Primeros pasos con la shell. Comandos básicos

Sumario

Primeros pasos con la shell

La shell, o intérprete de comandos, es la herramienta más simple y flexible de administración de un Sistema Linux. Como intérprete de comandos el modelo de trabajo es interactivo y consiste en un programa, la shell, que se encarga de procesar los comandos que el usuario teclea y de ofrecer las salidas correspondientes a las peticiones efectuadas por el mismo. Existen varios tipos de shell, con diferente soporte a comandos estándar y comandos específicos. En los Sistemas Linux actuales la shell más utilizada es **bash** (Bourne Shell Again), la cual suele estar en el directorio /bin (/bin/bash), aunque existen otros tipos de shell (como sh, zsh, etc.). Asociadas a las shell existe la posibilidad de creación de programas (scripts) que ejecutan automáticamente, o mediante invocación explícita, una secuencia de comandos de la shell que juntos realizan una tarea del sistema concreta, como por ejemplo hacer una copia de seguridad, enviar un correo automáticamente, descargar actualizaciones, etc. La programación de scripts y su mantenimiento es una tarea administrativa importante pues éstos constituyen una potente herramienta de administración del sistema a la que debemos sacarle partido.

Sintaxis básica de la shell

comando [opciones] [parámetros] [&]

Las **opciones** indican al comando el modo de funcionamiento Los **parámetros** son los datos de entrada sobre los que opera el comando El carácter **&** al final indica, si se especifica, que el comando se ejecuta en segundo plano, es decir, se retomará inmediatamente el prompt de la shell, para seguir introduciendo más comandos, pero el proceso invocado por el comando seguirá en ejecución en "segundo plano"

NOTA: Los corchetes [] indican opcionalidad, es decir, todo aquello que encierran indica que, en la sintaxis del comando, ese elemento es opcional, pudiendo ser especificado o no.

Ejemplo: chmod -R 777 ./archivos

Comando: chmod (cambio de permisos de un archivo) _Opciónes_: -R (indica cambio recursivo en componentes anidados, si es un directorio) _Parámetros_: 777 ./archivos (777, máscara de permisos y ./archivos el archivo o directorio sobre el que cambiamos permisos)

NOTA: "Atajos de teclado de las shell bash":http://ss64.com/bash/syntax-keyboard.html

Obtención de ayuda

Una de las principales necesidades de los usuarios que empiezan con Linux es poder acceder a recursos de ayuda sobre los comandos de la shell. Existen varios comandos que muestran ayuda de comandos shell. El más conocido y que muestra la información más completa es **man**. Este comando posee una sintaxis básica del modo:

man comando

Por ejemplo:

```
man ls
```

mostraría información sobre el comando ls. Una vez que se ejecuta el comando se entra en una sesión interactiva en la que se puede avanzar por la información mostrada en la pantalla. Para avanzar página usamos la tecla "espacio", para saltar de línea "intro" y para salir de la página del **man**ual pulsamos "q". Las páginas del manual se estructuran en varias secciones:

- Comandos, programas y aplicaciones
- Llamadas al sistema
- Funciones de librería
- Ficheros especiales
- Formatos de ficheros
- Juegos
- Otros
- Herramientas de administración del sistema

como es posible encontrar información sobre un comando en varias de las secciones puede restringirse a un ámbito de sección concreto, especificando el número en la llamada al comando. Por ejemplo:

```
man 5 ls
```

La opción -k del comando muestra otros comandos relacionados con el comando pasado como parámetro a man. Además muestra información de en qué sección se encuentra la información relacionada.

Un truco útil es redirigir la salida del comando man hacia un archivo, dónde posteriormente analizar la información, imprimirla, etc.

```
man grep > salida_grep.dat
```

Otros modos de ver ayuda en el sistema Linux son el comando **apropos** (ya desaparecido en algunas distros) y las opción --help (o -h) disponible en muchos de los comandos. También existe el comando **type** que indica información sobre ubicación de un comando especificado.

Otros comandos útiles relacionados con el entorno son **which** que nos da la ruta al ejecutable pasado como parámetro o **whereis**, que muestra información de las rutas en el sistema relacionados con el comando pasado como parámetro

```
which ifconfig
```

mostraría /**sbin**/**ifconfig**

```
whereis ifconfig
```

mostraría **ifconfig: /sbin/ifconfig /usr/share/man/man8/ifconfig.8.gz**

Variables de entorno. PATH

Las variables de entorno son pares del tipo CLAVE=VALOR definidos en un ámbito concreto de la sesión de trabajo del usuario. Pueden tener carácter local o global y son fundamentales para automatizar algunas tareas importantes. Por ejemplo, cuando invocamos a un ejecutable desde la shell debemos, o bien escribir la ruta absoluta al comando, por ejemplo /**usr**/**bin**/**who**, o bien escribir simplemente el nombre del comando **who** en este último caso el Sistema debe de saber de algún modo dónde tiene que buscar el comando invocado, pues de lo contrario arrojará un error del tipo "Unrecognized comand". La variable de entorno que almacena rutas de búsqueda predeterminada es PATH y se le asignan valores por defecto durante la creación de los usuarios. Es posible actualizar su valor y añadir nuevas rutas de búsqueda. Para visualizar las variables de entorno de vuestra sesión ejecutad el comando **env** veréis un listado de la definición de las variables de entorno para vuestro usuario. Las variables pueden tener ámbito local, para la sesión de trabajo actual, o ámbito global, es decir, que aplicarían siempre. Para crear una variables de entorno que solo aplique a la sesión de

trabajo actual escribimos por ejemplo:

```
$ VAR="HOLA MUNDO"
$ echo $VAR
HOLA MUNDO
$
```

Se define la variable de entorno local VAR con valor "HOLA MUNDO" (notad que hay que poner las comillas pues de lo contrario cogerá solo HOLA e intentará ejecutar MUNDO como un comando). A continuación se ejecuta el comando **echo** que envía a la salida estándar (la pantalla) el valor de la cadena pasada como parámetro, en este caso la variable de entorno. Por tanto la salida es "HOLA MUNDO". Si cerráis la consola, abrís una nueva y ejectuáis de nuevo el comando anterior veréis que no escribe nada, esto es porque la variable de entorno es local a la sesión de trabajo y se elimina cuando ésta concluye.

Para definir **variables de entorno de usuario** que no pierdan su valor después de cerrar sesión podemos editar el archivo **.bashrc** dentro del HOME del usuario (/home/<nombre_usuario>), añadiendo la línea CLAVE=VALOR correspondiente a la variable a definir. Para que coja el valor definido podéis reiniciar la shell o escribir el comando

```
source .bashrc
```

El archivo **.bashrc** se ejecuta cuando se inicia una sesión interactiva del usuario.

Para definir **variables de entorno globales** se utilizaba el comando **export**, sin embargo es mejor hacerlo editando el archivo de texto **/etc/environment** (aunque también se podría hacer en /etc/profile) y añadiendo la variable correspondiente en formato CLAVE=VALOR al archivo. Será necesario reiniciar para que el Sistema tenga en cuenta la nueva variable.

El archivo **/etc/rc.local** es una ubicación adecuada para invocar cualquier tipo de comando o script que se debe ejecutar después del inicio de cualquier nivel multiusuario. Debe de contener la línea **exit 0** al final de su contenido.

Algunos comandos básicos

- exit
- who
- date
- passwd
- echo
- clear
- history
- man
- apropos
- whereis
- grep
- basename
- locate

Opciones del comando date. Ejemplo: date +%a,%d-%m-%Y,%H:%M:%S

Literal	Descripción
%a	Nombre abreviado del día de la semana
%A	Nombre completo del día de la semana
%b	Nombre abreviado del mes
%B	Nombre completo del mes
%c	fecha y hora
%C	Dos primeros dígitos del año, ejemplo 20 de 2007
%d	Día del mes con dos dígitos, ejemplo 01
%D	Igual que indicar %m/%d/%y
%e	Día del mes con uno o dos dígitos, ejemplo 1, 10
%F	Fecha completa, igual que %Y-%m-%d
%h	igual que %b
%H	Hora en formato 24 horas con dos dígitos (00..23)
%I	Hora en formato 12 horas con dos dígitos(01..12)
%j	El día del año (001..366)
%k	Hora en formato 24 horas con uno o dos dígitos(0..23)
%l	Hora en formato 12 horas con uno o dos dígitos(1..12)
%m	Mes con dos dígitos(01..12)
%M	Minutos con dos dígitos (00..59)
%r	Hora completa en formato de 12 horas (ejemplo 01:23:45)
%R	Horas y minutos en formato de 24 horas, igual que %H:%M
%s	Segundos transcurridos desde 01/Ene/1970 00:00:00 (fecha epoch)
%S	Segundos con dos dígitos, (00..60)
%T	Hora completa en formato de 24 horas (ejemplo 13:23:45)
%u	Dia de la semana en número (1..7, 1 es lunes)
%U	Número de la semana en el año, domingo primer día de la semana (00..53)
%V	Número de la semana en el año, lunes primer día de la semana (01..53) Formato ISO
%w	Día de la semana en número (0..6, 0 es domingo)
%W	Número de la semana en el año, lunes primer día de la semana (00..53)
%y	Últimos dos dígitos del año
%Y	Año con cutro dígitos
%z	Huso o zona horaria numérica
%Z	Huso o zona horaria abreviación alfabética

Buscador de comandos en el history: Para activar la **reverse-i-search**, que permite realizar una búsqueda en los últimos comandos del histórico, y así poder recuperarlos y volverlos a ejecutar, pulsamos la combinación de teclas **CTRL+R**

Redirecciones y Tuberías

Redirecciones: Cuando trabajamos con la shell siempre existe un entorno de E/S asociado, esto es una Entrada y una Salida estándar. Por defecto la Entrada estándar es el teclado y la Salida estándar, la pantalla. También existe otra interfaz de salida, denominada Error estándar, que indica hacia dónde se envían las condiciones de error que pueden ocurrir durante la ejecución de los comandos. Por defecto el archivo de Error estándar también es la pantalla.

Es posible redirigir la Entrada y Salida de un proceso para que, por ejemplo, tome los datos de entrada de un fichero o los envíe a un fichero (recordad que en Linux todo se modela como un archivo, por lo que el teclado (stdin) es realmente un archivo y la pantalla (stdout) también se maneja a través de un archivo).

```
$ echo "HOLA MUNDO" > salida.dat
```

El comando anterior envía la salida del comando **echo** al archivo **salida.dat**. Si accedemos al contenido del archivo salida.date veremos que contiene el texto HOLA MUNDO.

```
$ wc < entrada.dat
```

El comando wc anterior tomaría la Entrada del archivo entrada.dat y mostraría el resultado correspondiente.

Cuando se redirige la Salida estándar hacia un archivo éste se crea sino existe y en caso de que exista se borra el contenido anterior y se sustituye por el nuevo. Para añadir contenido, sin borrar el contenido anterior, se utiliza el carácter doble >> en la redirección:

```
$ echo "HOLA MUNDO OTRA VEZ" >> salida.dat
```

añadiría al contenido del archivo el texto "HOLA MUNDO OTRA VEZ"

Tuberías: Las Tuberías (pipelines) son un mecanismo rudimentario, pero muy efectivo, de comunicación entre procesos. Básicamente las Tuberías se utilizan para hacer que la entrada de un programa sea la salida de otro. Es la combinación de dos mecanismos de redirección de Entrada y Salida, la redirección de la Salida del primer programa y la redirección de la Entrada del segundo. Esto se entenderá mejor con un ejemplo:

```
$ who | grep root
```

El comando anterior ejecuta un **who**, que muestra los usuarios conectados al Sistema, y dedirige su salida hacia la entrada del comando **grep**, que filtra los resultados por un patrón. En este caso el resultado es mostrar solamente las sesiones iniciadas en el Sistema por el usuario root.

El comando tee se utiliza para duplicar la salida estándar. Envía lo que recibe por entrada estándar a un fichero y a la salida estándar. Veamos un ejemplo:

```
cat /etc/passwd | tee fichero
```

Este comando tomaría como entrada la salida del comando cat /etc/passwd (que lista los contenidos del archivo /etc/passwd dónde se almacenan las cuentas de usuarios del sistema) y duplica la salida enviando una copia de lo que recibe por su entrada al fichero "fichero" y otra copia a la salida estándar (pantalla)

Ejecución de comandos en primer y segundo plano

Al ejecutar un comando desde la terminal, por defecto, la ejecución del mismo se hará en **primer plano (foreground)**, es decir, en modo interactivo, el usuario deberá esperar a que termine la ejecución del mismo para poder continuar introduciendo comandos nuevos. Si introducimos un comando que necesite un tiempo considerable para ejecutarse, como por ejemplo un comando find de búsqueda en el sistema de archivos, tendremos que esperar a que concluya para poder seguir introduciendo comandos en la sesión interactiva de terminal.

Para solucionar este problema podemos optar por ejecutar los comandos en **segundo plano (background)**, esto es, sin necesidad de esperar a que concluyan para poder seguir trabajando en modo interactivo con la terminal. ¿Cómo se consigue esto?, la respuesta es añadir el operador **&** al final del comando, de este modo invocamos el comando para que se ejecute en segundo plano. Veamos un ejemplo

```
sudo find / -name '''.conf > ./find.out &
```

Ejecutará la búsqueda de archivos con extensión .conf en todo el sistema de archivos raíz volcando el resultado de salida de ese comando al archivo find.out en el directorio actual, lo cual podría consumir bastante tiempo. Añadiendo el **&** al final invocaremos el comando en segundo plano y por tanto podremos seguir trabajando con la consola sin necesidad de esperar a que el comando concluya

Otro ejemplo

```
yes > /dev/null &
```

El comando yes es un comando para pruebas que lo que hace es enviar una salida continua del carácter "y" a la salida estándar. En este caso al redirigir la salida evitamos que envíe el resultado a la pantalla. Como queremos evitar que se llene de forma inútil espacio de disco utilizamos como archivo de salida /**dev**/**null**, de modo que la salida de yes no ocupará lugar en el sistema de archivos. Al añadirle el **&** al final del comando lo ejecutaremos en segundo plano, con lo que podremos seguir trabajando con la shell.

El comando jobs

En cualquier momento podemos ver los comandos que se están ejecutando en segundo plano mediante este comando. Además del comando la salida de jobs mostrará también un número que identifica el trabajo para poder interactuar con él de ser necesario

```
jobs
[1]+  Ejecutando              sudo find / -name '''.conf &
[2]+  Ejecutando              nohup cp -r /media/disco/20''' CIS/ &
```

El comando fg

Si en algún momento queremos traer a primer plano un trabajo que se está ejecutando en segundo plano, usaremos este comando. Si se invoca sin argumentos traerá a primer plano el trabajo más reciente ejecutado en segundo plano. Si se quiere traer a primer plano un trabajo determinado usamos el identificador numérico del trabajo para determinarlo. Veamos un ejemplo

```
fg %1
```

Traería a primer plano el trabajo con identificador 1 en segundo plano

El comando bg

Es un comando simétrico del anterior, permite enviar a segundo plano un comando ejecutado en primer plano sin necesidad de deternerlo. Para poder utilizarlo previamente debemos pulsar **CTRL+Z**, que detendrá el trabajo sin matarlo, para poder ejecutar éste en segundo plano. Veamos un ejemplo

```
* find / -ctime -1 > /tmp/changed-file-list.txt

* [CTRL-Z]
[2]+  Stopped                 find / -ctime -1 > /tmp/changed-file-list.txt

* bg
```

El comando find se ejecuta en primer plano, al no introducirse el **&** al final del comando, posteriormente el trabajo se detiene con CTRL+Z. Por último, mediante el comando **bg** enviamos el trabajo a segundo plano

Comando kill para trabajos en segundo plano

Si necesitamos matar un proceso que se está ejecutando en segundo plano usaremos el comando kill, utilizando como referencia al proceso del comando el identificador numérico que éste tiene asociado en segundo plano, es decir, el **[id]** que se visualiza mediante el comando **jobs**

```
kill %2
```

Mataría el proceso con [id] 2 en segundo plano

Ejecución de procesos independientes de la sesión

Otro problema común es como proceder cuando un trabajo toma tanto tiempo en terminar que es posible que se cierre la sesión de terminal en la que se invocó. Esto, debido al modelo de procesos jerárquico de Linux, causaría que al morir el padre, el proceso de la terminal, todos sus hijos, incluído el

comando que no ha terminado también sean eliminados. ¿Como evitamos esto?. La respuesta es mediante el comando **nohup**. Precediendo el comando a ejecutar del comando **nohup**, independizamos éste de la terminal desde la cual se invoca, de modo que si cerramos ésta el proceso seguirá ejecutándose en el sistema. Veamos un ejemplo

```
nohup find / -name '''.dat &
```

Ejecutaría el find en segundo plano y además lo haría de modo que si cerramos la sesión interactiva de terminal el find continuaría ejecutándose

Filtros y comandos de ordenación

Filtros:

Un filtro permite establecer algún tipo de criterio sobre la información que queremos mostrar. Por ejemplo, para ver en los logs del sistema aquellas acciones relacionadas con el proceso de logueo de los usuarios correspondiente al usuario root escribiríamos lo siguiente:

```
cat /var/log/auth.log | grep root
```

El comando **cat** lista los contenidos del archivo, a continuación envíe su salida a la entrada del comando **grep** (tubería) y grep filtra las líneas de texto del archivo en las que aparezca el término "root".

Por tanto uno de los usos del comando grep es buscar en las líneas de un determinado archivo, o salida de un comando, las ocurrencia de la expresión indicada. La sintaxis del comando es:

grep [-opciones] expresion_regular fichero(s)

donde las opciones modifican su funcionamiento, la **expresión regular** identifica el patrón a buscar (una expresión regular puede contener metacaracteres y comodines que sustituyen a otros caracteres y expresar patrones complejos) y **fichero(s)** es el o los ficheros en los que vamos a aplicar el comando.

Algunas opciones interesantes de grep:

- **-i**: Búsquedas case insensitive, es decir no sensibles a mayúsculas minúsculas
- **-r**: Búsquedas recursivas en el directorio actual y subdirectorios
- **-v**: Búsqueda inversa, muestra las líneas que no verifican el patrón de búsqueda de la expresión
- **-c**: Cuenta el número de coincidencias que verifican el patrón
- **-B N**: Donde N es un número positivo, muestra las N líneas antes (Before) de la coincidencia
- **-A N**: Donde N es un número positivo, muestra las N líneas después (After) de la coincidencia
- **-C N**: Donde N es un número positivo, muestra las N líneas antes y después de la coincidencia
- **-n**: Muestra el número de línea de la coincidencia

El trabajo con expresiones regulares dota de un gran potencial y expresividad a ciertos comandos de Linux. Mediante el uso de expresiones regulares y de otras herramientas todavía más potentes (como awk) podemos definir y caracterizar patrones textuales que nos permitirán simplificar tareas de búsqueda, sustitución y ordenación de textos.

La variable de entorno **GREP_OPTIONS** controla el funcionamiento de grep, por ejemplo para resaltar el color de las coincidencias en la búsqueda. Por ejemplo lo siguiente permite que las coincidencias se muestren resaltadas

```
export GREP_OPTIONS='--color=auto' GREP_COLOR='100;8'
```

Algunos ejemplos con grep

Mostraría las líneas del archivo /etc/group que coincidan con la expresión de búsqueda "ROOT" de modo no sensible a mayúsculas y minúsculas

```
grep -i "ROOT" /etc/group
```

Mostraría las líneas del archivo /etc/passwd que no contengan la expresión de búsqueda root

```
grep -v "root" /etc/passwd
```

Combinando las dos anteriores:

```
grep -v -i "ROOT" /etc/passwd
```

Mostraría las 3 líneas después de la coincidencia de búsqueda:

```
grep -A 3 -i "root" /etc/passwd
```

Algunas herramientas de ordenación:

Un comando útil cuyo objetivo es ordenar alfanumériamente el contenido de un archivo o la salida de un comando es **sort**. Su sintaxis es:

sort [opciones] fichero

Algunas opciones: -n, indica que la ordenación es numérica y no alfabética, -r hace una ordenación "inversa". Un ejemplo:

```
sort -r /etc/passwd
```

ordenaría, a la inversa (es decir empezando por las líneas que comienzan por z y finalizando con las que empiezan por a) los contenidos del archivo /etc/passwd

uniq

Comando utilizado en conjunto con sort y que elimina los duplicados después de una operación de tipo sort

```
sort -r /etc/passwd | uniq
```

El comando cut

Este es un comando útil porque permite **cortar** las salidas tabuladas de los comandos para seleccionar simplemente una columna, o intervalo, de la salida. Por ejemplo, si ejecutamos

```
ls -la /etc/rc2.d
```

La salida será:

```
total 12
drwxr-xr-x  2 root root 4096 Aug  1 00:38 .
drwxr-xr-x 94 root root 4096 Aug 20 21:13 ..
-rw-r--r--  1 root root  677 Jul 26  2012 README
lrwxrwxrwx  1 root root   31 May  1  2012 S14xe-linux-distribution -> ../init.d/xe-linux-distribution
lrwxrwxrwx  1 root root   23 Aug  1 00:38 S20clamav-daemon -> ../init.d/clamav-daemon
lrwxrwxrwx  1 root root   26 Aug  1 00:38 S20clamav-freshclam -> ../init.d/clamav-freshclam
lrwxrwxrwx  1 root root   20 Jul 12 22:29 S20nova-agent -> ../init.d/nova-agent
lrwxrwxrwx  1 root root   13 Jul 31 18:30 S23ntp -> ../init.d/ntp
lrwxrwxrwx  1 root root   15 May  1  2012 S50rsync -> ../init.d/rsync
lrwxrwxrwx  1 root root   19 May  1  2012 S70dns-clean -> ../init.d/dns-clean
lrwxrwxrwx  1 root root   18 May  1  2012 S70pppd-dns -> ../init.d/pppd-dns
lrwxrwxrwx  1 root root   14 Mar  5 03:52 S75sudo -> ../init.d/sudo
lrwxrwxrwx  1 root root   17 Jul 31 11:28 S91apache2 -> ../init.d/apache2
lrwxrwxrwx  1 root root   18 Jul 31 12:32 S99fail2ban -> ../init.d/fail2ban
lrwxrwxrwx  1 root root   21 May  1  2012 S99grub-common -> ../init.d/grub-common
lrwxrwxrwx  1 root root   18 May  1  2012 S99ondemand -> ../init.d/ondemand
lrwxrwxrwx  1 root root   18 May  1  2012 S99rc.local -> ../init.d/rc.local
```

Si solo queremos que se muestre una columna, por ejemplo la correspondiente al usuario propietario, podríamos utilizar el comando cut

```
ls -la /etc/rc2.d | cut -d' ' -f4
```

La expresión anterior indica: cortar, utilizando el **separador espacio (-d ' ')**, la salida del comando **ls -la /etc/rc2.d** y **mostrar por salida la columna 4 (-f4)** Por supuesto que puede especificarse otro separador, especificándolo en la opción **-d**, y de posición del campo, con la opción **-f**

Para más información ejecutar

```
cut --help
```

El comando find

Este es uno de los comandos más potentes y útiles de búsqueda en Linux. Admite una sintaxis muy flexible, aunque básicamente esta consiste en:

find directorio atributos

dónde _directorio_ indica el punto de partida de la búsqueda y _atributos_ se refiere a las características que deben poseer los archivos buscados. Además, el comando find soporta cierto nivel de interacción, en la medida en que se puede especificar, con la opción -exec, un comando para ejecutar por cada uno de los archivos que verifiquen la condición indicada en los atributos del comando. El comando find devuelve como salida una lista con los archivos que verifiquen la condición indicada mediante los atributos. Se puede especificar mediante el parámetro n (+n, más de n, -n, menos de n, o n, exactamente n) que la condición del atributo se satisfaga en relación a el número indicado por n. Veamos algunos de los atributos soportados:

- **-atime n**: accedido en los últimos n días
- **-mtime n**: modificados sus datos en los últimos n días
- **-ctime n**: estado del archivo modificado en los últimos n días
- **-group grupo**: cierto si el grupo del fichero es el indicado
- **-nogroup**: cierto si el fichero no tiene grupo válido
- **-nouser**: cierto si pertenece a un usuario no dado de alta en el sistema
- **-user usuario**: cierto si el fichero es propiedad del usuario
- **-inum n**: cierto si el archivo tiene el i-nodo n
- **-ls**: siempre cierto, se usa para mostrar información del fichero
- **-newer fichero**: cierto si el fichero es más nuevo que el fichero indicado
- **-name patrón**: cierto si el nombre del fichero está incluido en el conjunto indicado por la expresión regular del patrón
- **-exec**: opción muy importante, permite la ejecución de un comando. Tiene que terminar con el carácter ;. La expresión {} representa a la ocurrencia de cada archivo que verifique la condición
- **-ok**: igual que -exec, salvo que pide confirmación antes de ejecutar
- **-perm -mode**: cierto para aquellos ficheros que tienen los permisos indicados
- **-print**: siempre es cierto, usado para imprimir la ruta de los ficheros seleccionados
- **-type c**: cierto si el tipo de fichero es el indicado (c puede ser d, directorio, f, fichero normal, l, enlace)
- **-size n[c]**: cierto si el fichero es de n bloques (512 bytes), si se incluye la c, el tamaño es tratado como bytes

Se pueden utilizar operadores lógicos para dotar de mayor expresividad a las condiciones del comando. Las opciones de agrupación y operadores lógicos son: ! niega una condición -o O Lógico -a A Lógico () Los paréntesis se utilizan para agrupar condiciones

Ejemplos:

```
find . -name "cuentas'''"
```

Encuentra todos los archivos dentro del directorio actual (.) y todos los subdirectorios cuyo nombre empieza por "cuentas"

```
find . -user irene
```

Encuentra todos los archivos dentro del directorio actual y subdirectorios propiedad del usuario irene

```
find . -user irene -mtime 2
```

Encuentra todos los archivos propiedad de irene que hayan sido modificados en los últimos 2 días

```
find . -name "tmp'''" -exec rm {} \;
```

Encuentra todos los archivos cuyo nombre empiece por tmp y ejecuta un rm (borrado) sobre ellos. Notar que el ; ha sido escapado por ser un carácter con significado para el intérprete de comandos

```
find / \(-name '''.o -o -name '''.out\) -a -perm -o+w
```

En este ejemplo vemos una concatenación lógica con un "O lógico" "-o", dentro del paréntesis y un "Y lógico" "-a". Buscaría aquellos archivos que verifiquen que su nombre acaba en .o o .out y que al mismo tiempo tengan en su máscara de permisos o+w (permiso de escrita para otros). De nuevo es necesario escapar los caracteres con significado para la shell, en este caso los paréntesis

Comando para visualizar los archivos propiedad de root con setuid activado:

```
sudo find / -perm -u=rws -a -user root
```

Comando locate

Un comando similar a find que realiza búsquedas instantáneas, a diferencia de éste que busca recursivamente en el sistema de archivos según los criterios especificados, es **locate**. Con locate podemos obtener información de localización de archivos y directorios. Algunas opciones del mismo son:

- **-e**: busca solo archivos existentes que no hayan sido borrados
- **-i**: no distingue entre mayúsculas y minúsculas
- **-c**: muestra el número de archivos cuyo nombre coincide con el patrón de búsqueda indicado

```
locate ifconfig
```

Mostraría todos los archivos del sistema relacionados con el término ifconfig, es decir que contengan ese término en su nombre

La diferencia principal entre find y locate es que este último localiza los resultados de búsqueda en una base de datos que se actualiza periódicamente, mediante el procesamiento del sistema de archivos. La base de datos reside en el directorio /**var**/**lib**/**mlocate**/**mlocate.db** u el archivo de configuración /**etc**/**updatedb.conf** permite configurar el proceso de actualización de la misma

Sustituciones de Shell

Caracteres ` ` y expresión $()

Si encerramos un texto **`comando`** en Linux (Tecla también de los caracteres [y ^ que se obtiene pulsando la tecla y luego "espacio") obtendremos la ejecución del resultado del **comando indicado por el texto**. Esta construcción es sintácticamente equivalente a **$(comando) Por ejemplo:**

```
echo hola, soy `whoami`
```

Nos devolverá un saludo del tipo "hola, soy <nombre_usuario>", sustituyendo el <nombre_usuario> por el nombre del usuario de la sesión actual Es útil para obtener información dinámica, resultado de la ejecución de comandos. Se usa mucho en scripting y cuando queremos usar la salida textual de un comando como entrada de otro.

Un comando similar sería:

```
echo hola, son las: $(date +%H:%M)
```

Que muestra un texto indicando la hora actual, precedido de "hola, son las:..."

Imaginemos que tenemos un archivo install.dat dónde tenemos la siguiente línea:

apt-get install mysql apache2 php5 phpmyadmin slapd slurpd phpldapadmin

Que no es más que un conjunto de paquetes que queremos instalar. ¿Cómo podemos ejecutar ese comando sin tener que "copiar y pegar" el contenido a la consola?, es decir, ¿cómo podemos utilizar un texto arbitrario (en este caso dentro de un archivo) para que se ejecute como un comando de shell?. Así:

```
`cat install.dat`
```

Esto ejecutaría el resultado de listar el contenido del archivo de texto install.dat, que es, ni más ni menos, que lo que queremos hacer, sin necesidad de copiar ni pegar nada. Es muy útil cuando queramos hacer instalaciones desde tutoriales y no podemos copiar y pegar directamente los comandos en la terminal de destino (por ejemplo si es una máquina virtual o remota). Crearíamos un texto en un archivo local con el contenido a ejecutar, copiamos, por ejemplo con scp el archivo en el sistema destino y ejecutamos un comando similar al anterior.

Referencias

Introducción a grep

- https://www.thegeekstuff.com/2009/03/15-practical-unix-grep-command-examples
- http://docs.oracle.com/cd/E19620-01/805-7644/6j76klop3/index.html

Lista de comandos Linux/Debian:

- http://www.esdebian.org/wiki/lista-comandos-gnulinux-i
- http://www.esdebian.org/wiki/lista-comandos-gnulinux-ii
- http://www.esdebian.org/wiki/lista-comandos-gnulinux-iii

Debian Environment Variables:

- http://wiki.debian.org/EnvironmentVariables

Volver

JavierFP 16:31 08 ene 2019 (CET)