PRACTICA II

Gestión de ficheros

Desde el punto de vista del sistema operativo, los ficheros en UNIX se consideran como ***secuencias de bytes*** sin formato alguno. El sistema distingue solamente los siguientes tipos especiales:

– ficheros ordinarios

– dispositivos físicos

– canales de comunicación (***FIFOs***)

– directorios (ficheros directorio)

– enlaces simbólicos (***soft-links***)

Dentro de los ficheros ordinarios, el tipo de fichero viene dado en realidad por la interpretación que el usuario hace de su contenido. De este modo podemos distinguir ficheros ejecutables (programas compilados), ficheros ASCII (con texto legible), ficheros binarios (datos codificados en formato binario), ...

Los nombres de fichero pueden tener hasta 256 caracteres de longitud, y se admite cualquier caracter ASCII que no tenga ningún significado especial en el shell (`? * ' " / \ $ ...`). Se pueden poner tantos puntos (.) como se quiera en el nombre ya que UNIX no distingue entre nombre y extensión tal como sucede en sistemas como VMS y MSDOS. Por tanto, algunos nombres válidos serían:

```
mis.datos.txt
.login
directorio.marzo-1998.version_2
```

Los ficheros están organizados de manera jerárquica en forma de árbol, de forma que un fichero de tipo directorio contiene una lista de nombres de ficheros de cualquier tipo, incluidos otros ficheros directorio, de forma que esta jerarquía puede tener un número indefinido de niveles.

Convenciones a seguir en los nombres de ficheros

Antes de comenzar debemos aclarar varios términos comunes en UNIX en cuestión de nomenclatura:

– directorio ***raiz*** (***root directory***): es el directorio superior de toda la jerarquía del sistema ficheros. A diferencia de otros sistemas, el directorio raiz es único independientemente de que tengamos varias unidades de disco. El resto de unidades de disco o particiones aparecerán como subárboles del árbol principal.

– ***subdirectorio***: cualquier directorio que esté incluido en otro directorio superior.

– directorio ***padre*** (***parent directory***): el directorio superior en el cual está contenido un subdirectorio.

– directorio ***actual*** de trabajo (***current working directory*** o ***CWD***): es el directorio a partir del cual se buscan los nombres ***relativos*** de fichero usados por un proceso. El directorio actual es privado para cada proceso y se puede cambiar durante la ejecución.

– irectorio ***inicial*** de un usuario (***home directory***): el directorio personal de un usuario, el cual aparece como directorio actual al comienzo de sus sesiones.

Los nombres de fichero están formados por una concatenación de nombres de directorios seperados por la barra (/) y terminados con el nombre del fichero referenciado. Dentro de este formato, distinguimos dos tipos de nombres: ***absolutos*** y ***relativos***. Los nombres absolutos comienzan con la barra en primer lugar, y especifican el camino que se ha de seguir desde el directorio raiz para llegar al fichero en cuestión. Por ejemplo:

`/home/users/sotel99/practica1/ejercicio7/version.1`

Los nombres relativos no comienzan por la barra (/) y especifican el camino que se ha de seguir desde el directorio actual del proceso en ese instante para llegar al fichero indicado. Hay que tener en cuenta que el directorio actual es individual para cada proceso y puede cambiar de un momento a otro. Así que los nombres relativos suelen depender del momento en que se usan. Así, por ejemplo, si el shell del usuario tiene como directorio actual ***/home/users/sotel99***, el fichero anterior se podría referenciar como ***practica1/ejercicio7/version.1*** , mientras que si el directorio actual fuera ***/home/users/sotel99/practica1***, el mismo fichero se referenciaría como ***ejercicio7/version.1*** .

Tanto los nombres de fichero relativos como los absolutos se consideran formados por dos partes:

– el nombre base del fichero (***basename***): es el nombre del fichero quitando todos los nombres de directorios que lo precedan. Es decir, todo lo que siga a la última barra (/) hasta el final del nombre.

– el nombre del camino (***dirname*** o ***pathname***): es el resto del nombre una vez eliminado el nombre base o ***basename***. Por tanto, es la secuencia de directorios que hay que seguir antes de encontrar el nombre base del fichero.

Dentro del nombre de camino o ***pathname***, encontramos dos nombres de directorios que podemos usar:

– el punto '.', referencia al mismo directorio. Por ejemplo: ***./version.1*** indica que el nombre ***version.1*** se ha de buscar en el directorio actual.

– dos puntos seguidos '..' referencian al directorio padre del actual. Por ejemplo ***../version.1*** indica que el nombre ***version.1*** se ha de buscar en el directorio padre del actual.

Ambos nombres se pueden incluir repetidas veces en cualquier punto de un ***pathname***: Por ejemplo, ***/home/users/../users/sotel99/./practica1/../..*** está nombrando al directorio ***/home/users*** en realidad.

Estructura del sistema de ficheros: directorios estándares

El directorio raiz (***root directory***) es el directorio superior de toda la jerarquía y se representa con la barra sola (***/***). Colgando de este directorio aparece toda una serie de directorios que son comunes a todas las instalaciones de UNIX (la estructura del árbol de directorios sigue un estándar para que los usuarios sepan dónde encontrar cualquier cosa en el mismo sitio en todos los sistemas). Esta estructura normalizada es la siguiente:

– ***/bin*** contiene los programas de utilidad más usados del sistema. En las instalaciones modernas este directorio ha desaparecido, y en su lugar solo aparece un enlace simbólico a ***/usr/bin***.

– ***/etc*** contiene los ficheros de configuración del sistema y de casi todos los subsistemas de UNIX. Muchas aplicaciones de usuario también guardan la configuración por defecto en ficheros de este directorio.

– ***/tmp*** sirve para almacenar ficheros temporales. Muchos comandos y aplicaciones de UNIX crean ficheros temporales durante su funcionamiento de cuya existencia el usuario no tiene conciencia. Dichos ficheros son borrados automáticamente por los programas que los crean. El usuario también puede usar este directorio para almacenar ficheros de forma temporal. Sin embargo, el sistema considera que el contenido de este directorio puede ser borrado de forma automática, de forma que los ficheros que residen en este directorio desaparecerán tarde o temprano. Normalmente se borra al reinicializar el sistema y periódicamente cada varios días.

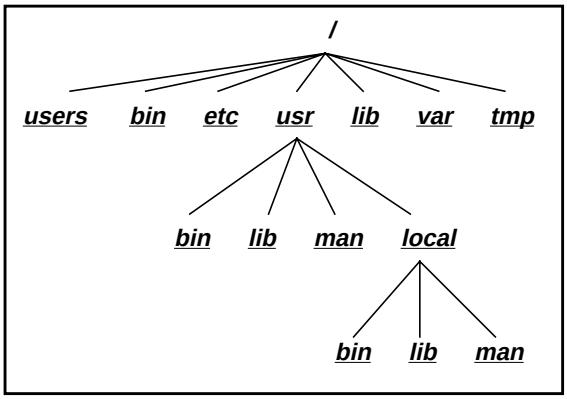
– ***/lib*** contiene las ***librerías*** o bibliotecas de funciones a las que se llama en los programas escritos en lenguajes de alto nivel y algunos ficheros de datos del sistema operativo. En muchos sistemas modernos, este directorio ha desaparecido y en su lugar aparece un enlace simbólico al directorio ***/usr/lib***.

– ***/dev*** contiene los ficheros de tipo dispositivo correspondientes a todos los periféricos del sistema: discos, cintas, terminales, cintas, ...

– ***/usr*** es un directorio que contiene normalmente todas las aplicaciones del sistema operativo. A su vez, alberga los directorios ***/usr/bin***, ***/usr/lib*** y ***/usr/man***, que contienen respectivamente los ejecutables, los datos y los manuales de las aplicaciones instaladas.

– los directorios personales de los usuarios residen normalmente bajo un subdirectorio común. Según los sistemas, este directorio común suele llamarse ***/home***, ***/usr/home***, ***/users***, ***/usr/users*** o incluso los usuarios pueden estar repartidos en varios directorios comunes

– ***/usr/local*** se usa normalmente para instalar los programas y aplicaciones que no son estándar de Unix (editor EMACS, procesador de textos LaTeX, etc...).



Atributos de un fichero

Para listar el contenido de los directorios se usa el comando ***ls***. Por defecto, ls no muestra los nombres de ficheros que comienzan por un punto (ficheros ocultos o ***hidden files***). a menos que se use la opción ***-a***. La opción ***-l*** produce un listado de cada fichero detallando toda la información adicional disponible:

– tipo de fichero (d: directorio) y permisos de acceso (rwxr-xr--): ***permission modes***.

– números de nombres que referencian a este fichero en el sistema: ***references count***.

– nombre del usuario propietario del fichero: ***owner userid***.

– nombre del grupo al que pertenece el propietario del fichero: ***owner group***.

– tamaño del fichero en bytes.

– fecha y hora en que el fichero fue modificado por última vez.

– nombre del fichero o directorio.

El primer carácter de la línea indica el tipo del fichero:

– d: ficheros de tipo directorio.

– b: dispositivos orientados a E/S de bloques.

– c: dispositivos orientados a E/S de caracteres.

– l: enlace simbólico a otro fichero: ***soft link***.

– s: socket asociado al sistema de ficheros: ***named socket***.

– p: FIFO asociada al sistema de ficheros: ***named pipe***.

– '-': fichero ordinario.

Los caracteres que forman los permisos codifican los derechos de acceso. Se agrupan de tres en tres. El primer grupo son los derechos del propietario. El segundo los derechos del grupo. El tercero son los derechos del resto de usuarios del sistema que no se incluyan en los casos anteriores. Las tres letras

indican:

– r: derecho de lectura

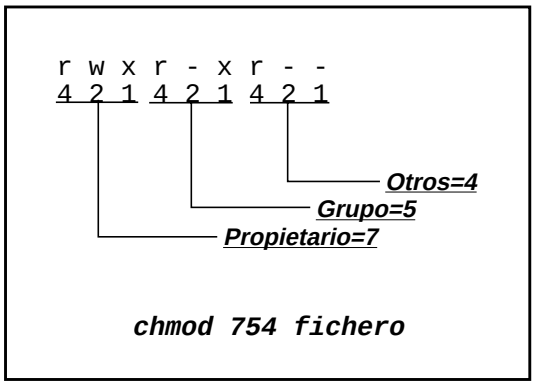
– w: derecho de escritura o modificación

– x: derecho de ejecución

Comandos básicos para gestionar ficheros

Los siguientes comandos son básicos para moverse por el sistema de ficheros y hacer consultas y modificaciones. Todos los nombres de fichero usados como argumentos pueden ser relativos o absolutos indistintamente.

mkdir nombre_dir	crea un directorio con el nombre indicado.
rmdir nombre_dir	borra el directorio especificado si está vacío.
pwd	devuelve el nombre del directorio actual en el shell.
cd nombre_dir	cambia el directorio actual del shell al nombre indicado.
ls	lista el contenido de un directorio (por defecto lista el actual).
cp fichero nuevo_fichero	crea una copia del fichero indicado con el nuevo nombre.
cp fichero [...] destino	copia los nombres indicados al directorio de destino.
mv fichero nuevo_fichero	cambia el nombre del fichero al nuevo nombre especificado.
mv fichero [...] destino	mueve los ficheros indicados al directorio de destino.
rm fichero [...]	borra los ficheros indicados
lp fichero [...] (o lpr)	manda a la impresora del sistema los ficheros especificados.
find . -name file -print	busca el fichero ***file*** a partir del subdirectorio actual.
chmod permisos fichero	cambia las protecciones del fichero.



Enlaces (Links)

Los enlaces permiten acceder a un fichero desde varios puntos en el sistema de ficheros evitando tener que copiar el fichero a todos esos puntos. Por ejemplo, si queremos acceder al fichero /etc/passwd varias veces y queremos evitar tener que movernos de nuestro /users/sotel/sotel99 o tener que escribir el path completo, podemos hacer lo siguiente desde nuestro directorio actual:

ln -s /etc/passwd passwdlink

Si ahora hacemos ***ls -l passwdlink*** veremos la siguiente información:

```
lrwxrwxrwx 1 sotel99 users 9 Apr 15:45 passwdlink -> /etc/passwd
```

Dispositivos

Como ya se ha mencionado con anterioridad, todos los dispositivos se encuentran como ficheros dentro del directorio ***/dev.*** De esta forma un usuario puede acceder a estos dispositivos como ficheros normales. Los ficheros en este directorio llevan asociados dos números: *major device number* y *minor device number.* Adicionalmente, estos dispositivos pueden ser de tipo carácter o de tipo bloque, lo cual se indica mediante una **c** o una **b**, respectivamente, en la primera columna de los atributos del fichero. En la siguiente tabla vemos algunos de los dispositivos más frecuentes:

Dispositivo	Descripción
/dev/console	Consola del sistema
/dev/mouse	Ratón
/dev/hda	Primer disco duro
/dev/hda1	Primera partición del primer disco
/dev/hda2	Segunda partición del primer disco
/dev/hdb	Segundo disco duro
/dev/hdb1	Primera partición del segundo disco
/dev/fd0	Primera unidad de floppy
/dev/sda	Primer disco SCSI
/dev/sdb	Segundo disco SCSI
/dev/lp0	Primer puerto paralelo (impresora)
/dev/null	Dispositivo vacio
/dev/ttyN	Consola virtual
/dev/ptyN	Pseudoterminal para entrar desde la red
/dev/ttySN	Puertos serie

Tabla 1: Lista de los dispositivos más importantes

Los números mayor y menor del dispositivo se pueden ver en la quinta y sexta columna respectivamente al hacer ***ls -l***. Todos los dispositivos del mismo tipo (hda, tty,...) tienen el mismo número mayor (y también el mismo controlador -driver- en el kernel), pero se distinguen por su número menor.

Intérprete de comandos: Shell

Hasta el momento el usuario ha estado usando un programa, quizás sin saberlo, que se denomina intérprete de comandos, o shell. El shell se puede ver como un lenguaje de programación que ejecuta los comandos que lee desde el terminal o bien desde un fichero. Para situar a los usuarios de DOS, el command.com no es mas que el intérprete de comandos de este sistema operativo, aunque proporciona mucha menos versatilidad que los shell que tratamos a continuación.

Existen varios intérpretes de comandos en Unix, e incluso el usuario podría programarse el suyo propio y ejecutarlo al abrir una sesión. Los dos shells más comunes son el Bourne Shell (/bin/sh) creado por Steve Bourne, y el C Shell (/bin/csh) escrito en la universidad de Berkeley por Bill Joy e inspirado en el anterior. El Bourne Shell fue el primero y no es demasiado confortable para el trabajo interactivo, aunque es más fácil de usar para programar. Por su parte, el C Shell ofrece algunas comodidades, como los *alias*, que evitan teclear largos comandos, el *history*, que evita teclear comandos ya tecleados con anterioridad, etc.

Sin embargo, no son el **sh** o el **csh** los intérpretes de comandos más usados en la actualidad. En su lugar, han ganado aceptación las versiones evolucionadas de cada una de las variantes: el **bash** (Bourne Again shell) y el **tcsh**. En esta práctica se explican primero algunas características comunes del trabajo interactivo en los dos intérpretes de comandos, dejando para la práctica siguiente las diferencias entre ellos y sus sintaxis de programación.

Metacaracteres

Como se vio anteriormente, un comando tiene un nombre, admite unas opciones que modifican su comportamiento y trabaja sobre un conjunto de ficheros que se le indican. Pero el shell, aparte de permitir introducir comandos, también realiza algunas funciones. El uso de los llamados *metacaracteres* es de gran ayuda a la hora de designar ficheros:

– El asterisco (*) puede sustituir a cualquier número de caracteres. El shell busca entre todos los nombres de fichero del directorio indicado (por defecto el actual) cuales pueden hacerse coincidir con el patrón indicado. **Ejemplo:** t* se sustituye por trabajo, ta, test, texto.txt, tb, t1, test.old

– El interrogante (?) se sustituye por un único carácter. **Ejemplo:** t? se sustituye por ta, tb, t1.

– Los corchetes [] se sustituyen por cualquiera de los caracteres escritos entre ellos. **Ejemplo:** [A-Z]* se sustituye por cualquier fichero que comience por letra mayúscula seguida de cualquier número de caracteres.

– La tilde (~) seguida del nombre de usuario se sustituye por el “home directory” de ese usuario.

Los metacaracteres son interpretados por el shell y proporcionan al comando la lista de ficheros que corresponde a la interpretación. Así el comando ***cat t****** haría que el shell ejecutase el comando ***cat*** al cual proporcionaría la lista de ficheros *trabajo, ta, test, texto.txt, tb, t1, test.old*. En otras palabras, el comando ***cat t****** sería equivalente a que el usuario escribiera “cat *trabajo ta test texto.txt t t1 test.old*”.

Cuando se introduce un comando, el shell busca un fichero ejecutable con ese mismo nombre y si lo encuentra crea un proceso y le pasa las opciones y ficheros que el usuario le ha introducido. Cualquier programa que ejecute el sistema operativo UNIX lo hará creando un nuevo proceso.

Entrada y salida estándar. Redireccionamiento

Otra característica interesante que ofrece el intérprete de comandos es la redirección de la entrada y salida estándar de un programa. Todos los programas pueden obtener datos de un que fichero que se denominan entrada estándar. Desde el punto de vista del sistema UNIX el terminal es un caso particular de fichero y su nombre es /dev/ttynnn (por ejemplo /dev/ttyp10). Lo mismo ocurre con la salida estándar. Si no se indica lo contrario, el shell conectará el terminal en el que se trabaja con la

entrada y salida estándar del programa que debe ejecutar.

Si queremos enviar un mensaje usando la utilidad mail, que no es más que un programa cuyo nombre absoluto es /bin/mail, deberemos escribir ***mail usuario***. A partir de ese momento mail leerá de su entrada estándar el mensaje que queremos enviar. Pero, ¿y si queremos enviar un fichero?. El shell permite redireccionar la entrada estándar de un programa con el símbolo menor (<). Por tanto escribiremos ***mail usuario < fichero***. ¿Cómo sabrá el programa mail que el mensaje se ha acabado?, ¿Hace falta que el fichero acabe por una línea en la que aparezca un punto en la primera columna?. No, recordemos que otra manera de acabar un mensaje era pulsando ^D que es el carácter que designa el fin de fichero. Si enviamos un mail escribiéndolo desde el terminal y pulsamos ^D es como si el fichero que escribíamos en el terminal hubiera acabado.

Los caracteres que permiten redireccionar entrada y salida estándar son (<) para la entrada y (>)para la salida, borrando el contenido original del fichero que se indica. Además (>>) permite redireccionar la salida añadiéndola al final del fichero especificado. También existe un carácter que se denomina **pipe** (|) que conecta la entrada y salida estándar de dos programas. El siguiente ejemplo permite saber cuantos ficheros hay en el directorio actual puesto que ***ls*** escribe una línea por fichero y ***wc*** cuenta las líneas de su entrada estándar:

```
ls -a | wc
```

El shell también permite agrupar la ejecución de comandos y redireccionar su salida estándar globalmente si se incluyen entre paréntesis. El carácter (;) indica ejecución secuencial entre los dos comandos que separa (primero uno, y cuando termine éste puede empezar el otro)

```
lac20_$ (date; who) > hoy.11.25
lac20_$ cat hoy.11.25
Mon  Apr  19 11:25:47 WET DST 1999
sotel21   ttyp10    Apr 19 10:26
sotel1    ttyp4     Apr 19 09:20
sotel4    ttyp3     Apr 19 09:21
sotel16   ttyp2     Apr 19 09:40
lac_20$
```

Filtros

En vista de la posibilidad de redireccionar la entrada o salida estándar de los diferentes comandos, en UNIX aparecen una serie de comandos especiales denominados **filtros**. La misión de un filtro es leer su entrada estándar, procesarla y escribir los resultados por su salida estándar. A continuación vemos los filtros mas comunes:

more muestra una página de su entrada estándar y espera que el usuario le indique seguir mostrando cada una de las siguientes páginas. También ofrece ayudas interactivas y tras el prompt "---More---(nn%)---" es posible pedir ayuda pulsando la tecla **h** y ver que otras opciones permite.

less Similar a ***more*** pero más potente (*less* es el opuesto de *more*). Permite volver hacia atrás y no necesita leer el fichero entero para empezar a mostrar la primera pantalla (como le ocurre a *more*) y como resultado es más rápido en responder.

tail Escribe las últimas 10 líneas de la entrada estándar en su salida estándar. Con ***tail -nn*** escribe las últimas ***nn*** líneas.

diff Para comparar ficheros. Recientemente la información sobre este comando no se encuentra en las páginas *man*, sino en la utilidad ***info*** de **GNU** (***info diff***)[1].

sort	ordena la entrada estándar y la escribe por la salida estándar. Por defecto ordena alfabéticamente en orden ascendente pero también puede ordenar descendentemente, numérico y por alguna columna que no sea la primera (ver ***man sort***).
wc	cuenta las líneas, palabras y caracteres de la entrada estándar.
tee nombre_fichero	envía su entrada estándar hacia su salida estándar pero además escribe también en el fichero que se le indique. Si se quiere copiar por pantalla se puede usar el nombre /dev/tty que es un alias del terminal de cada usuario.
pr	se usa para formatear el contenido de un fichero. Por defecto separa el fichero en páginas, cada una con su número, fecha y hora de impresión y el nombre del fichero. También permite formatear un fichero en varias columnas.
grep	busca la cadena de caracteres suministrada en los ficheros indicados. Si encuentra la cadena muestra el nombre del fichero y la línea donde aparece.
cut	Presenta a la salida estándar secciones de las líneas que alimentan la entrada estándar. Por ejemplo: ***cut -f2,3 -d:*** presentará a la salida estándar los campos (***f****ield)* 2 y 3, suponiendo que estos campos están marcados con el ***d****elimitador* ":". Con la opción ***-c3-8*** estaríamos indicando que queremos cortar las ***c****olumnas* 3 a 8.
paste	Comando opuesto a ***cut***. Es decir, pega líneas de distintos ficheros. Por ejemplo: ***paste -d: file1 file2*** presentaría a su salida estándar la primera fila de ***file1*** seguida de ":" y la primera fila de ***file2***; en la siguiente línea, la segunda fila de ***file1***, ":", y la segunda fila de ***file2***, etc, etc.

Ejecución secuencial y concurrente

Los comandos que tardan bastante tiempo en ejecutarse pueden ejecutarse en "paralelo" o "background" usando el carácter (&) al final de la línea. Este carácter (&) indica al shell que no espere a que el comando termine para volver a presentar el prompt. Si no queremos que se mezclen las salidas de varios comandos se pueden enviar con las entradas y salidas redireccionadas hacia ficheros. En el siguiente ejemplo vemos como, tras esperar un segundo en lanzar el comando ***date*** (debido al comando ***sleep 1***), las salidas de los comandos ***who*** y ***date*** se mezclan.

```
lac20_$ ( sleep 1 ; date ) &
[1] 635
lac20_$ who
sotel13 ttyp1    Apr  3 11:54 (:0.0)
sotel21 ttyp2    Apr  3 11:54 (:0.0)
Sat Apr  3 12:27:29 CEST 1999
sotel21 ttyp3    Apr  3 11:54 (:0.0)
sotel21 ttyp4    Apr  3 12:08 (lac21)
lac20_$
```

1. Si ejecutas ***info diff*** y quieres aprender a navegar por esta utilidad, pulsa "?". Básicamente, con el TAB puedes posicionar el cursor en los submenus, con ENTER entras en el submenu sobre el que está el cursor (también pulsando "m"), con "n" te mueves al siguiente submenu y con "p" al previo. Con "u" te vuelves al submenu "padre" (up) y con ^X-K-ENTER sales de la pantalla de ayuda. Pulsando "q" sales de la utilidad ***info***.

En los shell **csh**, **tcsh** y **bash**, se puede usar el comando ***jobs*** para listar los comandos que están actualmente en background. La ejecución de un comando si precederlo del carácter (&) se denomina ejecución en *foreground* o primer plano. Cuando se esta ejecutando un comando en primer plano, podemos pararlo (suspenderlo) con ^Z y el shell vuelve a presentar el prompt. Si queremos que ese comando parado siga corriendo en background o foreground debemos usar el comando ***bg*** o ***fg*** respectivamente. El comando ***bg*** afecta al último comando suspendido con ^Z mientras que ***fg*** afecta al último comando suspendido o al último que se mandó a ejecutar en modo background. Si quieres referirte a otro de los comandos suspendidos o en background utiliza ***jobs*** para conocer su número (n) y luego ejecuta ***bg %n*** o ***fg %n***. Recuerda que ^Z no manda el trabajo a background, sino que sólo lo para (suspende). A continuación tienes un ejemplo del uso de estos comandos:

```
lac-20_$ ( sleep 100 ; who ) & sleep 102 &
[1] 2649
[2] 2650
lac-20_$ jobs
[1]  + Running                        ( sleep 100; who )
[2]  - Running                        sleep 102
lac-20_$ fg %2
sleep 102
^Z
Suspended
lac-20_$ bg
[2]    sleep 102 &
lac-20_$ jobs
[1]  + Running                        ( sleep 100; who )
[2]    Done                           sleep 102
lac-20_$ fg
( sleep 100; who )
^Z
Suspended
lac-20_$ fg
( sleep 100; who )
sotel12  ttyp1    Apr  7 14:28 (:0.0)
sotel12  ttyp2    Apr  7 14:28 (:0.0)
sotel12  ttyp3    Apr 10 11:43 (:0.0)
sotel2   ttyp5    Apr 11 17:31 (lac-21)
lac-20_$
```

Inhibición de los metacaracteres

Si se desea que el shell no interprete algún carácter, éste se puede preceder de la barra invertida (\) o bien, si se trata de un grupo de caracteres, encerrarlos entre comillas dobles (") o simples('). Por ejemplo el comando ***echo*** hace un echo de los parámetros que se le pasan. Si se escribe algún metacaracter el shell lo interpretará y por tanto le pasará a ***echo*** el resultado de la interpretación.

Sustitución de comandos

Otro carácter con misión específica en el shell es la comilla invertida (` ) , veamos su uso con un ejemplo. siguiendo con el programa mail, si se desea enviar un mensaje a una lista de usuarios existen dos opciones: teclear mail seguido de todos los usuarios o bien ***mail `cat listacorreo`***, siempre que el fichero listacorreo contenga la lista de todos los usuarios. El shell al encontrarse comillas invertidas ejecutará el comando indicado (***cat***) y le pasará a mail el resultado de su ejecución como la lista de parámetros.

Expresiones regulares

```
lac20_$ echo you & me
[1] 639
you
me: Command not found.
[1]  + Done                          echo you
lac20_$ echo you \& me
you & me
lac20_$ echo "you & me"
you & me
lac20_$
```

```
lac20_$ cat listacorreo
sotel19
sotel20
sotel21
sotel22
sotel23
lac20_$ mail `cat listacorreo` < mimensaje
```

En muchos de los filtros de ficheros y editores (more, less, grep, vi, awk, etc) podemos usar una sintaxis que permite especificar cadenas. Esta sintaxis se llama ***expresión regular***. Donde quiera que puedas utilizar la operación ***/cadena*** para buscar una ***cadena***, puedes usar expresiones regulares en lugar de una cadena simple. De hecho, una cadena de caracteres no es más que la forma menos compleja de una expresión regular.

Una expresión regular consta de operadores que describen caracteres simples a buscar. Por ejemplo, la cadena **abc** consta de tres caracteres. Por cada carácter, podrías haber usado en su lugar una expresión compleja que describiera el carácter que estás buscando en esa posición de la cadena. El carácter (.) es uno de los operadores de expresión regular y representa "cualquier carácter simple". Por ejemplo, podrías utilizar la cadena **a.c** para referirte a cualquier cadena de tres caracteres que comience por **a** y acabe por **c** (por ejemplo, **adc**, **a#c**, **aSc**,...).

Otros comandos se comentan a continuación (más información en ***man egrep***):

– [ ] : rango de caracteres válidos. Ejemplo: [aAbB]cd indica una cadena de tres caracteres que empieza con a,A,b o B y acaba con cd; [a-z] es cualquier letra en minúscula; [0-9] cualquier dígito, etc. Si ponemos un (^) indicamos exclusión, por ejemplo [^aAbB] representa a cualquier carácter que no sea ninguno de los que siguen al carácter ^.

– ^ : comienzo de cadena.

– $: final de cadena.

– ( ) : delimitan un grupo o expresión regular.

– | : equivalente a un OR lógico.

– * : 0 o más veces el grupo anterior

– + : 1 o más veces el grupo anterior.

– ? : 0 o 1 vez el grupo anterior

– \ : escape. Permite especificar cadenas que contenga ^, $, *, etc. Para ello hay que antecederlos del carácter escape (\).

Ejemplo:

– ^#.*\.[a-z]+ca : Cadena que tiene un # al comienzo de línea, luego cualquier número de caracteres (indicado con .*), luego el carácter ".", a continuación un carácter en minúscula una

o más veces seguidos de una c y una a.

Gestión de procesos

Como ya se ha dicho anteriormente, el shell crea un nuevo proceso cada vez que se ejecuta un programa. Para ver los procesos que un usuario tiene se usa el comando ***ps***. Al ejecutar ***ps*** se ve que hay varios procesos ejecutándose simultáneamente: el ***shell*** y el propio ***ps***.

```
lac-19_$ ps
  PID TTY STAT TIME COMMAND
 2664  p5 S    0:00 login lac-20
 2665  p5 S    0:00 -csh
2721  p5 R    0:00 ps
lac-19_$ ps u
USER       PID %CPU %MEM  SIZE   RSS TTY STAT START   TIME COMMAND
asenjo    2664  0.1  3.2  1512  1004  p5 S    20:44   0:00 login lac-20
asenjo    2665  0.2  3.1  1716   984  p5 S    20:44   0:00 -csh
asenjo    2722  0.0  1.8   936   584  p5 R    20:47   0:00 ps u
lac-19_$ ps l
 FLAGS   UID   PID  PPID PRI  NI   SIZE   RSS WCHAN       STA TTY TIME COMMAND
100100   260  2664  2663   0   0   1512  1004  117319     S   p5  0:00 login
     0   260  2665  2664   0   0   1716   984  10a0f0     S   p5  0:00 -csh
100000   260  2733  2665  18   0    936   584  0          R   p5  0:00 ps l
lac-19_$
```

La opción ***u*** produce algo más de información que resulta interesante comentar:

– La columna USER indica quién es el usuario propietario del proceso.

– PID muestra el número de proceso. Este número es el que aparece al lanzar un comando a ejecutarse en background. Los PID se van asignando por orden cronológico, por tanto procesos lanzados posteriormente tienen identificadores mayores.

– %CPU y %MEM indican los porcentajes de CPU y memoria consumidos por los correspondientes procesos.

– SIZE indica el tamaño en Kbytes de memoria virtual ocupada por el código mas los datos y la pila del proceso.

– RSS indica la memoria física (en Kbytes) ocupada por el programa.

– El terminal asociado al proceso se muestra bajo la columna TTY (p10 es el terminal ttyp10).

– STAT indica el estado del proceso: **R** de ejecutable, **S** de sleeping, **T** de stopped, **Z** de zombie

– Bajo el encabezamiento START aparece la hora, minutos y segundos en que se lanzó la ejecución del proceso.

– TIME muestra el tiempo de CPU consumido por el proceso en minutos y segundos.

– El nombre del comando con sus argumentos aparece listado en la columna COMMAND.

Con el comando ***l*** aún conseguimos más información, destacando:

– El identificador del “proceso padre” aparece en la columna PPID. Los procesos están relacionados según una estructura jerárquica. Vemos que el proceso padre del proceso que ejecuta ***ps l*** es el proceso que ejecuta ***csh*** (el intérprete de comandos o shell).

– Bajo la columna NI encontramos la prioridad asignada con el comando ***nice***. Un valor positivvo indica menor prioridad.

Las opciones anteriores tan sólo muestran los procesos relacionados con el usuario que ejecuta el comando ***ps***, para ver todos los procesos que se ejecutan en la máquina se debe usar la opción ***a***.

El comando ***kill -TERM numero_proc*** se usa para abortar la ejecución de un comando que

tarda demasiado o que no se desea que finalice su ejecución normal. En el siguiente ejemplo se ve cómo al ejecutar una agrupación de comandos (van entre paréntesis) y enviarla a ejecutar en background, se crea un nuevo proceso con el shell que lee cada uno de los comandos, y para uno de ellos crea un proceso. En el listado se ve como, de los dos shells, el shell con identificador mayor ha creado un proceso sleep. El proceso ***ls*** se creará cuando sleep acabe, por eso no aparece listado. Al matar este shell aparece un mensaje confirmándolo y si ejecutamos ***ps*** de nuevo vemos como ***sleep*** aún permanece ejecutándose.

```
lac-19_$ ( sleep 300 ; ls ) &
[1] 2764
lac-19_$ ps
  PID TTY STAT TIME COMMAND
 2706  p5 S    0:00 -csh
 2764  p5 S    0:00 -csh
 2765  p5 S    0:00 sleep 300
 2766  p5 R    0:00 ps
lac-19_$ kill -TERM 2764
[1]    Terminated                    ( sleep 300; ls )
lac-19_$ ps
  PID TTY STAT TIME COMMAND
 2706  p5 S    0:00 -csh
 2765  p5 S    0:00 sleep 300
 2767  p5 R    0:00 ps
lac-19_$
```

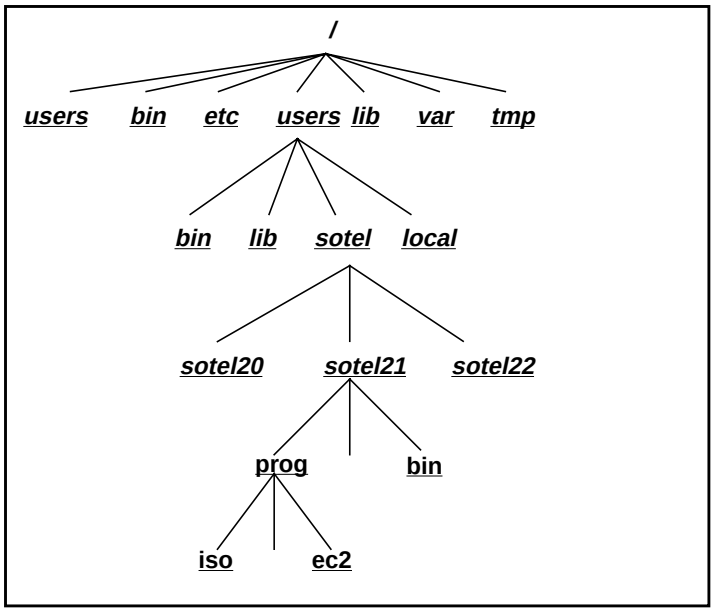
Ejercicios

Responde a cada cuestión especificando que comando has usado para su realización

1. ¿Cuántos subdirectorios hay en el directorio raiz?
2. ¿Cómo se llama el directorio en el cual usted entra en el sistema?
3. Dibuje la estructura jerárquica en la que se vea el emplazamiento del fichero **xterm** que está incluido en el directorio **/usr/X11/bin.**
4. Escriba un comando que le lleve a su “home directory” esté donde esté.
5. ¿Qué ficheros del directorio raiz son ejecutables por cualquier usuario?
6. Cree el directorio “pruebas” en su “home directory”. Copie en él todos los ficheros del directorio /tmp. Compruebe cuantos ficheros ha conseguido copiar. Posiciónese en el directorio pruebas y borre todos los ficheros que comiencen por letra mayúscula. Compruebe que efectivamente se han borrado. Borre el directorio pruebas.
7. Algún usuario de su grupo ha estado modificando uno de sus ficheros. Escriba un comando que le deniegue acceso de modificación, aunque pueda seguir leyéndolo.
8. Lea el man del comando ***chmod*** y encuentre otra forma de cambiar los permisos de un fichero sin utilizar los códigos octales mencionados en la página PracticaII-4.
9. ¿Cuántos ficheros hay en el directorio /dev que comiencen por los caracteres tty?
10. ¿Qué hace el comando ***cat > fichero***?
11. Ejecuta el comando ***stat fichero*** y estudia con el man que hace ese comando.
12. Mira la función de los comando ***uniq*** y ***comm***. Crea dos directorios en tu “home directory” y crea o copia ficheros en ellos. Ahora crea una línea de comando que usando uniq o comm te informen sobre que nombres de ficheros están repetidos y cuales no en los dos directorios que has creado.
13. Escriba un comando que averigüe cuántos usuarios hay en el sistema y escriba una lista

ordenada con los usuarios en el fichero usuarios.hoy.

14. ¿Qué shell está ejecutando? Ejecute el comando sh y a continuación el comando ***ps***. ¿Cuántos procesos hay? ¿De qué dos maneras puede matarse el proceso sh?
15. ¿Qué comandos se deberían teclear para obtener la lista de usuarios a dos columnas? ¿Cómo se pueden eliminar las líneas en blanco que sobran?
16. ¿Cuál es el nombre absoluto del fichero apropos? ¿Qué comando(s) usó para averiguarlo?
17. ¿Cuál es la diferencia entre nombres de fichero absolutos y relativos?
18. Explique que sucedería si escribiera el comando ***finger | sort -r | grep `whoami` | cut -c1-8.***
19. En el siguiente árbol de directorios



– ¿Qué comando se debe usar para que ***sotel21*** se cambie al directorio de ***sotel22***?

– ¿Qué comando debe escribir ***sotel21*** para volver a su "home directory"?

– Escriba el comando que cambia del directorio ***sotel21*** al directorio ***iso***.

– Escriba el comando que cambia del directorio ***ec2*** al directorio ***bin*** de ***sotel21***

– ¿Qué sucede si se escribe el comando ***cd iso*** cuando el directorio de trabajo es ***sotel21***?

– ***sotel20*** quiere crear un directorio denominado ejemplos en el directorio de ***sotel21***. ¿Es posible hacer ésto?. Si es posible ¿Cómo podría evitarlo ***sotel21***?. Si no es posible explique el motivo.

20. Escriba una línea de comando que liste por orden de identificador de proceso todos los procesos de la máquina que ejecutan el comando ***csh***.
21. Escriba una línea de comando que muestre todos los procesos que se ejecutan por orden de mayor a menor tiempo de ejecución.
22. Ejecuta el comando ***ypcat passwd > passwfile***. Con este fichero haremos algunas practicas de expresiones regulares, ***cut*** y ***paste***:

– Escribe una línea de comando que almacene en un fichero el UID de todos los usuarios sotel.

– Haz lo mismo, guardando ahora otro fichero el "home directory" de esos mismos usuarios.

– Pega los dos ficheros de forma que generes un único fichero con el formato UID:"home directory".

– ¿Podrías haber generado este fichero en un único paso?

– Usando ***egrep***, comprueba si en el ***fichero*** passwdfile existe alguna cadena que tenga al menos cuatro letras mayúsculas. Si no existe esa cadena en tu fichero, insertala y comprueba que la puedes encontrar mediante una expresión regular.