87> more agenda
aguirre claudia 5456789
burron regino  8719890
Gomez Yolanda 9218877
gomez roberto 3218956
gomez gabriel 3331811
rogomez@armagnac:87> grep gomez agenda
gomez roberto 3218956
gomez gabriel 3331811
rogomez@armagnac:87> grep -v gomez agenda
aguirre claudia 5456789
burron regino  8719890
```

```
Gomez Yolanda 9218877
rogomez@armagnac:87> grep -c gomez agenda
2
rogomez@armagnac:87> grep -i gomez agenda
Gomez Yolanda 9218877
gomez roberto 3218956
gomez gabriel 3331811
rogomez@armagnac:87> grep -n gomez agenda
4:gomez roberto 3218956
5:gomez gabriel 3331811
rogomez@armagnac:87>
rogomez@armagnac:87> grep -l gomez agenda
agenda
rogomez@armagnac:87> grep -w gomez agenda
gomez roberto 3218956
gomez gabriel 3331811
rogomez@armagnac:87>
```

Notas:
Dentro de la misma familia, se encuentran los comandos siguientes:

- `fgrep` no admite las expresiones regulares
- `egrap` admite expresiones regulares extendidas

9.2 El comando *sort*

Descripción: permite ordenar las líneas de un archivo texto. Por default, `sort` ordena en función de todos los caracteres de la línea, en orden creciente de los valores de caracteres ASCII.

Sintaxis:

```
sort [opciones] [llave de ordenamiento] [archivos]
```

Opciones:

- `-u` suprime las líneas conteniendo las llaves idénticas
- `-n` ordenamiento numérico
- `-r` ordenamiento en sentido inverso
- `-k` definición de campo llave

Ejemplos:

```
rogomez@armagnac:R20>cat numeros
uno         un
dos         deux
tres        trois
cuatro      quatre
cinco       cinc
rogomez@armagnac:21>sort numeros
cinco       cinc
cuatro      quatre
dos         deux
tres        trois
uno         un
rogomez@armagnac:22>
```

9.3 El comando *wc*

Descripción: permite contar el número de líneas, palabras y caracteres contenidos en los archivos

Sintaxis:

```
wc [opciones] [archivos]
```

Opciones:

- `-l` cuenta solo las líneas
- `-w` cuenta solo las palabras
- `-c` cuenta solo los caracteres

Ejemplos:

```
 rogomez@armagnac:22>wc /etc/passwd
20      37     752 /etc/passwd
 rogomez@armagnac:23>cat numeros
 uno        un
 dos        deux
 tres       trois
 cuatro     quatre
 cinco      cinc
 rogomez@armagnac:24>wc numeros
5      10      81 numeros
 rogomez@armagnac:25>wc -c /etc/passwd
     752 /etc/passwd
 rogomez@armagnac:26>
```

9.4 El comando *tail*

Descripción: imprime la parte final de un archivo en la salida estándar

Sintaxis:

`tail [-/n] [opciones] [archivo] +`

Opciones:

- `-n` imprime las últimas n líneas, (default 10 últimas)
- `+n` imprime a partir de la enésima línea (incluida)
- `-r` imprime las líneas en orden inverso

Ejemplos:

```
rogomez@armagnac:26>tail /etc/passwd
nobody:x:60001:60001:Nobody:/:
noaccess:x:60002:60002:No Access User:/:
nobody4:x:65534:65534:SunOS 4.x Nobody:/:
msql:x:85:10:Mini SQL:/local/Hughes:/bin/csh
mant:x:86:10:Mantenimiento:/tmp:/sbin/sh
nsuser:x:101:101:Http User:/local/ns:/bin/sh
root-mdg:x:0:0:Damian Guerra:/tmp:/bin/csh
root-gg:x:0:0:Guillermo Gutierres:/tmp:/bin/csh
root-im:x:0:0:Ixchell Morales:/tmp:/bin/csh
root-er:x:0:0:Edgar Romero:/tmp:/bin/csh
rogomez@armagnac:27>tail +3 numeros
tres       trois
cuatro     quatre
cinco      cinc
```

```
rogomez@armagnac:28>cat /etc/passwd | tail -4
root-mdg:x:0:0:Damian Guerra:/tmp:/bin/csh
root-gg:x:0:0:Guillermo Gutierres:/tmp:/bin/csh
root-im:x:0:0:Ixchell Morales:/tmp:/bin/csh
root-er:x:0:0:Edgar Romero:/tmp:/bin/csh
rogomez@armagnac:29>
```

9.5 El comando *head*

Descripción: imprime el principio de un archivo en la salida estándar

Sintaxis:

```
head [-n] [archivo]
```

Opciones:

`-n` imprime las n primeras líneas (default 10 primeras)

Ejemplos:

```
rogomez@armagnac:35>head -2 numeros
uno        un
dos        deux
rogomez@armagnac:36>head /etc/passwd
root:x:0:1:Super-User:/:/sbin/sh
daemon:x:1:1::/:
bin:x:2:2::/usr/bin:
sys:x:3:3::/:
adm:x:4:4:Admin:/var/adm:
lp:x:71:8:Line Printer Admin:/usr/spool/lp:
smtp:x:0:0:Mail Daemon User:/:
uucp:x:5:5:uucp Admin:/usr/lib/uucp:
nuucp:x:9:9:uucp Admin:/var/spool/uucppublic:/usr/lib/uucp/uucico
listen:x:37:4:Network Admin:/usr/net/nls:
rogomez@armagnac:37>
```

9.6 El comando *cut*

Descripción: elimina secciones de cada línea de los archivos pasado como argumento. El resultado se envia a salida estándar

Sintaxis:

```
cut []... []...
```

Opciones:

`-d DELIM` usiliza `DELIM` como delimitador en lugar del tabulador

Ejemplos:

```
rogomez@armagnac:45> cat passwd
root:x:0:0:root:/root:/bin/bash
murie:x:500:500:M Muriel Cordero:/home/murie:/bin/bash
practica:x:501:501:Usuario practicas:/home/practica:/bin/ksh
wizardi:x:502:502:Wizard nethack:/home/wizard:/bin/bash
rogomez@armagnac:45>  cut -f1,7 -d: passwd
root:/bin/bash
murie:/bin/bash
practica:/bin/ksh
```

```
wizardi:/bin/bash
rogomez@armagnac:45>
```

9.7 El comando *paste*

Descripción: mezcla líneas de archivos

Sintaxis:

```
paste []... []...
```

Opciones:

`-d DELIM` usiliza `DELIM` como delimitador en lugar del tabulador

Ejemplos:

```
rogomez@armagnac:587> cat num-esp
uno
dos
tres
cuatro
cinco
rogomez@armagnac:587>  cat num-fra
un
deux
trois
quatre
cinq
rogomez@armagnac:587>  paste num-esp num-fra
uno     un
dos     deux
tres    trois
cuatro  quatre
cinco   cinq
rogomez@armagnac:587>
```

9.8 El comando *uniq*

Descripción: elimina líneas duplicadas de un archivos que se encuentra ordenado

Sintaxis:

```
cut
```

Opciones:

`-u` solo despliega las líneas no repetidas

`-d` despliea las líneas no repetidas

`-c` solo imprime el número de líneas repetidas

Ejemplos:

```
rogomez@armagnac:11> more nums
uno
dos
tres
cuatro
uno
```

```
rogomez@armagnac:11> uniq nums
uno
dos
tres
cuatro
uno
rogomez@armagnac:11> sort nums > ordena
rogomez@armagnac:11> more ordena
cuatro
dos
tres
uno
uno
rogomez@armagnac:11> uniq ordena
cuatro
tres
dos
uno
rogomez@armagnac:11>  uniq -u ordena
cuatro
tres
dos
rogomez@armagnac:11>  uniq -d ord
uno
rogomez@armagnac:11> uniq -c ordena
1 cuatro
1 dos
1 tres
2 uno
rogomez@armagnac:11>
```

9.9 El comando *tr*

Descripción: traduce o borra caracteres

Sintaxis:

```
tr []... CONJUNTO1 [CONJUNTO2]
```

Opciones:

`-n` imprime las n primeras líneas (default 10 primeras)

Ejemplos:

```
rogomez@armagnac:67> cat prueba
Esto es una PRUEBA
rogomez@armagnac:67>  tr ''a-z'' ''A-Z'' < prueba
ESTO ES UNA PRUEBA
rogomez@armagnac:67>  tr ''A-Z'' ''a-z'' < prueba
esto es una prueba
rogomez@armagnac:67>  cat toto
No es lo mismo @ que *
rogomez@armagnac:67>  tr '@'  '*i' < toto
No es lo mismo * que *
rogomez@armagnac:67>  tr '*'  '@' < toto
No es lo mismo @ que @
```

```
rogomez@armagnac:67>
```

10 Comandos de comparación de archivos

10.1 El comando *comm*

Descripción: su salida se produce en 3 columnas. La primera contiene las líneas únicas del primer archivo, la segunda las únicas del segundo archivo y la tercera las líneas comunes a los dos archivos. Es importante tomar en cuenta que el comando asume que las líneas que componen los archivos estén ordenadas alfabéticamente, o en la misma posición dentro del archivo

Sintaxis:

```
comm
```

Opciones:

- `-1` indican que no visualizen la primera columna.
- `-2` indican que no visualizen la segunda columna
- `-3` indican que no visualizen la tercera columna.

Ejemplo:
Se consideran dos archivos, `coches1` y `coches2`, los cuales no estan ordenados.

```
rogomez@armagnac:351> more coches1
Renault
Peagout
Lamborgini
Ford
Ferrari
BMW
Mercedes
rogomez@armagnac:351> more coches2
Feat
Cooper
Peagout
Lamborgini
Chevrolet
Chysler
Tzuru
BMW
rogomez@armagnac:351> comm coches1 coches2
        Feat
        Cooper
        Peagout
        Lamborgini
        Chevrolet
        Chysler
Renault
Peagout
Lamborgini
Ford
Ferrari
BMW
Mercedes
        Tzuru
```

```
        BMW
rogomez@armagnac:351>
```

Se ordenan los archivos y después se ejecuta el comando `comm`

```
$ sort coches1 > c1
$ sort coches2 > c2
$ more c1
BMW
Ferrari
Ford
Lamborgini
Mercedez
Peagout
Renault
$ more c2
BMW
Chevrolet
Chysler
Cooper
Feat
Lamborgini
Peagout
Tzuru
$ comm c1 c2
                BMW
            Chevrolet
            Chysler
            Cooper
            Feat
Ferrari
Ford
                Lamborgini
Mercedez
                Peagout
Renault
            Tzuru
$
```

10.2 El comando *cmp*

Descripción: compara dos archivos e indica, si la hay, el lugar donde se produce la primera diferencia (número de carcter o byte, y la línea de la diferencia)

Sintaxis:

```
cmp [opciones]
```

Ejemplo:

```
$ more n1
uno
dos
tres
cuatro
cinco
$ more n2
uno
```

```
dos
tres
quatre
cinq
$ cmp n1 n2
n1 n2 differ: byte 14, line 4
$ cat a.fil
Este archivo con tres lneas.
Es casi igual al otro archivo,
pero alguna palabra es diferente.
$ cat b.fil
Este archivo con tres lneas.
Es casi igual al otro archivo,
pero alguna palabra es cambiada.
$ cmp a.fil b.fil
a.fil b.fil differ: char99, line 3
$
```

10.3 El comando *diff*

Descripción: compara dos archivos línea a línea e imprime el resultado en la salida estándar en un formato específico. Sigue el formato de los comandos del editor `vi` para igualar archivos Comando regresa lo que hay que hacerle al primer archivo para que sea igual que el segundo archivo. `C` se añade esta línea `A` se elimina esta línea `D`

Sintaxis:

```
diff
```

Opciones:

`-u` salida para usar con comando patch

Ejemplo:

```
rogomez@armagnac:56> more frutas1
manzana
naranja
nuez
rogomez@armagnac:57>  more frutas2
manzana
naranja
uva
rogomez@armagnac:58>  more frutas 3
naranja
nuez
melon
rogomez@armagnac:59>  diff frutas1 frutas2
3c3
< nuez
---------
> uva
rogomez@armagnac:60>  diff -e frutas1 frutas2
3c
uva
.
```

```
rogomez@armagnac:61>  diff frutas1 frutas3
1d0
< manzana
3a3
> melon
rogomez@armagnac:62>  diff -e frutas1 frutas3
3a
melon
.
1d
rogomez@armagnac:63>
```

10.4 El comando *patch*

Descripción: El comando permite actualizar un archivo de acuerdo a un conjunto de cambios generado por el comando `diff`.

Sintaxis:

```
patch [opciones] [archivo original
```

Opciones:

- `-b` permite definir un respaldo
- `-r` reestablece los archivos a un estado anterior

Ejemplo:

```
rogomez@armagnac:72> more a1
linea 1
linea 2
linea 3
rogomez@armagnac:73> more a2
linea 1
linea 3
linea 4
linea 5
rogomez@armagnac:74> diff -u a1 a2
--- a1  2008-10-30 15:56:19.000000000 -0600
+++ a2  2008-10-30 15:58:11.000000000 -0600
@@ -1,3 +1,4 @@
 linea 1
-linea 2
 linea 3
+linea 4
+linea 5
rogomez@armagnac:75> diff -u a1 a2 > parche
rogomez@armagnac:76>  patch < parche
patching file a1
rogomez@localhost CmdsUnix]$ more a1
linea 1
linea 3
linea 4
linea 5
rogomez@armagnac:77> more a2
linea 1
```

```
linea 3
linea 4
linea 5
rogomez@armagnac:78>
```

11 Comandos de manejo de disco

11.1 El comando *du*

Descripción: despliega el uso del disco de cada archivo y es recursivo para directorios

Sintaxis:

```
du [opcion] ... [archivo]...
```

Opciones:

- `-a` despliega todos los archivos, no solo los directorios
- `-c` al final despliega el total de lo desplegado
- `-h` despliega en un formato comprensible por el usuario

Ejemplo:

```
n{verbatim}
rogomez@armagnac:25> ls -F
a1  hello  hola  armagnac:/
rogomez@armagnac:25> du
4       ./armagnac:/cachafas
16      ./armagnac:
28      .
rogomez@armagnac:25> du -h
4.0K    ./armagnac:/cachafas
16K     ./armagnac:
28K     .
rogomez@armagnac:25> du -a
4       ./hello
4       ./hola
4       ./armagnac:/hello
4       ./armagnac:/cachafas
0       ./armagnac:/f1
4       ./armagnac:/f2
16      ./armagnac:
0       ./a1
28      .
rogomez@armagnac:25> du h*
4       hello
4       hola
rogomez@armagnac:25> du -c h*
4       hello
4       hola
8       total
rogomez@armagnac:25>
```

11.2 El comando *df*

Descripción: reporta el espacio usado y disponible de todos los sistemas de archivos montados.

Sintaxis:

```
df [opcion] ... [archivo]...
```

Opciones:

- `-a` despliega todos los archivos, no solo los directorios
- `-c` al final despliega el total de lo desplegado
- `-h` despliega en un formato comprensible por el usuario

Ejemplo:

```
rogomez@armagnac:25>  df
Filesystem           1K-blocks      Used Available Use% Mounted on
/dev/mapper/VolGroup00-LogVol00
                       7459216   4182608   2891584  60% /
/dev/sda1               101086     10159     85708  11% /boot
/dev/shm                127808         0    127808   0% /dev/shm
rogomez@armagnac:25>  df -h
Filesystem            Size  Used Avail Use% Mounted on
/dev/mapper/VolGroup00-LogVol00
                      7.2G  4.0G  2.8G  60% /
/dev/sda1              99M   10M   84M  11% /boot
/dev/shm              125M     0  125M   0% /dev/shm
rogomez@armagnac:25>  df -i
Filesystem            Inodes   IUsed   IFree IUse% Mounted on
/dev/mapper/VolGroup00-LogVol00
                     1925760  217245 1708515   12% /
/dev/sda1              26104      33   26071    1% /boot
/dev/shm               31952       1   31951    1% /dev/shm
rogomez@armagnac:25>
```

11.3 El comando *fdisk*

Descripción: utilería de manejo de particiones

Sintaxis:

```
fdisk [-u]  particion
```

Opciones:

- `u` despliega particiones en lugar de cilindros

Operadores:
Una vez invocado se despliga un "prompt" (`(Command (m for help):`). Es posible introducir cualquiera de las siguientes opciones:

Comando	Función
`b`	entra en modo linea de comandos de etiquetas de disco BSD
`m`	muestra la ayuda
`p`	despliega la tabla de particiones actual
`d`	borra una partición
`n`	crea una nuvea partición
`w`	escribe la tabal de particiones en el disco
`t`	establece el tipo de partición
`v`	verifica la particón
`L`	muestra la lista de tipos de particiones
`q`	sale de fdisk

Ejemplo:

```
root@armagnac:11> fdisk /dev/sda1
Command (m for help): m
Command action
   a   toggle a bootable flag
   b   edit bsd disklabel
   c   toggle the dos compatibility flag
   d   delete a partition
   l   list known partition types
   m   print this menu
   n   add a new partition
   o   create a new empty DOS partition table
   p   print the partition table
   q   quit without saving changes
   s   create a new empty Sun disklabel
   t   change a partition's system id
   u   change display/entry units
   v   verify the partition table
   w   write table to disk and exit
   x   extra functionality (experts only)

Command (m for help): p

Disk /dev/sda1: 106 MB, 106896384 bytes
255 heads, 63 sectors/track, 12 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

     Device Boot      Start         End      Blocks   Id  System

Command (m for help): L

0   Empty           1e  Hidden W95 FAT1 80  Old Minix       be  Solaris boot
1   FAT12           24  NEC DOS         81  Minix / old Lin bf  Solaris
2   XENIX root      39  Plan 9          82  Linux swap / So c1  DRDOS/sec (FAT-
3   XENIX usr       3c  PartitionMagic  83  Linux           c4  DRDOS/sec (FAT-
4   FAT16 <32M      40  Venix 80286     84  OS/2 hidden C:  c6  DRDOS/sec (FAT-
5   Extended        41  PPC PReP Boot   85  Linux extended  c7  Syrinx
6   FAT16           42  SFS             86  NTFS volume set da  Non-FS data
7   HPFS/NTFS       4d  QNX4.x          87  NTFS volume set db  CP/M / CTOS / .
8   AIX             4e  QNX4.x 2nd part 88  Linux plaintext de  Dell Utility
9   AIX bootable    4f  QNX4.x 3rd part 8e  Linux LVM       df  BootIt
a   OS/2 Boot Manag 50  OnTrack DM      93  Amoeba          e1  DOS access
b   W95 FAT32       51  OnTrack DM6 Aux 94  Amoeba BBT      e3  DOS R/O
c   W95 FAT32 (LBA) 52  CP/M            9f  BSD/OS          e4  SpeedStor
e   W95 FAT16 (LBA) 53  OnTrack DM6 Aux a0  IBM Thinkpad hi eb  BeOS fs
f   W95 Ext'd (LBA) 54  OnTrackDM6      a5  FreeBSD         ee  EFI GPT
10  OPUS            55  EZ-Drive        a6  OpenBSD         ef  EFI (FAT-12/16/
11  Hidden FAT12    56  Golden Bow      a7  NeXTSTEP        f0  Linux/PA-RISC b
12  Compaq diagnost 5c  Priam Edisk     a8  Darwin UFS      f1  SpeedStor
14  Hidden FAT16 <3 61  SpeedStor       a9  NetBSD          f4  SpeedStor
16  Hidden FAT16    63  GNU HURD or Sys ab  Darwin boot     f2  DOS secondary
17  Hidden HPFS/NTF 64  Novell Netware  b7  BSDI fs         fd  Linux raid auto
18  AST SmartSleep  65  Novell Netware  b8  BSDI swap       fe  LANstep
1b  Hidden W95 FAT3 70  DiskSecure Mult bb  Boot Wizard hid ff  BBT

Command (m for help): v
208781 unallocated sectors
Command (m for help): q
root@armagnac:12>
```

11.4 El comando *mkfs*

Descripción: utilería para la construcción de un sistema de archivos sobre un dispositivo, generalmente una partición de disco.

Sintaxis:

```
mkfs [-t sistema_archivos]  particion
```

Opciones:

- `V` despliega todos los comandos ejecutados.
- `t` sistema de archivos construir

Ejemplo

```
root@armagnac:33> mkfs -t ext3 /dev/hda1
```

11.5 El comando *fsck*

Descripción: utilería para verificar y reparar sistemas de archivos

Sintaxis:

```
fdisk [-u]  particion
```

Opciones:

- `t` especifica sistema archivos a verificar
- `N` no ejecuta, solo muestra lo que se va a hacer

Valores de regreso:
El código de salida del comando es la suma de alguna de las siguientes condiciones:

Comando	Función
0	sin errores
1	errores sistema archivos corregidos
2	sistema de archivos debe ser reinicializado
4	se dejaron errores en sistema archivos sin corregir
8	error operacional
16	error de sintaxis o uso
32	comando fsck cancelado por el usuario
128	error en biblioteca compartida

Ejemplo:

```
root@armagnac:81> fsck -N /dev/sda2
fsck 1.32 (09-Nov-2002)
[/sbin/fsck.ext3 (1) -- /] fsck.ext3 /dev/sda2
root@armagnac:82>
```

12 Los programas en red

Con las versiones BSD4.x de Unix el acceso a una red local es posible. El objetivo es que el usuario pueda acceder una máquina a partir de otra, con el fin de transferir datos a una gran velocidad.

Las principales aplicaciones son:

- Transferir archivos
- Tener una terminal virtual
- Ejecución, sobre una máquina, de programas a distancia

A continuación se describen los principales protocolos/comandos usados en máquinas Unix conectadas por una red local.

12.1 El protocolo *telnet*

Permite conectarse a otro sistema (no necesariamente Unix) y dialogar con ese sistema como si tuviéramos una terminal conectada directamente a él.

La sintaxis del protoccolo es:

```
telnet [ host ]
```

Una vez conectados, y después de presionar las teclas `<crtl> <]>` , se pasa al modo comandos de telnet. Este modo permite enviar caracteres especiales al sistema distante, de cerrar la conexión, de abrir una nueva, o de salirse de telnet

Los principales comandos bajo este modo son:

`?`	lista los comandos de telnet
`open`	abre una conexión
`close`	cierra la conexión en curso
`quit`	sale de telnet, cerrando la conexión
`send car`	envía un carácter especial al sitio distante
`send ?`	lista los cracteres especiales y su efecto

12.2 El protocolo *ftp* (file transfer protocol)

Permite conectarse a otro sistema distante, con el fin de transferir archivos. Es posible hacerlo en ambos sentidos, ya sea dejar archivos en la máquina remota o traerse archivos de la máquina remota.

Permite conectarse a computadoras que manejan un sistema diferente a Unix.

```
ftp [ host ]
```

Los principales comandos de `ftp` son:

`?`	lista los comandos de ftp
`!`	lanza un shell sobre el sistema local
`bye`	termina la sesión ftp
`cd direc`	cambiar directorio en sistema distante
`lcd direc`	cambiar de directorio en sistema local
`put arch`	envía el archivo arch1, que se llamará arch2 en el sistema distante. Un sinónimo de put es send
`get arch1`	recibe el archivo arch1, que se llamará arch2 en el sistema local. Un sinonimo

de get es recv

`mget` utilizado para recibir archivos utlizando el metacaracter `*`

`mput` permite enviar y recibir archivos utilizando el metacaracter `*`. Los archivos conservarán su mismo nombre en ambos sistemas

`prompt` elimina la opción de pregunta interactiva de `mget` y `mput`

12.3 Los comandos *r*

Este es un conjunto de comandos que permiten realizar cierto tipo de operaciones remotas entre dos máquinas que esten ejecutando un sistema operativo Unix. Con el fin de protegerse de posibles ejecuciones no deseadas, si el usuario *toto* de la máquina *A* desea ejecutar un comando en la máquina *B* se deben cumplir las siguientes condiciones:

- El usuario *toto* debe de tener una cuenta en la máquina *B*. Normalmente se tiene el mismo nombre de cuenta en ambas máquinas (*toto*)
- El archivo `/etc/host.equiv` de la máquina *B* debe tener una entrada para *A* o en su defecto el directorio hogar[3] de *toto* debe contener un archivo llamada `.rhosts` que contenga una entrada para *tequila*.

En muchos sistemas el archivo `.rhosts` es creado con una sola entrada, un caracter `+` lo cual le otorga permiso a todo mundo de hacer lo que sea en la máquina. Se recomienda eliminar dicho archivo o revisar periódicamente su contenido para evitar otorgarle permisos innecesarios a personas desconocidas o no deseadas.

Existen varios comandos que funcionan bajo este contexto, a continuación se explicarán los más importantes de ellos.

12.3.1 EL *rlogin* (remote login)

Permite conectarse a otro sistema Unix, de la misma forma que `telnet` . Su sintaxis es:

```
rlogin [ -l nombre ] host
```

Si no se utiliza la opción `-l`, `rlogin` conectará al usuario a la máquina distante con el mismo nombre que tiene en la máquina local. Los valores de las variables de ambiente `USER` y `TERM` son pasadas al programa `login` de la computadora distante.

Las peticiones de `rlogin` pueden estar precedidas del caracter `~` (tilde) y solo son efectivas si son el primer caracter de una línea, (después de un `<RETURN>`):

- `~.` cierra la conexión
- `~<crl><z>` suspende la conexión
- `~~` envía un `~`

Este comando, como todos el resto de los comandos-r no funciona si alguna de las dos máquinas no trabaja bajo el sistema Unix.

[3]directorio en el cual el usuario es posicionado cuando entra por primera vez al sistema (conocido también como directorio HOME).

12.3.2 El *rsh* (remote shell)

Permite ejecutar un comando sobre otra máquina Unix. Los archivos de entrada/salida estándar están asociados a la terminal, sin embargo no se aconseja utilizar rsh para ejecutar comandos interactivos distantes.

Su sintaxis es:

```
rsh host [ -l usuario ] [ comando ]
```

Si no se especifica el comando, entonces el usuario se conectará al sistema distante como si hubiera tecleado un `rlogin`.

Hay que tener cuidado con las redirecciones:

- `rgomez@cognac>rsh amenti ls > res.txt` crea un archivo `res.txt` local
- `rgomez@cognac>rsh amenti "ls > res.txt"` crea un archvio en la máquina `amenti`

Si el usuario no tiene el archivo `.rhosts` entonces se le pedirá su password. Lo mismo ocurre si en ese archivo no se le otorga la autorización de conexión a la máquina desde la cual se está ejecutando el `rsh`.

12.3.3 El *rcp* (remote copy)

Permite copiar archivos de una máquina a otra. Es imperativamente necesario tener un archivo `.rhosts` en la máquina distante que autorize al usuario a conectarse

La sintaxis del copiado remoto es:

```
rcp arch1 arch2
rcp [ -r ] archivo [ archivos ] directorio
```

donde `arch1` y `arch2` pueden tomar la forma `maquina:pathname`. Esta forma significa que el archivo se encuentra en el camino de acceso `pathname`, de la `maquina`. Lo mismo se aplica para los argumentos `directorio` y `archivo` en la segunda sintaxis. La opción `-r` permite especificar un directorio y de copiar recursivamente toda la sub-jerarquía que se encuentra en ese directorio.

Algunos ejemplos de este comando se presentan a continuación:

```
rogomez@svarga>rcp amenti: .login
rogomez@svarga>rcp eden:bin/arch1 svarga:bin
rogomez@svarga>rcp eden:bin/arch1 walhalla:bin/arch2
rogomez@svarga>rcp -r src empyree:src
```

13 Comandos relacionados con impresiones

Una de las actividades más comunes que realiza un usuario es la impresión de documentos. Es posible enviar a imprimir un documento directamente de la aplicación o utilizando algunos de los comandos que Unix proporciona para ello. Los siguientes comandos sirven para el control de las impresiones.

13.1 El comando *lpr* (line printer)

Descripción: el principal comando de impresión. Crea un trabajo de impresora en un área de spooling para una impresión subsecuente (un trabajo de impresión se divide en un archivo de control y otro de datos)

Sintaxis:

```
lpr [ opciones  ]  [ archivos ]
```

Opciones:

`-P dest` para elegir la impresora

`-# n` para obtener n copias

Ejemplo:

```
rogomez@armagnac:43> lpr abc
rogomez@armaganc:44> lpr -Pbali prog1.c results.txt
rogomez@armagnac:45>
```

13.2 El comando *a2ps*

Descripción: imprime un archivo ASCII en formato postcript

Sintaxis:

```
a2ps [ opciones ]  [ archivos ]
```

Opciones:

`-P` dest + para elegir impresora

`-#n` para obtener n ejemplares

`-1` imprime una página por hoja

`-l` imprime en modo landscape

`-p` imprime en modo portrait

`-n` despliega el número de líneas

Ejemplo:

```
rogomez@armagnac:809>a2ps numeros
[numeros (plain): 1 page on 1 sheet]
request id is CC-723 (1 file)
[Total: 1 page on 1 sheet] sent to the default printer
rogomez@armagnac:810>
```

13.3 El comando *lpq*

Descripción: permite ver el estado de las colas de espera de impresión

Sintaxis:

```
lpq [ opcion  ] [ usuario ]
```

Opciones:

`-P` dest para escoger la impresora

`-l` formato largo

Ejemplo:

```
rogomez@armagnac:810> lpq
lp is ready and printing
Rank Owner Job File Total Size
active  root    201     /etc/passwd     350 bytes
1st     toto    202     abc             546 bytes
rogomez@armagnac:811>
```

13.4 El comando *lprm* (line printer remove)

Descripción: permite suprimir los archivos en espera de ser impresos.

Sintaxis:

```
lprm [ opciones ]  [ #job]  [usuarios]
```

Opciones:

`-P dest`	para escoger la cola de espera
`-`	suprime todos los archivos del usuario
`job#`	borra el archivo que corresponde a ese número

Ejemplo:

```
rogomez@armagnac:810> lprm 202
dfA202sioux dequeued
cfA202sioux dequeued
rogomez@armagnac:811>
```

14 Resumen comandos Unix

La tabla de abajo presenta un resumen de los principales coandos Unix;

`awk`	busca y procesa patrones en un archivo
`cat`	concatena o despliega archivos
`comm`	compara archivos buscados
`cp`	copia archivos
`cpio`	almacena y extrae archivos en un formato archival
`diff`	despliega las diferencias entre dos archivos
`find`	encuentra archivos
`grep`	busca patrones de caracteres en archivos
`head`	despliega el encabezado de un archivo
`ln`	crea una liga a un archivo
`lpr`	imprime archivos
`ls`	despliega información sobre archivos
`mkdir`	crea un directorio
`more`	despliega un archivo por pantalla
`mv`	renombra un archivo
`od`	vacía un archivo
`pr`	hace paginación a un archivo
`rcp`	copia archivos desde o en una computadora remota
`rm`	remueve un archivo
`rmdir`	remueve un directorio
`sed`	editor stream
`sort`	busca y fusiona archivos
`spell`	checa errores ortográficos en un archivo
`tail`	despliega la última parte de un archivo
`tar`	almacena o extrae archivos de un archivo archival
`uniq`	despliega líneas de un archivo que son únicas
`wc`	despliega número de líneas, palabras y caracteres
`ftp`	transfiere archivos a través de la red
`mail`	manda o recibe correo electrónico
`mesg`	activa/desactiva la recepción de mensajes
`telnet`	se conecta a una computadora remota a través de la red
`write`	manda un mensaje a otro usuario
`cd`	cambia a otro directorio de trabajo
`chgrp`	cambia el grupo que está asociado con un archivo
`chmod`	cambia el modo de acceso de un archivo
`chown`	cambia el propietario de una clase
`date`	despliega la fecha y la hora
`df`	despliega la cantidad disponible del disco duro
`du`	despliega información del uso del disco
`file`	despliega clasificación de archivos
`finger`	despliega información detallada de usuarios
`kill`	termina un proceso
`nice`	cambia la prioridad de un comando
`nohup`	corre un comando que se mantendrá corriendo después de salir del programa
`ps`	despliega status de procesos
`ruptime`	despliega el status de computadoras conectadas a la red
`rwho`	despliega nombres de usuarios de computadoras conectadas a la red

`sleep`	proceso que duerme por un intervalo específico
`stty`	despliega o determina parámetros terminales
`umask`	determina una máscara de permisos para la creaciónn de archivos
`w`	despliega información de los usuarios del sistema
`who`	despliega nombres de usuarios
`cc`	compilador de C
`make`	guarda la concurrencia de los programas
`touch`	actualiza el tiempo de modificación de archivos
`admin`	crea o cambia las características de un archivo SCCS
`ci`	crea o guarda cambios en un archivo RCS
`co`	extrae una versión sin codificar de un archivo RCS
`delta`	guarda cambios en un archivo SCCS
`get`	crea una versión sin codificar de un archivo SCCS
`prs`	imprime la historia de un archivo SCCS
`rcs`	crea o cambia las características de un archivo RCS
`rlog`	imprime un sumario de la historia de un archivo RCS
`rmdel`	remueve un delta de un archivo SCCS
`at`	ejecuta un shell script a un determinado tiempo
`cal`	despliega un calendario
`calendar`	presenta recordatorios
`crontab`	programa un comando para que se corra a determinada hora
`echo`	despliega un mensaje
`expr`	evalúa una expresión
`fsck`	checa y repara filesystems
`rlogin`	entra a una computadora remota
`tee`	copia la entrada estándar a la salida estándar y a uno o más archivos
`tr`	reemplaza caracteres específicos
`tty`	despliega el camino a la terminal