UNIX/Linux

1. Características

- **Sistema multiusuario en tiempo compartido** → Pueden trabajar varios usuarios simultáneamente compartiendo CPU y recursos del sistema, y cada usuario puede ejecutar varias sesiones y procesos a la vez.
- **Reescrito en C** → Más transportable, pero más lento que la versión en ensamblador.
- **Interfase de usuario externa al núcleo del sistema operativo** → Puede reemplazarse por otras.
- **Enfocado al desarrollo de programas** → Se pueden crear programas complejos a partir de programas sencillos y primitivas del sistema.
- **Sistema de ficheros con estructura jerárquica en forma de árbol invertido de múltiples niveles** → Fácil mantenimiento.
- **Ficheros sin formato predeterminado** → Para el sistema son sólo secuencias de bytes.
- **Los ficheros en disco y los dispositivos E/S reciben el mismo tratamiento** → Las peculiaridades de cada dispositivo se encuentran en el núcleo del sistema.
- **Arquitectura transparente al usuario** → Facilidad de programación y transportabilidad.
- **Código abierto** → Posibilidad de depuración y mejora continuas.

2. Estructura

- **Intérprete de comandos (shell)** → Capa más externa del sistema, hace de interfase entre el usuario y el sistema. Es el proceso padre bajo el cual se crean los demás procesos del usuario.
- **Aplicaciones** → Intermedia entre el shell y el núcleo, está formada por el conjunto de programas que dan servicio al usuario y que realizan llamadas al núcleo.
- **Núcleo (kernel)** → Hace de administrador de recursos físicos y lógicos, y sirve de biblioteca de funciones básicas del sistema. Es monolítico, no existen niveles.
- **Hardware** → Capa más interna del sistema, gestionado exclusivamente por el núcleo.

3. Componentes

3.1. Intérprete de comandos (shell)

3.1.1. Funciones del shell

- Sustituye los valores de las variables shell por variables referenciadas.
- Genera nombres de fichero a partir de metacaracteres.
- Maneja la redirección de E/S y las tuberías (pipes).
- Realiza la sustitución de comandos.
- Verifica si el comando a ejecutar es interno o es un programa ejecutable externo, en este caso busca el programa.

3.1.2. Tipos y configuración del shell

- Los tipos de shell más importantes son los siguientes:
  - **Bourne Shell (sh)** → El primero que se desarrolló.
  - **California Shell (csh)** → No estándar e incompatible con Bourne Shell.
  - **Korn Shell (ksh)** → Compatible con Bourne Shell y mezcla de los dos anteriores.
  - **Bourne Again Shell (bash)** → Utilizado en Linux, mejora de Bourne Shell.
- Se pueden iniciar varias sesiones pulsando **Alt+F1**, **Alt+F2**... A cada sesión se le asigna un terminal diferente, por tanto es posible abrir varias sesiones con un mismo usuario.
- En **bash** existen los siguientes ficheros para su configuración:
  - **etc/passwd** → Se lee en el login del usuario y contiene información del tipo de shell a cargar y el directorio de arranque del usuario.
  - **etc/profile** → Fichero que establece las condiciones generales de configuración del shell para todos los usuarios. Se ejecuta en cada inicio de sesión.
  - **~/.bash_profile** → Está en el directorio de arranque de cada usuario y establece la configuración particular del shell. Se ejecuta en cada inicio de sesión.
  - **etc/bashrc** → Fichero que establece las condiciones generales de configuración del shell para todos los usuarios. Se ejecuta cada vez que se llama a bash.
  - **~/.bashrc** → Está en el directorio de arranque de cada usuario y establece la configuración particular del shell. Se ejecuta cada vez que se llama a bash.

3.1.3. Ejecución de comandos

- El mecanismo clásico de ejecución de los comandos externos por parte del shell es el siguiente:
  - Llamada al sistema **fork** para crear un proceso hijo paralelo (capacidad multitarea) que hereda sus ficheros estándares y el tratamiento de los diferentes sucesos.
  - En el nuevo proceso se realiza una llamada al sistema **exec** que reemplaza el código del proceso por el del comando a ejecutar.
  - Si el proceso hijo está en primer plano, el shell espera a la finalización de la ejecución del comando con la llamada al sistema **wait**. Si el proceso hijo está en segundo plano, el shell queda libre para esperar nuevos comandos.
- Los comandos internos del shell son subrutinas de éste, y por tanto no se crea ningún proceso hijo para ejecutarlas.
- Formas de ejecutar comandos:
  - **comando &** → Ejecuta el comando en segundo plano y el shell nos devuelve el control.
  - **comando1;comando2** → Ejecuta primero comando1 y después comando2.
  - **(comando1;comando2)** → Ejecuta en una subshell hija primero comando1 y después comando2, haciendo un grupo único con los comandos.
  - **comando1|comando2** → La salida de comando1 se comunica con la entrada de comando2 a través de una tubería (pipe).
  - **comando1 `comando2`** → La salida de comando2 se utiliza como argumento de comando1.

  - **comando1&&comando2** → Si comando1 acaba bien, se ejecutará comando2.
  - **comando1||comando2** → Si comando1 acaba mal, se ejecutará comando2.
- **alias nombre_alias = "comando [parámetros]"** → Cambia la forma de invocar un comando con unos determinados parámetros por un nombre. Valido hasta finalizar la sesión o hasta ejecutar **unalias nombre_alias.**

3.1.4. Variables shell

- Son zonas de memoria a las que se asigna un nombre, por convenio en mayúsculas, que pueden almacenar cualquier valor numérico o cualquier cadena de caracteres.
- Para asignar un valor a una variable → VARIABLE=valor.
- Para referirse al valor de una variable → $VARIABLE (sustitución de variables por el shell).
- No se declaran tipos de variable, es válido A=23 y A=palabra, pero al referirse a su contenido se puede distinguir entre un valor numérico ($A) o alfanumérico ("$A").
- Para asignar un valor a una variable mediante una operación matemática, es necesario incluir la operación dentro de un doble paréntesis → IMPORTE=$(($PRECIO*1.16)).
- Por defecto, al asignar un valor a una variable, ésta queda definida dentro del área local de datos del shell y por tanto no puede ser accedida por ninguno de sus procesos hijos. Para que esto sea posible la variable debe definirse en el área de memoria del entorno con el comando **export**.
- Existen unas variables con un valor predefinido para cada usuario antes del arranque del shell, y algunas de ellas son las siguientes:
  - **HOME** → Directorio de arranque.
  - **PATH** → Directorios de búsqueda de comandos y programas ejecutables.
  - **PS1** → Prompt del primer shell.
  - **PS2** → Prompt del segundo shell.
  - **TERM** → Tipo de terminal utilizado.
  - **TMOUT** → Segundos de espera de entrada de comandos antes de fin de sesión.
- Una vez arrancado, el shell se encarga de actualizar los valores para las siguientes variables:
  - **#** → Número de parámetros pasados en un comando.
  - **?** → Resultado de la ejecución de un comando (0 - OK, cualquier otro - error).
  - ***** → Contiene todos los parámetros pasados en un comando separados por espacios.
  - **$** → Número de identificación de proceso del shell.
  - **!** → Número de identificación del último proceso ejecutado en segundo plano.

3.1.5. Redirección de entrada y salida

- Un descriptor de fichero es un número que identifica a un fichero en los programas.
- Al iniciar un shell, el núcleo abre tres ficheros a disposición de los programas:
  - **fichero estándar de entrada** → Su descriptor de fichero es el 0 y es donde los programas leen su entrada. Por defecto es el teclado.
  - **fichero estándar de salida** → Su descriptor de fichero es el 1 y es donde los programas escriben su salida. Por defecto es el terminal.

  - **fichero estándar de errores** → Su descriptor de fichero es el 2 y es donde los programas escriben sus errores. Por defecto es el terminal.
- Redirección de entrada → Cualquier comando que lea por defecto la entrada estándar puede ser avisado para que tome dicha entrada de otro fichero.
- Redirección de salida → Cualquier comando que escriba por defecto en la salida estándar puede ser avisado para que escriba en otro fichero. En este caso primero se crea el fichero de salida y después se ejecuta el comando.
- Redirección de errores → Igual que la redirección de salida. Para enviar errores a ningún lugar se emplea el fichero **/dev/null**.
- El intérprete de comandos permite abrir archivos usando el comando propio exec con un descriptor de archivo arbitrario *n*. Estos descriptores pueden cerrarse empleando *n*<*&*-* y *n*>*&*- respectivamente para ficheros de entrada y de salida.

```
exec 3<entrada 4>salida # abrir archivos
exec 3<&- 4>&- # cerrar archivos
```

3.1.6. Filtros y tuberías (pipes)

- Un filtro es un proceso que lee su entrada en la entrada estándar y escribe su salida en la salida estándar. Por tanto, los filtros son redireccionables tanto en la entrada como en la salida.
- Una tubería es la construcción que realiza el shell para enlazar la salida de un comando con la entrada de otro. Todo ello se hace a través de la entrada estándar y la salida estándar, es decir, que los comandos implicados deben hacer uso de estos ficheros estándar. El shell ejecuta cada comando por orden en un proceso separado, esperando al resultado final del último comando.

3.1.7. Metacaracteres, caracteres especiales y entrecomillado

- Los metacaracteres son una serie de caracteres interpretados por el shell, que se encarga de transformarlos en nombres absolutos de ficheros. Son los siguientes:
  - **?** → Sustituye a un único carácter cualquiera, excepto el primer punto.
  - ***** → Sustituye a cero o varios caracteres cualesquiera, excepto el primer punto.
  - **[ ]** → Sustituye a los caracteres incluidos dentro de los corchetes o incluidos en un rango de caracteres si están separados por "-".
  - **[^]** → Sustituye a los caracteres no incluidos dentro de los corchetes o no incluidos en un rango de caracteres si están separados por "-".
  - **{ }** → Sustituye a la lista de cadenas de caracteres incluidos dentro de las llaves separadas por ",".
  - ~ (Alt+126) → Sustituye al directorio de arranque del usuario actual.
  - **~nombre_usuario** → Sustituye al directorio de arranque del usuario especificado.
  - ~+ → Sustituye al directorio de trabajo actual.
  - **. fichero** → Sustituye el punto por el contenido del fichero.
- Los caracteres especiales son una serie de caracteres interpretados por el shell, que se encarga de realizar las siguientes acciones:

- **#** → Cualquier cadena precedida por este carácter es considerada un comentario.
- **$** → Devuelve el contenido de la variable shell cuyo nombre va precedido por este carácter (sustitución de variables).
- **` `** → Devuelve el resultado del comando o serie de comandos encerrado entre estos caracteres (sustitución de comandos).
- **;** → Ejecuta secuencialmente en procesos separados todos los comandos unidos por este carácter (concatenación de comandos).
- **|** → Encadena la salida estándar del comando situado a la izquierda de este carácter con la entrada estándar del comando situado a la derecha (tubería).
- **<** → Envía a la entrada estándar de un comando el fichero indicado (redirección de entrada).
- **>, >&1** → Envía la salida estándar de un comando hacia el fichero indicado borrando previamente su contenido si ya existe (redirección de salida).
- **>>** → Envía la salida estándar de un comando hacia el fichero indicado añadiéndola a su contenido (redirección de salida).
- **2>, >&2** → Envía la salida de error de un comando hacia el fichero indicado borrando previamente su contenido si ya existe (redirección de errores).
- **2>>** → Envía la salida de error de un comando hacia el fichero indicado añadiéndola a su contenido (redirección de errores).
- **>&** → Asocia el descriptor de fichero cuyo número se encuentra a la izquierda de estos caracteres con el fichero ligado al descriptor de fichero cuyo número se encuentra a la derecha (redirección de ficheros).

– Para evitar que el shell interprete tanto los metacaracteres como los caracteres especiales se emplea el entrecomillado de la siguiente manera:
  - \ → Cualquier carácter precedido por backslash no es interpretado por el shell.
  - **' '** → Cualquier cadena encerrada entre comillas simples no es interpretado por el shell, excepto la comilla simple (').
  - **" "** → Cualquier cadena encerrada entre comillas dobles no es interpretado por el shell, excepto el backslash (\), el dólar ($), el acento grave (`) y la comilla doble (").

3.1.8. Expresiones regulares

– Una expresión regular es un patrón que define a un conjunto de cadenas de caracteres. Se construyen de forma análoga a las expresiones aritméticas utilizando los siguientes operadores:
  - **[ ]** → El patrón está formado por los caracteres incluidos dentro de los corchetes o incluidos en un rango de caracteres si están separados por "-".
  - **[^]** → El patrón está formado por los caracteres no incluidos dentro de los corchetes o no incluidos en un rango de caracteres si están separados por "-".
  - **.** → Representa cualquier carácter.
  - **^** → Al principio de una expresión regular representa el comienzo de una línea.
  - **$** → Al final de una expresión regular representa el final de una línea.

- \< → Al principio de una expresión regular representa el comienzo de una palabra.
- \> → Al final de una expresión regular representa el final de una palabra.
- | → Concatenación de expresiones regulares.
- **\(\)** → Almacena un patrón para añadirle un carácter de repetición posterior.
- **&** → Representa a la cadena de caracteres que coincidió con la expresión regular.
- **\n** → Representa a la cadena de caracteres que coincidió con la enésima expresión regular entre paréntesis.

– Existen ciertos caracteres que significan la repetición del patrón precedente o de una parte de él:
- **?** → El elemento precedente es opcional y debe coincidir al menos una vez.
- ***** → El elemento precedente debe coincidir de cero a más veces.
- **+** → El elemento precedente debe coincidir de una a más veces.
- **\{n,m\}** → El elemento precedente debe coincidir de "n" a "m" veces.

– Siempre que se usen expresiones regulares han de ir entrecomilladas con backslash (\), comillas simples (') o comillas dobles ("), según corresponda, para que el shell no interprete como metacaracteres o caracteres especiales los símbolos anteriores.

3.1.9. Shell scripts

– Un shell script es un fichero de texto con atributo de ejecución con una secuencia de comandos para el shell que incluye todo lo expuesto en los apartados anteriores. El ficheros de texto puede empezar con **#!/bin/bash** para indicar que se trata de un shell script.

– Pueden utilizar variables shell existentes o crear variables propias. También pueden recibir parámetros por posición desde la línea de comandos y ser utilizados como variables: $0 representa el comando introducido, $1 el primer parámetro, $2 el segundo parámetro...

– Una función en shell script es un conjunto de órdenes que se ejecutan con frecuencia dentro del script y al que se da un nombre para poder ser invocada. Así se declaran al principio del script:

```
nombre_función () {
comandos
}
```

Para invocarlas basta con poner su nombre sin paréntesis. También admiten el paso de parámetros posicionales. La forma de referirse a ellos es la misma que en el script. Si nos referimos a $1 dentro de la función el shell lo interpretará como el primer parámetro pasado a la función, mientras que $1 fuera de la función se interpretará como el primer parámetro del script.

3.1.10. Combinaciones de teclas habituales.

– En entornos Unix, existen algunas combinaciones de teclas que tienen un significado especial:
- **Ctrl-U**. Borra la línea donde se encuentra el cursor.
- **Ctrl-H**. Borra el carácter situado antes del cursor.
- **Ctrl-D**. Finaliza la entrada de datos, sale del shell si se está utilizando.
- **Ctrl-C**. Finaliza un programa en ejecución.
- **Ctrl-Z**. Ejecuta un programa como tarea en segundo plano.

  - **Ctrl-S**. Detiene la salida por pantalla.
  - **Ctrl-Q**. Activa nuevamente la salida por pantalla.
- El intérprete de comandos predeterminado, bash, permite la edición del histórico de comandos y el completado con tabulador para un uso interactivo:
  - **Flecha arriba**. Comienza la búsqueda en el histórico de comandos.
  - **Ctrl-R**. Comienza la búsqueda incremental en el histórico de comandos.
  - **TAB**. Completa el nombre de archivo en la línea de comandos.

3.2. Sistema de ficheros

3.2.1. Sistema de ficheros UNIX System V

- Ficheros → Conjuntos de datos con un nombre asociado que se guardan en un dispositivo de almacenamiento secundario. El nombre puede tener hasta 255 caracteres, evitando caracteres especiales (*, ?, $, etc.). No hay ninguna restricción respecto a las extensiones de los nombres. Para UNIX los ficheros son simples cadenas de bits sin formato alguno.
- Sistema de ficheros → Parte del sistema responsable de la administración de datos en dispositivos de almacenamiento secundario.
- Características del sistema de ficheros:
  - Los usuarios pueden crear, modificar y borrar ficheros.
  - Permite un crecimiento dinámico de los ficheros sin necesidad de definir tamaño máximo para cada uno.
  - Se almacenan por bloques numerados físicamente separados, cuyo tamaño fijo debe ser un múltiplo entero del tamaño de un sector del disco (cada uno de los bloques físicos de tamaño fijo en los que están divididas sus pistas) para aprovechar al máximo su espacio.
  - El fichero contiene información sobre el número del usuario que lo creó (propietario) y el número del grupo al que pertenece.
  - Cada fichero tiene tres tipos de acceso: lectura, escritura y ejecución. Estos tipos de acceso se extienden al propietario del fichero, al grupo del propietario y al resto de usuarios. Esta información está contenida en nueve bits que se distribuyen en el siguiente orden de izquierda a derecha:
    - Bit 0 → Permiso de lectura para el propietario del fichero.
    - Bit 1 → Permiso de escritura para el propietario del fichero.
    - Bit 2 → Permiso de ejecución para el propietario del fichero.
    - Bit 3 → Permiso de lectura para el grupo del propietario del fichero.
    - Bit 4 → Permiso de escritura para el grupo del propietario del fichero.
    - Bit 5 → Permiso de ejecución para el grupo del propietario del fichero.
    - Bit 6 → Permiso de lectura para el resto de usuarios.
    - Bit 7 → Permiso de escritura para el resto de usuarios.
    - Bit 8 → Permiso de ejecución para el resto de usuarios.
  - En el caso de los directorios, los permisos de acceso tiene el siguiente significado:

        - Lectura → Posibilidad de listar el contenido de un directorio.
        - Escritura → Posibilidad de crear y borrar ficheros en ese directorio.
        - Ejecución → Posibilidad de acceder a los ficheros del directorio.
    - Cada usuario puede estructurar los ficheros como desee.
    - Es posible hacer copias de seguridad de todos los ficheros.
    - Es posible cifrar y descifrar la información.
    - El usuario sólo ve los datos desde el punto de vista lógico, sin preocuparse de los dispositivos físicos ni de la transferencia de datos.
    - Tiene estructura de árbol invertido formado por directorios y subdirectorios que se inician en el directorio raíz.
    - Para acceder a un fichero debe indicarse el camino absoluto (desde el directorio raíz) o el camino relativo (desde el directorio actual).
- Organización del sistema de ficheros:
    - **Raíz (/)** → Único en el sistema, origen de los demás directorios.
    - **/bin** → Contiene comandos ejecutables comunes para todos los usuarios.
    - **/boot** → Contiene el ejecutable del núcleo, cargado en memoria al iniciar el sistema.
    - **/dev** → Contiene ficheros de dispositivo para la comunicación con periféricos.
    - **/etc** → Contiene ficheros de configuración usados por el sistema.
    - **/home** → Contiene los directorios de trabajo de los usuarios.
    - **/lib** → Contiene librerías del sistema y **/lib/modules** contiene los módulos del núcleo.
    - **/mnt** → Directorio donde se montan al arrancar el sistema los dispositivos de almacenamiento externo (CD-ROM, floppy, discos) y los sistemas de ficheros remotos NIS, según el fichero de configuración **/etc/fstab**.
    - **/opt** → Contiene paquetes de software adicionales de las aplicaciones.
    - **/proc** → Guarda información de los procesos del sistema.
    - **/sbin** → Contiene utilidades para la administración del sistema.
    - **/tmp** → Contiene ficheros temporales de trabajo del sistema.
    - **/usr** → Contiene datos constantes compartidos de sólo lectura. Formado por directorios como **/usr/bin** y **/usr/sbin** (comandos ejecutables menos frecuentes), **/usr/lib** (librerías para compiladores) y **/usr/local** (ficheros correspondientes a software local).
    - **/var** → Contiene datos variables que pueden ser compartidos o no, como ficheros temporales de las aplicaciones ubicadas en **/usr**, y ficheros de log y spool.
- Sistema de permisos de acceso a los ficheros → Utiliza los números de identificación de usuario (UID) y de grupo (GID) asignados en el sistema a un determinado usuario para permitirle o denegarle el acceso al fichero de la siguiente manera:
    - Si el UID del usuario es 0 (administrador) se le dan los permisos correspondientes al propietario del fichero.
    - Si el UID del usuario es igual al UID del propietario del fichero, se le dan los permisos correspondientes al propietario.

- Si el GID del usuario es igual al GID del grupo del propietario del fichero, se le dan los permisos correspondientes al grupo.
- Si no se da ninguno de los tres casos anteriores, se le dan los permisos correspondientes al resto de usuarios.

– Existen otros tres bits más en el caso de los ficheros ejecutables:
  - Bit set-uid → Si un programa tiene este bit activo, entonces al ejecutarlo creará un proceso que tendrá como identificador de usuario al identificador del propietario del fichero, es decir, para el sistema será como si lo hubiese lanzado el propietario.
  - Bit set-gid → Equivalente al bit set-uid, pero aplicado al grupo del propietario.
  - Sticky bit → Indica al núcleo que el fichero es un programa con capacidad para que varios procesos compartan su código, y por tanto debe permanecer en memoria aunque alguno de los procesos que lo utiliza finalice. Activando este bit en un directorio, se evita que un usuario que no es propietario de un archivo lo pueda eliminar del mismo.

– Existen los siguientes tipos de fichero:
  - **Ficheros ordinarios** → Son los ficheros tipo texto y los programas ejecutables.
  - **Directorios** → Ficheros que contienen información que permite localizar otros ficheros. Cada entrada en un directorio contiene el nombre del fichero y un número de inodo. No hay límite en el número de ficheros o subdirectorios que pueden contener. Sólo pueden ser modificados por el sistema operativo.
  - **Ficheros de dispositivo** → Proporcionan un mecanismo para relacionar dispositivos físicos con nombres de fichero. Cada dispositivo soportado está asociado al menos con un fichero especial del directorio **/dev**. No tienen tamaño definido y no pueden usarse con ellos los comandos habituales para el manejo de ficheros. Hay dos tipos:
    - **Tipo bloque** → Sirven para comunicarse con discos y cintas. Los datos se transfieren en bloques de longitud fija a través de buffers de E/S a los que el sistema asigna un descriptor con el nombre del dispositivo y la dirección del bloque de datos contenido.
    - **Tipo carácter** → Sirven para comunicarse con terminales e impresoras. Los datos se transfieren carácter a carácter sin necesidad de buffers. También permiten el acceso a dispositivos tipo bloque en modo RAW I/O.
  - **Tuberías FIFO** → Son ficheros que se comportan como tuberías. El proceso que escribe en la tubería no finaliza hasta que la información escrita es leída de la tubería. el proceso de lectura espera hasta que exista algo para leer antes de finalizar. La tubería no almacena datos, sólo vincula dos procesos que no tienen que estar en la misma línea de comandos ni tampoco ser ejecutados por el mismo usuario. Se crean con **mknod**.
  - **Enlaces** → Ficheros que apuntan a otros ficheros, existen dos tipos:
    - **Físicos** → A todos los efectos funcionan como copias del fichero original con el mismo número de inodo, aunque sólo existe un único fichero físico. Cualquier modificación realizada en uno de ellos se refleja en los demás y para borrar un fichero hay que borrar todos sus enlaces físicos. No es posible

usarlos entre ficheros pertenecientes a sistemas diferentes. Un enlace físico sobre un enlace simbólico produce una copia de dicho enlace simbólico.

- **Simbólico** → Permiten enlazar ficheros de distintos sistemas, si se borra el fichero original dejan de funcionar. Tienen el mismo número de inodo que el fichero al que apuntan, pero son un tipo de fichero diferente. Se pueden hacer enlaces simbólicos a ficheros, a enlaces físicos y a otros enlaces simbólicos.

– Estructura lógica del sistema de ficheros → Suele almacenarse en dispositivos de tipo bloque. El núcleo trabaja con el sistema de ficheros a nivel lógico, no a nivel físico. Formada por:

- **Boot o sistema de arranque** → Ocupa el principio del sistema de ficheros (primer sector), contiene el código para buscar el sistema operativo y cargarlo en memoria.
- **Superbloque** → Describe el estado de un sistema de ficheros y ocupa siempre el primer bloque lógico del disco. Si se pierde el superbloque, se perderían todos los datos almacenados. La información que contiene es la siguiente:
  - Tamaño del sistema de ficheros.
  - Tamaño de cada bloque de disco.
  - Lista de bloques libres.
  - Índice del siguiente bloque libre en la lista de bloques libres.
  - Tamaño de la lista de inodos.
  - Número de inodos libres.
  - Índice del siguiente inodo libre en la lista de inodos libres.
  - Campos de bloqueo de elementos de las listas de bloques e inodos libres.
  - Indicador de modificación del superbloque.
- **Lista de nodos índice (inodos)** → Tiene una entrada por cada fichero del sistema, durante el arranque el núcleo lee esta lista y crea una copia de ella en memoria. La información que guardan esta tabla de inodos es la siguiente:
  - Número de inodo.
  - UID y GID del propietario del fichero.
  - Tipo de fichero.
  - Derechos de acceso, set-uid, set-gid y sticky bit.
  - Fecha de última modificación.
  - Número de enlaces.
  - Entradas para los bloques de dirección → Son trece, los diez primeros apuntan a bloques directos y los tres últimos apuntan a bloques indirectos (el primero es un puntero indirecto simple, el segundo un puntero indirecto doble y el tercero un puntero indirecto triple). Esto quiere decir que ficheros con un tamaño menor o igual a 10 bloques lógicos son referenciados inmediatamente.

  El número de inodo 0 está reservado para marcar un fichero borrado y el 1 para bloques erróneos de disco. El primer inodo efectivo es el 2 que corresponde al directorio raíz y a partir de ahí todos los demás.

- **Bloques de datos** → Donde se encuentran propiamente los datos de los ficheros. Se accede a ellos a través del contenido de los directorios, que relaciona los nombres dados a ficheros y subdirectorios con sus respectivos números de inodo.

3.2.2. Sistema de ficheros Ext2/Ext3

- En 1992 se creó un nuevo sistema de ficheros llamado Extended File System (Ext), que permitía particiones de 2 GB y tenia nombres de ficheros de 255 caracteres, aunque presentaba diversos problemas. Un año mas tarde apareció un Second Extended File System (Ext2). Es un sistema de ficheros indexado, basado en inodos (índices).
- En este sistema, cada fichero es representado por una estructura, denominada inodo (inode). Cada inodo contiene la descripción del fichero, el tipo, derechos de acceso, propietario, fecha y hora, tamaño y punteros a los clusters del fichero. Ext2 implementa el concepto de enlace, en que varios nombres de ficheros pueden ser asociados al mismo inodo.
- Este inodo de Ext2 tiene un tamaño fijo entre 1 y 4 K, y no usa una FAT, sino una tabla de inodos distribuidos en un número determinable de grupos a través de la superficie, tiene un limite máximo de 4 GB por archivo, 4 TB por partición y detección de desmontaje incorrecto.
- Ext2 divide la partición en grupos de bloques, cada uno de los cuales contiene lo siguiente:
  - Un superbloque, que guarda información sobre el número de bloques e inodos existentes, el tamaño de los bloques, etc.
  - Un descriptor de grupo, que almacena información sobre la localización de los mapas de bits, el número de bloques e inodos libres en el grupo y el número de directorios que contiene. Esta información es importante ya que el sistema de ficheros intenta expandir los directorios uniformemente por el disco.
  - Dos mapas de bits que guardan información sobre los bloques libres y los inodos libres respectivamente. Cada mapa ocupa un bloque.
  - La tabla de inodos, en la que cada entrada ocupa 128 bytes, el doble de la de un sistema UNIX estándar. El espacio extra se utiliza de la siguiente manera:
    - En vez de diez direcciones directas y tres indirectas a bloques, Linux permite doce direcciones directas y tres indirectas.
    - Las direcciones pasan de 24 a 32 bits para manejar particiones mayores de 16 GB.
- La idea de dividir el sistema de ficheros en grupos de bloques es mantener los inodos y los bloques de datos más cerca, para evitar largas búsquedas. Cuando un fichero aumenta de tamaño, intenta situar los bloques nuevos en el mismo grupo de bloques que el resto del fichero, así como los nuevos ficheros tratan de colocarse en el mismo grupo de bloques que su directorio.
- Un sistema de ficheros especial de Linux es el directorio /proc, en el que se crea un subdirectorio de nombre igual a su PID por cada proceso existente en el sistema. Contiene información sobre los procesos, y otra mucha información acerca de la CPU, las particiones de disco, los dispositivos, los vectores de interrupción, los sistemas de ficheros, los módulos cargados, etc.

- El sistema de ficheros Ext3 es una implementación de Ext2 con journaling, que se basa en llevar un registro de todas las transacciones de lectura/escritura que sufre un sistema de archivos, y básicamente funciona de la siguiente manera:
  - Se bloquean las estructuras de datos afectadas por la transacción para que ningún otro proceso pueda modificarlas mientras dura la transacción.
  - Se reserva un recurso para almacenar el journal. Por lo general suelen ser unos bloques de disco, de modo que si el sistema se para de forma abrupta (corte eléctrico, avería, fallo del sistema operativo...) el journal siga disponible una vez reiniciado el sistema.
  - Se efectúan una a una las modificaciones en la estructura de datos. Para cada una:
    - Se apunta en el journal como deshacer la modificación y se asegura de que esta información se escribe físicamente en el disco.
    - Se realiza la modificación.
    - Si en cualquier momento se quiere cancelar la transacción se deshacen los cambios uno a uno leyéndolos y borrándolos del journal.
    - Si todo ha ido bien, se borra el journal y se desbloquean las estructuras de datos afectadas.
  - En el caso concreto de los sistemas de archivos, el journaling se suele limitar a las operaciones que afectan a las estructuras que mantienen información sobre:
    - Estructuras de directorio.
    - Bloques libres de disco.
    - Descriptores de archivo (tamaño, fecha de modificación...)
- El núcleo (kernel) de Linux cuenta con un Sistema de Ficheros Virtual (Virtual File System, VFS) que se usa siempre que se llama al Sistema de Ficheros. Esto permite que en Linux se puedan usar distintos sistemas de ficheros fácilmente. Aparte de ext2 y ext3, gracias al VFS de Linux podemos usar diversos sistemas de ficheros, como ReiserFS, Reiser4, UFS, XFS, JFS, FAT, UFS, UMSDOS, ZFS, etc.

3.3. Núcleo (kernel)

3.3.1. Primitivas del núcleo

- **fork** → Crea dos procesos idénticos pero independientes entre sí (padre e hijo). Se copian los datos, la pila y el código. Los ficheros abiertos por el padre los hereda el hijo con un puntero de lectura/escritura común.
- **exec** → Hace que un proceso cargue y ejecute un nuevo código y datos, sin retorno para el código invocante, pudiendo acceder a todos los ficheros abiertos o heredados previamente.
- **wait** → Hace que el proceso padre espere a la finalización de alguno de sus hijos. Este comando retorna el PID del proceso finalizado y su estado de finalización. La espera del padre no puede ser selectiva hacia un hijo en concreto.
- **exit** → Finaliza un proceso, vaciando buffers y cerrando ficheros.

- **pipe** → Crea un canal entre dos procesos, un tipo de fichero especial sobre el que se pueden hacer operaciones de lectura y escritura. El comando **pipe(p)** devuelve el descriptor p[0] para acceder al canal en lectura y p[1] en escritura.
- **signal** → Establece qué hacer cuando se producen sucesos (señales) externos o del sistema. Puede terminar el proceso, o ignorar la señal, o ejecutar un determinado procedimiento. En el caso de un **fork**, el proceso hijo hereda el contexto de señales del padre; en un **exec** se le asigna el tratamiento por defecto, que es la finalización del proceso.
- **open** y **creat** → Permiten abrir y crear ficheros en modo lectura (modo 0), escritura (modo 1) o en lectura/escritura (modo 2). A cada fichero se le asocia un puntero que señala el siguiente byte a leer o escribir, funciona de manera secuencial y es incrementado cuando se hace una llamada a una primitiva de entrada/salida.
- **close** → Cierra un fichero y elimina el indicativo asociado a él en el proceso.
- **read** y **write** → Permiten leer y escribir en los ficheros. Para un mismo proceso no puede haber varias lecturas en curso., read bloquea el proceso hasta que termina la operación de lectura.
- **lseek** → Permite realizar un acceso aleatorio a la información de un fichero.

3.3.2. Gestión de procesos

- Un programa es un conjunto de instrucciones y datos guardados en un sistema de almacenamiento secundario, mientras que un proceso es la ejecución concreta con asignación de recursos de un programa, es decir, la ejecución de una imagen.
- La imagen de un proceso se compone de:
  - **Segmento de código ejecutable** → Compartido por los procesos que ejecutan el mismo código. Sólo hay una copia en la memoria y está protegida contra escritura.
  - **Segmento de datos** → Contiene los datos del programa conocidos durante la compilación y los asignados dinámicamente en tiempo de ejecución.
  - **Segmento de pila** → Consecuencia de las llamadas a procedimientos, en cada llamada guarda los parámetros, la información de contexto y enlace (registros y dirección de retorno) y los datos locales del procedimiento.
- A cada proceso, el núcleo le asigna dos números: el identificador de proceso (PID) y el identificador de proceso padre (PPID). Si no se indica lo contrario (ignorar señal hangup), la finalización del proceso padre hace que todos sus procesos hijos también finalicen.
- Por ser un sistema multiusuario y multitarea, la CPU asigna un tiempo limitado de los recursos del sistema a cada proceso. Por esta razón, un proceso puede pasar por los siguientes estados:
  - **En ejecución** → En máquinas con un procesador sólo un proceso puede permanecer en este estado en un determinado instante y por un tiempo limitado. Este tiempo es asignado a cada proceso en función de su prioridad. Si el proceso debe esperar a una operación de E/S pasa a estado de latencia.

  - **En espera** → Cuando el tiempo de ejecución asignado a un proceso se agota pasa a este estado, en el que permanecerá hasta que se le vuelva a asignar recursos y pase nuevamente a estado de ejecución.
  - **En latencia** → Si un proceso en ejecución debe esperar a una operación de E/S o le faltan recursos pasa a ese estado, en el que permanecerá hasta que finalice la operación de E/S o se liberen los recursos. En este momento pasará a estado de espera.
- Cuando un proceso ejecuta una primitiva del núcleo, ésta se lleva a cabo en un modo más privilegiado que los comandos de usuario → Se crean dos pilas para el proceso, una para el modo usuario y otra para el modo núcleo. Esta última no puede ser compartida con otros procesos, por tanto es necesario hacer una copia de la imagen asociada al modo usuario sobre la asociada al modo núcleo.
- El algoritmo de asignación de tiempos de procesador esta basado en lo siguiente:
  - **Procesos en modo núcleo** → Tienen una prioridad mayor, viene definida por el código ejecutado. Tienen prioridad los sucesos ligados a la gestión del disco sobre los de periféricos lentos.
  - **Procesos en modo usuario** → La prioridad de cada proceso se evalúa periódicamente, y responde a la relación (tiempo transcurrido del proceso / tiempo asignado del procesador), que equivale al algoritmo "Round Robin".
- No existen segmentos de datos compartidos entre procesos, se comunican entre sí mediante señales, que son interrupciones software enviadas de forma asíncrona. Los procesos receptores de estas señales pueden ignorarlas, o invocar una rutina de tratamiento general del núcleo, o tratarlas con una rutina propia. Los procesos emisores deben tener el mismo propietario que los receptores. Normalmente por defecto provocan la finalización del proceso receptor.
- Esquemas de cooperación entre procesos:
  - El proceso padre crea un proceso hijo para ejecutar un mandato y tratar el resultado de la ejecución, o para que sea el proceso hijo el que lo lleve a cabo todo.
  - El proceso padre crea dos procesos hijos y liga con un pipeline la salida del hijo 1 con la entrada del hijo 2. Para que dos procesos puedan comunicarse por un pipeline es necesario que tengan un padre común. Sin embargo existen ficheros especiales llamados **tuberías FIFO** que permiten la comunicación entre procesos de diferentes padres. Funcionan como ficheros en disco, y pueden ser heredados por otros procesos.
  - Otra forma de comunicación asíncrona entre procesos que se ejecutan en diferentes máquinas son los **sockets**.
- Todos los procesos llevan asociados los números de identificación del usuario (UID) y grupo (GID) que los ejecuta. Estos números son utilizados para la gestión de los permisos de lectura, escritura y ejecución sobre los ficheros del sistema.
- Un proceso zombie es un proceso finalizado, al que su proceso padre no ha reconocido como tal porque no ha ejecutado la primitiva **wait**. Si existen muchos de estos procesos significa que el proceso padre está bloqueado, por tanto hay que eliminar al padre para que **init** los adopte y a su vez los elimine.

3.3.3. Gestión de memoria

- Swapper → Proceso que gestiona el trasvase entre la memoria principal y la memoria virtual.
- Se basa en el algoritmo del primer ajuste, y el proceso que se carga en memoria principal se hace sobre la base de la mayor permanencia en memoria virtual, favoreciendo a procesos pequeños.

3.3.4. Gestión de entrada/salida

- Los dispositivos tipo bloque que puedan trabajar en el modo bloque y en el modo carácter tendrán dos ficheros de dispositivo. Debido al uso de buffers un comando read o write no significa por tanto una entrada o salida física.
- Para crear un fichero de dispositivo se emplea el comando **mknod**:

  mknod ident_disp tipo_disp major_number minor_number

  El identificador de estos ficheros está formado por un nombre y dos dígitos, que indican el tipo de E/S, si es tipo bloque o carácter y el número asignado en el sistema. El tipo de dispositivo puede ser bloque (b) o carácter (c). El major number identifica una clase de dispositivo (listadas en **/etc/devices**), el minor number indica un elemento de la clase de dispositivo.
- El enlace entre los drivers de los periféricos y los procedimientos tipo read, write... se realiza mediante una tabla de configuración → Consta de un índice (que es el major number del periférico) y devuelve el punto de entrada del driver correspondiente.
- Hay que tener en cuenta lo siguiente:
  - El tratamiento de errores de entrada/salida a nivel usuario no tiene sentido porque no tiene por qué haber entrada/salida física.
  - Esto puede dar lugar a incoherencias en dispositivos secuenciales, por ello se asigna un único buffer por periférico.
  - Puede haber pérdida de información en caso de un corte de energía. Para evitarlo la primitiva **sync** realiza copias periódicas en disco del contenido de los buffers.
- El modo de acceso carácter en dispositivos tipo bloque hace que los datos no pasen por los buffers, sino a través del espacio de memoria del proceso usuario. Esto es más eficaz en ciertas operaciones de entrada/salida, y se utiliza mejor la organización física del disco.
- Existe una tabla intermedia entre los procesos y la tabla de inodos, que es la tabla de ficheros abiertos. En ella a cada uno de los ficheros abiertos se le asocia un puntero que señala al elemento siguiente. Este puntero es compartido si los procesos que acceden a él tienen relación padre-hijo, y no es compartido si se ha realizado una operación explícita con **open** o **creat**.

3.3.5. Arranque del sistema

- En el arranque de un sistema UNIX hay dos fases:
  - Arranque de la ROM → Característico de cada ordenador, en él se suele comprobar todo el hardware del sistema. Si todo es correcto, se lee del disco y se ejecuta un programa que carga en memoria el núcleo UNIX y le pasa el control. El fichero que contiene el núcleo normalmente se guarda en el directorio **/boot**.

- Arranque de UNIX → El sistema lanza el proceso **/etc/init** como origen de todos los procesos (PID=1). Se mantiene vivo hasta que se apaga la máquina, eliminando los procesos huérfanos. Este lee el fichero **/etc/inittab**, se trata de un fichero de texto que contiene las instrucciones para el arranque del sistema operativo.

– **/etc/inittab** → Formado por cuatro campos separados por dos puntos, son los siguientes:
  - Identificador → Uno o dos caracteres alfanuméricos que identifican las líneas.
  - Nivel → Nivel o niveles de ejecución para los cuales la instrucción es válida, hace de filtro de instrucciones para cada nivel. Si se deja vacío es válida para todos los niveles.
  - Acción → Contiene una palabra clave que le indica a **init** cómo debe ejecutar la instrucción, y algunas de ellas son las siguientes:
    * respawn → Arranca el proceso si no existe, si existe no hace nada. No espera a que termine y cuando finaliza lo vuelve a arrancar.
    * once → Arranca el proceso, no espera a que termine y no se vuelve a arrancar.
    * wait → Arranca el proceso si no existe, si existe no hace nada. Espera a que termine y después continúa.
    * boot → Actúa como once, pero sólo en el arranque del sistema.
    * bootwait → Actúa como wait, pero sólo en el arranque del sistema.
    * powerfail → Actúa como once, pero sólo se ejecuta cuando init recibe una señal de fallo de alimentación.
    * powerwait → Actúa como wait, pero sólo se ejecuta cuando init recibe una señal de fallo de alimentación.
    * off → Se envía una señal de aviso al proceso y tras unos segundos lo finaliza, si no existe no hace nada. Se usa para desactivar instrucciones temporalmente.
    * initdefault → Determina cuál es el nivel de ejecución inicial. Si este parámetro no existe el sistema pide un nivel de ejecución para arrancar.
    * sysinit → Actúa como wait, antes de acceder a la consola del sistema.
  - Programa → Se crea un proceso hijo (fork) y en este nuevo proceso el programa se ejecuta mediante exec.

– **/etc/init** trabaja en esta secuencia: explora **/etc/inittab**, busca la etiqueta initdefault, ejecuta las entradas con etiqueta boot o bootwait, ejecuta las entradas con etiqueta sysinit y después ejecuta el resto de forma secuencial.

– **/etc/init.d/rcs** → Es el archivo de inicialización principal del sistema. Ejecuta todos los scripts situados en el directorio **/etc/rcs.d/**, realizando varias funciones:
  - Define el nombre del ordenador.
  - Establece la fecha y la hora.
  - Verifica y monta los sistemas de ficheros.
  - Activa el fichero de memoria virtual (swap).
  - Actualiza el contenido de los buffers de entrada y salida del disco.
  - Borra ficheros existentes en /tmp.
  - Inicializa los sistemas de impresión.

- **/etc/rcs.local** → Es otro script que contiene normalmente órdenes de inicialización específicas del propio sistema. Puede ser iniciado tanto desde **/etc/init.d/rcs** como desde **/etc/inittab**.
- **/etc/getty** → Procesos normalmente arrancados con la opción respawn, hay uno por cada terminal conectado al sistema. La sintaxis es la siguiente: getty línea [velocidad]. "Línea" indica el nombre del fichero de dispositivo existente en **/dev**, y "velocidad" indica una etiqueta dentro del fichero de configuración **/etc/gettydefs**. Realiza lo siguiente por este orden:
  - Establece las opciones del terminal.
  - Muestra el contenido de **/etc/issue**.
  - Saca el prompt del login y espera una entrada.
  - Una vez introducido un usuario, ejecuta el programa **/bin/login** y espera.

  Cuando un usuario provoca un logout, hace que el proceso padre getty termine y vuelva a comenzar debido al "respawn" del fichero **/etc/inittab**.
- **/etc/gettydefs** → Formado por cinco campos separados por almohadillas, son los siguientes:
  - Etiqueta → Identifica una configuración de velocidad del terminal.
  - Modificadores iniciales → Indican cómo ha de programarse el terminal antes de ejecutar el login.
  - Modificadores finales → Indican cómo ha de programarse el terminal después de ejecutar el login.
  - Prompt login → Texto que aparece al solicitar el login.
  - Siguiente etiqueta → En caso de recibir un break, indica la etiqueta a la que acceder.
- **/bin/login** → Proceso invocado en cada terminal por **/etc/getty**, realiza lo siguiente:
  - Comprueba que el usuario introducido existe en el fichero **/etc/passwd**, en ese caso saca por pantalla el prompt establecido para el usuario en dicho fichero y solicita la contraseña. Nuevamente accede a **/etc/passwd** para validar la palabra clave.
  - Realiza dos llamadas al sistema, **setuid** y **setgid**, que asignan a los números de identificación del usuario (UID) y del grupo (GID) del sistema respectivamente aquellos que aparecen en **/etc/passwd**.
  - Cambia el directorio actual por el de arranque de usuario que aparece en **/etc/passwd**.
  - Por último, se ejecuta mediante exec (sustituyendo a login) el shell indicado en **/etc/passwd**, que ejecuta los ficheros de configuración **/etc/profile** del sistema y **~/.bash_profile** del usuario, y muestra el prompt.
  - Para cambiar el shell de un usuario sin modificar **/etc/passwd**, se puede introducir en **~/.bash_profile** la instrucción "exec nuevo_shell".
  - Al finalizar la sesión se ejecutan las órdenes del fichero **~/.bash_logout**.
- **/etc/passwd** → Cada entrada de este fichero contiene información sobre un usuario concreto. Formado por siete campos separados por dos puntos, son los siguientes:
  - Usuario → Nombre asignado al usuario. Debe tener entre uno y ocho caracteres.
  - Clave → Password de acceso, esta encriptada por el sistema.
  - UID → Número de identificación de usuario. El cero corresponde al administrador.

  - GID → Número de identificación del grupo del usuario. Asociado a una línea del fichero **/etc/group**.
  - Comentario → Espacio libre para incluir información del usuario.
  - Directorio arranque → Es el directorio inicial para el usuario cuando arranca el shell.
  - Programa inicio → Programa que se debe ejecutar cada vez que entre el usuario al sistema. Normalmente será el shell con el que se quiere trabajar.
- **/etc/group** → Es el equivalente a **/etc/passwd**, pero para los grupos. Formado por cuatro campos separados por dos puntos, son los siguientes:
  - Grupo → Nombre asignado al grupo.
  - Clave → No utilizado.
  - GID → Número de identificación del grupo.
  - Usuarios → Lista separada por comas de los nombres de usuarios que pueden convertirse en miembros del grupo con la orden **newgrp**.
- En todo momento UNIX está en un determinado nivel de ejecución, que determina cuántos y qué usuarios se pueden conectar al sistema. Se define por un dígito y por defecto el nivel de ejecución de arranque suele ser el multiusuario. Para cambiar el nivel de ejecución el administrador introduce **telinit nivel_ejecución**, lo cual provoca que todos los procesos que no pertenezcan a este nivel terminen.
- Al entrar a un nivel de ejecución se ejecutan todos los scripts ubicados en **/etc/rcnivel_ejecución.d/**. La primera letra del nombre del script determina la manera en que se ejecuta el script: aquellos cuyos nombres comienzan con K se ejecutan con el argumento stop, y los que comienzan con S se ejecutan con el argumento start. Los scripts se ejecutan de acuerdo al orden alfabético de sus nombres; de esta manera los scripts “stop” se ejecutan antes que los scripts “start” y los dos dígitos a continuación de K o S determinan el orden en que se ejecutan.
- Los scripts situados en **/etc/rcnivel_ejecución.d/** son sólo enlaces simbólicos que apuntan a los scripts situados en **/etc/init.d/**. Cada script también acepta como argumento “restart” y “force-reload”; estos métodos se pueden utilizar para reiniciar los servicios una vez que haya sido arrancado el sistema o forzarlos para que vuelvan a cargar sus archivos de configuración.

3.3.6. Parada del sistema

- Si se necesita desconectar el sistema o pasar a modo monousuario para realizar copias de seguridad es necesario que el administrador utilice la orden **shutdown**, de lo contrario se puede perder información debido a que debe vaciarse el contenido de los buffers y las tablas de ficheros hacia los discos para no perder información.
- **shutdown [-rfh] [-t segundos] [mensaje]** → Finaliza los procesos del sistema y los programas de los usuarios, desmonta todos los sistemas de ficheros excepto el raíz y vacía todos los buffers (**-r** reinicia el sistema; **-h** desconecta el sistema; **-f** hace una parada rápida; **-t segundos** espera los segundos indicados).

3.3.7. Procesos automáticos

- Un proceso demonio es un proceso encargado de una labor específica y que se ejecuta en segundo plano, por ejemplo **init**.
- **crond** → Proceso demonio que ejecuta programas a determinadas horas. Se encarga de leer el fichero **/etc/crontab** (que establece los procesos a lanzar cada hora, día, semana, mes) y los ficheros guardados en **/var/spool/cron** asignados a cada usuario (que establecen una serie de acciones y horas a las que deben ejecutarse).
- Para restringir el acceso a **crond** existen los ficheros **/etc/cron.allow** y **/etc/cron.deny**.
- **crontab [-ledc] [fichero]** → Manipula el fichero de configuración asignado al usuario, crontab fichero sustituye el fichero de configuración actual por "fichero" (**-l** lista el fichero de configuración; **-e** lo edita; **-d** lo borra; **-c** crea un nuevo directorio crontab). El fichero de configuración está formado por seis campos separados por espacios:

  minuto hora día mes día_semana orden_a_ejecutar

  Un asterisco en un campo indica que es válido cualquier valor en ese campo; una lista de enteros separados por comas indica que son válidos sólo esos valores; dos enteros separados por un guión indican un rango de valores válidos.
- Otra manera de ejecutar procesos automáticamente es mediante la orden **at** → A través del programa **atrun** (que debe ser ejecutado por **crond** cada cierto tiempo) se ejecutan las órdenes contenidas en su directorio de configuración **/var/spool/at**. La orden **at** permite introducir desde la entrada estándar hasta que pulsemos <Ctrl-d> una serie de trabajos a realizar y los copia en dicho directorio. Al lanzar esta orden el sistema nos asigna un número de trabajo para poder identificarlo. Hay dos modalidades:
  - **at [-m] [-f fichero] hora [fecha] [+incremento]** → **-f fichero** ejecuta las órdenes incluidas en "fichero"; **-m** envía un correo electrónico después de ejecutar el trabajo.
  - **at [-lr] [trabajos]** → **-l** muestra una lista de trabajos de usuario; **-r** borra trabajos en la cola de ejecución.
- **atq [-n]** → Visualiza los trabajos en cola sin ejecutar del usuario (**-n** lista los trabajos de todos los usuarios).
- **atrm [-afi] [trabajos]** → Borra de la cola trabajos del usuario (**-a** borra todos; **-f** borra de forma incondicional; **-i** borra después de preguntar).
- Para restringir el acceso a **atrun** existen los ficheros **/etc/at.allow** y **/etc/at.deny**.

3.4. Entorno gráfico X Window

- X Window es un sistema cliente-servidor que permite que las aplicaciones se ejecuten y se compartan a través de la red, aunque pueden ejecutarse sin necesidad de red. Está controlado por dos piezas de software, una ejecutándose en el cliente y otra en el servidor, que pueden estar en sistemas distintos o en la misma máquina.
- El servidor muestra la aplicación que se está ejecutando en el cliente, mientras que éste proporciona los programas y recursos necesarios para ejecutar una aplicación. Los recursos

residen en el cliente, mientras que la aplicación se muestra e interactúa con el servidor. Por tanto las aplicaciones pueden ejecutarse en una máquina local o en una máquina remota.

- X Window convierte en eventos las señales que recibe del ratón y del teclado, y responde a ellos realizando las acciones apropiadas. Esto permite trabajar en nuestra máquina con aplicaciones que no han sido desarrolladas para nuestro sistema operativo, ya que utilizamos los recursos del cliente remoto.
- XFree86 es una implementación del sistema X Window para plataformas Intel, se trata del software que se encuentra entre el interfaz gráfico (KDE, Gnome) y el hardware.
- Se configura mediante el fichero **/etc/X11/XF86Config**, que es posible modificar con la aplicación **xf86config**.
- Para arrancar el modo gráfico desde el modo texto introducir el comando **startx**.

4. Administración del sistema

4.1. Introducción

- La administración de un sistema UNIX es uno de los aspectos menos estándar. Durante la instalación se crea automáticamente la cuenta root con plenos derechos sobre el sistema. Para comprobar la configuración del sistema, introducir el comando **dmesg**.
- Responsabilidades del administrador sobre el hardware:
  - Verificación de la correcta instalación del hardware.
  - Comprobación del estado de los periféricos.
  - Instalación de nuevos dispositivos.
- Responsabilidades del administrador sobre el software:
  - Instalación y configuración del sistema operativo.
  - Creación y mantenimiento de los sistemas de ficheros.
  - Control sobre la utilización del sistema de ficheros y su crecimiento.
  - Diseño e implementación de rutinas para hacer copias de seguridad y recuperación.
  - Configuración y mantenimiento del software de cualquier dispositivo.
  - Actualización del sistema operativo en caso necesario.
  - Instalación del software de cualquier aplicación.
- Responsabilidades del administrador sobre los usuarios:
  - Dar altas y bajas de usuarios.
  - Permitir el acceso a los usuarios de forma controlada.
  - Evaluar las necesidades de los usuarios en cuanto a equipos se refiere.
  - Proporcionarles asistencia.
  - Mantenerles informados sobre los nuevos servicios.

4.2. Administración de aplicaciones

4.2.1. Instalación de aplicaciones

- Hay varias maneras de instalar una aplicación. Normalmente los directorios de instalación suelen ser: **/usr/**, **/usr/local/**, **/var/**, **/var/local/**, **/bin/**, **/sbin/** y **/lib/**.
- Mediante el código fuente comprimido en un fichero tgz o tar.gz, para ello hay que:
  - Descomprimir el fichero tgz con **tar zxvf fichero.tgz**, normalmente en el directorio **/usr/local/src**. Si la primera entrada del fichero comprimido no es un directorio (**tar t**), es necesario crear uno, copiar el fichero y extraer su contenido dentro del mismo.
  - Ejecutar el comando **./configure** para crear el fichero Makefile que se utilizará en la compilación e instalación. Compilar e instalar introduciendo **make&&make install**.
- Mediante ficheros binarios comprimidos en un fichero tgz o tar.gz, para ello hay que descomprimir el fichero con **tar zxvf fichero.tgz**, normalmente en el directorio **/usr/local/**.
- En sistemas basados en Red Hat (Fedora, SuSE, Mandrake) mediante paquetes RPM binarios o en código fuente, introduciendo el comando **rpm [opciones] paquete.rpm**. Los paquetes RPM operan como un programa de instalación para una aplicación determinada. Las opciones son:
  - **-U** → Instala un paquete nuevo o actualiza un paquete instalado previamente.
  - **-vh** → Muestra información durante la instalación.
  - **-e** → Desinstala un paquete.
  - **-q** → Comprueba si está instalado un paquete.
  - **-qpl** → Muestra un listado de todos los archivos incluidos en el paquete.
- En sistemas basados en Debian mediante paquetes DEB, introduciendo el comando **dpkg [opciones] paquete.deb**. El sistema de paquetes Debian requiere que el paquete esté configurado para evitar problemas de dependencia. Algunos paquetes también necesitan ser configurados para funcionar correctamente. Las opciones son:
  - **-i** → Instala un paquete nuevo o actualiza un paquete instalado previamente.
  - **--configure** → Configura un paquete.
  - **--purge** → Desinstala un paquete.
  - **-s** → Comprueba si está instalado y configurado un paquete.
  - **--info** → Muestra información sobre el contenido del paquete.

4.2.2. Modificación del núcleo utilizando código fuente.

- La nomenclatura de los núcleos Linux tiene tres partes: la versión del núcleo y los niveles de revisión y parche. Por ejemplo el kernel 2.4.20-8, representa la versión 2.4, revisión 20, parche 8. Las versiones señaladas con números impares, por ejemplo 2.5, son núcleos en desarrollo.
- Como regla general, se debería poder actualizar un núcleo fácilmente a otro nivel de revisión. Sin embargo, actualizar a una nueva versión requiere a su vez la actualización de las utilidades del sistema que interactúan íntimamente con el núcleo.

- Si se pretende ajustar el funcionamiento del núcleo para dar soporte a un nuevo dispositivo de entrada/salida, o si se quiere actualizar a una versión superior, habría que hacer lo siguiente:
  - Situarse en el directorio que contiene el código fuente del núcleo (**/usr/src/linux-*versión*/**) y para configurar se puede ejecutar **make config** (mediante un guión) o **make menuconfig** (mediante una serie de menús), ambos en línea de comandos, los cuales guardan la configuración en el fichero **/usr/src/linux-*versión*/.config**.
  - A continuación hay que recompilar el núcleo. Para ello ejecutaremos **make dep** para determinar los ficheros que han cambiado y los que necesitan ser compilados de nuevo. Después haremos **make clean** para eliminar los ficheros antiguos que no son necesarios y ejecutaremos **make bzImage** para crear la imagen de arranque y guardarla en disco.
  - Si hemos configurado alguna parte del núcleo como un módulo (fichero objeto que se puede insertar con **insmod** o eliminar con **rmmod** sin tener que reiniciar), deberemos ejecutar **make modules** para compilarlos y **make modules_install** para instalarlos en el directorio **/lib/modules/**. Para ver los módulos instalados introducir **lsmod**.
  - Para copiar el nuevo núcleo al directorio de arranque ejecutaremos **make install**. Si queremos crear un disquete de arranque ejecutaremos **make bzdisk**.
- Para aplicar algún parche habrá que copiarlo al directorio del código fuente **/usr/src/** y descomprimirlo con **tar zxvf parche.tgz**. Después ejecutaremos **patch -p0 <parche >errores**. A continuación realizaremos una limpieza de ficheros innecesarios eliminando la configuración anterior con **make mrproper**, y recompilaremos el núcleo con **make bzImage** y **make install**.
- Para instalar un nuevo núcleo habrá que copiarlo al directorio del código fuente **/usr/src/** y descomprimirlo con **tar zxvf kernel.tgz**. A continuación realizaremos una limpieza de ficheros innecesarios eliminando la configuración anterior con **make mrproper**, y recompilaremos el núcleo con **make bzImage** y **make install**.

4.3. Administración de usuarios y grupos

- Para añadir un usuario al sistema habría que hacer lo siguiente:
  - Añadir una entrada en **/etc/passwd** rellenando los campos expuestos anteriormente.
  - Añadir el usuario a la lista de usuarios que se pueden unir al grupo con **newgrp** en la entrada correspondiente al GID asignado.
  - Crear el directorio de arranque del usuario y cambiarle el propietario y el grupo por el del usuario con **chown** y **chgrp**.
  - Copiar en el directorio de arranque todos los ficheros de configuración necesarios (los definidos por defecto están en **/etc/skel**), y cambiarles los derechos de acceso, el propietario y el grupo.
  - Comprobar con **pwck** y **grpck** si existe alguna inconsistencia en **/etc/passwd** y **/etc/group** respectivamente.
- Para eliminar físicamente un usuario basta con borrar su entrada en **/etc/passwd** y su directorio de arranque. También se le puede dar de baja lógicamente poniéndole un asterisco (*) como primer carácter en la contraseña cifrada del fichero **/etc/passwd**. También es posible desactivar

la cuenta asignándole un shell nulo (**/dev/null**) y utilizar **chsh -s** para que se visualice un mensaje al entrar.

- Tanto el alta como la baja de usuarios lo realizan automáticamente los comandos **adduser** y **deluser** respectivamente.

4.4. Sistema de ficheros

- Es común en UNIX tener un sistema distribuido en varios discos con varias particiones y sistemas de ficheros diferentes. En Linux se crean como mínimo una partición para el sistema y otra para el fichero swap de memoria virtual. Pero para UNIX sólo existe un único disco lógico.
- Para crear un nuevo sistema de ficheros hay que realizar lo siguiente:
  - Si se ha conectado un nuevo disco hay que crear el fichero de dispositivo correspondiente con el comando **mknod**.
  - Después habrá que crear particiones en el nuevo disco con **fdisk** o **cfdisk**.
  - A continuación hay que formatear las particiones para que acojan un sistema de ficheros UNIX, es decir, que tengan un sector de arranque, un superbloque, una lista de inodos y bloques de ficheros → **mkfs [-ct sist_ficheros] dispositivo tamaño** (**dispositivo** es el fichero de dispositivo creado en **/dev**; **tamaño** es el número de bloques del sistema de ficheros; **-c** comprueba que todos los bloques están bien; **-t sist_ficheros** indica el tipo de sistema de ficheros a crear, si es admitido.
  - Por último hay que conectar el nuevo sistema de ficheros al sistema general. Esto se realiza con la llamada al sistema **mount [-t sist_ficheros]**, y debe hacerse con un directorio vacío en la estructura del árbol existente (punto de montaje) → **mount /dev/hd1 /mnt** conecta el disco hd1 con el directorio **/mnt**. El núcleo actualiza la tabla de montajes añadiéndole una entrada con los siguientes datos:
    - Número de dispositivo del sistema montado.
    - Puntero al superbloque del sistema montado.
    - Puntero al inodo raíz del sistema montado.
    - Puntero al inodo del directorio punto de montaje.
- Para desconectar un dispositivo, primero hay que comprobar que no tiene ningún fichero abierto (con **fuser**) y que nadie lo tenga como directorio de trabajo → **umount /dev/hd1**.
- El fichero **/etc/fstab** guarda todos los sistemas de ficheros que deben montarse en el arranque y cada entrada representa un sistema distinto. Tiene la siguiente estructura:

  dispositivo directorio_punto_montaje tipo_sistema_ficheros opciones_montaje
- Algunos comandos para la administración del sistema de ficheros son los siguientes:
  - **fsck [-t tipo] [-alr] dispositivo** → Utilidad de recuperación de sistemas de ficheros dañados (**-a** repara automáticamente; **-l** lista el nombre de todos los ficheros; **-r** pregunta antes de reparar).
  - **df [-i] [dispositivo]** → Muestra información del espacio libre en todos los dispositivos o en el dispositivo especificado (**-i** informa sobre inodos libres y en uso).

- **du [directorio]** → Informa del número de bloques que ocupa el directorio con todos sus ficheros y subdirectorios.

4.5. Impresoras

- El fichero **/etc/printcap** contiene información sobre las impresoras conectadas al sistema. En él se indica el nombre, el dispositivo y el directorio de la cola de impresión de cada impresora.
- Para definir una impresora por defecto hay que añadir en **/etc/profile** (todos los usuarios) o en **~nombre_usuario/.bash_profile** (usuario concreto) el comando "PRINTER=nombre_impr export PRINTER".
- Comandos relacionados con operaciones con impresoras:
  - **lpr -p nombre_impr fichero** → Imprime "fichero" en "nombre_impr".
  - **lpq -p nombre_impr** → Muestra el estado de la cola de impresión de "nombre_impr".
  - **lprm -p nombre_impr núm_job** → Borra el "núm_job" de la cola de "nombre_impr".
  - **lpc comando nombre_impr [núm_job] [nombre_usuario]** → Controla la impresora "nombre_impr " y sus colas de impresión. Los comandos disponibles son:
    - **enable/disable** → Permite/prohíbe el envío de peticiones de impresión.
    - **up/down** → Activa/detiene la impresora.
    - **restart/abort** → Reinicia/detiene el demonio de impresión (**lpd**).
    - **clean** → Elimina ficheros temporales.
    - **status** → Visualiza el estado de la impresora.
    - **exit** → Sale de **lpc**.

4.6. Servicios de red

4.6.1. Introducción

- El programa-demonio que se encarga de suministrar los servicios de red en **inetd** o **xinetd**, cuyo fichero de configuración es **/etc/inetd.conf** o **/etc/xinetd.conf** respectivamente. Los tipos de servicios proporcionados se encuentran en **/etc/services**, mientras que los protocolos soportados se encuentran en **/etc/protocols**.
- Cada línea de **/etc/inetd.conf** o **/etc/xinetd.conf** está formada por lo siguiente:

  servicio tipo_socket protocolo wait|nowait usuario programa opciones

  en el que tipo_socket puede ser stream, dgram, raw, rdm o seqpacket; usuario es con el que se ejecuta el programa que da el servicio (no es necesario que sea root). Por ejemplo:

  ftp stream tcp nowait root /usr/sbin/tcpd wu.ftpd -a
- Otros ficheros implicados son los siguientes:
  - **/etc/sysconfig/network** → Contiene la información de la red en la que se encuentra un host, con los siguientes parámetros:
    - **NETWORKING=YES|NO** → Configura o no los dispositivos de red.
    - **HOSTNAME=hostname** → Nombre completo del host.

        - **GATEWAY=dir_IP** → Dirección IP de la puerta de enlace.
    - **/etc/init.d/network** → Inicializa todos los dispositivos de red configurados.
    - **/etc/sysconfig/network-scripts** → Directorio que contiene una serie de scripts utilizados por el sistema para configurar los dispositivos de red.

4.6.2. Configuración DNS

- Configuración de un servidor primario DNS:
    - **/etc/named.conf** → Fichero leído por el demonio servidor DNS **named**, especifica el nombre del dominio del cual es servidor, el directorio donde están los ficheros DNS (que suele ser **/var/named**) y los ficheros en los que se especifican la resolución directa e inversa de nombres y direcciones IP.
    - **/var/named/db.dominio** → Especifica la conversión del nombre de ordenadores del dominio a direcciones IP.
    - **/var/named/db.dir_IP_red** → Especifica la conversión de la dirección IP de ordenadores del dominio a nombres.
    - **/var/named/db.cache** → Proporciona información sobre los servidores raíz del DNS.
    - **/var/named/db.127.0.0.local** → Especifica la interfaz de loopback del servidor.
- Configuración de un servidor secundario DNS:
    - Se necesitan **/var/named/db.127.0.0.local**, **/var/named/db.cache** y **/etc/named.conf**. En este último se indicarán la dirección IP del servidor primario del cual se deben leer las tablas con las direcciones y nombres del dominio del que se es servidor secundario, y los nombres de los ficheros en los que deberán guardarse.
- Configuración de un cliente DNS:
    - **/etc/hosts** → En redes pequeñas puede utilizarse en lugar de un servidor DNS para especificar los nombres de todos los equipos conectados a la red y sus correspondientes direcciones IP.
    - **/etc/networks** → Parecido a **/etc/hosts**, relaciona nombres de dominio con sus correspondientes direcciones de red.
    - **/etc/hosts.conf** → Indica si primero debe mirarse el fichero **/etc/hosts** y después consultar al servidor DNS, o al contrario para resolver un nombre. La entrada típica es "order hosts, bind", primero mira el fichero local y después el servidor.
    - **/etc/resolv.conf** → Especifica un nombre de dominio al que pertenece el host (parámetro **domain**) para hacer búsquedas de nombres pertenecientes a dicho dominio sin tener que escribir el nombre completo. También indica las direcciones IP (parámetro **nameserver**) de los servidores que podemos consultar.

4.6.3. Configuración DHCP

- Configuración de un cliente DHCP:
    - En primer lugar, para habilitar el uso de los archivos de configuración del directorio **/etc/sysconfig/network-scripts** hay que modificar el archivo **/etc/sysconfig/network**.

Este archivo debería contener la línea **NETWORKING=yes** para iniciar la red en el momento del arranque.

- En el directorio **/etc/sysconfig/network-scripts** existe el archivo de configuración **ifcfg-eth0** donde eth0 es el nombre de la tarjeta de red. Este archivo debe contener:
  - **DEVICE=eth0**
  - **BOOTPROTO=dhcp**
  - **ONBOOT=yes**
- Es necesario un archivo de configuración para cada dispositivo que se desee configurar para el uso de DHCP.

– Configuración de un servidor DHCP:

- Es necesario que se encuentre en la misma subred que los hosts a los que va a asignar las direcciones IP, y que esté activada la opción multicast (comprobar con **ifconfig -a**).
- Hay que añadir la ruta de la dirección broadcast para poder devolver la respuesta al host solicitante con **route add -host 255.255.255.255 dev interfaz**.
- Debe crearse el fichero **/etc/dhcp.conf** para guardar los parámetros de configuración del demonio DHCP, que son:
  - Tiempo que se concede la IP por defecto.
  - Tiempo máximo que se concede la IP por defecto.
  - Máscara de red.
  - Dirección de broadcast.
  - Puerta de enlace.
  - Servidores DNS.
  - Nombre de dominio.
  - Rango de direcciones disponibles.
- Debe crearse el fichero **/var/state/dhcp/dhcpd.leases** en vacío para guardar la información sobre las direcciones IP temporales concedidas, y ejecutar el demonio **dhcpd** e incluirlo en la lista de servicios a arrancar en el inicio del sistema.

4.6.4. Configuración NFS (Network File System)

– Permite compartir ficheros entre hosts de una red UNIX.

– Configuración de un servidor NFS:

- **/etc/exports** → Contiene la lista con los volúmenes que se van a compartir y cómo se compartirán. Cada fila del fichero tiene el siguiente aspecto:

  directorio_a_compartir IP_host_con_acceso (opciones)

  directorio_a_compartir IP_red_con_acceso/máscara_red (opciones)

  Las opciones pueden ser RO (sólo lectura) o RW (lectura y escritura).
- **/etc/hosts.allow** y **/etc/hosts.deny** → Indican los servicios ofrecidos, y los ordenadores que pueden acceder a dichos servicios y los que no pueden acceder respectivamente. Primero se comprueba **/etc/hosts.allow** y después **/etc/hosts.deny**. Si un ordenador no está en ninguno de los dos se asume que se le permite acceder al servicio.

Normalmente **/etc/hosts.deny** suele tener como única línea ALL:ALL para negar el acceso a cualquier ordenador que no esté en **/etc/hosts.allow**. En este último hay que incluir los servicios propios de NFS (portmap, lockd, mountd, rquotad, statd) y asociarlos a las direcciones IP incluidas en **/etc/exports**.

- Deben lanzarse los siguientes demonios por este orden: **portmap**, **rpc.mountd**, **rpc.nfsd**, **rpc.statd**, **rpc.lockd** y **rpc.rquotad**.

– Configuración de un cliente NFS:
  - Es necesario que estén ejecutándose **portmap**, **rpc.statd** y **rpc.lockd**.
  - Después hay que montar el directorio exportado del servidor en un directorio local.

    mount dirección_IP_servidor:directorio_exportado directorio_local
  - Se puede montar el directorio en el arranque del sistema añadiendo una línea al fichero **/etc/fstab** con la instrucción anterior, al igual que los dispositivos y unidades de disco.

4.6.5. Configuración Samba

– Permite compartir ficheros entre hosts de una red mixta UNIX/Windows.
– Para poner a disposición de ordenadores Windows recursos de Linux (servidor Samba):
  - Ejecutar el script **/etc/init.d/smb** para arrancar los demonios **smbd** y **nmbd**.
  - En **/etc/services** deben estar asignados a NetBIOS los puertos 137, 138 y 139.
  - Para configurar Samba hay que modificar el fichero **/etc/samba/smb.conf**, que determina cómo y qué recursos se compartirán y las restricciones aplicadas.
– Para poner a disposición de ordenadores Linux recursos de Windows (cliente Samba):
  - Se puede utilizar la aplicación **smbclient** para una tarea puntual. Para ver una lista de recursos disponibles introducir **smbclient -L dirección_IP_ordenador_Windows**.
  - Se pueden utilizar los programas **smbmount** y **smbumount** para montar y desmontar un volumen Windows como si fuera local, e incorporarlo a **/etc/fstab**.

4.6.6. Configuración NIS (Network Information System)

– Proporciona un punto de administración centralizado. Un servidor NIS se comporta de manera parecida a un servidor DNS, pero referido a la administración de cuentas de usuario en red.
– NIS proporciona una base de datos que se emplea para distribuir la información de acceso de los usuarios entre todos los ordenadores de la red, la cual se comporta como un único sistema con las mismas cuentas de usuario en cada uno de los ordenadores.
– Sólo puede haber un servidor NIS en cada dominio de una red.
– Configuración de un servidor NIS:
  - Al igual que un servidor DNS, puede configurarse como primario o como secundario.
  - Iniciamos el servicio mediante el programa **ypserv**, que puede arrancarse desde **/etc/init.d/ypserv**, cuyo fichero de configuración es **/etc/ypserv.conf**.
  - Después hay que crear un directorio para el dominio NIS **/var/yp/dominio_NIS** que contendrá los ficheros de configuración que el servidor distribuirá.

- Estos ficheros de configuración pueden crearse mediante la instrucción **make dbm**, incluida en el fichero Makefile del directorio **/var/yp**.

– Configuración de un cliente NFS:
  - Mediante la instrucción **ypset** se utiliza el programa **ypbind**, cuyo fichero de configuración es **/etc/yp.conf**, que se encarga de buscar el servidor NIS y de mantener el enlace con él.
  - En **/etc/yp.conf** se indica el dominio por defecto y el nombre del servidor NIS al que se conectará el host. Se pueden especificar varios dominios con sus correspondientes servidores: domain nombre_dominio [server servidor_NIS|broadcast]. Con la opción broadcast el host utiliza el servidor NIS que encuentre en la red.
  - Después hay que especificar los ficheros de configuración que importaremos mediante el fichero **/etc/nsswitch.conf**, en el que se indica qué ficheros de configuración del sistema utilizar: locales, NIS, DNS, etc. Normalmente se consulta el fichero de configuración local y después el ofrecido por el servidor NIS.

4.7. Medidas de seguridad

– Es necesario tomar una serie de medidas para evitar el acceso de los usuarios a ciertos ficheros y directorios estableciendo los siguientes permisos:
  - El directorio raíz debe estar en lectura-ejecución.
  - **/etc/passwd** sólo debe permitir la lectura.
  - **/dev** debe permitir en lectura-escritura sólo para el administrador.
  - Los directorios del sistema **/usr**, **/lib**, **/usr/lib** y **/etc** deben estar en lectura-ejecución.
  - El directorio **/tmp** no debe estar en ejecución para el grupo y el resto de usuarios.

– Los terminales desatendidos hay que desconectarlos, bien con el comando **lock** o bien con la variable shell TMOUT en **/etc/profile**.

– El administrador sólo se conectará como tal en los terminales seguros que especifique en el fichero **/etc/securetty**, y sólo lo imprescindible.

– Deben preservarse los datos almacenados en el sistema, planificando y realizando copias de seguridad de forma regular.

– Realizar auditorias periódicas del sistema con los comandos **netstat** y **top**.

– Desactivar los servicios no utilizados y potencialmente peligrosos como:
  - Los servicios NFS (nfsd, lockd, mountd, statd, portmap).
  - Los servicios RPC y NIS (rsh, rlogin, rexec, rpc). Es aconsejable utilizar el servicio de Secure Shell para accesos remotos (demonio **sshd**).
  - Los servicios Telnet y FTP. Es mejor utilizar **scpd** o **httpd** si es posible.
  - Los servicios POP, IMAP y SMTP.

  Para ello se utiliza el comando **chkconfig**, que indica qué servicios se encuentran configurados en el sistema para ejecutarse. En **/etc/services** hay una descripción de los servicios posibles y los puertos utilizados. Para actuar sobre los servicios iniciados:
  - **/etc/init.d/nombre_serv stop** o **chkconfig nombre_serv off** → Detiene nombre_serv.

- **/etc/init.d/nombre_serv start** o **chkconfig nombre_serv on** → Arranca nombre_serv.

Hay que tener en cuenta el fichero **/etc/inetd.conf** o **/etc/xinetd.conf**, que es utilizado por **inetd** o **xinetd** respectivamente para lanzar diversos servicios.

- Mantener permanentemente actualizado el sistema.
- Limitar los recursos del sistema fijando cuotas de disco para los usuarios.
- Es aconsejable emplear varios login de usuario para realizar diferentes tareas de administración: root para administración, daemon para administración automática, bin para poseer la mayoría de los ficheros generales...
- La política de seguridad debe estar perfectamente definida y documentada.
- Es necesario vigilar a los usuarios potencialmente peligrosos.
- Se debe eliminar de la variable PATH del administrador el directorio actual.
- Es necesario buscar regularmente en todo el sistema ficheros cuyo propietario sea "root" y que tengan el bit set-uid activo.
- Utilizar el fichero **/etc/shadow** protegido contra lectura (excepto para el administrador) en vez de **/etc/passwd** para guardar las contraseñas de los usuarios.
- Para evitar ataques de denegación de servicio (DoS) es necesario controlar el acceso a los servicios, limitando el número de peticiones que se pueden recibir y restringiendo el acceso a máquinas fiables, si es posible.
- Existen ficheros de registro de actividades del sistema creados por el demonio **syslogd** utilizando el fichero de configuración **/etc/syslog.conf**, que se lanza automáticamente al arrancar el sistema. El fichero de configuración especifica las reglas a seguir para gestionar el almacenamiento de mensajes del sistema, que suelen guardarse en **/var/adm**. Algunos de los ficheros de registro son debug, lastlog, faillog, messages, sulog, syslog, wtmp, utmp, btmp...
- Software de seguridad:
  - TCP Wrappers → Filtro entre la petición de un servicio y la ejecución del programa que proporciona ese servicio. El programa **tcpd** lee los ficheros **/etc/hosts.allow** y **/etc/hosts.deny**, formados por líneas que tienen el siguiente formato:

    lista_demonios : lista_máquinas [: comando_shell]

    La lista de demonios consta del nombre de los programas que realizan un determinado servicio, o el parámetro ALL para indicar todos los servicios. La lista de máquinas permitidas puede tener los siguientes valores:
    - **.dominio.org** → Patrón de dominio (todas las pertenecientes a ese dominio).
    - **158.42.** → Patrón de red (todas las pertenecientes a esa red).
    - **ALL** → Todas las máquinas.
    - **LOCAL** → Todas las máquinas cuyo nombre no contiene ".".
    - **UNKNOWN** → Máquinas con nombre o dirección desconocidos.
    - **KNOWN** → Máquinas con nombre o dirección conocidos.
    - **PARANOID** → Máquinas con nombre que no corresponde con su dirección.

- SSH (Secure Shell) → Establece conexiones cifradas entre dos máquinas que utilizan Telnet, FTP o remote login. El servidor debe estar ejecutando el demonio **sshd** y el cliente debe utilizar un programa que envíe los datos cifrados al servidor.

5. Resumen de comandos

5.1. Comandos generales

- **exit** → Fin de sesión, libera terminal.
- **who [am i]** → Muestra usuarios conectados al sistema.
- **w [usuario]** → Muestra usuarios conectados al sistema y sus respectivos procesos.
- **mail [usuario]** → Recepción y envío de correo electrónico.
- **write usuario** → Envía mensaje a otro usuario.
- **mesg [yn]** → Autoriza (**y**) o deniega (**n**) la recepción de mensajes.
- **date** → Muestra la fecha y hora actuales en formato "Thu Apr 02 19:08:49 MST 1992".
- **echo** → Muestra por pantalla una cadena de caracteres o el contenido de variables shell.
- **cal [mes] [año]** → Muestra el calendario del mes actual por defecto o el indicado.
- **uname [-a]** → Da información sobre el sistema UNIX (**-a** muestra toda).
- **passwd [usuario]** → Modifica la clave de acceso de un usuario.
- **lock** → Bloquea el terminal con una clave distinta a la clave de acceso.
- **man [-k término] [comando]** → Proporciona información acerca del comando o del término.
- **pwd** → Muestra el directorio actual en camino absoluto.
- **which fichero** → Busca en los directorios especificados en el PATH del usuario el fichero, devolviendo el camino absoluto de acceso.
- **whereis comando** → Devuelve el directorio donde reside el comando.
- **id [usuario]** → Devuelve el número de usuario (UID) y el número de grupo (GID) del usuario, o del actual si no se indica.
- **su [usuario]** → Permite cambiar al usuario indicado continuando con la sesión actual mediante la creación de una subshell hija.
- **newgrp [grupo]** → Permite cambiar al grupo indicado continuando con la sesión actual.
- **chsh [shell]** → Permite cambiar el fichero **/etc/passwd** para arrancar con un shell diferente.
- **chfn [usuario]** → Permite cambiar la información (nombre completo, teléfono, etc...) de una cuenta de usuario.

5.2. Comandos de manipulación de ficheros y directorios

- **ls [-lFAd] [ficheros]** → Lista el contenido de un directorio o muestra información de ficheros (**-l** añade más información; **-F** añade al final del nombre "/" a los directorios, "@" a los enlaces simbólicos y "*" a los ficheros ejecutables; **-A** lista todos los ficheros; **-d** muestra los nombres de los directorios sin listar su contenido).
- **cd camino** → Cambia de directorio actual indicando un camino absoluto o relativo al actual.

- **mkdir [-p] directorio** → Crea un directorio dentro del directorio actual (**-p** crea directorios intermedios automáticamente).
- **rmdir directorio** → Borra un directorio vacío dentro del directorio actual.
- **copy directorio1 directorio2** → Copia el contenido de un directorio a otro.
- **more fichero** → Muestra un fichero de texto pantalla a pantalla hacia adelante.
- **less fichero** → Muestra un fichero de texto pantalla a pantalla hacia delante y hacia atrás.
- **head [-N] fichero** → Muestra las N primeras líneas de un fichero de texto.
- **tail [-N] fichero** → Muestra las N últimas líneas de un fichero de texto.
- **cp [-r] ficheros destino** → Copia fichero(s) origen al directorio o fichero destino según un camino relativo o absoluto (**-r** copia directorios enteros con su contenido).
- **mv ficheros destino** → Mueve fichero(s) origen al directorio o fichero destino. También renombra ficheros y directorios si el origen y el destino son los mismos.
- **ln [-s] enlace destino** → Crea un enlace físico sobre el fichero destino (**-s** crea un enlace lógico).
- **rm [-r] ficheros** → Borra ficheros de forma irrecuperable, el fichero permanece en el sistema hasta que no se borren todos sus enlaces físicos (**-r** borra también subdirectorios dependientes).
- **file ficheros** → Muestra información sobre el tipo de fichero.
- **touch fichero** → Crea un fichero vacío, si ya existe le cambia la fecha y hora de modificación.
- **chmod modo ficheros** → Modifica los permisos de lectura, escritura y ejecución del propietario de los ficheros, del grupo del propietario y del resto de usuarios según una cadena de nueve bits por este orden expresada en octal en que 1 activa permiso y 0 lo desactiva.
- **umask [máscara]** → Muestra los permisos asignados por defecto a los ficheros nuevos. Para cambiar se debe restar al modo por defecto el modo deseado en octal y pasarlo como argumento.
- **chown usuario ficheros** → Cambia el propietario de los ficheros al usuario indicado.
- **chgrp grupo ficheros** → Cambia el grupo de los ficheros al grupo indicado.
- **find ruta -criterio -acción** → Busca dentro de la ruta todos los ficheros que se ajusten al criterio y realiza la acción especificada sobre ellos. La búsqueda de ficheros parte del directorio que indicamos como ruta y el criterio se puede ajustar a lo siguiente:
  - **-name fichero** → Ficheros que tengan el nombre indicado.
  - **-user usuario** → Ficheros que pertenezcan al usuario indicado.
  - **-group grupo** → Ficheros que pertenezcan al grupo indicado.
  - **-size +m** → Ficheros cuyo tamaño sea mayor a "m" bloques.
  - **-size -m** → Ficheros cuyo tamaño sea menor a "m" bloques.
  - **-mtime +d** → Ficheros modificados hace más de "d" días.
  - **-mtime -d** → Ficheros modificados hace menos de "d" días.
  - **-mtime d** → Ficheros modificados hace "d" días.
  - **-type d** → Busca directorios.
  - **-type f** → Busca ficheros ordinarios.

  Si se especifican varios criterios, se buscarán los ficheros que cumplan todos. Si se separan con el operador **-o** se buscarán los ficheros que cumplan cualquiera de ellos. En los criterios de búsqueda también se permite utilizar metacaracteres y caracteres especiales.

La acción a realizar puede ser sacar la ruta absoluta de los ficheros encontrados con **-print**, o ejecutar un comando determinado sobre los mismos con **-exec comando {} \;** siendo las llaves el argumento del comando y representando éstas a los ficheros encontrados.

5.3. Comandos de shell scripts

- **shift n** → Desplaza los argumentos de la entrada "n" veces a la izquierda. Los "n" primeros argumentos se pierden, afectando a las variables shell generales # (número de argumentos de entrada) y * (cadena de argumentos de entrada).
- **read var1, var2...** → Se lee una línea de la entrada estándar y la primera cadena de caracteres se asigna a "var1", la segunda cadena a "var2"... asumiendo como separador de cadenas de caracteres el espacio.
- **expr arg1 op1 arg2 [op2 arg3…]** → Evalúa los argumentos, les aplica los operadores y escribe el resultado en la salida estándar. Se suele utilizar para lo siguiente:
  - **Cálculos numéricos** → Utilizando los operadores +, -, *, / y %. Las expresiones pueden agruparse con ( ), teniendo en cuenta que los caracteres *, (, y) son especiales para el shell y deben ir entrecomillados.
  - **Comparaciones** → Utilizando los operadores =, !=, <, >, <= y >=. También pueden compararse palabras con estos operadores.
  - **Operaciones lógicas**→ Utilizando los operadores | (or), & (and) y : (arg1:arg2 devuelve el número de caracteres de arg1 que coinciden con la expresión regular arg2).
- **test -opción fichero / [-opción fichero]** → Evalúa si existe el fichero y si cumple con la característica indicada con el parámetro opción:
  - **-e** → Comprueba si el fichero existe.
  - **-f** → Comprueba si es un fichero ordinario.
  - **-d** → Comprueba si es un directorio.
  - **-L** → Comprueba si es un enlace simbólico.
  - **-s** → Comprueba si tiene un tamaño mayor que cero.
  - **-r** → Comprueba si tiene permiso de lectura.
  - **-w** → Comprueba si tiene permiso de escritura.
  - **-x** → Comprueba si tiene permiso de ejecución.
  - **-u** → Comprueba si tiene el bit setuid activado.
  - **-g** → Comprueba si tiene el bit setgid activado.
  - **-O** → Comprueba si somos propietarios del fichero.
  - **-G** → Comprueba si pertenecemos al grupo del fichero.
  - **-z cadena_caracteres** → Comprueba si la longitud de cadena_caracteres es cero.
  - **-n cadena_caracteres** → Comprueba si la longitud de cadena_caracteres no es cero.

  Devuelve 0 si se cumple la expresión y 1 si no se cumple. Los siguientes operadores pueden utilizarse para crear expresiones de evaluación más complejas: **-o** (or), **-a** (and) y **!** (not).
- **test cad1 oper cad2 / [cad1 oper cad2]** → Compara cadenas de caracteres expresadas entre comillas dobles, o el contenido de variables shell. Los operadores pueden ser los siguientes: =,

!=, < y >. Devuelve 0 si se cumple la expresión y 1 si no se cumple. Los siguientes operadores pueden utilizarse para crear expresiones de evaluación más complejas: **-o** (or), **-a** (and) y **!** (not).

- **test num1 oper num2 / [num1 oper num2]** → Compara números enteros, o el contenido de variables shell. Los operadores pueden ser los siguientes: -eq, -ne, -lt, -le, -gt y -ge. Devuelve 0 si se cumple la expresión y 1 si no se cumple. Los siguientes operadores pueden utilizarse para crear expresiones de evaluación más complejas: **-o** (or), **-a** (and) y **!** (not).
- **if cond1; then ord1 [elif cond2; then ord2] [else comandos3] fi** → Si se cumple la primera condición se ejecuta "ord1", si se cumple la segunda condición se ejecuta "ord2", si no se cumple ninguna de las anteriores se ejecuta "ord3". Las condiciones se evalúan con test
- **while condición; do comandos done** → Mientras se cumpla la condición evaluada con test, se ejecutarán los comandos especificados.
- **until condición; do comandos done** → Hasta que se cumpla la condición evaluada con test, se ejecutarán los comandos especificados.
- **for variable in lista; do comandos done** → Se repite la ejecución de los comandos tantas veces como elementos separados por espacios o tabuladores haya en "lista". En cada repetición "variable" toma el valor de uno de los elementos que forman "lista".
- **case variable in patrón1) ord1;; patrón2) ord2;; *) ord3;; esac** → Si la variable se ajusta al patrón1 se ejecuta "ord1", si se ajusta al patrón2 se ejecuta "ord2", si no se ajusta a ninguno de los anteriores se ejecuta "ord3". Los patrones pueden ser expresiones regulares.
- **break [n]** → Interrumpe una repetición en un bucle y sale "n" niveles.
- **continue [n]** → Sala a la "n" siguiente repetición del bucle.
- **exit [n]** → Detiene la ejecución del script con el código "n".
- **return [n]** → Sale de una función devolviendo el código "n".

5.4. Filtros

- **cat [ficheros]** → Muestra ficheros de texto por pantalla, si no se indica fichero lee la entrada estándar hasta que se pulsa <Ctrl-d> y vuelca el contenido en la salida estándar.
- **pg [ficheros]** → Muestra ficheros de texto pantalla a pantalla con avance y retroceso.
- **od [-bc] [ficheros]** → Realiza un volcado octal de cualquier fichero de texto por pantalla, si no se indica fichero lee la entrada estándar hasta que se pulsa <Ctrl-d> y vuelca el contenido en la salida estándar (**-b** muestra los bytes en octal; **-c** muestra los bytes como caracteres).
- **sort [-nrtk] [ficheros]** → Ordena alfabética y ascendentemente las líneas de ficheros cuyos campos estén separadas por un tabulador "\t" (**-n** ordena numéricamente; **-r** ordena descendentemente; **-t sep** especifica "sep" como carácter de separación; **-k cam1,cam2** ordena respecto a los campos cuyo número se encuentra entre "cam1" y "cam2", ambos inclusive; **-k 1.pos1,1.pos2** ordena respecto a los caracteres situados entre las posiciones "pos1" y "pos2", ambos inclusive). Ejemplo: "sort -t+ -k5,5r –k1.1,1.3n fichero" ordena descendente y alfabéticamente por el campo 5 del fichero, cuyos campos están separados por "+", y ascendente y numéricamente por los tres primeros caracteres.

- **uniq [-ucd] [ficheros]** → Lee la entrada estándar o los ficheros de texto especificados y filtra las líneas repetidas adyacentes, útil para las salidas de **sort** (**-u** muestra sólo una de las líneas idénticas adyacentes; **-c** hace que en cada línea aparezca el número de ocurrencias adyacentes; **-d** muestra sólo las líneas repetidas).
- **egrep [-linv] 'patrón' [ficheros]** → Busca cadenas de caracteres que respondan al patrón especificado en los ficheros indicados y muestra las líneas coincidentes (**-l** muestra sólo los nombres de los ficheros obtenidos como resultado; **-i** ignora la diferencia entre mayúsculas y minúsculas; **-n** indica el número de línea donde se encontró la coincidencia; **-v** busca cadenas de caracteres que no respondan al patrón). El patrón puede ser una cadena de caracteres o una expresión regular entre comillas simples.
- **wc [-lwc] [ficheros]** → Cuenta las líneas, palabras y caracteres de ficheros de texto (**-l** cuanta sólo líneas; **-w** cuenta sólo palabras; **-c** cuenta sólo caracteres). Asume como separación de línea el retorno de carro "\n", y como separación de palabra el espacio en blanco, el tabulador "\t" y el retorno de carro "\n".
- **cut -c pos1-pos2 [ficheros]** → Corta y envía a la salida estándar los caracteres situados entre las posiciones "pos1" y "pos2", ambos inclusive, de todas las filas de los ficheros indicados. El primer carácter se identifica con la posición 1.
- **cut [-d sep] -f cam1-cam2 [ficheros]** → Corta y envía a la salida estándar los campos delimitados por el tabulador "\t" (**-d** indica que el carácter de separación es "sep"), cuyo número está comprendido entre "cam1" y "cam2", ambos inclusive, de todas las filas de los ficheros indicados. El primer campo se identifica con el número 1.
- **tr [-dsc] cadena1 [cadena2]** → Lee la entrada estándar y sustituye los caracteres incluidos en "cadena1" por los de "cadena2" (**-d** borra todos los caracteres incluidos en "cadena1"; **-s** elimina los caracteres repetidos que estén dentro de "cadena1"; **-c** sustituye los caracteres no incluidos en "cadena1" por los de "cadena2"). Tanto "cadena1" como "cadena2" pueden ser analista de caracteres simples o un rango si están separados por "-". Ejemplo tr '\n' ' ' <fichero sustituye los caracteres fin de línea por espacios en blanco.
- **tee [-a] fichero** → Lee la entrada estándar, y vuelca su contenido en "fichero" y en la salida estándar (**-a** añade al contenido del fichero en vez de sobreescribirlo).

5.5. Comandos para el manejo de variables shell

- **export [variable]** → Traslada la variable desde el área local de datos al entorno, sin parámetros informa las variables trasladadas en el shell actual.
- **set** → Informa de las variables del shell actual en el área local y en el entorno.
- **unset [variable]** → Anula la variable del área local de datos, sin parámetros anula todas.
- **env** → Informa de las variables del shell actual en el entorno.

5.6. Comandos para el manejo de procesos

- **ps [-lef] [-u usuario] [-g grupo]** → Informa de los procesos que se están ejecutando en el sistema (**-l** da toda la información disponible; **-e** informa de todos los procesos; **-f** informa más de cada proceso; **-u** informa de los procesos lanzados por un determinado usuario; **-g** informa de los procesos lanzados por un determinado grupo de usuarios).
- **kill [-n] PIDs** → Envía una señal con el código numérico "n" (en Linux puede ser del 1 al 31) al proceso o procesos identificados por su número de proceso (PID). También puede utilizarse como llamada al sistema para la comunicación entre procesos.
- **trap ['comandos'] n1 [n2...]** → Hace que un proceso ejecute los comandos especificados cuando le llegan las señales con los códigos numéricos "n1", "n2"... Para ignorar una señal los comandos deben ser un espacio en blanco entre comillas dobles, para restaurar la acción por defecto no deben indicarse comandos.
- **nice [-n] comandos** → Permite cambiar la prioridad en la ejecución de un proceso. "n" puede ir de –19 (mayor prioridad) a +19 (menor prioridad). Sólo el administrador puede aumentar la prioridad de un proceso.
- **nohup comandos &** → Hace que un proceso en segundo plano no termine en caso de finalizar el shell que lo lanzó (desconexión de terminal). Por defecto la salida y los posibles errores se redireccionan hacia el fichero **nohup.out**.
- **jobs** → Muestra los procesos y sus respectivos números que se ejecutan en segundo plano.
- **fg** → Reinicia en primer plano un proceso suspendido con <Ctrl-z> indicando su número.
- **bg** → Reinicia en segundo plano un proceso suspendido con <Ctrl-z> indicando su número.

5.7. Comandos especiales

- **sed [-n] [-f fichord] [órdenes] [ficheros]** → Editor que por defecto lee en la entrada estándar ficheros de texto y los copia en la salida estándar, procesándolos línea a línea según las órdenes indicadas en un fichero o en la línea de comandos (**-n** hace que no se copien los ficheros de entrada a la salida estándar; **-f** indica el fichero donde se encuentran las órdenes). Las órdenes soportan expresiones regulares (encerradas entre / / y entrecomilladas con comillas simples o dobles según convenga) para manipular ciertas líneas, por ejemplo:
  - **'s/Walter/Jaime/g'** → Sustituye la palabra "Walter" por "Jaime" en todo el fichero.
  - **'y/aeiou/AEIOU'** → Sustituye los caracteres correspondientes a las vocales minúsculas por las vocales mayúsculas.
  - **'/^F/p'** → Saca en pantalla todas las líneas que empiecen por "F".
  - **'/^t/,$p'** → Saca en pantalla desde la línea que empieza por "t" hasta el final de fichero.
  - **'2,5d'** → Elimina desde la línea 2 hasta la línea 5.
  - **'/^a-q/d'** → Elimina líneas cuyo primer carácter esté comprendido entre a y q.
- **awk [-F sep] [-f fichord] ['[patrón] [{acciones}]'] [ficheros]** → Lenguaje de procesamiento de ficheros de texto, por defecto lee en la entrada estándar y procesa los ficheros especificados línea a línea. El patrón es opcional, si existe se compara la línea procesada con él y si coincide se

llevan a cabo las acciones indicadas, si no existe patrón se llevan a cabo sin restricción (**-F** indica que el carácter de separación de los campo es "sep"; **-f** indica el fichero donde se encuentran las órdenes). Si no se indica ninguna acción se visualiza la línea procesada. Los patrones que se reconocen son los siguientes:

- **BEGIN →** Acciones a realizar antes de procesar los ficheros.
- **END →** Acciones a realizar después de procesar los ficheros.
- **Expresiones regulares →** Deben ir encerradas por **/ /**.
- **Expresiones relacionales →** Hacen uso de los operadores comunes en C.
- **Expresiones de coincidencia de patrones →** Comparan el contenido de un campo de los ficheros procesados con una determinada expresión regular, utilizando los operadores **~** (coincide) y **!~** (no coincide).
- **Comentarios →** Líneas precedidas por el carácter **#**.
- **Expresiones de combinación de patrones →** Mediante los operadores lógicos AND (&&), OR (||) y NOT (!).

Los operadores que se pueden utilizar en las acciones son los siguientes:

- **Operadores de cálculo →** +, -, *, /, %, ^
- **Operadores de asignación →** =, +=, -=, *=, /=, %=, ^=
- **Operadores condicionales →** expr1? expr2:expr3
- **Operadores lógicos →** &&, ||, !, ~,!~
- **Operadores relacionales →** <, >, <=, >=, ==, !=
- **Operadores incrementales/decrementales →** ++, --

Se puede trabajar con matrices definiéndolas de la misma forma que en C (matriz[índice]), aunque no es necesario declararlas. Se mantienen una serie de variables de uso interno:

- **FILENAME →** Nombre del fichero procesado o entrada estándar (en este caso -).
- **FNR →** Número de la línea que se está procesando.
- **FS →** Carácter de separación de campos de entrada (por defecto espacio).
- **RS →** Carácter de separación de líneas de entrada (por defecto "\n").
- **NF →** Número de campos existentes en la línea que se está procesando.
- **NR →** Número de líneas que se han procesado.
- **OFS →** Carácter de separación de campos de salida (por defecto espacio).
- **ORS →** Carácter de separación de líneas de salida (por defecto "\n").
- **$0 →** Representa la línea procesada.
- **$n →** Representa el campo n de la línea procesada.

Existen también sentencias de control de flujo análogas a las de C:

- **if (condición) {órdenes} [else {órdenes}]**
- **do {órdenes} while (condición)**
- **while (condición) {órdenes}**
- **for (condición_inicio;condición_finalización;incremento) {órdenes}**
- **break, continue, exit**

Para la impresión de líneas se pueden utilizar los siguientes comandos (análogos a los de C):

- **print("cadena_caracteres" var1;var2)** → Los literales deben ir entre comillas dobles. Los argumentos separados por ";" se visualizan como campos separados mientras que los separados por espacios aparecen concatenados.
- **printf("cadena_con_caracteres_de_formato",var1,var2)** → Los caracteres de formato más comunes son **\n** (nueva línea), **\t** (tabulador), **%s** (cadena de caracteres), **%d** (número decimal) y **%n,mf** (número en coma flotante de n dígitos y f decimales).

Existen una serie de funciones numéricas, y son las siguientes:

- **atan2(y, x)** → Arcotangente en radianes de "y/x".
- **cos(x)** → Coseno de "x" (x en radianes).
- **sin(x)** → Seno de "x" (x en radianes).
- **exp(x)** → Función exponencial.
- **int(x)** → Trunca el número "x" a un entero.
- **log(x)** → Logaritmo neperiano de "x".
- **rand()** → Devuelve un número aleatorio entre 0 y 1.
- **sqrt(x)** → Raíz cuadrada de "x".

Existen una serie de funciones de tratamiento de cadenas, y son las siguientes:

- **gsub(r, s, t)** → Sustituye la cadena que verifica la expresión regular "r" por la subcadena "s" en la cadena total "t".
- **index(t, s)** → Devuelve la posición de la subcadena "s" en la cadena total "t".
- **lenght(s)** → Devuelve la longitud de la cadena "s".
- **match(s, r)** → Devuelve la posición en la cadena "s" donde se verifica la expresión "r".
- **split(s, a, r)** → Divide la cadena "s" en cada ocurrencia definida por la expresión regular "r" dentro de cada elemento de la matriz "a".
- **substr(s, i, n)** → Devuelve la subcadena formada por "n" caracteres a partir de la posición "i" de la cadena original "s".
- **tolower(s)** → Convierte letras mayúsculas en minúsculas en la cadena "s".
- **toupper(s)** → Convierte letras minúsculas en mayúsculas en la cadena "s".

5.8. Comandos para realización de copias de seguridad

- **cpio -o** → Lee la entrada estándar para obtener una lista de ficheros, y los copia en la salida estándar, junto al estado de dichos ficheros. Utilizado para la realización de copias de seguridad. Ejemplo: **ls | cpio -o >/dev/rmt/0h** vuelca el contenido del directorio actual al dispositivo.
- **cpio -i [patrones]** → Extrae ficheros de la entrada estándar si coinciden con los patrones que pueden aparecer en los argumentos (por defecto el patrón es *, todos los ficheros). los ficheros extraídos serán condicionalmente creados y copiados en el directorio actual según las opciones del comando. Utilizado para recuperar información volcada en un dispositivo. Ejemplo: **cpio -i </dev/rmt/0h** vuelca el contenido del dispositivo al directorio actual.
- **cpio -p directorio** → Lee la entrada estándar para obtener una lista de ficheros que son creados y copiados según las opciones del comando en el directorio que aparece como argumento. Ejemplo: **find /etc | cpio -p /seguridad** copia ficheros de /etc a /seguridad.

- **tar [opciones] [ficheros]** → Guarda o recupera de un dispositivo de almacenamiento una serie de ficheros contenidos en un archivo con extensión **.tar**. Los archivos suelen ser ficheros de dispositivo correspondientes a discos o cintas. Las opciones están formadas por una letra de función seguida de modificadores de la función. Las letras de función pueden ser las siguientes:
  - **c** → Crea un nuevo archivo destruyendo lo que había.
  - **r** → Añade ficheros al final del archivo.
  - **t** → Lista los nombres de todos los ficheros del archivo.
  - **u** → Añade ficheros al final del archivo si este no existe o si ha sido modificado.
  - **x** → Extrae ficheros del archivo.

  Los modificadores de función son los siguientes:
  - **f archivo** → Los ficheros serán almacenados o serán recuperados de "archivo". Si éste es el carácter "-" se utilizara como fichero de dispositivo la entrada o salida estándar.
  - **l** → Muestra mensajes sobre los enlaces simbólicos no encontrados.
  - **m** → Provoca que no se actualice la fecha de modificación del archivo.
  - **p** → Obtiene los permisos, propietarios y grupos originales escritos en el archivo.
  - **v** → Modo verboso (escribe el nombre de cada fichero procesado).
  - **w** → Pide confirmación de la acción a realizar con cada fichero.
  - **L** → Sigue los enlaces simbólicos.
- **compress [opciones] ficheros** → Guarda o recupera ficheros de archivos que han sido comprimidos mediante el algoritmo de Lempel-Ziv. Cada fichero tratado con **compress** es reemplazado por su equivalente comprimido al que se añade la extensión **.Z** manteniendo los derechos originales y la fecha. Para descomprimir se utiliza **uncompress ficheros.Z**. Un archivo **.tar.Z** es lo mismo que un archivo **.taz**. Las opciones son las siguientes:
  - **-c** → Muestra por la salida estándar el nombre de los ficheros sin comprimir.
  - **-d** → Descomprime ficheros comprimidos.
  - **-v** → Modo verboso (escribe el nombre de cada fichero procesado).
- **gzip [opciones] ficheros** → Guarda o recupera ficheros de archivos que han sido comprimidos mediante el algoritmo de Lempel-Ziv. Cada fichero tratado con **gzip** es reemplazado por su equivalente comprimido al que se añade la extensión **.gz** manteniendo los derechos originales y la fecha. Para descomprimir se utiliza **gunzip ficheros.gz**. Suele comprimir mejor **gzip** que **compress**. Un archivo **.tar.gz** es lo mismo que un archivo **.tgz**. Las opciones son las siguientes:
  - **-c** → Muestra por la salida estándar el nombre de los ficheros sin comprimir.
  - **-d** → Descomprime ficheros comprimidos.
  - **-l** → Muestra información de los ficheros comprimidos.
  - **-r** → Incluye el contenido de directorios.
  - **-t** → Comprueba la integridad de ficheros comprimidos.
  - **-v** → Modo verboso (escribe el nombre de cada fichero procesado).
  - **-rel_compr** → Establece una relación de compresión (1 menor, 9 mayor, defecto 6).

5.9. Comandos para servicios de red

- **ifconfig interfaz_red dir_IP netmask máscara_red [up|down]** → Muestra la configuración (sin parámetros) y asigna una dirección IP a una interfaz de red del sistema. Ejemplo: **ifconfig eth0 192.168.0.5 netmask 255.255.255.0 up**.
- **route** → Muestra la configuración (sin parámetros) y añade o elimina rutas estáticas en un host:
  - **route [add|del] [default|-net 0.0.0.0] gw dir_IP_gw** → Añade o borra la puerta de enlace por defecto del host.
  - **route add [-host dir_IP_host|-net dir_IP_red] [netmask máscara_red] [dev interfaz]** → Añade una ruta estática host (**-host**) o de red (**-net**).
- **netstat** → Visualiza las rutas y el estado de las conexiones. Las opciones son las siguientes:
  - **-a** → Muestra todas las conexiones activas.
  - **-i** → Muestra todos los dispositivos de red.
  - **-c** → Actualiza la información cada segundo.
  - **-r** → Muestra la tabla de enrutado.
  - **-n** → Muestra las direcciones en formato numérico.
  - **-t** → Muestra sólo las conexiones TCP.
- **top** → Muestra en pantalla los procesos que más CPU consumen.
- **rpcinfo -p host** → Muestra los demonios que se están ejecutando en "host".
- **traceroute** → Muestra la ruta recorrida por un paquete hasta su destino.
- **systat** → Lista los procesos en ejecución de una máquina.
- **strobe** → Realiza un escaneo de los puertos de la máquina y muestra la información.
- **hostname** → Devuelve el nombre del ordenador UNIX al que estamos conectados en la red.
- **domainname** → Devuelve el dominio del ordenador UNIX al que estamos conectados en la red.
- **nslookup** → Determina la dirección IP de un ordenador conociendo su nombre lógico dentro de un dominio, y viceversa.
- **telnet servidor** → Inicia una sesión especificando el ordenador al cual queremos conectarnos. Una vez dentro nuestro terminal es un ordenador remoto y todo lo que hagamos se enviará hacia el host remoto y se ejecutará en él.
- **ftp servidor** → Transfiere ficheros entre ordenadores sin importar las peculiaridades del emisor y del receptor, ya que ofrece una interfase entre ambos. Tiene las siguientes órdenes:
  - **cd** → Para moverse por los directorios remotos.
  - **ls** → Para listar ficheros en los directorios remotos.
  - **get fich1 fich2** → Obtiene ficheros del ordenador remoto, fich1 es el nombre del fichero original y fich2 es el nombre que queremos darle en nuestro ordenador local.
  - **put fich1 fich2** → Envía ficheros al ordenador remoto, fich1 es el nombre del fichero local y fich2 es el nombre que queremos darle en el ordenador remoto.
  - **mget, mput** → Lo mismo que get y put, pero admiten metacaracteres: *, ?...
  - **prompt** → Realiza transferencias sin solicitar confirmación.
  - **hash** → Hace que por cada KB transferido se visualice el carácter #.

  - **!** → Sale temporalmente al shell.
  - **?** → Lista todas las órdenes existentes.
  - **quit** → Desconexión y salida al shell
  - **close** → Desconexión.
- **finger [-ls] [usuario] [usuario@ordenador]** → Obtiene información de los usuarios conectados a un ordenador en red (**-l** formato largo; **-s** formato corto).
- **talk usuario [@ordenador] [tty]**→ Inicia una conversación con otro usuario a través de la red.
- **ping ordenador** → Permite saber si un ordenador responde dentro de una red enviándole paquetes de información. El ordenador destinatario contestará por cada paquete recibido y se imprimirán estadísticas del tiempo de respuesta.