El

Sistema Operativo

UNIX

(Rev. 2, 10/2001)

Física

Departamento de Ingeniería de Sistemas y Automática

1. Bibliografía

1. Waite, Prata, Martín. “Introducción al UNIX”. Anaya, 1986
2. Morgan R., MacGilton H., “Introducción al UNIX sistema V”, MacGraw-Hill, México, 1990
3. Lucas, Martín. “Sistema Operativo UNIX”. Paraninfo, 1987
4. Miller, Boyle. “UNIX for Users”. Blackwell, 1984
5. Silvester. “The UNIX System Guidebook”. Springer, 1984
6. Llanos D.R. “Curso de C bajo UNIX”, Universidad de Valladolid, 1998

2. Sistemas operativos

Antes de estudiar un sistema operativo en particular, debemos contestar a la pregunta de ¿qué es un sistema operativo? y ¿para qué nos sirve? Podríamos dar varias definiciones posibles, sin embargo, vamos a ver el sistema operativo en sus dos aspectos más habituales: como gestor del ordenador y como mediador entre el usuario y el ordenador.

En primer lugar, debe quedar claro que un sistema operativo es un conjunto de programas, es decir, nos encontramos ante varios elementos software programados para un máquina específica. Estos programas se encargan de varias tareas.

En su papel de **gestor** del ordenador, el sistema operativo debe gestionar los recursos del sistema informático (procesadores, memoria, discos, etc), entre los diferentes procesos (programas en ejecución) que compiten por ellos. Esto lo podemos ver más claramente en un sistema multitarea donde las tareas de gestión son más complicadas: supongamos que tenemos un par de programas ejecutándose; en un navegador de Internet estamos cargando una página y como tarda mucho, aprovechamos para ejecutar un programa de cálculo científico. Dado que existen dos programas en ejecución simultánea y el microprocesador de un ordenador habitualmente es único y por él tienen que pasar todas las instrucciones, queda claro que “alguien” debe decidir cuál de los dos programas tiene acceso, en un momento dado, al microprocesador. Lo mismo podemos decir del resto de los recursos como memoria, discos, etc. Si pensamos en un sistema operativo multiusuario y suponemos que dos usuarios quieren imprimir simultáneamente en la misma impresora, el sistema operativo deberá decidir que uno de los usuarios puede imprimir en ese momento y que el trabajo del otro usuario quede almacenado en una cola de impresión para cuando la impresora quede libre.

En su papel de **mediador**, el sistema operativo ofrece al usuario que utiliza el ordenador una especie de “máquina virtual” o “máquina extendida” más fácil de utilizar que si tuviera que acceder directamente al hardware. Pongamos como ejemplo un comando como *copy* que permite copiar un archivo de una ubicación a otra dentro de un disco o entre diferentes discos. Para realizar estas operaciones de forma correcta, deberíamos ser capaces de encontrar el archivo dentro del disco, una vez encontrado posicionar la cabeza lectora del disco en el punto correcto y comenzar a transmitir la información a la memoria y de la memoria a la ubicación destino. Pues bien, todas estas tareas de bajo nivel y que requieren un conocimiento del funcionamiento hardware del ordenador están ya programadas y nosotros, como usuarios del ordenador, no tenemos que preocuparnos, ya que el sistema operativo se encarga de ello; en el ejemplo que poníamos, el usuario

sólo debe usar el comando *copy* dando como parámetros la ubicación origen y la ubicación destino. Este procedimiento por el cual el usuario "ve" una máquina virtual más fácil de utilizar que el hardware subyacente se conoce como "Principio de embellecimiento".

Como resumen, podemos definir el sistema operativo de la siguiente manera:

> El sistema operativo es un conjunto de programas cuyas misiones son:
>
> a) Gestionar los recursos del sistema informático (procesadores, memoria, discos, etc), entre los diferentes procesos que compiten por ellos, y
>
> b) Ofrecer al usuario una especie de "máquina virtual" o "máquina extendida", más fácil de usar que el hardware subyacente ("Principio de embellecimiento").

2.1 Proceso

Un proceso en un programa en ejecución ("proceso secuencial"). Consta de:

- Texto, — Invariante
- Datos, — Estado
- Pila, — Estado
- Registros, etc. — Estado

3. Clasificación de los sistemas operativos

3.1 Tareas

En este punto clasificamos los sistemas operativos atendiendo al número de tareas que puede atender simultáneamente. Tenemos dos tipos:

- *Monotarea*: el sistema operativo solamente puede atender una tarea en un momento dado. Un ejemplo de S.O. de este tipo es MS-DOS.
- *Multitarea*: el sistema operativo puede atender varias tareas a la vez. A su vez estas tareas pueden provenir de un único usuario o de varios usuarios, lo cual dependerá de las capacidades del sistema operativo. Dentro de los sistemas operativos multitarea, existen los S.O. monousuario (por ejemplo, Windows NT) y multiusuario (por ejemplo, VMS y UNIX), donde el S.O. puede atender a un único usuario o a varios en la misma máquina, respectivamente.

3.2 Planificación

La planificación de un S.O. define cómo se reparte el tiempo de CPU entre los diversos procesos. Por supuesto, esto sólo tendrá sentido en S.O. multitarea donde puede ocurrir que en un momento dado varios procesos quieran utilizar el microprocesador y como este es único, debe especificarse la política de acceso. Existen varias formas de realizar esta planificación:

- Tiempo compartido (Round-Robbin): Se asigna el mismo tiempo para cada uno de los procesos
- Prioridades: Cada proceso tiene asignada una prioridad; hasta que no termina un proceso, no se cede la CPU al siguiente.
  - Estáticas: Las prioridades son fijas, no se modifican
  - Dinámicas: Existen ciertos criterios implementados en el S.O.
- Mixtas (VMS, UNIX): Existe una planificación concreta a base de asignar tiempos en función de prioridades. Si dos procesos tienen asignada una prioridad, se comparte el tiempo entre los dos.

Lo habitual es tener planificación mixta, lo cual ocurre en los sistemas operativos VMS y UNIX. En estos casos, a aquellos procesos poco activos se les suele dar una prioridad máxima (por ejemplo, un editor de textos) y aquellos que exigen mucho tiempo de computación, una baja prioridad (por ejemplo, una inversión de matrices). Esto tiene sentido porque un proceso poco activo como un editor de textos consume muy pocos recursos de CPU, es decir, dentro de un intervalo de tiempo dado, el tiempo de CPU que va a usar es muy poco ya que desde que el usuario pulsa una tecla hasta que pulsa la siguiente, el ordenador ha tenido tiempo de realizar otras muchísimas tareas. Sin embargo, sería incómodo para el usuario que desde que pulsa una tecla hasta que aparece el resultado en la pantalla, pasara mucho tiempo; esa es la razón para darle máxima prioridad. En el otro extremo están los procesos que requieren mucho tiempo de computación, es decir, mucho tiempo de acceso a la CPU, como una inversión de matrices, una integral numérica, etc. Con el objeto de no saturar la máquina con estos cálculos, y debido a que habitualmente no existe una restricción temporal para acabar los cálculos, se le asigna una prioridad baja.

3.3 Gestión de memoria

- Memoria real
- Memoria virtual (puede ser mayor que la real)

El S.O. que sólo utiliza memoria real quiere decir que el único lugar donde le es posible cargar el código de un programa es en la memoria física real, es decir, en la RAM. En implementaciones de S.O. que utilizan memoria virtual, es posible hacer uso de espacio de almacenamiento en disco como si fuera memoria adicional de la que dispone el ordenador, es decir, la memoria efectiva puede ser mayor que la real.

Físicas

3.4 UNIX

Es un sistema operativo:

◊ Multitarea,
◊ Multiusuario,
◊ Planificación mixta,
◊ Casi todas las implementaciones son de memoria virtual.

4. Historia

El S.O. Unix se gestó a finales de los años sesenta en los laboratorios Bell AT&T sobre un ordenador PDP-7. La razón de su origen se debe a que Ken Thompson, insatisfecho con el sistema operativo que utilizaba en su trabajo, decidió escribir su propio S.O. Inicialmente fue escrito en lenguaje ensamblador, pero más adelante se reescribió parte del sistema operativo en un nuevo lenguaje de programación denominado B (precursor del actual lenguaje C). Al mismo tiempo, otro programador de la misma compañía, Dennis Ritchie, padre del lenguaje C, entró en contacto con Unix y, junto con Ken Thompson, tradujo el Unix a este lenguaje.

Dada la imposibilidad de comercialización por parte de AT&T, se decidió distribuirlo con fines altruistas a Universidades, a cambio de un pago simbólico. Esta decisión tuvo dos consecuencias:

- Rápida extensión y uso en el mundo científico.
- Diversidad de versiones ya que no había quién dirigiera su desarrollo y evolución.

Para paliar este último inconveniente, en 1984 AT&T lanza el estándar Unix System V.

5. Estructura general del sistema operativo UNIX

Se puede dividir en varios componentes perfectamente diferenciados:

∗ Núcleo o Kernel: Comprende un 5-10% del código total.
∗ Caparazón o Shell: Actúa como intérprete de comandos.
∗ Programas de utilidad.

5.1 Kernel

Es el núcleo del S.O. UNIX. Tiene diversas tareas asignadas:

- Planificar, coordinar y gestionar la ejecución de los procesos. Para ello, hace uso de las prioridades asignadas a cada proceso y utiliza algoritmos específicos para repartir el tiempo entre los diversos procesos que compiten por él.
- Dar servicios del sistema, como entrada/salida y gestión de ficheros.
- Manejar las operaciones dependientes de hardware, es decir, realiza las funciones de más bajo nivel de manera que se oculten al usuario.

Un kernel típico puede constar de unas 20.000 líneas de código de las cuales un 70-80% está escrito en C y el resto depende de máquina. Para un PC ocupa unos 500 Kb y para máquinas grandes puede llegar a 2 Mb.

5.2 Shell

Desde el punto de vista del usuario, actúa como un intérprete de comandos. Es un programa que siempre está en ejecución.

El Shell lee las órdenes suministradas, las decodifica y lo comunica al núcleo para realizar la acción especificada.

Prácticamente, todas las órdenes son programas ejecutables que el shell busca en el sistema de ficheros, siguiendo el orden especificado en la variable global PATH.

Existen varios tipos de shells, función, principalmente, de la versión de UNIX utilizada:
- Bourne shell (System V, Xenix)
- C shell (Berkeley)
- Korn shell (Ambos)

El shell puede constar, en total, de unas 200.000 líneas de código en C.

Los diversos componentes del sistema operativo pueden verse gráficamente de la manera mostrada en la figura 1.

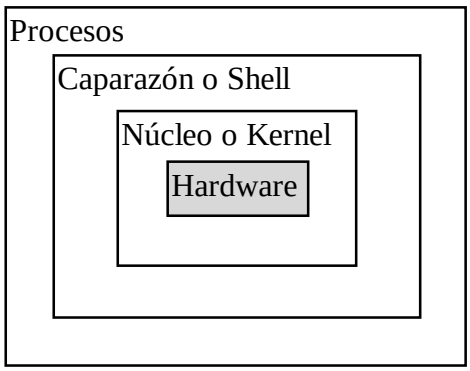
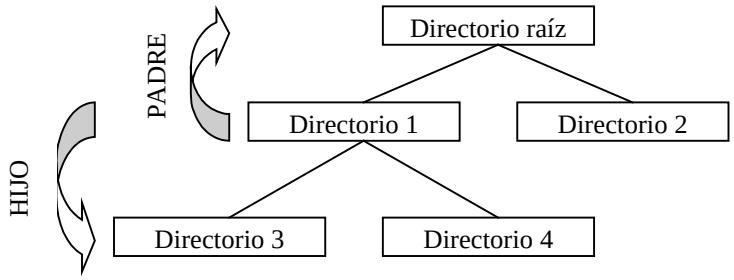


Figura 1. Estructura jerárquica de los componentes de UNIX.

6. Sistema de ficheros

UNIX emplea un sistema de ficheros jerárquico de directorios-ficheros.



No existe, a nivel de usuario, el concepto de *volumen*, ni de *dispositivo físico*. Es decir, el usuario no sabe en qué disco están los ficheros que está utilizando.

Un fichero (o archivo) es un conjunto de información al que se le da un nombre (nombre del fichero). Existen tres tipos de ficheros en UNIX:

- ◊ Ordinarios: Son cadenas de bytes terminadas con <ctrl>D (este código significa fin de fichero). Pueden ser texto, objetos, ejecutables, bibliotecas de módulos, ...
- ◊ Directorios: Contienen nombres de ficheros y su dirección física. Puede pensarse en ellos como carpetas que contienen ficheros y directorios. Un directorio dentro de otro directorio se denomina subdirectorio.
- ◊ Especiales: Asociados a dispositivos entrada/salida. Contienen referencias a los drivers (programas que manejan directamente los dispositivos y que forman parte del núcleo). Pueden ser de tipo "bloque" (apuntan a dispositivos tipo disco) y "carácter" (apuntan a dispositivos como terminales, impresoras, etc). Por convenio, residen en el directorio /dev.

Al elegir los *nombres de los ficheros*, es conveniente limitarse a utilizar sólo los caracteres que correspondan a letras, números, el carácter subrayado _ y el carácter punto . . Los ficheros cuyo nombre comience por punto permanecen ocultos.

En UNIX existe una jerarquía de directorios que para un sistema estándar sería:

/	Raíz
/dev	Fichero especiales de dispositivos
/lib	Bibliotecas del sistema
/bin	Órdenes más empleadas
/etc	Datos y órdenes restringidas al superusuario
/tmp	Ficheros temporales (se borra periódicamente)
/usr	Órdenes, bibliotecas y programas adicionales
/usr/lib	
/usr/bin	
/usr/man	
/usr/...	
/users	Directorios de usuarios (puede aparecer también como /home)

Los ficheros se especifican por:

```
{camino jerárquico}/nombre{.ext}
```

donde las llaves (`{}`) pueden ser o no necesarias. Los ficheros pueden constar de una extensión que es lo que aparece tras el punto (`.`). Un ejemplo sería:

```
/usr/lib/starbase/demos/star.f
```

Con esto estaríamos diciendo que queremos hacer referencia al fichero ordinario denominado `star.f` que se encuentra en el directorio `demos` que a su vez se encuentra en el directorio `starbase` que a su vez se encuentra en el directorio `lib` que a su vez se encuentra en el directorio `usr` que a su vez "cuelga" del directorio raíz. Dada la estructura jerárquica tipo árbol, al final, todo "cuelga" del directorio raíz.

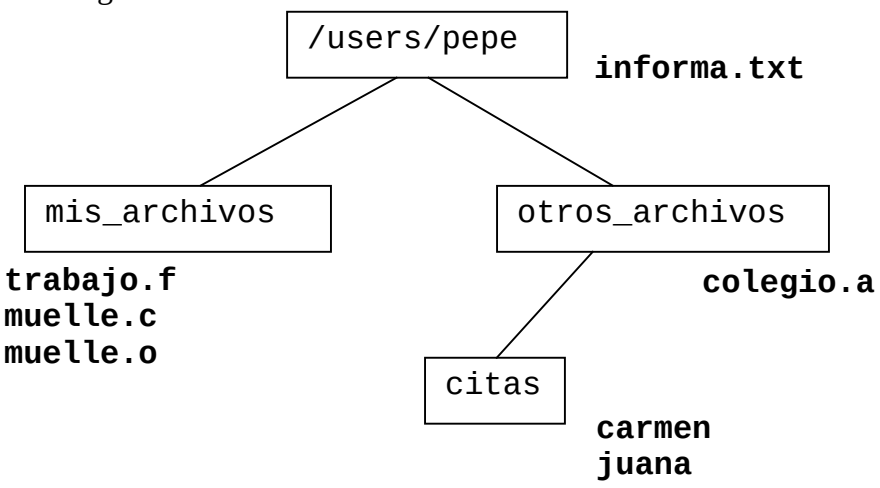
El directorio raíz es el único que no tiene nombre. Cada usuario tiene un directorio *HOME* que es el directorio asignado a ese usuario para que almacene sus ficheros. El camino de este directorio está contenido en la variable HOME.

Los ficheros pueden tener cualquier longitud hasta 256 caracteres. Como se ha dicho anteriormente, la extensión es opcional. En caso de que exista, por convenio significa:

- ◊ .f Programa fuente escrito en lenguaje FORTRAN
- ◊ .p Programa fuente escrito en lenguaje PASCAL
- ◊ .c Programa fuente escrito en lenguaje C
- ◊ .o Fichero objeto
- ◊ .a Biblioteca de módulos
- ◊ .h Fichero de "cabecera"

Un fichero puede ser referenciado por su nombre precedido del nombre del directorio o directorios que le contiene de la manera que pasamos a explicar a continuación. En cualquier momento, los comandos que se teclean al intérprete de comandos están dados en referencia al directorio actual, es decir, el directorio en el que nos "encontramos". Cuando nos referimos a un fichero lo podemos hacer en referencia al directorio actual (vía de **acceso relativa**) o en referencia al directorio raíz (vía de **acceso absoluta**). En la vía de acceso relativa podemos hacer uso de la notación `..` que significa una referencia al directorio padre. Para clarificar todo esto, pongamos un ejemplo:

Supongamos la siguiente estructura de ficheros-directorios:



Físicas

y que nos encontramos situados (más adelante se explica cómo nos *situamos* en un determinado directorio) en el directorio `otros_archivos`. Las referencias a los diferentes ficheros se harían de la siguiente manera:

Nombre del fichero	Acceso absoluto	Acceso relativo
informa.txt	/users/pepe/informa.txt	../informa.txt
trabajo.f	/users/pepe/mis_archivos/trabajo.f	../mis_archivos/trabajo.f
muelle.c	/users/pepe/mis_archivos/muelle.c	../mis_archivos/muelle.c
muelle.o	/users/pepe/mis_archivos/muelle.o	../mis_archivos/muelle.o
colegio.a	/users/pepe/otros_archivos/colegio.a	colegio.a
carmen	/users/pepe/otros_archivos/citas/carmen	citas/carmen
juana	/users/pepe/otros_archivos/citas/juana	citas/juana

Normalmente, será más corto y sencillo utilizar el acceso relativo excepto en determinados casos (por ejemplo, en nuestro caso, `/bin/fgrep`).

UNIX no mantiene versiones de ficheros, por lo que es necesario prestar especial atención a acciones como borrarlos o modificarlos.

Existen 3 ficheros estándar implementados en UNIX y que es importante conocer para realizar determinadas acciones:

◊ Entrada estándar (`stdin`): Teclado (0)
◊ Salida estándar (`stdout`): Pantalla (1)
◊ Errores estándar (`stderr`): Pantalla (2)

La redirección entrada/salida, que se explicará más adelante, permite cambiar estas asignaciones en cualquier momento.

7. Juego de caracteres UNIX

- Letras minúsculas: `a,b,c,...,z`
- Letras mayúsculas: `A,B,C,...,Z`
- Números: `0,1,...,9`
- Caracteres Especiales: `\ / - _ . ; [ ] ( ) < > & |` y los metacaracteres

Es de destacar que en Unix, la misma letra en minúscula y en mayúscula representa distinto carácter por lo que sería diferente un archivo llamado *lista* que un archivo llamado *Lista*. Los comandos, en particular, suelen estar en minúscula.

Los caracteres especiales no deben formar parte del nombre (en determinadas condiciones, se puede hacer pero no es recomendable) ya que tienen un significado especial para el intérprete de comandos. En particular, los metacaracteres hacen referencia a los símbolos asterisco ***** e interrogación **?**. El asterisco representa cualquier cadena de caracteres y la interrogación representa cualquier carácter. Se suelen utilizar para hacer referencia a varios archivos con un sólo comando, por ejemplo, si queremos borrar todos los ficheros fuente en pascal que empiezan por la letra *b*, en vez de borrar uno a uno, diríamos que queremos borrar el archivo *b*.p*. De la misma manera, si queremos borrar aquellos cuya segunda letra sea una *b*, usaríamos el nombre *?b*.p*.

7.1 Metacaracteres

`*` Es equivalente a cualquier cadena de caracteres.
`?` Es equivalente a cualquier carácter.

7.2 Caracteres especiales

`;` Equivale a un retorno de carro en la línea de órdenes.
`[]` Engloba grupos de caracteres, separados por coma `,` o por guión `-`. Por ejemplo, `fich[1,2].f` se refiere tanto a `fich1.f` como a `fich2.f`.
`<>` Sirven para redireccionar entrada / salida, respectivamente.
`&` Envía un proceso en "background" (se explica más adelante).
`|` Construcción de tuberías o "pipes" (se explica más adelante).
`\` Suprime el significado especial del carácter que le sigue. Por ejemplo, `\*` dejaría de ser equivalente a cualquier cadena de caracteres.

8. Clasificación de usuarios en UNIX

UNIX es un sistema operativo multitarea y multiusuario, por lo que se deben establecer ciertos mecanismos de tal manera que, simultáneamente, se protejan los datos de un usuario frente a otros y éstos puedan ser compartidos en caso necesario. UNIX posee un mecanismo de *permisos* asociados a cada fichero. Este mecanismo permite que los ficheros y directorios pertenezcan a un usuario en particular. UNIX también permite que los ficheros sean compartidos entre usuarios y grupos de usuarios. El comportamiento por defecto en la mayoría de los sistemas es que todos los usuarios pueden leer los ficheros de otro usuario, pero no pueden modificarlos o borrarlos.

Los grupos de usuarios se definen normalmente en función del tipo de usuario. Por ejemplo, en una Universidad, los usuarios pueden clasificarse como estudiantes, profesores, invitados, etc.

Cada usuario (perteneciente a un grupo de usuarios) tiene asociado un nombre, una palabra clave o password, un directorio y un proceso de arranque:

- Nombre: Identificación del usuario cuando entra en la máquina (login).
- Clave: Palabra oculta que sólo conoce el usuario.
- UID, GID: Números de identificación de usuario y grupo, respectivamente.
- Directorio: Directorio inicial donde se situará el usuario al entrar en el sistema.
- Proceso: Primer proceso que se arranca una vez dentro del sistema.

Existen diferentes categorías de usuarios en función de sus privilegios (lo que puede y no puede hacer):

- Superusuario o root: Es el administrador del sistema. Tiene todos los privilegios.
- Usuarios normales: El resto de usuarios que pertenecen a distintos grupos, los cuales pueden tener una serie de propiedades comunes.
- Usuarios especiales: Asignados a tareas específicas por el sistema, generalmente de información o manejo de aplicaciones ya instaladas de uso común a

usuarios externos o internos. Por ejemplo: mail (se encarga de recoger el correo y repartirlo a los diversos usuarios), lp (se encarga de aceptar trabajos de impresión y mandarlos a la impresora), bin, admin, ...

Desde el punto de vista de un usuario, el carácter `u` significa el propio usuario, `g` significa el conjunto de usuarios que pertenecen a su mismo grupo, `o` significa el resto de usuarios. Estos caracteres serán reconocidos por ciertos comandos u órdenes.

Los permisos que llevan asociados todos los ficheros y directorios se clasifican en *lectura* (read, r), *escritura* (write, w) y *ejecución* (execute, x). Estos permisos se pueden asignar al propio *usuario* (u), al *grupo* (g) y al *resto* (o).

El permiso de lectura permite a un usuario leer el contenidos del fichero, o, en el caso de directorios, obtener un listado de su contenido.

El permiso de escritura permite a un usuario escribir y modificar el fichero. Para directorios, permite al usuario crear nuevos ficheros dentro del directorio o borrar los que contiene.

El permiso de ejecución permite a un usuario ejecutar un fichero (debería ser un programa o script —fichero que contiene órdenes para el sistema). En el caso de directorios, permiso de ejecución significa que el usuario puede introducirse en dicho directorio.

9. Inicio de la sesión de trabajo

Como se ha dicho anteriormente, cada usuario tiene asociado un *nombre de usuario* o *cuenta* y una palabra clave o *password*. Para acceder a un sistema y poder trabajar en él, se debe realizar un proceso denominado “login in”. El sistema que esté listo para aceptar a un usuario presentará un mensaje como:

```
login:
```

a lo que se debe responder con el nombre de usuario asociado. Una vez introducido se pulsa la tecla ENTER o INTRO y el sistema presentará otro mensaje pidiendo la palabra clave:

```
password:
```

a lo que responderemos con nuestro password (lo que tecleemos no se presentará en pantalla para evitar que alguien lo pudiera ver), finalizando al pulsar la tecla ENTER o INTRO.

Si cometemos algún error o no tenemos acceso a la *cuenta*, el sistema responderá:

```
login incorrect
```

y nos pedirá de nuevo el *login*.

Si todo ha ido bien, el sistema ejecutará una serie de procesos y finalmente aparecerá un *prompt* o petición de órdenes. El prompt es un carácter o conjunto de caracteres que indica que podemos introducir comandos u órdenes; los más típicos son:

```
$
#
/home/usuario#
```

9.1 Secuencia de recepción de un usuario

init (proceso 0): Es el primer proceso que corre bajo control del sistema en el proceso de carga. Es el origen y administrador de todos los demás procesos del sistema.

getty: Establece el modo de trabajo de cada línea de conexión. Presenta un mensaje de bienvenida al sistema y un mensaje de recepción (login:). Se queda a la espera del tecleo del nombre de usuario seguido por <CR> (Retorno de carro o ENTER o INTRO).

login: Reemplaza a getty una vez leído el <CR>, e inicia la siguiente secuencia de acciones:
- Búsqueda del nombre de usuario.
- Petición de clave de usuario.
- Saca por pantalla el mensaje del día (opcional).
- Comprueba el correo y noticias.
- Tareas de "accounting" (día de entrada, hora, ...).
- Ajusta la identidad del proceso, UID, GID.
- Establece el directorio de trabajo $HOME.

shell: Sustituye a login. Ejecuta el procedimiento de comandos que fija el perfil de usuario, terminal, etc. Presenta el prompt y queda a la espera de la introducción de órdenes por parte del usuario. Existen distintos tipos de shell: /bin/sh, /bin/rsh, /bin/cs, ...

10. Finalización de la sesión de trabajo

Cuando se ha finalizado la sesión de trabajo, se debe teclear `exit` como respuesta al prompt o pulsar <CTRL-D> (mantener la tecla *Control* pulsada a la vez que se pulsa la letra *d*). Es importante finalizar la sesión de trabajo ya que de lo contrario cualquier persona podría utilizar nuestra cuenta y modificar nuestros ficheros a su antojo.

En los sistemas UNIX, **NO SE DEBE APAGAR EL ORDENADOR**; ya que para ello se deben ejecutar una serie de procesos previos.

11. Formato general de los comandos

```
verbo     argumento argumento ...
```

⇒ El verbo, o nombre del comando, es siempre necesario.
⇒ Los argumentos pueden ser necesario o no, dependiendo del comando u orden.
⇒ Los nombres de órdenes en UNIX son siempre en minúsculas y no pueden ser abreviados.

⇒ Los argumentos pueden ser de dos tipos:
- Opciones o adverbios, precedidos de un guión -
- Nombres de ficheros

UNIX consta típicamente de unos 300 comandos, por lo que a continuación se explican los más representativos.

12. Comandos de información general

12.1 date

Fecha y hora del sistema.

12.2 cal

Facilita un calendario. Sintaxis: `cal mes año`
Por ejemplo, `cal 01 1992` facilita un calendario de enero de 1992.

12.3 who

Información de quién está en el sistema.

12.4 whoami

Información de quién está en este terminal. Aparecerá mi nombre de usuario (¿Quién soy yo?).

12.5 pwd

Directorio en el que se está trabajando.

12.6 ps

Información de qué es lo que está haciendo el sistema. Si no le añadimos opciones, solamente dará información sobre mi terminal. Existen multitud de opciones, algunas de las más comunes son:

`ps -e`	Información de todo el sistema
`ps -l`	Formato largo
`ps -f`	Información de los comandos que el sistema está procesando

Según las opciones que le suministremos al comando, aparecerán diversos campos, entre los que cabe destacar:

UID:	Número de identificación del usuario
PID:	Número de identificación del proceso
PPID:	Número de identificación del proceso padre (aquel proceso que creó a éste)
PRI:	Prioridad (cuanto más alto sea este número, menor es la prioridad)
NI:	Número “nice” (es la prioridad efectiva; sólo se puede subir o lo que es lo mismo, bajar la prioridad)
TIME:	Tiempo de CPU en *min:sg*
COMD:	Nombre del comando que se está ejecutando

Un ejemplo típico sería:

```
$ps

PID TT STAT  TIME COMMAND
 24  3 S     0:03 (bash)
161  3 R     0:00 ps
```

12.7 df

Información de bloques libres en los discos montados.

12.8 man

Manual “on-line”.

`man comando`: Da por la salida estándar la misma información que figura en los manuales.

```
man -s nº_de_manual comando
```

Los manuales estándar de UNIX van numerados del 1 al 7:

1. Comandos de uso general.
2. Llamadas al sistema.
3. Rutinas de las bibliotecas del sistema.
4. Formatos de los ficheros.
5. Miscelánea (cabeceras, macros, caracteres, etc.).
6. Juegos.
7. Ficheros especiales de dispositivos.
9. Glosario de términos empleados en los manuales.

13. Comandos de comunicación

13.1 mail

Gestiona el correo electrónico. Tiene dos modos de funcionamiento:

Recepción:

`mail` Da el primer mensaje pendiente. Dispone de un lenguaje de comandos interactivo:

`<CR>`	Da el siguiente mensaje
`-`	Da el mensaje anterior
`s fichero`	Almacena el mensaje en fichero
`p`	Vuelve a dar el mensaje
`d`	Borra el mensaje
`q`	Salida

Envío:

```
mail nombre_de_usuario
...
<CTRL>D
```

13.2 write

Nota de entrega inmediata; el mensaje saldrá inmediatamente en el terminal del usuario al que vaya dirigida.

```
write nombre_de_usuario
...
<CTRL>D
```

13.3 mesg

Inhibe (con opción `-n`) o permite (con opción `-y`) la recepción de notas.

14. Comandos de manipulación de ficheros

14.1 cat

Dirige el contenido de ficheros a la salida estándar (normalmente, la pantalla).

```
cat nombre_de_fichero
```

14.2 cp

Realiza copia de ficheros.

```
cp fichero_fuente fichero_destino
```

14.3 mv

Traslado o cambio de nombre de un fichero.

```
mv fichero_antiguo fichero_nuevo
```

14.4 ln

Establece vínculos entre ficheros.

`ln fichero1 fichero2` Los ficheros 1 y 2 pasan a ser, físicamente, el mismo con dos nombres.

`ln fichero directorio`

14.5 rm

Borra ficheros. Algunas de las opciones más importantes:

- `-i` Pregunta antes de borrar.
- `-r` Borra un directorio de forma recursiva, borrando primero los ficheros que están en él.

14.6 ls

Lista el contenido de un directorio. Opciones más importantes:

- `-a` Lista todos los ficheros (incluso los ocultos).
- `-t` Ordena por fecha de creación.
- `-p` Marca los directorios.
- `-r` Invierte el orden.
- `-s` Indica el tamaño.
- `-l` Formato largo. Indica protecciones, propietario, grupo, tamaño y fecha de creación de cada fichero.
- `-F` Distingue entre ficheros ordinarios, directorios y ejecutables.

A la hora de interpretar una lista de ficheros en formato largo, conviene recordar lo que se dijo sobre las **protecciones**. Recuérdese que existían 3 niveles de protección (lectura

r, escritura w, ejecución x) sobre 3 tipos de usuarios (propietario u, grupo g, resto o). Al ejecutar la orden `ls -l`, se nos facilita la información sobre las protecciones del fichero con el formato:

```
-rwxrwxrwx
```

El primer carácter indica el tipo de fichero:

- `-` Fichero ordinario.
- `d` Directorio.
- `b` Fichero especial tipo bloque.
- `c` Fichero especial tipo carácter.

Los otros nueve caracteres expresan, en grupos de 3, los permisos de acceso a ese fichero. Un guión `-` en un campo implica que no existe ese tipo de permiso. Un ejemplo clarificador sería:

`-rwxr-x---` El propietario puede leer, escribir y ejecutar. El grupo puede leer y ejecutar. El resto de usuarios no tiene ningún acceso.

14.7 cd

Cambio de directorio.

`cd` Cambia al directorio principal del usuario (`$HOME`).
`cd nombre` Cambia al directorio denominado `nombre`.
`cd ..` Cambia al directorio padre del actual.

14.8 mkdir

Crea un directorio.

```
mkdir nombre
```

14.9 rmdir

Borra un directorio (debe estar vacío).

```
rmdir nombre
```

14.10 chown

Cambia el propietario de un fichero (deberá ser nuestro para poderlo hacer).

```
chown propietario fichero
```

14.11 chgrp

Cambia el grupo de un fichero.

14.12 chmod

Cambia los permisos de acceso de un fichero. Actúa sobre el propietario (`u`), el grupo (`g`), o el resto (`o`), añadiendo (`+`) o quitando (`-`) los permisos (`rwx`). Por ejemplo:

`chmod o+r fichero` Se da permiso de lectura al resto de usuarios.
`chmod g+wx fichero` Se da permiso de escritura y ejecución al grupo.
`chmod o-rwx fichero` Se quitan todos los permisos al resto de usuarios.

Existe la posibilidad de dar protecciones en *octal*.

Permisos en binario			Octal	Significado
u	g	o		
xxx	xxx	xxx		

000	000	000	000	No se puede hacer nada
001	000	000	100	El propietario puede ejecutar
111	000	000	700	
111	111	111	777	Totalmente desprotegido

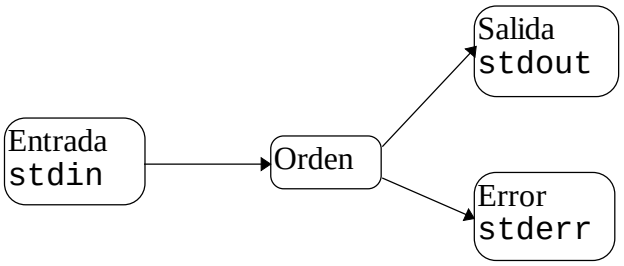
```
chmod 653 fichero
```

14.13 passwd

Para cambiar la palabra clave o *password*. Pide el antiguo y el nuevo dos veces.

15. Redirección de entrada/salida

La ejecución de un comando típico responde al siguiente esquema:



En dicha figura se observa que el comando, si necesita algún dato de entrada, va a esperar que este se le introduzca a través de teclado, que es la entrada por defecto en el sistema operativo. Si el comando ha de devolver alguna información respecto al resultado conseguido por su ejecución, por defecto lo enviará al dispositivo de salida por defecto que es la pantalla. De la misma manera, posibles errores en la ejecución del comando saldrían por el dispositivo de errores por defecto, que también es la pantalla.

Este comportamiento por defecto de los comandos (entrada por teclado, salida por pantalla) puede ser modificado a través de la denominada **redirección**.

15.1 Redirección simple

Puede variarse la entrada/salida estándar, redirigiéndola de/a un fichero, utilizando el formato:

```
comando <entrada >[>]salida 2>[>]errores
```

(los corchetes indican que lo que contienen es opcional)
donde:

- ◊ `entrada` fichero que sustituye a `stdin`
- ◊ `salida` fichero que sustituye a `stdout`
- ◊ `errores` fichero que sustituye a `stderr`

La utilización de >> en lugar de > significa añadir al final del fichero, si ya existe, en vez de al principio.

Los ficheros destino pueden ser alguno de los estándar. En este caso, se especifican con los símbolos `&1` (stdout), `&2` (stderr). Por ejemplo `cat <fichero1 >>fichero2 2>&1` redirecciona los errores a la pantalla.

Los fichero de salida y entrada deben ser diferentes.

Otros ejemplos:
```
ls >listado.lst
write curso3 <mensaje
```

Algunos comandos permiten emplear la *entrada inmediata*, con el símbolo `<<`
```
write curso6 <<fin
...
...
fin
```

15.2 Redirección encadenada

Permite que la salida de un comando se utilice como entrada al siguiente.

Formato:
`cmd1 | cmd2 | ..... | cmdN` "Pipeline" o "tubería"

La salida estándar de un comando de la "tubería" actúa como entrada estándar del siguiente.

No tiene sentido redireccionar la entrada, salvo en el primer comando.

No tiene sentido redireccionar la salida, salvo en el último comando.

Ejemplo:
```
ls -l | more
```

16. Manejo de la cola de impresión

Los comandos básicos para que el usuario se comunique con el gestor de impresión son:
- `lp`: Envía peticiones de impresión
- `lpstat`: Informa sobre el estado de la cola de impresión
- `cancel`: Cancela una o varias entradas de la cola de impresión

16.1 lp

Envía peticiones de impresión de ficheros.

```
lp [opciones] fichero[s]
```

donde las opciones pueden ser, entre otras:
`-n` Especifica un número de copias (`-n4`)

`-d` Especifica impresora de destino (`-dlaser`)

Una vez enviada la petición de impresión, `lp` devuelve un número de identificación de dicha petición, que se puede utilizar bien para cancelarla, bien para obtener información sobre el estado de la cola de impresión.

16.2 cancel

Cancela trabajos pendientes de impresión.

```
cancel [ident] [impresoras]
```

Si se especifica identificación de trabajos, se cancelan dichos trabajos, aunque estén siendo impresos en ese momento.

Si se especifica nombre de impresora, se cancela la impresión en curso en la misma.

Ejemplo:

```
lp fichero
     request-id is lp-205
     ...
cancel lp-205
```

16.3 lpstat

Da información sobre las colas de impresión.

```
lpstat [opciones]
```

donde opciones puede ser alguna de las siguientes:

`-s`	Resumen informativo
`-t`	Información total
`-d`	Nombre de la impresora por defecto
`-clista`	Lista de impresoras del sistema
`-olista`	Colas de las impresoras de la lista

Ejemplos:

```
lpstat -t
lpstat -o"laser"
```

17. Modos de ejecución de comandos

1. Modo inmediato: El comando se ejecuta al introducirlo en la línea de órdenes y teclear <RETURN>.
2. Background: Se genera un nuevo proceso, en un esquema padres-hijos.
3. Batch: Se genera un proceso específico para el comando, pero independiente del proceso del usuario.

17.1 Ejecución en background

Normalmente, son procesos que se ejecutan con prioridad más baja.
Basta terminar la línea de órdenes con el símbolo &.

```
ps -efl ; ls -l | lp &          Afecta al último comando
```

El nuevo proceso generado con & depende del proceso padre que inició el comando. Si éste desaparece, desaparece aquel. Una forma de inmunizar al *hijo* de la muerte del *padre* es utilizar el calificativo `nohup`:

```
nohup comando &
```

Cuando muere el proceso padre, `comando` pasa a depender del proceso inicial denominado `init` (nunca muere). La salida generada se deja en el fichero `nohup.out` en el directorio de trabajo del momento de lanzar el comando.

17.2 Ejecución en batch

Ejecución de órdenes en algún instante posterior.

```
at hora [fecha] [+incremento]
cmd1
cmd2
...
<CTRL>D
```

17.2.1 Especificación de la hora

a) `0815 am  9:30 pm  5 pm`
b) Especiales: `now, next, noon, midnight, zulu`

17.2.2 Espeficificación de la fecha

a) mes-día, año: `Jan 24,1990`
b) Especiales: `today, tomorrow,` días de la semana (en inglés)

17.2.3 Especificación del incremento

Un número, seguido de las palabras clave: `minutes, hours, days, months, years`

17.2.4 Ejemplos

```
echo "/ejec/calculon" | at 2300
```

A las 11pm se ejecuta `calculon`

```
at now +4 hours
     write `logname` <<! Es hora de irse !
     <CTRL>D
```

El sistema nos informa de la identificación del trabajo y la hora a la que se producirá:

```
atjob a.xxxxxxx at <hora><fecha>
```

17.2.5 Información del estado de at

```
at -l[job]      Presenta un resumen del estado de los trabajos pendientes
at -r[job]      Borra job de la cola
```

18. Editores de texto

Un editor de texto es simplemente un programa que se usa para editar ficheros que contienen texto, tales como una letra, un programa en C, un fichero de configuración del sistema.

En UNIX existen unos editores de textos básicos suministrados con el sistema. Los hay de dos tipos:

1. De líneas:
   - `edit` Introductorio, sencillo y poco potente
   - `ed` Versión un poco más flexible
   - `ex` Más versátil, es un editor potente
   - `sed` Editor no interactivo. Está pensado para trabajar con:
     a) Ficheros muy grandes.
     b) Secuencias de edición complicadas (por ejemplo, transformar una lista de DNIs en NIFs).
     c) Funciones de edición globales.
2. De pantalla (caracteres):
   - `vi` Editor típico de UNIX. Características principales:
     - Observación inmediata de la modificación.
     - Entorno edición-ejecución potente.
     - Comandos invisibles.
     - División del campo visual: trabajo y comandos.

Se puede acceder a `vi` desde `ex` y viceversa.
Se puede acceder a `vi` desde `more`.

18.1 Entorno more-vi-ex

18.1.1 more

Comando que permite presentar en pantalla el contenido de un fichero de texto de una forma filtrada.

Se puede utilizar de dos formas diferentes:

1. `more fichero1 fichero2 ...`
2. `comandos | more` Esta forma no admite edición

Presenta una pantalla (25 líneas) del primer fichero, y se queda esperando una orden, con un mensaje en la última fila:

`-more (? %)`

Órdenes:

<espacio>	Siguiente pantalla.
<intro>	Siguiente línea.
/cadena	Búsqueda de cadena en el texto.
n	Siguiente aparición de la cadena.
!cmd	Ejecución del comando cmd en background.
:n	Siguiente fichero.
f	Salta 44 líneas.
v	Edita con vi el fichero en curso. Al terminar, se retorna a more.
q	Fin.
p	Restaura la pantalla.

18.2 Editor vi

```
vi fichero1 [fichero2] ...
```

Existen otros muchos editores para UNIX, pero el único que es seguro encontrar en cualquier sistema UNIX es vi ("visual editor"). vi no es el editor más fácil de usar. Sin embargo, al ser tan común, y, muchas veces, necesario su uso, se explica a continuación su funcionamiento más básico.

vi es un editor pequeño (ocupa pocos recursos) y potente, pero muy difícil de usar. Sin embargo, una vez que uno se acostumbra a él, la edición es muy rápida.

En esta sección se da una introducción a vi. No se discutirán todas sus características, solamente aquellas necesarias para empezar. Se puede utilizar la ayuda de UNIX (man) para aprender alguna más de sus características.

18.2.1 Conceptos

Al usar vi, en cualquier momento se está en uno de tres modos de operación. Estos modos son conocidos como modo de comandos, modo de inserción, y modo de última línea.

Cuando se arranca vi, se está en el *modo de comandos*. Este modo permite usar ciertos comandos para editar ficheros o cambiar a otros modos. Por ejemplo, tecleando `x` en el modo de comandos borra el carácter bajo el cursor. Las teclas de dirección mueven el cursor por el fichero que se está editando. Generalmente, los comandos usados en el modo de comandos son de uno o dos caracteres de largo.

Insertar o editar texto se realiza en el *modo de inserción*. En vi, la mayor parte del tiempo se estará en este modo. Se puede iniciar el modo de inserción usando un comando como `i` (insertar) desde el modo de comandos. En el modo de inserción, se inserta texto en el documento desde la posición actual del cursor. Para finalizar el modo de inserción y volver al modo de comandos, se pulsa <ESC>.

El *modo de última línea* es un modo especial usado para ciertos comandos extendidos. Cuando se teclean estos comandos, aparecen en la última línea de la pantalla (de ahí su nombre). Por ejemplo, cuando se teclea `:` desde el modo de comandos, se salta al modo de última línea y se pueden usar comandos como `wq` (grabar el fichero y finalizar vi), o `q!` (finalizar vi sin grabar los cambios). El modo de última línea generalmente se usa para comandos que son más largos que un carácter. En el modo de última línea, se introduce un comando y se presiona <INTRO> para ejecutarlo.

18.2.2 Iniciando vi

El mejor modo de entender estos conceptos es lanzando vi y editando un fichero. En las pantallas de ejemplo inferiores, se mostrarán sólo unas pocas líneas de texto, como si la pantalla solamente tuviera seis líneas de alto (en vez de 24).

La sintaxis de vi es
`vi nombre_de_fichero`

Comenzar tecleando
`vi test`

Se verá algo como

```
_____________________________________________________________________________
|~_                                                                          |
|~                                                                           |
|~                                                                           |
|~                                                                           |
|~                                                                           |
|"test"_[New_file]___________________________________________________________|
```

La columna de caracteres ~ indica que se está en el final de fichero.

18.2.3 Insertando texto

Ahora se está en el modo de comandos. Para insertar texto en el fichero, pulsar `i` (que nos situará en el modo de inserción), y comenzar a teclear.

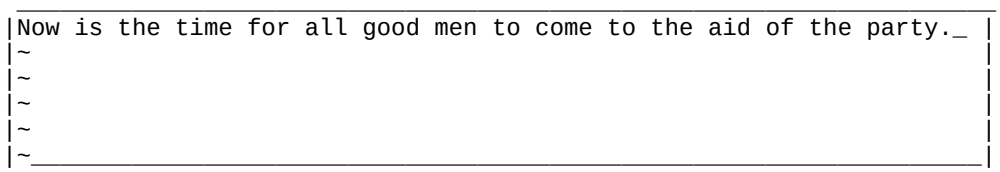
```
_____________________________________________________________________________
|Now is the time for all good men to come to the aid of the party._          |
|~                                                                           |
|~                                                                           |
|~                                                                           |
|~                                                                           |
|~___________________________________________________________________________|
```

En el modo de inserción, se pueden teclear tantas líneas como se quiera (presionando <INTRO> después de cada una, por supuesto), y se pueden corregir errores usando la tecla de retroceso.

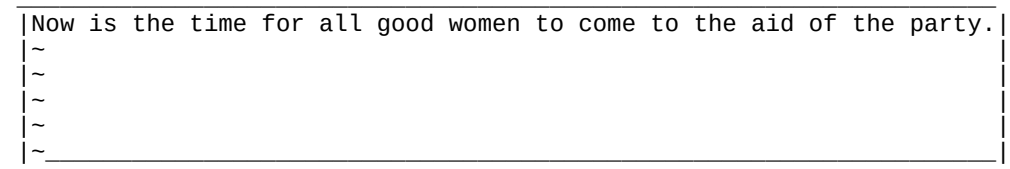
Para finalizar el modo de inserción, y volver al modo de comandos, presionar <ESC>.

En el modo de comandos, se pueden usar la teclas de dirección para moverse por el fichero. En este caso, como sólo tenemos una línea de texto, intentar moverse hacia arriba o abajo probablemente provocará que vi emita un pitido.

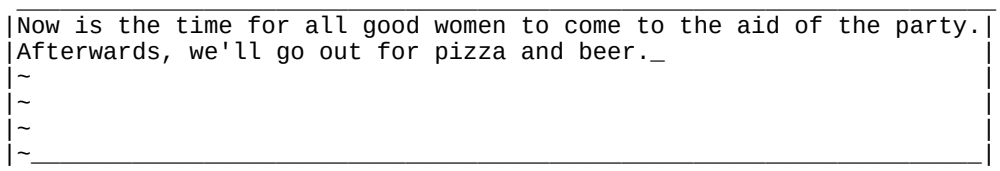
Existen varias maneras de insertar texto, aparte del comando `i`. Por ejemplo, el comando `a` inserta texto empezando después de la posición actual del cursor. Por ejemplo, usa la flecha izquierda para mover el cursor entre las palabras `good` y `men`.



Pulsa `a`, para empezar el modo de inserción, teclea `wo` y pulsa <ESC> para volver al modo de comandos.



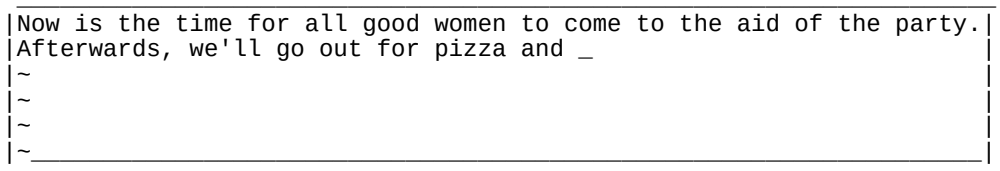
Para comanzar a insertar texto en la línea debajo de la actual, usa el comando `o`. Por ejemplo, pulsa `o` y teclea una o dos líneas:



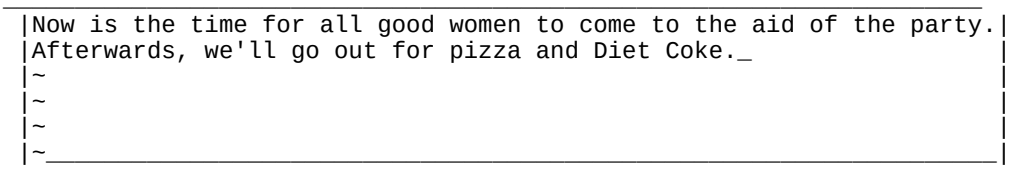
Recuerda que en cualquier momento estás o en modo de comandos (donde comandos como `i,a,o` son válidos), o en modo de inserción (donde insertas textos, seguido de <ESC> para volver al modo de comandos), o en modo de última línea (donde se introducen comandos extendidos, como se discutirá más adelante).

18.2.4 Borrando texto

Desde el modo de comandos, el comando `x` borra el carácter bajo el cursor. Si presionas `x` cinco veces, terminarás con:



Ahora pulsa `a`, inserta texto, seguido de <ESC>:



Puedes borrar líneas enteras usando el comando `dd` (pulsar dos veces seguidas las letra d). Si el cursor está en la segunda línea y tecleas `dd`:

```
|Now is the time for all good women to come to the aid of the party.|
|~                                                                  |
|~                                                                  |
|~                                                                  |
|~                                                                  |
|~__________________________________________________________________|
```

Para borrar la palabra sobre la que está el cursor, usa el comando `dw`. Sitúa el cursor en la palabra `good` y teclea `dw`.

```
|Now is the time for all women to come to the aid of the party.     |
|~                                                                  |
|~                                                                  |
|~                                                                  |
|~                                                                  |
|~__________________________________________________________________|
```

18.2.5 Cambiando texto

Puedes cambiar secciones de texto usando el comando `R`. Sitúa el cursor en la primera letra de `party`, pulsa `R`, y teclea la palabra `hungry`.

```
|Now is the time for all women to come to the aid of the hungry._   |
|~                                                                  |
|~                                                                  |
|~                                                                  |
|~                                                                  |
|~__________________________________________________________________|
```

Usar `R` para editar texto es parecido a usar los comandos `i` y `a`, pero `R` sobreescribe el texto, en lugar de insertarlo.

El comando `r` reemplaza el carácter bajo el cursor. Por ejemplo, mueve el cursor al inicio de la palabra `Now` y teclea `r` seguido por `C`:

```
|Cow is the time for all women to come to the aid of the hungry._   |
|~                                                                  |
|~                                                                  |
|~                                                                  |
|~                                                                  |
|~__________________________________________________________________|
```

El comando ~ cambia la letra bajo el cursor de mayúscula a minúscula y viceversa. Por ejemplo, si posicionas el cursor en la `o` de `Cow`, y pulsas repetidamente ~, conseguirás:

```
|COW IS THE TIME FOR ALL WOMEN TO COME TO THE AID OF THE HUNGRY.    |
|~                                                                  |
|~                                                                  |
|~                                                                  |
|~                                                                  |
|~__________________________________________________________________|
```

18.2.6 Comandos de movimiento

Hasta ahora ya sabes como usar las teclas de dirección para moverte por el documento. Además, puedes usar las comandos `h,j,k,l` para mover el cursor a la izquierda, abajo, arriba y derecha, respectivamente. Esto es útil en casos en los que las teclas de dirección no funcionen por cualquier causa.

El comando `w` mueve el cursor al comienzo de la siguiente palabra; `b` lo mueve al comienzo de la palabra anterior.

El comando `0` (cero) mueve el cursor al principio de la línea actual, y el comando `$` lo mueve al final de la línea.

Al editar ficheros largos, probablemente necesitarás moverte adelante y atrás por el fichero un "pantallazo" a la vez. Pulsando `<CTRL-F>` mueve el cursor una pantalla adelante, y `<CTRL-B>` lo mueve una pantalla hacia atrás.

Para mover el cursor al final de fichero, pulsa `G`. También puedes moverte a una línea en particular; por ejemplo, tecleando `10G` movería el cursor a la línea 10 del fichero. Para moverte al principio del fichero, usa `1G`.

Se pueden agrupar comandos de movimiento con otros comandos, como borrado. Por ejemplo, el comando `d$` borrará todo desde el cursor hasta el final de línea; `dG` borrará todo desde el cursor hasta el final de fichero, etc.

18.2.7 Guardando ficheros y finalizando vi

Para finalizar vi sin realizar cambios al fichero, usa el comando `:q!`. Cuando teclees los dos puntos `:`, el cursor se moverá a la última línea de la pantalla (modo de última línea).

```
 ____________________________________________________________________
|COW IS THE TIME FOR ALL WOMEN TO COME TO THE AID OF THE HUNGRY.     |
|~                                                                   |
|~                                                                   |
|~                                                                   |
|~                                                                   |
|~                                                                   |
|:___________________________________________________________________|
```

En el modo de última línea, ciertos comandos extendidos están disponibles. Uno de ellos es `q!`, que finaliza vi sin salvar. El comando `:wq` salva el fichero y después sale de vi. El comando `ZZ` (desde el modo de comandos, sin usar `:`) es equivalente a `:wq`. Recuerda que debes pulsar <INTRO> después de un comando tecleado en el modo de última línea.

Para guardar un fichero sin finalizar vi, simplemente usa `:w`.

18.2.8 Editando otro fichero

Para editar otro fichero, usa el comando `:e`. Por ejemplo, para finalizar editando “test”, y editar el fichero “foo”, usa el comando

```
 _____________________________________________________________________
|COW IS THE TIME FOR ALL WOMEN TO COME TO THE AID OF THE HUNGRY.      |
|~                                                                    |
|~                                                                    |
|~                                                                    |
|~                                                                    |
|~                                                                    |
|:e foo_______________________________________________________________|
```

Si usas `:e` sin salvar el fichero primero, obtendrás el mensaje de error

```
 _____________________________________________________________________
|No_write_since_last_change_(":edit!"_overrides)_____________________|
```

que significa simplemente que vi no quiere editar otro fichero hasta que no salves el primero. En este punto, puedes usar `:w` para salvar el original, y entonces usar `:e`, o puedes usar el comando

```
 _____________________________________________________________________
|COW IS THE TIME FOR ALL WOMEN TO COME TO THE AID OF THE HUNGRY.      |
|~                                                                    |
|~                                                                    |
|~                                                                    |
|~                                                                    |
|~                                                                    |
|:e! foo______________________________________________________________|
```

La `!` le dice a vi que “lo dices en serio” (edita el nuevo fichero sin guardar los cambios del primero).

18.2.9 Incluyendo otros ficheros

Si usas el comando `:r`, puedes incluir el contenido de otro fichero en el fichero actual. Por ejemplo, el comando

```
:r foo.txt
```

insertaría el contenido del fichero `foo.txt` en el texto en la localización actual del cursor.

18.2.10 Ejecutando comandos del caparazón

También puedes ejecutar comandos del caparazón o shell desde dentro de vi. El comando `:r!` funciona como `:r`, pero en lugar de leer el fichero, inserta la salida de un

comando dado en el buffer en la posición actual del cursor. Por ejemplo, si usas el comando

```
:r!  ls -F
```

llegarás a

```
 _____________________________________________________________________
|COW IS THE TIME FOR ALL WOMEN TO COME TO THE AID OF THE HUNGRY.      |
|letters/                                                             |
|misc/                                                                |
|papers/_                                                             |
|~                                                                    |
|~____________________________________________________________________|
```

También puedes realizar un "shell out", es decir, ejecutar un comando desde dentro de vi, y volver al editor cuando finalice. Por ejemplo, si usas el comando

```
:!  ls -F
```

el comando `ls -F` se ejecutará, y los resultados se verán en pantalla, pero no serán insertados en el fichero que estás editando. Si usas el comando

```
:shell
```

vi iniciará una instancia del shell, permitiendo situar temporalmente a vi "en espera" mientras ejecutas otros comandos. Simplemente sal del caparazón (usando `exit`) para volver a vi.

18.2.11 Consiguiendo ayuda

vi no facilita una ayuda interactiva, pero siempre se puede consular la ayuda suminstrada por `man`. vi es un programa sobre ex, es ex quien maneja muchos de los comandos de última línea en vi. Por lo tanto, además de leer la página de manual para vi, es conveniente ver la de ex también.

19. Programación del shell

El intérprete de comandos es programable.

Definición: **script** : fichero que contiene líneas de comandos. Es un fichero ordinario de texto.

Ejecución: Existen dos formas:
1. `sh fichero`
2. `fichero` (debe tener permiso de ejecución)

Ejemplo: Utilizando el editor vi escribimos:

```
mkdir /export/cuenta1/direc
```

```
cp /usr/pub/ascii /export/cuenta1/direc
```

Al fichero creado le damos permiso de ejecución (`chmod +x fichero`) y lo ejecutamos (creará un directorio e introducirá en él el fichero `/usr/pub/ascii`).

20. Variables del shell

Se pueden definir variables de la forma `variable=valor`

Para ver el contenido de una variable, se usa `echo $variable`

Existen en UNIX variables predefinidas que sirven para configurar el entorno de cada usuario. Algunas de ellas son:

MAIL:	Directorio que contiene el correo
PS1:	Prompt primario
PS2:	Prompt secundario
TERM:	Tipo de terminal
LOGNAME:	Nombre del usuario
HOME:	Nombre del directorio por defecto
PATH:	Caminos de búsqueda de los comandos