# 17

# BUSQUEDA DE ARCHIVOS

Mientras deambulamos por nuestro sistema Linux, una cosa se ha vuelto muy clara: ¡un sistema Linux típico tiene muchos archivos! Esto plantea la pregunta: "¿Cómo encontramos las cosas?" Ya sabemos que el sistema de archivos de Linux está bien organizado de acuerdo con las convenciones transmitidas de una generación de sistemas similares a Unix a la siguiente, pero la gran cantidad de archivos puede presentar un problema abrumador.

En este capítulo, veremos dos herramientas que se utilizan para buscar archivos en un sistema.

- locate - (localizar) : Buscar archivos por nombre
- find (buscar) : Buscar archivos en una jerarquía de directorios

También veremos un comando que se usa a menudo con comandos de búsqueda de archivos para procesar la lista de archivos resultante.

- xargs : Construye y ejecuta líneas de comando desde la entrada estándar

Además, presentaremos un par de comandos para ayudarnos en nuestras exploraciones.

- touch (toque) : Cambiar tiempos de archivo
- stat : Mostrar archivo o estado del sistema de archivos

## locate -- encuentra archivos de manera facil.

El programa locate realiza una búsqueda rápida en la base de datos de nombres de ruta y luego genera cada nombre que coincide con una subcadena determinada. Digamos, por ejemplo, que queremos encontrar todos los programas con nombres que comiencen con zip. Debido a que estamos buscando programas, podemos asumir que el nombre del directorio que contiene los programas terminaría en bin/. Por lo tanto, podríamos intentar usar la ubicación de esta manera para encontrar nuestros archivos:

```
[~]$ locate bin/zip
```

Si el requisito de búsqueda no es tan sencillo, podemos combinar la localización con otras herramientas como grep para diseñar búsquedas más interesantes.

```
[~]$ locate zip | grep bin
```

El programa de localización existe desde hace varios años y existen varias variantes de uso común. Los dos más comunes que se encuentran en las distribuciones modernas de Linux son slocate y mlocate, aunque generalmente se accede a ellos mediante un enlace simbólico llamado location. Las diferentes versiones de

localizar tienen conjuntos de opciones superpuestos. Algunas versiones incluyen la coincidencia de expresiones regulares (que trataremos en el Capítulo 19) y compatibilidad con comodines. Consulte la página del manual de localización para determinar qué versión de localización está instalada.

# find. -- busqueda de archivos de la manera dura.

Mientras que el programa de locate puede encontrar un archivo basándose únicamente en su nombre, el programa find busca archivos en un directorio determinado (y sus subdirectorios) basándose en una variedad de atributos. Vamos a dedicar mucho tiempo a 'find' porque tiene muchas características interesantes que veremos una y otra vez cuando comencemos a cubrir conceptos de programación en capítulos posteriores.

En su uso más simple, find recibe uno o más nombres de directorios para buscar. Por ejemplo, para producir una lista de nuestro directorio de inicio, podemos usar esto:

```
[~]$ find ~
```

En la mayoría de las cuentas de usuario activas, esto producirá una lista grande. Debido a que la lista se envía a la salida estándar, podemos canalizar la lista a otros programas.

Usemos wc para contar el número de archivos.

```
[~]$ find ~ | wc -l
```

¡Vaya, hemos estado ocupados! La belleza de Find es que se puede utilizar para identificar archivos que cumplan con criterios específicos. Lo hace mediante la aplicación (un poco extraña) de opciones, pruebas y acciones. Primero veremos las pruebas.

## Pruebas

Supongamos que queremos una lista de directorios de nuestra búsqueda. Para ello, podríamos agregar la siguiente prueba:

```
[~]$ find ~ -type d | wc -l
```

La adición de la prueba -type d limitó la búsqueda a directorios. Por el contrario, podríamos haber limitado la búsqueda a archivos normales con esta prueba:

```
[~]$ find ~ -type f | wc -l
```

| Tipo de archivo | Descripción |
|---|---|
| b | Bloque de archivo de dispositivo especial |
| c | Archivo de dispositivo especial de carácter |
| d | Directorio |
| f | Archivo regular |
| l | Enlace simbólico |

También podemos buscar por tamaño de archivo y nombre de archivo agregando algunas pruebas adicionales. Busquemos todos los archivos normales que coincidan con el patrón comodín *.JPG y que tengan un tamaño superior a un megabyte.

```
[~]$ find ~ -type f -name \"*.JPG\" -size +1M | wc -l 840
```

En este ejemplo, agregamos la prueba -name seguida del patrón de comodín.

Observe cómo lo encerramos entre comillas para evitar que el shell expanda el nombre de ruta. A continuación, agregamos la prueba -size seguida de la cadena + 1M. El signo más inicial indica que estamos buscando archivos más grandes que el número especificado.

Un signo menos inicial cambiaría el significado de la cadena para que sea más pequeño que el número especificado. No usar ningún signo significa "coincidir exactamente con el valor".

La letra final M indica que la unidad de medida es megabytes.

La Tabla enumera los caracteres que se pueden usar para especificar unidades.

```
find size units, + mayor, - menor que.
```

| Carácter | Unidad |
|---|---|
| b | Bloques de 512 bytes. Este es el valor predeterminado si no se especifica ninguna unidad. |
| c | Bytes. |
| w | Palabras de 2 bytes. |
| k | Kilobytes (unidades de 1.024 bytes). |
| M | Megabytes (unidades de 1.048.576 bytes). |
| G | Gigabytes (unidades de 1.073.741.824 bytes). |

find admite una gran cantidad de pruebas. La tabla 17-3 proporciona un resumen de los más comunes. Tenga en cuenta que en los casos en que se requiera un argumento numérico, se puede aplicar la misma notación + y - discutida anteriormente.

find tests-

| Prueba | Descripción |
|---|---|
| -cmin n | Coincidir con archivos o directorios cuyo contenido o atributos se modificaron por última vez hace exactamente n minutos. Para especificar hace menos de n minutos, use -n, y para especificar hace más de n minutos, use + n. |
| -cnewer archivo | Coincidir con archivos o directorios cuyo contenido o atributos se modificaron por última vez más recientemente que los de archivo. |
| -ctime n | Coincidir con archivos o directorios cuyo contenido o atributos se modificaron por última vez hace n * 24 horas. |
| -empty | Coincidir con archivos y directorios vacíos. |

| Prueba | Descripción |
|---|---|
| -group name | Coincidir con el archivo o directorios que pertenecen al nombre del grupo. El nombre puede expresarse como un nombre de grupo o como un ID de grupo numérico. |
| -iname pattern | Como la prueba -name pero no distingue entre mayúsculas y minúsculas. |
| -inum n | Haga coincidir archivos con el número de inodo n. Esto es útil para encontrar todos los enlaces físicos a un inodo en particular. |
| -mmin n | Coincide con archivos o directorios cuyo contenido se modificó por última vez hace n minutos. |
| -mtime n | Coincide con archivos o directorios cuyo contenido se modificó por última vez hace n * 24 horas. |
| -name pattern | Haga coincidir archivos y directorios con el patrón comodín especificado. |
| -newer file | Coincidir con archivos y directorios cuyo contenido se modificó más recientemente que el archivo especificado. Esto es útil al escribir scripts de shell que realizan copias de seguridad de archivos. Cada vez que haga una copia de seguridad, actualice un archivo (como un registro) y luego use buscar para determinar qué archivos han cambiado desde la última actualización. |
| -nouser | Coincidir con archivos y directorios que no pertenecen a un usuario válido. Esto se puede utilizar para encontrar archivos pertenecientes a cuentas eliminadas o para detectar actividad de atacantes. |
| -nogroup | Coincidir archivos y directorios que no pertenecen a un grupo válido. |
| -perm mode | Haga coincidir archivos o directorios que tengan permisos establecidos en el modo especificado. el modo puede expresarse mediante notación octal o simbólica. |
| -samefile name | Similar a la prueba -inum. Haga coincidir los archivos que comparten el mismo número de inodo que el nombre del archivo. |
| -size n | Coincidir con archivos de tamaño n. |
| -type c | Coincidir con archivos de tipo c. |
| -user name | Coincidir con archivos o directorios que pertenecen al nombre de usuario. El usuario puede expresarse mediante un nombre de usuario o mediante un ID de usuario numérico. |

```
[~]$ man find # muestra todos los detalles
```

## Operadores

Incluso con todas las pruebas que proporciona Find, es posible que necesitemos una mejor manera de describir las relaciones lógicas entre las pruebas. Por ejemplo, ¿qué pasaría si tuviéramos que determinar si todos los archivos y subdirectorios de un directorio tienen permisos seguros? Buscaríamos todos los archivos con permisos que no sean 0600 y los directorios con permisos que no sean 0700.

Afortunadamente, find proporciona una forma de combinar pruebas utilizando operadores lógicos para crear relaciones lógicas más complejas. Para expresar la prueba antes mencionada, podríamos hacer esto:

```
[~]$ find ~ \( -type f -not -perm 0600 \) -or \( -type d -not -perm 0700 \)
```

¡Ay! Seguro que se ve raro. Qué es todo esto? En realidad, los operadores no son tan complicados una vez que los conoces. La Tabla 'OPERADORES LÓGICOS' describe los operadores lógicos que se utilizan con find.

> OPERADORES LÓGICOS find.

| Operador | Descripción |
|---|---|
| -and | Haga coincidir si las pruebas en ambos lados del operador son verdaderas. Esto se puede abreviar a -a. Tenga en cuenta que cuando no hay ningún operador presente, -y está implícito de forma predeterminada. |
| -or | Coincide si una prueba en cualquier lado del operador es verdadera. Esto se puede acortar a -o. |
| -not | Coincidir si la prueba que sigue al operador es falsa. Esto se puede abreviar con un signo de exclamación (!). |
| () | Agrupe las pruebas y los operadores para formar expresiones más grandes. Se utiliza para controlar la precedencia de las evaluaciones lógicas. De forma predeterminada, find evalúa de izquierda a derecha. A menudo es necesario anular el orden de evaluación predeterminado para obtener el resultado deseado. Incluso si no es necesario, a veces es útil incluir los caracteres de agrupación para mejorar la legibilidad del comando. Tenga en cuenta que, dado que los paréntesis tienen un significado especial para el shell, deben estar entrecomillados al usarlos en la línea de comandos para permitir que se pasen como argumentos para encontrar. Por lo general, el carácter de barra invertida se usa para escapar de ellos. |

Con esta lista de operadores en la mano, vamos a deconstruir nuestro comando de búsqueda.

Cuando se ve desde el nivel superior, vemos que nuestras pruebas están organizadas como dos agrupaciones separadas por un operador -or.

```
(expresion 1) -or (expresion 2)
```

Esto tiene sentido porque estamos buscando archivos con un determinado conjunto de permisos y directorios con un conjunto diferente. Si buscamos tanto archivos como directorios, ¿por qué usamos -o en lugar de -and? A medida que Find explora los archivos y directorios, cada uno se evalúa para ver si coincide con las pruebas especificadas. Queremos saber si se trata de un archivo con permisos incorrectos o de un directorio con permisos incorrectos. No puede ser ambos al mismo tiempo. Entonces, si expandimos las expresiones agrupadas, podemos verlo de esta manera:

```
(archivo con permisos erroneos) -or (directorio con permisos incorrectos)
```

Nuestro próximo desafío es cómo probar los "permisos incorrectos". ¿Como hacemos eso? De hecho, no lo hacemos. Lo que probaremos es que "no buenos permisos" porque sabemos los que son "buenos permisos". En el caso de archivos, definimos bueno como 0600, y para directorios, lo definimos como 0700. La expresión que comprobará archivos para permisos "no buenos" es la siguiente:

```
-type f -and -not -perms 0600
```

pare directorios:

```
-type d -and -not -perms 0700
```

Como vimos en la tabla de operadores, el operador -an puede ser eliminado porque está implicito por defecto, así que si ponemos todo esto junto, obtenemos nuestro comando final:

```
find ~ (-type -f not -perm 0600) -or (-type d -not -perms 0700)
```

Sin embargo, los parentesis tienen un especial significado para la consola, debemos escapar de ellos previniendo que la consola intente interpretarlos, precediendo cada uno de ellos con una barra invertida.

Hay otra característica de los operadores lógicos que es importante comprender. Digamos que tenemos dos expresiones separadas por un operador lógico.

```
expr1 -operator expr2
```

En todos los casos, siempre se ejecutará expr1; sin embargo, el operador determinará si se realiza expr2. La Tabla 'lógica AND/OR' describe cómo funciona

> Lógica AND/OR en find.

| **Resultados de expr1** | **Operador** | **expr2 es. . .** |
|---|---|---|
| Verdadero | -and | Siempre realizado |
| Falso | -and | Nunca realizado |
| Verdadero | -or | Nunca realizado |
| Falso | -or | Siempre realizado |

¿Por qué pasó esto? Está hecho para mejorar el rendimiento. Tome -y, por ejemplo. Sabemos que la expresión expr1 -y expr2 no puede ser verdadera si el resultado de expr1 es falso, por lo que no tiene sentido realizar expr2. Del mismo modo, si tenemos la expresión expr1 -o expr2 y el resultado de expr1 es verdadero, no tiene sentido realizar expr2, ya que ya sabemos que la expresión expr1 -o expr2 es verdadera.

Bien, entonces ayuda a que vaya más rápido. ¿Porque es esto importante? Es importante porque podemos confiar en este comportamiento para controlar cómo se realizan las acciones, como veremos pronto.

## Acciones predefinidas

¡Hagamos un poco de trabajo! Tener una lista de resultados de nuestro comando de búsqueda es útil, pero lo que realmente queremos hacer es actuar sobre los elementos de la lista. Afortunadamente, find permite realizar acciones basadas en los resultados de la búsqueda. Hay un conjunto de acciones predefinidas y varias formas de aplicar acciones definidas por el usuario. Primero, veamos algunas de las acciones predefinidas enumeradas en la Tabla 'ACCOPMES PREDEFINIDAS'.

> ACCIONES PREDEFINIDAS con find.

| Acción | Descripción |
|---|---|
| -delete | Elimina el archivo que coincide actualmente. |
| -ls | Realice el equivalente de ls -dils en el archivo coincidente. La salida se envía a la salida estándar. |
| -print | Genere la ruta completa del archivo coincidente en la salida estándar. Esta es la acción predeterminada si no se especifica ninguna otra acción. |
| -quit | Salga una vez que se haya hecho una coincidencia. |

Para mayor informacion escribir en consola 'man find'.

En el primer ejemplo hacemos esto:

[~]$ find ~

Esto produce una liste de todos los archivos y subdirectorios contenidos dontro de nuestro directorio home.

Eso produce una lista porque la acción -print es implicita en cuanto otra acción es especificada. nuestro comando podría ser expresado como 'find ~ -print'

Podemos usar find para eliminar archivos que cumplan con ciertos criterios. Por ejemplo, para eliminar archivos que tienen la extensión de archivo .bak (que se usa a menudo para designar archivos de respaldo), podríamos usar este comando:

```
[~]$ find ~ -type f -name '*.bak' -delete
```

En este ejemplo cada archivo en el directorio del usuario y en sus subdirectorios es buscado por su terminacion en .bak cuando son encontrados son borrados.

Advertencia: No hace falta decir que debe tener mucho cuidado al utilizar la acción -delete. Siempre pruebe primero el comando sustituyendo la acción -print por -delete para confirmar los resultados de la búsqueda

Antes de continuar, echemos otro vistazo a cómo los operadores lógicos afectan las acciones. Considere el siguiente comando:

```
[~]$ find ~ -type f -name '*.bak' -print
```

Como hemos visto, este comando buscará todos los archivos normales (-tipo f) cuyo nombre termine con .bak (-name '* .bak') y generará la ruta relativa de cada archivo coincidente en la salida estándar (-print) . Sin

embargo, la razón por la que el comando se ejecuta de la manera en que lo hace está determinada por las relaciones lógicas entre cada una de las pruebas y acciones. Recuerde, hay, de forma predeterminada, una relación implícita entre cada prueba y acción. También podríamos expresar el comando de esta manera para que las relaciones lógicas sean más fáciles de ver:

```
[~]$ find ~ -type f -name '*.bak' -and -print
```

Con nuestro comando completamente expresado, veamos cómo los operadores lógicos afectan su ejecución:

| Prueba/Acción | Se realiza solo si ... |
|---|---|
| -print | -type f y -name '* .bak' son verdaderos |
| -name '* .bak' | -tipo f es cierto |
| -tipo f | Siempre se realiza, ya que es la primera prueba / acción en una relación -and. |

Debido a que la relación lógica entre las pruebas y las acciones determina cuáles de ellas se realizan, podemos ver que el orden de las pruebas y las acciones es importante. Por ejemplo, si tuviéramos que reordenar las pruebas y acciones para que la acción -print fuera la primera, el comando se comportaría de manera muy diferente.

```
[~]$ find ~ -print -and -type f -and -name '*.bak'
```

Esta prueba de comando imprimiría todos los archivos (la acción -print se evalua como verdadera) y luego para el tipo de archivo y el archivo de extension especifico.

## Acciones definidas por el usuario

Además de las acciones predefinidas, también podemos invocar comandos arbitrarios. La forma tradicional de hacer esto es con la acción -exec. Esta acción funciona así:

```
-exec command {};
```

Aquí command es el nombre de un comando, {} es una representación simbolica del directorio corriente y ; es un delimitador requerido para indicar la finalización del comando, aquí un ejemplo del uso de -exec para la acción parecida de -delete:

```
-exec rm '{}' ';'
```

Nuevamente, debido a que los caracteres de llave y punto y coma tienen un significado especial para el shell, deben estar entre comillas o escapar ''.

También es posible ejecutar una acción definida por el usuario de forma interactiva. Al usar la acción -ok en lugar de -exec, se le solicita al usuario antes de ejecutar cada comando especificado.

```
[~]$ find ~ -type f -name 'foo*' -ok ls -l '{}' ';'
```

En este ejemplo, buscamos por archicos con nombres que empiecen por foo y ejecuta el comando ls -l para cada que es encontrado. usando la acción -ok solicita al usuario antes que el comando ls sea ejecutado.

## Mejorando la eficiencia

Cuando se usa la acción -exec, lanza una nueva instancia del comando especificado cada vez que se encuentra un archivo coincidente. Hay ocasiones en las que podríamos preferir combinar todos los resultados de la búsqueda y lanzar una sola instancia del comando. Por ejemplo, en lugar de ejecutar comandos como este:

```
ls -l file1
ls -l file2
```

prodriamos preferir ejecutarlo de esta manera:

```
ls -l file1 file2
```

Esto hace que el comando se ejecute solo una vez en lugar de varias veces. Hay dos formas en que podemos hacer esto: la forma tradicional, usando el comando externo xargs, y la forma alternativa, usando una nueva función en find. Primero hablaremos de la forma alternativa.

Al cambiar el carácter de punto y coma final a un signo más, activamos la capacidad de buscar para combinar los resultados de la búsqueda en una lista de argumentos para una sola ejecución del comando deseado. Volviendo a nuestro ejemplo, esto ejecutará ls cada vez que se encuentre un archivo coincidente:

```
[~]$ find ~ -type f -name 'foo*' -exec ls -l '{}' ';'
```

lista todos loas archivos encontrados que empiecen por el nombre foo según criterio ls -l.

si cambiamos el comando como sigue:

```
[~]$ find ~ -type f -name 'foo*' -exec ls -l '{}' +
```

Obtenemos el mismo resultado, solo que en vez de ir saliendo por consola las acciones, la obtnemos de una sola vez cuando acabe el proceso.

## xargs

El comando xargs realiza una función interesante. Acepta la entrada de la entrada estándar y la convierte en una lista de argumentos para un comando específico. Con nuestro ejemplo, lo usaríamos así:

```
[~]$ find ~ -type f -name 'foo*' -print | xargs ls -l
```

Aquí vemos la salida del comando find canalizada en xargs, que, a su vez, construye una lista de argumentos para el comando ls y luego la ejecuta.

> **ANOTACION**: Si bien la cantidad de argumentos que se pueden colocar en una línea de comando esbastante grande, no es ilimitada. Es posible crear comandos que sean demasiado largos para que el shell los acepte. Cuando una línea de comando excede la longitud máxima admitida por el sistema, xargs ejecuta el comando especificado con el número máximo de argumentos posible y luego repite este proceso hasta que se agota la entrada estándar.

Para ver el tamaño máximo de la línea de comando, ejecute xargs con la opción --show-limits.

```
[~]$ find ~ -type f -name 'foo*' -print | xargs --show-limits ls -l
```

> **ANOTACION:**

> Para resolver el problema de los espacios en blanco de los archivos largos, existe la opción -print0 y para xargs --null or (-o):

```
[me@EliteDesk:~]$ find ~ -type f -name 'foo*' -print0 | xargs --null --
show-limits ls -l
```

## Regreso al patio de recreo

Es hora de darle un uso (casi) práctico a Find. Crearemos un parque infantil y probaremos algo de lo que hemos aprendido.

Primero, creemos un patio de recreo con muchos subdirectorios y archivos.

```
>$ mkdir -p playground/dir-{001..100}
>$ touch playground/dir-{001..100}/file-{A..Z}
```

¡Maravíllate con el poder de la línea de comandos! Con estas dos líneas, creamos un directorio de juegos que contiene 100 subdirectorios, cada uno con 26 archivos vacíos. ¡Pruébelo con la GUI!

El método que empleamos para lograr esta magia involucró un comando familiar (mkdir), una expansión de shell exótica (llaves) y un nuevo comando, touch. Al combinar mkdir con la opción -p (que hace que mkdir cree los directorios principales de las rutas especificadas) con la expansión de llaves, pudimos crear 100 subdirectorios.

El comando touch se usa generalmente para establecer o actualizar el acceso, cambiar y modificar las horas de los archivos. Sin embargo, si un argumento de nombre de archivo es el de un archivo inexistente, se crea un archivo vacío.

En nuestro patio de recreo, creamos 100 instancias de archivo-A. Vamos a encontrarlos.

```
[~]$ find playground -type f -name 'file-A'
```

Tenga en cuenta que, a diferencia de ls, find no produce resultados ordenados. Su orden está determinado por el diseño del dispositivo de almacenamiento. Podemos confirmar que en realidad tenemos 100 instancias del archivo de esta manera.

```
[~]$ find playground -type f -name 'file-A' | wc -l
100
```

A continuación, veamos cómo buscar archivos en función de sus tiempos de modificación. Esto será útil al crear copias de seguridad u organizar archivos en orden cronológico. Para hacer esto, primero crearemos un archivo de referencia con el que compararemos los tiempos de modificación.

```
[~]$ touch playground/timestamp
```

Esto crea un archivo vacío llamado marca de tiempo y establece su hora de modificación a la hora actual. Podemos verificar esto usando otro comando útil, stat, que es una especie de versión mejorada de ls. El comando stat revela todo lo que el sistema comprende sobre un archivo y sus atributos.

```
[~]$ stat playground/timestamp
```

Si usamos touch nuevamente y luego examinamos el archivo con stat, veremos que los tiempos del archivo se han actualizado.

```
[~]$ touch playground/timestamp
[~]$ stat playground/timestamp
```

A continuación usemos find para actualizar algunos de nuestros archivos playground

```
[~]$ find playground -type f -name 'file-B' -exec touch '{}' ';'
```

esto actualiza todos los archivos playground llamados file-B. usaremos find para identificar los archivos actualizados por compaaracion de los los archivos de la referencia timestamp.

```
[~]$ find playground -type f -newer playground/timestamp
```

Los resultados contienen las 100 instancias del archivo B. Dado que realizamos un toque en todos los archivos en el patio de recreo llamado file-B después de actualizar la marca de tiempo, ahora son "más nuevos" que la marca de tiempo y, por lo tanto, se pueden identificar con la prueba -newer.

Por último, volvamos a la prueba de permisos incorrectos que realizamos anteriormente y aplíquela a la zona de juegos.

```
[~]$ find playground \( -type f -not -perm 0600 \) -or \( -type d -not -perm
0700 \)
```

Este comando enumera los 100 directorios y 2600 archivos en el área de juegos (así como la marca de tiempo y el área de juegos en sí, para un total de 2,702) porque ninguno de ellos cumple con nuestra definición de "buenos permisos". Con nuestro conocimiento de operadores y acciones, podemos agregar acciones a este comando para aplicar nuevos permisos a los archivos y directorios en nuestro patio de recreo.

```
[~]$ find playground \( -type f -not -perm 0600 -exec chmod 0600 '{}' ';' \) -or
\( -type d -not -perm 0700 -exec chmod 0700 '{}' ';' \)
```

En el día a día, es posible que nos resulte más fácil emitir dos comandos, uno para los directorios y otro para los archivos, en lugar de este gran comando compuesto, pero es bueno saber que podemos hacerlo de esta manera. El punto importante aquí es comprender cómo los operadores y las acciones se pueden utilizar juntos para realizar tareas útiles.

## Opciones de find-(buscar)

Finalmente, tenemos las opciones, que se utilizan para controlar el alcance de una búsqueda. Pueden incluirse con otras pruebas y acciones al construir expresiones de búsqueda. La tabla que sigue enumera las opciones de búsqueda más utilizadas.

> Tabla de opciones de búsqueda más utilizadas

| Opción | Descripción |
|---|---|
| -depth | Búsqueda directa para procesar los archivos de un directorio antes que el directorio en sí. Esta opción se aplica automáticamente cuando se especifica la acción -delete. |
| -maxdepth levels | Establezca el número máximo de niveles que find descenderá a un árbol de directorio al realizar pruebas y acciones. |
| -mindepth levels | Establezca el número mínimo de niveles que find descenderá a un árbol de directorio antes de aplicar pruebas y acciones. |
| -mount | Búsqueda directa para no atravesar directorios que están montados en otros sistemas de archivos. |
| -noleaf | Búsqueda directa no para optimizar su búsqueda basándose en la suposición de que está buscando en un sistema de archivos similar a Unix. Esto es necesario al escanear sistemas de archivos DOS / Windows y CD-ROM. |

# Resumiendo

> Es fácil ver que localizar es tan simple como encontrar es complicado. Ambos tienen sus usos. Tómese el tiempo para explorar las muchas características de Find. Puede, con un uso regular, mejorar su comprensión de las operaciones del sistema de archivos de Linux.