1 Introducción.

Una vez instalado e inicializado un sistema Linux se dispone de dos vías fundamentales de interacción: una gráfica (si se instaló una interfaz X y se configuró adecuadamente) y una texto conocida como consola o terminal.

Al igual que Unix, Linux ofrece el mecanismo de consolas o terminales virtuales. Este consiste en que a partir de una entrada (el teclado) y con una salida (el monitor) se simulen varias terminales, donde el mismo, o distintos usuarios puedan conectarse indistintamente. De esta forma es posible tener más de una sesión abierta en la misma máquina y trabajar en ellas. Este mecanismo también facilita la característica multiusuario del sistema Linux pues las diferentes conexiones se pueden establecer con diferentes usuarios.

Por defecto, las consolas desde la uno a la seis tienen asociado un programa que permite conectarse al sistema en modo texto, mientras que la siete, si se instaló y activó el ``modo gráfico'', constituye una consola gráfica.

El cambio de una consola a otra se realiza a través de la combinación de teclas Alt y Fx (las teclas de Función), donde x oscila entre 1 y 12. De esta forma se pueden acceder un total de 24 consolas virtuales: para las doce primeras se utiliza el Alt izquierdo y para las otras doce el derecho. Por ejemplo para llegar a la consola 16 se presionarían las teclas Alt derecho y F4. No obstante normalmente solo se puede acceder a las consolas con algún proceso o funcionalidad definida.

Desde una consola gráfica para cambiar a otra tipo texto se debe además presionar la tecla Ctrl, pues las combinaciones Alt + Fx son capturadas e interpretadas por las aplicaciones gráficas de otra forma.

Así, si pulsamos Ctrl – Alt – F1 accedemos a la consola número 1 (tty1), con Ctrl – Alt – F5 accedemos a la consola número 5 (tty5), etc. Si estamos corriendo Debian en una maquina virtual Vmware nos encontraremos con el problema que dicho software utiliza la combinación Ctrl – Alt como hotkey para liberar el cursor del ratón, por lo que tendremos que acceder a la configuración del Vmware y cambiar este hotkey, a por ejemplo, Ctrl – Alt – Mayus.

Con la tecla Alt izquierda combinada con los cursores (derecho e izquierdo) se puede además, realizar un movimiento circular entre todas aquellas consolas que tengan un proceso asociado (texto, gráfico, etc.).

Si nos remitimos a una consola texto podremos apreciar que en ella se mostrará, el nombre de la distribución, la versión de la distribución, la versión del kernel y la arquitectura de la máquina. También aparecerá el nombre que se le asignó al sistema en la instalación y la palabra login.

password de dicho usuario (tened cuidado ya que al entrar dicho password no se muestra ningún eco en la pantalla). Si ambos son válidos se establecerá la conexión y se mostrará lo que se conoce como prompt del sistema, con forma similar a esta:

usuario@pinguino:~$

Aquí ha abierto sesión un usuario con nombre "usuario", en una máquina que se llama "pinguino", está actualmente en el directorio "~" y sabemos que es un usuario normal y no el root por que el prompt termina con un símbolo "$" (si fuera root terminaría con un símbolo "#").
Este entorno de texto donde nos encontramos y que nos permite introducir comandos es conocido comúnmente como Shell (caparazón). Este Shell es capaz de interpretar una gran gama de comandos y sentencias. Constituye a su vez un poderoso lenguaje de programación mediante scripts.
GNU-Linux tiene la filosofía de no obligar al usuario a utilizar un programa determinado para cada acción, sino que siempre da la libertad de elegir el programa que queremos utilizar. Lo mismo ocurre con el Shell que vayamos a utilizar para acceder al sistema. El Shell que más se usa es conocido como bash, aunque existen una gran variedad de ellos, como por ejemplo csh, ksh, etc.
Algunas características que merece la pena conocer de bash son:

- Auto completar durante la escritura. Al teclear uno o varios caracteres se puede pulsar TAB con el objetivo de que en caso de que pueda completarse de forma unívoca un comando, nombre de fichero o una variable (en dependencia del contexto), complete de forma automática (se escriba el resto de la palabra). Si existieran varias posibilidades para completar la palabra, se oirá un sonido y volviendo a pulsar TAB se mostrarán en pantalla todas las posibilidades existentes. En caso de existir muchas posibilidades (por defecto más de 100) se pregunta si se desea mostrarlas todas o no.
- Historial de comandos. Esta es una facilidad de muchos otros shells que permite el movimiento a través de los últimos N comandos ejecutados, en la sesión actual o en las anteriores. N por defecto es 1000, pero puede modificarse. Para moverse arriba y abajo se suelen utilizar los cursores.
- Poderosas estructuras de control para realizar scripts. (Procesos por lotes). Se pueden utilizar instrucciones if, for, while, select, case, etc.
- Definición de funciones y alias para comandos. Las funciones permiten definir subrutinas programadas usando el lenguaje de bash y los alias, asociar nombres a llamados a comandos con ciertas opciones y argumentos de forma más nemotécnica o abreviada.

1.1 Formas generales de obtener ayuda en Linux. (Modo Texto).

Existen múltiples y variadas formas de obtener ayuda en un sistema Linux. A continuación se describen algunas de ellas:

1) Muchos comandos poseen una opción para mostrar una ayuda breve acerca de su utilización. Esta opción usualmente es **-h**, **--help** ó -?.
2) El comando **man** formatea y despliega un manual bastante amplio acerca de comandos, formatos de ficheros de configuración, llamados al sistema, etc. Los manuales están disponibles y pueden instalarse en múltiples idiomas. Estos se dividen internamente en secciones. Un mismo objetivo puede estar representado en varias secciones. De no especificarse ninguna sección a través del primer argumento del comando se tomará la primera donde aparezca.

   Ejemplo: $ **man chmod**
3) Existe un conjunto de comandos integrados (builtin) al bash que no poseen un manual propio. Para ellos se puede emplear el comando help. Si se usa man sólo se mostrará la lista de los

comandos integrados. La ayuda que muestra help es un fragmento del manual de bash ($ **man bash**), el cual es muy amplio y resulta incómoda la búsqueda. Algunos de los comandos integrados al bash y que veremos más adelante son: cd, fg, bg, logout, exit, umask, set, help, source, alias, echo, kill, jobs y export.

Ejemplo: $ **help logout**

4) El programa **info** despliega información acerca de comandos, en ocasiones más amplia que la que brinda man. Las páginas de info poseen una estructura arbórea (nodos), a través de las cuales se puede navegar con ayuda de comandos especiales.

   Ejemplo: $ **info ln**

5) El comando **whatis** realiza una búsqueda en las secciones del man y muestra la descripción abreviada del comando en cada una de las secciones que aparezca. Este comando trabaja con una base de datos que se actualiza periódicamente y se crea con el comando makewhatis.

   Ejemplo: $ **whatis shadow**

6) El comando **apropos**, dada una palabra busca en toda la base de datos del whatis desplegando todo lo encontrado. Ejemplo: $ **apropos passwd**

7) Muchos programas se empaquetan con su documentación en diversos formatos. Normalmente estas ayudas se agrupan en el directorio **/usr/share/doc**. También existen los **HOWTOs** (Como lograr...) escritos en muchos idiomas y formatos para varios temas disponibles. Algunos se empaquetan como cualquier otro programa o se localizan en Internet,

8) En **Internet** de forma general existen una gran cantidad de grupos de noticias, listas de discusión, sitios Web y FTP sobre Linux. En muchos de ellos incluso se pueden encontrar paginas de ayuda sobre los comandos en castellano, mientras que en nuestra versión de Linux es posible que nos aparezcan en Ingles.

1.2 Usuarios y grupos.

Como ya se ha afirmado Linux es un sistema multiusuario, lo cual permite que varios usuarios puedan conectarse y trabajar en él de forma simultánea. Las conexiones como ya se ha visto se pueden realizar a través de varias terminales locales o utilizando servicios de red como el Telnet y el SSH.

Un usuario se caracteriza por su login el cual debe indicar para conectarse al sistema, además de su password o contraseña. Además puede poseer un conjunto de datos adicionales mencionados más adelante.

El usuario con más privilegios en Linux es aquel cuyo login es root. Este es el único con derechos suficientes para crear o eliminar a otros usuarios, además de acceder a todo el sistema de ficheros sin ninguna restricción.

En Linux además existen grupos de usuarios también administrados por root o por un usuario designado por este. Los grupos permiten otorgar los mismos privilegios a un conjunto de usuarios.

Siempre que se añada un usuario al sistema se creará un grupo con su mismo nombre, llamado grupo primario. Durante la creación o posteriormente, se podrá incorporar el usuario a otros grupos secundarios.

Tanto los usuarios como los grupos se identifican por el sistema a través de un identificador (ID) numérico. El usuario root siempre tiene el ID cero. Cada usuario cuando se conecta al sistema posee un identificador de usuario asociado (uid) y un identificador de grupo (gid).

Para comprobar el uid de nuestro usuario, y el gid del grupo principal al que pertenece, podéis ejecutar la orden **id**.

Al añadir un usuario también se creará un directorio base para el mismo con el nombre de su login. Este directorio se coloca por defecto en el directorio /home excepto para root, cuyo directorio base es /root.

La información asociada a los usuarios en un sistema Linux se guarda en el fichero /etc/passwd y las contraseñas y datos afines en /etc/shadow. Por su parte la información de los grupos, sus miembros y passwords están en /etc/group y /etc/gshadow respectivamente.

Para crear un usuario se usa el comando **useradd**, aunque es mucho mejor usar un script que viene instalado en Linux, que se llama **adduser**. Este script controla como funciona useradd y permite realizar funciones avanzadas, como crear automáticamente el directorio del usuario, configurar su perfil, etc.
Como ejemplo, vamos a realizar el siguiente ejercicio:
1) Creamos dos usuarios, uno con nombre margarita y otro con nombre floripondio. En ambos usamos de contraseña 123 por ejemplo.

a. sudo adduser margarita

b. sudo adduser floripondio

2) Creamos un grupo con nombre flores, y añadimos a los dos usuarios anteriores a dicho grupo.

a. sudo addgroup flores

b. sudo adduser margarita flores

c. sudo adduser floripondio flores

3) Editamos el fichero /etc/passwd y comprobamos como se han creado dos líneas, una para cada uno de los usuarios que hemos creado.

a. less /etc/passwd

4) Editamos el fichero /etc/shadow y veremos cómo se han creado también dos líneas, una para cada uno de los usuarios.

a. less /etc/shadow

5) Editamos el fichero /etc/group y comprobamos como se ha creado una línea para el grupo creado, donde además comprobamos que se han añadido como miembros los usuarios.

a. less /etc/groups

Para comprobar que los usuarios se han creado bien, vamos a realizar lo siguiente:

1) acceder al 4º terminal (Control – Alt – F4), y hacer un login con el usuario margarita. Una vez abierta sesión, ejecutar el comando whoami y el comando id. Deberíamos comprobar que efectivamente estamos en el grupo flores. Para cerrar la sesión usamos el comando logout.

2) Acceder al 5º terminal, hacer un login con floripondio y comprobad lo mismo.

(Para volver al terminal gráfico desde un terminal de texto, accedemos al 7º terminal).
Comprobaremos como toda la gestión de usuarios y grupos de Linux, en realidad se basa en la modificación de una serie de ficheros de texto que están en el directorio etc.
Intentad ahora crear otro usuario, con nombre pimiento y asignarlo como miembro del grupo verduras que también habrá que crear. Estas acciones realizarlas sin usar los comandos adduser y addgroup, y en su lugar modificar directamente los ficheros passwd, shadow y groups.
Una vez hecho, acceder a los terminales de texto y como hemos hecho anteriormente, comprobad que pimiento puede realizar login, y que pertenece a los grupos que debe pertenecer.
Un problema de añadir usuarios de esta forma, es que no se crea su directorio home, cosa que deberemos hacer a mano.

En las siguientes secciones mencionaremos y describiremos brevemente la utilidad de un grupo de comandos básicos para interactuar con un sistema Linux. Muchos de estos comandos están presentes también en los sistemas Unix aunque a veces varían sus opciones y argumentos en dependencia de las versiones. Por esto no puede encontrarse una documentación que diga la última palabra respecto a uno u otro comando, la práctica es la mejor forma de comprobar como funciona cada uno. Para su mejor comprensión clasificamos los comandos en varias categorías:

Comandos para manipular ficheros y directorios
Comandos para paginar, visualizar y editar ficheros
Comandos para hacer búsquedas de ficheros y patrones
Comandos para filtrar ficheros
Comandos para compactar y agrupar ficheros
Comandos para la comunicación entre usuarios
Comandos para desconectarse del sistema
Comandos para administrar usuarios y grupos
Comandos variados

1.3 Comandos para manipular ficheros y directorios.

Comando ls

El comando ls permite listar el contenido de un directorio.

Sintaxis: ls [opciones] [directorio | fichero]

Algunas opciones:

- -l : muestra la salida en formato largo.
- -R: lista recursivamente un directorio.
- -a : lista además los ficheros ocultos (sus nombres comienzan con punto).
- -h : muestra el tamaño de los ficheros en forma más legible (Ej.: 16M, 4k, etc.)

- -i : muestra el identificador del i-nodo asociado a cada elemento.

Ejemplos:

$ ls -hl /etc

$ ls -R /usr

$ ls -al

$ ls -ali ..

Un grupo de opciones que se suele utilizar bastante es lia (ls –lia)

Comando cd

El comando cd se utiliza para cambiar el directorio actual.

Sintaxis: cd [directorio]

Ejemplos:

$ cd /tmp

$ cd # cambia al directorio home del usuario.

$ cd ~ # cambia al directorio home del usuario.

Comando pwd

El comando pwd indica el camino absoluto del directorio en el cual nos encontramos actualmente.

Ejemplo:

$ **pwd # nos devuelve algo como /home/pepe/backup/pruebas**

Comando mkdir

El comando mkdir se utiliza para crear directorios.

Ejemplos:

$ mkdir bin

$ mkdir /bin

$ mkdir -p docs/linuxdocs/howtos/pdf # se crean los directorios intermedios si es necesario

Comando mv

El comando mv mueve un fichero hacia otro, o varios ficheros hacia un directorio. Este permite a su vez renombrar ficheros o directorios.

Sintaxis: mv [opciones] <fuente> <destino>

mv [opciones] <ficheros> <directorio>

Algunas opciones:

- -i : ejecuta el comando de forma interactiva, o sea, pregunta ante de sobrescribir el destino si existiera.
- -u : actualiza (upgrade) el destino con el fuente solo si este es más reciente.

Ejemplos:

```
$ mv mail.cf mail.cf.old        # renombra un fichero
$ mv -i *.txt /tmp              # mueve ficheros terminados en .txt al directorio /tmp
$ mv -u program.c src/          # actualiza el fichero destino si es menos reciente que el
fuente
```

Comando cp

El comando cp permite copiar un fichero en otro, o varios ficheros en un directorio.

Sintaxis:

cp [opciones] <fuente> <destino>

cp [opciones] <ficheros> <directorio>

Algunas opciones:

- -R : copia recursivamente un directorio.
- -i : utiliza una forma interactiva (pregunta antes de sobrescribir el destino).
- -l : hace enlaces fuertes a los ficheros fuentes en lugar de copiarlos.

Ejemplos:

```
$ cp /etc/passwd .    # copia un fichero en el directorio actual
$ cp -i /usr/bin/*sh /tmp   # copia interactivamente los ficheros terminados en sh en un  directorio llamado /tmp
```

Comando rm

El comando rm se utiliza para borrar (desenlazar) ficheros

Sintaxis: rm [opciones] <ficheros | directorios>

Algunas opciones:

- -r : borra recursivamente un directorio.
- -f : borra forzosamente en caso de que no se tenga permiso de escritura en forma directa.
- -i : ejecuta el comando de forma interactiva.

Ejemplos:

```
$ rm prueba
$ rm -i bin/*
$ rm -rf temp/
```

Comando chown

El comando chown se utiliza para cambiar el dueño y el grupo de un fichero. Existe también el comando chgrp que se emplea de forma similar pero para cambiar el grupo solamente. El dueño de

un fichero solo lo puede cambiar el usuario root mientras que el grupo además de root, lo puede cambiar el propio dueño, siempre que pertenezca al nuevo grupo.

El propietario y el grupo de un fichero se puede comprobar con un ls -l

Sintaxis:

chown [opciones] <dueño>[.grupo] <ficheros>

chown [opciones] <grupo> <ficheros>

Opción:

- -R en los directorios cambia el dueño y/o el grupo recursivamente.

Ejemplos:

```
# chown pepe.pepe tesis/ # cambia el propietario de tesis a pepe y el grupo de tesis al grupo pepe

# chown –R root /tmp/oculto # cambia todos los ficheros que esten en el directorio oculto y coloca como propietario al usuario root

# chgrp ftp /usr/ftp
```

1.4 Comandos para paginar, visualizar y editar ficheros

Comando cat

El comando cat concatena (conCATenate) ficheros y los imprime en la salida estándar. Si no se le pasa ningún argumento lee de la entrada estándar. Existe también zcat que hace lo mismo pero con ficheros comprimidos. Si solo se da el origen a cat, utiliza como salida la pantalla. Es decir, cat hola muestra por pantalla el fichero hola. Si solo se da la salida a cat (cat > fichero) utiliza como entrada el teclado.

Ejemplo:

```
# cat /etc/passwd /etc/shadow

$ cat > fichero

$ cat < origen > destino
```

Editor vi

El editor vi es el editor estándar de Unix y esta orientado a comandos. Existe una versión conocida como vim (Vi IMproved) muy poderosa que permite la edición de múltiples ficheros, edición automatica para varios lenguajes de programación, ayuda en línea, selección visual, varios niveles de deshacer, etc. Para algunos usuarios, vi resulta incómodo pues para utilizar todas sus potencialidades es necesario conocer muchas combinaciones de teclas, pero si se llega a dominar resulta muy funcional. Su principal virtud es que encontraremos vi en prácticamente cualquier

versión de Unix que usemos, cosa que no se puede decir de otros editores (joe, pico, edit, gedit, nano, emacs, etc.).

Básicamente vi posee dos modos de interacción: el de inserción (edición) y el de comandos. Para pasar al modo comando se pulsa **Esc** y para pasar al de inserción se pulsa **i**.

Algunos comandos útiles en vi (pulsando ESC para pasar al modo de comandos, que veremos en la ultima línea de la pantalla).

- dd - borra la línea actual.
- D - borra desde la posición actual hasta el final de la línea.
- dG - borra hasta el final del fichero.
- u - deshace el último comando.
- :q - sale del editor (si se han hecho modificaciones y no se ha salvado se genera un error).
- :q! - sale sin salvar.
- :w - salva.
- :wq - salva y sale.
- :x - salva y sale.
- <n><comando> - ejecuta el comando c n veces.

Comandos more y less

Los comandos more y less paginan (dividen en páginas) uno o varios ficheros y los muestran en la terminal (pantalla). De no indicárseles un fichero, paginan la entrada estándar (que se manda mediante una tuberia).

Se diferencian en las facilidades que brindan. Por ejemplo more es más restrictivo en cuanto al movimiento dentro del texto, mientras que less no limita este aspecto pues acepta el empleo de todas las teclas de movimiento tradicionales. Cuando se alcanza el final del último fichero a paginar, more termina automáticamente, no así less. También more muestra sucesivamente el porcentaje del fichero visto hasta el momento. Tanto less como more proveen una serie de comandos para moverse con facilidad dentro del texto paginado.

Ejemplos:

```
$ less /etc/passwd
$ more /etc/passwd
$ cat fichero | less
```

Algunas teclas que podemos usar mientras usamos estos programas son:

- q - permite interrumpir el proceso y salir.
- /p - realiza búsquedas del patrón p dentro del texto. Para repetir la búsqueda del mismo patrón sólo es necesario escribir /.
- [n]b - en more permite regresar n páginas (por defecto n es 1).

- [n]f - en more se adelantan n páginas y en less, n líneas.

El man, para dar formato a su salida, utiliza por defecto el paginador less. Existen además los comando zless y zmore que permiten paginar con less y more respectivamente, a los ficheros comprimidos sin necesidad de descomprimirlos previamente.

1.5 Comandos para hacer búsquedas de ficheros y patrones

Comando grep

El comando grep (Globally Regular Expressions Pattern) busca patrones en ficheros. Por defecto devuelve todas las líneas que contienen un patrón (cadena de texto) determinado en uno o varios ficheros. Utilizando las opciones se puede variar mucho este comportamiento. Si no se le pasa ningún fichero como argumento hace la búsqueda en su entrada estándar.

Sintaxis: grep [opciones] <patrón> [ficheros]

Algunas opciones:

- -c devuelve sólo la cantidad de líneas que contienen al patrón.
- -i ignora las diferencias entre mayúsculas y minúsculas.
- -H imprime además de las líneas, el nombre del fichero donde se encontró el patrón. Es así por defecto cuando se hace la búsqueda en más de un fichero.
- -l cuando son múltiples ficheros sólo muestra los nombres de aquellos donde se encontró al patrón y no las líneas correspondientes.
- -v devuelve las líneas que no contienen el patrón.
- -r busca en un directorio de forma recursiva.
- -n imprime el número de cada línea que contiene al patrón.

Ejemplos:

```
$ grep linux /usr/share/doc
$ grep root  /etc/passwd
# grep -n error /var/log/messages
$ grep -i pepe /etc/passwd
$ grep -c root /etc/group
$ grep -l -r -i img /var/www/html/
$ ls –lia | grep "carta roja"
```

Comando find

El comando find es uno de los más poderosos en un sistema Linux. Permite buscar de forma recursiva en un directorio a todos los ficheros que cumplan ciertas condiciones. Las condiciones pueden estar relacionadas con el nombre de los ficheros, el tamaño, los permisos, el tipo, las fechas de acceso y modificación, etc.

Sintaxis: find [camino] [opciones]

Algunas opciones:

- **-name <expresión>** permite especificar patrones para los nombres de los ficheros a buscar.
- **-iname <expresión>** permite especificar patrones para los nombres de los ficheros a buscar sin tener en cuenta mayúsculas y minúsculas.
- **-type <tipo>** permite indicar el tipo de fichero a buscar. Este puede ser d para directorios, f para ficheros regulares, l para enlaces simbólicos, b para dispositivos de bloque, c para dispositivos de carácter, p para tuberías y s para sockets.
- **-size +/-<n>** permite indicar el tamaño máximo y/o mínimo de los ficheros a buscar. Por defecto el tamaño se expresa en bloques de 512 bytes, pero si se precede este por un carácter c se referirá a bytes, k a kilobytes, w a palabras de dos bytes y b a bloques.
- **-perm [+|-]<modo>** permite referirse a aquellos ficheros cuyos permisos sean exactamente modo, incluya todos los de modo (signo -) o incluya alguno de los de <modo> (signo +). El valor de <modo> se expresa en forma numérica.
- **-exec <comando> ;** permite definir un comando a ejecutarse para cada resultado de la búsqueda. La cadena {} se sustituye por el nombre de los ficheros encontrados. El caracter ; permite indicar la finalización del comando. (Tanto {} como ; tienen que ir entre comillas o entre contrabarras para evitar que sea sustituido por el shell).

Ejemplos:

```
$ find /etc -name '*.conf'   # busca en /etc todos los ficheros con extensión conf

$ find / -size +10240k -size -20480k      # busca los ficheros cuyo tamaño esté entre 10M y 20M

$ find -perm +1000 -type d       # busca los directorios que posean el permiso t

# find / -name core -exec rm -i "{}" ";"      #  busca  todos  los ficheros que se nombren core y los borra interactivamente. Los signos "" se utilizan para proteger de la interpretación del shell
```

Comando locate

El comando locate busca en una base de datos, actualizada periódicamente, todos los paths en la jerarquía de ficheros que contengan una cadena determinada. Para crear esta base de datos o actualizarla se debe invocar por root el comando updatedb (o locate -u).

En las distribuciones actuales esta tarea se hace de forma automática, aunque el usuario root normalmente tiene que ejecutarlo al menos una vez para crear la base de datos).

Ejemplo:

```
$ locate passwd
```

1.6 Comandos para filtrar ficheros

Comando sort

El comando sort ordena las líneas de un fichero mostrándolas por la salida estándar. De no especificarse un fichero toma la entrada estándar.

Sintaxis: sort [opciones] [fichero]

Algunas opciones:

-r : ordena al revés.

-f : trata las mayúsculas y minúsculas por igual.

-g : ordena de forma numérica, de modo que no es necesario que los números se rellenen con ceros por la izquierda.

Ejemplo: **$ sort -f /etc/passwd**

Como ejercicio, cread un archivo llamado lista-desordenada con el vi y meter dentro 5 nombres desordenados. Comprobad como con la orden sort lista-desordenada > lista-ordenada creamos un fichero llamado lista-ordenada y que contiene la lista ordenada.

Comando uniq

El comando uniq elimina las líneas repetidas de un fichero ordenado, imprimiéndolo por la salida estándar o en otro fichero argumento. De no especificarse un fichero toma la entrada estándar.

Sintaxis: uniq [opciones] [fichero] [salida]

Algunas opciones:

-c : utiliza como prefijo en cada línea su número de ocurrencias.

-d : solo imprime las líneas duplicadas.

Ejemplo: **$ uniq -d lista.txt**

Comandos tail y head

Los comandos tail y head muestran respectivamente el final y el comienzo (10 líneas por defecto) de uno o varios ficheros. De no especificarse al menos un fichero toman la entrada estándar.

Sintaxis:

tail [opciones] [ficheros]

head [opciones] [ficheros]

Algunas opciones:

- -f para el caso de tail se ejecuta de forma sostenida o sea se continúa visualizando el final del fichero hasta que se interrumpa el proceso (Ctrl-c).
- -q no coloca los encabezamiento con el nombre de los ficheros cuando se indican varios (quiet).
- -<n> imprime las n últimas (primeras) líneas en lugar de las diez establecidas por defecto.

Ejemplos:

```
# tail -f /var/log/messages
# tail -20 /var/log/secure
# head -50 /var/spool/mail/pepe
# head -2 -q /etc/*.conf
```

Comando wc

El comando wc imprime el número de líneas, palabras y bytes de uno o varios ficheros. Si son varios ficheros hace también un resumen de los totales. De no especificarse un fichero toma la entrada estándar.

Sintaxis: wc [opciones] [ficheros]

Algunas opciones:

- -l sólo cuenta líneas.
- -c sólo cuenta bytes.
- -w sólo cuenta palabras.

Ejemplos:

```
$ wc -l /etc/passwd
$ wc -w /doc/dicciorario.txt
```

Comando cut

El comando cut nos permite cortar una línea de texto, para obtener un subconjunto en lugar de la línea completa. Podemos cortar por número de caracteres, por campos, etc.

Sintaxis: cut [opciones] [ficheros]

Algunas opciones:

- -c N-M corta desde el carácter número N hasta el carácter número M.
- -c N- corta desde el carácter número N hasta el final
- -c –N corta desde el principio hasta el carácter número N
- -c N,M corta el carácter número N y el carácter número M
- -d":" –f1 separa la línea en campos divididos por el carácter : y nos muestra sólo el primer campo
- -d"-" –f3 separa la linea en campos divididos por el carácter – y nos muestra sólo el 3 campo.

Ejemplos:

```
$ cut –c 3-9 /etc/passwd
$ cut –c d":" –f4 /etc/passwd
```

Comando sed

Sed realmente es un mini lenguaje de programación capaz de hacer sustituciones realmente complejas de cadenas de texto.

Sintaxis: sed –n –e"comandos" –f archivo [opciones] [ficheros]

Algunas opciones:

- -n no produce la salida estándar.
- -e"comandos" indica que comandos va a ejecutar sed
- -f si queremos que los comandos se tomen de un archivo

Los comandos de sed, se usan de la siguiente manera:

La estructura general es: **[inicio[,fin]] función [argumentos]**

Donde inicio y fin hacen referencia a las líneas que se van a tratar, función hace referencia a que acción vamos a realizar, y argumentos indica como se va a aplicar la acción.

Si queremos poner varios comandos en varias líneas, al final de cada línea se debe añadir la contrabarra (\) antes de pulsar intro para que nos deje seguir escribiendo.

Los comandos más habituales en sed son:

a\ Añade al final de la línea

c\ Cambia el contenido del patrón de texto

d Borra las líneas indicadas

s Realiza sustituciones

g Realiza sustituciones generales

i\ Inserta líneas

p Imprime líneas

q Abandona sed

r arch Lee el archivo y lo añade a la salida

w arch Copia las líneas en un archivo

= Da el número de línea

/^tex/ Realiza un grep simple buscando tex

Ejemplos:

$sed "3d" canciones Produce una salida donde se borra la 3ª línea de canciones

$sed "a\Hola" canciones Produce una salida donde después de cada línea se escribe hola en una línea nueva.

$sed "i\Hola" canciones Produce una salida donde antes de cada línea se escribe Hola en una nueva línea.

$sed –n 2p canciones Solo nos muestra la 2ª línea de canciones.

$sed –n 3,9p canciones Solo nos muestra las líneas 3 a la 9 de canciones

$sed –n '3,$p' canciones Solo nos muestra de la línea 3 al final. (fijaos como hay que usar aquí comillas simples para evitar que $p crea que es una variable).

$sed –n /^ho/ canciones Solo nos muestra las líneas que comiencen por ho

$sed –n '/^t/,$p' canciones Nos muestra desde la primera línea que comience por t hasta la ultima línea.

$sed 's/antes/después/g' canciones Sustituye todos los "antes" por "después" en el fichero canciones.

$sed 's/antes/después/' canciones Sustituye el primer antes por después en cada línea….

$sed '1,6s/antes/después/g' canciones Sustituye todos los antes por despues, pero solo en las líneas 1 a 6.

$sed '/^en/s/jose/pepe/g' canciones Sustituye todos los jose por pepe, pero solo en aquellas líneas donde se encuentre la cadena "en"

$sed 'y/[123]/[456]/' canciones Sustituye los 1 por 4, los 2 por 5, y los 3 por 6

$sed 's/juana/#&/' profesores Sustituye Juana, por la #Juana (& indicael texto buscado) pero poniendole un # delante, es decir, que la comenta.

$sed –n '/pirata/p' poema Solo nos muestra las líneas donde aparezca pirata en poema.

Una forma habitual de trabajar con sed es utilizando expresiones regulares. Este tipo de expresiones regulares no solo se usan con sed, sino con bastantes comandos de Linux, es por ello que vamos a detenernos para verlas un momento.
Estas expresiones regulares son bastante parecidas a los comodines que ya conocemos, como *

Carácter	Descripción
^	Apunta al comienzo de la línea
$	Apunta al final de la línea
.	Apunta a un único carácter
*	Apunta a cero o más ocurrencias del carácter previo
[]	Apunta a todos los caracteres entre los corchetes

Expresión regular	Descripción
/./	Apuntará a cualquier línea que contenga al menos un carácter
/../	Apuntará a cualquier línea que contenga al menos dos caracteres
/^#/	Apuntará a cualquier línea que comience con un '#'
/^$/	Apuntará a cualquier línea en blanco
/}^/	Apuntará a toda línea que termine con un '}' (sin espacios)
/} *^/	Apuntará a toda línea que termine con un '}' con cero o más espacios
/[abc]/	Apuntará a toda línea que contenga una 'a', 'b', o 'c' minúscula
/^[abc]/	Apuntará a cualquier línea que empiece con 'a', 'b', o 'c'

La mejor forma de habituarse al uso de estas expresiones regulares es usándolas. Intentemos realizar estos ejercicios:
1) Obtener por pantalla todas las líneas de un fichero que comiencen con el carácter a
2) Sustituir en un fichero el 1º carácter de cada línea por tres guiones
3) Sustituir en un fichero todas las vocales por la vocal U
4) Sacar por pantalla el contenido de un fichero, pero borrando todas las vocales.

Comando awk

Awk es un verdadero lenguaje de programación, especializado en análisis y procesamiento de patrones de texto. Viene a ser como un grep mezclado con sed.

Sintaxis: awk [-f archivo_programa] [-Fc] ['programa'] [variable=valor ...] [archivo]

donde:

- archivo_programa : especifica el archivo fuente del programa a aplicar a archivo.
- c : especifica el carácter delimitador de campos. Por defecto se considera el espacio en blanco.
- programa: especifica el conjunto de patrones e instrucciones a ejecutar con archivo. La diferencia con archivo_programa, es que en este caso hay que especificar los patrones e instrucciones en la línea de comandos. Para evitar interferencias con la shell deben ir encerradas entre comillas simples (').
- variable=valor : se utiliza para establecer los valores que tomarán las variables que utilice el programa.
- archivo : archivo que será procesado por awk. Si se especifica "-" se interpretará como la entrada estándar.

Como ya hemos indicado, awk es un verdadero lenguaje de programación, es decir, que cuenta con sus instrucciones secuenciales, alternativas, reiterativas, etc.

Awk se basa en el uso de variables, donde $0 se refiere a una línea completa, $1 al primer campo, $2 al segundo campo, etc.

Asi, por ejemplo, la siguiente instrucción nos muestro los nombres de usuarios del fichero passwd, que como ya sabemos cuenta con campos separados por el símbolo dos puntos.

```
awk –F:  `{ print $1 } ’  /etc/passwd
```

En el ejemplo anterior, podriamos haber creado un fichero, por ejemplo primercampo.awk que tuviera como contenido { print $1 } y entonces la linea podria haber quedado como:

```
Awk –F:  -f primercampo.awk  /etc/passwd
```

Si no indicamos el campo delimitador se usa por defecto el espacio en blanco. Cread un fichero texto.txt de texto de varias líneas, con varias palabras en cada linea y ejecutad la siguiente instrucción.

```
Cat texto.txt | awk '{ print "1ª y 2ª palabras: " , $1, $2,  " y fin" }'
```

Awk dispone de varias palabras reservadas que podemos usar para mostrar información sobre la entrada. Asi tenemos por ejemplo:

- FS (Field separator): contiene el carácter que indica a awk en qué punto del registro acaba un campo y empieza el siguiente. Por omisión es un espacio. Se puede indicar un carácter alternativo mediante una instrucción de asignación como FS = "/". Si se deja vacío, cada lectura se realiza dejando un carácter en cada campo.
- NF (Number of fields): contiene el número total de campos que contiene el registro que se está leyendo en cada momento.
- RS (Record separator): contiene el carácter que indica a awk en qué punto del archivo acaba un registro y empieza el siguiente. Es "\n" por omisión.
- NR (Number of record): contiene el número de orden del registro que se está procesando en cada momento.
- OFS (Output FS): contiene el separador de campos para la salida generada por awk. La instrucción print inserta en la salida un carácter de separación cada vez que aparece una coma en el código. Por ejemplo:

Asi, una línea de orden como la siguiente:

```
awk '{print NR, $0}'
```

nos muestra todas las líneas del fichero passwd ($0) pero con el numero de linea (NR) delante

```
awk '{print "Linea : ", NR, " = ", $0}'
```

awk también permite realizar condiciones sobre las líneas, por ejemplo:

```
cat /etc/passwd | awk –F: '$1 == "joancadi" {print "Esta linea tiene en su primer campo la palabra joancadi ->", $0, " y es la linea número ", NR}',
```

Vemos como al poner '$1 == "joancadi" en realidad estamos preguntado si el primer campo es la cadena joancadi, el {print solo se ejecutará si esa condición es verdadera.

Usemos el awk realmente como un lenguaje de programación…. Cread un fichero con nombre sumatorio.awk que tenga las siguientes líneas:

```
BEGIN { total = 0 }
      { total += $1 }
END { print "El total es", total }
```

Cread ahora un fichero con varios numero en varias filas, con nombre numeros.txt, y ejecutad:

```
cat numeros.txt | awk –f sumatorio.awk
```

Podéis comprobar en este ejemplo como podemos definir variables (total), realizar operaciones aritméticas sobre ellas (+=), realizar bucles, etc. El bucle BEGIN – END se ejecuta por cada línea del fichero que trate el awk.

También tenemos la estructura If – Else….

```
{ print "Procesando linea :", NR;
 if( $1 > 10 )
   { print $1 "es mayor que 10" }
 else
   { print $1 "es menor que 10" }
```

```
}
```

No podía faltar el for, claro….

```
{  for( i = 0; i < 10; i ++ )
 {
  j = 2 * i;   print i, j;
 }
}
```

Sin olvidar el Do – While, claro esta…

```
BEGIN { i = 0; do {print i; i ++} while ( i < 10 ) }
```

Tenemos funciones en awk, muchas funciones:

```
awk 'BEGIN { print "El seno de 30 grados es", sin( 3.141592653589 /
6 ); }' /dev/null
```

Algunas de las funciones más interesantes de awk, son:

length( )	Longitud del parámetro en bytes	rand( )	Número al azar entre cero y uno
srand( )	Inicia la semilla de generación al azar	int( )	Devuelve el parámetro convertido en un entero
substr(s,m,n)	Devuelve la subcadena de s en m con longitud N	index(s,t)	Posición de s donde aparece t, o cero si no está.
match(s,r)	Posición de s en que se cumple la expresión r.	split(s,a,fs)	Devuelve s en elementos separados por fs
sub(r,t,s)	Cambia en s la cadena t por r. Es $0 por omisión	gsub(r,t,s)	Igual, pero cambia todas las t en la cadena
sprintf(f,e,e…)	Devuelve cadena de formato f con las "e"	system(cmd)	Ejecuta cmd y devuelve el código de retorno
tolower(s)	Devuelve s convertida en minúsculas	toupper(s)	Devuelve s convertida en mayúsculas
getline Fuerza una lectura de fichero			

1.7 Comandos para compactar y agrupar ficheros

Comandos gzip y gunzip

Los comandos gzip y gunzip permiten compactar y descompactar (comprimir y descomprimir) respectivamente uno o varios ficheros.

Sintaxis:

gzip [opciones] <ficheros/directorio>

gunzip [opciones] <ficheros/directorio>

Algunas opciones:

- -r : dado un directorio comprime todos los ficheros presentes en él recursivamente.
- -1 a -9 : especifica el grado de la compresión (-1 menor y más rápida -9 mayor y más lenta).
- -S <sufijo> : permite especificar un sufijo o extensión para el fichero resultado (por defecto es gz).

Ejemplos:

```
$ gzip -9 * # Comprime todos los ficheros del directorio actual (su extensión cambia a .gz)
$ gunzip big-file.gz # descomprime el fichero big-file.gz
```

Todo lo compactado con gzip se puede descompactar con el Winzip de los sistemas Windows. También existen los pares de comandos zip y unzip (compatibles en ambos sentidos con Winzip), y compress y uncompress.

Comando tar

El comando tar (Tape Archiver) es una herramienta para agrupar varios ficheros aislados o el contenido de un directorio en otro fichero o dispositivo especial. El comando tar no comprime o compacta absolutamente nada, se limita a agrupar varios ficheros en uno solo, sin comprimirlos. Existe una opción (-z) que automáticamente ejecuta un gzip o un gunzip sobre el fichero agrupado.

Sintaxis: tar [opciones] <fuentes>

Algunas opciones:

- -c permite crear (tarear), es decir, agrupar ficheros en uno solo.
- -x permite extraer (destarear), es decir, desagrupar ficheros.
- -v activa el modo debug, donde se ven todos los mensajes.
- -f <fichero> agrupa o desagrupa en o hacia un fichero y no utilizando la salida o entrada estándar como es por defecto. (Ojo, esta opción la usaremos siempre).
- -z compacta o descompacta el fichero resultante una vez agrupado o desagrupado con gzip y gunzip respectivamente.
- -t lista el contenido de un fichero resultado de un agrupamiento.
- -M agrupa en volúmenes.

El comando tar conserva la estructura jerárquica original de lo agrupado excluyendo el carácter / que representa a la raíz. Algunas opciones se pueden emplear sin el carácter -, siempre y cuando no haya ambigüedades entre ellas o con los argumentos.

Ejemplos:

```
$ tar cvzf grande * # crea un fichero grande donde estarán agrupados todos los ficheros del directorio actual y que además estará comprimido.

$ tar xvzf grande # desagrupa en el directorio actual el fichero grande y ademas lo descomprime.

# tar cf uconf.tar /etc/passwd /etc/shadow /etc/groups # agrupa en el fichero uconf.tar los ficheros passwd shadow y groups

# tar tf uconf.tar # muestra los ficheros agrupados en uconf.tar

# tar xf uconf.tar # desagrupa el fichero uconf.tar

$ tar cMf /dev/fd0 /tmp/etc.tgz

$ tar xMf /dev/fd0
```

1.8 Comandos para desconectarse del sistema

Comando exit

El comando exit permite terminar el shell actual. Si se tiene un único shell es equivalente a desconectarse del sistema, pero si se está en un subshell sólo se terminará este, retornando al shell anterior.

Comando logout

El comando logout permite desconectarse del sistema a partir de un login shell (primer shell que se ejecuta al establecer la conexión).

La secuencia de caracteres Ctrl-d permite terminar el shell actual. Si es un login shell equivaldrá a hacer logout y si no, a hacer exit.

Comando halt, reboot, poweroff

Estos comandos nos permiten suspender, reiniciar o apagar el sistema.

Comando shutdown

Este comando nos permite "echar abajo" el sistema. Algunas opciones interesantes:

-c cancela un shutdown que esta en proceso de ejecución

-f Reinicia más rapido, ya que no controla la integridad de los sistemas de archivos

-h Cuando el sistema se apaga, apaga el ordenador (o lo intenta)

-r Cuando el sistema se apaga, intenta reiniciarlo

Despues de estas opciones, se le indica cuando queremos apagar el sistema:

now lo apaga inmediatamente, ahora.

20:00 lo apaga a las 8 de la tarde

+10 lo apaga en 10 minutos

```
# shutdown -h +4
```

1.9 Comandos para administrar usuarios y grupos

Comando useradd

El comando useradd permite añadir nuevos usuarios al sistema, además de establecer la información por defecto de los nuevos usuarios que se añadan.

Sintaxis: useradd [opciones] [login]

Ejemplos:

```
# useradd pepe # crea el usuario pepe con todas las opciones por defecto.
# useradd –D # muestra las opciones por defecto que se aplicarán a los usuarios nuevos.
```

Comando adduser

La orden useradd no se suele usar, ya que no configura correctamente las cuentas de usuario. En su lugar en los sistemas Linux actuales se ha creado un script denominado adduser que nos permite crear usuarios de una forma más amistosa.

Sintaxis: adduser [opciones] [login]

Adduser usuario grupo

Ejemplos:

```
# adduser pepe # crea un usuario pepe, preguntándonos por su contraseña, comentarios, etc.
# adduser # crea un usuario, preguntandonos por su nombre, su contraseña, comentarios, etc.
# adduser pepe cdrom # añade al usuario pepe, que debe existir ya, al grupo cdrom.
```

Comando addgroup

Nos permite crear grupos.

Sintaxis: addgroup grupo

Ejemplo:

```
# addgroup alumnos
```

Comando userdel

El comando userdel permite eliminar definitivamente un usuario del sistema.

Sintaxis: userdel [opciones] <login>

Ejemplo:

```
# userdel -r pepe     # elimina al usuario pepe y borra su directorio base.
                      Por defecto el directorio base se mantiene
```

Comando groupdel

El comando groupdel permite eliminar definitivamente un grupo del sistema.

Sintaxis: groupdel [opciones] grupo

Ejemplo:

```
# groupdel alumnos
```

Comando passwd

El comando passwd permite cambiar el password de un usuario. También puede bloquear, desbloquear y deshabilitar una cuenta. Si se invoca sin argumentos se asume el usuario actual.

Sintaxis: passwd [opciones] [login]

Ejemplos:

```
# passwd pepe        # coloca una contraseña para pepe

# passwd -d pepe     # deshabilita la cuenta del usuario pepe eliminando
su password

# passwd -l pepe   # bloquea la cuenta del usuario pepe poniendo un
signo ! delante de su password en el fichero /etc/shadow

# passwd -u pepe     # desbloquea la cuenta del usuario pepe
```

Comando usermod

El comando usermod se emplea para modificar algunas propiedades de los usuarios como: el login, el directorio base, el shell que se inicia al conectarse, los grupos a los que pertenece, la fecha de expiración de la cuenta, etc. También bloquea y desbloquea una cuenta.

Sintaxis: usermod [opciones] <login>

Ejemplos:

```
# usermod -s /bin/csh pepe          # coloca el shell csh para el
usuario pepe

# usermod -G users,disk pepe        # señala como grupos secundarios de
pepe a users y disk

# usermod -e 2001-10-20 pepe    # indica que la cuenta de pepe expirará
el 20 de octubre del 2001
```

Comando chfn

El comando chfn permite cambiar la información de contacto de un usuario. Esta incluye aspectos como: el nombre completo, la oficina de trabajo y los teléfonos. Se almacena en el fichero de usuarios /etc/passwd.

Sintaxis: chfn [opciones] [login]

Ejemplo:

```
# chfn pepe
```

Comando gpasswd

El comando gpasswd permite administrar los grupos. Se puede utilizar para añadir y eliminar usuarios, señalar un administrador e indicar un password de grupo.

Sintaxis: gpasswd [opciones] <grupo>

Ejemplos:

```
# gpasswd –A pepe admin        # señala como administrador del grupo admin al usuario pepe
$ gpasswd admin               # cambia el passwd del grupo admin
$ gpasswd –a joe admin        # añade el usuario joe al grupo admin
```

Comando su

El comando su permite ejecutar un shell (u otro comando) cambiando los identificadores del grupo y del usuario actual. Si se le pasa - como primer argumento ejecuta el shell como un login shell, o sea se creará un proceso de login tal y como ocurre naturalmente cuando un usuario se conecta al sistema. Si no se especifica el login del usuario se asume root.

Sintaxis: su [opciones] [login]

Ejemplos:

```
$ su -    # el guion permite abrir una shell para el usuario.
# su pepe
$ su --c "cat /etc/shadow"  # ejecuta un comando con los privilegios de root. (básicamente, realiza lo mismo que el comando sudo).
```

Comando sudo

En algunos sistemas, como puede ser Ubuntu, existe una orden denominada sudo, que viene a ser un su –c (es decir, permite ejecutar una orden como el usuario root).

Sintaxis: sudo

Ejemplos:

```
$ sudo gedit /etc/shadow # ejecuta gedit /etc/shadow como si lo ejecutará el root.
$ sudo su # Nos transforma en el root hasta que escribamos exit.
```

```
$ sudo su - # Nos transforma en el root, y abre un nuevo shell para el root.
```

Comando id

El comando id, imprime dado un usuario, sus identificadores de usuario y de grupo principal (gid y uid) así como del resto de los grupos a los que pertenece. Sin argumentos se asume el usuario actual.

Sintaxis: id [opciones] [login]

Ejemplo:

```
# id pepe        uid=502(pepe) gid=502(pepe) groups=502(pepe),100(users)
```

Comando chage

El comando chage nos permita gestionar la información sobre la caducidad de las contraseñas.

```
# chage Joancadi
```

Comando last

El comando last nos indica las últimas conexiones de usuario que han existido en el sistema.

```
# last
```

1.10 Comandos de redes

Comando tracepath

El comando tracepath nos permite trazar una ruta entre nuestro host y el host que le indiquemos, mostrándonos todos los pasos intermedios que va recorriendo.

```
# tracepath www.elpais.es
```

Comando write

El comando write se utiliza para enviar un mensaje a un usuario conectado al sistema. Por defecto el mensaje se envía a la última terminal donde se haya conectado el usuario. Los usuarios pueden deshabilitar la posibilidad de recibir mensajes utilizando el comando mesg.

Sintaxis: write <usuario> [terminal]

Ejemplos:

```
$ mesg  y # habilita la posibilidad de recibir mensajes
$ write pepe tty3
        Hola que tal   Ctrl-d
```

Si el usuario pepe está conectado a través de la terminal tty3 y tiene habilitada la posibilidad de recibir mensajes se mostrará en esta terminal:

Message from coco@deltha on tty2 at 16:35 ... (o el usuario que sea)

Hola que tal EOF

```
$ mesg      # muestra si está habilitada o no la posibilidad de recibir mensajes
$ mesg n    # deshabilita la posibilidad de recibir mensajes
```

Realmente este comando es interesante cuando al menos uno de los usuarios está conectado remotamente, a través de terminales.

Comando wall

El comando wall se emplea para enviar un mensaje a todos los usuarios conectados en el sistema que tengan habilitada la posibilidad de recibirlos (mesg y).

Ejemplos:

```
# wall
        Voy a apagar la máquina a
        las 2:00 PM
        El administrador
        Ctrl-d
```

Comando netstat

El comando netstat nos muestra información sobre las conexiones de red, las tablas de enrutamiento, las estadísticas de los interfaces de red, etc. Con este comando podemos conocer perfectamente que esta ocurriendo en nuestra red y como se comporta nuestro equipo en la misma.

```
# netstat –i      # estadísticas de nuestras conexiones de red.
# netstat –l –t   # aplicaciones que tenemos instaladas y están "escuchando" en la red usando un puerto tcp.
# netstat –l –u   # aplicaciones que tenemos instaladas y están "escuchando" en la red usando un puerto udp.
```

Comando route

El comando route gestiona la tabla de enrutamientos del sistema de red. Mediante estas tablas de enrutamiento podemos realizar poderosas acciones sobre la red, como manejar varias puertas de enlace, trabajar en distintas redes, etc.

```
# route
```

Comando nslookup

El comando nslookup nos permite interrogar a un servidor DNS para encontrar información sobre un host de la red.

```
# nslookup www.elpais.es
```

Comando tty

El comando tty imprime el dispositivo de carácter asociado a la terminal en la que se está trabajando.

Ejemplos:

```
$ tty                /dev/tty2
$ tty                /dev/pts/0
```

Comando who

El comando who muestra los usuarios conectados al sistema ya sea local o remotamente.

Sintaxis: who [opciones] [fichero] [am i]

Sin argumentos who muestra los logins de los usuarios conectados, por que terminal lo han hecho y en que fecha y hora.

Algunas opciones:

-H : imprime un encabezamiento para las columnas.

-w : indica si está activada o no la posibilidad de recibir mensajes por parte de cada usuario conectado (+ indica que si, - que no y ?, desconocido).

-i : imprime además para cada usuario conectado que tiempo lleva sin interactuar con el sistema (idle time). Si lleva menos de un minuto pone un punto y si es más de 24 horas la cadena ``old''.

-q : sólo muestra los logins de los usuarios conectados y la cantidad total de ellos.

Ejemplos:

```
$ who
$ who am I
```

Comando w

El comando w muestra también los usuarios conectados al sistema además de lo que están haciendo (proceso que ejecutan en ese momento) y otras informaciones.

Sintaxis: w [opciones] [usuario]

Sin argumentos este comando muestra una primera línea con: la hora en que arrancó el sistema, cuanto tiempo lleva funcionando, cuantos usuarios hay conectados (sin incluir las sesiones gráficas) y tres porcientos de carga de la CPU: durante el último, los 5 y los 15 minutos anteriores. A continuación se muestra una tabla cuyas columnas representan: el login de cada usuario conectado, por que terminal lo ha hecho, desde que host, a que hora, el idle time exacto, la cantidad de segundos de CPU que han empleado todos los procesos que ha ejecutado ese usuario (JCPU) y el tiempo (PCPU) y nombre del comando que ejecuta actualmente.

Ejemplos:

```
$ w
$ w pepe
```

Comando finger

El comando finger permite buscar y mostrar información asociada a los usuarios del sistema de acuerdo a sus nombres, apellidos o login. La información que muestra finger para cada usuario es:

- El login.
- El nombre y los apellidos.
- El directorio base.
- El shell.
- La oficina y el teléfono.
- El teléfono de la casa.
- La lista de terminales a través de las que está conectado con la fecha, tiempo sin interactuar (iddle time) y si está deshabilitada la posibilidad de recibir mensajes.
- La fecha y hora del último nuevo mensaje electrónico recibido y desde cuando no accede al buzón.
- El contenido del fichero .plan en el directorio base.

Ejemplos:

```
$ finger alina
$ finger castellanos
```

Comando ping

El comando ping permite enviar paquetes ICMP (Internet Control Message Protocol) del tipo ECHO_REQUEST a otra computadora, con el objetivo de saber si esta es alcanzable a través de la red. Además muestra un resumen estadístico acerca del tanto por ciento de paquetes perdidos y las velocidades de transmisión. Este comando se ejecuta por defecto sostenidamente por lo que para interrumpirlo se debe hacer Ctrl-c.

Sintaxis: ping [opciones] <máquina>

Algunas opciones:

-c <n> : envía n paquetes exactamente.

-i <n> : espera n segundos entre los envíos.

-s <n> : envía paquetes de n bytes. Se le suman los 8 bytes del header del paquete ICMP.

-q : sólo despliega el sumario final.

Ejemplos:

```
$ ping www.iesromerovargas.net
```

Comando ifconfig

El comando ifconfig permite configurar por parte de root las interfaces de red. Los usuarios distintos de root lo pueden invocar también con fines informativos. Para ello deben especificar el camino completo (/sbin/ifconfig) pues por defecto este no está en el path de los usuarios comunes. Sin argumento ifconfig despliega información acerca de la configuración y funcionamiento actuales de las interfaces de red activas.

Ejemplos:

```
# ifconfig
# ifconfig –s     # Muestra un resumen de los interfaces
# ifconfig eth0   # Muestra información solo sobre el interface eth0
```

Ifconfig también puede usarse para activar y desactivar los interfaces de red.

```
# ifconfig eth0 up
# ifconfig lo down
```

Ifconfig puede usarse también para configurar los interfaces de red, aunque suele ser mejor configurarlos editando el fichero /etc/network/interfaces.

```
# ifconfig eth0 hw ether 00:30:CA:52:0A:F0    # Cambia la dirección MAC
del interface
# ifconfig eth0 address 192.168.100.2
# ifconfig wlan0 netmask 255.255.255.0
```

Comando hostname y dnsdomainname

El comando hostname nos devuelve el nombre de equipo (host) y nos permite cambiarlo. El comando dnsdomainname hace lo mismo pero para un nombre FQDN (Fully Qualified Domain Name).

```
# hostname

# dnsdomainname

# hostname pepito       # Hace que el nombre de equipo pase a ser pepito.
```

Hay que tener mucho cuidado al cambiar el nombre de equipo, ya que es muy posible que tengamos programas y comandos que están preparados para trabajar y encontrar nuestro equipo con el nombre actual (sudo por ejemplo). Nos podemos encontrar que estos programas funcionen mal si cambiamos el nombre del equipo. El nombre del equipo se almacena en el fichero /etc/host.conf.

1.11 Comandos varios

Comando echo

El comando echo muestra en su salida todo lo que se le pase como argumento.

Ejemplo:

```
$ echo Hola amigo...
```

Hola amigo...

Comando set

El comando set sirve para gestionar las variables y funciones del shell. Si se ejecuta set directamente sin parámetros, nos devuelve los nombres y valores de varias funciones del shell.

```
# set
```

En bash no es necesario asignar variables mediante set variable=valor, sino que se puede usar directamente variable=valor. Sin embargo, nos podemos encontrar trabajando con otros shells, como csh en los que es obligatorio usar el set para asignar valores a las variables.

Comando history

El comando history nos permite controlar el histórico de comandos que se van almacenando en el shell. El histórico se almacena por defecto en el fichero ~/.history lo que nos indica que existe un histórico diferenciado para cada usuario. Si ejecutamos el comando history nos devolverá la relación de comandos para ese usuario.

```
# history
```

Una posibilidad es enviar la salida de la orden history a un fichero, con lo que conseguimos tener una relación de los comandos que hemos ejecutado.

Comando uname

El comando uname nos da información sobre el sistema, como el nombre del kernel, su versión, el nombre de la maquina, etc.

```
# uname -a
```

Comando date

El comando date sirve para consultar y cambiar la fecha y hora del sistema.

```
# date
```

El comando date tiene bastantes opciones, y con el se pueden hacer bastantes cosas. Es recomendable echarle un vistazo a sus páginas de manual para entenderlos bien. Veamos algunos ejemplos.

```
$ date --date="2 days ago"          # nos da la fecha de hace dos días
$ date --date="3 months 1 day"      # nos da la fecha que será en 3 meses y un día.
$ date --date="25 Dec"              # nos da el dia que será el 25 de Diciembre del año actual
$ date "+%B %d"                     # nos devuelve el nombre del mes actual y el número de día del mes.
$ date --set="2006-6-20 11:59 AM"   # ajusta la fecha del sistema
$ date --set="+3 minutes"           # adelante la hora del sistema en 3 minutos.
```

Comando time

El comando time sirve para cronometrar cuanto tiempo tarda en ejecutarse un comando cualquiera, y que recursos consume. Puede ser una orden interesante para comprobar el rendimiento de los sistemas.

```
# time wc /etc/passwd
```

Comando uptime

El comando uptime nos indica las el "uptime" del sistema. En informática se conoce como uptime el tiempo que un equipo lleva "levantado", es decir cuanto tiempo lleva el sistema funcionando. En un servidor es muy importante que este uptime sea lo más elevado posible. Uptime nos devuelve cuanto tiempo lleva el equipo levantado, el número de usuarios que están conectados, y la media de las cargas de sistema para los últimos 1, 5, y 15 minutos.

```
# uptime
```

Comando top

El comando top nos muestra los procesos que se están ejecutando en nuestra maquina, y cuantos recursos están ocupando. Es una orden muy interesante por que nos permite ver como se encuentra

nuestro sistema de "cargado", que procesos son los que están consumiendo más recursos, etc. Si pulsamos h en la pantalla de top, veremos una pantalla de ayuda donde podemos ordenar la lista de procesos según su consumo de cpu, memoria, etc.

```
# top
```

Comando dmesg

Nos da una lista con todos los mensajes que se han producido durante el arranque del sistema.

```
# dmesg
```