UNIVERSIDAD DE ALCALÁ

Departamento de Automática

Grado en Ingeniería Informática

Práctica 1: La consola de Linux (I)

Sistemas Operativos

Índice

1. Competencias asociadas a la práctica

1. Comprender qué es el software libre así como las diferencias respecto al software propietario.
2. Asimilar la importancia de manejar adecuadamente la consola de Unix.
3. Diferenciar los conceptos de sistema operativo y distribución.
4. Saber utilizar órdenes útiles en Unix como: *clear*, *date*, *echo* y *who*.
5. Tener capacidad de buscar información a través del sistema de ayuda de Unix y a través de Internet.
6. Saber utilizar órdenes de búsqueda de información en Unix: *man*, *whatis*, *apropos* e *info*.
7. Saber utilizar los caracteres comodines en las órdenes de Unix.
8. Comprender la estructura de directorios de un sistema Unix.
9. Comprender la diferencia entre directorio de conexión, directorio actual y directorio raíz.
10. Saber utilizar los directorios "." (punto) y ".." (punto-punto) para moverse por el sistema de archivos de Unix.
11. Saber utilizar la trayectoria absoluta y relativa para nombrar archivos en las órdenes de Unix.
12. Ser capaz de moverse con soltura por la estructura de directorios de un sistema Unix.
13. Ser capaz de realizar las tareas comunes en la gestión de archivos, como crear, borrar copiar y mover archivos (y directorios, como tipo especial de archivos en Unix).
14. Ser capaz de visualizar el contenido de archivos de tipo texto utilizando órdenes de Unix: *cat*, *head* y *more*.
15. Ser capaz de comprender la diferencia entre enlace fuerte y enlace débil y su importancia a la hora de compartir información.
16. Ser capaz de manejar enlaces fuertes y enlaces débiles en la consola de Unix utilizando la orden *ln*.
17. Ser capaz de gestionar adecuadamente los permisos y propietarios de los archivos en un sistema Unix con las órdenes *chmod*, *chown*, *umask* y *chgrp*.
18. Ser capaz de realizar tareas comunes en el manejo de un SO como instalar o desinstalar paquetes a través de la consola: órdenes *aptget* y *aptcache*.
19. Ser capaz de realizar tareas comunes para archivar infomación a través de la consola: empaquetar archivos con la orden *tar*, o comprimir/descomprimir archivos con las órdenes *gzip* y *gunzip*, respectivamente.

2. Introducción

El estudio de los Sistemas Operativos (SSOO) tradicionalmente se ha centrado en SSOO de tipo Unix. Los sistemas Unix son un conjunto de SSOO cuyos orígenes se remontan a la Universidad de Berkeley en los años 60, y han ejercido una gran influencia en el desarrollo de la informática hasta la actualidad. En esencia, Unix es un sistema operativo (SO) creado en los años 60 por Ken Thompson y Dennis Ritchie que ha tenido una gran importancia en la historia y presente de la informática. Actualmente existen diversos SSOO creados bajo la inspiración de UNIX, de los que se dicen que son SSOO tipo Unix. Tal vez el SO tipo Unix más conocido y extendido en la actualidad es Linux, sin embargo hay muchos más: Solaris, AIX, FreeBSD, NetBSD, etc.

El motivo fundamental de utilizar Unix es que se plasman en él de una manera muy clara diversos conceptos básicos que en otros SSOO quedan ocultos por las capas de abstracción. Por lo tanto, existe casi una unanimidad en las asignaturas de SSOO de todas las universidades para utilizar Unix como base. Hay que tener en cuenta que Unix ha marcado la evolución de la informática en muy distintos ámbitos. Por ejemplo, Unix revolucionó el mundo de los SSOO al ser el primer SO programado en un lenguaje de alto nivel, C, que de hecho se diseñó con la tarea específica de implementar SSOO. La práctica totalidad de lenguajes de programación utilizados en la actualidad han derivado directamente de C (C++, Java, C#), o bien han estado muy influidos por él (PHP).

Otro área en donde Unix ha desempeñado un papel fundamental es en Internet, cuyo germen surgió a finales de los años 60 y se alineó con Unix desde sus inicios. Unix sirvió de plataforma para desarrollar toda la infraestructura de Internet, e Internet, a su vez, sirvió para que Unix se desarrollara como un SO orientado al trabajo en red. Finalmente, y desde una perspectiva más práctica, los sistemas Unix son los más extendidos en grandes sistemas, fundamentalmente servidores, por lo que existe además un gran interés práctico. Hay un mercado laboral importante para administradores de Unix.

La práctica está orientada para servir como primer contacto con un sistema Unix, en nuestro caso Linux. Se pretende mostrar el funcionamiento básico de Unix, y facilitar los conocimientos elementales para poder trabajar bajo esta plataforma.

Linux es un SO que tiene una historia y un contexto social bastante particular que ha impregnado su evolución. Conocer este contexto es fundamental para entender por qué Linux es lo que es. A continuación, se hace una breve introducción a todas estas cuestiones de carácter más filosófico e histórico que técnico.

2.1. Software libre

Linux es el producto más conocido de un movimiento mucho más amplio conocido como software libre. El software libre es aquel software que está sujeto a un tipo de licencia que, lejos de restringir la libertad del usuario, está diseñada para preservar dicha libertad. En este contexto, se entiende que la libertad del usuario está constituida por cuatro libertades fundamentales: (1) libertad para ejecutar, (2) libertad para distribuir, (3) libertad para estudiar el programa (ver el código fuente), y finalmente (4) libertad para modificar el programa. Por lo tanto se considera que es software libre todo aquél software que el usuario puede utilizar sin ningún tipo de restricción, copiarlo y distribuirlo, cuyo código fuente está disponible para ser estudiado, y que además se puede utilizar

Figura 1: Logotipo del proyecto GNU.

para modificar el programa o utilizar parte de ese código en terceras aplicaciones. La restricción fundamental que se introduce es que todo programa que utilice código libre, debe ser a su vez libre.

El movimiento del software libre fue creado por Richard Stallman a principios de los años setenta. Por aquél entonces Stallman trabajaba en el Laboratorio de Inteligencia Artificial del MIT, y lo abandonó para fundar la *Free Software Foundation* (FSF) y el proyecto GNU (*Gnu is not Unix*), en el que quería implementar un sistema Unix completo, esto es, un SO Unix libre, junto con todas las aplicaciones necesarias para el usuario, también libres. GNU, en vez de empezar creando un SO libre, empezó a implementar todo el ecosistema de aplicaciones de desarrollo: editores de texto (emacs), compiladores (gcc), depuradores (gdb), y un largo etcétera.

Actualmente, el software libre ha evolucionado para generar una serie de corrientes que amplían el concepto de libertad dentro un contexto tecnológico para abordar otros ámbitos distintos del software. Tal vez el más conocido sea el movimiento de la cultura libre, que defiende un acceso libre a recursos culturales, y cuyo máximo exponente es la Wikipedia, por todos conocida. Además, hay notables intentos para ampliar la filosofía del software libre a creaciones sujetas a propiedad intelectual, como son textos, documentación o imágenes. En este ámbito es conocido *Creative Commons*, un conjunto de licencias que permite al autor introducir distintas restricciones en la utilización de sus obras.

2.2. Linux

Linux es un SO de tipo Unix, desarrollado inicialmente por el finlandés Linus Torvals. En sus orígenes, Linux nació como un divertimento de Linus, que quería probar las funciones avanzadas de su recién estrenado 386. Llegado un momento en el que el núcleo de Linux era mínimamente publicable, Linus decidió publicar su código en Internet bajo una licencia libre a principio de los 90. A partir de entonces, distintos voluntarios distribuidos por todo el mundo empezaron a contribuir con código y Linux experimentó una rápida evolución, marcando lo que sería la evolución del software libre hasta la actualidad: redes de desarrolladores colaborando a través de Internet, al contrario del modelo centralizado de GNU alrededor de la FSF. Si bien hoy en día este modelo de desarrollo (a veces nombrado *modelo de bazar*) está coexistiendo con otros modelos más centralizados, fundamentalmente debido a la cada vez mayor presencia de grandes empresas en torno a Linux y el software libre, y a la creación de grandes fundaciones en torno a los proyectos más significativos, como Firefox o Gnome.

Un elemento fundamental de Linux que suele generar confusión es que cuando se

Figura 2: Tux, la mascota de Linux.

habla de Linux, estrictamente hablando, se habla únicamente del SO. Es decir, Linux es un SO, y por lo tanto no tiene aplicaciones que, en definitiva, es lo que el usuario necesita. Precisamente cuando Linux nació, el proyecto GNU maduró lo suficiente como para tener una amplia gama de aplicaciones de alta calidad operativas, desde compiladores (gcc), a editores de texto (vi, emacs), herramientas de desarrollo (make, gdb) y un largo etcétera. La unión de dichas herramientas con Linux dio lugar a lo que se conoce como GNU/Linux, un sistema perfectamente funcional basado en software libre, tal y como el proyecto GNU pretendía conseguir.

Actualmente Linux es un SO muy extenso, fundamentalmente en entornos de servidores, pero también en equipos de escritorio y sistemas empotrados. Linux soporta un amplio abanico de arquitecturas y ámbitos de aplicación, desde servidores de altas prestaciones (el ordenador más potente del mundo corre bajo Linux), hasta la telefonía móvil con Android (un SO basado en Linux). Además, las colaboraciones en el desarrollo se han venido profesionalizando al contar con el respaldo de gigantes de la industria como IBM o Intel.

Linux es utilizado en un amplio rango de aplicaciones:

- Informática de usuario, contando con un amplio rango de entornos gráficos (GNOME, KDE) y herramientas (OpenOffice, Firefox).
- Servidores. Es donde Linux tiene la mayor cuota de mercado. Linux se suele utilizar con aplicaciones como Apache (servidor web), BIND (servidor DNS) o Sendmail (servidor de correo electrónico).
- Sistemas empotrados. Linux se ha portado a plataformas utilizadas habitualmente en sistemas empotrados, y por lo tanto es posible encontrar coches que utilizan Linux o incluso satélites espaciales.
- Dispositivos inteligentes. Un sector con un fuerte crecimiento en la actualidad son los dispositivos inteligentes, en donde los *smartphones*, como el *iPhone* o *Samsung*, son cada vez más habituales. En este contexto, SSOO pesados son adaptados para poderse utilizar con este tipo de dispositivos. Como ejemplo encontramos Android, un SO desarrollado por Google y que está basado en Linux.

2.3. Distribuciones

Un concepto que suele generar confusión es el de distribución. Como se ha comentado anteriormente, Linux es un SO, y sólo un SO. Por lo tanto, Linux por sí mismo es inútil para el usuario, necesita además un conjunto de aplicaciones, un programa de instalación y soporte como actualizaciones de software. Normalmente todos estos elementos son empaquetados en lo que se denomina una distribución. Así pues, distribución es el conjunto formado por Linux, distintas aplicaciones, y un programa de instalación.

Distribuciones hay muchas, y aunque todas utilizan Linux, son diferentes entre sí porque incluyen distintos programas y, por lo general, están enfocadas a distintos tipos de usuario. A continuación se enumeran las distribuciones más conocidas:

Ubuntu Distribución relativamente reciente que ha alcanzado una gran popularidad. Está basada en Debian, pero utiliza software más actualizado y con un proceso de instalación más amigable. Es conocida por su acabado cuidado, y por tener un buen equilibrio entre facilidad de uso y potencia.

Debian Está mantenida por un grupo de voluntarios y se base estrictamente en los principios del software libre, de tal manera que no se incluye en la distribución ningún programa que no se ajuste perfectamente a los parámetros del software libre. Tiene ciclos de liberación muy largos, y por lo tanto el software suele estar algo desactualizado. Por el contrario, es la distribución más testeada, y por lo tanto estable, lo que la hace especialmente indicada para servidores. Se la considera una distribución poco amigable y sus seguidores suelen ser usuarios avanzados.

Red Hat Red Hat es una empresa que por mucho tiempo tenía la distribución de Linux más popular, hasta que decidió dedicarse exclusivamente al entorno empresarial, por lo que dejó de mantener la distribución de Red Hat para dedicarse en exclusiva a la Red Hat Enterprise Linux.

Fedora Cuando Red Hat dejó de mantener su distribución no empresarial un grupo de voluntarios decidió seguir manteniendo la distribución de Red Hat, que se pasó a llamar Fedora.

SUSE Distribución alemana pensada para usuarios no expertos, por lo general cuenta con una presentación muy cuidada y fácil manejo.

Ejercicios

1. Buscar la diferencia entre software libre, software propietario y dominio público en Internet.
2. ¿Es Linux el único SO libre? En caso negativo, buscar ejemplos de SSOO libres.
3. ¿Es el software libre necesariamente gratis? Utilizar Internet en caso de ser necesario.
4. Enumerar al menos cinco distribuciones.

5. Seleccionar las tres principales distribuciones y describir a quién está dirigida la distribución y en qué contextos es recomendable utilizarla.

6. Seleccionar cinco tareas comunes que se realicen en el ordenador, como navegar en Internet, chatear, o editar textos, y buscar un programa libre con el que se pueda hacer cada tarea.

7. ¿Todos los programas libres están hechos para un entorno Unix?

8. ¿Qué diferencia existe entre Unix y Linux?

3. Introducción a la práctica

Actualmente Linux ofrece un entorno gráfico muy potente, atractivo, y fácil de utilizar, con un gran número de aplicaciones que satisfacen un amplio espectro de necesidades. En torno a Linux hay una leyenda negra que afirma que es un sistema complejo, difícil de utilizar, y en gran medida se debe a la consola; y precisamente el objetivo de la práctica es introducir el manejo de la consola.

La consola es un terminal en modo texto con la que se accedía a los SSOO antes de la aparición de los entornos gráficos.

Actualmente el uso de la consola no es necesario para manejar un SO a nivel de usuario, pero sigue siendo necesario su conocimiento para un profesional de la informática y para fines didácticos. Es necesario aprender el manejo de la consola porque los entornos gráficos abstraen muchos elementos del SO para simplificar su uso, lo cual es muy adecuado para un usuario, pero un profesional debe conocer lo que hay por debajo. Por otra parte, estos conceptos son los que se estudian en la asignatura, y, por lo tanto, manejar la consola supone un complemento necesario a las clases teóricas. Los entornos gráficos son una extraordinaria ayuda para los usuarios, carecen de importantes limitaciones cuando se pretende extraer el máximo rendimiento al SO. En el caso de Unix la consola es especialmente potente, mucho más que el entorno gráfico, lo cual es especialmente importante en la administración de sistemas. Por último, una vez que se supera la fase de aprendizaje (y probablemente también superado el trauma de enfrentarse a la consola por primera vez), el manejo de la consola resulta mucho más cómoda que el entorno gráfico; el teclado es más rápido que el ratón.

3.1. Entorno del laboratorio

El laboratorio está formado por un PC que va a actuar como servidor centralizado donde trabajará cada alumno, y un conjunto de ordenadores personales conectados entre sí a través de una red de área local formada por los ordenadores del laboratorio L4, L5 y L6. El servidor dispone de su propia versión de Unix ya instalada. A cada grupo de laboratorio se le asignará un nombre de usuario para conectarse a dicho servidor. Los puestos de trabajo tienen instalado como SO local Windows o Linux, sin embargo para las prácticas se utilizará Linux.

Para simplificar la gestión de los equipos del laboratorio se ejecutará Linux dentro de una máquina virtual. Una máquina virtual consiste en un programa que simula ser un

ordenador físico sobre el cuál puede ejecutarse cualquier programa, por ejemplo, un SO. Dentro de un entorno virtual se opera exactamente igual que si se estuviera trabajando sobre el hardware físico, pero se facilita mucho la gestión, por ejemplo, reponer un sistema virtualizado dañado es tan sencillo como copiar un archivo, mientras que reponer un sistema real implica toda la reinstalación del SO. Otra ventaja es que una máquina virtual es copiable, es decir, la máquina virtual del laboratorio se puede copiar en una memoria USB y luego utilizarse en casa. Por último, una ventaja no menor, es que con una máquina virtual se puede instalar Linux en cualquier ordenador sin necesidad de reparticionar el disco duro, con los riesgos que tiene dicha operación.

Para acceder a la máquina virtual hace falta seguir el siguiente proceso. Primero se arranca la máquina siguiendo las instrucciones proporcionadas por el profesor de la asignatura. Posteriormente se realiza el *login* al sistema, con el usuario y contraseña que se indiquen en clase. En este momento se está dentro de un sistema Linux no virtualizado (el *host*), por lo que es necesario arrancar la máquina virtual, cuyo icono aparece en el escritorio. A partir de entonces se abre una ventana en la que aparece todo el proceso de arranque de Linux, todo lo que se haga dentro de la ventana será igual a estar delante de una máquina física con Linux. Una vez haya arrancado es necesario volver a introducir los datos del *login*, y a partir de entonces se está en el entorno virtual en donde se realizarán las prácticas con seguridad.

La totalidad de las prácticas se realizarán dentro del terminal, haciendo caso omiso al entorno gráfico que, en principio, resulta trivial de utilizar y por lo tanto no se explicará. Existen varias formas de acceder al terminal, la más sencilla es ir al menú y seleccionar la aplicación “Terminal” o “Consola”.

3.2. Cambio de contraseña

Una de las primeras labores a realizar si todavía no tiene contraseña es introducir una. Para introducir o modificar una contraseña se utiliza la orden *passwd*. A los efectos de las prácticas de la asignatura NO se deberá CAMBIAR EL PASSWORD sin previamente consultar con los profesores del laboratorio.

Antes de cambiar la contraseña, passwd pide la contraseña actual como medida de seguridad. A continuación, se pide la nueva contraseña y, si ésta es válida, se vuelve a pedir para confirmar que el usuario no se confundió al teclearla, puesto que los caracteres no aparecen en pantalla. Sólo entonces se cambia la contraseña. Es conveniente cambiar la clave de acceso periódicamente por razones de seguridad.

Una recomendación básica aplicable a cualquier situación es que a la hora de elegir contraseña esté formada por 6 caracteres como mínimo, y al menos uno de los caracteres sea numérico o un carácter especial. Con esto conseguimos dificultar mucho los ataques de fuerza bruta y los ataques de diccionario. Ejemplos de contraseñas son los siguientes.

- Contraseñas recomendadas: *asadc0*, *g0tr2is*, *!as#ws*, etc.
- Contraseñas no recomendadas: *hola*, *inicio*, *asf1*, *fernando*, etc.

Es importante resaltar que Unix distingue entre mayúsculas y minúsculas, por lo que las contraseñas: *Tracy123*, *tracy123*, *trAcy123* y *TraCY123* son distintas.

Un ejemplo de cambio de contraseña puede ser el siguiente.

```
$ passwd
Changin password for so25
Old password: (contraseña actual)
New password: (nueva contraseña)
Re-enter new password: (teclear de nuevo la nueva)
$ (la contraseña ha sido cambiada)
```

3.3. Procedimiento de desconexión

Al igual que es necesario seguir una secuencia determinada para poder acceder al sistema, también es necesario indicar cuándo se da por finalizada una sesión de trabajo. De esta forma se libera el terminal para que pueda ser empleado por otro usuario y se evita que otros usuarios puedan acceder a nuestra información, dañarla o destruirla. La desconexión se consigue sin más que teclear la orden *exit*, o bien pulsando simultáneamente las teclas *Ctrl+d*, a lo que el sistema responde con el mensaje

```
logout
```

indicando que da por finalizada la conexión. Si lo que queremos es apagar el sistema, deberemos introducir la orden

```
shutdown -h now
```

o bien apagarlo mediante el entorno gráfico.

Ejercicios

1. Realizar un *login* al equipo del laboratorio.
2. Si se dispone de equipo propio, instalar una distribución de Linux en dicho equipo.
3. Realizar un *login* al servidor, por medio de la orden `ssh` (`ssh 172.29.22.64`) y con el *login* y contraseña de tu usuario. ¿Qué es lo que ha ocurrido? Utiliza Internet para averiguar la respuesta.

4. Manejo básico de la consola

4.1. Intérprete de órdenes (la consola)

Existen diversos mecanismos para que un usuario se comunique con el SO. Por lo general, los usuarios interaccionan con procesos, y los procesos se comunican con el SO por medio de algo denominado *llamadas al sistema*. A veces es necesario una comunicación más directa con el SO, de manera que el usuario pueda realizar operaciones más básicas, como mover un archivo. Los SO modernos suelen utilizar interfaces gráficas, sin embargo también posibilitan una interfaz más rudimentaria, basada en texto, llamado intérprete de órdenes, consola, *shell*, o terminal.

En definitiva, un intérprete de órdenes es un programa en modo texto encargado de hacer de intermediario entre el usuario y el SO. El usuario introduce órdenes en el intérprete, y éste las interpreta, como bien dice su nombre, y realiza las acciones oportunas con el SO para ejecutar dichas órdenes. Análogamente, el intérprete obtiene la salida del SO y se lo presenta al usuario. Los intérpretes de órdenes funcionan en modo texto, y es la manera habitual de trabajar, por ejemplo, en la administración de servidores UNIX, porque ofrecen una interfaz sencilla y extremadamente potente. El intérprete es la forma de comunicación más cercana al SO que, sin programar, un usuario puede utilizar. Por lo tanto, es algo que todo profesional de la informática, y más concretamente un ingeniero, debe conocer.

Un intérprete de órdenes es un programa normal y corriente. De hecho, existen varios intérpretes disponibles: *tch*, *sh*, o *csh*, por ejemplo. Con diferencia, el intérprete más extendido es *bash* (*Bourne Again SHell*), que es el que se utiliza en el laboratorio. Por ejemplo, para ver la version de *bash* que se está ejecutando se puede introducir la siguiente orden:

```
bash --version
```

y se obtendrá una salida similar al siguiente texto

```
simpson@evergreen:~$ bash --version
GNU bash, version 3.2.48(1)-release (i486-pc-linux-gnu)
Copyright (C) 2007 Free Software Foundation, Inc.
```

La línea en la cual se introducen las órdenes se conoce como *prompt*, y es importante porque muestra información útil para el usuario: en primer lugar el nombre del usuario, *simpson* en este caso, y el nombre de la máquina (*evergreen*). A continuación, aparece el directorio de trabajo (~), que se explicará posteriormente.

Con el ejemplo anterior se puede apreciar la forma general de una orden en Unix. La estructura general de cualquier orden en Unix es:

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

siendo *nombre* el nombre del programa u orden a ejecutar, y suele ir seguido por una serie de modificadores con sus respectivos parámetros. Lo más común es que todas las órdenes admitan modificadores que suelen comenzar con el signo - (menos) y por una única letra, o bien – (dos guiones) seguidos de una palabra, que es más fácil de memorizar, pero más largo de escribir. Por lo general, todo modificador se puede poner tanto en formato reducido como en formato nemotécnico. El propósito de los modificadores es 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, la orden *ls* muestra el contenido de un directorio:

```
homer@evergreen:~$ ls nombres.txt
```

sin embargo podemos obtener más detalles de los contenidos del directorio con el modificador “-l”:

```
homer@evergreen:~$ ls -l
total 4 -rw-r--r-- 1 homer homer 26 2010-01-28 18:20 nombres.txt
```

los parámetros se pueden concatenar, por lo tanto la orden

```
$ ls -l -a
```

es equivalente a la orden

```
$ ls -la
```

La orden *ls* es el equivalente en Unix a 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 señalar que Unix casi siempre distingue las mayúsculas de las minúsculas, al contrario que otros sistemas operativos como MS-DOS o Windows.

4.2. Atajos

Dado lo mucho que se utiliza el terminal, existe una multitud de trucos para poder trabajar cómodamente con el terminal, de hecho, una vez que se conocen, resulta mucho más rápido y cómodo trabajar con el terminal que con el interfaz gráfico.

Tal vez el atajo más utilizado es el tabulador, que permite completar órdenes y nombres de archivos. Por ejemplo, si se escribe "l" y luego se pulsa el tabulador, aparecerá una lista de todas las órdenes que empiezan por "l." Lo mismo se aplica al autocompletado de nombres de archivos. Otro truco que se utiliza constantemente es el historial de órdenes, al que se accede utilizando las teclas del cursor arriba y abajo.

Aunque no sean atajos propiamente dichos, es necesario conocer que existen combinaciones de teclas especiales que tienen funciones útiles. El más importante es *Crtl+c*, que cancela la ejecución de una orden. Otro carácter especial muy utilizado es *Crtl+z*, que detiene temporalmente la ejecución de una orden, devolviendo al usuario al terminal. Esta cuestión se verá más en detalle en la siguiente práctica. Se puede retomar la ejecución de la orden ejecutando *fg*. Se pueden probar estas dos funciones, por ejemplo, mediante la orden *sleep 10*, que detiene la ejecución del terminal durante 10 segundos.

4.3. Ayuda

Existe un número ingente de órdenes en Linux, y cada orden cuenta con un gran, a veces grandísimo, número de parámetros. Evidentemente resulta imposible aprender de memoria toda esta información, además de innecesario. Linux cuenta con un excelente sistema de ayuda, y, de hecho, es habitual utilizarlo de manera constante incluso en un

ámbito profesional con años de experiencia en administración. Conocer el sistema de ayuda es imprescindible para poder trabajar cómodamente con Linux.

En primer lugar, la mejor ayuda que hay en todo lo relacionado con Linux no está en Linux, sino en Internet. Existe una extensísima cantidad de documentación de alta calidad, y un buen buscador nos puede facilitar la información que necesitamos con rapidez. Es necesario acostumbrarse a utilizar Internet para buscar información.

Casi todas las órdenes admiten el modificador –help y -h (ambos son equivalentes), que muestran ayuda sobre dicha orden. También existen herramientas específicas para encontrar información, la más conocida, y utilizada, es la orden *man*, y se usa tal y como se detalla a continuación:

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.). Para moverse en la pantalla de ayuda, se puede utilizar las teclas de cursor y se sale pulsando la tecla "q". El manual está dividido en un conjunto de secciones de las cuales las tres primeras son estándar:

- Sección 1 - Ordenes de usuario
- Sección 2 - Llamadas al sistema
- Sección 3 - Biblioteca 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 ...
SYNOPSIS man [-acdfFhkKtwW] [-m system] [-p string] [-C config_file] [-M path] ...
DESCRIPTION man formats and displays the on-line manual pages. This version knows ...
```

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
```

Aparte de las páginas *man*, existen otras órdenes útiles para buscar información; las dos más importantes son *apropos* e *info*. Se deja como ejercicio buscar información sobre dichas órdenes utilizando *man*.

4.4. Algunas órdenes útiles

A continuación se muestran algunas órdenes útiles con las que ir practicando el manejo de la consola.

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 actuales 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 del 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 '+Día: %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:

\n carácter de salto de linea

\r carácter de retorno de carro (*carriage return*)

\t carácter de tabulación

\v carácter de tabulación vertical

**** escribe el carácter \ (*backslash*)

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
```

4.5. Caracteres comodines

En Unix existen caracteres especiales cuya combinación permite que el intérprete de órdenes los sustituya por un grupo de caracteres. Estos caracteres se denominan caracteres comodín, y son fundamentales para nombrar grupos de archivos al ejecutar una orden. Por ejemplo, el carácter '*' especifica cualquier secuencia de caracteres, mientras que "*.*" se interpreta como dos secuencias de caracteres cualquiera separados por un punto. De entre los caracteres especiales existentes destacamos los siguientes:

***** Se sustituye por un conjunto cualquiera de caracteres (cero o más 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 último una 'b'
```

Aparte de los caracteres comodín, existen otros caracteres cuyo significado es especial para la *shell*. Entre estos destacamos: < , > , | , &, cuyos significados se explicarán en prácticas posteriores.

Ejercicios

1. Buscar en Internet un tutorial de manejo elemental de *bash* y leerlo.
2. Utilizar las páginas *man* para encontrar los argumentos que admite la orden *ls*.
3. Utilizar *man* para encontrar información sobre las páginas del propio *man*.
4. Averiguar cómo y para qué se utilizan las órdenes *apropos* e *info*.
5. Averiguar qué versión de Linux se encuentra instalado en el laboratorio.
6. ¿Cuánto espacio ocupa el archivo */etc/passwd*? Utiliza los atajos descritos.
7. ¿Cuál es la diferencia entre la utilización de \r, \n, \c en la orden echo?
8. Visualizar la hora en el formato siguiente: En el día de hoy (dia) del (mes) de (año), a las (horas) horas y (minutos) minutos.
9. Mostrar 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?
10. Con la ayuda de la orden *man*, averiguar el contenido y los campos del archivo *passwd*. ¿Es lo mismo el archivo que la orden *passwd*?
11. Encontrar un navegador Web en modo texto que pueda utilizarse dentro del terminal.
12. Encontrar una orden que genere claves de forma aleatoria.
13. Buscar los juegos que están instalados en el sistema.

5. Introducción al sistema de archivos de Unix

Comprender el funcionamiento de Unix es en gran parte comprender cómo funciona su sistema de archivos. Un **fichero** o **archivo** se puede definir como un conjunto de datos con un nombre asociado. Los archivos suelen residir en dispositivos de almacenamiento secundario, tales como cintas, discos duros, CD-ROMs, DVDs, discos flexibles, etc. Sin embargo, en Unix la ubicación de los archivos es transparente, es decir, estén donde estén, todo el sistema se muestra al usuario en un único directorio que empieza en el raíz (/), al contrario de, por ejemplo, Windows, en donde existen unidades (c:, d:, a:, ...). La forma de operar de Unix permite un acceso más elegante a los archivos, además de otras ventajas en la administración del sistema.

Una cuestión fundamental de Unix es que todo en este sistema operativo se representa mediante archivos, desde los archivos regulares, hasta los dispositivos como, por ejemplo, tarjetas de red. Este hecho, en principio sorprendente, es una abstracción que simplifica enormemente muchas tareas, ya que permite hacer operaciones muy complejas utilizando las órdenes comunes para manipular archivos.

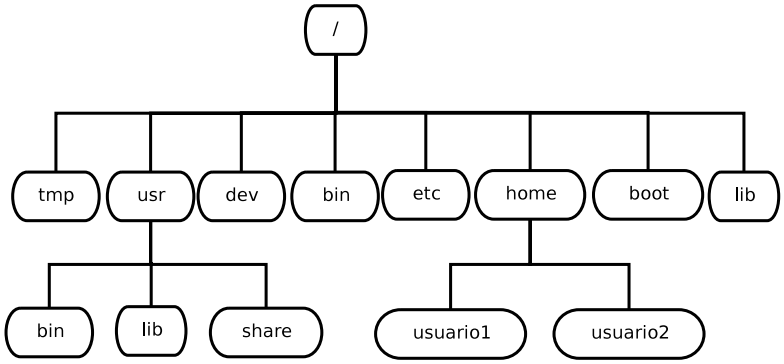

Figura 3: Ejemplo de árbol de directorios.

5.1. 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 la conoce con el nombre de estructura en **árbol invertido**. Un ejemplo típico de árbol de directorios es el mostrado en la figura 3.

La jerarquía de directorios de Unix estandariza una serie de directorios con sus funciones asociadas. De este modo, se sabe cuál es la función de cada directorio de todos los sistemas Unix, proporcionando una estructura limpia y fácil de mantener. Aunque existe un gran número de directorios, sin embargo, sólo los directorios fundamentales se describen a continuación:

/home Directorio que contiene los directorios *home* (o directorios de inicio) de los usuarios. Hay un subdirectorio por cada usuario del sistema. Más adelante se explicará qué es esto.

/etc Directorio que contiene los archivos de configuración del sistema.

/bin Directorio que contiene los ejecutables esenciales del sistema, como *bash* o *ls*.

/sbin Directorio que contiene los ejecutables esenciales para *la administración del sistema*. Sólo el administrador tiene acceso a estas órdenes.

/mnt Directorio en donde comúnmente se montan los sistemas de archivos. Por ejemplo, si se quiere acceder a una memoria USB, se debe *montar* este dispositivo sobre el directorio */mnt*, de manera que todo lo que colgara de */mnt* sería el contenido de la memoria.

/usr Directorio que contiene los archivos de las aplicaciones de los usuarios.

- **/usr/bin** Directorio que contiene los archivos ejecutables de las aplicaciones de los usuarios, por ejemplo, */usr/bin/firefox*.
- **/usr/lib** Directorio que contiene las bibliotecas de las aplicaciones de los usuarios.

/usr/share/man Directorio que contiene las páginas *man*.

/root Directorio *home* del administrador del sistema, conocido como usuario *root* o *superusuario*.

/dev Directorio que contiene los archivos de dispositivo, es decir, los archivos que representan cada uno de los dispositivos del sistema. Hay que recordar que en Unix todo lo que no sea un proceso es un archivo.

/boot Directorio que contiene los archivos necesarios para el arranque del sistema, como la imagen del SO y la configuración del gestor de arranque.

/lib Directorio con las bibliotecas del sistema.

/tmp Directorio en donde se guardan los archivos temporales.

Esta organización de los directorios, en principio confusa, es extremadamente útil. Sólo por poner un ejemplo, supongamos que tenemos un sistema corrupto y que es necesario reinstalarlo, pero sin perder los datos y la configuración anterior. Para conseguirlo bastaría con realizar una copia de respaldo de los directorios */home* y */etc*, de manera que se salvaguarda los archivos de los usuarios y la configuración del sistema. Se procedería con la reinstalación del SO y ya sólo quedaría copiar el contenido de */home* y */etc*.

5.2. Directorios raíz, de conexión y actual

Algunos directorios dentro del sistema de archivos tienen 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 único 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 colgar 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 sistema 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 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]**). Normalmente el directorio de conexión se ubica en */home/usuario*, en donde usuario es el nombre del usuario.

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. Estos directorios son utilizados internamente por el SO para mantener la jerarquía de directorios, pero también son muy necesarios cuando manejamos el intérprete de órdenes.

Un directorio se considera vacío si y sólo 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 poder moverse 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.

5.3. Trayectorias relativas y absolutas

Los archivos se identifican en la estructura de directorios por lo que se conoce como camino, ruta, trayectoria o *path name*. Evidentemente, poder identificar adecuadamente un archivo es fundamental para poder ejecutar un amplísimo número de órdenes. Para simplificarlo se utilizan dos formas de nombrar a los archivos:

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 las rutas absolutas siempre parten 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*.

[1] Una variable de entorno es una variable definida dentro de un intérprete de órdenes. Son importantes porque algunos aspectos fundamentales del sistema se configuran por medio de las variables de entorno, como los lugares en los que se buscan los archivos ejetutables (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*.

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 encontramos en cada momento. Por ejemplo, si nuestro directorio de conexión fuera */home/homer*, el camino para referenciar el archivo *passwd* sería *~/../../etc/passwd*[2], sea cual fuere nuestro directorio actual.

6. 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.

6.1. Listado de archivos

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 (por ejemplo, MS-DOS) podemos cambiar de directorio mediante la orden *cd*. La orden sin argumentos cambia el directorio de trabajo por el directorio de conexión. 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. En algunos sistemas no se encuentra instalado por defecto.

6.2. Visualización de archivos

cat [opciones] [archivos] Visualiza el contenido de uno o más archivos de texto (ASCII) en la pantalla. Si no se le pasa como argumento ningún archivo, leerá de la

[2] Recordemos que el carácter '~' identifica nuestro directorio de conexión.

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. Se utiliza mucho con redirecciones, como se verá en una práctica posterior.

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).

6.3. Copia, borrado y movimiento de archivos

rm [opciones] archivos_o_directorios Elimina archivos o directorios. Para borrar un directorio, debe estar previamente vacío. Si ejecutamos *rm* sobre un directorio, se irá pidiendo confirmación del borrado para cada uno de los archivos y subdirectorios del directorio. Si borramos uno o más archivos, con la opción **[-i]** nos irá pidiendo confirmación. 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]**)[3]. 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.

[3] Es habitual encontrar órdenes que puedan aplicarse recursivamente. En este caso, para que operen recursivamente, se suele utilizar el parámetro "-r".

6.4. Creación y eliminación de directorios

mkdir [opciones] directorios. Crea directorios.

rmdir [opciones] directorios. Borra directorios vacíos.

6.5. 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 que si un archivo o directorio comienza con un punto, el sistema lo tratará como un archivo oculto, y no aparecerá al ejecutar la orden *ls* sin indicarlo explícitamente. Tradicionalmente, los archivo ocultos son archivos carentes de interés habitual, por ejemplo archivos en los que las aplicaciones guardan información interna.

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. Es importante tener en cuenta que los intérpretes de órdenes de Unix consideran distintos los caracteres en mayúsculas de los caracteres en minúscula[4]. Por lo tanto no son equivalentes los archivos: `programa.c` y `Programa.c`.

Una consideración importante es que Unix no presupone ningún significado a las extensiones, a diferencia de otros sistemas operativos. La manera de identificar qué archivo es ejecutable y cuál no lo es, se realiza por una serie de permisos asociados a los archivos, pero no por su extensión.

6.6. Gestión de permisos

Un aspecto clave de todo SO multiusuario es garantizar que sólo los usuarios autorizados puedan acceder a determinados recursos. En el caso de Unix existe un control en el acceso a los archivos por medio de tres permisos asociados a las tres acciones que se pueden hacer con un archivo (leer, escribir y ejecutar) que se aplican a tres tipos de usuarios.

- Permiso de lectura "r". Se permite acceder para leer el contenido del archivo sin modificarlo.
- Permiso de escritura "w". Se permite modificar el contenido del archivo.
- Permiso de ejecución "x". El archivo puede ser ejecutado, por ejemplo un programa compilado o un *shell script*. Cuando este permiso actúa sobre un directorio se le denomina permiso de acceso, pues regula la posibilidad de utilizar ese directorio en operaciones de búsqueda y la posibilidad de ser utilizado como directorio de trabajo.

[4] Los sistemas que distinguen entre mayúsculas y minúsculas suelen denominarse *case sensitive*.

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.

Los archivos, además de los tres permisos descritos, tienen también un *usuario propietario* y un *grupo propietario*. 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).

Los permisos asociados a un archivo se pueden visualizar por medio de la orden “ls -l”. 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 actúa (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
```

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 sólo 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 a *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
```

6.7. Órdenes varias

info [orden] Es una evolución de las páginas *man* en forma de hipertexto. Quizás no está tan extendido su uso como esta última orden, pero en algunas aplicaciones es la mejor fuente de información.

whatis [orden] Busca palabras en una base de datos propia.

apropos [cadena] Busca entradas en las páginas *man* que contengan información sobre la palabra que se está buscando.

file [opciones] archivos Determina el tipo de los archivos dados como argumentos. Por ejemplo, listamos el contenido de un directorio (ls) y tenemos tres archivos de los que queremos conocer el tipo. Con *file arch1 arch2 arch3* el resultado que se muestra por pantalla es:

```
arch1: symbolic link to /root
arch2: ASCII text
arch3: directory
```

scp [opciones] [[usuario@]host:]archivos Copia archivos entre dos máquinas remotas de forma segura. Usa la aplicación *ssh* para la transferencia de datos, y usa la misma autenticación y proporciona la misma seguridad que *ssh*. Una opción interesante es [-r], que hace una copia recursiva del directorio especificado.

df [opciones] [archivo] Muestra información sobre el sistema de archivos que contiene al archivo o, por omisión, sobre todos los sistemas de archivos.

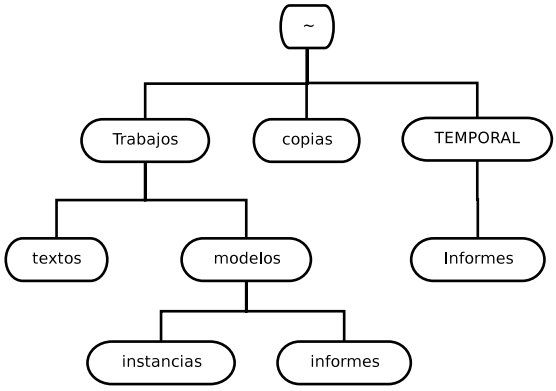

Figura 4: Árbol de directorios del ejercicio.

du [opciones] [archivo] Muestra un resumen del uso de disco para cada archivo, recursivamente para directorios. El parámetro *-h* permite visualizar el tamaño en KB y MB.

cal [opciones] [[mes]año] Muestra un calendario mensual o anual, con el formato especificado en [opciones].

id [opción] [nombre_de_usuario]. Muestra el ID (número de identificación) de usuario y grupo, así como los grupos a los que se pertenece.

Ejercicios

(Nota: Se pueden crear archivos vacíos con la orden *touch*, mirar páginas del manual para más información).

1. Situarse en su directorio de conexión y crear el árbol de directorios de la figura 4 sin cambiar en ningún momento de directorio, es decir, sin usar la orden *cd*.
2. Borrar la estructura de directorios sin moverse del directorio.
3. Situarse en el directorio de conexión y crear el árbol de directorios de la figura 4 cambiando de directorio, es decir, utilizando la orden *cd* para ir entrando en cada subdirectorio.
4. Una vez hecho lo anterior, comprobar mediante una sola orden y sin cambiar de directorio que está correctamente creado.
5. Situarse en el directorio *Informes* y, mediante un camino relativo al directorio de trabajo, situarse en *instancias*.
6. Volver al directorio de conexión utilizando una ruta absoluta.
7. Averiguar cuál es el directorio de conexión y crear dentro del directorio de la práctica un directorio que se llame *tmp*.

8. Crear dos directorios a partir de su directorio *tmp*, llamados: *compartido* y *publico*, con las siguientes características: En *compartido* sólo podrán leer y escribir los componentes de su mismo grupo además de usted. En *publico* sólo podrá escribir usted, pero todo el mundo podrá leer de él.

9. Determinar qué regula cada uno de los permisos de acceso (*r*, *w* y *x)* aplicados a un directorio. En particular, indicar qué permisos mínimos son necesarios para:

   a) Copiar un archivo dentro del directorio.

   b) Eliminar un archivo del directorio.

   c) Ver los nombres de los archivos que hay en el directorio.

   d) Ver el contenido de los archivos que hay en el directorio.

10. Crear un archivo oculto y posteriormente borrarlo.

11. Seleccionar un directorio y averiguar el tamaño que ocupa.

12. Obtener el espacio libre en el disco duro.

13. Visualizar las 10 primeras líneas del archivo */etc/passwd*.

14. Visualizar las 10 últimas líneas del archivo */etc/passwd*.

7. Enlaces

El concepto de enlace en Unix es similar al concepto de enlace directo de Windows. Un enlace es básicamente una referencia a un archivo, y tiene bastantes aplicaciones. En Unix hay dos tipos de enlace: el enlace **fuerte**, físico o duro, y enlace **débil**, simbólico o blando. Las diferencias entre ambos se explican a continuación:

7.1. Enlaces fuertes

Se crean con la orden *ln*:

```
ln nombre_archivo_origen nuevo_nombre_archivo
```

Se basan en la compartición del nodo-i[5] por parte de los diferentes nombres de archivo, es decir, la nueva entrada que se genera en el directorio para mantener el *nuevo_nombre_archivo* tendrá asociado el mismo número de nodo-i que el *nombre_archivo_origen*. Por lo tanto, el enlace fuerte así construido es indistinguible de *nombre_archivo_origen*. Básicamente, con los enlaces fuertes se consigue que un único archivo tenga varios nombres diferentes asociados. Aunque la utilidad práctica de los enlaces fuertes es más bien limitada, son fundamentales para mantener el sistema de archivos, como se verá en Sistemas Operativos Avanzados.

[5] Número que identifica unívocamente a un archivo dentro de un sistema operativo Unix. También denominado inode.

El nodo-i al que se enlaza mantiene control del número de nombres por los que se puede acceder al archivo a través de uno de sus campos, que actúa como contador. Así, siempre que se realiza un nuevo enlace de este tipo se incrementa el campo destinado a mantener el número de enlaces, y cuando se borra un archivo mediante la orden rm se decrementa el campo, por lo que el borrado efectivo del archivo sólo se lleva a cabo cuando este campo toma el valor 0.

Este tipo de enlaces es el que utiliza el SO para mantener la conectividad entre los diferentes directorios del árbol, por lo que no es posible realizar enlaces fuertes sobre directorios.

Otra limitación de los enlaces fuertes es la imposibilidad de realizarse entre diferentes sistemas de archivos, al estar basados en la compartición del nodo-i y ser éste local a cada sistema de archivos.

7.2. Enlaces débiles

Los enlaces débiles serían el equivalente en Unix a los accesos directos de Windows, pero en Unix mantienen una función más esencial. Los enlaces débiles, por ejemplo, juegan un papel vital en el arranque de un sistema Unix. Se crean con la orden *ln* y el modificador -s:

```
ln -s nombre_archivo_origen nuevo_nombre_archivo
```

Esta clase de enlace asocia a *nuevo_nombre_archivo* un nodo-i libre que se utiliza para almacenar el *nombre_archivo_origen*. Unix define un nuevo tipo de archivo para implementar estos enlaces, los archivo de tipo *enlace débil*, que pueden ser reconocidos al realizar un listado largo de los archivos de un directorio porque en el campo en el que se indica el tipo de cada archivo aparece un carácter 'l' (*link*).

Al utilizar un nodo-i distinto, los enlaces débiles permiten realizar enlaces a directorio y a archivos que se encuentren en diferentes sistemas de archivos.

Al no estar basados en la compartición del nodo-i no tienen efecto en el campo que contiene el número de enlaces del archivo origen, pues este campo sólo mantiene el número de enlaces fuertes y al depender, por su implementación, del nombre y lugar del archivo origen, si éste se borra, mueve, o cambia de nombre, ya no será posible acceder a sus datos.

Como puede observarse, en ambos tipos de enlace se mantiene una sola copia del archivo, con independencia del número de enlaces que tenga, por lo que puede deducirse fácilmente que el objetivo de la creación de enlaces es el aprovechamiento del espacio en disco.

Ejercicios

1. Crear en el directorio *informes* un archivo llamado *archivo1* (por ejemplo, con *vi* o con cualquier editor gráfico). Crear, en el directorio *instancias*, un segundo archivo llamado *archivo2*. Crear un enlace fuerte a `archivo1` llamado *enlace1* y un enlace débil a *archivo2* llamado *enlace2*.

2. Emplear la orden y la opción adecuadas para visualizar el número de enlaces fuertes de los archivos *archivo1* y *archivo2*.

3. Modificar el contenido de los archivos *archivo1* y *archivo2* y comprobar cuáles son ahora los contenidos de *enlace1* y *enlace2*.

4. Borrar los archivos *archivo1* y *archivo2*. Verificar qué ha ocurrido. ¿Sigue existiendo el enlace fuerte? ¿Y el débil?

5. Crear un enlace débil llamado *enlace3*, que esté situado en el directorio `copias`, al directorio *Trabajos*. El resultado debe ser que *enlace3* contenga todos los subdirectorios y archivos que contiene *Trabajos*. Comprobarlo. Intentar hacer lo mismo con un enlace fuerte. ¿Se puede?

6. Mediante representación simbólica, conceder todos los derechos al propietario, al grupo y a otros del archivo *archivo3*. Hacer lo mismo para `archivo4`, pero con representación octal.

7. Quitarle todos los derechos al propietario, grupo y a otros del directorio `textos` e intentar acceder a él.

8. ¿Qué orden y cómo la emplearía para cambiar el propietario de un archivo cualquiera?

8. Instalación y desinstalación de programas

Una de las grandes ventajas de utilizar Linux es el sistema de gestión de software que incluyen la mayoría de distribuciones. El software, a diferencia de Windows, no se distribuye en forma de ejecutables autoinstalables, sino en **paquetes**. Un paquete es un archivo comprimido que contiene todos los archivos necesarios para la ejecución de un programa (binarios, configuración, páginas man, etc), así como metainformación sobre el paquete: qué dependencias tiene, qué ficheros lo componen, etc. De esta manera, cuando se instala un paquete toda esa información pasa a una base de datos centralizada, de manera que se tiene un control exacto de todos los archivos que hay en el sistema, facilitando extraordinariamente la gestión y aumentando, con mucho, la consistencia del sistema.

Otra ventaja de la utilización de los paquetes, y del software libre, es que existen repositorios centralizados con todos los paquetes contenidos en una distribución, de manera que puede automatizarse todo el proceso de instalación: localización del paquete, descarga, resolución de dependencias y conflictos y, finalmente, la instalación. Es decir, instalar un programa es tan sencillo como ejecutar una orden, del resto se encarga el sistema de gestión de paquetes.

Un problema que tienen los paquetes en Unix es que no existe un único tipo de paquetes, sino que existen varios, en función de la distribución que se esté utilizando. Se puede sintetizar en que existen dos tipos de paquetes: los *rpms* y los *deb*. El primer tipo es el utilizado por Red Hat, Fedora y derivados, mientras que los deb los utiliza Debian y derivados, como Ubuntu. Los programas utilizados para gestionar cada uno de ellos

varían en función del sistema, mientras que los paquetes rpm se gestionan con la orden *rpm*, los *deb* se gestionan con *dpkg*.

Afortunadamente existen herramientas que abstraen el sistema de paquetes subyacentes, y permiten una gestión muy sencilla de los mismos. En el caso de Debian y Ubuntu se utiliza apt-get. De esta manera, si quisiéramos, por ejemplo, instalar firefox, tan sólo se tendría que ejecutar la siguiente orden como usuario *root*:

```
apt-get install firefox
```

mientras que si quisiéramos eliminarlo sería de la siguiente forma:

```
apt-get remove firefox
```

También es posible consultar qué paquetes hay disponibles en el repositorio. Por ejemplo, si queremos ver qué paquetes se pueden instalar relacionados con firefox, podríamos utilizar la siguiente orden:

```
apt-cache search firefox
```

Ejercicios

1. ¿Por medio de qué archivo se configura el funcionamiento de *apt-get*?
2. Instalar, si es posible, la orden *sl* e indicar para qué sirve.
3. Desinstalar la orden *sl*.

9. Almacenamiento y compresión de ficheros

A continuación se describen algunas órdenes que resultarán útiles para archivar información en un sistema Unix.

tar

```
Sintaxis: tar [opciones] [nombres de ficheros ...]
```

Esta orden permite almacenar varios ficheros en uno solo. Por ejemplo, la siguiente orden:

```
$ tar cvf backup.tar prueba1
```

permite almacenar todos los ficheros del directorio *prueba1* (respetando su estructura) en el fichero *backup.tar*. El primer argumento de la orden (*cvf*) representa alguna de las opciones disponibles. En este caso, la opción *c* indica que se creará un nuevo archivo. La opción *v* permite visualizar los ficheros y directorios que se van almacenando, mientras que *f* indica que el siguiente argumento (*backup.tar*) será el nombre utilizado para el nuevo archivo. El resto de argumentos (en el ejemplo están limitados al directorio *prueba1*) consistirán en los ficheros o directorios que se pretende archivar.

Si se ejecuta la siguiente orden:

```
$ tar xvf backup.tar
```

se extraerá la información procedente del fichero *backup.tar* y se depositará en el directorio actual. Si los ficheros han sido almacenados con nombres de ruta absolutos, la operación de extracción utilizará también esos mismos nombres de ruta.

Si se introduce:

```
$ tar tvf backup.tar
```

puede utilizarse para obtener información sobre el contenido del archivo *backup.tar*, sin necesidad de extraerla. De esta forma se pueden consultar los nombres de los archivos, y en qué orden fueron almacenados.

gzip

A diferencias de algunas utilidades de archivo disponibles en otros sistemas operativos, tar no comprime de forma automática, sino que se utiliza otras órdenes como gzip. *tar* y *gzip* se suelen usar juntos (*tar* para juntar archivos y *gzip* para comprimirlos), sin embargo son programas completamente independientes. Por ejemplo, la orden

```
$ gzip backup.tar
```

comprimirá el fichero *backup.tar*, y lo sustituirá por el nombre de fichero *backup.tar.gz*. Es decir, la versión comprimida del fichero original. Para recuperar el contenido original se utilizará la siguiente orden:

```
$ gzip -d backup.tar.gz
```

o bien se puede utilizar la orden *gunzip*:

```
$ gunzip backup.tar.gz
```

Otro mandato similar a *gzip* es *compress*, aunque hoy en día prácticamente no se utiliza. La diferencia consiste en la extensión que se añade al fichero comprimido (*.Z*). La orden para invertir el resultado de *compress*, es *uncompress*.

Ejercicios

1. Utilizar la orden *tar* para crear un fichero que contenga a todos sus ficheros. Comprimir este fichero mediante la orden *gzip* y copiar el resultado a un nuevo directorio llamado *p01_tmp*. Utilizar las órdenes *tar* y *gzip* para tener en el directorio *p01_tmp* la estructura de ficheros original.