El shell de linux: Comando find

La sintaxis es muy simple:

```
find [ruta] [expresión_de_búsqueda] [acción]
```

La [ruta] es cualquier directorio o path que se quiera indicar y desde donde inicia la búsqueda, ejemplos pueden ser "/etc", "/home/sergio", "/", "." si no se indica una ruta se toma en cuenta entonces el directorio donde se este actualmente, es decir el directorio de trabajo actual, que es lo mismo que indicar punto ".". De hecho es posible indicar más de un directorio de búsqueda como se verá más adelante en un ejemplo.

La [expresión_de_búsqueda] es una o más opciones que puede devolver la búsqueda a realizar en si o acciones a realizar sobre la búsqueda, si no se indica ninguna expresión de búsqueda se aplica por defecto la opción *-print* que muestra el resultado de la búsqueda.

La [acción] es cualquier comando de Linux invocado a ejecutarse sobre cada archivo o directorio encontrado con la [expresión_de_búsqueda].

Los tres argumentos anterior son enteramente opcionales

Búsquedas básicas

El siguiente ejemplo busca todos los archivos cuyo nombre sea "reporte" desde la raíz:

```
find / -name reporte
find / -i name Reporte  (lo mismo, pero sin tomar en cuenta mayúsculas y minúscul
as)
```

El uso de expresiones regulares en lo que se busca es válido:

```
find / -name "[0-9]*"     (todo lo que empiece con un dígito)
find / -name "[Mm]*"      (todo lo que empiece con un la letra M o m)
find / -name "[a-m]*.txt" (todo lo que empiece entre a y m y termine en ".txt")
```

Busca bajo /home todos los archivos que pertenezcan al usuario mario

```
find /home -user mario
(lo mismo y que contengan con "enero" como en reporte_enero2011)
find /home -user mario -name "*enero*"
```

No estás limitado a un solo directorio, indica más de uno a buscar antes de las expresiones:

```
find /etc /usr /var -group admin
(busca en tres directorios todos los archivos o
subdirectorios que pertenezcan al grupo 'admin')
```

Otro ejemplo: Imaginemos que quiero listar los directorios y archivos que hay en el directorio actual:

```
$ find . -maxdepth 1 -type d    (Directorios)
$ find . -maxdepth 1 -type f    (Archivos)
```

Búsquedas a través del tiempo

Varias opciones aceptan argumentos numéricos, estos pueden ser indicados de tres maneras posibles:

```
+n    busca valores mayor que n
-n    busca valores menor que n
n     busca exactamente el valor n
```

Buscar todos los archivos que hayan cambiado en los últimos 30 minutos:

```
find / -mmin -30 -type f


los modificados exactamente hace 30 minutos:
find / -mmin 30 -type f
```

O si deseas buscar en un rango específico de minutos, con este ejemplo buscarías todos los directorios que hayan cambiado hace más de 10 minutos (+10) y menos de 30 (-30)

```
find / -mmin +10 -mmin -30 -type d


aunque lo anterior sería mas exacto decir los modificados hace 11 minutos o más
y 29 minutos o menos, ya que como se vio anteriormente +n y -n indican
"mayor que" y "menor que", el ejemplo correcto sería entonces:
find / -mmin +9 -mmin -31 -type d
```

`find` ofrece varias opciones de búsqueda por tiempo, pero las principales son: *-amin, -atime, -cmin, ctime, -mmin y -mtime*. "min" es para periodos de minutos y "time" para periodos de 24 horas.

Los que empiezan con "a" (access) indica el tiempo en que fue accedido (leido) por última vez un archivo. Los que empiezan con "c" (change) indica el tiempo que cambió por última vez el status de un archivo, por ejemplo sus permisos. Los que empiezan con "m" (modify) indica el tiempo en que fue modificado (escrito) por última vez un archivo.

Una consideración a tener con las búsquedas *-atime, -ctime y -mtime* es que el tiempo se mide en periodos de 24 horas y estos son siempre truncados, con ejemplos es más claro:

```
find . -mtime 0  (busca archivos modificados entre ahora y hace un dia)
find . -mtime -1  (busca archivos modificados hace menos de un dia)
find . -atime 1  (busca archivos accedidos entre hace 24 y 48 horas)
```

```
find . -ctime +1  (busca archivos cuyo status haya cambiado hace más de 48 horas)
```

Comparaciones con *-and, -or y -not*

`find` también incluye operadores booleanos que la hace una herramienta aun más últil:

```
find /home -name 'ventas*' -and -mmin 120

find /home -name 'reporte[_-]*' -not -user sergio

find /home -iname '*enero*' -or -group gerentes
```

El primer ejemplo busca todos los archivos que comiencen con 'ventas' Y que hayan sido modificados o cambiados en las últims dos horas (120 minutos).

El segundo ejemplo busca todos los archivos que comiencen con 'reporte' y después siga un _ o un - y que NO pertenezcan al usuario sergio.

El tercer ejemplo busca todos los archivos que contengan la palabra enero, Enero, ENERO, etc. (sin importar si lleva mayúsculas o minúsculas) O cualquier otro archivo que encuentre que pertenezca al grupo 'gerentes'.

Estas opciones de booleanos tienen su correspondiente abreviatura:
-and se puede indicar también como *-a*
-or se puede indicar también como *-o*
-not se puede indicar también como *!*

El tamaño si importa

Una de las actividades básicas de un administrador de sistemas Linux es monitorear el tamaño de archivos, sobre todo de usuarios. Con `find` es muy fácil realizar búsquedas por tamaño, se indica con la opción *-size*, se aplican las mismas reglas para argumentos númericos (+n -n n).

```
find /var/log -size +15000k -name "*.jpg" (busca archivos mayores a 15 megas del tipo jpg)

find $HOME -800c  (busca en tu home todos los archivos menores a 800 bytes (799 realmente))

(archivos de tamaño comprendidos entre 1mb y 10mb)

find . -size +1000k -and -size -10000k
```

Se admiten cuatro parámetros después del número en *-size*:

c = bytes
w = 2 byte words
k = kilobytes
b = 512-byte bloques

Para buscar archivos vacios puedes entonces hacer lo siguiente:

```
find . -size 0c

(Aunque la opción -empty hace lo mismo más eficientemente)

find . -empty
```

Cualquiera de los ejemplos anteriores dará un aburrido listado de los archivos y sus rutas. Si lo que quieres es realizar una acción (ejecutar un comando) sobre estos usa entonces la opción entonces *-exec*.

A escena *-exec*, el poder aumenta

-exec permite ejecutar acciones sobre el resultado de cada línea o archivo devuelto por `find`, o en otras palabras permite incorporar comandos externos para ejecutar sobre cada resultado devuelto. Muy interesante. Asi por ejemplo, si queremos buscar todos los archivos mayores a 3 megas en /var y además mostrar su salida en formato `ls`, podemos hacer lo siguiente:

```
find /var -size +3000k -exec ls -lh {} \;
```

Después de `ls -lh` que nos devuelve una salida formateada de `ls` se indica la cadena '{}' que se sustituye por cada salida de find.

No hay límite para lo que se puede lograr, asi por ejemplo, borrar todo lo mayor a un mega en /tmp.

```
find /tmp -size +1024k -exec rm -f {} \;
```

Por cierto si usas la versión GNU de `find` (y creo que todos los que usamos Linux la tenemos, compruébalo con `find --version`)), lo anterior también funciona directamente con la opción *-delete*:

```
find /tmp -size +3000k -delete      (lo mismo que usar -exec con rm)
```

El shell de linux: Comando sort

sort es uno de los comandos que utilizamos mucho a la hora de realizar scripts.

Nos permite ordenar los registros o líneas de uno o más archivos.
La ordenación se puede hacer por el primer carácter, por el primer campo de la línea o por un campo distinto al primero en el caso de ficheros estructurados.

Podemos ordenar el contenido de un fichero de la siguiente manera:

sort fichero
Se realizaría la ordenación y el resultado se mostraría por pantalla. Así que, si lo que queremos es obtener el resultado de la ordenación en un fichero, haríamos:

sort fichero > ficheroordenado
Si lo que queremos es ordenar varios ficheros y añadir el resultado a otro, podemos indicar varios ficheros en la línea de entrada:

sort fichero1 fichero2 > fichero3
Y si lo que queremos es ordenar un fichero y dejar el resultado de la ordenación en el mismo fichero, podemos hacerlo con el parámetro -o (output):
sort -o f1 f1

Veamos una lista de los parámetros que pueden resultarnos más útiles a la hora de usar este comando:

- -f : Este parámetro nos sirve para indicar que las mayúsculas y las minúsculas se van a tratar de forma diferente y que por tanto se va a seguir un ordenamiento alfabético.
- -n : Este parámetro nos sirve para ordenar los campos numéricos por su valor numérico.
- -r : Nos permite realizar una ordenación inversa, es decir, de mayor a menor.
- +número : Este parámetro nos sirve para indicar la columna o campo por el que vamos hacer la ordenación. Esta sintaxis está en desuso y se va a eliminar. En su lugar se utilizará la siguiente sintaxis:
- -k numero : De este modo especificaremos por qué columna o campo vamos a realizar la ordenación en las versiones más recientes de Linux.
- --field-separator= separador. Normalmente, se usa como delimitador de campos el espacio en blanco. Podemos utilizar el parámetro --field-separator para indicar que vamos a usar otro delimitador de campo cualquiera. Ej: --field-separator=, La opción abreviada de --field-separator es -t.
- -u : Nos permite suprimir todas las líneas repetidas después de realizar la ordenación.

Y algunos ejemplos con dichos parámetros:

Obtener un listado de los ficheros del directorio actual, ordenado por tamaño de archivo:
$ ls -l | sort +4n
Obtener un listado de los ficheros del directorio actual, ordenado de mayor a menor por tamaño de archivo:
$ ls -l | sort -r +4n
Obtener un listado de los ficheros del directorio actual, ordenado por nombre del archivo:
$ ls -l | sort +7
Ordenar un fichero eliminando las líneas repetidas:
$ sort -u fichero
Ordenar un fichero pen el que los campos están separados por comas, por el campo número 3:
$ sort -t, +3

Veamos un ejemplo en el que ordenemos usando la sintaxis actual para ordenar por columnas: Imaginemos que queremos ver un listado de usuarios del fichero /etc/passwd ordenado por uid:
$ cat /etc/passwd| sort -t":" -k3n

Con -k3 le indicamos a sort que queremos ordenar por la columna 3. Y, al añadir la opción -n le indicamos que ordene por orden numérico.
Y si quisiéramos realizar la ordenación de mayor a menor:
$ cat /etc/passwd| sort -t":" -k3nr

Un ejemplo que uso mucho, cuando quiero eliminar las líneas repetidas de un archivo y dejar el contenido en el mismo archivo:
$ sort -o fichero -u fichero

El shell de linux: Comando stat

El comando stat nos muestra una información muy completa acerca de archivos o sistemas de ficheros. Como todos los comandos, tiene muchas opciones, así que pondré sólo las que más he usado. Para consultar el resto, podéis usar el man.

Veamos una salida de ejemplo, cuando ejecutamos stat pasándole como parámetro un fichero:
$ stat comprime.sh

File: `comprime.sh'
Size: 262 Blocks: 8 IO Block: 4096 archivo regular
Device: 804h/2052d Inode: 1785294 Links: 1
Access: (0644/-rw-r--r--) Uid: (1000/enam0000) Gid: (100/ users)
Access: 2008-04-03 18:45:29.000000000 +0200
Modify: 2008-01-30 17:56:08.000000000 +0100
Change: 2008-03-04 23:32:02.000000000 +0100

Fijáos si la información es completa: Este comando nos reporta el nombre del archivo, su tamaño, los bloques que ocupa, el tipo de archivo (regular), información física de donde se encuentra (dispositivo/inode), los permisos estandar, los dueños del archivo y las tres marcas de tiempo Unix.

Y si lo que queremos hacer es obtener tan sólo un dato concreto, podemos hacerlo de la siguiente manera:

$ stat -c %u fichero Nos muestra el User ID del propietario del fichero.

$ stat -c %U fichero Nos muestra el nombre de usuario del propietario del fichero.

$ stat -c %g fichero Nos muestra el Group ID del propietario del fichero.

$ stat -c %G fichero Nos muestra el nombre del grupo al que pertenece propietario del fichero.

$ stat -c %n fichero Nos muestra el nombre del fichero.

$ stat -c %F fichero Nos muestra el tipo del fichero.

$ stat -c %A fichero Nos muestra los derechos de acceso.

$ stat -c %a fichero Nos muestra los derechos de acceso en formato octal.

$ stat -c %x fichero Nos muestra la fecha y hora del último acceso.

$ stat -c %y fichero Nos muestra la fecha y hora de la última modificación.

$ stat -c %z fichero Nos muestra la fecha y hora del último cambio.

El shell de linux: Comando grep

El comando **grep** nos permite buscar, dentro de los archivos, las líneas que concuerdan con un patrón. Bueno, si no especificamos ningún nombre de archivo, tomará la entrada estándar, con lo que podemos encadenarlo con otros filtros.

Por defecto, grep imprime las líneas encontradas en la salida estándar. Es decir, que podemos verlo directamente la pantalla, o redireccionar la salida estándar a un archivo.

Como tiene muchísimas opciones, vamos a ver tan sólo las más usadas:

-c En lugar de imprimir las líneas que coinciden, muestra el número de líneas que coinciden.
-e PATRON nos permite especificar varios patrones de búsqueda o proteger aquellos patrones de búsqueda que comienzan con el signo -.
-r busca recursivamente dentro de todos los subdirectorios del directorio actual.
-v nos muestra las líneas que no coinciden con el patrón buscado.
-i ignora la distinción entre mayúsculas y minúsculas.
-n Numera las líneas en la salida.
-E nos permite usar expresiones regulares. Equivalente a usar **egrep**.
-o le indica a grep que nos muestre sólo la parte de la línea que coincide con el patrón.
-f ARCHIVO extrae los patrones del archivo que especifiquemos. Los patrones del archivo deben ir uno por línea.
-H nos imprime el nombre del archivo con cada coincidencia.

Veamos algunos ejemplos:

- Buscar todas las líneas que contengan palabras que comiencen por **a** en un archivo:
$ grep '\<a.*\>' archivo

Otra forma de buscar, sería:
$ cat archivo | grep "\<a.*\>"

- Mostrar por pantalla, las líneas que contienen comentarios en el archivo /boot/grub/menu.lst:
$ grep "#" /boot/grub/menu.lst

- Enviar a un fichero las líneas del archivo /boot/grub/menu.lst que no son comentarios:
$ grep -v "#" /boot/grub/menu.lst

- Contar el número de interfaces de red que tenemos definidos en el fichero /etc/network/interfaces:
$ grep -c "iface" /etc/network/interfaces

- Mostrar las líneas de un fichero que contienen la palabra BADAJOZ o HUELVA:
$ grep -e "BADAJOZ" -e "HUELVA" archivo

- Mostrar las líneas de un fichero que contienen la palabra BADAJOZ o HUELVA, numerando las líneas de salida:
$ grep -n -e "BADAJOZ" -e "HUELVA" archivo

- Mostrar los ficheros que contienen la palabra TOLEDO en el directorio actual y todos sus subdirectorios:
$ grep -r "TOLEDO" *

Veamos algunos ejemplos con expresiones regulares:

- Obtener la dirección MAC de la interfaz de red eth0 de nuestra máquina:
$ ifconfig eth0 | grep -oiE '([0-9A-F]{2}:){5}[0-9A-F]{2}'

Sacamos la dirección MAC de la interfaz eth0 de nuestra máquina haciendo un:
ifconfig eth0

Y aplicando el filtro grep:
grep -oiE '([0-9A-F]{2}:){5}[0-9A-F]{2}'

Las opciones que he usado en grep son:

-o Indica que la salida del comando debe contener sólo el texto que coincide con el patrón, en lugar de toda la

línea, como es lo habitual.
-i Lo he usado para que ignore la distinción entre mayúsculas y minúsculas.
-E Indica que vamos a usar una expresión regular extendida.

En cuanto a la expresión regular, podemos dividirla en dos partes:
([0-9A-F]{2}:){5} Buscamos 5 conjuntos de 2 carateres seguidos de dos puntos
[0-9A-F]{2} seguido por un conjunto de dos caracteres.

Como las direcciones MAC se representan en hexadecimal, los caracteres que buscamos son los números del 0 al 9 y las letras desde la A a la F.

- Extraer la lista de direcciones de correo electrónico de un archivo:
grep -Eio '[a-z0-9._-]+@[a-z0-9.-]+[a-z]{2,4}' fichero.txt

Utilizo las mismas opciones que en el caso anterior:

-o Indica que la salida del comando debe contener sólo el texto que coincide con el patrón, en lugar de toda la línea, como es lo habitual.
-i Lo he usado para que ignore la distinción entre mayúsculas y minúsculas.
-E Indica que vamos a usar una expresión regular extendida.

Analicemos ahora la expresión regular:
[a-z0-9._-]+@[a-z0-9.-]+[a-z]{2,4}

Al igual que antes, la vamos dividiendo en partes:
[a-z0-9._-]+ Una combinación de letras, números, y/o los símbolos . _ y - de uno o más caracteres
@ seguido de una arroba
[a-z0-9.-]+ seguido de una cadena de letras, números y/o los símbolos . y -
[a-z]{2,4} seguido de una cadena de entre dos y cuatro caracteres.

- Obtener la dirección IP de la interfaz de red eth1 de nuestra máquina:
$ ifconfig eth1 | grep -oiE '([0-9]{1,3}\.){3}[0-9]{1,3}' | grep -v 255

En el ejemplo anterior, hemos tomado la información que nos ofrece ifconfig:
ifconfig eth1

Hemos filtrado dicha información con el comando grep, obteniendo todas las direcciones IP que aparecen:
grep -oiE '([0-9]{1,3}\.){3}[0-9]{1,3}'

Por último, hemos filtrado la salida del comando anterior, para eliminar la dirección de broadcast junto con la máscara de red para quedarnos sólo con la dirección IP de la máquina:
grep -v 255

La línea anterior no mostraría las líneas que no contengan el valor 255, es decir, las direcciones de broadcast y máscara de red.

Analicemos ahora el comando grep:
grep -oiE '([0-9]{1,3}\.){3}[0-9]{1,3}'

Al igual que en los otros dos ejemplos de expresiones regulares uso las opciones -oiE en el comando grep:

-o Indica que la salida del comando debe contener sólo el texto que coincide con el patrón, en lugar de toda la línea, como es lo habitual.
-i Lo he usado para que ignore la distinción entre mayúsculas y minúsculas.
-E Indica que vamos a usar una expresión regular extendida.

En cuanto a la expresión regular:

'([0-9]{1,3}\.){3}[0-9]{1,3}'
([0-9]{1,3}\.){3} Representa 3 bloques de entre uno y tres dígitos separados por puntos.

Observemos que como el punto es un metacaracter, tengo que usar el caracter de escape \ para que no sea interpretado como un metacaracter, sino como un caracter normal.

El shell de linux: Awk

Awk busca ciertos patrones en la entrada, y la procesa de la manera especificada. Awk tiene una gran funcionalidad, pero esta mayor funcionalidad tiene su coste reflejado en una mayor complejidad del lenguaje.

awk dispone de un lenguaje completo, sintácticamente similar a C que tiene una gran potencia a la hora de reconocer patrones en la entrada, ya que permite especificar combinaciones de expresiones regulares.

Además, no es necesario procesar la entrada línea a línea. Awk permite escoger el carácter que indica el fin de un registro y procesar la entrada de registro en registro (En el lenguaje awk, un 'registro' es el equivalente a una 'línea').
Awk separa automáticamente cada registro en campos que pueden utilizarse individualmente.

Por defecto, un registro es una línea del fichero, lo que significa que el separador de registros es '\n'.

Por defecto, un campo es todo aquello que esté separado por espacios en blanco, es decir, una palabra. El separador de campos por defecto es '[\t]' (espacio y tabulador).

Una posible sintaxis de awk sería:

awk [fichero_entrada]

Un programa de awk es una secuencia de sentencias **patrón-acción**, con un formato determinado, en el que las acciones se ejecutarán si en el registro actual se cumple el patrón.

El formato es el siguiente:

patrón {accion}

Suele ser necesario encerrar los programas de awk entre comillas, para evitar que el shell las interprete como caracteres especiales.

Hay que tener en cuenta dos cosas:
- Si no hay patrón, las acciones se ejecutan en todos los registros.
- Si no hay acciones, lo que se hace es ejecutar la acción por defecto, que es copiar el registro en la salida estándar.

Veamos un par de ejemplos o tres de uso de awk:

* Mostramos el nombre de usuario de todos los usuarios logueados en la máquina:
who|awk '{print $1}'

* Borramos todas las líneas vacías de un fichero:
awk '!/^$/ {}' fichero

* Mostramos el nombre de usuario y el intérprete que usa:
awk 'BEGIN {FS=":"}; {print $1,$NF | "sort"}' /etc/passwd

* Mostramos el nombre completo del usuario y su login:
awk 'BEGIN {FS=":"}; {print $1,$5 | "sort"}' /etc/passwd