Sistemas Operativos 1
Introducción a los comandos Linux

Edwin Salvador

28 de enero de 2016

Índice

1. Introducción

El Sistema Operativo UNIX se inició en una DEC PDP-7 desechada, en los Laboratorios Bell durante 1969.En 1973, Ritchie y Thompson reescribieron el núcleo del UNIX en C, rompiendo así con la idea de que los sistemas operativos deben estar escritos en lenguaje ensamblador.

Hacia 1974 fue introducido en las universidades "con fines educacionales" y al cabo de pocos años estaba ya disponible para uso comercial. El UNIX es un sistema portable (se ejecuta en una extensa variedad de arquitecturas), flexible, potente, con entorno programable, multiusuario y multitarea.

A partir de la primera versión del UNIX (creada por AT&T) han ido apareciendo algunas variantes. Algunas de las más importantes son:

- **AT&T** Ken Thompson, un programador de AT&T Bell Laboratories, con un conjunto de expertos desarrollaron un sistema operativo flexible y totalmente compatible con las distintas necesidades de los programadores. Según parece, el nombre de UNIX proviene del sistema operativo multiusuario MULTICS, entendiendo UNIX como la versión monousuario de MULTICS.
- **BSD** Berkeley Software Distribution desarrolló la primera versión de UNIX basándose en la versión 7 de AT&T (1978). BSD UNIX incorpora una serie mejoras desarrollas por la universidad de Berkeley para hacer el sistema más accesible para los usuarios. Desde entonces, BSD se ha convertido en el estándar académico de UNIX. Existe una versión intel de este sistema llamada FreeBSD.
- **SCO UNIX** System Laboratories (filial de AT&T) desarrolló una variante de UNIX (la System V). Esta empresa fue comprada por Novell y al producto le llamaron SCO (Santa Cruz Operation) UNIX.
- **XENIX** Este sistema es la aportación de Microsoft al mundo UNIX en los PCs. Después del desarrollo de XENIX, Microsoft se fusionó con AT&T dando lugar al sistema operativo System V/386 v3.2.
- **SunOS** Sun Microsystems ha contribuido enormemente a introducir UNIX en el mercado informático mediante la promoción de SunOS y de sus estaciones de trabajo. SunOS es el resultado de la colaboración entre Sun y AT&T (System V v4.0)
- **AIX** AIX es un producto de IBM. No es un sistema operativo tan conocido pese tener un muy buen rendimiento y no causar problemas (de hecho es uno de los más utilizados en el sector de los grandes servidores).

2. Sistema de ayuda

UNIX dispone de forma estándar de un completo sistema de ayuda. Podemos obtener ayuda sobre cualquier comando o sobre cualquier aspecto del sistema mediante el comando man, cuyo formato es:

```
man [seccion] materia
o
man -k clave
```

donde:

- **materia** es el elemento (comando, llamada al sistema, etc.) sobre el cual se solicita información
- **sección** es el capítulo del manual en el que se busca la información sobre la materia en cuestión. Este argumento es opcional, y en caso de no especificarse, se busca información acerca de la materia seleccionada en todos los capítulos del manual, mostrándose la primera información que se encuentre.

Tipo de Shell	Shell estándar	Clones libres
AT&T Bourne shell	sh	ash, bash, bash2
Berkeley "C" shell	csh	tcsh
AT&T Korn shell	ksh	pdksh, zsh
Otros intérpretes	–	esh, gush, nwsh

Figura 1: Intérpretes de comandos en Linux/UNIX

- **La opción –k** seguida de un argumento permite buscar información mediante una palabra clave. Si nosotros necesitamos información sobre un comando cuyo nombre no recordamos, pero sabemos algo de lo que hace, probamos a buscar información mediante una palabra clave, mostrándonos las páginas de ayuda de todos los tópicos en cuya página aparezca la palabra clave que hemos especificado.

Ejemplo 1 Supongamos que deseamos encontrar ayuda sobre el compilador de C del sistema, pero que no nos acordamos de cómo se llama. En ese caso, ejecutaríamos el siguiente comando,para pedir ayuda al sistema sobre todo aquello en cuya página aparezca la palabra "compiler": `man -k compiler`

Otra posibilidad de ayuda que tienen la mayoría de los sistemas UNIX es a través del propio comando. Cuando un comando se invoca con argumentos no válidos (bien sea por ser incorrectos, o por ser insuficientes) el mismo comando nos muestra un breve forma de uso (usage) del mismo. Por ejemplo, si ejecutamos el siguiente comando `cp` en el que faltan los dos argumentos para el comando `cp`, el sistema responderá con un mensaje del tipo:

```
Uso: cp [-hip] [--] src destino
o: cp [-hip] [--] src1 ... srcN directorio
o: cp {-R | -r} [-hip] [--] dir1 ... dirN dir_destino
```

No obstante, este mecanismo no está estandarizado, por lo que la salida producida en este caso puede variar de un sistema a otro. En algunos sistemas, por ejemplo, los comandos admiten una opción `--help` para mostrar ayuda sobre sí mismos.

3. El intérprete de comandos

El intérprete de comandos es el programa que recibe lo que se escribe en el terminal y lo convierte en instrucciones para el sistema operativo. En otras palabras, el objetivo de cualquier intérprete de comandos es ejecutar los programas que el usuario teclea en el prompt del mismo. El prompt es una indicación que muestra el intérprete para anunciar que espera una orden del usuario. Cuando el usuario escribe una orden, el intérprete ejecuta dicha orden. En dicha orden,puede haber programas internos o externos: los programas internos son aquellos que vienen incorporados en el propio intérprete, mientras que los externos son programas separados.

En el mundo Linux/UNIX existen tres grandes familias de Shells como se muestra en la Figura 1. Estas se diferencian entre si básicamente en la sintaxis de sus comandos y en la interacción con el usuario.

3.1. Sintaxis de los comandos

Los comandos tienen la siguiente sintaxis: `programa arg 1 arg 2 ... arg n`. Se observa que, en la "línea de comandos", se introduce el programa seguido de uno o varios argumentos. Así, el intérprete ejecutará el programa con las opciones que se hayan escrito.

Variable	Descripción
DISPLAY	Dirección IP a donde se envían los gráficos de los clientes X.
HOME	Directorio personal.
HOSTNAME	Nombre de la máquina.
MAIL	Archivo de correo.
PATH	Lista de directorios donde buscar los programas.
PS1	Prompt.
SHELL	Intérprete de comandos por defecto.
TERM	Tipo de terminal.
USER	Nombre del usuario.

Figura 2: Variables de entorno más usuales

Cuando se quiere que el comando sea de varias líneas, se separa cada línea con el carácter barra invertida (\). Además, cuando se quiere ejecutar varios comandos en la misma línea, los separa con punto y coma (;). Por ejemplo:

```
# make modules ; make modules install
```

En los comandos también se pueden utilizar los comodines:

- El asterisco (*) es equivalente a uno o más caracteres en el nombre de un archivo. Ej: ls *.tex lista todos los archivos que terminan en ".tex".
- El signo de interrogación (?) es equivalente a un único caracter. Ej: ls boletin1.te? lista el archivo boletin.tex completando el último carácter.
- Un conjunto de caracteres entre corchetes es equivalente a cualquier carácter del conjunto. Ej: ls curso linux.t[aeiou]x lista curso "linux.tex" seleccionando la e del conjunto.

3.2. Variables de entorno

Una variable de entorno es un nombre asociado a una cadena de caracteres. Dependiendo de la variable, su utilidad puede ser distinta. Algunas son útiles para no tener que escribir muchas opciones al ejecutar un programa, otras las utiliza el propio shell (PATH, PS1,...). La tabla 2 muestra la lista de variables más usuales.

La forma de definir una variable de entorno cambia con el intérprete de comandos, se muestra tcsh y bash siendo los dos más populares en el ámbito Linux:

```
bash: export VARIABLE=Valor
tcsh: setenv VARIABLE Valor
```

Por ejemplo, para definir el valor de la variable DISPLAY sería:

```
bash: export DISPLAY=localhost:0.0
tcsh: setenv DISPLAYlocalhost:0.0
```

3.3. Redireccionamiento de E/S

La filosofía de Linux/UNIX es en extremo modular. Se prefieren las herramientas pequeñas contareas puntuales a las meta-herramientas que realizan todo. Para hacer el modelo completo es necesario proveer el medio para ensamblar estas herramientas en estructuras más complejas. Esto se realiza por medio del redireccionamiento de las entradas y las salidas.

Filtros	Función
sort	Ordena las líneas de un texto
cut	Corta secciones de una línea
od	Convierte archivos a forma octal u otras
paste	Une líneas de diferentes archivos
tac	Concatena e imprime archivos invertidos
tr	Traduce o borra caracteres
uniq	Elimina líneas repetidas
wc	Cuenta bytes, palabras y líneas

Figura 3: Algunos Filtros en línea de comandos Linux/UNIX

Todos los programas tienen por defecto una entrada estándar (teclado) y dos salidas: la salida estándar (pantalla) y la salida de error(pantalla). En ellos se puede sustituir la entrada y salidas estándar por otro dispositivo utilizando los caracteres "<" y ">", es decir, hacer que se lea un archivo que contenga las opciones a ejecutar y un archivo de salida, respectivamente.

Por ejemplo, si se desea realizar una transferencia de archivos por ftp automática utilizando el programa ncftp con unas determinadas instrucciones preestablecidas. Se puede crear un archivo de entrada con dichas instrucciones y ejecutar el programa de la siguiente forma:

```
$ cat > getxwpe
open
ftp.rediris.es
user anonymous abc@cd.es
cd /sites/ftp.redhat.com/pub/redhat/linux/7.1/en/powertools/i386/RedHat/RPMS
mget xwpe*
bye
^ d
$ ftp -ni < getxwpe
```

Si por ejemplo se quisiera saber los archivos que empiezan por i o I y almacenarlo en un archivo el comando ls [iI]* > listado.txt sería suficiente.

Es importante resaltar que el carácter de redirección de salida `>` destruirá el archivo al cual apunta, si este existe, para ser reemplazadopor uno nuevo con los resultados del proceso. Si se desea anexar la información a uno ya existente debe usarse doble caracter >>.

3.4. Tuberías o pipes

En la línea de comandos la integración entre diferentes programas se realiza por medio de la redirección de las entradas y salidas a través de pipes o tuberías. Una tubería o pipe es una combinación de varios comandos que se ejecutan simultáneamente, donde el resultado del primero se envía a la entrada del siguiente. Esta tarea se realiza por medio del caracter barra vertical "|". Por ejemplo, si se quieren ver todos los archivos que hay en el directorio `/usr/bin`, se ejecuta lo siguiente:

```
ls /usr/bin | more
```

.

De este modo, la salida del programa `ls` (listado de todos los archivos del directorio/usr/bin) irá al programa `more` (modo paginado, es decir, muestra una pantalla y espera a que pulsemos una tecla para mostrar la siguiente). Dentro de esta estructura se han construido una serie de programas conocidos como "filtros" los cuales realizan procesos básicos sobre textos (Figura 3).

Algunos filtros han llegado a ser tan complejos que son en si, un lenguaje de procesamiento de texto,

de búsqueda de patrones, de construcción de scripts, y muchas otras posibilidades. Entre ellos podemos mencionar herramientas tradicionales en Linux/UNIX como awk y sed y otras más modernas como Perl.

3.5. Programación shell

La programación del shell es una de las herramientas más apreciadas por todos los administradores y muchos usuarios de Linux/UNIX ya que permite automatizar tareas complejas, comandos repetitivos y ejecutarlos con una simple llamada o hacerlo automáticamente a horas escogidas sin intervención de personas.

La programación shell en UNIX/Linux es, en cierto sentido, equivalente a crear archivos .BAT en DOS. La diferencia es que en UNIX/Linux es mucho más potente. Estos scripts pueden usar un sin número de herramientas como:

- Comandos del sistema Linux/UNIX (ej: ls, cut)
- Funciones intrínsecas del shell (ej: kill, nice)
- Lenguaje de programación del shell (ej: if/then/else/fi)
- Programas y/o lenguajes de procesamiento en línea. (ej: awk, sed, Perl)
- Programas propios del usuario escritos en cualquier lenguaje.
- El lenguaje de programación de cada shell provee de una amplia gama de estructuras de control que no serán vistas en este tema de introducción.

4. Comandos básicos de UNIX

4.1. Comandos para el manejo de ficheros

- **ls** El comandols lista unconjuntode ficheros,el contenidode undirectorio,el contenidode un árbolde directorios o cualquier combinación de los anteriores. Su formato es: ls [opciones] nombre. El formato del listado lo establecen las opciones. Algunas de las más usuales son:
  - **-l** muestra un listado largo, que contiene información detallada de los ficheros.
  - **-a** lista todos los ficheros, incluyendo aquellos cuyo nombre comienza por el caracter '.'
  - **-R** lista los directorios de forma recurrente
  - **-t** lista en orden cronológico, comenzando por los más recientemente actualizados.
- **cp/mv** Los comandos cp y mv se emplean respectivamente para copiar y mover ficheros, o incluso para copiar subárboles de directorios en el caso de cp. El formato de ambos comandos es:

  ```
  cp/mv [opciones] origen1 [origen 2 ... origen n ] destino
  ```

  , donde $origen_i$ son los ficheros, conjuntos de ficheros especificados mediante comodines, o directorios que se copian o mueven. Cuando se copian o mueven múltiples ficheros, el destino debe ser obligatoriamente un directorio. `destino` es el nombre de fichero destino o el directorio al que se copia o se mueve. Las opciones más comunes son:
  - **-f** No avisar si la operación machaca ficheros destino.
  - **-i** Avisar y pedir confirmación si la operación machaca ficheros destino.

- **-u** No copiar ni mover ficheros que sobreescriban a ficheros de igual nombre con fecha de última modificación igual o posterior a la de los mismos.
- **-r** Copiar subdirectorios de forma recurrente (sólo cp).

- **chmod** El comando chmod se usa para seleccionar autorizaciones de acceso a un archivo o directorio. Es posible asignar tres clases de autorización: Leer (indicado por una "r"), escribir (indicado por una "w") y ejecutar (válido únicamente para programas; indicado por una "x").

  Hay tres grupos de personas a los que se puede otorgar cada una de las autorizaciones (leer, escribir y ejecutar): el propietario del archivo o directorio (conocido como Usuario), el grupo al que pertenece el propietario (conocido como Grupo) y todos los demás (Otros)

  El siguiente es el formato básico:

  ```
  chmod grupos[+|-]permisos fichero
  ```

  Por ejemplo, este comando:

  ```
  chmod o+x editor.pl
  ```

  otorga a todos los demás (Otros) autorización para ejecutar el archivo editor.pl (un script en perl). Este comando:

  ```
  chmod go-w mydata.dat
  ```

  quita (el signo menos) el permiso de escribir (w) de los conjuntos de usuarios Grupo y Otros.

  También se pueden representar permisos en formato octal, es decir:

  ```
  r = 4 w = 2 x = 1
  rwx = 7 (4+2+1)
  rw- = 6 (4+2)
  r-x = 5 (4+1)
  rw-r--r-- = 644
  rw------- = 600
  rwxr-xr-x = 755
  ```

  Para cambiar los permisos para que sólo el propietario pueda leerlo y escribirlo, teclee:

  ```
  chmod 600 <fichero>
  ```

  Para que además sea ejecutable por todos:

  ```
  chmod 755 <fichero>
  ```

4.2. Comandos para el manejo de la Entrada/Salida

- **cat** El comando cat escribe el contenido de uno o más ficheros de texto en la salida estándar. Su formato es:

  ```
  cat [opciones] [fichero 1 fichero 2 ... fichero n]
  ```

  donde $fichero_i$ son los ficheros cuyos contenidos se escriben en la salida estándar. En caso de que no se especifique ningún fichero, o que se especifique el carácter '-' como nombre de fichero, cat escribe su entrada estándar sobre la salida estándar. Algunas de las opciones más frecuentes son:

  - -b Enumera todas las líneas que no estén en blanco, a partir de 1.
  - -n Enumera todas las líneas, tanto las que están en blanco como las que no.

- **more** El comando more permite visualizar el contenido de un fichero de texto página a página. Normalmente este comando es utilizado por otros comandos o por terceras aplicaciones para visualizar su salida. Ejemplo de comando que hace esto suele ser man. El formato del comando more es:

```
more [opciones] fichero 1 [fichero 2 ... fichero n ]
```

donde $fichero_i$ son los ficheros cuyos contenidos se muestran página a página. Las opciones más comunes son:

- **-n** donde n es el número de líneas que se muestran por cada página.
- **-f** hace que more cuente líneas lógicas en lugar de físicas. Esto evita que las líneas largas se muestren usando varias líneas en pantalla, forzando a que se muestren truncadas.
- **-p** suprime el scroll. En su lugar, por cada página limpia la pantalla y muestra el texto a continuación.
- **-c** suprime el scroll. En su lugar, por cada página comienza escribiendo en la primera línea de la pantalla, y a continuación escribe las líneas de texto, borrando la porción de cada línea de pantalla que no se use.
- **-s** compacta varias líneas en blanco consecutivas en una sola línea en blanco.
- **+n** donde n es un número. Comienza en la línea n-ésima.
- `+/patrón` busca la primera ocurrencia en el texto del patrón, comenzando en dicho punto la presentación.

- **echo** Los comandos **echo** y **print** muestran en la salida estándar una cadena dada, entendiendo una cadena como una secuencia de palabras separadas por caracteres de tabulación o espacios en blanco. Tras la cadena mostrada se produce un salto de línea. El formato de ambos comandos es:

  ```
  echo cadena, print [-n] cadena
  ```

  donde -n indica que no se debe producir el salto de línea a continuación de la cadena.

- **read** El comando read lee de la entrada estándar el valor de una o más variables. El formato del comando es:

  ```
  read variable 1 [variable 2 ... variable n ]
  ```

  donde $variable_i$ son los nombres de las variables que se leen.

  El comando read lee una línea completa de texto, asignando una palabra a cada variable. Las palabras se supone que están delimitadas por tabuladores o espacios en blanco. En caso de que se lean más palabras que variables, todas las palabras "de sobra" al final de la línea se asignarán a la última variable. Si el número de palabras es menor que el número de variables, las últimas variables reciben como valor una cadena vacía.

- **grep** El comando grep toma como entrada uno o más ficheros, y muestra en la salida estándar aquellas líneas de los ficheros de entrada en la que se encuentre una subcadena que cumpla un patrón dado. Si se especifican múltiples ficheros de entrada, cada línea de salida va precedida por el nombre del fichero. Si no se especifica un fichero de entrada, o si se especifica el carácter '-' como nombre de fichero, grep lee de la entrada estándar.

  El formato del comando grep es:

  ```
  grep [opciones] patron [fichero 1 fichero 2 ... fichero n ]
  ```

  donde $fichero_i$ son los ficheros cuyas líneas se procesan y patrón es el patrón que se busca. Este puede ser una expresión regular de la forma que se van a describir a continuación. Es una buena costumbre encerrar el patrón entre comillas simples. Por defecto, interpreta el patrón como una expresión regular básica. Las opciones más comunes son:

  - **-E** Interpreta el patrón como una expresión regular extendida.
  - **-F** Interpreta el patrón como una o más cadenas fijas, separadas por caracteres de nueva línea.
  - **-h** Suprime el nombre de fichero al principio de cada línea aun en el caso de que se procesen múltiples ficheros.
  - **-i** No distingue entre mayúsculas y minúsculas.
  - **-l** Muestra sólo una lista con los ficheros de la entrada que en algún lugar contienen el patrón.

Metacarácter	Significado
.	Representa a cualquier carácter
[lista de caracteres] o [carácter$_1$ – carácter$_n$]	Representa a uno cualquiera de los caracteres de la lista, o a cualquier carácter comprendido entre carácter$_1$ y carácter$_n$ según el orden ASCII. Si el primer carácter tras el corchete [es el carácter '^', el significado se invierte, es decir, representa a todos los caracteres que no están en la lista o en el intervalo. Dentro de los corchetes, los metacaracteres '$', '*', y '/' pierden su significado especial.
*	Pospuesta a cualquier expresión y significa cero o más ocurrencias de dicha expresión.
^	Antepuesta a cualquier expresión regular, indica que la expresión debe aparecer al comienzo de la línea solamente.
$	Pospuesta a cualquier expresión regular, indica que la ex-presión debe aparecer al final de la línea solamente.
\	El significado de cualquier metacarácter puede ser ignorado antecediéndole por la barra inversa ('\'), en cuyo caso el metacarácter se interpreta de forma literal.

Figura 4: Expresiones regulares

- **-v** Hace que grep muestre las líneas que no contienen el patrón.
- **-w** Requiere que el patrón coincida con una palabra completa
- **-f f** Indica a grep que lea la expresión regular del fichero f en lugar de la línea de comandos

Una expresión regular es una plantilla de texto construida mediante caracteres literales y alguno(s) de los metacaracteres siguientes, y cuya finalidad es representar a un conjunto de cadenas. Si una cadena puede ser representada mediante la expresión regular, se dice que la cadena "satisface" dicha expresión.

Los metacaracteres con los que podemos escribir las expresiones regulares básicas en UNIX son:

- **who** El comando who proporciona información sobre los usuarios conectados a la máquina. Su formato es:

  ```
  who [opciones]
  ```

  Si es invocado sin opciones, proporciona la siguiente información por cada usuario conectado en el momento: Nombre de usuario, dispositivo lógico (tty) al que está conectado, tiempo que lleva conectado (normalmente, fecha y hora de conexión), nombre de la máquina o display X desde el que se conecta. Las opciones más comunes son:

  - **-m** Igual que "who am i"
  - **-q** Proporciona el nombre de los usuarios conectados e indica cuántos hay en total.
  - **-u** Tras la hora de conexión, muestra el tiempo (horas y minutos) que el usuario lleva inactivo. Un punto (.) indica que el usuario ha estado activo en el último minuto, y la cadena 'old' indica que el usuario lleva más de 24 horas inactivo.

- **sort** El comando sort se emplea para ordenar, fusionar ordenadamente o comprobar si están ordenadas todas las líneas del fichero o ficheros de entrada. Por defecto, sort escribe en la salida estándar. Su formato es:

  ```
  sort [opciones] [fichero 1 fichero 2 ... fichero n]
  ```

  donde $fichero_i$ son los ficheros de entrada. Si no se especifica fichero de entrada, o si se especifica '-' como fichero de entrada, sort leerá de la entrada estándar.

  El comando sort considera cada línea como una lista de campos de texto, estando dichos campos delimitados por espacios en blanco o por caracteres de tabulación. Para comparar entre si dos líneas, inicialmente se comparan por parejas todos los campos, hasta que se termina con la lista de campos, o hasta que se encuentra una diferencia. En caso de que la comparación haya llegado al final con el resultado de que ambas líneas son iguales, aún se hace una última comparación de ambas líneas caracter a caracter, tomándose el resultado final de ésta comparación. Las opciones más comunes del comando sort son:

  - **-c** comprueba si los ficheros de entrada están todos ordenados. Caso de no estarlo alguno de ellos,se presenta un mensaje de error y sort termina con un estado de 1.

- **-m** fusiona todos los ficheros de entrada (línea a línea) en un único fichero ordenado. Para ello es necesario que los ficheros de entrada estén ordenados. Fusionar es más rápido que ordenar, pero nótese la necesidad de que los ficheros de entrada estén ordenados.
- **-b** ignorar los espacios en blanco al principio de cada línea.
- **-d** ignorar todos los caracteres excepto letras, números y espacios en blanco.
- **-f** considerar las letras minúsculas como su correspondiente mayúscula
- **-i** ignorar caracteres no ASCII.
- **-n** considerar que los campos que tengan formato de uno o más dígitos, opcionalmente precedidos por un signo '-' y terminados en un punto decimal y un numero de dígitos, es un campo numérico y como tal se tiene en cuenta en las comparaciones.
- **-r** ordenar en orden inverso (de mayor a menor)
- **-o f** generar como salida un fichero con nombre f.
- **-t s** considerar que los campos están delimitados por el carácter s
- **+p 1 -p 2** Especifica p 1 como el índice del primer campo que se usa como clave de ordenación, siendo opcionalmente p 2 el índice del primer campo que no interviene como clave de ordenación. En caso de no especificarse p 2, se usa como clave de ordenación el resto de los campos hasta el final de la línea.

5. Ejercicios

5.1. Ejercicio 1

En Unix se almacena la información de los usuarios registrados en el sistema en el archivo /etc/passwd, cuyo formato es el siguiente:

```
#nombre:contrasena:UID:GID:comentarios:directorio_home:shell_defecto
jperez:X:1130:103:Juan Perez:/home/jperez:/bin/bash
```

NOTAS:

Otro que puede ser de utilidad es el comando `wc`. Este cuenta el número de líneas, palabras o letras de un archivo, y su sintaxis la siguiente:

```
wc [opcion...] [archivo...]
```

Si se omite el argumento archivo, wc tomará los datos (naturalmente) de la entrada estándar. La lista de opciones más importantes es la siguiente:

- **-c** Cuenta el número de bytes.
- **-l** Cuenta el número de líneas.
- **-w** Cuenta el número de palabras.

Como ejemplo, se pueden contar las líneas del archivo /etc/passwd y de esta manera se sabrá rápidamente cuantos usuarios tiene definidos el sistema:

```
wc -l /etc/passwd
```

Escriba comandos Unix para responder a las siguientes cuestiones:

a) A partir del archivo /etc/passwd obtenga aquellos usuarios que no trabajen por defecto con el shell /bin/bash.

b) A partir del mismo archivo obtenga el número de usuarios que tienen su directorio raíz dentro de la carpeta /home.

c) Obtenga aquellos usuarios, **ordenados de forma descendente**, cuyo username comienza por m, el comando debe guardar el resultado en un archivo /home/SU_USUARIO/practica/ejercicio1c.

d) **Resolver el problema como un solo comando.** Obtener todos los ficheros del directorio actual incluyendo los ocultos ordenados alfabéticamente descendentemente. El resultado debe ser parecido a:

```
.npm
nohup.out
.mysql_history
mail
.lesshst
historial.txt
.gitconfig
.gconf
.dbus
.composer
cloudera
.cache
.bash_history
appendonly.aof
..
.
```

5.2. Ejercicio 2

Dada la estructura de directorios representada a continuación, y suponiendo que usted se encuentra en el directorio boletines, escriba en una línea **un comando** para realizar las siguientes operaciones:

```
/
    +---home
        | + ---- practica
        | + ----boletines
        | + ---05-senales
            | | + ----ej1
            | | + ----ej2
            | | + ----ej3
        | + ---07-mensajes
            | | + ----ej1
            | | + ----ej2
            | | + ----ej3
        | + ---08-sockets
    +---tmp
```

a) Borre el directorio `05-senales`.

b) Copie, sin cambiar de directorio, el directorio `07-mensajes` y todo su contenido al directorio `/tmp`.

c) Póngale los permisos de acceso más restrictivos al directorio que acaba de copiar en el apartado anterior, de forma que solamente puedan hacer una copia del mismo los usuarios que pertenecen al mismo grupo de trabajo que usted.

d) Renombre el directorio `08-sockets` como `08-sockets-inet`.

e) Cree un archivo `conten.ndx` que contenga una lista ordenada alfabéticamente en orden creciente y con las líneas numeradas de todas las entradas que contiene el directorio actual.

6. Referencias

- Reina, T. (2006). Laboratorio de Sistemas Operativos.