EL SISTEMA OPERATIVO UNIX

1.- Introducción al sistema operativo UNIX.

1.1.- Breve evolución histórica.

El sistema operativo UNIX nació en 1.969 como respuesta a la frustración de los programadores y a la necesidad de nuevas herramientas informáticas.

En 1.965, Ken Thompson, programador de los Laboratorios Bell [división de investigación de AT&T (American Telephone and Telegraph Company)] estaba trabajando en un programa llamado Space Travel, que simulaba el movimiento de los planetas en el sistema solar. El programa se ejecutaba en un gran ordenador construido por General Electrics, el GE645, que utilizaba un sistema operativo llamado Multics, implementado en lenguaje PL/1. Multics había sido desarrollado en el Instituto Tecnológico de Massachusetts (MIT), y fue uno de los primeros sistemas operativos diseñado para atender distintos usuarios simultáneamente. Sin embargo, la utilización del ordenador GE resultaba cara y lenta (cada ejecución del programa Space Travel costaba alrededor de 70 dólares). Thompson encontró un computador más pequeño y poco utilizado, el PDP-7 construido por Digital Equipment Corporation (DEC), al que transfirió el programa Space Travel para su ejecución. En 1.969 Thompson creó un nuevo sistema operativo para ese computador que llamó UNIX, en el que se incluían los conceptos avanzados de Multics, así como algunos aspectos interesantes de otros sistemas operativos existentes. Como la mayoría de los sistemas operativos, UNIX fue originalmente escrito en lenguaje ensamblador.

El trabajo de Thompson tiene tal éxito que junto con Dennis Ritchie y otros colaboradores consiguen que el UNIX sea operativo en el sistema de los Laboratorios Bell en 1.971. Al principio, UNIX funcionaba únicamente sobre ordenadores fabricados por Digital Equipment, PDP-7, PDP-11/40 y PDP-11/70. Este último muy extendido en centros de investigación y universidades, en donde estratégicamente se autorizó a utilizar UNIX por un precio mínimo, consiguiendo así que cada año se graduaran miles de estudiantes de Informática con alguna experiencia en ejecución y modificación de UNIX.

Con idea de hacerlo fácilmente transportable a otros sistemas informáticos, en 1.973, se reescribe UNIX en un lenguaje de alto nivel, el C, creado por Ritchie y

Kernigham. Actualmente el 95 % del sistema operativo UNIX está escrito en C, quedando sólo una pequeña parte escrita en lenguaje ensamblador, que es la parte del núcleo que interacciona directamente con el hardware.

Entre las universidades a la que llegó UNIX, se encontraba la Universidad de Berkeley en California. Allí se modificó el sistema incorporando una variante notable: la utilización de la memoria virtual paginada. Surge en 1.978 BSD UNIX (BSD Berkeley Software Distribution), que luego tras sucesivas versiones, se ha convertido en un estándar académico.

En 1.983 AT&T comienza el desarrollo de una versión estándar del UNIX para el mercado comercial, el UNIX System V. Posteriormente, surgirán nuevas versiones mejoradas, hasta que en 1.988 se introdujo UNIX System V versión 4.0, en el que se fusionan las características más populares del UNIX de Berkeley y de otros sistemas UNIX, lo que redujo la necesidad de crear variantes UNIX a los fabricantes.

Actualmente cada fabricante ha ido desarrollando su propio estándar de UNIX, con distintos nombres, puesto que UNIX es una marca registrada de AT&T, y siempre dentro de los dos tipos principales de UNIX: tipo BSD y tipo System V (AT&T SVID, el interfaz System V; HP, el UP-UX ; DEC, el ULTRIX; Microsoft, el XENIX; IBM, el AIX; y otros muchos como el SCO o el Linux para los PCs).

1.2.- Sistema operativo Linux.

Linux es una implementación para PCs con interfaz System V de libre distribución. Muchas utilidades de Linux se acogen a la GNU General Public License o GNU Copyleft, lo que permite a los desarrolladores crear programas para el público en general, pudiendo todos los usuarios tener acceso libre a los programas con la posibilidad de modificarlos, pero sin limitar el código modificado, es decir, que el resto de usuarios tiene derecho también a utilizar el nuevo código. Por ello, cualquier versión de Linux incorpora siempre el código fuente completo. El mantenimiento, traducción de programas y nuevos desarrollos suelen realizarse también sin ánimo de lucro. Esta es la razón de que exista no una, sino muchas distribuciones de Linux.

La distribución de Linux corre a cargo de distintas compañías, cada una de las cuales tiene su propio paquete de programas, aunque todas facilitan un núcleo de ficheros que conforman una versión de Linux (ejemplo: kernel 2.0.18). Entre las distribuciones existentes se encuentran, por ejemplo, Slackware, Debian Linux y Red Hat. Esta última es

la instalada actualmente en los ordenadores destinados para las prácticas, necesitando aproximadamente 200 Mbytes para su instalación completa.

La principal ventaja de Linux sobre otros sistemas operativos para PCs, es que de forma gratuita se ofrece un sistema completo con las características multiusuario y multitarea, así como el resto de características del sistema operativo UNIX que se describen en el siguiente apartado.

1.3.- Características de UNIX.

El sistema operativo es un software del sistema que controla y coordina las actividades del ordenador. Supone un entorno de mantenimiento y creación de programas, una interfaz sofisticada de operaciones para el programador (intérprete de comandos), y lleva a cabo la correcta gestión de los recursos del sistema.

El sistema operativo UNIX, así como el Linux, tienen algunas características comunes a la mayoría de los sistemas operativos, aunque también tienen algunas características singulares.

Transportabilidad

La utilización del lenguaje de programación C, ha hecho a UNIX un sistema operativo transportable, funcionando en un amplio abanico de máquinas que van desde portátiles hasta macrocomputadoras, y que pueden comunicarse de forma precisa y efectiva entre sí, sin necesidad de añadir ninguna interfaz de comunicaciones especial.

Multiusuario

Bajo UNIX, varios usuarios (dependiendo de la máquina, hasta más de cien) pueden compartir los recursos del computador simultáneamente, pudiendo ejecutar cada usuario diferentes programas a la vez.

UNIX proporciona medidas de seguridad que permite a los usuarios acceder sólo a los datos y programas para los que tiene permiso.

Multitarea

Cada usuario puede ejecutar varias tareas (programas) a la vez, permitiéndose al usuario conmutar entre las diferentes tareas en ejecución.

Sistema de ficheros jerárquico

UNIX proporciona a los usuarios la capacidad de agrupar datos y programas de forma jerárquica, permitiendo una gestión fácil.

Independencia de los dispositivos de Entrada/Salida

Las operaciones de entrada y salida son independientes del dispositivo porque UNIX trata todos los dispositivos (impresoras, terminales, discos,...) como ficheros. De esta forma, admite cualquier número y cualquier tipo de dispositivo. Con UNIX se pueden redirigir las salidas (o entradas) de las órdenes a (o desde) cualquier dispositivo o fichero.

Comunicaciones y redes

Las utilidades de comunicaciones y redes de UNIX superan a otros sistemas operativos. Las funciones comunicativas inherentes al UNIX se diseñaron para admitir múltiples tareas y múltiples usuarios alejados entre sí, permitiendo el intercambio de información entre usuarios del mismo sistema (comunicaciones internas) o de otros sistemas (comunicaciones externas), e incluso el acceso a otros sistemas remotos.

Interfaz de usuario: la shell programable

La interacción del usuario con UNIX se controla por la shell, que es un poderoso intérprete de comandos programable, que permite la implementación de guiones (Shell Script), los cuales son programas con varios comandos que aumentan la potencia del sistema, permitiendo por ejemplo ejecutar diferentes acciones a la vez, personalizar comandos o enlazar procesos.

Utilidades y servicios del sistema

El sistema UNIX incluye diferentes órdenes diseñadas para realizar una gran cantidad de funciones solicitadas por los usuarios. Entre éstas se encuentran: utilidades de edición y de formateo de texto, utilidades de manejo de ficheros, utilidades de correo electrónico y herramientas del programador. Además, UNIX proporciona una serie de servicios que facilitan la administración y el mantenimiento del sistema.

2.- Sesión de UNIX.

El proceso de establecer contacto con el sistema operativo UNIX consta de una serie de indicadores y entradas de usuarios que comienzan y finalizan una sesión. Una sesión es el período de tiempo que se está utilizando el computador.

2.1.- Proceso de conexión al sistema (login).

Para que un usuario pueda acceder al sistema UNIX, debe tener un identificador, que es proporcionado por el administrador del sistema, y una palabra clave, que sólo conocerá el propio usuario, protegiendo de esta forma el acceso de cualquier otro usuario a su información.

Una vez arrancado el sistema, se visualizan algunos mensajes de identificación, que varían de uno a otro (en nuestro caso, Red Hat Linux release 5.0 (Hurricane)), y aparece el indicador de *login*, introduciéndose el identificador (en nuestro caso, el identificador es el mismo para todos los terminales ("*usuario*")).

puestoN login: usuario <Enter>

Luego el sistema pide la palabra clave, que en nuestro caso es *puestoN*, siendo *N* el número de puesto de cada terminal.

password: puestoN <Enter>

Al objeto de proteger la palabra clave del resto de usuarios, UNIX no devuelve un eco de la misma en pantalla.

Si el sistema no reconoce al usuario o su palabra clave, muestra en pantalla el mensaje de "*login incorrect*", y se repite el proceso. En caso contrario, se termina el proceso de conexión, indicando el sistema que está preparado para recibir comandos visualizando el prompt o indicador del sistema ($ o %, normalmente, según la shell). En nuestro caso, aparece

[usuario@puestoN usuario]$

2.2.- Proceso de desconexión del sistema (logout).

El proceso de salir del sistema cuando se ha terminado se denomina desconexión del sistema. Para ello, es necesario que el prompt se visualice en pantalla, no pudiendo desconectar el sistema en mitad de un proceso. Por tanto, el usuario debe asegurarse que ha finalizado todos sus trabajos, antes de finalizar la sesión.

Para llevar a cabo la desconexión, se utiliza el comando *logout*:

$ *logout* <Enter>

También se puede usar *exit*:

$ *exit* <Enter>

o se teclea <Ctrl-D>.

El sistema UNIX contesta mostrando los mensajes de desconexión que el administrador del sistema haya preparado. Después la pantalla muestra el mensaje estándar de bienvenida al sistema y el indicador de *login*. Esto permite saber que el usuario se ha desconectado y que el terminal está preparado para el siguiente usuario.

Es muy importante tener en cuenta que después de llevar a cabo la desconexión del sistema, **el usuario** habrá finalizado su sesión con UNIX, pero **no debe apagar el terminal**, ya que al mantener información de entrada/salida del sistema de ficheros en la memoria intermedia de la computadora, se puede dañar el sistema de ficheros. Será el administrador del sistema (usuario *root* o superusuario) quien procederá a apagar el sistema, rebotándolo correctamente o usando los comandos *halt*, *reboot* o *shutdown*. Con el comando *shutdown* se apaga el sistema de forma segura, evitando que cualquier usuario entre en el sistema, notificando a todos los usuarios que el sistema va a apagarse, esperando el tiempo especificado y enviando una señal a todos los procesos para que puedan salir sin problemas. Después, *shutdown* ejecuta el apagado o rearranque, según se haya seleccionado en la línea de comandos. Con *halt* o *reboot* se para o rearranca el sistema inmediatamente sin notificarlo a los usuarios.

2.3.- Introducción al intérprete de comandos: línea de comandos

Al iniciar la sesión, entra en funcionamiento el intérprete de comandos, un programa denominado shell (caparazón), mostrándose en pantalla el prompt del sistema.

UNIX ofrece al usuario multitud de comandos para ser ejecutados. Cada línea de comando está formada por tres campos: nombre del comando, opciones y argumentos.

$ nombre_comando [-opciones] [argumentos] <Enter>

Los campos están separados por uno o más espacios, y los corchetes indican que se trata de campos opcionales.

Nombre del comando: Cualquier comando o programa de utilidad válido en UNIX funciona como un nombre de comando.

UNIX es un sistema sensible a mayúsculas y minúsculas (prueba, Prueba, PRUEBA o pRueBa son diferentes), y sólo acepta nombres de comandos en minúsculas.

Opciones: Acompañando al comando pueden aparecer una serie de flags que indican opciones o variantes del mismo. Normalmente van precedidas de un signo menos (-), se suelen designar por una letra minúscula, y pueden aparecer varias opciones en la línea de comando.

Argumentos: Son la información adicional que necesita el comando para poder operar, como puede ser un nombre de fichero.

La línea de comando acaba cuando se teclea <Enter>.

En una misma línea de comando pueden introducirse varios comandos, separados por ";".

Después de recibir un comando, la shell lo procesa y ejecuta, o devuelve un mensaje de error. Cuando termina, el control vuelve a la shell, apareciendo de nuevo el prompt del sistema para indicar al usuario que ya puede introducir el siguiente comando.

Cuando se quiere detener la ejecución de un comando, se teclea <Ctrl-C> o <Del>.

Para corregir errores de mecanografiado se puede utilizar la tecla de retroceso (Back Space) o <Ctrl-H> (carácter de borrado) con lo que el cursor se mueve hacia la

izquierda borrando el carácter correspondiente. Con <Crtl-U> (carácter de eliminación) se borra una línea completa y el cursor se mueve a una línea en blanco.

2.4.- Ayuda: manual en línea.

El comando *man* (manual) muestra las páginas del manual de usuario en versión electrónica (manual on line (en línea)). Para conseguir información acerca de un comando, se escribe:

$ *man* nombre_de_comando <Enter>

En caso de no conocer el nombre del comando que se debe utilizar, se puede usar la opción –k de *man* o el comando *apropos*, que proporcionan una lista de comandos que incluyen en su página del manual una palabra indicada.

$ *man -k* palabra_relacionada <Enter>

$ *apropos* palabra_relacionada <Enter>

Para salir del manual en línea, se escribe q.

Otra forma de obtener información de comandos, en caso de estar disponible, es con *help*:

$ nombre_de_comando *--help* <Enter>

3.- El sistema de ficheros de UNIX.

3.1.- Estructura jerárquica.

La estructura utilizada para almacenar y gestionar los ficheros se denomina sistema de ficheros. Los ficheros de UNIX se encuentran estructurados jerárquicamente como un árbol. Si un fichero contiene un programa o datos, se denominará fichero. Si lo que contiene son las direcciones de otros ficheros, se le llamará directorio. El nivel de

directorio más alto se denomina raíz (*root*) y se representa por el símbolo "/" . Todo el resto de directorios cuelgan directa o indirectamente del raíz.

En UNIX el concepto de fichero es universal. Los dispositivos del sistema como los terminales, las impresoras o los discos, se consideran ficheros.

Los directorios empleados por el sistema se denominan directorios del sistema y contienen ejecutables, programas fuentes, documentación, bases de datos para la gestión del sistema, etc.

La organización de la estructura de ficheros de UNIX varía según la instalación. Una estructura estándar incluye los siguientes directorios y ficheros:

/ Directorio raíz (*root*) del sistema de ficheros.

/bin Contiene los comandos básicos del sistema, es decir, los programas ejecutables (binarios).

/dev Contiene los ficheros de dispositivos (devices).

/etc Contiene varios ficheros administradores (ficheros diversos) como el fichero de contraseñas, o el de inicialización de conexión de terminales.

/lib Contiene ficheros de librerías de UNIX, como las subrutinas de C.

/tmp Almacena ficheros temporales creados durante la ejecución de algún programa y que desaparecen cuando se rebota el sistema.

/unix Es el programa del kernel o núcleo de UNIX. Cuando el sistema comienza su ejecución, se carga de disco a memoria y empieza a ejecutarse. Primero se carga el fichero */boot* y luego éste lee */unix*.

/usr Es el sistema de ficheros del usuario. De este directorio suelen colgar los ficheros de usuarios. Al igual que en el directorio raíz /, existen directorios */usr/bin*, *usr/lib* y *usr/tmp* con funciones similares.

/usr/adm Zona de administración del sistema.

/usr/include Contiene los ficheros de cabecera .h para programación en C.

/usr/man Es el manual on line.

/usr/pub Zona para ficheros públicos.

/usr/src Zona para código fuente de utilidades y biblioteca.

/usr/src/cmd Zona de fuentes de comandos de */bin* y */usr/bin*.

/usr/spool Directorio para programas de comunicaciones.

3.2.- Nombres de ficheros y directorios.

UNIX ofrece libertad a la hora de nombrar sus ficheros y directorios. La longitud máxima de un nombre de fichero depende de la versión de UNIX y del fabricante del sistema, permitiendo nombres de al menos 14 caracteres, y la mayoría por encima de 255 caracteres. Se puede utilizar una combinación de números y caracteres, pero sin incluir caracteres que tienen un significado especial para la shell (¡, @, #, $, %, &, *, (,), [,], {, }, ‘, ‘‘, \, /, |, ;, < y >). La única excepción es el directorio raíz, que se nombra siempre con una barra inclinada a la derecha (/). Ningún otro fichero puede usar este nombre. Hay que tener en cuenta que UNIX es sensible a mayúsculas: las letras mayúsculas son distintas de las minúsculas. Se debe evitar utilizar espacios en el nombre de un fichero, ya que UNIX utiliza espacios para indicar dónde finaliza un comando o nombre de fichero y dónde comienza otro.

Se pueden utilizar extensiones de nombres de ficheros para clasificarlos y describirlos. Las extensiones son la parte del nombre del fichero que sigue a un punto y en la mayoría de los casos son opcionales. Algunos compiladores de lenguajes de programación requieren de nombres con una extensión específica (por ejemplo la extensión .c para el lenguaje C).

3.2.1.- Uso de comodines.

Se pueden utilizar máscaras o caracteres comodín para generalizar nombres de ficheros:

*	Se sustituye por cualquier cadena de caracteres, incluida la cadena vacía.
?	Se sustituye por un carácter cualquiera.
[...]	Se sustituye por cualquier carácter de los encerrados entre corchetes.

Por ejemplo, la expresión *[a-z]** se sustituye por cualquier cadena que comience por una letra minúscula.

3.3.- Directorio de conexión.

Cada usuario del sistema tiene asociado un directorio, al que accede automáticamente cuando se conecta al sistema. Este directorio se denomina directorio de conexión (*home*). Suele colgar de otro directorio, que a su vez contendrá todos los directorios *home* de usuarios que pertenezcan a un mismo grupo. Los ficheros creados por el usuario, se almacenarán por defecto en este directorio, y heredarán el directorio *home* de su predecesor. Es conveniente que el usuario también organice jerárquicamente sus ficheros, creando subdirectorios.

3.4.- Directorio de trabajo.

Mientras un usuario esté trabajando en el sistema UNIX, estará siempre asociado a un directorio. El directorio con el que está asociado, o en el que trabaja, se denomina directorio de trabajo o directorio actual. Algunos comandos permiten ver o cambiar el directorio de trabajo.

3.5.- Caminos absoluto y relativo.

Todos los ficheros tienen una ruta de acceso (*path*) que los localizan en el sistema de ficheros. Los ficheros en su directorio de trabajo son accesibles inmediatamente. Para acceder a los ficheros que se encuentran en otros directorios es necesario especificar el nombre del fichero particular con su ruta de acceso.

La ruta de acceso puede determinarse mediante un camino absoluto o un camino relativo.

Un **camino absoluto** (camino completo) se traza desde el directorio raíz al fichero, atravesando todos los directorios intermedios. Especifica exactamente dónde buscar un fichero, por lo que se puede utilizar para localizar un fichero en el directorio de trabajo o en cualquier otro directorio. Siempre comienza con el nombre del directorio raíz, con la barra inclinada a la derecha (/). Los nombres de otros directorios y ficheros también se separan con barras inclinadas a la derecha (/). (Ejemplo: */usr/juan/primero*).

Un **camino relativo** se traza desde el directorio de trabajo a un fichero. Al igual que el camino absoluto, el camino relativo puede describir un camino a través de varios directorios. Nunca hay una barra inclinada a la derecha (/) al principio de un camino relativo, ya que comienza desde el directorio actual. (Ejemplo: *juan/primero* desde el directorio de trabajo *usr*).

3.6.- Tipos de ficheros y permisos.

El sistema de ficheros de UNIX consiste en un conjunto de ficheros, que pueden ser de tres tipos:

- **Ficheros ordinarios**. Ficheros que sólo contienen datos.
- **Ficheros especiales**. Ficheros que representan dispositivos físicos como terminales e impresoras, permitiendo el acceso a los mismos, y que también tienen otros propósitos.
- **Directorios**. Contienen nombres de ficheros y subdirectorios, así como punteros hacia esos ficheros y subdirectorios.

Para determinar el tipo de fichero, se puede utilizar el comando *file*.

UNIX admite usuarios y grupos de usuarios. Ambos están identificados por un número entero no negativo. Un usuario puede pertenecer a varios grupos. Mediante el comando *groups* se obtiene la lista de grupos a la que pertenece un usuario. Todo fichero UNIX tiene un propietario y un grupo propietario.

Los ficheros en UNIX tienen una serie de **protecciones** que indican qué operaciones pueden realizarse con el mismo, y qué usuarios son los que pueden realizarlas. Las protecciones de un fichero sólo pueden ser modificadas por el propietario del mismo (alguien que posea los **permisos** adecuados) y pueden ser de tres tipos:

1) Lectura: Se identifican con la letra "**r**" (*read*).
2) Escritura: Se identifican con la letra "**w**" (*write*).
3) Ejecución: Se identifican con la letra "**x**" (*execute*).

Las protecciones o permisos se identifican mediante una máscara que consta de 10 caracteres. El primer carácter identifica el tipo de archivo:

- **"-"*:*** Se trata de un archivo ordinario.
- **"d"**: Se trata de un directorio.
- **"b"**: Se trata de un archivo especial asociado a un dispositivo que trabaja con bloques (de datos). Por ejemplo los disquetes, el disco duro, etc.
- **"c"**: Se trata de un archivo especial asociado a un dispositivo que trabaja con caracteres. Por ejemplo la impresora.

Los tres siguientes caracteres dentro de esa máscara identifican las protecciones del archivo en relación a su propietario. Estos tres caracteres poseen el siguiente formato: **rwx**. De manera que si el propietario posee derecho de lectura sobre el archivo aparecerá la "**r**", y si no un guión "**-**" (indica ausencia de dicho permiso). De igual forma se tratarían los permisos de escritura ("**w**") y ejecución ("**x**").

Los tres caracteres que siguen se utilizan para indicar las protecciones del archivo respecto al grupo de usuarios al que pertenece el propietario (los usuarios suelen organizarse en grupos). Por lo demás, su formato es idéntico al caso anterior: **rwx**.

Los tres últimos caracteres sirven para establecer las protecciones del archivo respecto a cualquier usuario (que puede no ser el propietario, ni pertenecer al grupo de éste). Estos tres caracteres siguen utilizando el mismo formato anterior: **rwx**.

Con el comando *ls -l* , que se verá posteriormente, se muestra en la primera columna los permisos de cada uno de los ficheros listados.

Ejemplo: -rwxr-xr-- 2 pilar 16 Jun 25 12:28 prueba

El primer campo indica que se trata de un fichero ordinario, que el propietario tiene todos los permisos, que el grupo tiene permisos de lectura y ejecución, y que el resto de usuarios sólo tiene permiso de lectura sobre el fichero prueba.

3.6.1.- Modificación de permisos

El **comando *chmod*** modifica los permisos de un fichero. Hay que especificar, como primer parámetro del comando, los permisos para los tres grupos de usuarios como un número de tres dígitos. Cada dígito representa los tres tipos de acceso para cada tipo de usuario. Por ejemplo, los permisos del fichero prueba se representarían con los dígitos 754, que en binario se escribiría como 111 101 100. Como segundo parámetro tenemos que especificar el nombre del fichero cuyos permisos queremos modificar.

Para poner sólo permiso de lectura para el fichero prueba a todos los tipos de usuarios, tendríamos que ejecutar el siguiente comando:

```
$ chmod 444 prueba
```

Una forma alternativa de cambiar los permisos es especificar los tipos de acceso y tipos de usuarios, mediante una letra identificativa de cada uno de ellos:

Tipos de usuarios:

- **u** Usuario propietario
- **g** Grupo propietario
- **o** Resto de usuarios
- **a** Todos los usuarios

Tipos de permisos:

- **r** Permiso de lectura
- **w** Permiso de escritura
- **x** Permiso de ejecución
- **-** Ningún permiso

Concesión de permisos:

+	Autorización de permiso
-	Denegación de permiso
=	Autorización de todos los permisos

El ejemplo anterior, podría haberse hecho de diversas formas:

```
$ chmod ugo+r prueba
$ chmod a+r prueba
$ chmod u+r, g+r, o+r prueba
```

3.7.- Principales ficheros de dispositivos.

El sistema de ficheros de UNIX se caracteriza en que a los ficheros se accede por su nombre, y ese acceso es independiente del dispositivo físico en el que resida el fichero (disco flexible o disco duro).

En el directorio */dev* se encuentran ficheros asociados con los dispositivos del sistema. Algunos de ellos son los siguientes:

/dev/lp → asociado a la impresora.

/dev/ram → asociado con la memoria.

/dev/fd0 → asociado a la unidad de disquete A:

/dev/fd1 → asociado a la unidad de disquete B:

/dev/hd → asociado al disco duro.

/dev/hd1, */dev/hd2* ... */dev/hdn* → particiones del disco duro.

/dev/tty1, */dev/tty2* ... */dev/ttyn* → asociados a terminales.

/dev/console → asociado a la consola.

Un dispositivo especial es */dev/null*. Se trata de un pozo sin fondo. Todos los datos que se escriben allí, se pierden para siempre. Esto puede ser muy útil si se quiere ejecutar una orden y no se necesita ni la salida estándar ni los errores estándares. Además, si se utiliza */dev/null* como un archivo de entrada, se creará un fichero con longitud cero.

Todos los dispositivos UNIX, incluidos los sistemas de directorios de unidades removibles, cuelgan del árbol de directorios principal. Cada vez que se cambia un disco de unidad hay que indicárselo al sistema. Esto se hace *montando* y *desmontando* volúmenes: unir/desunir la estructura de directorio de un disco al sistema de directorios principal. Por

ejemplo, si se desea montar un disquete de la unidad *fd0* (unidad A:) en el directorio */usr*, se haría así:

/etc/mount /dev/fd0 /usr (⇔ pegar el árbol de *fd0* en */usr*)

Para desmontarlo se haría:

/etc/umount /dev/fd0

En clase **no se utilizarán** estos comandos para evitar el montaje o desmontaje indeseado de volúmenes, dando lugar a fallos en el sistema de ficheros.

3.8.- Ficheros de inicialización o de perfil.

Al igual que en DOS, el archivo *autoexec.bat* contiene comandos que se ejecutan al iniciarse el sistema, UNIX tiene varios de estos archivos (por lo general, los archivos que empiezan por un punto suelen destinarse a tareas de inicialización).

Cada vez que un usuario se conecta al sistema, se ejecutan una serie de pasos hasta que por último se invoca a la shell. Lo primero que hace la shell es ejecutar los ficheros de perfil. El primero es el perfil del sistema */etc/profile* (*Bourne-shell*) o */etc/cshrc* (*C-shell*), que se ejecuta para todos los usuarios, y el segundo (fichero *.profile* (*Bourne-shell*) o *.login* (*C-shell*)) en el directorio propio del usuario, que sólo se ejecuta para el usuario propietario. Estos ficheros contiene comandos y variables que afectan al entorno del usuario conectado, como pueden ser PATH, SHELL o HOME. Se trata de ficheros fácilmente modificables con cualquier editor, al ser ficheros en código ASCII.

Más adelante se verán las principales variables de entorno utilizadas en la *shell*.

4.- Comandos relativos al sistema de ficheros.

Los comandos relativos al sistema de ficheros de UNIX se agrupan en comandos para moverse por dicho sistema de ficheros y comandos de directorios y ficheros.

En dichos comandos se utiliza "/" como **directorio raíz**, "**.**" para representar al **directorio actual** y "**..**" para representar al **directorio padre del directorio actual**.

En la mayoría de los comandos que aparecen a partir de este apartado, sólo se incluye una breve descripción de los mismos, así como algún ejemplo. Para obtener más información en cuanto a todas las opciones posibles, se pueden utilizar el manual on line *man*, la guía de comandos UNIX, o cualquier libro de UNIX.

4.1.- Comandos para moverse por el sistema de ficheros.

Comando *pwd*: Visualiza la ruta de acceso del directorio de trabajo actual.

Ejemplo:
```
$ pwd
/usr/MiDirectorio
```

Comando *cd*: Cambia el directorio de trabajo por el directorio especificado.

Ejemplo:
```
$ pwd
/usr/mio
$ cd fuente
$ pwd
/usr/mio/fuente
$ cd ..
$ pwd
/usr/mio
```

Comando *ls*: Se utiliza para visualizar el contenido de un directorio especificado. Si no se especifica ningún parámetro, lista la información en orden alfabético. Si no se especifica un directorio concreto, el listado se hace sobre el directorio actual.

Algunas opciones de *ls*:

-l: Formato largo. Muestra, además del nombre del fichero o directorio, el tipo de archivo, número de enlaces, tamaño y tiempo de la última modificación.

-t: Ordena el listado por el tiempo de modificación, en lugar de hacerlo por orden alfabético.

-r: Muestra el listado en orden alfabético inverso.

-a: Muestra también los ficheros ocultos.

4.2.- Comandos de directorios y ficheros.

Comando *mkdir*: Crea un nuevo subdirectorio a partir del directorio actual o del directorio que especifiquemos como parámetro en el comando.

Ejemplo:
```
$ pwd
/usr/pepe
$ mkdir memos
$ mkdir memos/importante
$ cd memos/importante
$ pwd
/usr/pepe/memos/importante
```

La opción ***-p*** permite crear una estructura de directorios completa en una sola línea.

Ejemplo:
```
$ pwd
/usr/pepe
$ mkdir -p memos/importante
$ cd memos/importante
$ pwd
/usr/pepe/memos/importante
```

Comando *rmdir*: Elimina el directorio especificado. Sólo pueden ser eliminados aquellos directorios que estén vacíos (no pueden contener ficheros ni directorios).

Ejemplo:
```
$ cd memos
$ pwd
/usr/pepe/memos
$ rmdir importante
```

Comando *cp*: Copia un fichero en otro. Pueden copiarse ficheros de distintos directorios, especificando la ruta de cada uno de ellos.

Ejemplos:
```
$ cp fichero1 fichero2
$ cp /usr/fichero1 fichero2
```

Comando *mv*: Mueve un archivo de un sitio a otro y permite renombrar ficheros y directorios.

Ejemplos: $ *mv* copia copia2
$ *mv* copia /usr/juan

Comando *rm*: Borra un fichero. Con la opción -r borra un directorio y, de forma recursiva, todos sus subdirectorios.

Ejemplos: $ *rm* copia2
$ *rm –r* directorio

4.3.- Manejo de archivos DOS bajo UNIX.

Si durante el proceso de instalación del sistema operativo, se incluye la instalación de la componente de instalación apropiada, se dispondrán de los conocidos como **comandos *m-***, que son equivalentes a comandos DOS y se escriben con la *m* delante.

Entre estos comandos se encuentran los siguientes:

Comando *mcopy*: Copia los archivos especificados a la nueva ruta de acceso.

Comando *mdir*: Ofrece un listado de directorios.

Comando *mdel*: Elimina los archivos especificados.

Comando *mmd*: Crea un directorio.

Comando *mrd*: Suprime un directorio, que debe estar vacío.

Comando *mren*: Asigna un nuevo nombre a un archivo DOS, ya existente.

Comando *mcd*: Cambia de directorio a la ruta de acceso especificada.

Comando *mlabel*: Etiqueta el sistema de archivos DOS.

Comando *mtype*: Muestra el contenido de texto de un archivo DOS.

Comando *mformat*: Formatea un disquete.

Estos comandos facilitan el **copiado de ficheros en disquetes**, ya que se puede utilizar la designación del DOS (unidad A:) en lugar de la designación */dev/fd0*, que implica utilizar los comandos *mount* y *umount*.

Se puede obtener más información sobre los comandos *m-* con *man*, escribiendo:

$ *man mtools*

5.- Editores de texto.

Un editor de texto es una herramienta que facilita la creación de nuevos archivos y la modificación de los ya existentes.

Con cada sistema operativo se suministra, al menos, un programa editor, que podrá ser un **editor de línea** o un **editor de pantalla completa**.

En un **editor de línea**, la mayoría de los cambios se aplican a una línea o a un grupo de líneas a la vez. Para realizar un cambio, se debe especificar primero el número de línea de texto y después el cambio. Los editores de línea normalmente son difíciles de utilizar, ya que no se puede ver el alcance y el contexto de la tarea de edición. Son buenos para operaciones globales como búsquedas, reemplazamientos y copias de grandes bloques de texto en un archivo.

Un **editor de pantalla** visualiza una pantalla completa del texto que se edita y permite mover el cursor a lo largo de la pantalla para realizar cambios. Cualquier cambio que se realice se aplica al archivo y se obtiene una respuesta inmediata en la pantalla. Se puede ver fácilmente el resto del archivo moviendo toda la pantalla a la vez. Los editores de pantalla son más cómodos que los editores de línea.

El sistema operativo UNIX soporta tanto editores orientados a línea (*ed* y *ex*, por ejemplo) como editores orientados a pantalla (*vi*).

5.1.- Editor *vi*.

El editor de textos *vi* (intérprete visual) es uno de los más utilizados por los usuarios de UNIX, ya que se encuentra disponible en cualquier versión de este sistema operativo. Su mayor punto a favor es su capacidad para realizar operaciones sobre el texto (inserciones, copias, borrados, modificaciones, etc.) de una manera rápida y eficiente. Su mayor desventaja es que tales operaciones se deben realizar mediante comandos un tanto difíciles de memorizar.

El gran beneficio de dominar *vi* es tener la seguridad de que se podrá trabajar en multitud de sistemas. Es necesario recordar que un editor de textos no es lo mismo que un procesador de textos puesto que éste permite cambiar fuentes, tamaños de letra, hacer subrayados, etc..., mientras que el editor no. Para dar formato a texto en *vi* se insertan una serie de códigos que después son interpretados por otro programa, el *formateador de texto*. En particular, *vi* posee una serie de comandos que ningún procesador de textos iguala en eficiencia y complejidad.

Se debe tener en consideración que *vi* es en realidad el modo visual de otro editor llamado *ex*.

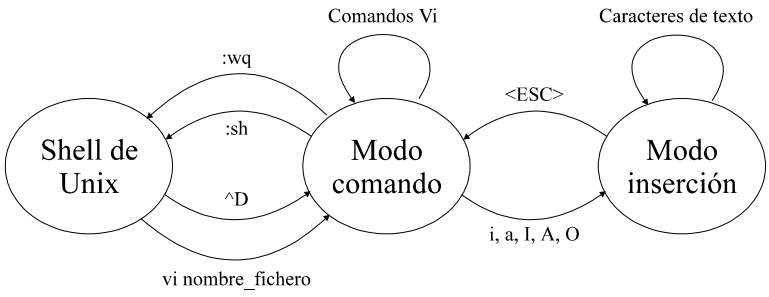
5.1.1- Comienzo y fin de una sesión de edición.

En este apartado se explica cómo arrancar el editor *vi*, introducir texto, introducir comandos, salvar el texto en un fichero y abandonar el editor.

El editor *vi* funciona con dos **modos** distintos **de trabajo**.

- Por una parte está el ***modo de inserción***, donde cada tecla tiene un valor real (cada tecla que pulsamos aparece en la pantalla).
- Por otra parte está el ***modo de comando***, donde cada tecla tiene un significado especial (cada tecla pasa a ser un comando).

Hay distintas teclas que nos llevan de un modo a otro. Por ejemplo, si estamos en *modo comando*, la tecla <i> nos lleva a *modo de inserción*, y estando en *modo inserción* la tecla <ESC> nos lleva a *modo comando*. Desde el *modo comando* podemos retornar a la shell de UNIX, terminando así la sesión con *vi*. Este proceso aparece descrito en la siguiente figura.



Modos de trabajo de *vi*

Una **sesión de trabajo típica** conlleva los siguientes pasos:

1) Arrancar el editor.

- Desde la shell del sistema operativo introducimos el comando *vi* seguido por el nombre que queramos dar al fichero:

$ *vi* primero.doc

- Se borrará la pantalla y aparecerá lo siguiente:

```
-
~
~
~
~
~
~
~
~
'primero.doc' [New File]
```

2) Seleccionar el *modo de inserción* de texto.

- Pulsando la tecla <i> seleccionamos el *modo de inserción* de texto. No aparece nada en pantalla, pero ya podemos comenzar a introducir texto.

3) Introducir un texto.

Por ejemplo, escribiremos el párrafo siguiente, asegurándonos de pulsar <Enter> al final de cada línea:

```
El editor de textos UNIX se llama vi
(intérprete visual).
El editor vi es actualmente el
intérprete visual del editor ex.
Cuando utilice vi, descubrirá que estos
dos editores están interrelacionados
durante una sesión de edición.
```

4) Volver al *modo de comandos.*

- Para abandonar el modo de inserción de texto pulsamos la tecla <ESC> (escape).
- De nuevo, la pantalla no cambia ni presenta ningún indicador, pero volveremos al modo de comandos de *vi*, desde el que podremos introducir comandos de edición.

5) Guardar el texto en un fichero y salir de *vi*.

- Para escribir el texto en un fichero llamado *primero.doc* tecleamos **:w** y pulsamos <Enter>.
- Después de que aparezca :w en la parte inferior de la pantalla, *vi* presenta el siguiente mensaje:

```
"primero.doc" [New file] 8 lines, 221 characters
```

- Para abandonar *vi* y volver al indicador del sistema, tecleamos **:q** (o **:q!)** y pulsamos <Enter>. Aparecerá el indicador del sistema en pantalla.

Los pasos anteriormente descritos ilustran una *sesión de edición completa de principio a fin*. Se ha empleado el editor *vi* para crear un pequeño fichero de texto.

Los comandos **:w** y **:q** son realmente comandos del editor *ex*. Los dos puntos inician el modo de comandos de *ex* y **w** y **q** indican operaciones de escritura y salida. Podemos utilizar estos comandos de forma conjunta **:wq**. Para escribir el texto en un fichero sólo si se ha modificado éste, se utiliza **:x** o bien **ZZ**.

5.1.2.- Movimientos del cursor y pantalla.

Una de las operaciones más básicas en el empleo de un editor es desplazar el cursor de un lugar a otro. Muchas de las funciones de edición de *vi* trabajan de forma dependiente de la posición y movimientos del cursor.

5.1.2.1.- Movimientos de una posición cada vez.

Se realizan con las teclas de desplazamiento del cursor. Los movimientos del cursor se consideran comandos de *vi*, por lo que debemos estar en el *modo de comandos* para usarlos (si no estamos seguros del modo en que estamos, pulsaremos la tecla <ESC>; si se oye un pitido, entonces ya estábamos en modo comando).

5.1.2.2.- Repetición y cancelación de pulsaciones.

Antes de continuar el estudio de los comandos de desplazamiento del cursor, se muestra un sencillo ejemplo para repetir y cancelar pulsaciones, elementos muy útiles para la edición de texto.

1. Introducir dos líneas de texto.

- Abrimos el fichero primero.doc mediante *vi* primero.doc, situamos el cursor al final del fichero, pulsamos la tecla <i> para pasar a modo inserción y pulsamos la tecla <Enter> para comenzar en un nuevo párrafo.
- Introducimos dos nuevas líneas de texto:

```
Estas dos líneas muestran qué puede hacer
con las teclas de repetición y cancelación.
```

- Finalmente pulsamos la tecla <ESC>, obteniendo una pantalla como la que sigue:

```
El editor de textos UNIX se llama vi
(intérprete visual).
El editor vi es actualmente el
intérprete visual del editor ex.
Cuando utilice vi, descubrirá que estos
dos editores están interrelacionados
durante una sesión de edición.

Estas dos líneas muestran qué puede hacer
con las teclas de repetición y cancelación.
~
```

2. Repetir las pulsaciones.

- Para repetir estas dos líneas pulsamos el punto <**.**> pero no <Enter>.
- La pantalla que obtendremos ahora será:

```
El editor de textos UNIX se llama vi
(intérprete visual).
El editor vi es actualmente el
intérprete visual del editor ex.
Cuando utilice vi, descubrirá que estos
dos editores están interrelacionados
durante una sesión de edición.

Estas dos líneas muestran qué puede hacer
con las teclas de repetición y cancelación.

Estas dos líneas muestran qué puede hacer
con las teclas de repetición y cancelación.
~
```

3. Cancelar las pulsaciones.

- Para cancelar el comando pulsamos la tecla <u>, pero no <Enter>.

- El párrafo copiado desaparecerá y la pantalla volverá a ser la original.
- Para restaurar la copia podemos pulsar la tecla <u> de nuevo, y si la pulsamos una vez más, volveremos a borrar la copia.

4. Abandonar sin salvar las dos líneas extra.

- Para salir de *vi* sin salvar las líneas extra usaremos **:q!** y pulsaremos <Enter>.
- Si sólo se pone **:q** deberemos contestar afirmativamente al mensaje de salida.

5.1.2.3.- Ir a los extremos de una línea.

- Para ir al principio de la línea actual pulsamos <^>.
- Para ir al final de la línea actual pulsamos <**$**>.

5.1.2.4.- Movimientos de una palabra.

- Para avanzar una palabra en la línea actual pulsamos <w> ó <W>.
- Para retroceder una palabra en la línea actual pulsamos <b> ó <B>.
- Se puede usar un multiplicador, como el 4 en <4><w> para desplazarse un determinado número de palabras de una sola vez.

5.1.2.5.- Movimientos de una línea.

- El paréntesis izquierdo <(> retrocede al principio de la línea actual. Si el cursor se encuentra al principio de una línea, se desplaza el cursor al comienzo de la línea anterior.
- El paréntesis derecho <)> avanza al comienzo de la línea siguiente (pueden usarse multiplicadores).

5.1.2.6.- Movimientos por párrafos.

- La llave izquierda <{> retrocede al principio del párrafo actual. El principio del párrafo es normalmente la línea en blanco que le precede.
- La llave derecha <}> avanza al comienzo del párrafo siguiente (pueden usarse multiplicadores).

5.1.2.7.- Desplazamientos alrededor de la pantalla.

- Con <H> nos desplazamos a la parte superior de la pantalla.
- Con <M> nos desplazamos al centro de la pantalla.
- Con <L> nos desplazamos a la parte inferior de la pantalla.

5.1.2.8.- Ir a una línea en concreto.

Con <G> podemos mover el cursor a cualquier línea del texto, aunque dicha línea no se encuentre en pantalla en la actualidad. Basta con teclear un número de línea delante del comando, por ejemplo, con <5><0><G> iremos a la línea 50. Con <G> exclusivamente vamos a la última línea del texto.

5.1.3.- Ajuste de la presentación en pantalla.

Esta sección describe las distintas formas en que podemos ajustar la presentación en pantalla.

5.1.3.1.- Paginación.

Paginar es presentar una pantalla completamente llena de texto:

- Con <Ctrl-B> retrocedemos una página.
- Con <Ctrl-F> avanzamos una página.

5.1.3.2.- Desplazamiento (scroll).

Desplazar es mover la pantalla la mitad cada vez:

- Con <Ctrl-U> desplazamos hacia arriba.
- Con <Ctrl-D> desplazamos hacia abajo.

5.1.3.3.- Control de la presentación en pantalla.

- Con <z><Enter> situamos la línea actual en la primera línea de la pantalla.
- Con <z><.> situamos la línea actual en el centro de la pantalla.
- Con <z><-> situamos la línea actual en la última línea de la pantalla.

5.1.3.4.- Borrar mensajes del sistema.

A veces el sistema escribe mensajes en la pantalla mientras se está trabajando. Otras veces el mismo *vi* puede escribir caracteres extraños. En todo caso, el comando

<Ctrl-L> se emplea para borrar mensajes de pantalla y volver a presentar aquellas líneas sobre las que se escribió encima.

5.1.4.- Introducción de texto nuevo.

Después de haber creado un fichero es frecuente tener que añadirle más texto. El editor *vi* dispone de varios comandos mediante los cuales se puede insertar el nuevo texto,
dependiendo de dónde tenga que colocarse dicho texto en relación con el ya existente:

- Si queremos que el nuevo texto vaya antes del texto existente se inserta el nuevo texto.
- Si queremos que el nuevo texto vaya después del texto existente se añade el nuevo texto.
- Si queremos que el nuevo texto vaya en una línea nueva por encima o por debajo del texto ya existente, podemos abrir una línea nueva.

5.1.4.1.- Insertar texto.

- Para insertar texto nuevo por delante del cursor se utiliza el comando <i> en el *modo de comandos*.
- Para insertar texto al principio de la línea actual se puede mover el cursor al principio de la línea y utilizar <i> o bien dejar el cursor donde está y utilizar <I>.

5.1.4.2.- Añadir texto al final.

- Pulsando <a> añadiremos texto después de la posición actual del cursor, desplazándose a la derecha el texto existente delante del texto nuevo.
- Para insertar texto al final de una línea sin desplazar el cursor hasta el final de la misma podemos usar <A>.

5.1.4.3.- Insertar una línea.

- Tecleando <O> se inserta una línea nueva por encima de la actual y se sitúa el cursor al comienzo de la misma.
- Tecleando <o> se inserta una línea nueva por debajo de la actual y se sitúa el cursor al comienzo de la misma.

5.1.5.- Supresión de texto.

Los comandos para suprimir texto tienen muchas variantes. Se pueden suprimir caracteres, palabras, líneas, oraciones y párrafos.

5.1.5.1.- Suprimir caracteres.

- Para borrar un sólo carácter basta con desplazar el cursor hasta el carácter y pulsar <x>.
- Para suprimir varios caracteres consecutivos movemos el cursor hasta el primer carácter y tecleamos un número delante de x, por ejemplo <4><x>.

5.1.5.2.- Suprimir palabras.

- Para suprimir una palabra desplazamos el cursor hasta la primera letra de la palabra y tecleamos **dw** desde el *modo de comando* de *vi*.
- Si la palabra contiene signos de puntuación usaremos **dW**.
- Para borrar varias palabras utilizaremos **ndw**, siendo *n* un número.

5.1.5.3.- Suprimir líneas.

- Para suprimir una línea desplazamos el cursor a cualquier posición en la misma y pulsamos **dd**.
- Para borrar más de una línea, precedemos el comando de un número (**ndd**). Podemos usar el comando <u>, visto anteriormente, para recuperar líneas borradas por error.
- También podemos suprimir texto desde la posición actual del cursor hasta el principio (**d^**) o el final (**d$**) de la línea.

5.1.5.4.- Suprimir oraciones.

- Para suprimir una oración, desplazaremos el cursor al principio de la oración y pulsaremos <**d**><**)**>. Podemos usar números para borrar más de una oración, por ejemplo **2d(**.
- También se puede suprimir parte de una oración desplazando el cursor al lugar apropiado y tecleando **d(** para borrar hasta el principio de la oración o **d)** para borrar hasta el final de la oración.

5.1.5.5.- Suprimir párrafos.

- Para suprimir un párrafo, desplazaremos el cursor al principio del mismo y pulsaremos <**d**><**}**>. Podemos usar números para borrar más de un párrafo, por ejemplo **2d}**.
- También se puede suprimir parte de un párrafo desplazando el cursor al lugar apropiado y tecleando **d{** para borrar hasta el principio del mismo o **d}** para borrar hasta el final del párrafo.

5.1.6.- Movimiento de texto.

Mover texto es similar a borrarlo pero con la diferencia de que el texto borrado no se borra en realidad, simplemente se almacena temporalmente en otro lugar. Para mover texto de un punto A a otro B se siguen cuatro pasos genéricos:

1) Desplazar el cursor al punto A.

2) Borrar el texto con uno de los comandos de supresión descritos en la sección anterior.

3) Desplazar el cursor al punto B.

4) Insertar el texto con un comando "poner" que se explica en este apartado.

Advertencia: Si borramos texto con la intención de presentarlo en otro lugar con el comando "poner", puede suceder que antes de haber ejecutado la inserción se ejecute algún comando que nos borre el *buffer* donde se almacena el texto borrado. Estos comandos pueden ser: pasar a modo inserción, ejecutar cualquier otro comando de borrado, etc.

5.1.6.1.- Mover oraciones.

- Suprimimos una oración cualquiera con **d)** y desplazamos el cursor a la posición donde deseamos mover la línea.
- Ejecutamos <**P**> para insertar la oración eliminada por delante del cursor (o por encima de la línea actual).
- Ejecutamos <**p**> para insertar la oración eliminada por detrás del cursor (o por debajo de la línea actual).

El movimiento satisfactorio de un texto depende claramente de colocar el cursor en la posición adecuada y de utilizar el comando correcto.

5.1.6.2.- Mover bloques.

- Podemos transponer dos caracteres adyacentes situando el cursor sobre el primero y pulsando <**x**><**p**>. Por ejemplo, supongamos que hemos escrito "*uqe*" en lugar de "*que*". Si desplazamos el cursor a la *u* y pulsamos <**x**><**p**>, la **x** borra la *u* y la **p** pone el carácter suprimido después del cursor (el cual está situado ahora debajo de la *q*).
- También se pueden mover palabras, líneas y párrafos de un sitio a otro borrándolos de su lugar original, con comandos ya vistos como **dw**, **dd** y **d}** e insertándolos en una nueva posición destino con **p** ó **P**.

5.1.7.- Búsqueda y sustitución de texto.

5.1.7.1.- Buscar texto.

- Para buscar un texto desde la posición actual hasta el final del fichero, basta con teclear una barra (/) seguida por el texto y pulsar <Enter>. El cursor se desplazará hasta la primera aparición del texto.
- Si queremos repetir la misma búsqueda, usaremos <**n**>.
- Para invertir la dirección de la búsqueda (desde el cursor hasta el comienzo del fichero), comenzaremos la búsqueda con una interrogación (**?**) en vez de con una barra (/).

5.1.7.2.- Sustituir texto.

Una de las características más útiles de un programa editor de textos como *vi* es la posibilidad de buscar cada aparición de una palabra o expresión y reemplazarla por otra. Esta característica permite corregir muchos errores ortográficos o cambiar la terminología del texto. La sustitución de texto se realiza mediante el comando **s**, que se utiliza mediante la siguiente sintaxis:

: inicio, final s /busca /cambia /g

- Comenzamos con dos puntos (**:**), indicando el cambio al *modo de comandos*.
- A continuación se introducen los números de línea inicial y final entre los que se desea buscar.
- **s** es el comando de sustitución.
- Es necesario indicar a continuación el texto que se está buscando y el texto por el que se quiere sustituir.
- Finalmente con **/g** indicamos que deben reemplazarse todas las apariciones del texto buscado, y no sólo la primera.
- Como último paso es necesario pulsar la tecla <Enter> para que se realicen los cambios.

Veamos un ejemplo:

:1,$s/vi/video-editor/g Sustituye *vi* por *video-editor* en todo el fichero.
:1,$s/video-editor/vi/g Sustituye *video-editor* por *vi* de nuevo.

Podemos restringir la búsqueda seleccionando números de línea específicos para el comienzo y el final. Un número de línea puede ser un número, un texto de búsqueda o un símbolo tal como **.** (línea actual) y **$** (última línea). Seguidamente se muestran algunos ejemplos:

5,. Desde la línea 5 hasta la línea actual.
.-3,$ Desde 3 líneas antes de la actual hasta el final del fichero.

5.1.8.- Funciones adicionales.

- Con **:! nombre_comando** se puede ejecutar un comando *UNIX* dentro de *vi*.
- Con **:r nombre_fichero** insertamos un fichero.

5.2.- Otros editores de texto.

En UNIX se disponen de otros editores, como es el caso del editor ***emacs*** (edición de macros), que se invoca con:

$ *emacs* [nombre_de_fichero]

Al iniciarse muestra en pantalla algunas de las teclas de control. Entre ellas, las principales son las siguientes.

- <Ctrl-x> <Ctrl-c> - Salir.
- <Ctrl-h> - Ayuda.
- <Ctrl-x> <u> - Deshacer.
- <Ctrl-x> <Ctrl-s> - Guardar. (Según si se invocó *emacs* con nombre de fichero o no, se pide ahora el nombre del fichero. Al guardar, renombra la versión anterior del fichero añadiéndole el carácter "~").
- <Ctrl-x> <Ctrl-w> - Guardar como.
- <Ctrl-u> <Esc> <!> - Ejecutar una orden de UNIX desde *emacs*. (Muestra el mensaje *Shell command:* para que se introduzca un comando).

El comando ***cat***, que se verá posteriormente, concatena ficheros de texto, pero también entradas estándar por teclado, pudiendo utilizarse para editar textos sencillos.

Ejemplo:
```
$ cat > fichero  <Enter>
Esto es una prueba  <Enter>
con cat  <Enter>
<Ctrl-c> o <Ctrl-d>
$ cat fichero  <Enter>
Esto es una prueba
con cat
$
```

6.- Terminales virtuales y entorno gráfico.

6.1.- Terminales virtuales.

Cuando se ejecuta UNIX en un terminal físico, se dispone de hasta 6 terminales virtuales en los que se pueden abrir de forma simultánea 6 sesiones diferentes con el mismo o diferente usuario. Para conmutar entre los diferentes terminales virtuales se utiliza <Alt-Fn>, donde Fn corresponde a una tecla de función comprendida entre F1 y F6.

Una utilidad inmediata es, por ejemplo, mantener un terminal virtual como intérprete de comandos (shell) y otro como editor de textos.

Con el comando *ps*, que se verá posteriormente, se muestra en pantalla que los terminales (TTY) activos se corresponden a procesos */bin/login* y los no activos a */ sbin/mingetty*.

6.2.- Entorno gráfico.

En UNIX es posible la utilización de interfaces gráficas de usuario (IGU) que son entornos gráficos de ventanas, conocidos como X-Windows, siempre y cuando hayan sido instaladas previamente.

El sistema X-Windows es un potente entorno operativo gráfico que da soporte a muchas aplicaciones en red, al ser un sistema cliente/servidor. Linux permite ejecutar aplicaciones en ventana a través de una red heterogénea al incorporar la implementación XFree86 del estándar X11 de X-Windows para PCs, creado en el MIT. XFree86 y X-Windows también se pueden ejecutar en máquinas individuales.

X-Windows admite las posibilidades de multiprocesado de UNIX, por lo que XFree86 admite las posibilidades de multiprocesado de Linux. Cada ventana que se muestra bajo X-Windows puede ser una tarea distinta que se ejecuta bajo Linux.

Una vez instalado XFree86, el archivo *XF86Config* contiene los parámetros de configuración de la instalación de X-Windows: rutas completas a los archivos necesarios para el sistema, rutas a los directorios de tipos de letra, tipo de teclado, características del ratón, opciones del servidor, modos de vídeo, dispositivos y pantalla.

Red Hat Linux incluye una utilidad llamada *Xconfigurator*, la cual se trata de una herramienta controlada por medio de menús que hace preguntas sobre la tarjeta de vídeo, monitor y ratón, creando a partir de ellas un archivo *XF86Config*.

Cuando se trabaja con un entorno gráfico, la conmutación entre terminales virtuales se realiza con <Ctrl-Alt-Fn>, siendo Fn una tecla de función comprendida entre F1 y F7, utilizándose F7 para pasar al entorno gráfico, ya que es el terminal virtual asignado por Linux para tal propósito.

En la distribución Red Hat Linux, con la orden ***startx*** se ejecuta un servidor X con una pantalla muy similar al entorno Windows 95 de Microsoft. Algunas características son las siguientes:

- *Ventanas* que se pueden desplazar, cambiar de tamaño o cerrar.
- *Menú desplegable de arranque*, que también se activa haciendo clic con botón izquierdo del ratón en ventana central, y que permite desplegar sucesivos menús correspondientes a diferentes opciones, como son aplicaciones (entre ellas se encuentran los editores de texto gráficos *Emacs* y *Xedit*), juegos, multimedia, utilidades del sistema, etc...

- En los menús desplegados, aparece un botón para mostrar barras con iconos de aplicaciones.
    - *Panel de control* con los siguientes iconos:
        - Configuración del sistema de ficheros.
        - Manejador de paquetes.
        - Sistema de ayuda.
        - Configuración del núcleo.
        - Configuración del modem.
        - Configuración de red.
        - Configuración de impresora.
        - Hora y fecha.
        - Editor de niveles de ejecución del sistema.
        - Configuración de usuarios y grupos.
    - Ventana para el intérprete de comandos Linux.

Otra opción disponible de entorno gráfico X Windows es el sistema de ventanas que se ejecuta con ***openwin***.

$ [*/usr/openwin/bin/*]*openwin*

El path que va entre corchetes es opcional, al igual que para cualquier fichero ejecutable, dependiendo de si se ha incluido o no el path del fichero ejecutable con la variable de entorno PATH.

El sistema de ventanas abierto con *openwin* presenta las siguientes características:

- Inicialmente aparece una ventana para conmutar entre escritorios virtuales, que presenta un triángulo en la esquina superior izquierda, al igual que otras ventanas. Haciendo clic con el botón izquierdo del ratón en dicho triángulo, se minimiza la ventana en forma de icono, y con el botón derecho aparece un menú con diferentes opciones de manejo de ventanas.
- Al hacer clic con el botón derecho del ratón en la ventana central, se muestra una ventana con tres opciones: *Workspace* para trasladar menú a un área de trabajo de un escritorio virtual, *Programs* para desplegar un menú con diversas utilidades y *Exit* para salir del sistema de ventanas, pidiendo confirmación.

- Haciendo clic con el botón derecho del ratón en *Programs*, aparecen las siguientes opciones:
    - *cmdtool (CONSOLE)*; *cmdtool* y *shelltool*: Ventanas de entrada de comandos para el shell.
    - *Textedit*: Editor de texto.
    - *Properties*: Propiedades del área de trabajo: colores, menús, iconos, ratón,...
    - *Clock*: Muestra ventana de reloj.

6.2.1- Editores gráficos.

Como se ha indicado anteriormente, entre las aplicaciones X Windows, se encuentran los editores de texto gráficos, como son *Emacs* y *Xedit* con *startx*, y *Textedit* con *openwin*.

Textedit es un editor de texto con diversos menús de opciones desplegables, y que puede utilizarse para editar shell scripts de forma más cómoda que con el editor *vi*.

En la parte superior de la ventana del editor ***Textedit*** se incluyen 4 opciones de menús, que también se muestran en un menú haciendo clic con el botón derecho del ratón dentro de la ventana de edición. Al seleccionar cada opción aparece el correspondiente menú desplegado:

File: Abrir, salvar, salvar como, añadir fichero o limpiar documento.
View: Situar cursor en posiciones concretas.
Edit: Deshacer, copiar, pegar o cortar.
Find: Distintas opciones de búsqueda con o sin reemplazamiento.

7.- Comandos de manejo de ficheros de texto.

En este apartado se exponen algunos de los comandos utilizados para manejar ficheros de texto, incluyéndose una breve descripción de los mismos, así como algún ejemplo. Para obtener más información en cuanto a todas las opciones posibles, se pueden utilizar el manual on line *man*, la guía de comandos UNIX, o cualquier libro de UNIX.

Comando *cat*: Permite concatenar archivos. Si la salida de la concatenación se omite, *cat* muestra por pantalla el contenido de los ficheros especificados para la concatenación. Si se omite la entrada, el fichero de salida contendrá los caracteres que se introduzcan en el terminal. Por lo tanto, además de para concatenar ficheros, sirve para visualizarlos y crear nuevos archivos. Para especificar el fichero de salida, es necesario utilizar el carácter de redirección de salida (>). Si lo que se desea es crear un nuevo fichero, introduciendo su contenido por teclado, debe teclearse el carácter ^D para indicar fin de fichero.

Ejemplos: Crear fichero nuevo:

```
$ cat > saludos
hola, amigos!
^D
```

Listar contenido de fichero:

```
$ cat saludos
hola, amigos!
```

Listar contenido de varios ficheros:

```
$ cat saludos saludos
hola, amigos!
hola, amigos!
```

Concatenar fichero:

```
$ cat saludos saludos > saludos2
$ cat saludos2
hola, amigos!
hola, amigos!
```

Comando *pg*: Visualiza un fichero de texto pantalla a pantalla.

Ejemplos: $ *pg* fichero

$ *pg* +2 fichero

Comando *more*: Es un filtro que permite recorrer hacia abajo un fichero de texto pantalla a pantalla.

Ejemplo: $ *more* fichero

Comando *less*: Visualiza un fichero de texto pantalla a pantalla, pudiendo mover el fichero hacia atrás y hacia delante.

Ejemplo: $ *less* fichero

Comando *cmp*: Compara dos ficheros de texto o dos ficheros binarios, mostrando el número de línea y el primer carácter donde se encontró la primera diferencia.

Ejemplo: $ *cmp* fichero1 fichero2

Comando *diff*: Busca diferencias entre líneas de dos archivos o directorios.

Ejemplos: $ *diff* fichero1 fichero2

$ *diff* directorio1 directorio2

Comando *comm*: Este comando muestra las líneas comunes de dos ficheros ordenados. Saca un listado en tres columnas. En la primera columna muestra las líneas no coincidentes del primer fichero, en la segunda columna las líneas no coincidentes del segundo fichero y, en la tercera columna, las líneas comunes a los dos ficheros. La opción -[1][2][3] suprime la columna correspondiente.

Ejemplo:
```
$ cat f1
carlos
juan
pepe
^D
$ cat f2
alvaro
fernando
jose
juan
^D
$ comm f1 f2
        alvaro
carlos
        fernando
        jose
                juan
pepe
```

Comando *sort*: Ordena el contenido de un archivo en orden alfanumérico. Por defecto, la salida del fichero ordenado se muestra por pantalla. Para enviarla a un fichero hay que usar redireccionamiento. Presenta las siguientes opciones:

-b Ignora los espacios en blanco iniciales.

-d Usa solamente letras, dígitos y espacios en blanco para realizar la ordenación. Ignora los caracteres de puntuación y control.

-f No hace diferencias entre mayúsculas y minúsculas.

-n Los números se ordenan por sus valores aritméticos.

-o Almacena la salida en el fichero especificado.

-r Realiza la ordenación en orden inverso.

Ejemplos: Ordenar un fichero y mostrar su salida por pantalla:

```
$ cat nombres
manolo
isabel
luisa
ana
nuria
david
^D
$ sort nombres
ana
david
isabel
luisa
manolo
nuria
```

Ordenar un fichero y guardar el fichero ordenado en otro:

```
$ sort nombres > nombresorden
```

Ordenación numérica en orden inverso:

```
$ cat numeros
12
doce
7.1
siete
$ sort -nr numeros
12
7.1
siete
doce
```

Comando *wc*: Cuenta las líneas, palabras y caracteres de un fichero de texto. Su formato es:

wc [-*lwc*] fichero

donde:

-l Cuenta número de líneas

-w Cuenta número de palabras

-c Cuenta número de caracteres

Ejemplo: Contar el número de líneas de un fichero:

$ *wc -l* fichero

Comando *head*: Muestra las primeras líneas de un fichero especificado.

Ejemplo: $ *head* fichero

Comando *tail*: Recorta y muestra las últimas líneas de un fichero.

Ejemplo: Mostrar las últimas 100 líneas del archivo diario:

$ *tail* –100 diario

Comando *cut*: Recorta verticalmente campos de archivos.

Ejemplo: Imprimir los campos 3 y 6 del fichero datos. Los campos están separados por el carácter ":":

$ *cut* –f3,6 –d: datos

Comando *paste*: Une líneas de ficheros horizontalmente.

Ejemplo:
$ *cat* fich1
luis
juan
$ *cat* fich2
antonio
pepe
$ *paste –d" "* fich1 fich2
luis antonio
juan pepe

8.- Redireccionamiento y tuberías.

8.1.- Redireccionamiento de entrada/salida.

Todo proceso de UNIX tiene asociado una entrada estándar y una salida estándar. Normalmente los comandos leen datos a procesar desde el teclado y escriben en la pantalla, pero mediante las primitivas de ***redirección*** se pueden tomar los datos desde ficheros y enviar la salida a otros lugares que no sean la pantalla. El ***redireccionamiento***

especifica la entrada y la salida estándar de un comando. Para modificar la entrada estándar se utiliza el símbolo < y para modificar la salida estándar el símbolo >.

a) Redireccionamiento de la salida:

El siguiente comando no muestra el listado del directorio por pantalla, sino que lo envía al fichero salida. `$ ls -l > salida`

b) Redireccionamiento de la entrada:

El siguiente comando copia el contenido del fichero entrada en el fichero salida.

`$ cat < entrada >salida`

El operador ">" destruye la información del fichero de salida especificado. Si no queremos destruirla, sino que queremos añadir información a continuación del final del fichero, debemos utilizar el operador ">>".

Ejemplo:
```
$ cat > saludo
hola
^D
$ cat >> saludo
hola de nuevo
^D
$ cat saludo
hola
hola de nuevo
```

En resumen, los símbolos de redirección son:

<	Toma la entrada de un archivo.
>	Envía la salida a un archivo.
>>	Añade la salida a un archivo.

8.2.- Tuberías.

Los **pipes**, conocidos también como **tuberías**, **tubos** o **conducciones** son un mecanismo utilizado para comunicar procesos.

Un pipe (especificado con el símbolo "|") hace que la entrada del segundo comando sea la salida del primero. De esta forma se puede conectar una secuencia de comandos utilizando una conducción, en lugar de introducir cada comando independientemente y guardar los resultados en archivos intermedios.

Ejemplos:

El siguiente comando cuenta el número de ficheros y directorios del directorio actual: $ *ls -l | wc -l*

El siguiente comando lista el contenido del directorio actual ordenado alfanuméricamente: $ *ls -l | sort*

Se pueden combinar redireccionamientos y conducciones.

Ejemplo: $ *cat* ventas | *sort* > ventasordenado

8.- Comandos de interés especial.

Comando *passwd*: Permite establecer o cambiar la palabra clave de un usuario. La nueva palabra clave debe introducirse dos veces para que sea validada.

Ejemplos: Si no había palabra clave:
$ *passwd*
New UNIX password: *Palabra_clave_nueva*
Retype new UNIX password: *Palabra_clave_nueva*

Para modificar palabra clave existente:
$ *passwd*
Changing password for *Identificator_usuario*
(Current) UNIX password: *Palabra_clave_antigua*
New UNIX password: *Palabra_clave_nueva*
Retype new UNIX password: *Palabra_clave_nueva*

Nota: En clase no se debe cambiar la palabra clave de los terminales de la sala.

Comando *who*: Muestra los usuarios conectados al sistema. Con ***who am I*** (¿quién soy yo?) sólo se da información del propio usuario.

Ejemplo: $ *who*

root	tty1	Nov 7	8:27
raquel	tty3	Nov 7	8:38
paco	tty6	Nov 7	9:02

Comando *ps*: Muestra el estado de los procesos activos en el sistema. Según las opciones utilizadas, la información se dispone en diferentes columnas. Por defecto son: PID (número de identificación del proceso), TTY (número del terminal que controla el proceso), STAT

(estado del proceso), TIME (tiempo que el proceso lleva ejecutándose) y COMMAND (nombre de la orden).

El estado del proceso (STAT) viene dado por alguno de los siguientes códigos:

R - Ejecutando.
S - Dormido.
D - Espera de uso de disco.
T - Parado.
Z - Zombie (el proceso ha terminado o ha sido eliminado pero el proceso padre no lo ha comunicado al sistema).
W - Proceso sin páginas residentes.
P - Espera de páginas de memoria.

Cuando se utiliza sin ninguna opción, visualiza información acerca de los procesos activos del usuario. Algunas opciones son:

-a Se visualiza información sobre el estado de todos los procesos activos.

-u Imprime en formato de usuario mostrando el nombre de usuario y la hora de inicio.

-x Muestra los procesos que no son controlados por el terminal (lanzados por el sistema).

Ejemplo: $ *ps*

PID	TTY	STAT	TIME	COMMAND
357	1	S	0:00	/bin/login --root
1891	1	S	0:00	-bash
1904	1	R	0:00	ps

Comando *df*: Informa de la cantidad de espacio libre en disco.

Ejemplo: $ *df*

Fylesystem	1024-blocks	Used	Available	Capacity	Mounted on
/dev/hdc2	595195	458084	106367	81%	/

Comando *echo*: Toma los argumentos que se le pasan y los convierte en salida estándar (pantalla). Es útil en secuencias de instrucciones de shell para pedir una entrada o para informar del estado de un proceso.

La sintaxis es: *echo* [-*n*] [-*e*] cadena ...

cadena es la cadena de caracteres que se quiere sacar. También se pueden incluir variables con el símbolo $ delante (ejemplo $var), mostrando así el contenido de esas variables. Se admite la combinación de varias cadenas de caracteres y varias variables.

-n indica que se suprime la actuación normal (añadir una nueva línea después de la salida).

-e permite la interpretación de las siguientes secuencias de caracteres en la cadena:

\a	Alerta (señal acústica).
\b	Retroceso.
\c	No imprimir una nueva línea al final.
\f	Salto de página.
\n	Nueva línea.
\r	Retorno de carro.
\t	Tabulador.
\v	Tabulador vertical.
\\	Una barra inclinada inversa.
\nnn	El carácter cuyo código octal ASCII es nnn.

Ejemplos:

```
$ echo "Juan"
Juan
$ echo Juan
Juan
$ echo 'Juan'
Juan
$ echo
$
$ echo –e "\a"
(Señal acústica)
$ echo –e "Teclea S o N \c"
Teclea S o N
$ var=20
$ echo $var     (Variables de usuario)
20
$ echo $SHELL    (Variables de entorno)
/bin/bash
$ echo $PATH
/usr/local/bin : /bin : /usr/bin : /home/usuario/bin
$ echo datos $var $ SHELL
datos 20 /bin/bash
```

Comando *printf*: Muestra datos dando formato. Se parece al *printf* del lenguaje de programación C. La sintaxis es:

printf formato [argumentos]

El formato es una cadena de caracteres que se muestran en pantalla junto con una serie de órdenes de formato o **secuencias de escape** que definen la manera en que se visualizan los argumentos:

\"	Doble comillas.
\onnn	Carácter con valor octal nnn (0 a 3 dígitos).
\\	Una barra inclinada inversa
\a	Alerta (señal acústica).
\b	Retroceso.
\c	No imprimir una nueva línea al final.
\f	Salto de página.
\n	Nueva línea.
\r	Retorno de carro.
\t	Tabulador horizontal.
\v	Tabulador vertical.
\xnnn	Carácter con valor hexadecimal nnn (1 a 3 dígitos).
%%	Un carácter %.

Los argumentos que se escriben con el comando *printf*, se incluyen formateados con el mismo orden dentro del formato, utilizándose para ello **controles** parecidos al C. Algunos de estos controles son los siguientes:

%b	Cadena de caracteres.
%c	Un único carácter.
%d	Decimal.
%x	Hexadecimal.
%e	Notación científica.

Ejemplos:

```
$ printf "\nPrueba: %b %d %e %c\n" mar $var $var blas

Prueba: mar 23 2.300000e+01 b
$
$ printf "\a"
```

(produce un pitido)

10.- Otros comandos.

Comando *grep*: Busca una secuencia especificada (cadena de caracteres) en un archivo o lista de archivos. Como primer parámetro se debe especificar la cadena a buscar y, posteriormente, indicar los nombres de cada uno de los ficheros en los que se quiere realizar la búsqueda (se pueden usar comodines). Si no se indica ninguna opción, *grep* muestra las líneas en las que se ha encontrado la cadena especificada.

Para especificar la cadena se pueden usar metacaracteres. El metacarácter "." sustituye cualquier carácter. El metacarácter "^" indica principio de línea y "$" final de línea.

Algunas opciones de *grep* son:

-c Cuenta las líneas concordantes.

-i No distingue entre mayúsculas y minúsculas a la hora de realizar la búsqueda.

-l Sólo se muestran los nombres de los archivos en los que se encontró la cadena especificada.

-n Muestra delante de cada línea el número de línea relativo dentro del fichero.

-v Visualiza sólo aquellas líneas que no coinciden con el patrón indicado.

Ejemplos:

Buscar la cadena "hola", sin hacer distinción entre mayúsculas y minúsculas, en los ficheros f1.txt y f2.txt:

$ *grep -i* hola f1.txt f2.txt

Mostrar los nombres de los ficheros, dentro del directorio etc, en los que aparecen cadenas de 5 caracteres que empiezan y terminan por la letra a:

$ *grep -l* "a...a" /etc/*

Contar el número de líneas, dentro del fichero fich, que comienzan por "Si":

$ *grep -c* "^Si" fich

Otros ejemplos:

```
$ cat > miarchivo
Desearía que hubiera una forma mejor de aprender
UNIX. Algo como tomar una píldora diaria de UNIX.
<Ctrl-d>
$ grep UNIX miarchivo
UNIX. Algo como tomar una píldora diaria de UNIX.
$ grep dos p1 p2
p1: Hay dos libros.
p2: Estos son los dados.
p2: Están atados.
$ grep "#include" *.c
```

Comando *find*: Busca en la estructura de directorios, a partir del directorio indicado, el fichero o ficheros que correspondan con un patrón determinado. Como primer parámetro, hay que indicar el directorio a partir del cual se va a realizar la búsqueda. El resto de los parámetros detallan los criterios de la búsqueda y las acciones a llevar a cabo sobre los ficheros encontrados.

Algunos **criterios de búsqueda** son:

-name "nombre" Busca los fichero con nombre "*nombre*".

-size t Busca los ficheros que tienen un tamaño de *t* bloques.

-size -t Busca los ficheros que tienen un tamaño inferior a *t* bloques.

-size +t Busca los ficheros que tienen un tamaño superior a *t* bloques.

-atime 0 Busca los ficheros accedidos en el día de hoy.

-atime -d Busca los ficheros accedidos hace menos de *d* días.

-atime +d Busca los ficheros accedidos hace más de *d* días.

-user prop Busca los ficheros cuyo propietario es *prop*.

-newer fich Busca los ficheros más recientes que el fichero *fich*.

Los criterios de búsqueda pueden combinarse para formar criterios compuestos. Si lo que queremos es indicar que se cumplan varios criterios aunque no se den al mismo tiempo, debemos especificarlos utilizando el operador ***-o*** entre dichos criterios.

Las **acciones a llevar a cabo** sobre los ficheros encontrados pueden ser:

-print Escribe en pantalla los nombres de los ficheros encontrados.

***-exec comando *** Ejecuta el comando *comando*. Para especificar que los ficheros encontrados actúen como parámetros de la orden, debemos indicarlo sustituyendo el valor del parámetro correspondiente por los caracteres **{}**.

Se pueden omitir criterios de búsqueda y acciones a llevar a cabo.

Ejemplos:

Buscar los ficheros, a partir del directorio actual, cuyos nombres comiencen por a, b o c e imprimirlos por pantalla:

$ *find* . *-name* "[abc]*" *-print*

Imprimir los nombre de los ficheros que cuelguen de los subdirectorios D1 o D2 del directorio actual y comiencen por una letra mayúscula:

$ *find* ./D1 ./D2 *-name* "[A-Z]*" *-print*

Encontrar los ficheros, en toda la estructura de directorios, que tengan un tamaño superior a 100 bloques:

$ *find* / *-size* +100 *-print*

Buscar los ficheros accedidos en la última semana:

$ *find* / *-atime* -7 *-print*

Encontrar todos los ficheros con extensión txt cuyo propietario sea alum:

$ *find* / *-name* "*.txt" *-user* alum *-print*

Imprimir el contenido de todos los ficheros cuya extensión sea txt:

$ *find / -name* "*.txt" *-exec cat {} *

Mostrar ficheros y directorios del directorio /etc que empiecen por p:

$ *find* /etc/p*

Mostrar directorios del directorio /etc que empiecen por p:

$ *find* /etc/p* *-type d*

Comando *cal*: Devuelve un calendario del año [y el mes] especificados.

Ejemplos: $ *cal*

$ *cal* 12 1998

$ *cal* 1998 | *more*

Comando *date*: Retorna el día y la hora del sistema. El superusuario (root) puede establecer el día y la hora.

Ejemplo: $ *date*

Sat Nov 7 21:18:24 MET 1998

Comando *time*: Determina el tiempo que tarda un programa en ejecutarse. Se escribe el nombre del programa o comando a ejecutarse detrás del comando *time*.

Ejemplos: $ *time ls-l*

$ *time vi* (se sale con :q!)

Comando *sleep*: Suspende la ejecución durante un intervalo de tiempo (en segundos por defecto), pudiendo especificar segundos (***s***), minutos (***m***), horas (***h***) o días (***d***).

Ejemplos: $ *sleep* 10

$ *sleep* 5*s*

$ *sleep* 1*m*

$ *sleep* 2*h*

$ *sleep* 3*d*

Con <Ctrl-c> se aborta la ejecución de cualquier comando.

Comando *clear*: Limpia la pantalla y sitúa el prompt del sistema en la primera línea.

Ejemplo: *$ clear*

Comando *alias*: Permite definir un nombre (alias) para un comando. La configuración de un alias desde la línea de comandos lo mantiene vigente sólo durante la sesión que se esté ejecutando. Para que el alias se active cada vez que se conecta al sistema, se debe incluir su definición en el archivo .profile o .login.

Ejemplo: $ *alias* listar=”*ls –l*”
$ listar

Comando *kill*: Permite el envío de una señal a un proceso que está ejecutándose actualmente. Normalmente se utiliza este comando para detener el proceso de ejecución, de ahí el nombre *kill*, ya que se utiliza para “matar” (kill) el proceso. La sintaxis es:

$ *kill* [-señal] PID

PID es el identificador del proceso al que se quiere enviar la señal especificada. Con el comando *ps* se puede ver el PID de un proceso.

La expresión -señal se refiere al nombre o número de la señal a enviar.

Con ***kill –l*** se imprime la lista de nombres de señal que pueden ser utilizados con *kill*, bien con el nombre de señal o bien con el número de señal.

La señal predeterminada es **SIGTERM**.

La señal **SIGKILL** (**9**) para finalizar el proceso no puede ser ignorada por ningún proceso, salvo cuando el proceso está utilizando un servicio del kernel y no puede recibir señales. En este caso los procesos pueden quedar bloqueados, siendo necesario para resolverlo que el administrador del sistema lo apague.

Ejemplo: $ *kill* –9 125

11.- Intérprete de comandos de UNIX.

11.1.- La *shell* como intérprete de comandos.

Al comenzar una sesión de UNIX, la interacción con el sistema es controlada por un programa llamado *shell* o intérprete de comandos, y por medio de éste es como el usuario se entiende con el sistema. La *shell* es un programa que interpreta y ejecuta los comandos conforme se proporcionan desde el terminal. No se requiere ningún privilegio especial para ejecutarla, ya que para el kernel (núcleo del sistema) es un programa más.

Entre las características más comunes de la *shell* están la interpretación de guiones de *shell* (*shell scripts*), la expansión de comodines en nombres de archivos, la combinación de comandos para formar interconexiones, la recuperación de comandos previos, etc. Un **guión de *shell*** es un archivo que contiene secuencias de comandos de *shell*, igual que si se hubieran tecleado. Los guiones de *shell* permiten adecuar el entorno de trabajo sin tener que realizar programación "real".

La *shell* se utiliza con tres propósitos principales:

1) Como intérprete de comandos: Cuando la *shell* se utiliza como intérprete de comandos, el sistema espera recibir comandos en el prompt. Los comandos pueden contener símbolos especiales (p.e. comodines) que permiten abreviar nombres de archivos o redireccionar la salida y la entrada de un comando.

2) Para personalizar el ambiente de trabajo dentro de una sesión: Una *shell* de UNIX define variables para controlar el comportamiento del sistema cuando se entra en él. Este tipo de variables definen, por ejemplo, cuál será el directorio que se usará como directorio hogar (*home directory*), o el archivo donde se almacenará el correo. El sistema define estas variables por sí mismo; el usuario puede definir otras variables en los archivos de inicialización que son leídos y ejecutados cada vez que se accede al sistema.

3) Como lenguaje de programación: Todas las *shells* en UNIX contienen un conjunto especial de comandos que pueden usarse para crear *programas de shell*. De hecho, muchos de los comandos de *shell* pueden usarse como si fueran comandos de UNIX, y viceversa. Los *programas de shell* son útiles para ejecutar comandos de forma

iterativa o condicional como en un lenguaje de alto nivel. También pueden ejecutarse comandos de forma secuencial mediante archivos ejecutables.

Los **pasos** que se llevan a cabo al ejecutar un comando con la *shell* **como intérprete de comandos** son los siguientes:

1.- La *shell* visualiza el prompt del sistema y espera a que se teclee un comando.

2.- El usuario introduce una línea de comando, terminándola con <Intro>.

3.- La *shell* analiza el comando, y localiza el programa correspondiente.

4.- La *shell* solicita al sistema la ejecución del programa, o devuelve un mensaje de error.

5.- Cuando el programa termina, el control vuelve a la *shell*, que nuevamente visualiza el prompt del sistema.

11.2.- Tipos de *shell*.

En UNIX se disponen diferentes tipos de *shells*, pudiendo el usuario cambiar la *shell* que se esté ejecutando.

El comando *echo* permite la visualización de la *shell* que se está ejecutando:

```
$ echo $SHELL
/bin/sh
$
```

Al dar de alta un usuario, se produce una entrada en el archivo de contraseñas de usuarios (*/etc/passwd*). Entre los campos de dicha entrada se encuentra la *shell* utilizada por el usuario al entrar:

login_name: Nombre para entrar.

encrypted_password: Palabra clave para entrar.

user_ID: Nombre o número que UNIX utiliza para identificar al usuario.

group_ID: Nombre del grupo del usuario.

user_information: Descripción del usuario.

login_directory: Directorio inicial del usuario.

login_shell: *Shell* utilizada por el usuario al entrar.

Entre las *shells* más utilizadas se tienen las siguientes.

Bourne-shell:

Es la más antigua y la primera de las *shells* principales. Fue desarrollada en AT&T por Steven R. Bourne a comienzo de los setenta. Utiliza como prompt del sistema el símbolo "$", está disponible a través del comando ***sh*** y el fichero ejecutable es */bin/sh*. Ésta es la *shell* oficial que se distribuye con los sistemas UNIX. Muchos usuarios la prefieren para uso interactivo. Casi todos los guiones de *shell* siguen los convencionalismos de la *Bourne-shell*.

En resumen, la *Bourne-shell* proporciona:

- Una sintaxis sencilla para escribir programas en lenguaje *shell*.
- Compatibilidad con otros tipos de *shells*.
- Un estándar para casi todos los sistemas UNIX.

Entre los recursos importantes que ofrece están los siguientes:

- Operadores para la ejecución en segundo plano (*background*, símbolo "&"), o ejecución condicional de comandos.
- Variables sustituibles, tanto nombradas como numeradas (parámetros). Estas últimas son los que contienen los argumentos de un comando ("$0", "$1", ...).
- Exportación de variables específicas a un proceso hijo.
- Tres formas de entrecomillado.
- Ejecución de comandos en *subshells*.
- Notificación automática de llegada de correo electrónico.
- Inclusión de datos de entrada para un comando en un *guión de shell* como parte del mismo.
- Ejecución de comandos en archivos de inicialización antes de leer cualquier entrada.

Bourne-Again-shell:

Es una copia de la *Bourne-shell*, que es la predeterminada bajo Linux. Utiliza como prompt del sistema el símbolo "$", y está disponible a través del comando ***bash*** correspondiente al fichero ejecutable */bin/bash*. Proporciona varias características optimizadas como, por ejemplo, edición de comandos, histórico de comandos y terminación de comandos.

C-shell:

Se desarrolló como parte de BSD UNIX. Su sintaxis es similar a la del lenguaje de programación C, y es más compleja y proporciona utilidades más potentes (mediante determinados comandos) que la *Bourne-shell*. Utiliza como prompt del sistema el símbolo "%", y está disponible a través del comando ***csh*** correspondiente al fichero ejecutable */ bin/csh*.

Algunas de sus características más comunes son:

- La posibilidad de recuperar comandos previos mediante un *mecanismo de "historia"*. Dispone de un buffer de historia de comandos ejecutados. El comando ***history*** permite a la *shell* recordar los comandos que ha ido ejecutando, pudiendo seleccionar comandos, o partes de comandos previos.
- *Control de trabajos* o capacidad de conmutar entre procesos y controlar su avance.
- Formas flexibles de sustitución de variables.
- Se pueden definir *alias* mediante el comando ***alias***, que permite abreviar líneas de comando o reemplazar nombres de comandos por otros nombres más familiares o descriptivos.
- *Compleción* del nombre de ficheros. Se puede introducir sólo una parte del nombre del fichero, siendo la *shell* la encargada de completarlo. Es útil también para pedir listados de ficheros que contengan un prefijo común.
- Operadores adicionales, como en el lenguaje C.
- Notificación automática de llegada de correo electrónico.
- El uso del carácter "~" para representar el directorio base de un usuario cuando va siguiendo a un *login*.
- La cadena de la forma *a{x1, x2, ... , xn}b* en un nombre de archivo representa una lista de nombres.
- Capacidad para calcular expresiones numéricas generales y asignar el resultado a una variable.

Korn-shell:

Es la más nueva y no del todo conocida. Se trata de una síntesis de *Bourne-shell* (sintaxis) y *C-shell* (características), además de características propias, siendo una *shell* potente, rápida y fácil de utilizar. Creada por David Korn de los Laboratorios Bell de AT&T en 1982, presentando versiones mejoradas en 1986 y 1988. Se incluye como

característica estándar de la versión 4 de System V y otros sistemas, y sigue los convencionalismos de la *Bourne-shell*.

Utiliza como prompt del sistema el símbolo "$", y está disponible a través del comando ***ksh*** correspondiente al fichero ejecutable */bin/ksh*.

Proporciona las siguientes características:

- Edición interactiva de la línea de comandos, incluida la complementación de nombres de archivo, con las mismas características de la *C-shell*.
- Posibilidad de editar la lista de comandos efectuados. La edición on-line es exclusiva de *Korn-shell*, permitiendo volver a llamar a comandos previos y editarlos. Está basado en la edición de *vi*, *emacs* y *gmacs*.
- Mejores definiciones de funciones que ofrecen variables locales y permiten escribir funciones recursivas.
- Comparación extendida de patrones para nombres de archivos y otras construcciones.
- Capacidad de extraer la porción de una cadena especificada por un patrón.
- Capacidad para cambiar.
- Notificación automática de llegada de correo electrónico.
- Tiempo de ejecución más rápido.

C-shell mejorada:

La *C-shell mejorada* es una versión mejorada de la *C-shell* que ha adquirido mucha popularidad. Algunos de los recursos adicionales que ofrece son:

- Capacidad para editar la línea de comandos de forma interactiva.
- Llamada sencilla a comandos ejecutados con anterioridad, los cuales se pueden editar. Complementación interactiva de nombres de archivos y comandos.
- Marcas de la hora en la lista histórica.
- Capacidad para programar la ejecución periódica de un comando.

Shells restringidas:

Tienen un entorno de ejecución con posibilidades más controladas que las *shells estándar*. Proporcionan seguridad, limitando los comandos que el usuario puede ejecutar.

La ***Bourne-shell restringida*** utiliza como prompt del sistema el símbolo "$", y está disponible a través del comando ***rsh*** correspondiente al fichero ejecutable */bin/rsh*.

La ***Korn-shell restringida*** utiliza como prompt del sistema el símbolo "$", y está disponible a través del comando ***rksh*** correspondiente al fichero ejecutable */bin/rksh*.

En resumen, los ficheros ejecutables para diferentes tipos de *shell* y el prompt del sistema por defecto de cada una son los siguientes:

Shell	**Fichero ejecutable**	**Prompt**
*Bourne (**sh**)*	*/bin/sh*	*$*
*Bourne-Again (**bash**)*	*/bin/bash*	*$*
*C (**csh**)*	*/bin/csh*	*%*
*Korn (**ksh**)*	*/bin/ksh*	*$*
*Bourne restringida (**rsh**)*	*/bin/rsh*	*$*
*Korn restringida (**rksh**)*	*/bin/rksh*	*$*

Puede realizarse el **cambio de la *shell*** que está en ejecución, salvo en el caso de las *shells restringidas*. El cambio puede ser ***temporal*** o ***permanente***.

a) Cambio temporal: invocando a la *shell* que se desea ejecutar. Por ejemplo:

```
$ csh
%
.
.
.
% exit
$
```

El comando ***exit*** o <Ctrl-d>, hacen que se vuelva a la *shell* original.

También se puede cambiar la *shell* mediante la asignación $ *SHELL=/bin/csh.*

b) Cambio permanente: utilizando el comando ***chsh*** (*change shell*).

Ejemplo:
```
$ chsh
Password: PuestoN
Changing the login shell for usuario
Enter the new value, or press return for the default
Login shell [/bin/bash]: /bin/csh
```

Una vez ejecutado este comando, se debe volver a ejecutar el fichero de login para poder regresar a la *shell* original.

11.3.- Variables de *shell*.

Al igual que cualquier lenguaje de programación, *shell* permite el uso de *variables de shell*, pudiendo soportar dos tipos de variables: **variables de entorno** y **variables de usuario**.

11.3.1.- Variables de entorno.

Las **variables de entorno** se conocen también como **variables estándar**, y son aquellas que definen el entorno de trabajo. Sus nombres son conocidos por el sistema, y se utilizan para mantener control de las cosas esenciales. Normalmente son creadas por la propia *shell* en concordancia con el modo y las características dadas por el administrador del sistema al dar de alta al usuario.

Se puede cambiar el valor de las variables de entorno, pero son cambios temporales y se aplican sólo a la sesión actual. La próxima vez que se haga una conexión al sistema, éste las asignará nuevamente. Si se quiere que los cambios sean permanentes, se deben incluir en los archivos de inicialización *.profile* o *.login*. El hecho de que sean redefinibles por el usuario agrega flexibilidad al sistema y permite adaptar el entorno al gusto o necesidades de cada uno.

Para visualizar el conjunto de variables de entorno asignadas se utiliza el comando ***set***.

Ejemplo:
```
$ set      <Intro>
HOME=/usr/estudiantes/david
LOGNAME=david
PATH=:/bin:/usr/bin
PS1=”$”
PS2=“>”

$
```

Los nombres de las variables de entorno a la izquierda del signo igual (“=”) se muestran en letras mayúsculas. Para los nombres de variables, en general, se pueden utilizar minúsculas, mayúsculas o cualquier combinación de ellas, siendo necesario especificar el nombre exacto de la variable (incluyendo mayúsculas) para referirse a una variable.

Con el comando *echo* se muestra el valor de la variable de entorno especificada precedida de $.

Ejemplo: $ *echo $SHELL* <Intro>
/bin/bash
$

Las principales variables de entorno se explican seguidamente.

***HOME* (Camino de acceso al directorio particular del usuario)**

Cuando un usuario se conecta al sistema, la *shell* asigna la ruta de acceso completa de su directorio de conexión a la variable *HOME*. La variable *HOME* se utiliza por algunas órdenes de UNIX para localizar el directorio de conexión. Por ejemplo, el comando *cd* sin argumentos comprueba esta variable para determinar la ruta de acceso del directorio de conexión, y luego fija el sistema al directorio de conexión del usuario.

***PATH* (Caminos de acceso)**

La variable *PATH* contiene los nombres de los directorios en los que la *shell* busca para la localización de los comandos o programas (ficheros ejecutables) en la estructura del directorio. Por ejemplo, *PATH=:/bin:/usr/bin*.

Los diferentes caminos están separados por dos puntos ("**:**"). Si el primer carácter es dos puntos, la shell lo interpreta como "**.:**" (punto, dos puntos), lo que significa que el directorio actual es el primero en la lista y es el primero en el que se busca.

Si todos los archivos ejecutables de un usuario se localizan en un subdirectorio llamado *mibin*, por ejemplo, que se encuentra en el directorio de conexión, se podría añadir al *PATH* escribiendo *PATH=:/bin:/usr/bin:$HOME/mibin* y tecleando <Intro>.

***SHELL* (Tipo de *shell*)**

La variable *SHELL* contiene la ruta de acceso completa de la *shell* de conexión.

Ejemplo: *SHELL=/bin/sh*

***PS1* (Indicador primario)**

La variable *PS1* almacena la cadena utilizada como signo de indicador (prompt del sistema). Por ejemplo, el signo del indicador principal de la *Bourne-shell* es el signo de dólar ("$").

***PS2* (Indicador secundario)**

La variable *PS2* asigna el signo del indicador que se visualiza siempre que se teclee <Intro> antes de finalizar la línea de comando, y la *shell* espera el resto del comando. Por ejemplo, el indicador secundario del *Bourne-shell* es el signo de mayor que (">").

Otras variables de entorno son las siguientes:

***IFS* (Separador de campo interno en línea de comandos)** (" " por defecto)
***TERM* (Tipo de terminal que se está utilizando)**
***TZ* (Zona horaria para el comando *date*)**
***USERNAME* o *LOGNAME* (Nombre de usuario para entrar al sistema)**
***MAIL* (Nombre completo de ruta del buzón de correo)**

Hay algunas variables de entorno que dependen del tipo de *shell*.

11.3.2.- Variables de usuario.

Las variables de usuario son definidas por el usuario cuando escribe un *guión de shell*. Se pueden modificar y utilizar libremente con el programa de *shell*.

La principal diferencia entre variables de *shell* y otros lenguajes de programación reside en que en la programación de *shell* las variables no son de tipo definido, es decir, no hay que especificar si una variable es una cadena, un entero, etc.

La definición y asignación de valores a variables dependerá del tipo de *shell* utilizada. Así, con *Bourne-shell* la sintaxis es: *nombre_variable=valor_variable* (sin que haya espacios ni antes ni después del igual ("=")).

Con *C-shell* la sintaxis es: *set nombre_variable=valor_variable* (puede haber espacios antes y después del igual).

El nombre de una variable de la *shell* empezará por una letra (mayúscula o minúscula) y no por un número.

Para acceder al valor de una variable se antepone al nombre de la variable el signo de dólar ("$"). Por ejemplo, con el comando *echo* se muestra el valor de la variable de usuario especificada precedida de $. Así, si se tiene una variable VAR=15:

```
$ echo $VAR    <Intro>
15
$
```

El comando ***unset*** se utiliza para eliminar una variable. Por ejemplo, si se quiere eliminar la variable VAR, se escribirá:

```
$ unset VAR      <Intro>
```

11.4.- Expresiones numéricas.

Las variables en la *shell* son únicamente alfanuméricas y para darle tratamiento numérico hay que recurrir al comando ***expr***, que evalúa expresiones aritméticas.

Este comando trabaja con los **operadores aritméticos** *suma* (+), *resta* (-), *multiplicación* (*), *cociente de división* (/) y *resto de división* (%).

Sólo admite números enteros y hay que dejar un espacio entre el operador y cada término de la expresión.

Ejemplo:
```
$ expr 3 + 4
7
```

Los operadores de la multiplicación y de la división tienen precedencia más baja que el de la suma. Para cambiar esta precedencia es necesario usar paréntesis.

También se pueden utilizar **operadores relacionales** como *mayor que* (>), *mayor o igual que* (>=), *menor que* (<), *menor o igual que* (<=), *igual que* (=) o *diferente a* (!=).

Utilizando comillas simples inversas se pueden asignar comandos a variables.

Ejemplo:
```
$ n=`pwd`
$ echo $n
/usr/ana
```

Aplicando la sustitución de un comando por su resultado, se puede dar tratamiento numérico al contenido de una variable.

Ejemplo:
```
$ a=3
$ a=`expr $a + 2`
```

$ *echo* $a
5

11.5.- Lectura de variables desde teclado.

El comando ***read*** (no disponible en ***csh***) acepta datos desde el teclado o provenientes de un fichero mediante redirección, y los almacena en la variable indicada. Se pueden aceptar datos para más de una variable con el mismo *read*. En tal caso, la primera variable tomará la primera cadena, la segunda variable la segunda cadena, y así sucesivamente. Si sobran cadenas al terminar la distribución, quedan asignadas a la última variable. Igualmente, cualquier variable sobrante no se asigna.

Para que se produzca una interactividad entre la máquina y el usuario, el comando *read* se puede combinar con la utilización del comando *echo*.

Ejemplo:
$ *cat* > nombre
echo Introduce un nombre:
read Nombre Apellidos
echo Tu nombre es $Nombre
echo Tus apellidos son $Apellidos
<Ctrl-d>
$ *sh* nombre
Introduce tu nombre:
Antonio Luque Portero
Tu nombre es Antonio
Tus apellidos son Luque Portero
$

11.6.- Metacaracteres de la *shell*.

El sistema operativo UNIX reconoce algunos caracteres especiales conocidos como **metacaracteres** que sirven para realizar funciones específicas de *shell*. Seguidamente se indican estos metacaracteres, algunos de los cuales ya se han explicado en apartados anteriores.

- ***** Carácter comodín que se sustituye por cualquier cadena de caracteres, incluida la cadena vacía.
- **?** Carácter comodín que se sustituye por un carácter cualquiera.
- **[...]** Se sustituye por cualquier carácter de los encerrados entre paréntesis.
- < Redirecciona la entrada estándar (toma la entrada de un archivo).

> Redirecciona la salida estándar (envía la salida a un archivo).

>> Redirecciona la salida estándar añadiéndola (añade la salida a un archivo).

>& Se utiliza para redireccionar la salida de error.

| Se utiliza para crear pipes (tuberías o conducciones) que "conducen" la salida de un comando o proceso a la entrada de otro.

; Permite escribir varios comandos (separados por ";") en una línea de comandos.

& Se utiliza detrás de nombres de procesos para ejecutarlos en segundo plano (*background*).

= Se utiliza para asignar el valor (derecha) a una variable (izquierda).

**$** Se utiliza delante del nombre de una variable para referirse al valor de dicha variable. Si el nombre de la variable se pone entre llaves (${variable}) se evitan confusiones cuando la variable está concatenada con texto.

Se utiliza para añadir comentarios en *shell scripts*.

( ) Sirve para agrupar una serie de comandos separados por punto y coma (";") que se ejecutan secuencialmente.

**** Anula la interpretación de los metacaracteres que le siguen (p.e. *echo* * muestra en pantalla un asterisco).

`...` Las comillas simples inversas permiten ejecutar el comando entrecomillado y asignar su salida a una variable (p.e. listar =`ls -l`).

'...' Los apóstrofes o comillas simples hacen que caracteres especiales entrecomillados pierdan su significado (p.e. *echo* '$var' muestra $var en lugar del valor de la variable var).

"..." Las comillas dobles permiten que una cadena con espacios en blanco entrecomillada sea interpretada como una única entrada, en lugar de interpretarla como más de una (p.e. cadena="abc def").

&& Ejecutar lo que sigue a && si se ejecuta lo que le precede (Y progresivo).

|| Ejecutar lo que sigue a || si no se ejecuta lo que le precede (O progresivo).

$# Variable incorporada que almacena el número de parámetros posicionales pasados a un programa (*shell script*).

$0 Variable incorporada que almacena el nombre del programa de *shell*.

$n Variable incorporada que almacena el parámetro o argumento pasado en el momento de llamar a un programa en la posición n, siendo n=1, 2, 3, ...

$* Variable incorporada que almacena en una única cadena todos los argumentos pasados en el momento de llamar al programa de *shell*.

$? Variable incorporada que almacena el código de la última orden o programa de *shell* ejecutado desde el programa de *shell*.

Ejemplo de variables incorporadas:

$ *cat* prueba
echo "El nombre del comando es" $0
echo "El segundo parámetro es" $2
echo "El número total de parámetros es" $# " y los parámetros son" $*
$ prueba p1 p2 p3 p4
El nombre del comando es prueba
El segundo parámetro es p2
El número total de parámetros es 4 y los parámetros son p1 p2 p3 p4

12.- Programación en la *shell*.

12.1.- Programación de *shell scripts*.

Todas las *shells* en UNIX contienen un conjunto especial de comandos que pueden usarse para crear *programas de shell*, conocidos como *guiones de shell* o *shell scripts*. Los *shell scripts* son útiles para ejecutar comandos de forma iterativa o condicional como en un lenguaje de alto nivel. También pueden ejecutarse comandos de forma secuencial mediante archivos ejecutables.

En todos los sistemas operativos, los usuarios tienden a repetir el uso de determinada secuencia de comandos, porque esta secuencia en sí realiza una tarea específica. Agrupar comandos en un programa da como beneficio para el usuario un ahorro considerable de tiempo. Para crear este tipo de utilidades de tipo *batch* en UNIX, se dispone del lenguaje de programación de la propia *shell*. Es un lenguaje interpretado, lo que significa que cada línea del programa es leída y a continuación ejecutada por la propia *shell*. Esta característica le proporciona una ventaja notable sobre los lenguajes que requieren compilador, en cuanto se refiere al mantenimiento de los programas. Los programas se escriben en código ASCII y para modificarlos sólo hay que editarlos nuevamente y realizar los cambios necesarios.

El lenguaje de programación *shell* es mucho más de lo que se necesita para hacer utilidades tipo *batch*. Es un lenguaje completo, de alto nivel, que incluye las estructuras fundamentales de la programación. Maneja variables, dispone de instrucciones que controlan procesos iterativos tales como ***for***, ***while*** y ***until***, o condicionales como ***if*** o ***case***, permite parámetros de sustitución como las variables incorporadas, y una serie de instrucciones adecuadas para desarrollar utilidades muy sofisticadas.

Lo más atractivo del lenguaje de programación *shell* es la posibilidad de emplear todos los comandos del sistema dentro de los propios programas o *shell scripts*, pudiéndose combinar con las instrucciones del lenguaje *shell* para crear innumerables aplicaciones.

Los *shell scripts* son también de gran ayuda para el administrador del sistema: programas pequeños realizan excelentes tareas de control de los recursos del sistema y se encargan de mantener la seguridad del sistema.

Una vez editado el *shell script* con un nombre cualquiera, existen varias formas para ejecutarlo:

a) Se puede convertir en un archivo ejecutable modificando los permisos, que en principio son sólo de lectura y escritura, mediante el comando *chmod*. Luego se ejecutará como cualquier otro comando de UNIX desde la línea de comandos, añadiendo al nombre del *shell script* los argumentos necesarios.

Para que el archivo con el permiso de ejecución activado pueda ejecutarse, debe estar en uno de los caminos de búsqueda especificados en la variable de entorno *PATH*. Si el directorio en el que se encuentra el archivo en cuestión no se encuentra en la ruta de búsqueda, habrá que añadir a ésta el nombre del directorio.

b) Se puede ejecutar un archivo de *shell script* sin permiso de ejecución activado, precediéndolo de un nombre de *shell*: *nombre_de_shell nombre_de_archivo*.

Ejemplo: $ *sh* mifichero <Intro>

De esta forma, se ejecuta un archivo en una *shell* determinada. Se llama primero a la *shell* especificada, y se pasa luego el nombre del archivo como parámetro para ejecutar el archivo.

12.2.- Estructuras de control.

Según el tipo de *shell* utilizada la sintaxis de las estructuras de control es diferente, aunque el funcionamiento es el mismo. A continuación se explican las estructuras de control utilizadas para la programación de *shell scripts* en la *Bourne-shell (**sh***) (*shell* más extendida), compatibles con la *Bourne-Again-shell* (***bash***), que es la predeterminada de Linux.

12.2.1.- Estructuras condicionales.

Las sentencias condicionales se utilizan en los *shell scripts* para decidir la parte del programa que se ejecutará dependiendo de las condiciones especificadas.

12.2.1.1.- Comparación de expresiones.

El **comando *test*** realiza pruebas sobre ficheros, directorios, cadenas y enteros. Permite la comparación lógica de dos operadores (numéricos o cadenas).

La sintaxis del comando *test* en la *Bourne-shell* es la siguiente:

test expresión

o bien

[expresión]

Ambos métodos son procesados de la misma forma, devolviéndose un 0 para ejecución correcta o verdadera o un 1 para ejecución incorrecta o falsa.

Con el comando *test* se pueden realizar los siguientes tipos de comparaciones:

- Comparación de cadenas.
- Comparación numérica.
- Operadores de ficheros.
- Operadores lógicos.

1.- Comparación de cadenas:

test [*-n -z*] (variable|cadena)

-n devuelve ejecución correcta si la longitud de la cadena es mayor de cero.
-z devuelve ejecución correcta si la longitud de la cadena es cero.
Sin parámetros devuelve ejecución correcta si no es la cadena nula.

test variable1|cadena1 (= | !=) variable2|cadena2

Las comparaciones son:

= devuelve ejecución correcta si las dos cadenas son iguales.
!= devuelve ejecución correcta si las dos cadenas son distintas.

Si dos cadenas no tienen el mismo tamaño, el sistema completará las más pequeñas añadiendo espacios después del texto para poder realizar la comparación.

2.- Comparación numérica:

test variable1|entero1 operador variable2|entero2

Los operadores de comparación pueden ser los siguientes:

-eq devuelve ejecución correcta si los dos datos son iguales.
-ne devuelve ejecución correcta si los dos datos son distintos.
-gt devuelve ejecución correcta si el dato1 es mayor que el dato2.
-ge devuelve ejecución correcta si el dato1 es mayor o igual que el dato2.
-lt devuelve ejecución correcta si el dato1 es menor que el dato2.
-le devuelve ejecución correcta si el dato1 es menor o igual que el dato2.

3.- Operadores de ficheros:

test [*-f -r -w -x -s -d*] variable|cadena

-f devuelve ejecución correcta si el fichero existe y es ordinario.
-r devuelve ejecución correcta si el fichero existe y tenemos derecho de lectura sobre él.
-w devuelve ejecución correcta si el fichero existe y tenemos derecho de escritura sobre él.
-x devuelve ejecución correcta si el fichero existe y tenemos derecho de ejecución sobre él.
-s devuelve ejecución correcta si el fichero existe y no está vacío.
-d devuelve ejecución correcta si es directorio y existe.

4.- Operadores lógicos:

Se pueden utilizar los operadores lógicos (*NOT* (*NO* o *!*), *AND* (*Y* o *-a*) y *OR* (*O* o *-o*) para comparar expresiones usando las reglas lógicas.

Ejemplos del comando *test* están incluidos en los ejemplos de sentencias condicionales e iterativas.

En caso de estar trabajando con la *C-shell*, las comparaciones se realizan de forma parecida al lenguaje de programación C (por ejemplo, con los operadores ==, !=, >, >=, < o <=), en lugar de las vistas para la *Bourne-shell*.

12.2.1.2.- Estructura *if*.

La **sentencia *if*** evalúa una expresión lógica para tomar una decisión.

La sintaxis es la siguiente:

```
if condición
then
        comandos si la condición se cumple
else
        comandos si la condición no se cumple
fi
```

Ejemplo: Acepta un número y decide si es menor que 20.

```
clear
echo Introduce un número
read n
if [ $n –lt 20 ]
then
        echo $n es menor que 20
```

```
else
        echo $n no es menor que 20
fi
```

Las condiciones *if* se pueden anidar usando *elif* (*else if*).

```
if condición
then
        comandos si la condición se cumple
elif condición 2
then
        comandos si la condición 2 se cumple
elif condición 3
then
        comandos si la condición 3 se cumple
.....
else
        comandos si la condición no se cumple
fi
```

12.2.1.3.- Estructura *case*.

La **sentencia *case*** se utiliza para ejecutar diferentes comandos dependiendo de un valor discreto o un rango de valores que coincida con la variable especificada.

La sintaxis es la siguiente:

```
case palabra o parámetro in
        patrón1)        comandos;;
        patrón2)        comandos;;
        .............   .................
        *)              comandos;;
esac
```

En los patrones se pueden especificar varios valores discretos separados con "|" (valor1 | valor2 | valor3 ...).

La última condición debe ser un asterisco ("*") y se ejecutará si no se cumple ninguna de las otras.

Para cada una de las condiciones especificadas se ejecutan todos los comandos (o sentencias) asociados hasta encontrar el punto y coma doble (";;").

Ejemplo: Construcción de un menú de opciones.

```
clear
echo
echo Introduzca el nombre del fichero
read fich
clear
echo "

        1.- MOSTRAR FICHERO
        2.- BORRAR FICHERO
        3.- RENOMBRAR FICHERO

        Teclee opción:"
read opcion
case $opcion in
        1)      clear
                cat $fich;;
        2)      rm $fich;;
        3)      echo Introduzca nuevo nombre
                read nuevo
                mv $fich $nuevo;;
        *)      echo Opción errónea;;
esac
```

12.2.2.- Estructuras iterativas.

Las sentencias iterativas o de bucles se utilizan en los *shell scripts* para repetir una serie de comandos o sentencias un determinado número de veces o hasta que se cumpla cierta condición.

Para forzar la salida de un bucle se usa el **comando *break***.

12.2.2.1.- Estructura *for*.

La **sentencia *for*** se utiliza para ejecutar un conjunto de comandos un número especificado de veces.

Su sintaxis es la siguiente:

```
for variable in lista de valores
do
        comandos
done
```

La *shell* examina la lista de valores, almacena el primer valor en la variable del bucle y ejecuta los comandos que hay entre las palabras *do* y *done* (cuerpo del bucle). A

continuación, se asigna el segundo valor a la variable de bucle y se ejecuta otra vez el cuerpo del bucle. Esto se repite para todos los valores de la lista.

Ejemplo: Muestra la lista especificada.

```
for i in lunes martes miércoles jueves viernes
do
        echo $i
done
```

12.2.2.2.- Estructura *while*.

La **sentencia *while*** se utiliza para ejecutar un conjunto de comandos mientras sea verdadera la condición del bucle.

La sintaxis es la siguiente:

```
while condición
do
        comandos
done
```

Los comandos del cuerpo del bucle (entre las palabras *do* y *done*) se ejecutan repetidamente mientras la condición del bucle es cierta (cero). La condición del bucle debe eventualmente retornar a falsa (distinto de cero) para evitar un bucle infinito que se ejecute indefinidamente.

Ejemplo 1: Suma un entero un determinado número de veces.

```
echo Introduce un entero
read entero
echo Introduce el número de veces
read n
i=1
total=0
while test $i –le $n
do
        total=`expr $total + $entero`
        i=`expr $i + 1`
done
echo total = $total
```

Ejemplo 2: Construcción de un menú iterativo de opciones.

```
clear
while true
```

```
do
        echo
        echo Introduzca el nombre del fichero
        read fich
        clear
        echo "

                1.- MOSTRAR FICHERO
                2.- BORRAR FICHERO
                3.- RENOMBRAR FICHERO
                4.- SALIR

                Teclee opción:"
        read opcion
        case $opcion in
                1)      clear
                        cat $fich;;
                2)      rm $fich;;
                3)      echo Introduzca nuevo nombre
                        read nuevo
                        mv $fich $nuevo;;
                4)      echo Adios
                        break;;
                *)      echo Opción errónea;;
        esac
done
```

12.2.2.3.- Estructura *until*.

La **sentencia *until*** se utiliza para ejecutar un conjunto de comandos mientras sea falsa la condición del bucle.

La sintaxis es la siguiente:

```
until condición
do
        comandos
done
```

Los comandos del cuerpo del bucle (entre las palabras *do* y *done*) se ejecutan repetidamente mientras la condición del bucle es falsa (no cero). La condición del bucle debe eventualmente retornar a verdadera (cero) para evitar un bucle infinito que se ejecute indefinidamente.

Esta sentencia es útil cuando se desea escribir *shell scripts* cuya ejecución depende de que ocurran otros sucesos.

El cuerpo del bucle podría no ejecutarse nunca si la condición es verdadera (cero) la primera vez que se ejecuta.

Ejemplo: Lista los argumentos de un comando.

```
until [ $# -eq 0 ]
do
        echo $1
        shift
done
# el comando shift desplaza a la izquierda los argumentos
```

Si el fichero se llama lista, al ejecutarlo con argumentos:

```
$ lista p1 p2 p3
p1
p2
p3
$
```

12.3.- Definición de funciones.

Como sucede con otros lenguajes de programación, los programas de *shell* permiten utilizar funciones. Una función es una parte de un programa de *shell* que realiza un proceso concreto y que puede ser realizada más de una vez en ese programa. Al escribir una función se pueden escribir programas de *shell* sin necesidad de duplicar el código.

La sintaxis utilizada es la siguiente:

```
nombre_funcion( ) {
        sentencias
}
```

Una llamada a una función se hace de la siguiente manera:

```
nombre_funcion [parámetro1 parámetro2 parametro3 ...]
```

Los parámetros son opcionales, y una función puede analizar los parámetros como si se tratasen de los parámetros posicionales pasados a un programa de *shell*.

Ejemplo: Este programa mostrará Día: Martes.

```
Muestrames( ) {
        case $1 in
                1)      echo “Día: Lunes”;;
                2)      echo “Día: Martes”;;
                3)      echo “Día: Miércoles”;;
                4)      echo “Día: Jueves”;;
                5)      echo “Día: Viernes”;;
                6)      echo “Día: Sábado”;;
                7)      echo “Día: Domingo”;;
                *)      echo “Parámetro no válido”;;
        esac
}
Muestrames 2
```

12.4.- Salida de un programa.

El **comando *exit*** hace que un *shell script* termine de forma inmediata. Además, como opción, se puede utilizar un número después del comando *exit* que proporciona un estado de salida (o código de retorno) al proceso padre, en caso de que el programa de *shell* actual haya sido llamado por otro programa. El proceso padre puede comprobar el código de salida y tomar una decisión de acuerdo con él. La variable incorporada **$?** contiene la condición de salida devuelta por el último comando.

Ejemplo: Se abandona la *shell* hija con estado 5.

```
$ sh
$ exit 5
$ echo $?
5
```

13.- Conceptos avanzados del entorno UNIX.

13.1.- Procesos en UNIX.

Un **proceso** o **tarea** es una instancia de un programa en ejecución. Mientras que un programa es sólo un conjunto de instrucciones, un proceso es algo dinámico, al utilizar los recursos de un sistema en ejecución. Por otra parte, un único programa puede iniciar varios procesos.

El concepto de proceso es algo fundamental en UNIX, así como en cualquier sistema *multiusuario* (varias personas pueden usar simultáneamente el computador) y *multitarea* (el sistema puede trabajar de forma concurrente en varias tareas).

13.1.1.- Identificador de proceso (PID).

Gestionar varios usuarios y llevar a cabo la multitarea son trabajos del sistema operativo, siendo necesaria una cuidadosa planificación de la asignación de los recursos. Para ello, el sistema asigna a cada proceso un número identificativo. Este identificador se conoce como ***PID*** o ***identificador de proceso***. Es un número único para cada proceso activo. Cuando un proceso finaliza su ejecución, el PID puede asignarse a un nuevo proceso.

UNIX mantiene en memoria una tabla de procesos donde cada entrada corresponde a un proceso. Allí se almacenan datos referentes a cada proceso que detallan su estado. El número de entradas de la tabla de procesos es limitado dependiendo de cada sistema. Cuando un proceso finaliza, la entrada correspondiente en la tabla queda libre.

La variable incorporada **$$** contiene el PID del proceso que se está ejecutando.

13.1.2.- Arranque de múltiples procesos.

Cada vez que se conecta un usuario al sistema se ejecuta la *shell* definida en el fichero de inicialización, conocida como *shell de conexión*. Esta *shell* es un proceso, y cada comando o programa que se ejecute iniciará como mínimo un nuevo proceso con un PID único, lo que no impide que la *shell de conexión* continúe ejecutándose.

Se dice que los procesos “viven” y engendran nuevos procesos, es decir, tienen procesos hijos. Cuando un proceso llama a otro o ejecuta ciertos comandos, crea un proceso hijo. Mientras el proceso hijo está activo, el proceso padre suspende su ejecución (está “durmiendo”), y la reanuda (“despierta”) cuando recibe una señal que indica que el proceso hijo ha finalizado. Así, al introducir un comando desde la *shell* como intérprete de comandos, se crea un proceso hijo del proceso correspondiente a la *shell* en ejecución.

Otra forma de iniciar múltiples procesos es utilizando **pipes** o **tuberías** en una línea de comandos. Por ejemplo, para imprimir una lista de los 10 archivos modificados más recientemente en el directorio actual, se escribe:

$ *ls -lt | head | lp*

Los comandos separados por la barra vertical (“|”) se inician al mismo tiempo. Ninguno de ellos está subordinado a otro, sino que todos están subordinados al proceso que

se estaba ejecutando cuando fueron introducidos (por ejemplo, subordinados a la *shell* actual). En este sentido, los comandos situados a ambos lados del símbolo de pipe o tubería pueden considerarse como procesos hermanos.

13.1.3.- Tipos de procesos.

Hay dos tipos de procesos: uno lo forman los **procesos de usuario** y el otro se refiere a los **procesos del sistema**. Estos últimos son los programas que ejecuta el *kernel* o núcleo del sistema operativo para llevar a cabo el mantenimiento del sistema en sí mismo.

Hay comandos cuya ejecución no genera un proceso y son controlados directamente por el *kernel*. Ejemplos de ello son los comandos de programación de la *shell* (*for*, *while*, *echo*, etc.) o la asignación de valores a variables.

UNIX es un sistema multitarea, lo cual significa que puede ejecutar varias tareas de forma concurrente. Atender a cada usuario es un ejemplo de tal afirmación, pero además, cada usuario puede ejecutar varias tareas propias a la vez.

Hay dos maneras de ejecutar una tarea o proceso:

1) En modo inmediato, conocido como ***proceso en primer plano*** o ***proceso en foreground***.
2) En modo desatendido, conocido como ***proceso en segundo plano***, ***proceso de fondo*** o ***proceso en background***.

13.1.3.1.- Procesos en primer plano.

Generalmente son procesos interactivos que requieren la presencia del usuario en el terminal. Como el terminal se puede utilizar para la entrada o salida de datos de un único proceso, no puede haber más de un programa ejecutándose de este modo.

En este caso, un proceso superior espera a que uno o más de sus subordinados termine antes de continuar.

Para que el proceso padre no espere a que terminen los procesos hijos para continuar, los puede convertir en procesos en segundo plano.

13.1.3.2.- Procesos en segundo plano (*background*).

Se puede ejecutar un proceso como una tarea en segundo plano (***background***) introduciendo el comando que inicia el proceso y añadiendo a continuación el signo **&**. El

proceso no queda asociado al terminal, realizándose las entradas y salidas mediante redirección o tuberías.

Ejemplo:
```
$ lp report.txt &
3146
$
```

El prompt del sistema reaparece sin esperar a que finalice el proceso, pudiendo continuar con otros trabajos. Pero el proceso ejecutado en segundo plano es un proceso subordinado a la *shell* actual.

Al ejecutar un proceso en segundo plano el sistema muestra en pantalla el PID correspondiente y se encarga de ponerlo en la tabla de procesos y controlar su ejecución, sin la asistencia del usuario.

La variable incorporada **$!** contiene el PID del último proceso ejecutado en *background*.

13.1.4.- Comandos relativos a procesos.

En apartados anteriores se han visto los comandos principales relativos a procesos:

Comando *who*: Muestra los usuarios conectados al sistema.

Comando *ps*: Muestra el estado de los procesos activos en el sistema.

Comando *sleep*: Suspende la ejecución durante un intervalo de tiempo.

Comando *time*: Determina el tiempo que tarda un programa en ejecutarse.

Comando *kill*: Finaliza un proceso en ejecución.

Otros comandos referentes a procesos son:

Esperar la finalización de tareas: ***wait***.

Continuar ejecución después de desconexión: ***nohup***.

Planificación de procesos: ***at***, ***batch***, ***cron*** y ***crontab***.

Prioridad de procesos: ***nice*** y ***renice***.

13.1.5.- Variables exportadas.

En UNIX las variables son locales a cada proceso.

Ejemplo:
```
$ a=10
$ echo $a
10
```

```
$ cat > otro_proceso
echo $a
a=5
echo $a
<Ctrl-d>
$ otro_proceso

5

$
```

Sin embargo, muchas veces se hace necesario que las variables de un proceso sean conocidas por los procesos hijos. El **comando *export*** sirve para exportar las variables de un proceso a todos sus procesos hijos y su descendencia.

El proceso hijo no puede modificar la variable exportada y tampoco es posible exportar una variable al proceso padre.

Ejemplo:

```
$ cat > otro_proceso
echo $a
b=20
proceso_hijo
<Ctrl-d>
$ cat > proceso_hijo
echo $a
echo $b
<Ctrl-d>
.......

$ a=10
$ export a
$ otro_proceso
10
10

$
$ echo $b


$
```

En el ejemplo, la variable *a* ha sido exportada desde el proceso de *shell* al proceso hijo (otro_proceso) y por tanto a proceso_hijo. Sin embargo la variable *b*, definida dentro de otro_proceso, es local a dicho proceso, por lo que no existe para el proceso hijo (proceso_hijo). Para el proceso padre (*shell*) tampoco se reconoce la variable definida en el

proceso hijo (otro_proceso). Poniendo *export* b dentro de otro_proceso, entonces la variable *b* se exportaría a proceso_hijo y sí sería identificada por el mismo.

13.2.- Enlaces en UNIX.

Un enlace es un tipo especial de ficheros en UNIX que no contiene datos, sino que es una referencia a otro fichero. Esto significa que se pueden crear nombres adicionales para un archivo existente y referirse al mismo archivo con nombres diferentes. Para tal propósito se utiliza el **comando *ln***.

Comando *ln*: Crea enlaces simbólicos entre ficheros o directorios, permitiendo operar con un fichero o un directorio mediante distintos nombres y distintos privilegios.

Ejemplo: $ *ln* /home/usuario/fichero_origen nombre_nuevo