Universidad Nacional Autónoma de Nicaragua

Departamento de Computación

Ingeniería en Telemática

Práctica 0: Introducción a UNIX

Sistemas telemáticos

1. Introducción

Cuesta creer que el sistema operativo Unix universal fuese ideado hace más de un cuarto de siglo por una sola persona, Ken Thompson de Bell Laboratories. Al haberse desligado su empresa de un ambicioso proyecto que nunca llegó a funcionar, Ken decidió ocupar su tiempo en diseñar un sistema operativo orientado fundamentalmente al desarrollo de programas. Con la siguiente práctica se pretende aprender las nociones básicas sobre el uso de este sistema.

UNIX es un sistema operativo multitarea y multiusuario:

- **Multitarea**: En un instante de tiempo puede haber varios procesos cargados en memoria compitiendo por el uso de la CPU. En los sistemas monoprocesador, los distintos procesos deben repartirse el tiempo de CPU. Este reparto lo lleva a cabo el planificador (scheduler). En los sistemas multiprocesador puede haber varios procesos ejecutándose en paralelo.

- **Multiusuario**: Puede haber varias personas utilizando simultáneamente los recursos del sistema. Cada usuario va a tener asignada una cuenta de usuario a través de la cual accede al sistema. Cada cuenta tiene asociada un nombre de usuario (username o login) por el que el sistema la reconoce. Cada cuenta es personal y solo podrá ser manipulada por su propietario.

Un sistema UNIX es cualquier sistema operativo que cumpla con una serie de especificaciones conocidas como POSIX (Portable Operating System UNIX). Sistemas UNIX existen muchos: Solaris, AIX, HP-UX, etc. Actualmente existe un sistema UNIX de particular importancia, Linux, que tiene una historia un tanto particular.

Linux es un sistema operativo tipo UNIX que naci´o siendo un pasatiempo para su creador, un estudiante finlandés llamado Linus Torvalds. Linus tuvo una sencilla idea que cambió el rumbo de Linux: publicó su código fuente en Internet y permitió que cualquier persona lo descargara y utilizara, es decir, convirtió Linux en software libre. A partir de entonces miles de programadores de todo el mundo se unieron al proyecto y crearon lo que hoy en día es Linux, un sistema operativo libre y de gran calidad que está instalado en millones de máquinas de todo el mundo.

2. Introducción a las órdenes en UNIX

La estructura general de cualquier orden en UNIX es:

```
nombre [-modificadores] [parámetros]
```

siendo `nombre` el nombre del programa u orden a ejecutar. Lo más común es que todas las órdenes admitan modificadores que suelen comenzar con el signo - (menos) y por una única letra, permitiendo caracterizar, alterar o configurar el funcionamiento de una orden. Estos modificadores pueden ir seguidos sin necesidad de colocar espacios en blanco entre ellos, por ejemplo

```
$ ls -l –a
```

es lo mismo que teclear

```
$ ls –la
```

La orden `ls` es el equivalente en UNIX de la orden `dir` de MSDOS, y sirve para visualizar los archivos que contiene un directorio. Más adelante en la práctica se verá en detalle el funcionamiento de esta orden. Los parámetros deben estar separados por espacios en blanco y suministran información adicional a la orden.

Ejemplo:

```
$ ls -l .profile
$ cat xxx
```

Además es importante volver a señalar que UNIX distingue las mayúsculas de las minúsculas, al contrario que otros sistemas operativos como MS-DOS o Windows.

2.1 Órdenes básicas

En esta sección se explican algunas órdenes cuyo conocimiento es imprescindible para poderse mover dentro del intérprete de órdenes. Por sí mismas no aportan gran utilidad, pero son un complemento importante de otras órdenes y van a ser de gran utilidad para el resto de las prácticas.

man

```
Sintaxis: man término
```

En UNIX es posible obtener información detallada en línea sobre cualquier aspecto del sistema sin más que teclear man y el término sobre el que queremos obtener información. Esto muestra en pantalla las páginas del Manual de Referencia de UNIX asociadas al término (orden, archivo, llamada al sistema, etc.).

Este manual está dividido en un conjunto de secciones de las cuales las tres primeras son estándar:

- Sección 1 – Órdenes de usuario
- Sección 2 – Llamadas al sistema
- Sección 3 – Bibliotecas de funciones estándar de C

Ejemplo de uso:

```
$ man man
man(1) man(1)
NAME man - format and display the on-line manual pages manpath - determine user's search
SYNOPSIS man [-acdfFhkKtwW] [-m system] [-p string] [-C config file] [-M path] [-P pager]
DESCRIPTION man formats and displays the on-line manual pages. This version knows about
```

En el ejemplo, (1) indica el número de sección. Cuando un término tiene varias entradas en diferentes secciones man término muestra siempre la información contenida en la primera de ellas. Si se quiere consultar otras secciones es necesario indicarlo por ejemplo:

```
$ man nro_sección término
```

En la tabla 1 se muestran las operaciones básicas con man.

Operación	Secuencias de teclas	Descripción
Avance de página	Control + f Espacio	Muestra la siguiente pantalla
Retroceso de página	Control + b	Muestra la pantalla anterior
Salir de la ayuda	q	Finaliza el man
Localización adelante	/palabra	Localiza la siguiente ocurrencia de la palabra dad como parámetro
Localización atrás	?palabra	Localiza la ocurrencia anterior de la palabra dada como parámetro

Tabla 1. Operaciones básicas con man

clear

```
Sintaxis: clear
```

Limpia la pantalla y sitúa el prompt del sistema en la parte superior de la pantalla.

date

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

Esta orden muestra la hora y la fecha actual del reloj del sistema. La opción –u provoca que la hora se exprese en formato UTC (Coordinated Universal Time). En el parámetro format se pueden especificar diferentes directivas que permiten caracterizar la salida.

La cadena format estará constituida por cualquier combinación de caracteres ordinarios y directivas. Una directiva está formada por el carácter% junto con un campo opcional que permite especificar la precisión y la anchura y un carácter de terminación que determina el comportamiento de la directiva. Se puede obtener una lista de los caracteres de terminación y su significado mediante man.

Ejemplo:

```
$ date
Wed Sep 27 16:58:03 CET 2001
$ date '+D´ıa: %m/ %d/ %y %n Hora: %H: %M: %S'
Día:10/08/01
Hora:12:45:05
```

echo

```
Sintaxis: echo -e [arg] ... –
```

Escribe las cadenas que recibe como argumentos, añadiendo al final un salto de línea.

Ejemplo:

```
[p21g3@lsotm p21g3]$ echo REPITE ESTO
REPITE ESTO
```

echo también permite escribir secuencias de escape empleando para ello una sintaxis similar a la de C; con los siguientes significados:

\b escribe un carácter de borrado (backspace)
\c escribe una línea sin añadir un salto de línea al final
\f carácter de salto de página (form-feed)

\n carácter de salto de línea (new-line)
\r carácter de retorno de carro (carriage return)
\t carácter de tabulación
\v carácter de tabulación vertical
**** escribe el carácter \ (backslash)
\NNN escribe el carácter ASCII correspondiente a la representación octal NNN. NNN debe ser una secuencia de 1, 2, 3, 4 dígitos.

Como el carácter \ tiene significado especial para el intérprete de órdenes, es necesario evitar que éste lo procese. Para ello se encierra la expresión que lo contiene entre comillas simples o dobles. echo es utilizado con frecuencia dentro de programas de shell y para visualizar el contenido de las variables del intérprete de órdenes como se estudiará en prácticas posteriores.

who

```
Sintaxis: who
```

Esta orden permite conocer qué usuarios están conectados actualmente en el sistema, mostrando además el terminal asociado a cada sesión de cada usuario (un usuario puede tener activas varias sesiones simultáneamente) y la hora de establecimiento de la sesión. Cuando se emplea el parámetro am i el sistema muestra únicamente la información referida al propio usuario.

Ejemplo:

```
$ who am i
lsotm!p21g3 pts/1 Sep 27 07:50
$ who
lsotm!p21g3 pts/1 Sep 27 07:50
tdcli00 ttyp2 Sep 27 23:14
chan ttyp1 Sep 27 07:50
root console Sep 27 20:32
```

uname

```
Sintaxis: uname [ -snrvmia ]
```

Suministra diversa información sobre el sistema UNIX en el que se está conectado, sin opciones muestra la versión particular del sistema que nos atiende.

```
$ uname
Linux
```

Las opciones más comunes son:

-a muestra toda la información sobre el tipo de sistema que se está utilizando. Equivale a todas las siguientes opciones:

-m tipo de hardware que se está utilizando.
-s nombre del sistema.
-n nombre por el que se identifica el sistema en la red.
-r revisi´on del sistema operativo.
-v versi´on del sistema operativo.

Ejemplo:

```
$ uname -a
HP-UX motruo-0 A.08.00 E 9000/835 43405112
```

2.2 Caracteres comodines

En UNIX existen un conjunto de caracteres cuya combinación permite que el intérprete de órdenes los sustituya por un grupo de caracteres. Estos caracteres se denominan caracteres comodines. De entre los existentes, destacamos los siguientes:

***** Se sustituye por un conjunto cualquiera de caracteres (cero o m´as caracteres).

```
$ ls * --> muestra los archivos
           del directorio de trabajo
$ ls ab* --> muestra los archivos
             que empiecen por 'ab' en
             el directorio de trabajo
```

? Se sustituye por un carácter cualquiera (siempre un carácter).

```
$ ls a?b --> muestra los archivos cuyos
             nombres tengan tres caracteres, el
             primero sea una 'a' y el ´ultimo una 'b'
```

[...] Se sustituye por uno de los caracteres del conjunto.

```
$ ls ab[123] --> muestra (si existen) los archivos
                 'ab1', 'ab2' y 'ab3'
```

[!...] Se sustituye por un carácter NO contenido en el conjunto.

```
$ ls [!ab]* --> muestra todos los archivos cuyo
                nombre NO comienza por ni por
                'a' ni por 'b'
```

Aparte de los caracteres comodín, existen otros caracteres cuyo significado es especial para la shell. Entre estos destacamos: < , > , | , & cuyo significado se explicarán en prácticas posteriores. De modo general para todos los caracteres especiales y comodín, se habilitan distintas maneras para forzar a que la shell los interprete literalmente como los caracteres que son, sin sustituirlos. Los modos de escapar los caracteres especiales y comodín son los siguientes:

- Utilizando el carácter \ se pueden escapar caracteres individualmente (se escapa el carácter siguiente).
- Cuando se quiere escapar cadenas completas se recurre a encerrarlos entre comillas, tanto simples (') como dobles (").

```
$ ls 'test?&' --> muestra el archivo con nombre test?&
$ ls \*a --> muestra el archivo de nombre *a
```

2.3 Caracteres de control

Se denominan caracteres de control a aquellos que tienen un significado especial para el intérprete de órdenes y suelen tener un efecto sobre el proceso que se esté ejecutando en ese momento. Los más importantes son:

<Crtl> c Termina o aborta la ejecución de la orden que se está ejecutando.

<Crtl> z Detiene temporalmente la ejecución del proceso, devolviendo el control de la consola al usuario. Se puede volver a ejecutar el proceso con la orden fg.

<Crtl> s Detiene la visualización en pantalla.

<Crtl> q Reanuda la visualización en pantalla.

<Crtl> d Es utilizado por aquellos programas que aceptan datos desde teclado para indicar el final de los datos.

3. Introducción al sistema de archivos de UNIX

Comprender el funcionamiento de UNIX es en gran parte comprender cómo funciona su sistema de archivos. Un archivo se puede definir como un conjunto de datos con un

identificador asociado. Los archivos suelen residir en dispositivos de almacenamiento secundario, tales como cintas, discos duros, CD-ROMs, DVDs, discos flexibles, etc.

Algunos SS.OO. imponen a los archivos una estructura determinada bien definida. Este no es el caso de UNIX, donde un archivo no es más que una secuencia de bytes, siendo los programas los encargados de interpretar esta secuencia y darle significado según sus necesidades. Es decir, la sintaxis (forma) del acceso a los datos de un archivo viene impuesta por el sistema, y es la misma para todos los programas, y la semántica (significado) de los datos es responsabilidad del programa que trabaja con el archivo.
La mayoría de los SS.OO. permiten varios tipos de archivos diferentes; en el caso de UNIX, tenemos los siguientes tipos: archivos normales, directorios, dispositivos especiales tipo bloque (discos, cintas, etc.), dispositivos especiales tipo carácter (terminales, algunas cintas, etc.), pipes con nombre para permitir comunicación entre procesos (IPC) y enlaces simbólicos.

Un sistema de archivos se debe entender como aquella parte del sistema responsable de la administración de los datos en dispositivos de almacenamiento secundario. Por lo tanto, es el sistema de archivos el que debe proporcionar los medios necesarios para un acceso y almacenamiento seguro y privado de la información.

3.1 Estructura jerárquica del sistema de archivos

En UNIX los archivos están organizados en lo que se conoce como directorios. Un directorio no es más que un archivo especial, que contiene información que permite localizar otros archivos. Los directorios pueden contener a su vez nuevos directorios, que se denominan **subdirectorios**. A la estructura resultante de esta organización se le conoce con el nombre de estructura en **árbol invertido**. Un ejemplo típico de árbol de directorios es el mostrado en la figura 1.

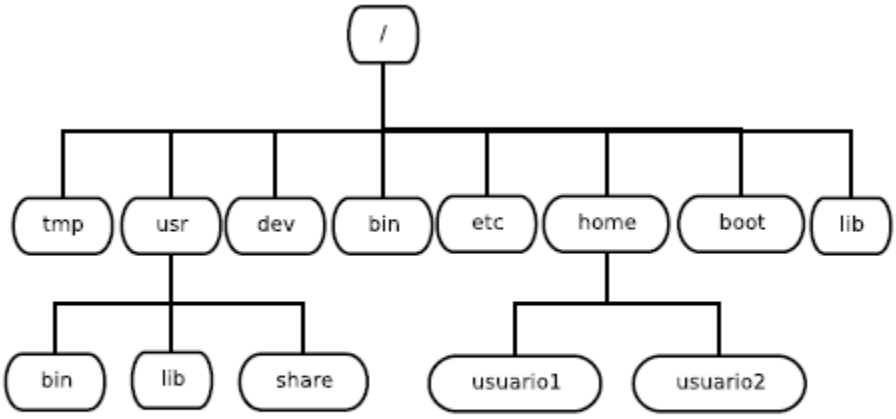

Figura 1: Ejemplo de árbol de directorios

3.2 Directorios raíz, de conexión y actual

Algunos directorios dentro del sistema de archivos tienen una especial importancia, hasta el punto de que se les ha dado un nombre propio. A continuación se detallan los más importantes.

Directorio raíz (/) Todos los archivos y directorios dependen de un solo directorio, denominado directorio raíz o **root**, que se representa por el símbolo slash (/). En caso de que en el sistema tengamos varios dispositivos físicos de almacenamiento secundario (normalmente discos duros), todos deben depender del directorio raíz, y el usuario tratará cada uno de los discos como un subdirectorio que depende de root (aunque tengamos distintos discos físicos, en UNIX todos ellos forman parte de un único disco lógico, al contrario que en otros sistemas, en los que cada disco físico supone, al menos, un disco lógico). A esta operación se la conoce con el nombre de montaje de un subsistema de archivos.

Directorio de conexión (~) También llamado directorio **home**. Cuando un usuario abre una sesión en UNIX, comienza en un lugar específico y privado dentro del sistema de archivos, que es su directorio de conexión. Se trata de un directorio que el administrador del sistema crea o asigna cuando da de alta a un usuario. El usuario podrá crear libremente archivos y subdirectorios en este directorio, y ni el sistema ni los demás usuarios (salvo el administrador) podrán acceder a ellos sin autorización del propietario. Se representa con el símbolo ~ (este símbolo se corresponde con el código ASCII 126 y se obtiene tecleando el número 126 del teclado numérico mientras se mantiene pulsada la tecla [ALT]).

No importa dónde se esté dentro del sistema de archivos, siempre se puede regresar al directorio de conexión con la orden cd sin argumentos.

El sistema facilita una variable de entorno[1], denominada HOME, que contiene el path hasta nuestro directorio de conexión, de tal manera que estas dos órdenes causarían el mismo efecto: *cd y cd $HOME*.

Directorio actual (.) Cada directorio tiene dos subdirectorios especiales. El primero tiene por nombre punto (.), que es el directorio de trabajo o actual (directorio en el que nos encontramos operando en cada momento). El segundo tiene por nombre punto-punto (..) y es el directorio padre del directorio de trabajo.

[1] Una variable de entorno es una variable definida dentro de un intérprete de órdenes. Son importantes porque algunos aspectos importantes del sistema se configuran por medio de las variables de entorno, como los lugares en los que se buscan los archivos ejecutables (variable PATH). Una variable de entorno se puede visualizar mediante la orden echo (echo $PATH) y se pueden ver todas las variables de entorno definidas con la orden set.

Un directorio se considera vacío si solamente contiene punto y punto-punto, y estos dos directorios nunca deben intentar eliminarse, puesto que los directorios punto y punto-punto, realmente, son dos punteros a los directorios actual y ascendente del actual respectivamente, y muchas órdenes UNIX los usan para determinar cuáles son dichos directorios.

Los directorios "." y ".." son imprescindibles para poderse mover dentro del sistema de archivos, por ejemplo, si queremos cambiar de directorio de trabajo al padre del actual, utilizaremos la orden "*cd ..*".

La manera de conocer en todo momento el directorio de trabajo es a través de la orden **pwd** (print working directory) sin argumentos.

3.3 Trayectorias relativas y absolutas

Los archivos se identifican en la estructura de directorios por lo que se conoce como nombre de camino, trayectoria o path name. Fijándonos en la figura 1, distinguiremos las tres formas existentes para referenciar un archivo o directorio:

Camino absoluto o completo Consistente en el conjunto de directorios por los que hemos de pasar, partiendo del directorio raíz, para llegar al archivo o directorio que queremos referenciar. Puesto que siempre parte del directorio raíz, siempre comenzará con el carácter / (este carácter es, igualmente, el separador de los distintos directorios que componen el camino, y no ha de confundirse con el carácter \, que tiene la misma función en otros sistemas operativos). Identifica de modo único a un directorio o archivo dentro del sistema de archivos. Por ejemplo, el camino absoluto del archivo **passwd** será: /etc/passwd.

Camino relativo Hay dos formas de referenciar un archivo usando un camino relativo:

- Partiendo del directorio actual: es el camino que deberemos dar para referenciar un archivo o directorio, tomando como punto de partida no el directorio raíz, sino el de trabajo actual. Por ejemplo, si nuestro directorio actual es /etc, la cadena httpd/httpd.conf (camino relativo al directorio actual) identifica al archivo /etc/httpd/httpd.conf (camino absoluto). En los caminos relativos podremos utilizar los identificadores "." y ".." para ascender por la jerarquía de directorios. Por ejemplo, si nos encontramos en el directorio /usr y queremos hacer referencia al archivo passwd, su camino relativo será: ../etc/passwd.

- Partiendo del directorio de conexión: es el camino que deberemos dar para referenciar un archivo o directorio, tomando como punto de partida nuestro directorio de conexión, independientemente de en qué directorio nos encontremos en cada momento. Por ejemplo, si nuestro directorio de conexión fuera

/home/pepe, el camino para referenciar el archivo passwd sería ~/../../etc/passwd2, sea cual fuere nuestro directorio actual.

3.4 Denominación de archivos

El nombre de un archivo en UNIX puede tener hasta 255 caracteres y aunque no existe el concepto de extensión de un archivo, es posible incluir el carácter ' . ' tantas veces como se desee, la única consideración a tener en cuenta en este sentido, es la referente a los nombres de archivo cuyo primer carácter es '.' a los que históricamente se denominan archivos ocultos, ya que por defecto no son mostrados por la orden ls. Una denominación más precisa para los archivos ocultos, sería la de archivos sin interés habitual, ya que se suelen utilizar para configurar aplicaciones y normalmente el usuario no tiene interés en verlos.

Aunque en principio cualquier carácter es válido para construir un nombre de archivo, es recomendable no emplear caracteres con significado especial para el intérprete de órdenes pues esto complica el acceso a los mismos al ser necesario emplear mecanismos de escape para evitar su interpretación.

Es importante tener en cuenta que los intérpretes de órdenes de UNIX consideran distintos los caracteres en mayúsculas de los caracteres en minúscula. Por lo tanto no son equivalentes los archivos: programa.c y Programa.c. Ejemplos de nombres de archivos:

```
programa, programa.c, programa.c.primera version, base datos.2001
```

3.5 Niveles y modos de acceso a la información

Todo usuario tiene la posibilidad de crear, modificar y borrar archivos. Cada archivo tiene tres modos de acceso diferentes:

- Acceso para lectura "r". Se permite acceder para leer el contenido del archivo sin modificarlo.

- Acceso para escritura "w". Se permite modificar el contenido del archivo.

- Acceso para ejecución "x". El archivo puede ser ejecutado (por ejemplo un programa compilado o un shell script). Cuando éste permiso actúa sobre el directorio se le denomina permiso de acceso, pues regula la posibilidad de utilizar ese directorio en operaciones de búsqueda y de ser utilizado como directorio de trabajo.

En los sistemas multiusuario se hace necesario controlar el acceso a la información asegurando su privacidad. Cada usuario del sistema tiene un número identificador (UID)

único dentro del sistema y también pertenece a un grupo de usuarios identificado unívocamente dentro del sistema por un número (GID). A estos números se les asocia unos nombres lógicos que se denominan login, en el caso del UID, y nombre de grupo, en el caso del GID. Por lo tanto, una manera de identificar la información perteneciente a un usuario (el caso que nos ocupa son sus archivos), es que cada archivo lleve asociado el identificador de usuario propietario de la información y la del grupo a que este usuario pertenece.

De la combinación de los tres modos de acceso explicados anteriormente (rwx) y los atributos propietario de un archivo y grupo al que pertenece dicho propietario, surgen los denominados niveles de acceso a la información, que son tres:

- Nivel de usuario. Modo de acceso para el propietario del archivo.

- Nivel de grupo. Modos de acceso para cualquier usuario que pertenezca al mismo grupo que el del propietario del archivo.

- Nivel de otros. Modos de acceso para los usuarios del sistema que ni son el propietario del archivo ni pertenecen a su mismo grupo.

Así pues, cada tipo de acceso puede aplicarse a distintos usuarios, al final se suele representar por medio de una cadena de nueve letras agrupadas en tres grupos, cada uno con tres letras. La cadena tiene la forma rwx rwx rwx, de tal manera que el primer grupo representa al usuario, el segundo al grupo y el último al resto de usuarios, y a cada grupo se le asignan permisos de lectura (r), escritura (w) y ejecución (x).

3.6 Cambio de permisos, propietario y grupo de un archivo

Como acabamos de ver, a cada archivo se le asocian un conjunto de permisos de acceso (rwx) relativos a cada uno de los tres niveles de la jerarquía de usuarios: nivel de usuario, nivel de grupo y el nivel otros (formado por el resto de los usuarios del sistema). A continuación veremos las órdenes existentes para manejar los permisos asociados a un archivo.

chmod

La orden chmod (change mode) permite cambiar los permisos de un archivo. La utilización de esta función está restringida exclusivamente al propietario del archivo, cuyos permisos (modo) se quieren modificar, o al administrador del sistema. La sintaxis es la siguiente:

```
chmod modo archivo1 [archivo2 ... ]
```

donde `modo` supone el modo de acceso al archivo que se desea asignar. Hay dos formas diferentes de utilizar la orden:

- Modo absoluto. Consiste en indicar mediante números de tres dígitos para cada nivel de la jerarquía de usuarios, cada uno de los permisos. Los permisos aquí se representan con tres números en octal. La cadena alfabética de 9 dígitos se convierte a binario poniendo un 1 en el caso de que se quiera activar el permiso (rwx) o un 0 en el caso de querer desactivarlo.

  Por ejemplo, si se desea asignar al archivo prog.c permisos rwxr-xr– , la secuencia de dígitos sería: 111-101-100. Con estos dígitos se forman 3 grupos, cada uno perteneciente a los permisos asociados a cada uno de los niveles de la jerarquía de usuarios (111- usuario, 101- grupo, 100- otros), y se asocia a cada uno su valor en octal (7- usuario, 5- grupo, 4- otros).

  Ejemplo:

```
$ ls -l prog.c
-rw-r----- 1 p21g3 lsotm 1024 Sep 3 14:58 prog.c
$ chmod 754 prog.c
$ ls -l
-rwxr-xr-- 1 p21g3 lsotm 1024 Sep 3 14:58 prog.c
```

- Modo relativo. En este caso la referencia a los permisos asignados al archivo se realiza de acuerdo con la siguiente sintaxis:

```
chmod {ugoa}{+-=}{rwx} archivo
```

  El primer campo indica a quién se le modifica el permiso: u (usuario), g (grupo), o (otros), a (todos). El segundo campo indica si se activa (+), se desactiva (-) o se fija (=) un permiso. Cuando se utiliza = los permisos que no aparecen se eliminan. El tercer campo indica sobre qué modos de acceso se actá (r, w, x). Ejemplo:

```
$ chmod a-x prog.c
$ ls -l prog.c
-rw-r--r-- 1 p21g3 lsotm 1024 Sep 3 14:58 prog.c
$ chmod go+w prog.c
$ ls -l prog.c
-rw-rw-rw- 1 p21g3 lsotm 1024 Sep 3 14:58 prog.c
$ chmod gu-rw prog.c
$ ls -l prog.c
-------rw- 1 p21g3 lsotm 1024 Sep 3 14:58 prog.c
$ chmod u=r prog.c
$ ls -l prog.c
-r-------- 1 p21g3 lsotm 1024 Sep 3 14:58 prog.c
```

umask

La orden umask permite fijar la máscara de los permisos que tendrá por defecto cualquier archivo que cree un determinado usuario. Su estructura es:

```
umask [###]
```

Llamado sin parámetros, el resultado son tres dígitos octales que indican de izquierda a derecha el valor de la máscara que determina los permisos iniciales para el propietario, el grupo y para el resto de usuarios. Cada dígito octal de la máscara contiene tres dígitos binarios, un 1 binario indica que cuando se cree un nuevo archivo, el permiso asociado (rwx) será borrado y un 0 binario indica que se activará dicho permiso (al contrario de cómo se construían los permisos en octal con la orden chmod). Así, si se desea que a partir de este momento los archivos y directorios que se creen tengan una máscara con todos los permisos desactivados para los niveles de grupo y de otros y que no se limpie ningún permiso para el nivel de propietario, la orden a introducir será:

```
$ umask 077
```

Esta máscara se aplica negada mediante una operación *and* a los permisos indicados en el momento de la creación de un archivo. Es decir, la operación que se realiza es la siguiente:

```
Permisos_establecidos_en_la_creación & ~valor_umask
```

Se puede comprobar su funcionamiento utilizando la orden touch, que, en principio, sirve para modificar la fecha de modificación de un archivo, aunque también se puede utilizar para crear archivos vacíos. En la página correspondiente del manual viene información detallada de su uso.

chown

La orden chown cambia el propietario de un archivo. Su sintaxis es la siguiente:

```
chown propietario_nuevo archivos
```

Ejemplo:

```
$ ls -l a.out
-rwxr-xr-x 1 p21g3 lsotm 2894 Nov 11 16:51 a.out
$ chown p22g6 a.out
$ ls -l a.out
-rwxr-xr-x 1 p22g5 lsotm 2894 Nov 11 16:51 a.out
```

En muchos sistemas esta orden solo puede ser ejecutada por el administrador del sistema.

chgrp

```
Sintaxis: chgrp grupo nuevo archivos
```

La orden chgrp cambia el grupo al que pertenece un archivo. Es necesario pertenecer al grupo nuevo para poder cambiarlo. Ejemplo:

```
$ ls -l a.out
-rwxr-xr-x 1 p21g3 lsotm 2894 Nov 11 17:04 a.out
$ chgrp itt a.out
$ ls -l a.out
-rwxr-xr-x 1 p21g3 itt 2894 Nov 11 17:04 a.out
```

4. Órdenes relacionadas con el sistema de archivos

A continuación se muestran ciertos aspectos a tener en cuenta de algunas órdenes relacionadas con el sistema de archivos de UNIX. No se pretende dar una redacción exhaustiva del funcionamiento de las órdenes, puesto que para ello se podrá, en cualquier caso, acudir a la ayuda en línea que el propio sistema ofrece, pero sí una pequeña orientación sobre el uso de estas órdenes. La exposición de las órdenes se hará siguiendo una clasificación funcional de éstos.

4.1 Listado de archivos

ls [opciones] [archivo|directorio] Visualiza el contenido de un directorio. Si no se especifica un archivo ni un directorio, se muestra el contenido del directorio de trabajo actual. Las opciones más interesantes son las siguientes:

[-a]: muestra los archivos ocultos (los que empiezan por .)
[-F]: Introduce un carácter / tras los nombres de los subdirectorios, un * tras los archivos ejecutables, un @ para enlaces débiles, un | para FIFOs y nada para archivos normales.
[-l]: lista los archivos en formato largo, incluyendo información adicional.
[-i]: muestra el número de i-nodo asociado a cada archivo.
[-R]: lista recursivamente los subdirectorios encontrados.

A continuación se muestra una posible salida de la orden ls con la opción [-l] para un supuesto archivo llamado fich1, describiendo a continuación cada uno de sus campos:

```
-rw-rw-r-- 1 usuario5 grupo7 576 dic 15 23:32 fich1
```

-rw-rw-r–: modos de acceso al archivo, como se explicó, el propietariodel archivo puede establecer que otros usuarios (para sí mismo, los integrantes de su grupo y el resto de forma independiente) tengan permisos de lectura (r), escritura (w) y ejecuci´on (x) sobre sus archivos y directorios. De esta manera hay tres grupos con tres permisos posibles. El primer carácter indica el tipo de archivo que es (d para un directorio, l para un enlace, - para un archivo regular ...). El resto de caracteres indican los tres permisos vistos para los tres grupos.

1 Número de enlaces fuertes que tiene el archivo. Un enlace es un mecanismo que permite hacer referencia a un archivo a través de diferentes nombres. Con ello se logra un aprovechamiento del espacio en disco. Normalmente este número suele ser 1, si es mayor, indica que existen varias entradas que hacen referencia a la misma información almacenada en disco.

usuario5 Es el código que identifica al usuario propietario del archivo o directorio en el sistema, coincide con el login de la cuenta.

Grupo7 Indica el nombre del grupo propietario del archivo o directorio. Normalmente coincide con el grupo principal al que pertenece el propietario.

576 Tamaño del archivo en bytes.

dic 15 23:32 Fecha de la última modificación del archivo fich1 Nombre del archivo

La distinción entre archivos ordinarios y directorios también se puede apreciar si observamos el primer carácter empezando por la izquierda de cada fila. Las entradas cuyo carácter es una “d” son directorios y los que aparece un “-” son archivos ordinarios. Existe una entrada especial “l” que hace referencia a un enlace simbólico, como ya se ha explicado anteriormente.

cd [ruta] Cambio de directorio. Hemos comentado anteriormente que, por defecto, la secuencia de entrada en UNIX nos sitúa en nuestro directorio de trabajo. Pero al igual que otros sistemas operativos (p. ej. MS-DOS) podemos cambiar de directorio mediante la orden cd directorio. La orden cd sin argumentos vuelve a nuestro directorio de trabajo. Si queremos volver al directorio de nivel superior basta con utilizar *cd ..*

dir [archivos] Lista el contenido de los directorios, es equivalente a ls.

tree [opciones] [directorio] Lista el contenido del directorio en formato árbol, de forma recursiva. Indica también cuántos archivos y directorios existen en el árbol mostrado.

4.2 Visualización de archivos

cat [opciones] [archivos] Visualiza el contenido de uno o más archivos de texto (ASCII) por la pantalla. Si no se le pasa como argumento ningún archivo, leerá de la entrada estándar (teclado) hasta que se pulse Ctrl-D (^D). Una vez hecho esto, mostrará lo que acabamos de escribir. Si se le pasa más de un archivo como argumento, mostrará su contenido secuencialmente (concatenando el contenido de todos ellos). Una utilidad de esta orden es la de visualizar el contenido de un archivo sin tener que editarlo. Otra posible utilidad es la de crear un archivo cuyo contenido sea el contenido concatenado de una lista de archivos. Un ejemplo de este último caso es: *cat lunes martes miércoles jueves viernes > diario*. Esta línea haría que en el archivo diario apareciera el contenido de los archivos lunes, martes, ... de forma ordenada. Esta es una forma de hacer la copia de varios archivos en uno sin que se sobrescriba el contenido.

head [opciones] [archivos] y **tail [opciones] [archivos]** En muchas ocasiones no se pretende visualizar el archivo de texto completo, sino que con algunas líneas nos es suficiente. Las órdenes head y tail se pueden utilizar para visualizar las primeras o últimas líneas de un archivo de texto respectivamente. Por defecto, este número de líneas es 10, aunque puede cambiarse con la opción [-n]. Por ejemplo, para mostrar las 3 primeras y últimas líneas del archivo arch1:

```
head -n 3 arch1 ; tail -n 3 arch1
```

more [opciones] [archivos] Visualiza por pantalla el contenido de un archivo de texto, pantalla a pantalla. Para avanzar la visualización línea a línea se usa la tecla [INTRO], mientras que para hacerlo pantalla a pantalla se emplea la tecla [ESPACIO] y para terminar la visualización la tecla q(quit).

4.3 Copia, borrado y movimiento de archivos

rm [opciones] archivos_o_directorios Elimina archivos o directorios. Para borrar un directorio, debe estar vacío. Si hacemos rm de un directorio, irán apareciendo recursivamente preguntas de confirmación de borrado para cada uno de los archivos y subdirectorios del directorio. Si hacemos rm de uno o más archivos, con la opción [-i] nos irá pidiendo confirmación de borrado. En cambio, con la opción [-f] fuerza el borrado de los archivos, incluso si están protegidos contra escritura.

cp [opciones] archivos_o_directorios destino Copia archivos_o_directorios (para copiar directorios, es imprescindible la opción [-r]). Como mínimo necesita dos argumentos. El primero es el archivo o directorio que queremos copiar, y el segundo es el nombre del archivo o directorio destino.

Si se hace una copia de un archivo en un archivo destino que ya existe, el contenido de este último se sobrescribirá con el del primero; el sistema pide confirmación de sobrescritura. No se permite la copia de varios archivos en uno (concatenación).

Si se hace una copia de un directorio en un directorio destino que ya existe, el contenido del primero se añade al del segundo. Sí se permite la copia de varios archivos en un directorio y la copia de varios directorios en otro.

mv [opciones] origen destino Su funcionalidad es idéntica a la de cp, excepto en que mv provoca que los archivos o directorios origen desaparezcan de su localización inicial.

4.4 Creación y eliminación de directorios

mkdir [opciones] directorios. Crea directorios.

rmdir [opciones] directorios. Borra directorios vacíos.

5. Entrada, salida y salida de errores estándar. Redirecciones y tuberías

En UNIX, cuando un proceso abre un archivo, el kernel le entrega un número entero positivo, conocido como descriptor de archivo, que el proceso utilizará para realizar operaciones posteriores de E/S. Todo proceso en el momento de ser creado tiene asignados tres archivos denominados entrada estándar, salida estándar y salida de errores estándar, tal y como se refleja en la tabla 2.

Decriptor	Denominación
0	**stdin** (entrada estándar) por defecto está asociada con el teclado del terminal
1	**stdout** (salida estándar) asociada con la pantalla del terminal
2	**stderr** (salida de errores estándar) asociada con la pantalla del terminal. Se usa para sacar mensajes importantes cuando se producen situaciones anómalas

Tabla 2Descriptores de archivos estándar

Los descriptores estándar de entrada, salida y error pueden ser redirigidos para conseguir que la información se envíe o se tome de un archivo en lugar del terminal del usuario.

5.1 Redirección de la entrada estándar (<)

Este operador permite redirigir la entrada de datos de un proceso desde un archivo sustituyendo, así, al teclado del terminal.

Ejemplo: La siguiente orden enviará el contenido del archivo saludo.txt al usuario aldo. La orden mail tomará la información necesaria para su funcionamiento (el texto del mensaje) del archivo saludo.txt en lugar del terminal.

```
$ mail aldo < saludo.txt
```

5.2 Redirección de la salida estándar (>, >>)

Estos operadores permiten redirigir la salida de un proceso a un archivo en lugar de la pantalla del terminal. El primero de ellos, >, redirige eliminando el contenido anterior del archivo y el segundo, >>, redirige añadiendo la salida del proceso al final del archivo. En ambos casos, el archivo se crea si no existía.

Ejemplo: En este caso la salida de la orden *banner* no será enviada al terminal sino que será almacenada en el archivo saludo.txt.

```
$ banner -w40 hola
$ banner -w40 hola > saludo.txt
```

5.3 Tuberías

Puede conectar procesos usando el operador de tubería (|). En Linux, a diferencia de MS-DOS, los procesos conectados mediante tuberías se pueden ejecutar de manera simultánea y se vuelven a planear automáticamente cuando los datos fluyen a través de ellos. A modo de ejemplo sencillo, podría usar el comando sort para organizar la salida de ps.

Ejemplo sin tuberías, tendríamos que realizar varios pasos:

```
$ ps > salidaps.txt
$ sort salidaps.txt > salidapsordenada.txt
```

Ejemplo más elegante conectando los procesos mediante una tubería:

```
$ ps | sort > salidapsordenada.txt
```

Probablemente querrá ver la paginación de los datos de salida en la pantalla, así que puede conectar un tercer proceso, more, todo en la misma línea de comando:

```
$ ps | sort | more
```

No existe prácticamente límite alguno en el número de procesos que se pueden conectar.

6. Ejercicios

1. ¿Qué sistema operativo se encuentra instalado en el laboratorio?

2. ¿Cuál es la diferencia entre la utilización de \r, \n, \c en la orden echo?

3. Visualice la hora en el formato siguiente: En el día de hoy (día) del (mes) de (año), a las (horas) horas y (minutos) minutos, termino el ejercicio 3.

4. Muestre todos los archivos del directorio /etc que comiencen por i y terminen por b.

    a. ¿Qué archivos tienen como segunda letra una s?
    b. ¿Qué archivos tienen como tercera letra una consonante?

5. Averigue cuál es su directorio de conexión y cree dentro del directorio de la práctica un directorio que se llame 'tmp'.

6. Cree dos directorios a partir de su directorio tmp, llamados: compartido y publico, con las siguientes características: En compartido solo podrán leer y escribir los componentes de su mismo grupo además de usted. En publico solo podrá escribir usted, pero todo el mundo podrá leer de ´el.

7. Utilice las utilidades de redirección para incluir en la cabecera de un archivo ya creado la fecha actual. Incluya también al final del archivo el número de líneas que lo componen. No es necesario que el archivo resultante sea el mismo que se tomó inicialmente para efectuar estas modificaciones.

8. Determine qué regula cada uno de los permisos de acceso (r, w, x) aplicados a un directorio. En particular indique qué permisos mínimos son necesarios para:
    a. Convertir el directorio en directorio de trabajo.
    b. Copiar un archivo dentro del directorio.
    c. Eliminar un archivo del directorio.
    d. Ver los nombres de los archivos que hay en el directorio.
    e. Ver el contenido de los archivos que hay en el directorio.

Explique DETALLADAMENTE (incluyendo ejemplos de las pruebas realizadas) sus conclusiones.

9. Con la ayuda de la orden man, averigurar el contenido y los campos del archivo passwd ¿Es lo mismo el archivo que la orden passwd?

10. Crear dentro de compartido un directorio llamado permisos. ¿A qué se debe que tenga los permisos iniciales que se le han otorgado? ¿Es esto modificable? ¿Cómo? Crear otro directorio al mismo nivel llamado perms2 cuyos permisos estén automáticamente deshabilitados.