Comandos básicos de Unix

Dr. Roberto Gómez Cárdenas
ITESM-CEM Dpto. Ciencias Computacionales*

16 de agosto de 2000

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. El documento sólo describe parte de lo desplegado por dicho comando. 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 origenes 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. Pero Multics era grande, lento y requería recursos esenciales de la computadora. Thompson encontró una computadora más pequeña adonde transfirió el programa Space Travel para ejecutarlo en él. 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 operatio adaptó algunos de los conceptos avanzadaos de Multics. Existían ya otros sistemas operativos distintos de Multics que tenían más o menos las mismas capacidades y Unix se aprovechó del trabajo que se había 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 al 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 de 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 máquina y que solo trabajara en una computadora (o una familia de computadoras). Por lo que transferir Unix de una computadora a otro requería una reescritura importante de los programas.

*Disponible como Reporte Interno No. ITESMCEM-DCC-2000-1

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 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 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 lo que es el nacimiento, en 1975 de la versión 6 de los laboratorios AT&T de los Laboratorios Bell. Después de la presentación de esta versión 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. Algunas de estos nuevos mecanismos fueron introducidos en els 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 results 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 provoco 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 susrena V aportando modificaciones menores y/o más importantes. La mayor parte de los 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 XENIX

Xenix es la primera versión de Unix diseñada para microcomputadoras, aún es utilizada. Esta versión proviene de la versión 7 y ha sido convertido progresivamente en un sistema V versi'on 2.

[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.

XENIX influenció Sistema V versión 3, la mayor parte de estas funciones fueron incorparados en el Sistema V versión 3.2

2.2 El sistema OSF/1

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

Entre los estandares 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.

2.3 El sistema SCO Unix

Este nombre hace referencia a SCO Open Desktop y SCO Open Server Release 3 producidos por Santa Cruz Operations Inc. (que funciona sobre procesadores 486). Este sistema operativo es una implmentación de V.3.2.5.

2.4 El sistema SunOS

Es el sistema operativo de tipo BSD más conocido que ha introducido, en el mundo Unix, funcionalidades importantes (entre la más importante esta NFS). Sun a querido reemplazar SunOS por Solaris pero ha cedido a la presión de los usuarios: Sun continua proporcionando los dos sistemas operativos.

2.5 El sistema Solaris

Es una implementación del sistema V.4 propuesto por Sun. Hay que mencionar que Solaris 2.x a veces es denominado SunOS 5.x.

2.6 El sistema HP-UX

Es la versión de Unix de Hewlett-Packard que sigue las características del Sistema V incorporando varias características de OSF/1[2]. HP-UX ha sido consideraablemente 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.

2.7 El sistema DEC OSF/1

La versión OSF/1 de Digital Equipment Corporation se parece en gran medida a un sistema BSD generica del 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.

2.8 El sistema IRIX

Las primeras versiones de IRIX incorporan numerosas características de BSD pero estas han desaparecido en el transcurso del tiempo a favor de una conformidad a V.4.

[2] OSF: Open Software Foundation.

2.9 El sistema AIX

El sistema operativo de IBM de tipo Sistema V, también ofrece diferentes funcionalidades de V.4, BSD y OSF/1 (además de las inevitables características propias a IBM).

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 Linux Torvaldas, (Linux es el Unix de Linus, Linus Unix) aunque otras personas han contribuido (y contribuyen) a su desarrollo. Linux es globalmente de tipo BSD. Tecnicamente, el nombre de Linux hace referencia al corazón del sistema operativo (el núcleo y algunos controladores de periféricos) pero eso nombre también se aplica al software de dominio público, donde las fuentes son de origen variado, que constituyen una *distribución*.

2.11 El sistema Minix

Minix es un sistema operativo desarrollado por Andrew Tanenbaum con fines pedagogicos. Pensado en un principio para ser ejecutado a partir de discos flexibles, en una PC compatible.

Minxi fue la fuente de inspiración de Linus para desarrollar el sistema operativo Linux.

2.12 El sistema FreeBSD

FreeBSD es un sistema operativo Unix BSD avanzado para arquitecturas Intel (x86), DEC Alpha y PC-98. Es atendido por un gran equipo de personas repartidas en todo el mundo.

2.13 El sistema OpenBSD

El proyecto OpenBSD produce una multiplatoforma libre del sistema operativo Unix 4.4 BSD. Los esfuerzos de los integrantes del proyecto van dirigidos a reforzar la portabilidad, estandarización, seguridad, "correctness" 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.

2.14 El sistema BSD/OS

El núcleo de BSD/OS esta inspirado del núcleo del sistema operativo 4.4 BSD de la Universidad de California Berkeley, con mejorías de BSDi. Es una plataforma de red cliente/servidor rápida, escalable y que soporta multitasking. Cuenta con una pequeña huella, memoria virtual (opcional) y memoria 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 esta 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.

El shell se compone principalmente de la líena de comandos. El shell es el encargado de interpretar lo que el usuario desea hacer y, si es posible, de llevarlo a cabo. En caso de que no sea posible despliega un mensaje de error.

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

Una forma de ilustrar como trabajan las partes anteriores 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 (conocidad 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 usurio teclee más comandos.

El presente documento esta enfocado a lo que son las dos últimas partes. Por un lado se explica todo lo relacionado con el shell y por el otro se da una lista de los comandos más importantes de lo que es el 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).

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. su matrícula). 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. Está formado 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 traves 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: rgomez
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: rgomez
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 le sigue negando el acceso al usuario la máquina se apaga o el sistema se desactiva.

Si todo se 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.

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 palabra 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 y el korne-shell.

La línea de comandos empieza en el *prompt*. Por default el prompt es representado por un caracter aunque esto puede ser modificado por el usuario. Por ejemplo el prompt por default 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.

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 esta listo para ejecutar otro comando.

Para poder capturar una línea de comando muy larga, es posible de 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. Ahora bien, 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 default, si se quieren utilizar estas opciones se tiene que 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 que hacer, las opciones como hacerlo y los argumentos sobre quien 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 que no se de ninguna se toma la opción por default. Generalmente estan precedidas de un caracter `-` (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 sistema operativos, no 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 de los casos se hará lo que se el comando indique. En el segundo ejemplo mientras que el segundo no será reconocido por el sistema y desplegará el mensaje de error correspondiente a este echo: `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 tiene que realizar 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.

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 los 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, y por redirección de la 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 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 los archivos `arch1` y `arch2` serán copiados una 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 la 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 de dicho 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:

```
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 enviarla 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  [ ] [ ]
```

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.

4 Comandos relacionados con archivos

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 1).

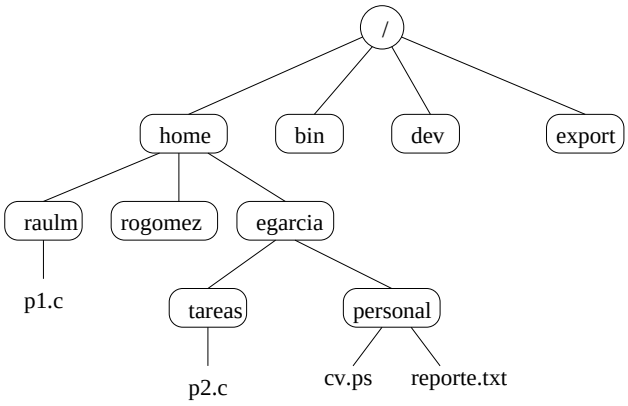

Figura 1: *Los diferentes pasos para comunicar dos procesos a través de un pipe*

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 ls

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

Sintaxis:

```
ls [opciones]
```

Algunas opciones:

- `-a` lista los archivos que comienzan con un "."
- `-l` listado en formato largo
- `-d` si el argumento es un directorio lista el nombre del archivo directorio y no su contenido
- `-s` da el tamaño de los archivos en kilo-bytes
- `-u` despliega la última hora de acceso en lugar de la última hora de modificación
- `-t` acomoda los archivos en función de la hora de la última modificación

`-i` imprime el número de referencia (i-node) de los archivos

`-C` lista los archivos en columnas

`-g` muestra el propietario del grupo de un archivo en un formato largo

Ejemplo:

```
rogomez@armagnac:45>ls -l
total 4
d r w x r - x r - x  1 cachafas      512 Oct 12  10:13 Sundraw
d r w x r - x r - x  1 cachafas      512 Dec 11  20:13 Sunpaint
- r w x r - x r - x  1 cachafas      512 Sep 15  18:13 toto
- r w - r - x - - x  1 cachafas      512 Jan 12  01:14 curso.html
rogomez@armagnac:46>
```

4.2 El comando *more*

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

Sintaxis:

```
more nombre-archivo
```

Ejemplo:

```
rogomez@armagnac:201>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:202>
```

4.3 EL comando *cat*

Descripción: 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:207>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:208>
```

4.4 El comando *pwd*, print working directory

Descripción: despliega el camino de acceso del directorio actual (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:61>pwd
/home/dic/rogomez/Articulos
rogomez@armagnac:62>
```

4.5 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, cuando se entro al sistema. Dando como parámetro `..` remonta en el directorio del padre.

Sintaxis:

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

Ejemplo:

```
rogomez@armagnac:210>cd Cursos/
/home/rogomez/Cursos
rogomez@armagnac:211>cd ..
/home/rogomez
rogomez@armagnac:212>
```

4.6 El comando *rm* (remove)

Descripción: borra el nombre de un archivo o, si ese nombre fuera el último (el numero 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)
- `-i` interactivo, (pregunta)

Ejemplo:

```
rogomez@armagnac:12>rm -i toto.txt
rm: remove toto.txt (yes/no)? y
rogomez@armagnac:13>rm prog.c
rogomez@armagnac:14>rm -i Tareas/
rogomez@armagnac:15>
```

4.7 El comando *mkdir*

Descripción: utilizado en la creación de directorios.

Sintaxis:

```
mkdir directorio  [ directorio ]
```

Ejemplos:

```
rogomez@armagnac:525>mkdir /usr/usr2/alumno1
rogomez@armagnac:526>mkdir test direc1 rep2
rogomez@armagnac:527>mkdir arbo arbo/rep1
rogomez@armagnac:528>
```

4.8 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:453> rmdir Tareas
rogomez@armagnac:454> rmdir Proyectos
rogomez@armagnac:455>rmdir Temporal
rmdir: directory "Temporal": Directory not empty
rogomez@armagnac:456>rm Temporal/*
rogomez@armagnac:457>
```

4.9 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:231> ls -lg e1
- r w - r w - r w -  1    toto       daemon       0 Oct 12 18:20  e1
rogomez@armagnac:232> chmod 755 e1
rogomez@armagnac:233> ls -lg e1
- r w x r - x r - x  1    toto       daemon       0 Oct 12 18:20  e1
rogomez@armagnac:234> chmod a-x e1
rogomez@armagnac:235> ls -lg e1
- r w - r - - r - -  1    toto       daemon       0 Oct 12 18:20  e1
rogomez@armagnac:236> chmod g+x  e1
rogomez@armagnac:237> chmod o-r  e1
rogomez@armagnac:238> ls -lg e1
- r w - r - x - - -  1    toto       daemon       0 Oct 12 18:20  e1
rogomez@armagnac:239>
```

4.10 El comando *cp*

Descripción: copia un archivo ordinario

Sintaxis:

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

Ejemplos:

```
rogomez@armagnac:239>cp arch1 arch2
rogomez@armagnac:240>cp arch1 direc
rogomez@armagnac:241>cp arch1 direc/arch2
rogomez@armagnac:242>cp arch1 arch2 arch3 direc
rogomez@armagnac:243>
```

Otros:

- `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

4.11 El comando *mv*, move

Descripción: desplaza un archivo o lo renombra

Sintaxis:

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

Ejemplos:

```
rogomez@armagnac:244>mv arch-a arch-b
rogomez@armagnac:245>mv direc1 direc2
rogomez@armagnac:246>mv arch1 arch2 arch3 arch4 direc
rogomez@armagnac:247>mv arch1 ../bin/fx
rogomez@armagnac:248>
```

4.12 El comando *ln*, link

Descripción: crea un nuevo nombre para un archivo, útil para archivos sistemas, evitar copiar dos archivos con el mismo nombre (ahorrar espacio). Solamente con la opción -s se puede cambiar el nombre a un directorio

Sintaxis:

```
ln archivo nuevo-nombre
ln -s  archivo otro-nombre
```

Ejemplos:

```
rogomez@armagnac:248> ls -li /* listado largo con i-node */
total 2
  14411 -rw-r--r-- 1 rgomez    412 Oct 10 16:56 arch1
  14412 -rw-r--r-- 1 rgomez    167 Oct 10 16:57 archivo
rogomez@armagnac:249>ln arch1 fichier
rogomez@armagnac:250>$ ls -li
total 3
  14411 -rw-r--r-- 2 rgomez    412 Oct 10 16:56 arch1
  14411 -rw-r--r-- 2 rgomez    412 Oct 10 16:56 fichier
  14412 -rw-r--r-- 1 rgomez    167 Oct 10 16:57 archivo
rogomez@armagnac:251>
```

Otros:
En BSD 4.x es posible crear ligas simbólicas entre archivos o directorios correspondientes a sistemas de archivos diferentes. Por ejemplo:

```
rogomez@armagnac:252> ls
arch1 dir1
rogomez@armagnac:253> ln -s arch1  fichier
rogomez@armagnac:254> ls -li
total 3
  14411 -rw-r--r-- 1 rgomez   412 Oct 10 16:56 arch1
  14412 lrwxrwxrwx 1 rgomez     4 Oct 10 17:09 fichier -> arch1
  38935 drwxr-xr-x 2 rgomez   512 Oct 10 17:09 dir1
rogomez@armagnac:255> ln -s dir1 repertoire
rogomez@armagnac:256> ls -li
total 4
   14411 -rw-r--r-- 1 rgomez   412 Oct 10 16:56 arch1
   14412 lrwxrwxrwx 1 rgomez     4 Oct 10 17:09 fichier -> arch1
   38935 drwxr-xr-x 2 rgomez   512 Oct 10 17:09 dir1
   14413 lrwxrwxrwx 2 rgomez     4 Oct 10 17:10 dir1 -> repertoire
rogomez@armagnac:257>
```

4.13 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 mayusculas y minusculas
- `-n` despliega el número de línea

Ejemplos:

```
          rogomez@armagnac:15>cat agenda
          aguirre claudia 5456789
          burron regino  8719890
          gomez roberto 3218956
          gomez gabriel 3331811
          zapata adolfo 4782911
  rogomez@armagnac:16>grep gomez agenda
          gomez roberto 3218956
          gomez gabriel 3331811
          rogomez@armagnac:17>cat numeros
          uno        un
          dos        deux
          tres       trois
          cuatro     quatre
          cinco      cinc
          rogomez@armagnac:18>grep cinc numeros
          cinco      cinc
          rogomez@armagnac:19>
```

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

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

4.14 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
- `-b` ignorar los blancos en principio de línea

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>
```

4.15 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>
```

4.16 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>
```

4.17 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>
```

5 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.

5.1 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>
```

5.2 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>
```

5.3 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>
```

5.4 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>
```

6 Otros comandos

Los comandos descritos en esta sección no pudieron ser agrupados o pertenecen a un grupo muy reducido. El lector encontrará comandos para manejo de terminales procesos, del manual y otros.

6.1 El comando *passwd* password

Descripción: el comando passwd permite modificar el password de un usuario, para esto el usuario deberá de conocer el password de la cuenta que desee modificar.

Sintaxis:

```
passwd [ -l |  -y ] [ -afs]  [ -d [ usename]  ]
  [ -e username]  [- F filename]
  [ -n numdays username]  [-x numdays username]
  [ username ]
```

Ejemplo:

```
rogomez@armagnac:121>passwd
Changing password
Old password:   <antiguo password>
New password:  <nuevo password>
Re-enter new password:  <nuevo password>
rogomez@armagnac:122>
```

6.2 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:122>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
                         :
                         :
```

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

6.3 El comando *echo*

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

Sintaxis:

```
echo [ argumentos ]
```

Ejemplo:

```
rogomez@armagnac:284>echo esto es una prueba
esto es una prueba
rogomez@armagnac:285>
```

6.4 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:231>which xeyes
/home/dic/rogomez/xeyes
rogomez@armagnac:232>which opnet
opnet: Command not found
rogomez@armagnac:233>
```

6.5 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 [ [ - ] acCegjklnrSuUvwx  ]  |  [ num ]
   [ kernel name]    [ c-dump-file  ] [ swap-file ]
```

Algunas opciones:

- `-a` información de procesos "poseídos" por otros
- `-l` información en formato largo
- `-r` restringe la salida a los procesos que están "corriendo"
- `-x` incluye procesos no relacionados con la terminal en la que se tecleó el comando

Ejemplo:

```
rogomez@armagnac:233>ps -agx
 PID  TT  STAT  TIME  COMMAND
   0  ?   D 0:07 swapper
   1  ?  IW 0:00 /sbin/init
   2  ?  D 0:02 pagedaemon
  51  ?  S 0:05 portmap
1786  co  S 0:05 xclock -digital -geometry +675 +-2
2242  p0  S 0:02 xvile ejecuta.c
2331  p0  R 0:00 ps -agx
2139  p4  IW 0:00 telnet sunlab
rogomez@armagnac:234>
```

Nota:
El comando kill número-proceso permite "matar", o terminar, con la ejecución de un proceso

6.6 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:233>uname -a
SunOS mexico 4.1.3_U1 2 sun4c
rogomez@armagnac:234>
```

6.7 El comando *tty*

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

Sintaxis:

```
tty [ -l ] [ -s] ...
```

Opciones:

`-l` imprime número línea asincrónico a la cual la terminal del usuario esta conectado

`-s` imprime todos los parámetros en hexadecimal

Ejemplo

```
rogomez@armagnac:38>tty
/dev/pts/6
rogomez@armagnac:39>
```

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

6.8 El comando *date*

Descripción: despliega la fecha y la hora

Sintaxis:

```
date [-u] [format] +
```

Opciones:

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

`+format` la impresión del comando puede ser reformateada para que se entienda mejor.

Ejemplo:

```
rogomez@armagnac:39> date
Fri Mar 12 19:59:08 CST 1999
rogomez@armagnac:40> date '+DATE: %d-%n-10%y%nHEURE: %H:%M:%S'
DATE: 10-10-1988
HEURE: 16:01:47
rogomez@armagnac:41>
```

6.9 El comando *who*

Descripción: despliega los usuarios conectados.

Sintaxis:

```
who
```

Ejemplo:

```
rogomez@armagnac:41>who
rogomez console Oct 10 09:48
rogomez ttyp0 Oct 10 11:18
mimoso  ttyp1 Oct 10 12:54
rogomez@armagnac:42>
```

Nota:
Una variante es whoami que despliega información correspondiente a la persona conectada a la terminal donde se tecleo ese comando.

7 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.

7.1 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

7.2 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`

7.3 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.

7.3.1 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>`):

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

- `~.` 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.

7.3.2 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`.

7.3.3 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
```

8 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