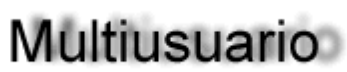
5. Uso de la Shell

5.1. Multiusuario



Como se ha dicho, el sistema UNIX es multiusuario y multiproceso, esto significa que varios usuarios pueden estar ejecutando varios procesos a la vez. Para poder hacer esto, el sistema tiene que tener unas determinadas características:

- Sistema de seguridad de acceso. Se debe garantizar que un usuario tiene a salvo su trabajo libre de accesos no permitidos. En el caso del sistema UNIX a través de un password (ver capítulo de gestión de usuarios).
- Seguridad de ficheros. Se debe garantizar que los usuarios no accedan y por tanto cambien o destruyan ficheros del sistema o de otros usuarios. En UNIX se utilizan tres tipos de permisos y tres modos de acceso que se verán posteriormente.
- Compartición de recursos. Los recursos comunes deben ser repartidos equitativamente entre todos los usuarios (ver capítulo de gestión de recursos). En el UNIX se utilizan sistemas de *spooling.*
- Control de recursos. También se debe llevar la cuenta de los gastos de los distintos usuarios (uso de CPU, papel, etc.). Se llama *accounting*.
- Administración de sistemas. El sistema debe garantizar mecanismos para hacer *backups* de los trabajos de los usuarios y la posible recuperación de errores (ver capítulo de gestión de ficheros).

Como hemos dicho, la seguridad del sistema se lleva a cabo en dos niveles, por un lado pidiendo el *password* al usuario, y por otro lado y una vez dentro del sistema, no permitiendo el acceso a ficheros no autorizados. El password es una palabra secreta que sólo conoce el propietario de la misma y que debe introducir en el sistema cuando entra al mismo. Para proteger el secreto cuando se introduce esta palabra, el terminal se pone en blanco de tal manera que los caracteres introducidos no se vean reflejados en la pantalla. El usuario siempre puede cambiar su password a través de un comando del sistema operativo.

El segundo nivel de seguridad es el acceso a los ficheros. Cada fichero del sistema, incluyendo los directorios, tienen un usuario propietario y por tanto un grupo (cada usuario pertenece a un grupo). De esta manera, el acceso al mismo dependerá de si se es el propietario, de si se es del mismo grupo de usuarios, o de si se es de otro grupo cualquiera. Y es el propietario del fichero el que puede cambiar estos tres permisos de entrada.

A la vez existen tres tipos de acceso al fichero: para leerlo (r), para cambiarlo (w), o para ejecutarlo (x); (en el caso de un directorio hacer un `ls`, incluir un fichero dentro o hacer un `cd` a él) pudiéndose cambiar estos accesos a cada tipo de usuario. Teniendo en cuenta esto, existen en total 9 permisos que el propietario del fichero puede cambiar y que se pueden ver al hacer un listado de un directorio en modo largo, es decir, con la opción -l (ll):

```
-rw-rw-rw- 2 grupo 7 alumnos 3654 Nov 24 12:17 fichero
uuugggooo
```

El primer trío de la izquierda rwx corresponde a los permisos del usuario (uuu), cuando aparezca una letra significará que se tiene ese permiso y cuando aparezca un "-" que no, los otros dos tríos son los del grupo (ggg) y los de los otros usuarios (ooo). La letra que falta indicará como ya hemos visto el tipo de fichero, siendo el "-" el correspondiente a un fichero regular y "d" a un directorio.

Los permisos de un determinado fichero se pueden cambiar a través de la orden **`chmod`** (cambio de modo), que es un poco especial ya que no tiene opciones pero si los permisos que se quieren cambiar, y toma como argumentos los ficheros o directorios afectados. La forma de poner los permisos se compone de tres partes, por un lado a que tipo de usuarios afecta el cambio, simbolizados por tres letras: la u para el usuario, la g para el grupo y la o para los otros usuarios, por otro lado si se da o se quita el permiso, simbolizado por + ó -, y por último el tipo de permiso de lectura (r), escritura (w) y ejecución (x). Así, para quitar los permisos de lectura y escritura a todos menos el propietario del fichero *nominas* habría que hacer:

```
chmod go-rw nominas
```

Otra forma de utilizar este comando es dando como permiso un número octal de tres cifras, donde cada cifra es un tipo de usuario y donde cada dígito binario del octal es un tipo de permiso, por ejemplo 777 (sería 111 111 111) daría todos los permisos a todo el mundo. Por ejemplo, si quisiéramos hacer ejecutable una macro escrita en un fichero de texto podríamos hacerlo de dos formas:

```
chmod +x macro
chmod 755 macro
```

Hay que tener en cuenta que el inodo del fichero contiene 12 bits de control de acceso de los cuáles 9 corresponden a estos permisos. Los otros 3 bits son conocidos como setuid, setgid y sticky. El primero de ellos servirá para que un ejecutable tome como usuario el propietario del ejecutable y no al usuario que lo ha ejecutado. Por ejemplo el fichero /etc/passwd sólo puede ser editado por el root, sin embargo cuando ejecutamos passwd lo podemos cambiar, esto es debido a que passwd tiene activado el setuid. Lo mismo el segundo pero para el grupo. El tercer bit (el bit pegajoso) se usa también en ejecutable cuando queremos que permanezcan permanentemente en memoria (lo que querría cualquier virus). Hay que tener en cuenta que la manipulación de estos bits es peligrosa ya que ciertos agujeros de seguridad se basan en su manipulación.

Por defecto, todos los accesos están permitidos, a no ser que esté definido por el comando **`umask`** algún otro acceso. Este comando toma como argumento una máscara (código de inhibición) que es restado al 666 (acceso total sin ejecución ya que los ejecutables lo tienen activado por defecto) cada vez que se crea un fichero (no aplicable a directorios). Normalmente se ejecuta cuando arranca una shell `umask 022` lo que equivale a unos permisos 644 (`rw-r--r--`).

Otros comandos para manejarse en el sistema multiusuario son:

`who`	para decir que usuarios están en el sistema.
`chown usuario fichero`	para cambiar el propietario de un fichero.
`chgrp grupo fichero`	para cambiar el grupo propietario de un fichero.
`su usuario`	para cambiar temporalmente de usuario.
`login usuario`	para cambiar de usuario sin retorno.
`passwd [usuario]`	para cambiar el password de un usuario.
`write usuario`	para mandar un mensaje a otro usuario.
`mesg y, n`	para mandar un mensaje a otro usuario.

Los dos últimos sirven para comunicar usuarios que estén en ese momento en el sistema. Con el primero conseguiremos que le aparezca en la pantalla del otro usuario el mensaje que le hemos escrito (terminará en control-d), con el segundo evitaremos que nos aparezcan estos mensajes (`mesg n`), sobre todo cuando se edita, ya que puede resultar muy molesto. Existen otras maneras más sofisticadas de comunicar usuarios:

- El comando `finger usuario@maquina`. Nos permite conocer el estado de un usuario en un sistema determinado. Nos muestra el tiempo que lleva sin conectarse, su fichero `.plan`, y si tiene correo sin leer. Es peligroso, por lo que la mayoría de sistemas tiene deshabilitado este servicio.
- El comando `wall`. Permite mandar un mensaje (broadcast) de texto a todos los usuarios conectados.
- El comando `talk usuario@maquina`. Como el `write`, pero permite establecer una comunicación completa (similar al IRC).

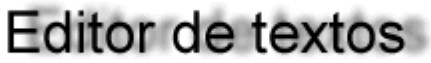
Hay una excepción en la utilización de los ficheros en los sistemas UNIX, ya que existe un encargado del mismo que tiene privilegios sobre el resto de los usuarios, es el llamado superusuario (en el que nos convertiremos después del curso). Por convención, el nombre del superusuario es `root` y como este nombre indica su directorio de trabajo es el directorio raíz y tendrá los suficientes privilegios como para acceder a cualquier fichero del sistema.

Por último, para compartir los recursos comunes del sistema entre varios usuarios, en concreto la impresora, lo que se utiliza es el mecanismo de *spooling*, de tal manera que cuando varios usuarios intentan acceder a la impresora a la vez, este mecanismo almacena los requerimientos e impone un orden de envío a la misma (se forma una cola de impresión).

Para utilizar el spooling existen varios comandos en el UNIX, los más usados son:

`lp ficheros`	para enviar a la impresora un fichero(s) (lpr en LINUX).
`lpstat`	para ver la cola de impresión (lpq en LINUX).
`cancel trabajo`	para cancelar un envío anterior (lprm en LINUX).

5.2. Editor vi



En todos los sistemas UNIX y formando parte del mismo como un comando más, existe una familia de editores que nos permitirán procesar texto. Esta familia procede del editor ed, que era el editor original del UNIX, pero que prácticamente ya no se usa.

La familia consta de tres miembros, por un lado el **`ex`** que es la versión moderna y mejorada del **`ed`**, el **`edit`** que es un subconjunto del `ex` para principiantes, y el **`vi`**, que a diferencia de los otros editores que eran de línea, es de pantalla (un editor de línea sólo trabaja con una línea, por lo que es muy incómodo de usar, un editor de pantalla nos permite trabajar en todo el texto, aunque sólo nos muestra una ventana del mismo). Este último además realiza llamadas al `ex` para realizar determinadas funciones.

En el sistema nos podremos encontar con otros editores como el emacs y tendremos la tentación de usarlo, ya que el `vi` es muy poco amigable o sinceramente desagradable de usar, pero tienen la desventaja de que es universal y por eso se escoge en este curso.

Funcionamiento

Todos los editores funcionan de igual manera, es decir, a través de un buffer en memoria. Cuando se lee un fichero desde el editor, este se carga en memoria en un buffer sobre el cual se hacen las modificaciones y una vez terminado el trabajo se vuelca el contenido de la memoria en el fichero. Esto tiene una ventaja y es que si se hace algo mal podemos dejar el trabajo sin salvar (pasar a disco), pero también tiene una gran desventaja, y es que si ocurre una fallo en la corriente o en el sistema, todo el trabajo realizado sobre la memoria se perderá, por eso es muy importante realizar volcados a disco cada cierto tiempo.

El editor `vi` también trabaja con un buffer en memoria que conviene salvar cada cierto tiempo. Para ello tendremos que ponerlo en modo comando y hacer ":w", pero ¿qué significa modo comando? El `vi` tiene dos modos de trabajo, uno donde introduciremos caracteres que formarán parte del texto y otro donde esos caracteres son acciones sobre el editor (como veremos más adelante), pero antes lo tendremos que arrancar, tomando normalmente como argumento el fichero que queremos editar:

```
$ vi fichero
```

si el fichero ya existe, aparecerá en pantalla el mismo, si es nuevo aparecerá una tilde "~" en las líneas que estén sin ocupar, en este caso la primera, donde se realizará la escritura del texto.

Cuando ejecutemos el `vi` nos lo encontraremos en modo comando, con lo cual esperará que le introduzcamos una orden (carácter) para actuar y pasar a modo texto. Para pasar a modo comando de nuevo se utiliza un carácter especial que es el de escape (Esc) y que se puede conseguir con la tecla de escape (arriba a la izquierda del teclado). No hay ningún problema en pulsar varias veces esa tecla, a parte de un pitido por cada par de pulsaciones. En la pantalla del `vi` y debido a su universalidad, sólo aparece el texto sobre el que se escribe, por lo que no sabemos (con algún indicativo en pantalla) en que modo estamos, por lo que se suele pulsar varias veces la tecla escape para garantizar el modo comando, esto unido a lo anterior, hace de una sesión de varios usuarios de `vi`, un concierto de pitidos.

Comandos

El `vi` tiene casi un centenar de comandos, de los que habitualmente sólo se utilizan una docena. Estos comandos se pueden dividir en varios grupos dependiendo de la función a realizar.

Movimiento del cursor

Existen más de 40 comandos de este tipo, pero los principales son cuatro, indicativos de las cuatro direcciones sobre las que nos podemos mover:

```
h : izquierda
l : derecha
j : abajo
k : arriba
```

la dirección a la que apuntan es la misma que la posición que ocupan en el teclado, restringiéndose su movimiento a la zona ocupada por el texto (si intentamos salir de esa zona nos avisará con un pitido). También se podrán utilizar las flechas del teclado.

Hay que incluir dentro de los comandos de movimiento la combinación de caracteres avance de línea (10) y retorno de carro (13), que se consiguen con la tecla return o enter. Esta servirá para dar a una línea por concluida y pasar a la siguiente en la primera posición.

Otros comandos de movimiento menos usados aparecen en la siguiente tabla, donde "^" significa que se debe pulsar antes del comando la tecla de control (un carácter de control es otro carácter más):

```
b  :   principio palabra (begin)
e  :   final de la palabra (end)
0  :   comienzo de línea
$  :   final de línea
^d :   12 líneas abajo (down)
^f :   24 líneas abajo (forward)
^u :   12 líneas arriba (up)
^b :   24 líneas arriba  (back)
nG :   ir a la línea n (go)
^g :   número de línea actual
```

Entradas a modo texto

Fundamentalmente hay tres comandos para escribir texto, el *a* para añadir a partir de la posición del cursor, el *i* para insertar delante del cursor y el *o*, *O* para añadir una nueva línea.

```
a :    añadir (append)
i :    insertar (insert)
o :    abrir debajo una línea
O :    abrir arriba una línea
```

Hay que recordar que en todo el entorno Unix las mayúsculas y las minúsculas son diferentes, por eso también existen las variantes A y I, la primera añadirá texto al final de la línea actual y la segunda insertará texto al comienzo de la misma. Si queremos escribir caracteres especiales antes tendremos que señalarlos con el carácter ^v.

Borrado y alteración

Una vez que hayamos escrito algo, ese texto se puede borrar o alterar pasando a modo comando. Para borrar se utilizan dos comandos fundamentales (y sus variantes) el x y el d. El x para borrar el carácter sobre el que estamos y el d (que se suele utilizar modificado) para borrar y pasar a una zona de memoria especial donde se puede recuperar lo borrado.

```
x :    borra un carácter
d :    borrar poniendo en el buffer
r :    reemplazar un carácter (replace)
R :    reemplazar varios caracteres
c :    borrar e insertar (change)
```

El *r* es el comando de hacer sustituciones de texto, si utilizamos la versión minúscula, sólo podremos sustituir un carácter, pasando de nuevo a modo comando, si utilizamos la mayúscula nos pondremos permanentemente en modo sustitución (hasta que con escape pasemos a modo comando), con lo que podremos sustituir varios caracteres. El comando *c* es una combinación del comando *d* y el *i*, primero borra lo indicado y después pasa a modo inserción para escribir de nuevo lo borrado.

Deshacer cambios

En caso de error, existe un comando para deshacer los cambios realizados, este es el *u* (undelete), del cual hay la versión minúscula para deshacer un sólo cambio o la mayúscula para deshacer todos los de la línea.

```
u : deshacer un cambio (undelete)
U : deshacer todos los cambios de la línea
```

Salida

Una vez que hayamos realizado la edición, normalmente querremos salvar nuestro trabajo en un fichero y abandonar la edición. Existen desde el vi varias formas, la más usual es la que parece a continuación:

```
ZZ : salva en el fichero y sale
```

Separación y unión de líneas

Hay dos comandos fundamentales para cortar y unir líneas, el primero y obvio es el retorno de carro, el segundo es J (join), observar que se utiliza la mayúscula.

```
<enter> : separar dos líneas
J : unir dos líneas (join)
```

Copiado y recuperación

A parte del fichero y del buffer temporal donde se realiza el trabajo de edición, hay una serie de buffers temporales auxiliares donde se colocan los últimos cambios o borrados que hayamos hecho, y hay una serie de comandos que nos permiten trabajar con estos buffers para hacer copias de texto.

Hay fundamentalmente tres comandos que nos permiten trabajar con estos buffers, por un lado está el comando *d* que ya hemos visto, y con el cual realizamos borrados de texto que colocamos temporalmente en un buffer. Y por otro lado están el comando *y* (yank) y el *p* (put), el *y* copia el texto seleccionado y lo coloca en un buffer, y el *p* hace lo contrario, coge la información del buffer y la coloca en el texto.

```
y : copiar dejando en el buffer (yank)
p : poner el buffer en el texto después (put)
P : poner el buffer en el texto antes
```

Modificadores de comandos

Hay ciertos comandos, sobre todo los del último apartado, que casi nunca se utilizan solos, sino que lo hacen con sus modificadores. Hay varios tipos:

- Repetir el comando: Esto hace que el comando actúe en toda la línea. Así si *d* borra, *dd* borra toda una línea.
- Poner un número delante del comando: Hace que ese comando se repita tantas veces como el número indicado. Así, *4dd* borrará 4 líneas.
- Alcances: Hay una serie de caracteres que después de un comando modifican el alcance del mismo. Estos alcances son:
    - `e :  hasta el final de la palabra.`
    - `w :  hasta el comienzo de la siguiente palabra.`
    - `b :  hasta el comienzo de la palabra.`
    - `$ :  hasta el final de la línea.`
    - `0 :  hasta el comienzo de la línea.`
    - `) :  hasta el comienzo de la siguiente frase.`
    - `) :  hasta el comienzo de la frase.`
    - `} :  hasta el final del párrafo.`
    - `{ :  hasta el comienzo del párrafo.`

Así, *de* borrará hasta el final de la palabra donde esté situado el cursor, o *d$* borrará hasta el final de la línea.

Cualquiera de estos modificadores puede utilizarse conjuntamente con otros, de esta manera *2dw* borrará dos palabras.

Hemos dicho que con estos comandos se utilizan unos buffers especiales donde se colocan los cambios que estamos realizando, en vi no sólo existe un buffer para realizarlos, sino que hay varios que se nombran con los números y letras del alfabeto, aunque lo normal es utilizar el buffer por defecto.

Para utilizarlos se pone delante del comando unas comillas, el código del buffer, y por último el comando (incluso con otros modificadores).

Así, *"c4dd* borra cuatro líneas y las coloca en el buffer c. Para recuperar las cuatro líneas borradas bastaría hacer: *"cp*. Recordad que si no nos interesa lo recuperado, siempre podemos volver atrás con el comando *u*. Esta forma de utilizarlo puede servirnos para ir inspeccionando los buffers uno a uno.

Llamadas al editor de líneas ex

El vi está capacitado para utilizar los comandos del editor de líneas ex, de tal manera que tenemos otro conjunto de comandos a los cuales se accede en modo comando anteponiéndoles el carácter ":". De echo se puede pasar de un editor a otro utilizando el comando Q (pasa al ex) y el : (pasa al vi). En muchos casos estos comandos son tan importantes como los originales del vi.

Además se pueden utilizar varios comandos juntos como: *:r !who* que ejecuta el comando *who*, lo lee, y lo coloca en el fichero de edición.

Cabe destacar que los comandos del ex admiten delante de ellos dos números que son el rango de filas donde actúan, así, si escribimos *:3,5w fic*, estamos haciendo que se escriban las líneas 3, 4 y 5 en el fichero fic. Lo mismo se puede hacer con *:co* y *:m*, para *:r* sólo hace falta poner la línea detrás de la cual se colocará el fichero leído. Para indicar todas las líneas del texto se utiliza la *g*, que significa que se hará globalmente, esta opción es muy importante en los comandos que se verán posteriormente.

Los comandos de búsqueda y sustitución son muy utilizados. Estos se hacen en combinación con el ex. Los dos comandos del vi para realizar estas tareas son el */* y el *?*. El primero busca un patrón hacia adelante y el segundo hacia atrás. Además tenemos el comando *n* que repite la búsqueda. A continuación aparece una tabla de los más utilizados:

Comando	Descripción	Ejemplo

:wq	Salva el texto en un fichero y sale del editor	:wq
:w fichero	Escribe el texto en un fichero	:w pepe
:q!	Sale del editor sin salvar el texto	:q!
:r fichero	Lee un fichero y lo coloca en el cursor	:r pepe
:! comando	Ejecuta un comando de la shell desde el editor	:! who
:co	Copia unas determinadas filas	:3,5co7
:m	Mueve unas determinadas filas	:4,8m9

La estructura de un comando de este tipo es la siguiente:

```
:dirección/orden/parámetros
```

donde dirección consta de las líneas donde se debe realizar la acción y el patrón (secuencia de caracteres) a buscar, orden será lo que se quiere hacer después de encontrado el patrón, y parámetros normalmente el número de veces que se quiere realizar la operación. Veamos unos ejemplos:

```
:1,7/can /s//perro /          Se substituye de la 1 a la 7 can por
perro una vez.
:g/can /s//perro /g           En todo el fichero y todas las veces.
:/patrón/s/patrón/nuevo/      Busca patrón y sustituye por nuevo.
:g/p/s/p/n/                   Busca p y lo sustituye por n globalmente.
```

En el primer ejemplo la dirección serían las líneas 1 a 7 y el patrón *can* , una vez buscado se dice con la orden *s//* que se sustituya ese patrón (habría que repetirlo pero por defecto es el mismo, por lo tanto se escribe *//*) por el nuevo patrón *perro* . Si se coloca como parámetro la *g* se realizará la sustitución en todos los encuentros.

Por último hay que decir que desde el vi se pueden editar varios ficheros a la vez (uno detrás de otro), para ello se deberá ejecutar el vi con varios ficheros contiguos: vi f1 f2 f3. Los comandos son:

```
:e fichero          Editar otro fichero.
:n                  Pasar al siguiente fichero en edición múltiple.
```

Opciones del editor

Por último y brevemente, hay que decir que el vi se puede personalizar y ponerle otras opciones como los espacios por tabulador, o el número de columnas de texto. Para ello se utiliza el comando del ex `:set`. Las opciones más comunes son:

```
ai          identar.
ic          ignorar mayúsculas.
nu          numerar las líneas.
redraw      reescribir.
sw          número de espacios en ai.
ts          número de espacios en tab.
w           número de líneas de texto.
all         ver todas las opciones.
```

Un ejemplo de utilización de este comando, y quizás el más útil, sería para poner delante del texto como referencia en número de la línea:

```
:set nu
```

5.3. Búsqueda

Existen una serie de comandos muy utilizados en la administración de un sistema, estos comandos los repasaremos en este tema. Son comandos sencillos de línea pero que se utilizan frecuentemente para localizar información en el sistema:

El primero de ellos es el comando **grep** que se utiliza asociado a una canalización para localizar un patrón entre "algo" que produce otro comando, aunque su versión completa aparece a continuación:

```
grep [opciones] PATRÓN [ARCHIVO...]
grep [opciones] [-e PATRÓN | -f FICHERO] [ARCHIVO...]
```

que buscará PATRÓN (normalmente encerrado entre apóstrofes) en los ARCHIVOS y producirá una línea en la pantalla por cada coincidencia. El patrón se puede dar también con la opción -e útil cuando éste empieza por "-". Si el patrón está escrito en un fichero utilizaremos la opción -f. Otras opciones interesantes son:

`-c` En lugar de imprimir por salida estándar las líneas coincidentes, imprime la cantidad de líneas que coincidieron en cada archivo.
`-H` Imprimir el nombre del archivo con cada coincidencia.
`-r` Buscar recursivamente dentro de todos los subdirectorios del directorio actual.
`-i` Ignora si son mayúsculas o minúsculas.

El patrón de búsqueda normalmente es una palabra o una parte de una palabra, también se pueden utilizar expresiones regulares, para realizar búsquedas más flexibles.

Ejemplos de uso de este comando son:

```
ps -ef | grep rafa        Pintará en pantalla en formato largo los procesos de rafa
grep 'pepe' fichero       Pintará las líneas con pepe en el fichero
```

Otro comando de búsqueda es **find**. Se utiliza este comando para buscar archivos dentro de una jerarquía de directorios. La búsqueda, como veremos más adelante, se puede realizar mediante varios criterios. La sintaxis de este comando es:

```
find [camino...] [expresión]
```

El camino será donde queremos que empieze la búsqueda y la expresión se conforma de:

- opciones,
- pruebas (test) y
- acciones

y empieza con algunos de los siguientes caracteres: `'-', '(', ')', ',', o '!'`. La acción por defecto es la de escribir los resultados por pantalla. Las opciones afectarán normalmente al test (por ejemplo para medir tiempos, escoger el comienzo del día (`-daystart`) o utilizar las últimas 24 horas) y los test (pruebas) más frecuentes son:

```
-atime n       Ficheros abiertos exactamente hace n días.
-amin n        Ficheros accedidos hace n minutos.
-mtime n       Modificados hace n días.
-links n       Ficheros con n links.
-newer file    Modificados después del fichero file.
-size n        Con tamaño exactamente n bloques (bloques de 512
bytes).
```

```
-type c         Tipo de fichero f=texto, d=directorio, etc.
-name 'patron' Con nombre patron (se pueden usar comodines).
-perm p         Con permisos p (en octal).
-user UID       Con propietario numérico UID.
-mtime +7       Modificado hace más de 7 días.
-size -100      Menores de 100 bolques.
```

Aunque existen muchas más que se pueden comprobar haciendo un `man find`. Estos test también se pueden combinar de forma lógica con los operadores AND (-a, -and), OR (-o, -or) y NOT (!, -not), donde el AND es el que se toma por defecto.

```
-atime +60 -mtime +120    abiertos hace más de 60 días y
modificados hace más de 120.
-atime +7 -o -mtime -10   abiertos hace más de 7 días o
modificados hace menos de 10.
```

En cuanto a las acciones se puede decir que escriba los resultados por pantalla o que los muestre de forma extendida o que ejecute un comando, esto último se consigue con la acción `-exec` (hay una versión que pregunta que es `-ok`) que tiene que estar terminada con los dos caracteres "`\;`" y donde "`{}`" sustituyen a los ficheros encontrados.

```
find -name 'pep*' -exec cat {} \; busca los ficheros que
empiecen por pep y los presenta en pantalla
```

Existen dos comandos más de búsqueda, el primero de ellos es **`whereis`** que localiza donde está un determinado comando, tanto su código binario (`-b`), su fuentes (`-s`) o la página del manual (`-m`).

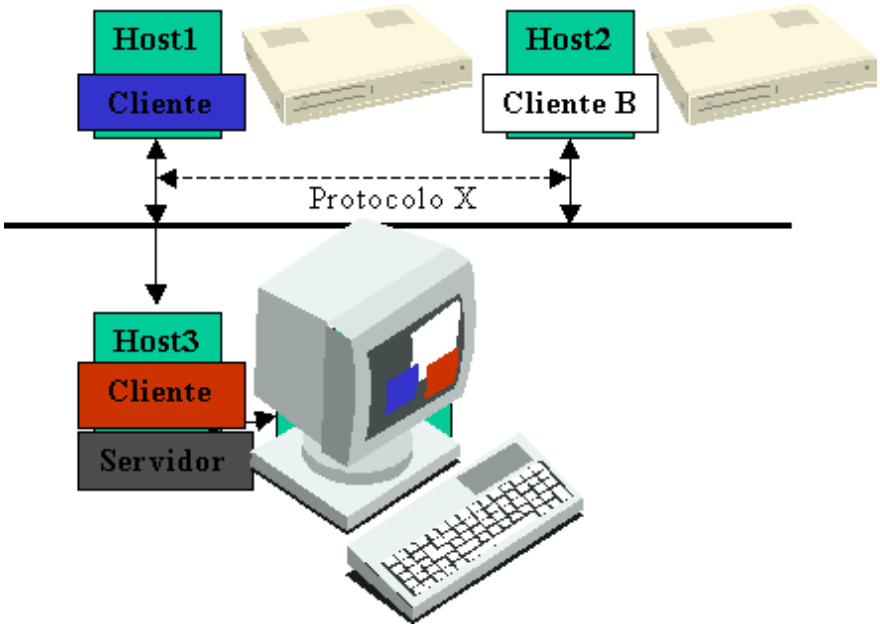
```
whereis -b whereis
```

El último comando que vamos a ver es **`locate`**, que es un comando de búsqueda de archivos, bastante parecido al comando `find`. La diferencia de `locate` es que la búsqueda la hace en una base de datos indexada para aumentar significativamente la velocidad de respuesta. `locate` realmente no busca en el disco del sistema, sino en un archivo con la lista de todos los archivos que existen en el GNU/Linux.

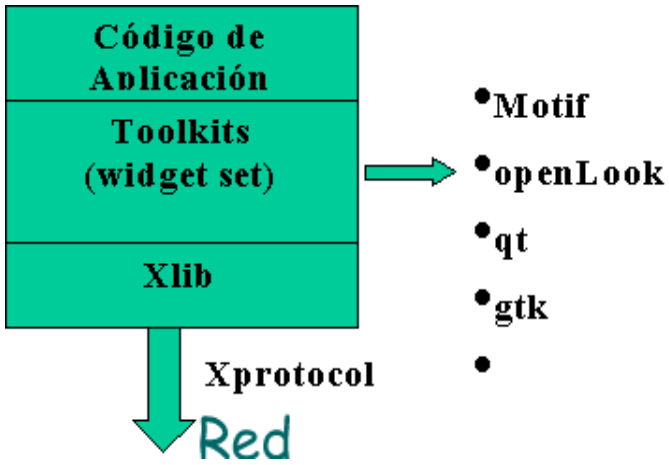
5.4. Entorno Gráfico

A pesar de que en Linux disponemos de varias consolas de texto virtuales a través de la pulsación conjunta de las teclas de función y de alt y control, esto en entornos modernos no es suficiente, por eso el MIT (Robert Sheifler) dentro del proyecto ATHENA (con la colaboración de DEC) introdujo un soporte de entorno gráfico de usuarios (GUI) llamado X Window (proviene de un desarrollo anterior conocido como sistema W) que actualmente es controlado por el X-Consortium de OSF.

Una peculiaridad fundamental del entorno X Window es que está diseñado para ofrecer servicios a través de red (la información se transmite con el protocolo X) siguiendo una arquitectura de tipo cliente/servidor (en la figura se ven tres computadores, los tres son clientes X, pero sólo uno de ellos es el servidor X al que los tres acceden) por lo tanto un cliente X puede ejecutar una sesión gráfica usando un servidor X remoto, por lo tanto es independiente de la localización y de la plataforma (hardware y Sistema Operativo).



Si un programador quiere crear una aplicación X deberá usar la librería Xlib que nos permite abstraernos del proticolo X y centrarnos en la aplicación (creación de ventanas y primitivas de dibujo). Aún así está librería es incómoda de usar por lo que por encima de ella de han creado herramientas que se conocen como Toolkit (creación de distintos tipos de ventanas, botones, desplegables, etc.) y que permiten un uso más amigable con programación orientada a objetos, las más importantes son X Toolkit y OSF / Motif Toolkit, quizás la más conocida, aunque es comercial.



Existen multitud de aplicaciones (clientes) X:

- `xterm`. El emulador de terminal usado por la mayoría de las aplicaciones en modo texto dentro de X.
- `xdm`. El gestor de sesiones, maneja las entradas al sistema (los login).
- `xclock`. Un reloj simple
- xcalc. Una calculadora.
- `xman`. Un lector de páginas de manual para X.
- xload. Un Indicador de carga del sistema.
- xedit. Un editor de textos.
- xset. Cliente para cambiar comportamiento del sistema X.
- xmodmap. Cliente para cambiar el mapa asignado al teclado.
- xlsfonts y xfontsel para ver y escoger las fuentes del sistema.
- xeyes. Aparecen unos ojos vigilando el puntero del ratón.
- xkill. Para terminar un cliente X.

y muchos más. Pero el cliente X más importante es el gestor de ventanas (wm). Es el que hace la comunicación entre el sistema y el usuario y el que colocará las ventanas X en su lugar, las redimensionará, permitirá su iconización y movimiento así como el aspecto de sus marcos. Además manejará todos los menús e iconos.

El sistema X en los Linux es el Xfree de tipo GPL, en concreto la versión 11 y la revisión 6 (X11R6). Este sistema proporciona todas las herramientas básicas para construir aplicaciones cliente X.

La distribución estándar XFree86 incluye twm, el clásico controlador de ventanas del MIT, aunque hay disponibles controladores mucho más avanzados como Open Look Virtual Window Manager (olvwm) y un controlador de ventanas muy popular entre los usuarios de Linux, el `fvwm`. Es un pequeño controlador que requiere menos de la mitad de la memoria usada por twm. Proporciona aspecto de 3 D para las ventanas, así como un escritorio virtual si el usuario desplaza el ratón al borde de la pantalla (la pantalla entera es desplazada como fuese mucho más grande de lo que realmente es). `fvwm` es altamente configurable y permite acceso a todas las funciones tanto desde el teclado como desde el ratón.

Un paso más adelante sobre el gestor de ventanas es construir un escritorio (desktop) completo, los dos más utilizadas actualmente son la KDE y la Gnome. El primero de ellos usa el toolkit Qt y el segundo el GTK.

Arranque

Antes de usar el sistema X (XFree86), deberemos configurarlo, esto se hace a través de un fichero de texto: `/etc/X11/XF86Config`, pero normalmente antes de proceder a su edición es más sencillo utilizar la herramienta `Xconfigurator` que a través de una serie de preguntas lo hará por nosotros. Respuestas típicas que deberemos conocer es el tipo de driver de pantalla que utilizamos (tarjeta gráfica) o tipo de monitor (frecuencias horizontales y verticales), el ratón se tomará del fichero `/etc/sysconfig/mouse`. Si estamos usando un servidor, también deberemos dar el tipo de servidor adecuado localizado en `/usr/X11R6/bin/XF86_<server>`.

Una vez configurado el Xfree86 procederemos a arrancar la sesión gráfica utilizando el comando **`startx`** o alguna de sus variantes **`X11`**, o **`xstart`** (normalmente hace de interfase entre el usuario y el comando `xinit` que es el que realmente arranca el entorno X, es decir, el servidor y los clientes definidos) que leerá la macro `/etc/X11/xinit/xinitrc` donde se pueden redefinir parámetros de teclado (`xmodmap`), clientes para arrancar (xterm,mail,etc..), entorno de ventanas, etc.

Si quisieramos arrancar sólo el servidor X deberíamos utilizar el comando **`x`**. Cuando queramos abandonar la sesión gráfica (volver a la de texto) lo haremos con la combinación: CRTL+ALT+BACKSPACE.

También se puede arrancar el sistema con el administrador de pantalla (X display manager) `xdm` (al haber escogido el nivel 5 de arranque), en este caso el fichero sería `/etc/X11/xdm/Xsession` que es similar a `xinitrc`. El xdm es una parte suplementaria de sistema X que se encarga del manejo de sesiones y haría las veces - en términos de texto - de los demonios getty / login. Por lo tanto se encargará de los ingresos en el sistema y arrancar un gestor de ventanas que puede ser también un escritorio. En este caso abandonaremos la sesión saliendo de la sesión (o abreviando con CRTL+ALT+BACKSPACE). También podemos pasarnos a 6 consolas de texto con CTRL+ALT+F1...F6 y regresa al servidor X con CTRL+ALT+F7.

Una vez arrancado cada servidor será visto desde el punto de vista del usuario como un nombre DISPLAY almacenado en una variable de entorno del mismo nombre de la forma a:

```
hostname:displaynumber.screennumber
```

donde hostname es el nombre de la máquina que tiene la pantalla, displaynumber es el número de terminal dentro de la máquina (si es un PC o una workstation normalmente sólo habrá uno empezando desde cero) y screennumber es el número de pantalla (en el caso hipotético y extraño de que varias pantallas compartan un teclado).

Sesiones remotas

En este entorno y debido a que el sistema X es independiente de la plataforma (hardware y software) podemos el servidor arrancado en una máquina y el cliente en otra, y bastaría con que indicásemos donde queremos producir la salida utilizando la nomenclatura anterior (definir la variable de shell `DISPLAY`).

Por ejemplo si estamos en el computador alfa donde tenemos arrancado el servidor X, podríamos arrancar clientes en otro computador beta y que la salida de estos clientes fuera en el servidor alfa. Para ello deberíamos autorizar el uso a través del comando **`xhost`** de la forma (o denegarle este permiso a otro hipotético gamma):

```
$ xhost beta                (xhost -gamma)
```

La forma de arrancar los clientes sería (en este caso los ojos):

```
$ xeyes -display alfa:0.0&
```

Y si no quisieramos poner siempre el display definiríamos la variable DISPLAY de acuerdo a la shell que estemos usando:

```
% setenv DISPLAY alfa:0
$ DISPLAY=alfa:0; export DISPLAY
```

El arranque de sesiones remotas X puede ser un agujero de seguridad por eso se utilizan servicios de autentificación complementarios (basados en "cookies") como `xauth`.