

LOGIN Y PASSWORD

El proceso de Login distingue a un usuario de otro. Identifica a cada usuario con un nombre.

Tras arrancar el sistema:

Linux login: nombre de usuario

Password: palabra clave que da acceso al usuario para pasar al sistema

- Fijar un *password*:

% password

- Cambiar el *password*:

% password

CONSOLAS VIRTUALES

Linux = sistema multitarea: se pueden ejecutar diversos procesos simultáneamente.

Cada tarea en una consola virtual.

Para cambiar de consola:

ALT-F1 ALT-F8

Cada una de las consolas ofrece el *prompt* de entrada para poder acceder al sistema como si de otro usuario se tratara.

FICHEROS Y DIRECTORIOS

FICHERO: cualquier información salvada en disco identificada con un nombre.

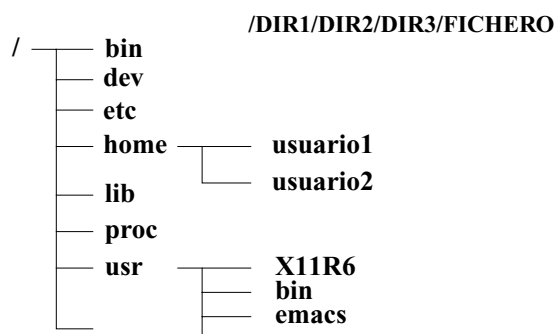
El nombre puede tener cualquier carácter excepto /

DIRECTORIO: colección de ficheros identificada con un nombre

Estructura de árbol:

/ Directorio raíz

PATH: identificación de un camino para alcanzar un fichero indicando el conjunto de directorios por lo que se atraviesa:



FICHEROS Y DIRECTORIOS

PATH ABSOLUTO: respecto a /

Ej: /home/usuario1/directorio1/fichero1

PATH RELATIVO: respecto al directorio en el que localmente nos encontremos

Ej: Si estamos en /home/usuario1

directorio1/fichero1

Directorio *home*: /home/usuario1 ~/directorio1/fichero1

~

Para referirnos a los ficheros de otros usuarios:

~usuario2/directorio1/fichero1 = /home/usuario2/directorio1/fichero1

- Comando `pwd` (*print work directory*): devuelve la posición actual absoluta dentro del árbol de directorio

Ej: %pwd
 /home/usuario1/directorio1

COMANDOS BÁSICOS DE LINUX

MOVIÉNDONOS ENTRE DIRECTORIOS:

- **cd (change directory)**

`% cd directorio_destino`

`% cd /usr/bin`
└──┬──┘
path absoluto

`% pwd`

`/home/usuario1`

`% cd directorio1/subdirectorio1`

└──┬──┘
path relativo

`% cd ..`

`%pwd`

`/home/usuario1`

`% cd directorio1/subdirectorio1`

`% cd ..`

`% cd ../../`

`% pwd`

`% pwd`

`/home/usuario1/directorio1`

`/home/usuario1`

`% cd`

`%pwd`

`/home/usuario1/directorio1/subdirectorio1`

`% cd`

`%pwd`

`/home/usuario1`

COMANDOS BÁSICOS DE LINUX

COPIAR, BORRAR, MOVER, CREAR

- **cp (copy)**

`% cp fichero1 fichero2`

`% cp -r directorio1 directorio2`

- **rm (remove)**

`% rm fichero1`

`% rm -i fichero1` (pregunta para confirmar)

`% rmdir directorio` (debe estar vacío)

- **mv (move)**

`% mv fichero1 fichero2` (renombrar al fichero)

`% mv fichero1 subdirectorio1` (cambia el fichero1 al subdirectorio1)

- **mkdir (make directorio)**

`% mkdir subdirectorio1` (crea el subdirectorio1 debajo de la posición actual dentro del árbol)

COMANDOS BÁSICOS DE LINUX

LISTAR FICHEROS, VER FICHEROS

- **ls (list)** (para ver el contenido del directorio actual)
 - % ls** **% ls -a** (lista ficheros ocultos)
.nombrefichero
 - % ls directorio1/subdirectorio1** **% ls -l** (ofrece información extra de los ficheros listados)
(para ver el contenido del directorio indicado)
 - % ls -t** (lista los ficheros ordenados temporalmente)
- **cat (catch)** (para ver el contenido de un fichero)
 - % cat fichero1**
- **more** (para ver el contenido de un fichero página a página)
 - % more fichero1** **space** avance de página
 - b** retroceso de página
 - /cadena** búsqueda de cadena en el fichero
 - q** abandonar la visualización del fichero
- **less** (variante del comando more)

COMANDOS BÁSICOS DE LINUX

- **file fichero** (muestra el tipo de fichero que es por el que se pregunta)
 - % file cuenta.c readme texto**
 - cuenta.c** executable
 - readme** ascii text
 - texto** English text
- **touch fichero** (cambia la fecha de modificación del fichero sin alterar el contenido. Si no existe crea uno con el nombre indicado pero vacío)

COMANDOS BÁSICOS DE LINUX

PÁGINAS DE MANUAL. AYUDA *ONLINE*.

- **man (manual)** (sistema de ayuda online)

Para preguntar por un comando de usuario, de administración, formato de ficheros, ...

Ej. `%man comando`

El manual está organizado en 9 secciones:

Sección 1: descripción de los comandos disponibles a todos los usuarios

Sección 2: descripción de las llamadas al sistema, las entradas al kernel

Sección 3: funciones disponibles en las librerías

Sección 4: ficheros especiales (dispositivos, comunicaciones, ...)

Sección 5: formato de ficheros usados por el sistema

Sección 6: juegos

Sección 7: paquetes macros

Sección 8: comandos de administración del sistema

Sección 9: rutinas del kernel

`% man sección comando`

COMANDOS BÁSICOS DE LINUX

`% man -f comando` informa sobre que partes del manual habla del comando.
Si no se especifica la sección, da como resultado la primera que encuentra.

`% apropos palabra_clave` te informa de todas las páginas de manual donde
puedes encontrar algo relacionado con *palabra_clave*

COMANDOS BÁSICOS DE LINUX

ALIAS

Permite simplificar los comandos que se teclean asociándolos a otras palabras.

```
% alias ll='ls -l'          ejecutará el comando ls -l como respuesta a teclear ll
% alias cd='cd \!* ;pwd;ls' ejecutará el comando cd (repetiendo los argumentos) seguido de pwd y
                           de ls como respuesta a teclear cd
```

HISTORY

Cada comando que se ejecuta desde un terminal se va almacenando en una lista de historia cuya longitud está controlada por la variable *history*.

```
% history                (devuelve la lista de los n últimos comandos ejecutados)
```

```
1 cat prueba
```

```
2 ls -l
```

```
3 cp prueba prueba2
```

```
4 rm pp
```

COMANDOS BÁSICOS DE LINUX

- Recuperación de comandos:

```
% !!                    repite el último comando dado
% !3                   repite el comando 3 de la lista history
% !c                   repite el último comando que empezaba por c (cp prueba prueba2)
% !-3                 repite el comando en tercer lugar de la lista empezando por la cola (ls -l)
% comando !$          utiliza como argumento del comando el mismo argumento que el comando anterior
% comando !S          utiliza como argumento del comando el último argumento del comando anterior
% comando !*          utiliza como argumentos del comando todos los argumentos del comando anterior
% ^antiguo^nuevo     modifica la cadena antiguo del comando anterior por nuevo
% !3:s/antiguo/nuevo  modifica la cadena antiguo del comando 3 de la lista history por nuevo
% ^antiguo^nuevo:p    modifica la cadena antiguo del comando anterior por nuevo pero sin ejecutarlo
```

IMPRESIÓN DE DOCUMENTOS

<code>% lpr fichero</code>	manda el fichero a la impresora definida por defecto
<code>% lpr -Pnombre_impr fichero</code>	manda el fichero a la impresora nombre_impresora (distinta a la por defecto)
<code>% lpq</code>	ofrece la lista de los ficheros que están en cola para ser impresos en la impresora por defecto
<code>% lpq -Pnombre_impr</code>	ofrece la lista de los ficheros que están en cola para ser impresos en la impresora nombre_impresora
<code>% lprm %núm_trabajo</code>	elimina el trabajo identificado de la cola de impresión
<code>% lprm -</code>	elimina todos los trabajos del usuario de la cola de impresión
<code>% lprm -Pnombre_impr %núm_trabajo</code>	elimina el trabajo identificado de la cola de impresión
<code>% lprm usuario1</code>	elimina todos los trabajos de usuario1 de la cola de impresión. (Para manejo del root)

DIRECTORIOS BÁSICOS

Qué contienen los principales directorios del sistema Linux?

<code>/home</code>	directorio de usuario
<code>/bin</code>	comandos esenciales de UNIX, por ej., <i>ls</i>
<code>/usr/bin</code>	otros comandos (distinción arbitraria con el directorio anterior)
<code>/usr/sbin</code>	comandos usados por root para la administración del sistema
<code>/boot</code>	algunos ficheros que participan en el proceso de “botado del sistema”
<code>/etc</code>	ficheros que usan otros subsistemas como de interconexión con otras máquinas, NFS, mail ...
<code>/var</code>	ficheros de administración (ficheros <i>log</i>)
<code>/var/spool</code>	almacenamiento temporal de los ficheros que van a ser impresos
<code>/usr/lib</code>	librerías estándares. Se usan en los <i>links</i> .
<code>/usr/lib/X11/</code>	distribución del sistema de <i>X window</i>
<code>/usr/include</code>	los ficheros <i>include</i> usados en la programación
<code>/usr/src</code>	fuentes a programas construidos en el sistema
<code>/etc/skel</code>	ejemplos de ficheros de comienzo que pueden ser copiados en el directorio <i>home</i> .

SHELLS

- SHELL: programa que interpreta y ejecuta los comandos del usuario donde nos encontramos después de haber hecho *login* en el sistema poderoso lenguaje de programación (*shells cripts*)

- TIPOS DE SHELLS:

bsh (*Bourne shell*)

csh (*C shell*)

- bash: *Bourne again shell*. (/bin/bash)

compatible con el Bourne Shell adaptando algunas mejoras que ofrece C shell.

Ofrece edición en línea

- csh: *C shell*. (/bin/csh)

mayormente compatible con el bsh a nivel de uso interactivo.

Diferente a nivel de programación

No ofrece edición en línea

- otros: sh, tesh, ksh, ...

Nosotros usaremos **bash**.

% echo \$SHELL responde con el tipo de shell en uso.

% chsh para cambiar el shell en uso.

Los caracteres * y ?

- sirven para hacer referencia a varios ficheros de una vez

* representa cualquier cadena de caracteres

? representa cualquier carácter simple

- ejemplos:

```
%ls
cap1 cap2 cap3 suma.c prueba resta.c
```

```
%ls cap?
cap1 cap2 cap3
```

```
%ls prue??
prueba
```

```
%ls *.c
suma.c resta.c
```

```
% ls cap[12]
cap1 cap2
```

```
%ls *u*
suma.c prueba
```

```
% ls cap[1-3]
cap1 cap2 cap3
```

```
%ls c*
cap1 cap2 cap3
```

```
%ls s*c
suma.c
```

```
%ls *a
prueba
```

Redireccionamiento

- stdin (standard input) es la entrada estándar
- stdout (standard output) es la salida estándar

- Muchos comandos toman su entrada de stdin y mandan su salida a stdout.

- El shell toma como stdin el teclado

- El shell toma como stdout la pantalla

- ejemplo:

% cat mult.c este comando lee datos del fichero mult.c y los manda a stdout (la pantalla)

% cat si no especificamos fichero de entrada lee los datos de stdin y los envia a stdout

```
%cat
esto es una prueba
esto es una prueba
adiós
adiós
Ctrl-D
%
```

Redireccionamiento

- ejemplo:

% sort si no especificamos fichero de entrada lee los datos de stdin y los envia a stdout (sort ordena alfabéticamente)

```
%sort
galletas
naranjas
lentejas
Ctrl-D      ← fin de fichero
galletas
lentejas
naranjas
%
```

- redireccionamiento de la salida:

nos permite enviar la salida a un fichero en lugar de a la pantalla

```
% sort > compra
galletas
naranjas
lentejas
Ctrl-D
%
```

Redireccionamiento

- redireccionamiento de la entrada:

nos permite tomar la entrada de un fichero en lugar de del teclado (no siempre es necesaria)

```
%sort < lista
galletas
lentejas
naranjas
%
```

- redireccionamiento entre comandos (pipes)

nos permite enviar la salida de un comando como entrada de otro comando

```
% ls | sort -r
```

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

podemos conectar más de dos comandos:

```
% ls /usr/bin | sort -r | head -5
```

Redireccionamiento

- redireccionar la salida de un comando a un fichero es *destrutivo*

```
% ls > lista_ficheros
```

destruye el anterior contenido (si lo había) de lista_ficheros

- es posible usar ">>" para redireccionar

```
% ls >> lista_ficheros
```

añade la salida de "ls" al final de lista_ficheros

- redireccionamiento no destructivo

```
% set -o noclobber
```

esta opción impide la destrucción de un fichero existente al usar la redirección

- es posible, si queremos, forzar la escritura del fichero con ">|"

```
% ls >| lista_ficheros
```

- "set -o noclobber" es equivalente a "set -C"

- para eliminar su efecto tenemos "set +C"

Redireccionamiento

- stderr: es la salida de errores estándar

- podemos redireccionar la salida de errores

```
% gcc suma.c 2> lista_errores
```

- podemos redireccionar simultáneamente la salida y la salida de errores

```
% gcc suma.c &>lista_errores
```

- redireccionar a /dev/null

```
% gcc suma.c 2> /dev/null
```

```
% gcc suma.c 2> lista_errores > /dev/null
```

PROPIEDAD Y PERMISOS DE FICHEROS

Dado que Linux permite multiusuarios, los ficheros creados poseen propietarios, y a través de los permisos se puede prohibir el acceso de ciertos propietarios a los ficheros/directorios de otros.

Permission denied

respuesta que da el sistema cuando no tenemos permiso de acceso al fichero/directorio que pretendemos.

- **PERMISOS FICHERO:** distintas formas de uso de un fichero.

Read: permiso de lectura. Se puede ver el contenido de un fichero.

Write: permiso de escritura. Se puede cambiar o borrar el fichero.

Execute: permiso de ejecución. Se puede ejecutar el fichero como un programa o *shell script*.

- **PERMISOS DIRECTORIO:** distintas formas de uso de un directorio.

Read: permiso de lectura. Se puede listar el contenido del directorio (*ls*).

Write: permiso de escritura. Se puede añadir o borrar ficheros en el directorio.

Execute: permiso de paso. Se puede entrar en el directorio (*cd*).

- **NIVELES DE PERMISOS:**

owner: el usuario que creó el fichero

group: grupo de usuarios. Cada usuario pertenece, al menos, a un grupo.

other: el resto del mundo que no sea owner ni pertenezca al grupo.

all: agrupa a los tres anteriores

PROPIEDAD Y PERMISOS DE FICHEROS

INTERPRETANDO LOS PERMISOS

% ls -l cursolinux

```

fichero
↑
- r w - r - - r - -      1 carmen   users      505 Mar 13 19:05      cursolinux
↓
permisos
owner
permisos
group
permisos
other

r: read
w: write
x: execute

directorio      link
↑              ↑
d r w - r - - r - -      l r w - r - - r - -

Ejemplos:      - r w x r - x r - x      - r w - - - - -
    
```

PROPIEDAD Y PERMISOS DE FICHEROS

DEPENDENCIAS DE LOS PERMISOS

Para acceder a un fichero es **NECESARIO** que el directorio en el que se encuentre posea permiso de paso (x)

Por ej. El directorio home del usuario juan tiene los siguientes permisos:

```

d r w x - - - - -      1 juan      users      512 Jun 24 13:43 juan
% ls -l
- r w x r w x r w x      1 juan      users      512 Jun 24 13:44 fichero1
- r w x r w x r w x      1 juan      users      512 Jun 24 13:51 fichero2
- r w x r w x r w x      1 juan      users      512 Jun 24 13:54 fichero3
    
```

NO HAY ACCESO a lo ficheros porque **NO HAY PERMISO DE PASO** en el directorio ni al grupo ni al “resto del mundo”

- Permisos típicos:

```

- r w - r - - r - -      para los ficheros
d r w x r - x r - x      para los directorios
    
```


PROPIEDAD Y PERMISOS DE FICHEROS

PERMISOS POR DEFECTO

Se introducen dentro del fichero de comienzo del shell `.bash_profile` (para *bash shell*)

`umask d1d2d3` cada dígito responde a un nivel:
d₁: usuario
d₂: grupo
d₃: resto del mundo

Se calcula igual que con el comando `chmod` y el número resultante se resta de 777

Ej. Si queremos: nivel de usuario: todos los permisos, $400 + 200 + 100 = 700$
nivel de grupo: permiso de lectura y ejecución, $40 + 10 = 50$
nivel de "resto del mundo": ningún permiso, 0
 $777 - 750 = 027$, por tanto

`umask 027`

LINKS

Permiten dar a un simple fichero más de un nombre

inode number nombre del fichero

cada nombre de fichero en un directorio es un link al *inode*

`% ls -i fichero` devuelve el inode del fichero

Ej. `% ls -i pepe`
22192 pepe

HARD LINKS: asocia distintos nombres de fichero al mismo inode. Se pueden crear dentro de un mismo sistema de ficheros.

`% ln pepe juan`

`% ls -i pepe juan`

22192 juan 22192 pepe ambos son el mismo fichero. Cambios en uno aparecen en el otro.

`% ls -l pepe juan`

- r w - r - - r - - 2 root root 214 Jul 13 17:44 juan

- r w - r - - r - - 2 root root 214 Jul 13 17:43 pepe

número de *hard links* del fichero

LINKS

SYMBOLIC LINKS: permite dar otro nombre a un fichero pero NO hace un *link* con el inode.

Se crea un fichero que apunta a otro fichero.

Cuando se invoca al fichero nuevo realmente se accede al fichero original.

```
% ln -s pepe juan
```

```
% ls -li pepe juan
```

```
22195 juan    22192 pepe
```

```
% ls -l pepe juan
```

```
l r w x r w x r w x l root  root    3 Jul   13  17:44 juan -> pepe
```

```
- r w - r - - r - - 1 root  root    214 Jul  13  17:43 pepe
```

Los permisos en los links simbólicos no se usan. Ellos heredan los permisos de los ficheros a los que apuntan.

Control de procesos (jobs)

-proceso o **job**: es cada tarea que realiza el sistema

- se inicia un proceso cada vez que se corre un programa

- el comando "ps" muestra la lista de procesos

```
% ps
```

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

- existen opciones de "ps" que permiten ver los procesos de todos los usuarios de un sistema (ps -aux)

- es posible correr más de un proceso a la vez

- pueden estar en **foreground** o **background**

- **foreground**: es cuando el proceso está en modo interactivo

- puede recibir datos del teclado

- no tenemos prompt mientras haya un proceso en foreground

- no podemos hacer nada mientras el proceso dure

- sólo puede haber un proceso en foreground

Control de procesos (jobs)

- **background**: es cuando el proceso se desarrolla sin interacción
 - el prompt sigue apareciendo tras arrancar un proceso en background
 - podemos correr otros programas o comandos a la vez que el proceso original corre en background
 - puede haber múltiples procesos en background
- **suspender un proceso (Ctrl-Z)**
 - el proceso se para temporalmente
 - podemos hacer que siga ejecutandose en background o foreground
 - el trabajo continuará exactamente donde paró
- **interrumpir un proceso (Ctrl-C)**
 - el proceso es eliminado
 - no podemos recuperarlo

Control de procesos (jobs)

- ejemplo:

```
% yes
y
y
y
y
...
```

```
%yes > /dev/null
```

- en ambos casos el proceso está en foreground e impide la ejecución de otros comandos (no hay prompt)

- Ctrl-C elimina el proceso

- para ejecutar el programa en background (“&”):

```
% yes > /dev/null &
% [1] 164
%
```

[1] -->> job number, es el número de trabajo asignado

164 -->> es el PID asignado por el sistema

- el programa se ejecuta en background y aparece el prompt

- para **chequear** el estado del proceso (“jobs”):

```
%jobs
[1]+  Running      yes > /dev/null &
%
```

- también “ps” nos muestra el proceso

Control de procesos (jobs)

- para eliminar el proceso que corre en background (“kill”):

```
% kill %1
```

```
% kill 164
```

- son equivalentes

- para chequear si se ha eliminado el proceso (“jobs”):

```
%jobs  
[1]+  Terminated  yes > /dev/null  
%
```

- comandos “fg” y “bg”:

- permiten reanudar un trabajo suspendido

- fg: reanuda el trabajo en foreground

- bg: reanuda el trabajo en background

- ejemplo:

```
%yes > /dev/null  
Ctrl-Z  
[1]+  Stopped  yes > /dev/null  
%
```

```
% fg  
yes >/dev/null
```

```
%bg  
[1]+  yes >/dev/null &  
%
```

Control de procesos (jobs)

- “fg” y “bg” sin argumentos:

- actúan sobre el último trabajo que paró

- este aparece señalado por un + al hacer jobs

- podemos dar como argumento el job number (no PID):

```
% fg %2
```

```
% bg %2
```


EL EDITOR *vi*

- editor de texto
- ficheros compuestos por texto: cartas, programas en C, ficheros de configuración
- no es el más fácil
- de uso muy extendido en UNIX/Linux
- es muy potente
- tres modos de operación:
 - modo comando: - es el modo al arrancar *vi*
 - cambia a los otros modos
 - se ejecutan comandos cortos
 - modo de inserción: - permite escribir texto
 - modo de línea: - se ejecutan comandos largos, además de cortos
 - aparecen en la última línea del fichero

EL EDITOR *vi*

- sintaxis:
vi nombre_de_fichero

Ejemplo:
- *vi prueba*



"prueba" [New file]

- se encuentra en modo comando

- para insertar texto --> PULSAR **i**
(modo de inserción)
--> teclear el texto, este aparecerá a partir de la posición del cursor en la pantalla

- para volver al modo comando --> PULSAR **Esc**

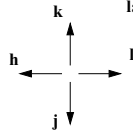
EL EDITOR vi

- otras formas de insertar texto:

- pulsar **a**
el texto se añade tras la posición del cursor
- pulsar **o**
el texto se añade en la línea siguiente
- pulsar **O**
el texto se añade en la línea anterior

- funciones en modo comando:

- podemos movernos por el texto usando las flechas del teclado o las teclas h, j, k, l



- pulsar **h**: mueve el cursor a la izqda.
- pulsar **j**: mueve el cursor hacia abajo
- pulsar **k**: mueve el cursor hacia arriba
- pulsar **l**: mueve el cursor a la dcha.

- podemos borrar texto:

- pulsar **x**: borra el carácter bajo el cursor
variantes: 5x, 7x, ...
- pulsar **dd**: borra la línea en la que se encuentra el cursor
variantes: 2dd, 5dd, ...
- pulsar **dw**: borra la palabra en la que se encuentra el cursor
variantes: 2dw, 5dw, ...

EL EDITOR vi

- funciones en modo comando:

- podemos modificar texto

- pulsar **r**: sustituye el carácter bajo el cursor por aquél que tecleemos
- pulsar **R**: sustituye los caracteres tras el cursor por aquellos que tecleemos (pulsar Esc al final)
- pulsar **~**: sustituye minúsculas por mayúsculas y viceversa

- más sobre movernos por el texto

- pulsar **0**: nos lleva al principio de la línea
- pulsar **\$**: nos lleva al final de la línea
- pulsar **Ctrl-E**: mueve el cursor una pantalla completa hacia delante
- pulsar **Ctrl-B**: mueve el cursor una pantalla completa hacia atrás
- pulsar **G**: mueve el cursor al final del fichero
- variantes: 1G, 10G, ...
- pulsar **H**: mueve el cursor a la 1ª línea de la pantalla
- variantes: 2H, 10H, ...
- pulsar **L**: a la última línea de la pantalla
- variantes: 2L, 10L, ...

EL EDITOR vi

- funciones en modo comando:

- más sobre movernos por el texto

- pulsar **w**: de palabra en palabra hacia delante
- pulsar **e**: nos lleva al final de la palabra
- pulsar **b**: de palabra en palabra hacia atrás

- se pueden combinar acciones:

- **db**: borra la palabra anterior
- **dL**: borra hasta el final de la pantalla
- **d6L**: borra hasta la 6ª línea antes del final de pantalla

- otros:

- **u**: deshace el último cambio
- **U**: recupera la línea como estaba al principio
- pulsar **⏪**: repite el último comando que hizo un borrado
- **fc**: busca el carácter "c" en la línea actual
- **Fc**: lo mismo hacia atrás
- **cw**: cambia la palabra donde está el cursor por el texto que se teclee (terminar con Esc)
- **p**: recupera el último trozo borrado a partir del cursor
- **P**: recupera el último trozo borrado antes del cursor

EL EDITOR vi

- funciones en modo de línea:

para entrar en modo de línea

- 1) pulsamos **⏪**
- 2) aparece '⏪' en la última línea de la pantalla
- 3) escribimos el comando y pulsamos **enter**

:q --> para salir del editor si no se ha modificado nada

:q! --> para salir del editor sin salvar aunque se hayan hecho modificaciones

:w --> para salir del editor salvando las modificaciones

:w nombre_de_fichero --> para salvar en el fichero que nombremos

:r nombre_de_fichero --> para incluir un fichero a partir de la posición del cursor

:set opción --> para fijar opciones en el editor

- : set number / nonumber
- : set autoindent / noautoindent
- : set all

:S/cadena1/cadena2 --> sustituye cadena1 por cadena2

:1,\$s/cadena1/cadena2 --> sustituye todas las ocurrencias de cadena1 por cadena2

EL EDITOR vi

- funciones en modo de línea:

:! comando --> escapa al shell para ejecutar el comando

!:bash --> para dar más de un comando, al final con Ctrl-D se vuelve al editor

:ab edcus Centro de Calculo de la Universidad de Sevilla

-->> define una abreviatura

:map cad1 cad2

-->> define una macro

:map q :wq

- búsqueda de texto

:/cadena -->> localiza la 1ª aparición de cadena a partir del cursor

:?cadena -->> localiza la 1ª aparición de cadena antes del cursor

- en modo comando podemos repetir la última búsqueda

pulsando n (busca hacia delante)

pulsando N (busca hacia detrás)

SHELL SCRIPTS

- son ficheros de texto que permiten agrupar comandos

- ejemplo:

```
#!/bin/sh
# shell script para crear e imprimir libro
cat cap1 cap2 cap3 > libro
wc -l libro
lp libro
```

#!/bin/sh identifica el fichero como shell script

la segunda línea es un comentario (#)

el resto de las líneas son los comandos que el shell ejecutará

Es necesario dar permiso de ejecución al fichero

```
chmod +x nombre_de_fichero
```

FILTROS

- filtros: son programas que realizan lo siguiente
 - toman datos de stdin
 - procesan estos datos
 - mandan el resultado a stdout
- su entrada y su salida pueden redireccionarse:
- ejemplos ya conocidos: cat, sort, head
 - cat: no realiza ningún procesado
 - sort: ordena alfabéticamente
 - con la opción -r ordena de modo inverso
 - con la opción -n ordena numéricamente
 - head: deja pasar sólo las primeras líneas
 - mediante opciones controlamos la cantidad de líneas que pasan
- otros:
 - tail: equivalente a head pero con las últimas filas
 - grep: busca en su entrada la aparición de una cadena y deja pasar sólo la línea que la contiene
 - awk: comando muy potente, permite un procesado complejo de los datos de entrada

FILTROS

- ejemplos

```
% ls -l | grep drwx
```

mostrará sólo los directorios

```
% ls /dev | tail -20
```

mostrará los últimos 20 dispositivos de /dev

```
% vi tlf
#!/bin/sh
# shell script para agenda
grep "$*" << END
Juan López          954211340
Pedro Pérez        954534177
Maria Martín       954223319
.....
end
```

este shell script toma un argumento y extrae del fichero la línea donde el mismo aparece:

```
% tlf Pérez
Pedro Pérez        954534177
```

```
% tlf Jua
Juan López         954211340
```

FILTROS (AWK)

- awk:
 - permite un procesamiento complejo de los datos de entrada
 - es **más que un comando** es un lenguaje de programación
 - es muy versátil y potente

- el tipo de filtrado que realiza awk no está prefijado (como en grep o en tail) ha de especificarse mediante un programa escrito en el lenguaje AWK

- en su uso más básico:

- especificamos el programa en la línea de comando
- la entrada se proporciona por el teclado
- la salida se obtiene en pantalla

```
% awk '{print $1 " " $3}'
referencia  precio  precio_rebajado
referencia  precio_rebajado
20134      30000  28500
20134      28500
13456      5700   4495
13456      4495
Ctrl D
%
```

FILTROS (AWK)

- otra opción:

- especificamos el programa en la línea de comando
- la entrada se proporciona desde un fichero
- la salida se obtiene en pantalla

```
% vi almacen
referencia  precio  precio_rebajado
20134      30000  28500
13456      5700   4495
34222      64000  55900
.....
% awk '{print $1 " " $3}' almacen
referencia  precio_rebajado
20134      28500
13456      4495
34222      55900
.....
%
```

- podemos redireccionar la salida a otro fichero

```
% awk '{print $1 " " $3}' almacen > nuevos_precios
```

FILTROS (AWK)

- otra opción:

- especificamos el programa en un fichero

```
% vi precios.awk
{
print $1 " " $3
}
% awk -f precios.awk almacen > nuevos_precios
%
```

- esto es útil pues los programas pueden ser complejos

-otro ejemplo:

```
% vi media.awk
{sum += $2}
END {print sum/NR}
-----
% vi notas
examen1      7.5
examen2      5
examen3      3
 practica1    4.5
 practica2    9
.....
%awk -f media.awk notas
5.25
```

FILTROS (AWK)

- el lenguaje awk permite sentencias de control de flujo del tipo de los lenguajes habituales de programación

- if (expr) statement
- while (expr) statement
- do statement while (expr)
- for (expr ; expr ; expr) statement
- break

- reconoce secuencias de escape:

```
\a alert
\t tabulador
\n newline
\r return
\' ''
```

- considera dos tipos de datos:

- numéricos: pueden ser:
 - enteros (-2) ,
 - decimales (1.08)
 - incluye notación científica (-1.1e4, .28E-3)

- cadenas

- operadores: =, +, -, *, /, %, ^, !, &&, <, >, >=, <=, ==, !=, +, -, *, /, %, ^, !,

FILTROS (AWK)

- dispone de una librería de funciones aritméticas:

- `cos(x)`, `sin(x)`
- `int(x)`, `sqrt(x)`
- `exp(x)`, `log(x)`
- `rand()`, `srand(expr)`, `srand()`

-permite escribir con formato al estilo del lenguaje C

- `printf format, expr-list`
- `print expr1, expr2, ...`

- permite definir funciones al estilo de C

Variables del shell

- definición de variables:

- son internas al shell (sólo el shell puede acceder a ellas)
- útiles en los shell scripts

- para asignar valor a una variable:

operador "="

- para referirnos al valor asignado a una variable:

prefijo "\$"

- ejemplo:

```
% mens='hola'
% echo $mens
hola
```

```
% echo `hola`
hola
```

igual resultado

- Importante: no escribir espacios en las definiciones de variables

- el comando "set" muestra la lista de las variables definidas

VARIABLES DEL ENTORNO

- entorno:

- es el conjunto de variables que son accesibles por todos los comandos que se pueden ejecutar

- las variables definidas en el shell pueden exportarse al entorno con el comando export

- se pueden configurar algunos comandos a través de las variables del entorno

- algunas variables del entorno

- PAGER --> controla si las páginas de manual se muestran parando de página en página o no

ejemplo:

```
% PAGER =cat
% export PAGER
% man ls
```

- si cambiamos de nuevo el valor de una variable, ya no necesitamos exportarla, se hace automáticamente

```
% PAGER=more
% man ls
```

- cuidado con los espacios!

VARIABLES DEL ENTORNO

- las páginas de manual de cada comando informan de si este usa alguna variable del entorno

- las variables del entorno también guardan información sobre la sesión abierta

- algunas variables del entorno

- HOME --> contiene el nombre del directorio home

```
% echo $HOME
```

- PS1 --> define el prompt

```
% PS1='nuevo comando: '
nuevo comando:
```

```
nuevo comando: PS1='\w# '
/home/pepe#
```

- PATH --> contiene una lista de directorios separados por el carácter ":" en los que localizar ficheros ejecutables

evita tener que dar el path completo a comandos y ejecutables

```
% echo $PATH
```

Variables del entorno

- algunas variables del entorno

- PWD -->> contiene el nombre del directorio de trabajo

```
% echo $PWD
```

- OLDPWD -->> contiene el nombre del anterior directorio de trabajo

```
% echo $OLDPWD
```

- GROUPS -->> contiene una lista de los grupos de los cuales forma parte el usuario

```
% echo $GROUPS
```

- BASH -->> contiene el nombre del ejecutable del shell

```
% echo $BASH
```

- BASH_VERSION -->> contiene el número de versión del shell

```
% echo $BASHVERSION
```

- HOSTNAME -->> contiene el nombre de la máquina

```
% echo $HOSTNAME
```

Variables del entorno

- algunas variables del entorno

- HISTFILE -->> contiene el nombre del fichero en el que el comando history se salva por defecto es .bash_logout

```
% echo $HISTFILE
```

- HISTSIZE -->> contiene el número de comandos a almacenar en el comando history, el valor por defecto es 500

```
% echo $HISTSIZE
```

- HISTFILESIZE -->> contiene el número de líneas del fichero history, el valor por defecto es 500

```
% echo $HISTFILESIZE
```

Ficheros de inicialización

- ficheros de inicialización:

- son shell scripts
- se ejecutan automáticamente con ciertas acciones
- en ellos se pueden definir y exportar variables

- algunos ficheros de inicialización:

`/etc/profile` --> es fijado por el administrador de sistemas y se ejecuta al hacer login

`$HOME/.bash_profile` --> es fijado por el usuario y se ejecuta al hacer login

`$HOME/.bashrc` --> es fijado por el usuario y se ejecuta al abrir un shell sin login

`$HOME/.bash_logout` --> es fijado por el usuario y se ejecuta al cerrar la sesión

Ficheros de inicialización

- ejemplos:

`$HOME/.bash_logout`

```
#$HOME/.bash_logout
clear
echo "SHOSTNAME: $(whoami) logged out at $(date)"
HISTORY=0
\rm $HOME/core
\rm $HOME/.bash_history
```

`$HOME/.bashrc`

```
#$HOME/.bashrc
alias la='ls -a'
alias ll='ls -l'
alias cp='cp -i'
alias mv='mv -i'
alias rm='rm -i'
export PSI='su comando, por favor >'
set -o noclobber
export HISTSIZE=20
export PATH=/bin:/usr/bin:/usr/local/bin
```

ARCHIVO Y COMPRESIÓN DE FICHEROS

COMPRESIÓN

Reducir la cantidad de espacio en disco requerida para almacenar un fichero o conjunto de ficheros.

USANDO GZIP:

gzip toma un fichero, lo comprime, salva el fichero comprimido con la extensión .gz y borra el fichero original no comprimido.

```
Ej. % ls -l prueba
-rw-r--r-- 1 pepe curso 312996 Jan 30 21:44 prueba
% gzip prueba
% ls -l
-rw-r--r-- 1 pepe curso 103441 Jan 30 21:45 prueba.gz
% gzip -l prueba.gz (da información sobre la compresión realizada)
compressed  uncompr.      ratio      uncompressed_name
103441      312996      67%      prueba
```

Para volver al fichero original no comprimido:

```
% gunzip fichero.gz
```

ARCHIVO Y COMPRESIÓN DE FICHEROS

ARCHIVO

Empaquetado de un conjunto de ficheros en uno simple manteniendo la información de propiedades y permisos de cada uno.

USANDO TAR (tape archive):

```
% tar funcionopciones ficheros      funcion: función a realizar
                                       opciones: opciones de la función
```

Lista de funciones:

- c: crear un nuevo archivo
- x: extraer ficheros de un archivo
- t: listar el contenido de un archivo
- r: añadir ficheros al final de un archivo
- d: comparar ficheros de un archivo a los existentes en el sistema de ficheros

Lista de opciones más comunes:

- v: ofrece en pantalla información del empaquetamiento o desempaquetamiento mientras se ejecuta.
- f fichero: especifica el nombre del fichero a leer o escribir

ARCHIVO Y COMPRESIÓN DE FICHEROS

Ejemplos de creación de un archivo:

```
Siendo docum un directorio
% tar cf docum.tar docum      (empaqueta el directorio docum en el archivo docum.tar)
% tar cvf docum.tar docum
docum/
docum/fichero1
docum/fichero2
docum/fichero3
..... (resto de los ficheros del directorio)
% tar cvvf docum.tar docum    (más v's más información durante el proceso)
drw-r--r--  1      pepe/curso 0   Jan 30 21:45  docum/
-rw-r--r--  1      pepe/curso 43  May 3  22:11  docum/fichero1
-rw-r--r--  1      pepe/curso 72  Jan 13 09:45  docum/fichero2
-rw-r--r--  1      pepe/curso 88  Sep 23 13:13  docum/fichero3
```

ARCHIVO Y COMPRESIÓN DE FICHEROS

Ejemplos de extracción de un archivo:

```
% tar xvf docum.tar          (creará un subdirectorio docum y dentro de él todos los archivos
                             desemapaquetados con los mismos propietarios y permisos que el
                             original)

docum/
docum/fichero1
docum/fichero2
docum/fichero3
% cd docum
% tar cvf docum.tar *        empaquetará todos los ficheros que están en el directorio actual
                             pero NO empaqueta el directorio en sí (docum).
```

Extracción de ficheros individualizados:

```
% tar xvf tarfile ficheros
% tar xvf docum.tar docum/fichero2  crearía un subdirectorio docum y en él el fichero2
```

USNADO TAR CON GZIP: concatena los dos programas

```
% tar cvzf docum.tar.gz docum
% tar xvzf docum.tar.gz
```

Conexiones remotas

- conexión remota

- nos convertimos en usuarios de un ordenador distinto a aquél en el que estamos directamente conectados
- usamos nuestro computador como terminal del otro
- es necesario tener cuenta en el ordenador remoto y conocer su dirección IP

- telnet:

- es una facilidad de internet para hacer login remoto
- algunos sistemas están disponibles al público en general mediante una cuenta cuyo username es "guest" y admite como password la dirección de e-mail (o no pide)
- es interactivo: ofrece un prompt (telnet>) y se introducen comandos en línea
- no es necesario teclear completamente los comandos de la línea de comando

Conexiones remotas

- ejemplo de conexión

```
% telnet
telnet> help
Commands may be abbreviated. Commands are:

close      close current connection
.....
open       connect to a site
quit       exit telnet
.....
z          suspend telnet
environ    change environment variables (environ ?
           for more)

telnet>
telnet> open
(to) pcte6.dte.us.es
Trying 150.214.141.177 ...
Connected to pcte6.dte.us.es
Escape character is ^].
Debian GNU/Linux 2.1 pcte6.dte.us.es

pcte6 login:
```

FTP

FTP (*File Transfer Program*) programa en protocolo TCP/IP usado por los computadores para realizar transferencia entre copias de ficheros, grupos de ficheros y/o directorios.

- Transferencia bidireccional entre nuestro sistema y el remoto.

HAY que conocer el nombre del Dominio Internet de la máquina remota (*hostname*) (IP address)

HAY que tener cuenta de usuario en la máquina remota. (*password*)

Tipos de ficheros a transferir:

ejecutables y fuentes

ascii

gráficos

tar

multimedia, video, audio

hojas de cálculo y bases de datos

Formatos posibles de los ficheros a transferir:

ascii (texto) tipo por defecto

binario (imagen)

HAY que elegir el formato correcto antes de hacer la transferencia.

FTP

% ftp hostname

ftp>

% ftp

ftp> open hostname

- Comando básicos:

! comando ejecuta el comando en el computador local

Ej. ftp> !pwd

/home/usuario1

! sales provisionalmente del ftp para ejecutar comandos en el shell local

Control_D vuelve al ftp

ascii fija el formato de la transferencia a formato ascii

binary,image fija el formato de la transferencia a formato binario

bye,close,exit,quit finaliza la sesión de FTP

cd directorio_remoto cambio de directorio dentro del árbol del ordenador remoto

dir lista el contenido del directorio de la máquina remota

ftp> dir docu* listadocu vuelca el listado de los ficheros que empiezan por docu del directorio remoto dentro del fichero local listadocu.

FTP

get fichero_remoto [fichero_local] lista el contenido del directorio de la máquina remota

hash informa del porcentaje de información transferida.

Función *Toggle*. Si está en off al invocar hash pasa a off, y al revés.

```
ftp> hash
Hash mark printing on
ftp> get aventura.tar
200 PORT command successful.
150 Opening BINARY mode data connection for aventura.tar (212355bytes)
#####
```

help, ? resumen de comandos

lcd [directorio] cambio de directorio en la máquina local. (Suponiendo que estoy en /home/usuario1)

ftp> lcd directorio1

ftp> !pwd

/home/usuario1/directorio1

FTP

ls [directorio remoto] [fichero local] dir

mget ficheros_remotos transfiere copias de los ficheros especificados al directorio local

ftp> mget fichero1 fichero2 fichero3

mput ficheros_locales transfiere copias de los ficheros locales especificados al directorio remoto

open sistema_remoto abre una conexión FTP al sistema especificado por su nombre de dominio Internet o por su PI address

ftp> open ftp.apple.com

ftp> open 130.43.2.3

put fichero_local [fichero_remoto] transfiere copia del fichero local especificado al directorio remoto

pwd muestra el directorio de trabajo en la máquina remota

type muestra el tipo de formato al que esta fijada la transferencia

Para abortar una sesión de FTP Control-C

FTP anónimo: tipo de acceso de sólo lectura en la máquina remota. Típica entre usuarios de Internet.

% ftp [hostname]

Name (hostname): anonymous

Password: dirección de e-mail (si se tiene)

Conexiones remotas

- rlogin, rsh, ssh:

son comandos para conectarse a otras máquinas de la red

```
% rlogin [-l username] host
```

- hace un login del usuario `username` en la máquina remota `host`

- si no se especifica `username` supone el mismo `username` en las dos máquinas (local y remota)

```
% rsh [-l username] host [command]
```

- ejecuta un comando en una máquina remota

- si no se especifica comando, hace login usando `rlogin`

```
% ssh [-l username] hostname [command]
```

- hace login o ejecuta comandos en una máquina remota

- encripta la comunicación entre las máquinas lo que lo convierte en una forma segura de transmitir información por la red

- sustituye a `rlogin` y `rsh` cuando hay problemas de inseguridad

Conexiones remotas

- rcp, scp:

permiten copiar ficheros de una máquina remota sin abrir una sesión en ella

```
% rcp filename1 filename2
```

- copia ficheros entre máquinas

- con la opción `-r` copia directorios

- el fichero o directorio remoto se especifica:

```
username@hostname:path
```

- si se especifica como `hostname:path` considera el mismo `username` en las dos máquinas

- si el `path` no es completo se considera relativo al directorio `home` en la máquina remota

```
% scp filename1 filename2
```

- copia ficheros entre máquinas pero encripta la comunicación

- es la versión segura de `rcp`

CALCULADORA ONLINE. bc

BC es un lenguaje y un compilador para hacer cálculos aritméticos con precisión arbitraria.

La sintaxis es sustancialmente la del lenguaje C.

`% bc [-lwsqv] [fichero]`

Precisión:

`length`: número de dígitos con los que se trabaja

`scale`: número de dígitos fraccionarios con los que se trabaja

Ej, `length=7, scale=3` 1234.567

Tipos de variables:

variables simples

arrays

Variables especiales:

`ibase`: sistema numérico para representar los números de entrada. Por defecto: base 10

`obase`: sistema numérico para representar los números de salida. Por defecto: base 10

`last`: variable que toma el valor del último número escrito

`scale`

CALCULADORA ONLINE. bc

Operadores:

`+` suma `-` resta `*` multiplicación `/` división `%` resto de la división

`^` potencia `>` mayor que `>=` mayor igual `<` menor que `<=` menor igual

`==` igualdad `!=` distinto `=` asignación `&&` and `||` or

Sentencias de programación:

`if (expresión) sentencias`

`while (expresión) sentencias`

`for (expresión1;expresión2;expresión3) sentencias`

`halt, break, continue, return, quit, ...`

Funciones matemáticas:

`s(x)` seno de x. En radianes

`c(x)` coseno de x. En radianes

`a(x)` arcotangente de x. En radianes

`l(x)` logaritmo natural de x.

`e(x)` exponencial de x.

`j(n,x)` función de Bessel de orden entero n de x

CALCULADORA ONLINE. bc

Ejemplo1:

```
%bc
a=3
(a*a)%3
0
quit
```

Ejemplo2:

```
% bc
define a(x,y){
z=x*y
return (z)
}
.....
a(3,2)
6
```

Ejemplo3:

```
% bc
for (i=1,i<=5;i++)
i
1
2
3
4
5
```

Estas funciones se podían haber escrito en unos ficheros y después invocar bc con el nombre del fichero como argumento.

OTROS COMANDOS

mtools

```
mcopy:      % mcopy pepe a:
mcd         % mcd a:\dir1\subdir2
mtype      % mtype a:\fichero1
mdel       % mdel a:\fichero1
mdir       %mdir a:\dir1
```

quota solo responde si se ha sobrepasada la cuota

quota -v responde con la cuota consumida y el limite que tenemos (Kbytes)

Filesystem	usage	quota	limit	timeleft	files	quota	limit	timeleft
/	208645	320000	325000		8704	14000	15000	

date muestra la fecha en pantalla

```
Tue Feb 1 12:08:41 MET 2000
```

at at HH:MM <rt>

```
at> comandos
```

```
at> Control-d
```

source: ejecuta los ficheros de configuración