Escuela Politécnica Superior de Elche

SISTEMAS INFORMÁTICOS EN TIEMPO REAL

2º Ingeniería Industrial

INTRODUCCIÓN AL SISTEMA OPERATIVO UNIX

Luis Miguel Jiménez
Rafael Puerto

Departamento de Ingeniería de Sistemas Industriales
Área de Ingeniería de Sistemas y Automática

A INTRODUCCIÓN A UNIX

¿Qué es Unix?

Se podría definir Unix como un sistema operativo multiusuario y de tiempo compartido. Unix es muy popular en los entornos dedicados al desarrollo de programas y a la preparación de documentos, aparte de ser prácticamente el servidor en red (web, ftp, ...) por excelencia. Actualmente, su ámbito se extiende desde los computadores personales (en cualquiera de sus versiones) hasta los grandes sistemas o mainframes.

La primera versión de Unix fue escrita por Ken Thompson en los laboratorios Bell de AT&T a finales de los años 60. Inicialmente, el sistema se desarrolló íntegramente en lenguaje ensamblador para posteriormente ser reescrito en C y adquirir prácticamente la forma en la que lo conocemos hoy. Los autores hicieron público el código del sistema por lo que alcanzó gran difusión entre la comunidad científica. En 1974 se introdujo en las universidades "con fines educacionales", y al cabo de pocos años se encontraba ya disponible para uso comercial.

En el paso de comercialización aparecieron dos líneas: System V y BSD. La primera de ellas es la original de AT&T y la segunda fue desarrollada en la universidad de Berkeley. Aunque básicamente las dos líneas comerciales de Unix partieron del mismo Unix original, poco a poco fueron distanciándose hasta resultar en dos versiones prácticamente incompatibles.

Características de Unix

Las principales características de UNIX se podrían resumir en las siguientes:

- **Portabilidad**: está todo escrito en C, con especial aislamiento de las rutinas dependientes del hardware.
- **Modularidad**: Unix está concebido de manera que estimule la descomposición de una tarea grande en módulos más pequeños, con funciones bien definidas, fáciles de poner a punto e integrar.
- **Independencia de dispositivo**: Las entradas/salidas están integradas en el sistema de ficheros. Los ficheros y los dispositivos de E/S son tratados de una manera uniforme con el mismo conjunto aplicable de llamadas al sistema.
- **Sistema de ficheros jerárquico**: sistema de árbol de directorios que permite la unión de diversos sistemas de ficheros con el sistema principal.
- **Interfaz con el usuario simple e interactiva** (a la vez que potente): el "shell" consiste en un programa independiente del núcleo del S.O. que el usuario puede manipular (sustituir, configurar,...). La sintaxis de los comandos es bastante homogénea.
- **Proporciona un entorno de programación completo:** existe una amplia gama de herramientas para el desarrollo de programas. Por ejemplo, los filtros son utilidades muy simples que se concentran en realizar muy bien una sola tarea. Pueden

combinarse de forma muy flexible (mediante tubos y redirecciones) para resolver tareas más complejas.

- **Mantenimiento y evolución fáciles:** consecuencia directa de la modularidad. El sistema sigue evolucionando y se perfecciona y enriquece con nuevas funcionalidades.

1 Presentación al sistema

Debido a que Unix es un sistema multiusuario, lo primero que se debe hacer es identificarse para que el sistema pueda responder al usuario de un modo individual. De este modo se tiene acceso a los ficheros propios y se establece la sesión de acuerdo con las preferencias que el usuario haya seleccionado. Cada usuario tiene asociado un nombre dentro del sistema a la vez que una palabra de paso o password. Cuando un usuario se conecta a una máquina Unix aparece un mensaje parecido al siguiente (depende del sistema):

```
Wellcome to Linux SuSE 6.0
Login:
```

En este momento el sistema está pidiendo al usuario que se identifique (Logging In o abreviadamente login) y que introduzca su password. Si el nombre del usuario y el password asociado son correctos se permite a dicho usuario la entrada al sistema :

```
Wellcome to Linux SuSE 6.0
Login:rpuerto
Password:

$
```

En el caso de que no se introduzca un nombre de usuario válido con un password asociado válido, aparecerá un mensaje de entrada incorrecta “Login incorrect” y se volverá a pedir la identificación.

Una vez se ha entrado en el sistema aparece el “prompt” o indicador de la línea de comandos y ya se puede empezar a trabajar con el sistema. Dependiendo del tipo de sistema, el prompt puede variar, indicando en unos casos sólo el símbolo $ u otro parecido (# está reservado para el root o superusuario), y en otros, junto con este símbolo, el nombre de la máquina, el directorio actual, etc.

Cuando iniciamos una sesión en un determinado sistema, éste nos ubica en lo que se denomina directorio HOME. Normalmente, cada usuario tiene un directorio donde almacena sus datos y aplicaciones personales. Para saber cual es nuestro directorio personal utilizaremos el comando **pwd** que muestra el directorio en el que nos encontramos en el instante actual. Por ejemplo:

```
[rpuerto]$ pwd
/home/rpuerto
[rpuerto]$
```

Obviamente, más de un usuario puede estar dentro del sistema. Para saber quien está trabajando en ese momento, una vez dentro, utilizaremos el comando **who** y obtendremos una lista de los usuarios que están actualmente en el sistema y desde donde están conectados (consola, terminal, máquina remota, etc.).

```
[rpuerto]$ who
rpuerto   tty1     Mar  8 21:16
```



```
root      tty2     Mar  8 21:28
rpuerto   :0       Mar  8 21:13 (console)
jsabater  ttyp3    Mar  8 21:14
lmjimenez ttyp4    Mar  8 21:26
[rpuerto]$
```

Este ejemplo indica que hay 5 usuarios utilizando la máquina. Aunque uno de ellos tenga dos sesiones abiertas, para el sistema cada sesión es totalmente independiente. La información adicional que se muestra es la terminal desde la que se está conectado (o dirección IP en el caso de conexión remota), y la fecha y hora de conexión.

Para salir del sistema necesitamos ejecutar el comando **exit** el cual se encarga de salvar los datos temporales que tengamos y nos expulsa del sistema.

En Unix, los usuarios se pueden distribuir el lo que se denominan *grupos*, que como su nombre indica no es más que un grupo de usuarios que tiene determinados permisos en el sistema. Además, existe un usuario especial llamado *superusuario* o *root* que es el encargado de administrar el sistema y por tanto no tiene restricciones dentro de él. En un sistema Unix, cada grupo de usuarios, e incluso cada usuario, puede tener unos privilegios distintos tal y como se verá más adelante.

2 El “shell” de Unix

Cuando se entra en el sistema, tal y como se ha comentado en el apartado anterior, aparece el “prompt” indicando que el sistema esta preparado para recibir órdenes. Cuando aparece este símbolo, no es el núcleo del sistema operativo quien se está dirigiendo al usuario, sino un intermediario llamado intérprete de comandos o “shell”. El shell es solamente un programa como cualquier otro pero capaz de hacer algunas cosas muy interesantes. El hecho de que el shell se encuentre entre el usuario y el kernel tiene muchas ventajas:

- Proporciona un entorno sencillo de manejo del kernel. Oculta al usuario la complejidad de trabajar directamente sobre el kernel. El shell se puede ver como una capa intermedia entre el verdadero sistema operativo (kernel) y el usuario y tiene la misión de facilitar y automatizar tareas.

- Permite abreviaturas de nombres de archivos (símbolos comodín: *, ?). Se puede tomar todo un conjunto de nombres de archivos como argumentos de un programa especificando el patrón de los nombres; el shell encontrará los nombres de archivos que igualen dicho patrón.

- Redireccionamiento de entrada – salida: se puede lograr que la salida de cualquier programa se dirija hacia un archivo en lugar de a la pantalla. Se puede hacer también que la entrada venga también de un archivo en vez del teclado.

- Permite personalizar el entorno: el usuario puede definir sus propios comandos y abreviaturas.

Cuando se inicia una sesión, el shell se ejecuta y toma como parámetros lo que se denomina “entorno”. El entorno especifica todos los parámetros del shell y algunos del sistema para esa sesión. En el se define el PATH (ruta) por defecto, el símbolo del prompt, el directorio de trabajo etc. Este “entorno”, se puede modificar dentro de una sesión incluso de forma permanente. Para ver las variables de entorno de una determinada sesión (cada sesión o usuario puede tener sus propias variables de entorno) ejecutaremos el comando **set.**

Una característica importante de los sistemas Unix es la existencia permanente de ayuda en línea. Es decir, en cualquier momento se puede consultar el manual de un determinado comando. Para ello sólo hay que escribir en la línea de comandos (en el prompt) **man nombrecomando**. Por ejemplo, si queremos saber cómo se utiliza el comando *who*, no tenemos más que hacer *man who* tal y como se muestra a continuación.

```
[rpuerto]$ man who

WHO(1)                                                    WHO(1)
NAME
       who - show who is logged on
SYNOPSIS
       who  [-imqsuwHT]  [--count]  [--idle] [--heading] [--help]
       [--message] [--mesg] [--version] [--writable]  [file]  [am
       i]
DESCRIPTION
       This  documentation  is no longer being maintained and may
       be inaccurate or incomplete.  The Texinfo documentation is
       now the authoritative source.

       This  manual  page  documents  the GNU version of who.  If
       given no non-option arguments, who  prints  the  following
       information for each user currently logged on:

              login name
              terminal line
              login time
              remote hostname or X display

.......

[rpuerto]$
```

Este manual en línea cubre casi todos los comandos y aspectos de Unix. Además, como veremos más adelante, se divide en secciones donde cada una de ellas trata aspectos o comandos diferentes. Ni que decir tiene que el uso de *man* se hace si no imprescindible (¡incluso para los expertos¡), al menos muy recomendable.

Por último, decir que para ejecutar un determinado comando simplemente es necesario escribir éste (con su sintaxis correcta) en la línea de comando y pulsar la tecla ENTER. La ubicación de estos comandos y la explicación de los comandos básicos de Unix se tratan en los puntos siguientes.

3 El sistema de Ficheros

Se denomina sistema de ficheros en Unix a la forma de organizar los ficheros dentro de un medio físico de almacenamiento secundario del sistema.
Un fichero en Unix (y en la mayoría de sistemas operativos) se puede ver como una abstracción del espacio de almacenamiento secundario (discos, cintas, CD's, ...). En Unix existen tres tipos de ficheros:

- Ficheros regulares: representan un fichero convencional de datos (programa, texto, ...)
- Directorio: lugar dentro del sistema de ficheros global que contiene ficheros
- Especial: representa un dispositivo del sistema.

El sistema Unix proporciona un esquema de directorios jerárquicos. Un directorio actúa como un contenedor de ficheros o de otros directorios, lo que resulta en una gran estructura arborescente, con frecuencia descrita como árbol de directorios. Las órdenes, los ficheros de datos, e incluso los dispositivos hardware pueden ser representados como entradas de directorio. El primero de estos directorios, es el denominado *directorio raíz*, denotado por "/". A partir de éste "cuelgan el resto de directorios y ficheros.

Cada fichero (bien sea regular, directorio o especial) tiene asociados una serie de atributos:

- Tipo de fichero
- Propietario del fichero (owner UID)
- Grupo propietario (owner GID)
- Permisos de acceso (permission bits)
- Número de enlaces
- Instantes de creación, último acceso y última modificación
- Tamaño

Todos estos atributos se pueden visualizar e incluso cambiar como veremos más adelante. Como ejemplo, el comando "ls –l" visualiza los ficheros del directorio actual junto con sus atributos:

```
$ ls –l
total 132
-rwxr-xr-x   1 root     root         4588 Mar  2 16:55 Ej1
-rwxr-xr-x   1 root     root         4516 Mar  2 17:09 Ej2
-rwxr-xr-x   1 root     root         4670 Mar  2 17:16 Ej3
-rwxr-xr-x   1 root     root         4601 Mar  2 17:33 Ej5
-rwxr-xr-x   1 root     root         4622 Mar  2 17:36 Ej6
-rwxr-xr-x   1 root     root         4588 Mar  2 16:54 a.out
-rw-------   1 root     root       102400 Mar  2 17:36 core
-rwx------   1 root     root           19 Mar  2 16:55 ejemplo
drwxr-xr-x   2 root     root         1024 Mar  9 10:54 posix
$
```

Especial atención merece el atributo de permisos de acceso, pues en buena medida de él depende la seguridad del sistema.

Cada archivo se encuentra asociado con tres clases: un usuario (o propietario), un grupo y el resto (otros). Los permisos o privilegios para cada una de estas clases son leer (r), escribir (w)y ejecutar (x). Estos permisos son especificados por separado para el usuario, el grupo y los demás. Cada fichero tiene asignado 9 bits donde cada bit a 0 o a 1 quita o da el permiso correspondiente según el patrón que se muestra a continuación:

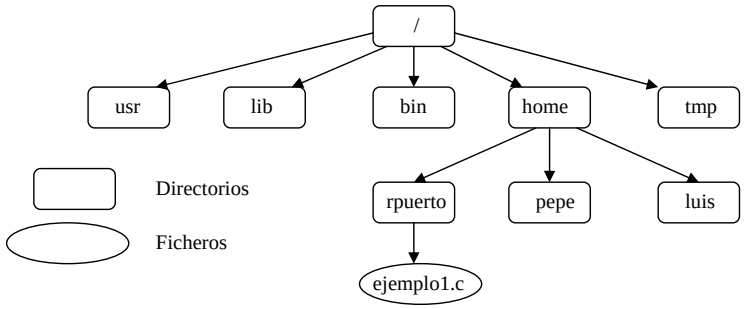
Usuario			Grupo			Otros		
r	w	x	r	w	x	r	w	x
Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

De esta forma, en el ejemplo anterior, el directorio "posix" tiene los siguientes permisos: todos los permisos para el propietario y permisos de lectura y ejecución para el grupo y resto de usuarios. La primera letra "d" significa que es un directorio. Más adelante veremos cómo modificar estos permisos.

Un fichero se identifica desde un punto cualquiera del árbol de directorios mediante su vía o ruta de acceso (path). Esta ruta de acceso puede ser de dos tipos:

- Absoluta: comienza por /
  - /home/rpuerto/ejemplo1.c
  - /vmlinuz
- Relativa: no comienza por / por lo que se considera relativa al directorio de trabajo actual
  - ejemplo1.c, asumiendo /home/rpuerto como directorio de trabajo actual
  - vmlinuz, asumiendo / como directorio actual
- Las entradas . y .. pueden utilizarse para formar vías de acceso, donde "." denota el directorio actual y ".." el directorio inmediatamente superior.
  - ../../vmlinuz, asumiendo /home/rpuerto como directorio de trabajo actual

La figura siguiente muestra la estructura en árbol típica del sistema de ficheros Unix.



Obviamente, en la figura anterior se ha querido simplificar un poco (por claridad) ya que cada directorio puede contener varios directorios más y muchos ficheros (incluso el raíz).

Unix utiliza una estructura de directorios típica, que aunque puede cambiar entre distintas versiones, es prácticamente estándar. La estructura es la siguiente:

Directorio	Descripción
/ (Raíz)	Directorio principal del sistema de ficheros. De él "cuelgan" los demás.
bin	Ejecutables básicos (cp, ls, ...)
lib	Bibliotecas de lenguajes (libc.a, libm.a, ...)
usr	bin: comandos de usuario lib: otras bibliotecas local: sofware local man: manuales
etc	Ficheros de administración (passwd)
dev	Ficheros especiales (uno por cada dispositivo de E/S: tty0, fd0, ...)
Tmp	Ficheros temporales
home	Directorios de usuarios

4 Comandos Básicos

En este punto se van a describir una serie de comandos básicos de Unix. Estos comandos básicos se pueden dividir en dos categorías: comandos orientados a ficheros y comandos orientados a procesos. La descripción que se va a dar aquí de estos comandos va a ser la más habitual, pero casi todos tienen más opciones y son más potentes que lo que aquí se presenta. Por tanto, se sugiere que se emplee el manual en línea (*man*) para ver todas las posibilidades que tiene cada uno de ellos.

A.4.1 Comandos orientados a ficheros

Los comandos orientados a ficheros son todos aquellos que modifican, manipulan o visualizan información acerca de ficheros (tanto regulares como directorios). Aquellos que se consideran más importantes se describen a continuación.

- **Visualización del contenido de un directorio:** el comando que visualiza el contenido de un directorio es *ls*. Por ejemplo:

```
$ ls
Ej1 Ej2 Ej3 Ej5 Ej6 a.out core ejemplo posix
$
```

Como se irá viendo casi todos los comandos suelen tener opciones y la forma más o menos estándar (depende de la versión) de especificar las opciones es escribiendo detrás del comando "-opción". Por ejemplo, una opción interesante de ls es "–l", la cual muestra el contenido de un directorio con información adicional (muestra los permisos de los ficheros, etc.). Por ejemplo

```
$ ls -l
total 132
-rwxr-xr-x   1 root     root         4588 Mar  2 16:55 Ej1
-rwxr-xr-x   1 root     root         4516 Mar  2 17:09 Ej2
-rwxr-xr-x   1 root     root         4670 Mar  2 17:16 Ej3
-rwxr-xr-x   1 root     root         4601 Mar  2 17:33 Ej5
-rwxr-xr-x   1 root     root         4622 Mar  2 17:36 Ej6
-rwxr-xr-x   1 root     root         4588 Mar  2 16:54 a.out
-rw-------   1 root     root       102400 Mar  2 17:36 core
-rwx------   1 root     root           19 Mar  2 16:55 ejemplo
drwxr-xr-x   2 root     root         1024 Mar  9 10:54 posix
$
```

Como se puede apreciar, la información que ofrece "ls –l" de cada fichero del directorio es: Permisos, numero de enlaces, propietario del fichero, grupo al que pertenece, tamaño, fecha, hora y nombre.

Otra opción interesante de ls es "–a", la cual permite visualizar los ficheros ocultos (en Unix, los ficheros ocultos son aquellos que comienzan por "." , por ejemplo ".profile"):

Aparte, se pueden concatenar opciones, es decir, si deseamos visualizar el contenido de un directorio con toda la información, e incluyendo los ficheros ocultos, ejecutaremos "ls –al".

Si queremos comprobar si un determinado fichero existe en el directorio actual, utilizaremos ls de la siguiente forma: *"ls nombre fichero"*, en el caso de que el fichero exista, éste aparecerá en pantalla, en caso contrario se presentará un mensaje indicando que el fichero no existe.
Como se comentó en el punto 3, el shell permite utilizar símbolos comodín para denotar ficheros. Estos símbolos comodín son *, que denota cualquier cadena de caracteres (incluida la cadena vacía) y ? que denota un sólo carácter. Por ejemplo:

- *p denota todos los ficheros que terminan en p
- ej* denota todos los ficheros que comienzan por ej
- ej*p denota todos los ficheros que comienzan por ej y terminan en p
- ej? denota todos los ficheros que comienzan por ej y terminan en cualquier carácter, pero el nombre es de logitud 3, ? denota un sólo carácter.
- ??p denota todos los ficheros cuyo nombre tiene tres caracteres y terminan en p.

Así pues, ls se puede utilizar junto con las expresiones anteriores, es decir, ls –l *p mostrará todos los ficheros que terminan en p.

- **Copia de ficheros:** El comando que copia un fichero es *cp*, y su forma de utilización más básica es "*cp fich_origen fich_destino"*, por ejemplo

```
$ls
Ej1 Ej2 Ej3 Ej5 Ej6 a.out core ejemplo posix
$cp Ej1 Ej1copia
$ls
Ej1 Ej1copia Ej2 Ej3 Ej5 Ej6 a.out core ejemplo posix
```

Otra forma de utilizar *cp* es "*cp fich_origen directorio_destino"*, el cual copia el o los ficheros origen en el directorio destino. Con esta forma también se pueden utilizar símbolos comodín, por ejemplo *"cp *p /tmp"* copia todos los ficheros del directorio actual que terminen por "p" en el directorio tmp.

- **Mover y renombrar ficheros**: Unix sólo permite mover ficheros de un sitio a otro. Ahora bien, si se mueve un fichero dentro de un mismo directorio a la vez que se le cambia el nombre, el efecto producido es el de renombrar dicho fichero. El comando que realiza esta acción es *mv* cuya sintaxis de utilización es "*mv fich_origen fich_destino"* . Por ejemplo:

  - mv ej1.c ej2.c renombra el fichero ej1.c a ej2.c
  - mv ej1.c /home/ramon/ej2.c mueve el fichero ej1.c al directorio /home/ramon con el nombre ej2.c

  Otra sintaxis que permite mover uno o varios ficheros a un directorio es "*mv fich_origen directorio"*, por ejemplo

  - mv ej.* /home/ramon mueve todos los ficheros del directorio actual que comiencen por ej. al directorio /home/ramon

- **Borrado de ficheros**: el comando Unix para borrar ficheros (no directorios) es *rm*. La sintaxis de utilización de este comando es *"rm nombre_ ficheros"*, donde *nombre_ficheros* puede ser un sólo nombre, varios separados por espacios o bien un nombre con símbolos comodín. Por ejemplo:

  - rm ej1.c
  - rm ej1.c ej2.c
  - rm *.c

  ¡Ojo!, una vez se borra un fichero en Unix, no se puede recuperar.

- **Cambiar de directorio**: el comando que cambia de directorio en Unix es *cd*, y su sintaxis es "*cd nombre_directorio"*. Por ejemplo:

  - cd rpuerto
  - cd /usr/local

  Existen además en cada directorio dos entradas por defecto: "." y ".." donde "." denota el directorio actual y ".." el directorio inmediatamente superior. Por ejemplo:

  - cd . no hace nada, permanece en el directorio actual
  - cd .. cambia al directorio inmediatamente superior
  - cd ../ramon cambia al directorio ramon que cuelga del directorio inmediatamente superior al actual.

- **Creación de directorios**: para crear directorios se utiliza el comando *mkdir*, cuya sintaxis de utilización es *"mkdir nombre_directorio"*. Por ejemplo:

  - mkdir programas
  - mkdir /tmp/basura

  La creación de directorios (así como de ficheros en general) se permite siempre y cuando el usuario que intenta crearlo tenga permiso de escritura en el directorio donde pretende crear.

- **Borrado de directorios**: para borrar directorios se utiliza el comando *rmdir* cuya sintaxis es *"rmdir nombre_directorio"*. Un directorio no se puede borrar mientras se está dentro de él, debe estar completamente vacío y además se debe tener permiso de escritura sobre él. Ejemplos:

  - rmdir programas
  - rmdir /tmp/basura

- **Modificación de permisos**: la modificación de permisos en Unix es uno de los temas más importantes de cara a la seguridad del sistema ya que la seguridad se basa en decidir qué ficheros se pueden o no leer o modificar por parte de los usuarios. El único usuario que tiene completa libertad para leer o modificar todos los ficheros del sistema es el superusuario, root o administrador.

En Unix, los permisos de los ficheros se modifican mediante el comando *chmod*. La forma más fácil de utilizar este comando (existen otras más potentes y muchas opciones de utilización) es la siguiente: "*chmod usuario [+-] permisos*" donde

- *usuario* puede tomar uno o varios de las siguientes letras "u", "g", "o", "a"; donde "u" implica que se va a modificar los permisos correspondientes al propietario del fichero, "g" los permisos que el grupo de ese usuario tiene en ese fichero, "o" el resto de usuarios (los que no pertenecen a su grupo) y "a" todos los usuarios

- + ó – implica activar o desactivar un determinado permiso

- *permisos* puede tomar los valores (entre otros) "r", "w", "x" que son permiso de lectura, escritura o ejecución respectivamente.

Aclaremos esto con un ejemplo: supongamos que tenemos el fichero "core" que se muestra a continuación que sólo tiene permiso de lectura para el propietario:

```
$ls –l core
-r--------   1 root     root       102400 Mar  2 17:36 core
$
```

Si deseamos cambiar los permisos y dar permiso de lectura y ejecución para los miembros del grupo:

```
$chmod g+rx core
$ls –l core
-r--r-x---   1 root     root       102400 Mar  2 17:36 core
$
```

Si deseamos que cualquier usuario, aparte de los del grupo del propietario, pueda ejecutar dicho fichero:

```
$chmod a+x core
$ls –l core
-r-xr-x--x   1 root     root       102400 Mar  2 17:36 core
$
```

Para permitir todos los permisos a todo el mundo:

```
$chmod a+rwx core
$ls –l core
-rwxrwxrwx   1 root     root       102400 Mar  2 17:36 core
$
```

Por supuesto esta ultima opción no es nada recomendable ya que todo el mundo puede escribir en el fichero y por tanto borrarlo.

Por último, si se desea que el fichero anterior sólo pueda ser accesible por el dueño:

```
$chmod go-rwx
$ls –l core
-rwx------   1 root     root       102400 Mar  2 17:36 core
$
```

- **Visualización del contenido de un fichero**: para visualizar el contenido de un fichero se utiliza el comando *cat* cuya sintaxis es "*cat nombre_fichero*". Por ejemplo:

    - cat Ej1.c

    - cat ../ramon/Ej2.c

  cat visualiza el fichero por la salida estándar, normalmente la pantalla.

- **Búsqueda de ficheros**: en un sistema Unix existen miles de ficheros distribuidos por toda la estructura de directorios del sistema de fichero. Es por tanto necesario disponer de un mecanismo que permita de una forma fácil la búsqueda de uno o varios ficheros. El comando Unix (uno de ellos) que realiza esta tarea es *find*. La utilización más básica de *find* es "*find directorio –name nombre_fichero*".

  find busca a partir del directorio especificado (es decir en toda la estructura de directorios que cuelga del directorio especificado y en él mismo) el fichero o ficheros (si se utilizan símbolos comodín) que se especifican en *name*. En caso de encontrar el o los ficheros, éstos se muestran en pantalla, y en caso de no existir dichos ficheros se muestra un mensaje de "no encontrado".

  Ejemplos:

    - find . –name ejemplo, busca el fichero ejemplo en el directorio actual y en todos los que cuelgan de él.
    - find / -name *.c, busca a partir del directorio raíz, es decir, en todo el sistema de ficheros, aquellos ficheros que terminen en ".c"
    - find /usr/local –name netscape*, busca a partir del directorio /usr/local todos los ficheros que comiencen por "netscape".

A.4.2 Comandos orientados a procesos

Puesto que Unix es un sistema operativo multiusuario y multitarea, y por tanto permite ejecutar procesos de forma concurrente, es normal que existan varios procesos en ejecución en un mismo instante de tiempo. Es por tanto necesario tener un mecanismo que permita obtener la información del estado de los procesos, manipular e incluso poder provocar la terminación de alguno de ellos ("matar un proceso").

Algunos de los comandos que implementan estos mecanismos se muestran a continuación.

- **Obtener información de los procesos del sistema**: el comando que permite obtener esta información es *ps*, el cual muestra en el dispositivo de salida estándar (pantalla) la información de los procesos que se están ejecutando en el sistema. Por ejemplo:

```
$ps
PID TTY STAT TIME COMMAND
196   1 S    0:00 -bash
197   2 S    0:00 -bash
198   3 S    0:00 /sbin/mingetty tty3
199   4 S    0:00 /sbin/mingetty tty4
200   5 S    0:00 /sbin/mingetty tty5
201   6 S    0:00 /sbin/mingetty tty6
215   1 S    0:00 sh /usr/X11R6/bin/startx
216   1 S    0:00 xinit /usr/X11R6/lib/X11/xinit/xinitrc --
219   1 S    0:00 /usr/X11R6/bin/afterstep
229   1 S    0:00 /usr/X11R6/bin/Animate 7 4 /usr/share/afterstep/base.24bpp 0
230   1 S    0:00 /usr/X11R6/bin/Wharf 9 4 /usr/share/afterstep/base.24bpp 0 8
247  p1 S    0:00 bash
290  p1 S    0:00 vi
292  p2 R    0:00 ps
$
```

En este caso, (ps sin opciones) se presenta para cada proceso que se está ejecutando en el sistema, la siguiente información: PID (identificador del proceso), terminal donde se está ejecutando, tiempo en ejecución (hh:mm) y por último el programa correspondiente a dicho proceso.

ps sin ningún otro parámetro muestra sólo los procesos de usuario, pero no los internos del sistema. Si se desea visualizar todos los procesos se debe utilizar la opción –a:

El comando *ps* tiene muchas opciones, las cuales se pueden consultar mediante el sistema de manual en línea con el comando *man ps*.

- **"Matar un proceso"**: Sólo el superusuario o administrador puede matar cualquier proceso. Un usuario sólo puede matar aquellos que haya creado él mismo. Para matar un proceso en Unix lo que se hace es enviarle una señal de terminación mediante el comando *kill*. Al igual que *ps*, *kill* posee gran variedad de opciones que se pueden consultar en el manual en línea. La forma más normal de terminar un proceso es haciendo "*kill –9 PID"*, donde PID es el número entero que identifica a un determinado proceso dentro del sistema.

  Por ejemplo, si tenemos corriendo en el sistema los siguientes procesos:

```
$ps
PID TTY STAT TIME COMMAND
196   1 S    0:00 -bash
197   2 S    0:00 -bash
198   3 S    0:00 /sbin/mingetty tty3
199   4 S    0:00 /sbin/mingetty tty4
200   5 S    0:00 /sbin/mingetty tty5
201   6 S    0:00 /sbin/mingetty tty6
215   1 S    0:00 sh /usr/X11R6/bin/startx
216   1 S    0:00 xinit /usr/X11R6/lib/X11/xinit/xinitrc --
219   1 S    0:00 /usr/X11R6/bin/afterstep
229   1 S    0:00 /usr/X11R6/bin/Animate 7 4 /usr/share/afterstep/base.24bpp 0
230   1 S    0:00 /usr/X11R6/bin/Wharf 9 4 /usr/share/afterstep/base.24bpp 0 8
247  p1 S    0:00 bash
290  p1 S    0:00 vi
292  p2 R    0:00 ps
$
```

  y deseamos eliminar el proceso correspondiente a la aplicación *vi* con PID 290, haremos:

```
$kill –9 290
$
```

  Para asegurarnos que el proceso ha sido eliminado volveremos a ejecutar el comando ps:

```
$ps
PID TTY STAT TIME COMMAND
196   1 S    0:00 -bash
197   2 S    0:00 -bash
198   3 S    0:00 /sbin/mingetty tty3
199   4 S    0:00 /sbin/mingetty tty4
200   5 S    0:00 /sbin/mingetty tty5
201   6 S    0:00 /sbin/mingetty tty6
215   1 S    0:00 sh /usr/X11R6/bin/startx
216   1 S    0:00 xinit /usr/X11R6/lib/X11/xinit/xinitrc --
219   1 S    0:00 /usr/X11R6/bin/afterstep
229   1 S    0:00 /usr/X11R6/bin/Animate 7 4 /usr/share/afterstep/base.24bpp 0
230   1 S    0:00 /usr/X11R6/bin/Wharf 9 4 /usr/share/afterstep/base.24bpp 0 8
247  p1 S    0:00 bash
292  p2 R    0:00 ps
$
```


- **Ejecución en background**: cuando se trabaja en una terminal, a menudo se desea ejecutar más de un programa sobre la misma terminal (siempre y cuando sólo hay uno interactivo). Supongamos que se desea hacer algo que consume mucho tiempo, como buscar un fichero en todo el sistema de ficheros, pero no se desea esperar a que termine para hacer otra cosa. Entonces se puede teclear lo siguiente "*find / -name libc* >find.out &"*

  El símbolo > se comentará más adelante, lo que permite ejecutar un comando y que el shell quede libre para seguir trabajando es el símbolo &. Cuando ejecutamos un comando con & al final, aparece en pantalla un número correspondiente al PID del proceso que se acaba de lanzar y el shell queda libre para seguir trabajando.

Existen multitud de comandos orientados a procesos. Aquí sólo se han descrito los básicos, pero se recomienda que se consulte por medio del manual en línea al menos *nohup*, *nice* y *at*.

5 Comandos avanzados

A.5.1 Entrada y salida estándar

Muchos comandos UNIX toman su entrada de algo conocido como entrada estándar y envían su salida a la salida estándar (a menudo abreviado como *"stdin"* y *"stdout"*). El intérprete de comandos configura el sistema de forma que la entrada estándar es el teclado y la salida la pantalla.

Veamos un ejemplo con el comando *cat*. Normalmente *cat* lee datos de los ficheros cuyos nombres se pasan como argumentos en la línea de comandos y envía estos datos directamente a la salida estándar. Luego, usando el comando

```
/home/rpuerto# cat fichero1 fichero2
```

mostrará por pantalla el contenido del fichero fichero1 seguido por fichero2.

Si no se le pasan nombres de ficheros a *cat* como parámetros, leerá datos de *stdin* (normalmente el teclado) y los enviará a *stdout*, (normalmente la pantalla o consola) Veamos un ejemplo.

```
/home/rpuerto# cat
Aquí escribo hola.
Aquí escribo hola.
Adiós.
Adiós.

  CTRL-D
/home/rpuerto#
```

Como se puede ver, cada línea que el usuario teclea es inmediatamente reenviada al monitor por *cat*. Cuando se está leyendo de la entrada estándar, los comandos reconocen el fin de la entrada de datos cuando reciben el carácter EOT (end-of-text, fin de texto). Normalmente es generado con la combinación **CTRL-D**

Veamos otro ejemplo. El comando *sort* toma como entrada líneas de texto (de nuevo leerá desde *stdin* si no se le proporcionan nombres de ficheros en la línea de comandos), y devuelve la salida ordenada a *stdout*. Pruebe lo siguiente:

```
/home/rpuerto# sort
barcelona
cantabria
alicante

   CTRL-D
alicante
barcelona
cantabria
/home/rpuerto#
```

Podemos ordenar alfabéticamente una lista de palabras

A.5.2 Redireccionando la entrada y salida

El shell puede redireccionar la entrada estándard y/o la salidad estándar de cualquier programa:

Redireccionamiento de la salida:

`prg [argumentos]> nombre_fichero` (sobreescribe sobre el fichero nombre_fichero)

`prg [argumentos] >> nombre_fichero` (añade al final del fichero nombre_fichero)

Redireccionamiento de la entrada:

`prg [argumentos]< nombre_fichero`

Ahora, supongamos que queremos que la salida de *sort* vaya a un fichero para poder salvar la lista ordenada de salida. El intérprete de comandos nos permite redireccionar la salida estándard a un fichero usando el símbolo ">".

```
/home/rpuerto# sort > lista_ordenada
barcelona
cantabria
alicante

   CTRL-D
/home/rpuerto#
```

Como puede ver, el resultado de *sort* no se muestra por pantalla, en su lugar es salvado en el fichero lista_ordenada. Visualizamos el fichero.


```
/home/rpuerto# cat lista_ordenada
alicante
barcelona
cantabria
/home/rpuerto#
```

Con esto hemos conseguido guardar en un fichero la lista ordenada que hemos introducido aleatoriamente.

Supongamos ahora que teníamos guardada nuestra lista de compra desordenada original en el fichero lista_desordenada. Una forma de ordenar la información y salvarla en un fichero podría ser darle a *sort* el nombre del fichero a leer en lugar de la entrada estándar y redireccionar la salida estándar como hicimos arriba.

```
/home/rpuerto# sort lista_desordenada > lista_ordenada
/home/rpuerto# cat lista_ordenada
alicante
barcelona
caantabria
/home/rpuerto#
```

Hay otra forma de hacer esto, redireccionando la entrada estándar usando el símbolo "<".

```
/home/rpuerto# sort < lista_desordenada
alicante
barcelona
cantabria
/home/rpuerto#
```

Redirección no destructiva

El uso de ">" para redireccionar la salida a un fichero es destructivo: en otras palabras, el comando

```
/home/rpuerto# ls > fichero1
```

sobreescribe el contenido del fichero "fichero1". Si en su lugar, usamos el símbolo ">>", la salida será añadida al final del fichero nombrado, en lugar de ser sobreescrito.

```
/home/rpuerto# ls >> fichero1
```

añade la salida de *ls* al final de fichero1.

A.5.3 Filtros

Un filtro es un programa que lee datos de la entrada estándar, los procesa de alguna forma, y devuelve los datos procesados por la salida estándar.

Usando la redirección la entrada estándar y/o salida estándar pueden ser referenciadas desde ficheros. *sort* es un filtro simple: ordena los datos de entrada y envía el resultado a la salida estándar. *cat* es incluso más simple, no hace nada con los datos de entrada, simplemente envía a la salida cualquier cosa que le llega.

Tuberias (pipes)

El shell usa una tuberia (pipeline) para conectar la salida estándar de un programa directamente a la entrada estándar de otro. El símbolo es " | "

```
/home/rpuerto# ls –C1
      lista_desordenada
      lista_ordenada
      a_fichero
      b_fichero
/home/rpuerto# ls > directorio
/home/rpuerto# sort -r directorio
      lista_ordenada
      lista_desordenada
      directorio
      b_fichero
      a_fichero
/home/rpuerto#
```

Aquí, salvamos la salida de *ls* en un fichero, y entonces ejecutamos *sort -r* (el argumento –r indica orden inverso en el orden alfabético, consultar *man sort* para mas detalles sobre otros argumentos) sobre ese fichero. Pero esta forma necesita crear un fichero temporal (en este caso le hemos llamado directorio, que se introduce en el listado de directorio) en el que salvar los datos generados por *ls*.

La solución es usar las pipes.

```
      /home/rpuerto# ls | sort -r
      lista_ordenada
      lista_desordenada
      b_fichero
      a_fichero
      /home/rpuerto#
```

Podemos "entubar" más de dos comandos a la vez. El comando *head* es un filtro que muestra la primeras líneas del canal de entrada (aquí la entrada desde una pipe). Si queremos ver el último fichero del directorio actual en orden alfabético, usaremos:

```
      /home/rpuerto# ls | sort -r | head -1
      lista_ordenada
      /home/rpuerto#
```

A.5.4 Comandos útiles.

A continuación vamos a ver una serie de comandos añadidos a los ya estudiados anteriormente. Al igual que antes, la descripción que se va a dar aquí va a ser escueta, y casi todos tienen más opciones y son más potentes que lo que aquí se presenta. Por tanto, se sugiere que se emplee el manual en línea (*man*) para ver las posibilidades que tiene cada uno de ellos.

```
more [-l] [+número_línea] [fichero1 fichero2 . . . ficheroN]
```


more, uno de los comandos más útiles, se utiliza para visualizar ficheros largos en pantalla y que los volcados se detengan a partir de un determinado número de líneas para su correcta visualización. Además, es el comando recomendado para ver ficheros de texto ASCII. La única opción interesante es "*-l*" , que indica al comando que no se desea interpretar el carácter *Ctrl-L* como carácter de "nueva página". El comando comenzará en la línea número_línea especificada.

Al ser more un comando interactivo, hemos resumido a continuación las órdenes más comunes:

P Pasar a la siguiente pantalla de texto.

D Pasar 11 líneas de pantalla, o aproximadamente la mitad de una pantalla normal: 25 líneas.

/ Busca una expresión regular. La construcción de una expresión regular puede ser muy complicada por lo que es recomendable teclear simplemente el texto a buscar.

Por ejemplo,

```
/sapo
```
Ret

buscará la primera ocurrencia de *"sapo"* en el fichero a partir de la posición actual. La misma tecla seguida por un ρ buscará la siguiente ocurrencia de la última expresión buscada.

N Buscará la próxima aparición de la expresión regular especificada.

Q Terminar more.

Ejemplos:

```
/home/rpuerto# cd /etc
/etc# ls|more
...

/etc#cd
/home/rpuerto#ls /etc >listado
/home/rpuerto#more listado
```

```
head [-líneas] [fichero1 fichero2 . . . ficheroN]
```

head mostrará las primeras diez líneas de los ficheros especificados, o las primeras diez líneas de la *stdin* si no se especifica ningún fichero en la línea de comandos. Cualquier opción numérica se tomará como el número de líneas a mostrar, por ejemplo

```
/home/rpuerto#head  -15  listado
```

mostrará las primeras quince líneas del fichero `listado`.

```
tail [-líneas] [fichero1 fichero2 . . . ficheroN]
```

Como *head*, *tail* mostrará solo una parte del fichero, en este caso el final del fichero, las últimas

diez líneas del fichero, o que provengan de la *stdin*. *tail* también acepta la opción de especificar el número de líneas, como en el caso anterior.

```
file [fichero1 fichero2 . . . ficheroN]
```

El comando *file* intenta identificar que tipo de formato tiene un fichero en particular. Debido a que no todos los ficheros tienen extensión u otras formas de identificarlos fácilmente, este comando realiza algunas comprobaciones rudimentarias para intentar comprender exactamente qué contiene el fichero.

Hay que tener cuidado ya que es bastante posible que *file* realice una identificación incorrecta.

Ejemplos:

```
/home/rpuerto# file listado
listado:ASCII file
```

```
grep [-nvwx] [-número] expresión [fichero1 fichero2 . . . ficheroN ]
```

Uno de los comandos más útiles de Unix es *grep*, utilizado para buscar expresiones en un fichero de texto. La forma más sencilla de usarlo es:

```
/home/rpuerto# cat lista_ordenada
      alicante
      barcelona
      cantabria
/home/rpuerto# grep ante lista_ordenada
       alicante
/home/rpuerto# grep pa listado
...
```

Una de los problemas de esta forma de usarlo, es que simplemente muestra las líneas que contienen la palabra a buscar, no aporta información acerca de donde buscar en el fichero, es decir el número de línea. Pero dependiendo de lo que se pretenda puede ser hasta más que suficiente, por ejemplo, si se buscan los errores de la salida de un programa, se puede probar:

```
/home/rpuerto# a.out |grep error
...

/home/rpuerto#
```

donde `a.out` es el nombre del programa.

Cuando es necesario conocer donde están las palabras a buscar, es decir el número de línea, hay que utilizar la opción "*n*". Si lo que se desea es ver todas las líneas donde no se encuentra la expresión especificada, entonces utilice la opción "*v*".

6 Vi. El editor universal

vi (pronunciado "vi ai" en inglés, o "uve i") es en realidad el único editor que se puede encontrar en prácticamente cualquier instalación de Unix. Este editor fue escrito originalmente en la Universidad de California en Berkeley y se puede encontrar versiones en casi cualquier edición de Unix, (incluído Linux). Al principio cuesta un poco acostumbrarse a él, pero tiene muchas características muy potentes.

Incluso si vi no se convierte en su editor de texto normal, el conocimiento de su uso no será desperdiciado. Es casi seguro que el sistema de Unix que use tendrá alguna variante del editor vi. Muchas herramientas de Unix, aplicaciones y juegos usan un subconjunto del grupo de comandos de vi.

Para ejecutar vi, simplemente tiene que teclear las letras vi seguidas del nombre de fichero que desea crear.

```
/home/rpuerto# vi fichero_desordenado
```

Aparecerá una pantalla con una columna de tildes (~) en el lado izquierdo. vi está ahora en modo de comando. Cualquier cosa que teclee será interpretado como un comando, no como texto que usted desea escribir.

Vi tiene dos modos básicos:

- **Command mode (modo comando)**: Cada tecla presionada se interpreta como un comando.
- **Insert mode (modo inserción)**: Cada tecla presionada se interpreta como un carácter (texto).

Para introducir texto, tiene que teclear un comando. Los comandos de entrada básicos son los siguientes:

- i insertar texto a la izquierda del cursor.
- I inserta texto antes del primer carácter de la línea.
- a añadir texto a la derecha del cursor.

Dado que se está al comienzo de un fichero vacío, no importa cual de estos usar. Escriba uno de ellos, y después teclee el siguiente texto

```
cantabria  Ret
alicante   Ret
barcelona  Ret
E
```

Fíjese que tiene que pulsar la tecla ε para finalizar la inserción y volver al modo de comando.

Comandos de movimiento del cursor

- h mueve el cursor un espacio a la izquierda.

- j mueve el cursor un espacio abajo.
- k mueve el cursor un espacio arriba.
- l mueve el cursor un espacio a la derecha.

Comandos para borrar texto

- x borra el carácter que hay en el cursor.
- dd borra la línea donde está el cursor.

Ejemplo: Mueva el cursor a la primera línea y póngalo de modo que esté bajo la c. Pulse la letra **X** y la c desaparecerá. Ahora pulse la letra **I** para cambiarse al modo de inserción y vuelva a teclear la **C**. Pulse **E** cuando haya terminado.

Salvar un fichero

- :w salvar (escribir al disco).
- :q salir.
- :q! Salir sin salvar los cambios
- :x salvar y salir

El comando para salir de vi es ":q". Si quiere combinar el salvar y salir, escriba ":wq".También hay una abreviación para ":wq" que es "ZZ".

Más comandos de vi

Comandos de movimiento del cursor

- w mueve al principio de la siguiente palabra.
- e mueve al final de la siguiente palabra.
- E mueve al final de la siguiente palabra antes de un espacio.
- b mueve al principio de la palabra anterior.
- 0 mueve al principio de la línea actual.
- ^ mueve a la primera palabra de la línea actual.
- $ mueve al final de la línea.
- G mueve al final del fichero.
- 1G mueve al principio del fichero.
- nG mueve a la línea n.
- % va al paréntesis correspondiente.
- H mueve a la línea superior en pantalla.
- M mueve a la línea de en medio de la pantalla.
- L mueve al final de la pantalla.

Además de los comandos ya vistos de vi, se recomienda consultar el manual en linea de vi, (*man vi*) pues se descubrirá que vi es mucho más potente de lo que a primera vista parece. Vi también posee comandos para cortar y pegar (al estilo windows), realizar busquedas y reemplazos, y otras herramientas que resultan muy útiles cuando se está escribiendo un texto o las fuentes de un programa, (por ejemplo un script del shell).

7 Programación en Shell. Scripts sencillos

A veces, la realización de una tarea resulta tediosa debido a la cantidad de comandos que tenemos que escribir en la consola de forma secuencial. Para evitar esto, el shell permite la ejecución automática de un grupo de comandos almacenados en un fichero. Utilizado como lenguaje de programación, el shell procesa grupos de comandos almacenados en ficheros llamados **scripts**. Los scripts permiten agrupar líneas de comandos para ejecutarlos con un solo comando.

Variables

El shell utiliza variables de cadena para representar tanto números como texto. Hay tres tipos de variables:

- Variables de usuario.
- Variables de shell.
- Variables de sólo lectura.

Ejemplos de variables:

Variables de usuario:

✓ Definidas por el programador

Variables shell:

✓ HOME: al entrar en el sistema, el shell toma el nombre del camino del directorio domicilio y lo asigna a HOME.
✓ PATH: controla el camino de búsqueda en los directorios para encontrar el mandato tecleado.
✓ PS1: almacena el prompt.
✓ PS2: almacena el prompt secundario.

Variables de sólo lectura:

✓ $0: almacena el nombre del comando.
✓ $1,$2,$3..$9: almacena los nueve primeros argumentos de la línea de comandos.

Los scprits empiezan siempre con una línea en la que se indica para que shell han sido escritos, y por tanto qué shell debe utilizarse para su ejecución (normalmente sh). El formato de esta línea es el siguiente:

```
#/bin/sh
```

(siempre que mi shell se encuentre localizada en el subdirectorio /bin de mi sistema).

A continuación vamos a ver algún ejemplo sencillo de scripts, aunque la potencia de estos pequeños programas es mucho mayor que la aquí vista. Los scripts, por ejemplo, suelen usarse durante el proceso de arranque de la máquina para establecer una configuración apropiada, incluso hay multitud de scripts que realizan tareas de administración del sistema.

Ejemplo sencillo: Programa para realizar un listado con atributos –la

```
/home/rpuerto#vi mi_primer_script
...
(entrando en vi a modo inserción)

#!/bin/sh
#programa mi_primer_script de listado de ficheros
#umh-isa 1999

echo listando...
ls –la

(modo comando)
:wq
/home/rpuerto#
```

Ejemplo sencillo: este sencillo script desciende recursivamente a partir del directorio actual borrando ficheros core

```
/home/rpuerto#vi limpieza
...
(entrando en vi a modo inserción)

#!/bin/sh
#programa limpieza de limpieza de ficheros core
#umh-isa 1999

echo limpiando...
find  ./ -name core -exec rm  {} \;

(modo comando)
:wq
/home/rpuerto#
```

Una vez creado el fichero de texto que contiene los comandos a ejecutar, sólo resta dar permisos de ejecución a dicho fichero. Para ejecutar el script sólo tenemos que escribir el nombre del fichero correspondiente en la línea de comandos.

8 C en entorno Unix :

En este apartado vamos a introducir el compilador C más común del mundo UNIX, con el fin de realizar una serie de ejercicios con él. Escribiremos nuestras fuentes utilizando el vi, para más tarde, utilizando el compilador, crear nuestros ejecutables.

Compiladores de C (cc, gcc)

El compilador de C que normalmente se encuentra instalado es **gcc** que es un compilador de C de GNU de libre distribución.

A continuación se verán los parámetros básicos que nos permitirán compilar programas en C con este compilador. Se recomienda consultar la ayuda en linea de los compiladores (*man gcc*) y las referencias que en ella aparecen, pues existen muchas posibilidades no contempladas en esta práctica que pueden resultar muy útiles.

Caso 1: Tenemos un programa en C en un fichero llamado *miprograma.c* y queremos compilarlo. Suponemos que sólo utiliza cabeceras estándar de C y no utiliza ninguna otra librería de programas. La orden para compilarlo será:

```
/home/rpuerto$ gcc –o miprograma miprograma.c
```

Con esto obtendremos, si no hay errores de compilación, un ejecutable que se llamará *miprograma* (la opción *–o* indica cómo se va a llamar el ejecutable que se va a crear—por defecto se llamará *a.out*).

Caso 2: El mismo programa de antes utiliza ahora aritmética en coma flotante. Esto hace necesario el enlazarlo con la librería matemática de C, que de forma estándar se llama *libm.a*. A la hora de incluir esta librería utilizamos la opción *–l*:

```
/home/rpuerto$ gcc –o miprograma –llibm.a miprograma.c
```

Con esto el compilador encuentra sin problemas la librería *libm.a*, ya que es estándar de C y se encuentra junto con el compilador. Si quisiéramos utilizar una librería y/o fichero de cabecera no estándar de C le tendremos que indicar al compilador donde puede encontrar estos ficheros.

Si tenemos nuestro programa dividido en dos ficheros, digamos *miprograma.c* y *vector.c* podemos obtener nuestro ejecutable con las órdenes:

```
/home/rpuerto$ gcc –o miprograma.o –c miprograma.c
/home/rpuerto$ gcc –o vector.o –c vector.o
/home/rpuerto$ gcc –o miprograma vector.o miprograma.o
```

Primero compilamos nuestro dos fuentes pero no los enlazamos con ninguna librería (opción *–c*) y posteriormente enlazamos nuestros dos ficheros intermedios (con extensión *.o*).

Herramientas de compilación: make

Durante la fase de desarrollo de nuestro programa será necesario compilarlo una y otra vez, eliminando los errores que hayamos introducido en el código y/o añadiendo nuevas funcionalidades al mismo. Para no tener que volver a escribir cada vez la orden de compilación

(que puede llegar a ser bastante larga) podemos utilizar la herramienta *make* que está incorporada en todo entorno Unix.

Make permite ejecutar ciertas órdenes con el fin de alcanzar un cierto objetivo teniendo en cuenta unas ciertas dependencias, y utiliza un fichero de configuración para indicar relaciones objetivos—dependencias. La estructura básica de un fichero para *make* (que podemos llamar: *makefile* o *Makefile*) es:

```
<objetivo1>:<dependencia1_1><dependencia1_2>...<dependencia1_N>
<TAB>   <orden1_1>
<TAB>   <orden1_2>
...
<TAB>   <orden1_M>
```

Un *makefile* puede contener tantos objetivos como queramos y cuando llamamos a *make* sin ningún parámetro se intentará realizar el primer objetivo que encuentre. Tanto los objetivos como las dependencias serán normalmente ficheros y un objetivo se considerará realizado si la hora de creación del fichero objetivo es posterior a cualquiera de la hora de las dependencias. En caso contrario se ejecutarán las órdenes asociadas con el objetivo (que normalmente estarán encaminadas a generar, en otras cosas, el fichero nombrado como objetivo).

Ejemplo:

Vamos a ver como podríamos hacer un *Makefile* para el caso de varios ficheros fuente como hemos visto anteriormente

```
/home/rpuerto$ cat Makefile
# Makefile para miprograma.c

miprograma: miprograma.c vector.c
        gcc –o miprograma.o –c miprograma.c
        gcc –o vector.o –c vector.c
        gcc –o miprograma miprograma.o vector.o
/home/rpuerto$
```

El problema de este *Makefile* es que si modificamos, por ejemplo, el fichero *vector.c* al volver a ejecutar *make* se volverán a compilar tanto *vector.c* como *miprograma.c* (aunque este último no lo hayamos modificado). Esto lo podemos arreglar modificando un poco el *Makefile*

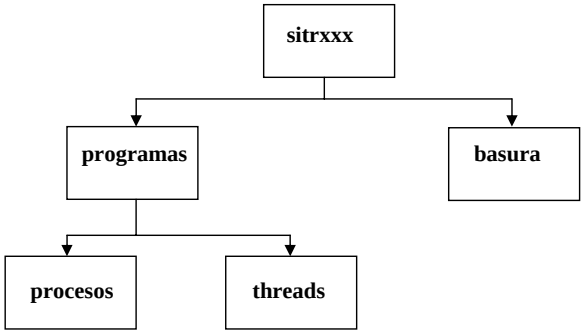
```
/home/rpuerto$ cat Makefile
# Makefile mejorado

miprograma: miprograma.o vector.o
        gcc –o miprograma miprograma.o vector.o

miprograma.o: miprograma.c
        gcc –o miprograma.o –c miprograma.c

vector.o: vector.c
        gcc –o vector.o –c vector.c

borrar:
        rm *.o
/home/rpuerto$
```

Con este segundo ejemplo, si modificaramos *vector.c* sólo se volverian a rehacer los objetivos *vector.o* (porque depende del *vector.c* que hemos modificado) y *miprograma* (porque depende de *vector.o*, que se ha vuelto a compilar).

A modo de ejemplo, también se ha añadido un nuevo objetivo, *borrar*, que no tiene dependencias (por eso se ejecutará siempre que lo llamemos) y que nos puede servir para borrar los ficheros **.o* que tengamos. Como no es el primer objetivo del *Makefile*, no se ejecutará al llamar a *make*. Para ejecutarlo tendríamos que utilizar la orden: *make borrar*.

9 Ejercicios propuestos.

1. Entrar en el sistema con el ***login*** y ***password*** correspondiente y comprobar cual es el directorio actual y quien está trabajando en el sistema en ese instante
2. Puesto que Unix es un sistema operativo multiusuario es conveniente proteger los datos personales. Para ello se debe cambiar los permisos de acceso del directorio personal y deja sólo permisos de lectura y escritura para el dueño.
3. Comprobar cual el directorio personal y crear dentro de él el siguiente árbol de directorios:



4. Copiar el fichero ***test*** que se encuentra en el directorio ***/home/sitr/pub*** al directorio ***basura***.
5. Duplicar el fichero ***test*** con nombre ***test1*** y borrar el fichero ***test***.
6. Renombrar el fichero ***test1*** a ***test*** y moverlo al directorio ***procesos***
7. Visualizar el contenido del fichero ***test***.
8. Comprobar los permisos del fichero ***test*** y modificarlos según los siguientes patrones:
   - Todos los permisos para todos los usuarios
   - Todos los permisos para el dueño, lectura y ejecución para el grupo y ninguno para el resto
   - Sólo lectura y ejecución para el dueño.
9. Hacer ejecutable (dar permiso de ejecución) para el fichero ***test*** y ejecutarlo.
10. Borrar el directorio ***basura***.
11. Comprobar los procesos que se están ejecutando en el sistema (en su sesión). Visualizar todos los procesos que se ejecutan en el sistema (usar ***man ps***).
12. Ejecutar el programa ***vi*** en background y comprobar que realmente se está ejecutando. Una vez comprobado, eliminar dicho proceso.
13. Probar todos los ejemplos del apartado 5 de este manual.
14. Escribir con el editor un programa en C que imprima en pantalla la frase “Esta asignatura es demasiado fácil para un genio como yo”.
15. Compilar y ejecutar el programa del ejercicio anterior.