第十八章 查找文件

As we have wandered around our Linux system, one thing has become abundantly clear: a typical Linux system has a lot of files! This begs the question, “how do we find things?” We already know that the Linux file system is well organized according to conventions that have been passed down from one generation of Unix-like system to the next, but the sheer number of files can present a daunting problem. In this chapter, we will look at two tools that are used to find files on a system. These tools are:

因为我们已经浏览了 Linux 系统，所以一件事已经变得非常清楚：一个典型的 Linux 系统包含很多文件！ 这就引发了一个问题，“我们怎样查找东西？”。虽然我们已经知道 Linux 文件系统良好的组织结构，是源自 类 Unix 的操作系统代代传承的习俗。但是仅文件数量就会引起可怕的问题。在这一章中，我们将察看 两个用来在系统中查找文件的工具。这些工具是：

* locate – Find files by name
* locate – 通过名字来查找文件
* find – Search for files in a directory hierarchy
* find – 在目录层次结构中搜索文件

We will also look at a command that is often used with file search commands to process the resulting list of files:

我们也将看一个经常与文件搜索命令一起使用的命令，它用来处理搜索到的文件列表：

* xargs – Build and execute command lines from standard input
* xargs – 从标准输入生成和执行命令行

In addition, we will introduce a couple of commands to assist us in or exploration:

另外，我们将介绍两个命令来协助我们探索：

* touch – Change file times
* touch – 更改文件时间
* stat – Display file or file system status
* stat – 显示文件或文件系统状态

**locate - 查找文件的简单方法**

The locate program performs a rapid database search of pathnames and outputs every name that matches a given substring. Say, for example, we want to find all the programs with names that begin with “zip.” Since we are looking for programs, we can assume that the directory containing the programs would end with “bin/”. Therefore, we could try to use locate this way to find our files:

这个 locate 程序快速搜索路径名数据库，并且输出每个与给定字符串相匹配的文件名。比如说， 例如，我们想要找到所有名字以“zip”开头的程序。因为我们正在查找程序，可以假定包含 匹配程序的目录以”bin/”结尾。因此，我们试着以这种方式使用 locate 命令，来找到我们的文件：

[me@linuxbox ~]$ locate bin/zip

locate will search its database of pathnames and output any that contain the string “bin/zip”:

locate 命令将会搜索它的路径名数据库，输出任一个包含字符串“bin/zip”的路径名：

/usr/bin/zip

/usr/bin/zipcloak

/usr/bin/zipgrep

/usr/bin/zipinfo

/usr/bin/zipnote

/usr/bin/zipsplit

If the search requirement is not so simple, locate can be combined with other tools such as grep to design more interesting searches:

如果搜索要求没有这么简单，locate 可以结合其它工具，比如说 grep 命令，来设计更加 有趣的搜索：

[me@linuxbox ~]$ locate zip | grep bin

/bin/bunzip2

/bin/bzip2

/bin/bzip2recover

/bin/gunzip

/bin/gzip

/usr/bin/funzip

/usr/bin/gpg-zip

/usr/bin/preunzip

/usr/bin/prezip

/usr/bin/prezip-bin

/usr/bin/unzip

/usr/bin/unzipsfx

/usr/bin/zip

/usr/bin/zipcloak

/usr/bin/zipgrep

/usr/bin/zipinfo

/usr/bin/zipnote

/usr/bin/zipsplit

The locate program has been around for a number of years, and there are several different variants in common use. The two most common ones found in modern Linux distributions are slocate and mlocate, though they are usually accessed by a symbolic link named locate. The different versions of locate have overlapping options sets. Some versions include regular expression matching (which we’ll cover in an upcoming chapter) and wild card support. Check the man page for locate to determine which version of locate is installed.

这个 locate 程序已经存在了很多年了，它有几个不同的变体被普遍使用着。在现在 Linux 发行版中发现的两个最常见的变体是 slocate 和 mlocate，但是通常它们被名为 locate 的 符号链接访问。不同版本的 locate 命令拥有重复的选项集合。一些版本包括正则表达式 匹配（我们会在下一章中讨论）和通配符支持。查看 locate 命令的手册，从而确定安装了 哪个版本的 locate 程序。

Where Does The locate Database Come From?

locate 数据库来自何方？

You may notice that, on some distributions, locate fails to work just after the system is installed, but if you try again the next day, it works fine. What gives? The locate database is created by another program named updatedb. Usually, it is run periodically as a cron job; that is, a task performed at regular intervals by the cron daemon. Most systems equipped with locate run updatedb once a day. Since the database is not updated continuously, you will notice that very recent files do not show up when using locate. To overcome this, it’s possible to run the updatedb program manually by becoming the superuser and running updatedb at the prompt.

你可能注意到了，在一些发行版中，仅仅在系统安装之后，locate 不能工作， 但是如果你第二天再试一下，它就工作正常了。怎么回事呢？locate 数据库由另一个叫做 updatedb 的程序创建。通常，这个程序作为一个 cron 工作例程周期性运转；也就是说，一个任务 在特定的时间间隔内被 cron 守护进程执行。大多数装有 locate 的系统会每隔一天运行一回 updatedb 程序。因为数据库不能被持续地更新，所以当使用 locate 时，你会发现 目前最新的文件不会出现。为了克服这个问题，可以手动运行 updatedb 程序， 更改为超级用户身份，在提示符下运行 updatedb 命令。

**find - 查找文件的复杂方式**

While the locate program can find a file based solely on its name, the find program searches a given directory (and its subdirectories) for files based on a variety of attributes. We’re going to spend a lot of time with find because it has a lot of interesting features that we will see again and again when we start to cover programming concepts in later chapters.

locate 程序只能依据文件名来查找文件，而 find 程序能基于各种各样的属性， 搜索一个给定目录（以及它的子目录），来查找文件。我们将要花费大量的时间学习 find 命令，因为 它有许多有趣的特性，当我们开始在随后的章节里面讨论编程概念的时候，我们将会重复看到这些特性。

In its simplest use, find is given one or more names of directories to search. For example, to produce a list of our home directory:

find 命令的最简单使用是，搜索一个或多个目录。例如，输出我们的家目录列表。

[me@linuxbox ~]$ find ~

On most active user accounts, this will produce a large list. Since the list is sent to standard output, we can pipe the list into other programs. Let’s use wc to count the number of files:

对于最活跃的用户帐号，这将产生一张很大的列表。因为这张列表被发送到标准输出， 我们可以把这个列表管道到其它的程序中。让我们使用 wc 程序来计算出文件的数量：

[me@linuxbox ~]$ find ~ | wc -l

47068

Wow, we’ve been busy! The beauty of find is that it can be used to identify files that meet specific criteria. It does this through the (slightly strange) application of options, tests, and actions. We’ll look at the tests first.

哇，我们一直很忙！find 命令的魅力所在就是它能够被用来识别符合特定标准的文件。它通过 （有点奇怪）应用选项，测试条件，和操作来完成搜索。我们先看一下测试条件。

**Tests**

Let’s say that we want a list of directories from our search. To do this, we could add the following test:

比如说我们想要目录列表。我们可以添加以下测试条件：

[me@linuxbox ~]$ find ~ -type d | wc -l

1695

Adding the test -type d limited the search to directories. Conversely, we could have limited the search to regular files with this test:

添加测试条件-type d 限制了只搜索目录。相反地，我们使用这个测试条件来限定搜索普通文件：

[me@linuxbox ~]$ find ~ -type f | wc -l

38737

Here are the common file type tests supported by find:

这里是 find 命令支持的普通文件类型测试条件：

|  |  |
| --- | --- |
| *Table 18-1: find File Types* | |
| File Type | Description |
| b | Block special device file |
| c | Character special device file |
| d | Directory |
| f | Regular file |
| l | Symbolic link |

|  |  |
| --- | --- |
| *表18-1: find 文件类型* | |
| 文件类型 | 描述 |
| b | 块设备文件 |
| c | 字符设备文件 |
| d | 目录 |
| f | 普通文件 |
| l | 符号链接 |

We can also search by file size and filename by adding some additional tests: Let’s look for all the regular files that match the wild card pattern “\*.JPG” and are larger than one megabyte:

我们也可以通过加入一些额外的测试条件，根据文件大小和文件名来搜索：让我们查找所有文件名匹配 通配符模式“\*.JPG”和文件大小大于1M 的文件：

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

840

In this example, we add the -name test followed by the wild card pattern. Notice how we enclose it in quotes to prevent pathname expansion by the shell. Next, we add the -size test followed by the string “+1M”. The leading plus sign indicates that we are looking for files larger than the specified number. A leading minus sign would change the meaning of the string to be smaller than the specified number. No sign means, “match the value exactly.” The trailing letter “M” indicates that the unit of measurement is megabytes. The following characters may be used to specify units:

在这个例子里面，我们加入了 -name 测试条件，后面跟通配符模式。注意，我们把它用双引号引起来， 从而阻止 shell 展开路径名。紧接着，我们加入 -size 测试条件，后跟字符串“+1M”。开头的加号表明 我们正在寻找文件大小大于指定数的文件。若字符串以减号开头，则意味着查找小于指定数的文件。 若没有符号意味着“精确匹配这个数”。结尾字母“M”表明测量单位是兆字节。下面的字符可以 被用来指定测量单位：

|  |  |
| --- | --- |
| *Table 18-2: find Size Units* | |
| Character | Unit |
| b | 512 byte blocks. This is the default if no unit is specified. |
| c | Bytes |
| w | Two byte words |
| k | Kilobytes (Units of 1024 bytes) |
| M | Megabytes (Units of 1048576 bytes) |
| G | Gigabytes (Units of 1073741824 bytes) |

|  |  |
| --- | --- |
| *表18-2: find 大小单位* | |
| 字符 | 单位 |
| b | 512 个字节块。如果没有指定单位，则这是默认值。 |
| c | 字节 |
| w | 两个字节的字 |
| k | 千字节(1024个字节单位) |
| M | 兆字节(1048576个字节单位) |
| G | 千兆字节(1073741824个字节单位) |

find supports a large number of different tests. Below is a rundown of the common ones. Note that in cases where a numeric argument is required, the same “+” and “-” notation discussed above can be applied:

find 命令支持大量不同的测试条件。下表是列出了一些常见的测试条件。请注意，在需要数值参数的 情况下，可以应用以上讨论的“+”和”-“符号表示法：

|  |  |
| --- | --- |
| *Table 18-3: find Tests* | |
| Test | Description |
| -cmin n | Match files or directories whose content or attributes were last modified exactly n minutes ago. To specify less than n minutes ago, use -n and to specify more than n minutes ago, use +n. |
| -cnewer file | Match files or directories whose contents or attributes were last modified more recently than those of file. |
| -ctime n | Match files or directories whose contents or attributes were last modified n\*24 hours ago. |
| -empty | Match empty files and directories. |
| -group name | Match file or directories belonging to group. group may be expressed as either a group name or as a numeric group ID. |
| -iname pattern | Like the -name test but case insensitive. |
| -inum n | Match files with inode number n. This is helpful for finding all the hard links to a particular inode. |
| -mmin n | Match files or directories whose contents were modified n minutes ago. |
| -mtime n | Match files or directories whose contents were modified n\*24 hours ago. |
| -name pattern | Match files and directories with the specified wild card pattern. |
| -newer file | Match files and directories whose contents were modified more recently than the specified file. This is very useful when writing shell scripts that perform file backups. Each time you make a backup, update a file (such as a log), then use find to determine which files that have changed since the last update. |
| -nouser | Match file and directories that do not belong to a valid user. This can be used to find files belonging to deleted accounts or to detect activity by attackers. |
| -nogroup | Match files and directories that do not belong to a valid group. |
| -perm mode | Match files or directories that have permissions set to the specified mode. mode may be expressed by either octal or symbolic notation. |
| -samefile name | Similar to the -inum test. Matches files that share the same inode number as file name. |
| -size n | Match files of size n. |
| -type c | Match files of type c. |
| -user name | Match files or directories belonging to user name. The user may be expressed by a user name or by a numeric user ID. |

|  |  |
| --- | --- |
| *表18-3: find 测试条件* | |
| 测试条件 | 描述 |
| -cmin n | 匹配的文件和目录的内容或属性最后修改时间正好在 n 分钟之前。 指定少于 n 分钟之前，使用 -n，指定多于 n 分钟之前，使用 +n。 |
| -cnewer file | 匹配的文件和目录的内容或属性最后修改时间早于那些文件。 |
| -ctime n | 匹配的文件和目录的内容和属性最后修改时间在 n\*24小时之前。 |
| -empty | 匹配空文件和目录。 |
| -group name | 匹配的文件和目录属于一个组。组可以用组名或组 ID 来表示。 |
| -iname pattern | 就像-name 测试条件，但是不区分大小写。 |
| -inum n | 匹配的文件的 inode 号是 n。这对于找到某个特殊 inode 的所有硬链接很有帮助。 |
| -mmin n | 匹配的文件或目录的内容被修改于 n 分钟之前。 |
| -mtime n | 匹配的文件或目录的内容被修改于 n\*24小时之前。 |
| -name pattern | 用指定的通配符模式匹配的文件和目录。 |
| -newer file | 匹配的文件和目录的内容早于指定的文件。当编写 shell 脚本，做文件备份时，非常有帮助。 每次你制作一个备份，更新文件（比如说日志），然后使用 find 命令来决定自从上次更新，哪一个文件已经更改了。 |
| -nouser | 匹配的文件和目录不属于一个有效用户。这可以用来查找 属于删除帐户的文件或监测攻击行为。 |
| -nogroup | 匹配的文件和目录不属于一个有效的组。 |
| -perm mode | 匹配的文件和目录的权限已经设置为指定的 mode。mode 可以用 八进制或符号表示法。 |
| -samefile name | 相似于-inum 测试条件。匹配和文件 name 享有同样 inode 号的文件。 |
| -size n | 匹配的文件大小为 n。 |
| -type c | 匹配的文件类型是 c。 |
| -user name | 匹配的文件或目录属于某个用户。这个用户可以通过用户名或用户 ID 来表示。 |

This is not a complete list. The find man page has all the details.

这不是一个完整的列表。find 命令手册有更详细的说明。

**操作符**

Even with all the tests that find provides, we may still need a better way to describe the logical relationships between the tests. For example, what if we needed to determine if all the files and subdirectories in a directory had secure permissions? We would look for all the files with permissions that are not 0600 and the directories with permissions that are not 0700. Fortunately, find provides a way to combine tests using logical operators to create more complex logical relationships. To express the aforementioned test, we could do this:

即使拥有了 find 命令提供的所有测试条件，我们还需要一个更好的方式来描述测试条件之间的逻辑关系。例如， 如果我们需要确定是否一个目录中的所有的文件和子目录拥有安全权限，怎么办呢？ 我们可以查找权限不是0600的文件和权限不是0700的目录。幸运地是，find 命令提供了 一种方法来结合测试条件，通过使用逻辑操作符来创建更复杂的逻辑关系。 为了表达上述的测试条件，我们可以这样做：

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

Yikes! That sure looks weird. What is all this stuff? Actually, the operators are not that complicated once you get to know them. Here is the list:

呀！这的确看起来很奇怪。这些是什么东西？实际上，这些操作符没有那么复杂，一旦你知道了它们的原理。 这里是操作符列表：

|  |  |
| --- | --- |
| *Table 18-4: find Logical Operators* | |
| Operator | Description |
| -and | Match if the tests on both sides of the operator are true. May be shortened to -a. Note that when no operator is present, -and is implied by default. |
| -or | Match if a test on either side of the operator is true. May be shortened to -o. |
| -not | Match if the test following the operator is false. May be abbreviated with an exclamation point (!). |
| () | Groups tests and operators together to form larger expressions. This is used to control the precedence of the logical evaluations. By default, find evaluates from left to right. It is often necessary to override the default evaluation order to obtain the desired result. Even if not needed, it is helpful sometimes to include the grouping characters to improve readability of the command. Note that since the parentheses characters have special meaning to the shell, they must be quoted when using them on the command line to allow them to be passed as arguments to find. Usually the backslash character is used to escape them. |

|  |  |
| --- | --- |
| *表18-4: find 命令的逻辑操作符* | |
| 操作符 | 描述 |
| -and | 如果操作符两边的测试条件都是真，则匹配。可以简写为 -a。 注意若没有使用操作符，则默认使用 -and。 |
| -or | 若操作符两边的任一个测试条件为真，则匹配。可以简写为 -o。 |
| -not | 若操作符后面的测试条件是真，则匹配。可以简写为一个感叹号（!）。 |
| () | 把测试条件和操作符组合起来形成更大的表达式。这用来控制逻辑计算的优先级。 默认情况下，find 命令按照从左到右的顺序计算。经常有必要重写默认的求值顺序，以得到期望的结果。 即使没有必要，有时候包括组合起来的字符，对提高命令的可读性是很有帮助的。注意 因为圆括号字符对于 shell 来说有特殊含义，所以在命令行中使用它们的时候，它们必须 用引号引起来，才能作为实参传递给 find 命令。通常反斜杠字符被用来转义圆括号字符。 |

With this list of operators in hand, let’s deconstruct our find command. When viewed from the uppermost level, we see that our tests are arranged as two groupings separated by an -or operator:

通过这张操作符列表，我们重建 find 命令。从最外层看，我们看到测试条件被分为两组，由一个 -or 操作符分开：

( expression 1 ) -or ( expression 2 )

This makes sense, since we are searching for files with a certain set of permissions and for directories with a different set. If we are looking for both files and directories, why do we use -or instead of -and? Because as find scans through the files and directories, each one is evaluated to see if it matches the specified tests. We want to know if it is either a file with bad permissions or a directory with bad permissions. It can’t be both at the same time. So if we expand the grouped expressions, we can see it this way:

这很有意义，因为我们正在搜索具有不同权限集合的文件和目录。如果我们文件和目录两者都查找， 那为什么要用 -or 来代替 -and 呢？因为 find 命令扫描文件和目录时，会计算每一个对象，看看它是否 匹配指定的测试条件。我们想要知道它是具有错误权限的文件还是有错误权限的目录。它不可能同时符合这 两个条件。所以如果展开组合起来的表达式，我们能这样解释它：

( file with bad perms ) -or ( directory with bad perms )

Our next challenge is how to test for “bad permissions.” How do we do that? Actually we don’t. What we will test for is “not good permissions,” since we know what “good permissions” are. In the case of files, we define good as 0600 and for directories, as 0711. The expression that will test files for “not good” permissions is:

下一个挑战是怎样来检查“错误权限”，这个怎样做呢？我们不从这个角度做。我们将测试 “不是正确权限”，因为我们知道什么是“正确权限”。对于文件，我们定义正确权限为0600， 目录则为0711。测试具有“不正确”权限的文件表达式为：

-type f -and -not -perms 0600

and for directories:

对于目录，表达式为：

-type d -and -not -perms 0700

As noted in the table of operators above, the -and operator can be safely removed, since it is implied by default. So if we put this all back together, we get our final command:

正如上述操作符列表中提到的，这个-and 操作符能够被安全地删除，因为它是默认使用的操作符。 所以如果我们把这两个表达式连起来，就得到最终的命令：

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

However, since the parentheses have special meaning to the shell, we must escape them to prevent the shell from trying to interpret them. Preceding each one with a backslash character does the trick.

然而，因为圆括号对于 shell 有特殊含义，我们必须转义它们，来阻止 shell 解释它们。在圆括号字符 之前加上一个反斜杠字符来转义它们。

There is another feature of logical operators that is important to understand. Let’s say that we have two expressions separated by a logical operator:

逻辑操作符的另一个特性要重点理解。比方说我们有两个由逻辑操作符分开的表达式：

expr1 -operator expr2

In all cases, expr1 will always be performed; however the operator will determine if expr2 is performed. Here’s how it works:

在所有情况下，总会执行表达式 expr1；然而由操作符来决定是否执行表达式 expr2。这里 列出了它是怎样工作的：

|  |  |  |
| --- | --- | --- |
| *Table 18-5: find AND/OR Logic* | | |
| Results of expr1 | Operator | expr2 is... |
| True | -and | Always performed |
| False | -and | Never performed |
| Ture | -or | Never performed |
| False | -or | Always performed |

|  |  |  |
| --- | --- | --- |
| *表18-5: find AND/OR 逻辑* | | |
| expr1 的结果 | 操作符 | expr2 is... |
| 真 | -and | 总要执行 |
| 假 | -and | 从不执行 |
| 真 | -or | 从不执行 |
| 假 | -or | 总要执行 |

Why does this happen? It’s done to improve performance. Take -and, for example. We know that the expression expr1 -and expr2 cannot be true if the result of expr1 is false, so there is no point in performing expr2. Likewise, if we have the expression expr1 -or expr2 and the result of expr1 is true, there is no point in performing expr2, as we already know that the expression expr1 -or expr2 is true. OK, so it helps it go faster. Why is this important? It’s important because we can rely on this behavior to control how actions are performed, as we shall soon see..

为什么这会发生呢？这样做是为了提高性能。以 -and 为例，我们知道表达式 expr1 -and expr2 不能为真，如果表达式 expr1的结果为假，所以没有必要执行 expr2。同样地，如果我们有表达式 expr1 -or expr2，并且表达式 expr1的结果为真，那么就没有必要执行 expr2，因为我们已经知道 表达式 expr1 -or expr2 为真。好，这样会执行快一些。为什么这个很重要？ 它很重要是因为我们能依靠这种行为来控制怎样来执行操作。我们会很快看到…

**预定义的操作**

Let’s get some work done! Having a list of results from our find command is useful, but what we really want to do is act on the items on the list. Fortunately, find allows actions to be performed based on the search results. There are a set of predefined actions and several ways to apply user-defined actions. First let’s look at a few of the predefined actions:

让我们做一些工作吧！从 find 命令得到的结果列表很有用处，但是我们真正想要做的事情是操作列表 中的某些条目。幸运地是，find 命令允许基于搜索结果来执行操作。有许多预定义的操作和几种方式来 应用用户定义的操作。首先，让我们看一下几个预定义的操作：

|  |  |
| --- | --- |
| *Table 18-6: Predefined find Actions* | |
| Action | Description |
| -delete | Delete the currently matching file. |
| -ls | Perform the equivalent of ls -dils on the matching file. Output is sent to standard output. |
| -print | Output the full pathname of the matching file to standard output. This is the default action if no other action is specified. |
| -quit | Quit once a match has been made. |

|  |  |
| --- | --- |
| *表18-6: 几个预定义的 find 命令操作* | |
| 操作 | 描述 |
| -delete | 删除当前匹配的文件。 |
| -ls | 对匹配的文件执行等同的 ls -dils 命令。并将结果发送到标准输出。 |
| -print | 把匹配文件的全路径名输送到标准输出。如果没有指定其它操作，这是 默认操作。 |
| -quit | 一旦找到一个匹配，退出。 |

As with the tests, there are many more actions. See the find man page for full details. In our very first example, we did this:

和测试条件一样，还有更多的操作。查看 find 命令手册得到更多细节。在第一个例子里， 我们这样做：

find ~

which produced a list of every file and subdirectory contained within our home directory. It produced a list because the -print action is implied if no other action is specified. Thus our command could also be expressed as:

这个命令输出了我们家目录中包含的每个文件和子目录。它会输出一个列表，因为会默认使用-print 操作 ，如果没有指定其它操作的话。因此我们的命令也可以这样表述：

find ~ -print

We can use find to delete files that meet certain criteria. For example, to delete files that have the file extension “.BAK” (which is often used to designate backup files), we could use this command:

我们可以使用 find 命令来删除符合一定条件的文件。例如，来删除扩展名为“.BAK”（这通常用来指定备份文件） 的文件，我们可以使用这个命令：

find ~ -type f -name '\*.BAK' -delete

In this example, every file in the user’s home directory (and its subdirectories) is searched for filenames ending in .BAK. When they are found, they are deleted.

在这个例子里面，用户家目录（和它的子目录）下搜索每个以.BAK 结尾的文件名。当找到后，就删除它们。

Warning: It should go without saying that you should use extreme caution when using the -delete action. Always test the command first by substituting the -print action for -delete to confirm the search results.

警告：当使用 -delete 操作时，不用说，你应该格外小心。首先测试一下命令， 用 -print 操作代替 -delete，来确认搜索结果。

Before we go on, let’s take another look at how the logical operators affect actions. Consider the following command:

在我们继续之前，让我们看一下逻辑运算符是怎样影响操作的。考虑以下命令：

find ~ -type f -name '\*.BAK' -print

As we have seen, this command will look for every regular file (-type f) whose name ends with .BAK (-name ‘\*.BAK’) and will output the relative pathname of each matching file to standard output (-print). However, the reason the command performs the way it does is determined by the logical relationships between each of the tests and actions. Remember, there is, by default, an implied -and relationship between each test and action. We could also express the command this way to make the logical relationships easier to see:

正如我们所见到的，这个命令会查找每个文件名以.BAK (-name ‘\*.BAK’) 结尾的普通文件 (-type f)， 并把每个匹配文件的相对路径名输出到标准输出 (-print)。然而，此命令按这个方式执行的原因，是 由每个测试和操作之间的逻辑关系决定的。记住，在每个测试和操作之间会默认应用 -and 逻辑运算符。 我们也可以这样表达这个命令，使逻辑关系更容易看出：

find ~ -type f -and -name '\*.BAK' -and -print

With our command fully expressed, let’s look at how the logical operators affect its execution:

当命令被充分表达之后，让我们看看逻辑运算符是如何影响其执行的：

|  |  |
| --- | --- |
| Test/Action | Is Performed Only If... |
| -print | -type f and -name '\*.BAK' are true |
| -name ‘\*.BAK’ | -type f is true |
| -type f | Is always performed, since it is the first test/action in an -and relationship. |

|  |  |
| --- | --- |
| 测试／行为 | 只有...的时候，才被执行 |
| -print | 只有 -type f and -name '\*.BAK'为真的时候 |
| -name ‘\*.BAK’ | 只有 -type f 为真的时候 |
| -type f | 总是被执行，因为它是与 -and 关系中的第一个测试／行为。 |

Since the logical relationship between the tests and actions determines which of them are performed, we can see that the order of the tests and actions is important. For instance, if we were to reorder the tests and actions so that the -print action was the first one, the command would behave much differently:

因为测试和行为之间的逻辑关系决定了哪一个会被执行，我们知道测试和行为的顺序很重要。例如， 如果我们重新安排测试和行为之间的顺序，让 -print 行为是第一个，那么这个命令执行起来会截然不同：

find ~ -print -and -type f -and -name '\*.BAK'

This version of the command will print each file (the -print action always evaluates to true) and then test for file type and the specified file extension.

这个版本的命令会打印出每个文件（-print 行为总是为真），然后测试文件类型和指定的文件扩展名。

**用户定义的行为**

In addition to the predefined actions, we can also invoke arbitrary commands. The traditional way of doing this is with the -exec action. This action works like this:

除了预定义的行为之外，我们也可以唤醒随意的命令。传统方式是通过 -exec 行为。这个 行为像这样工作：

-exec command {} ;

where command is the name of a command, {} is a symbolic representation of the current pathname and the semicolon is a required delimiter indicating the end of the command. Here’s an example of using -exec to act like the -delete action discussed earlier:

这里的 command 就是指一个命令的名字，{}是当前路径名的符号表示，分号是要求的界定符 表明命令结束。这里是一个使用 -exec 行为的例子，其作用如之前讨论的 -delete 行为：

-exec rm '{}' ';'

Again, since the brace and semicolon characters have special meaning to the shell, they must be quoted or escaped.

重述一遍，因为花括号和分号对于 shell 有特殊含义，所以它们必须被引起来或被转义。

It’s also possible to execute a user defined action interactively. By using the -ok action in place of -exec, the user is prompted before execution of each specified command:

也有可能交互式地执行一个用户定义的行为。通过使用 -ok 行为来代替 -exec，在执行每个指定的命令之前， 会提示用户：

find ~ -type f -name 'foo\*' -ok ls -l '{}' ';'

< ls ... /home/me/bin/foo > ? y

-rwxr-xr-x 1 me me 224 2007-10-29 18:44 /home/me/bin/foo

< ls ... /home/me/foo.txt > ? y

-rw-r--r-- 1 me me 0 2008-09-19 12:53 /home/me/foo.txt

In this example, we search for files with names starting with the string “foo” and execute the command ls -l each time one is found. Using the -ok action prompts the user before the ls command is executed.

在这个例子里面，我们搜索以字符串“foo”开头的文件名，并且对每个匹配的文件执行 ls -l 命令。 使用 -ok 行为，会在 ls 命令执行之前提示用户。

**提高效率**

When the -exec action is used, it launches a new instance of the specified command each time a matching file is found. There are times when we might prefer to combine all of the search results and launch a single instance of the command. For example, rather than executing the commands like this:

当 -exec 行为被使用的时候，若每次找到一个匹配的文件，它会启动一个新的指定命令的实例。 我们可能更愿意把所有的搜索结果结合起来，再运行一个命令的实例。例如，而不是像这样执行命令：

ls -l file1

ls -l file2

we may prefer to execute it this way:

我们更喜欢这样执行命令：

ls -l file1 file2

thus causing the command to be executed only one time rather than multiple times. There are two ways we can do this. The traditional way, using the external command xargs and the alternate way, using a new feature in find itself. We’ll talk about the alternate way first.

这样就导致命令只被执行一次而不是多次。有两种方法可以这样做。传统方式是使用外部命令 xargs，另一种方法是，使用 find 命令自己的一个新功能。我们先讨论第二种方法。

By changing the trailing semicolon character to a plus sign, we activate the ability of find to combine the results of the search into an argument list for a single execution of the desired command. Going back to our example, this:

通过把末尾的分号改为加号，就激活了 find 命令的一个功能，把搜索结果结合为一个参数列表， 然后执行一次所期望的命令。再看一下之前的例子，这个：

find ~ -type f -name 'foo\*' -exec ls -l '{}' ';'

-rwxr-xr-x 1 me me 224 2007-10-29 18:44 /home/me/bin/foo

-rw-r--r-- 1 me me 0 2008-09-19 12:53 /home/me/foo.txt

will execute ls each time a matching file is found. By changing the command to:

会执行 ls 命令，每次找到一个匹配的文件。把命令改为：

find ~ -type f -name 'foo\*' -exec ls -l '{}' +

-rwxr-xr-x 1 me me 224 2007-10-29 18:44 /home/me/bin/foo

-rw-r--r-- 1 me me 0 2008-09-19 12:53 /home/me/foo.txt

we get the same results, but the system only has to execute the ls command once.

虽然我们得到一样的结果，但是系统只需要执行一次 ls 命令。

**xargs**

The xargs command performs an interesting function. It accepts input from standard input and converts it into an argument list for a specified command. With our example, we would use it like this:

这个 xargs 命令会执行一个有趣的函数。它从标准输入接受输入，并把输入转换为一个特定命令的 参数列表。对于我们的例子，我们可以这样使用它：

find ~ -type f -name 'foo\*' -print | xargs ls -l

-rwxr-xr-x 1 me me 224 2007-10-29 18:44 /home/me/bin/foo

-rw-r--r-- 1 me me 0 2008-09-19 12:53 /home/me/foo.txt

Here we see the output of the find command piped into xargs which, in turn, constructs an argument list for ls command and then executes it.

这里我们看到 find 命令的输出被管道到 xargs 命令，反过来，xargs 会为 ls 命令构建 参数列表，然后执行 ls 命令。

Note: While the number of arguments that can be placed into a command line is quite large, it’s not unlimited. It is possible to create commands that are too long for the shell to accept. When a command line exceeds the maximum length supported by the system, xargs executes the specified command with the maximum number of arguments possible and then repeats this process until standard input is exhausted. To see the maximum size of the command line, execute xargs with the –show-limits option.

注意：当被放置到命令行中的参数个数相当大时，参数个数是有限制的。有可能创建的命令 太长以至于 shell 不能接受。当命令行超过系统支持的最大长度时，xargs 会执行带有最大 参数个数的指定命令，然后重复这个过程直到耗尽标准输入。执行带有 –show–limits 选项 的 xargs 命令，来查看命令行的最大值。

Dealing With Funny Filenames

处理古怪的文件名

Unix-like systems allow embedded spaces (and even newlines!) in filenames. This causes problems for programs like xargs that construct argument lists for other programs. An embedded space will be treated as a delimiter and the resulting command will interpret each space-separated word as a separate argument. To overcome this, find and xarg allow the optional use of a null character as argument separator. A null character is defined in ASCII as the character represented by the number zero (as opposed to, for example, the space character, which is defined in ASCII as the character represented by the number 32). The find command provides the action -print0, which produces null separated output, and the xargs command has the –null option, which accepts null separated input. Here’s an example:

类 Unix 的系统允许在文件名中嵌入空格（甚至换行符）。这就给一些程序，如为其它 程序构建参数列表的 xargs 程序，造成了问题。一个嵌入的空格会被看作是一个界定符，生成的 命令会把每个空格分离的单词解释为单独的参数。为了解决这个问题，find 命令和 xarg 程序 允许可选择的使用一个 null 字符作为参数分隔符。一个 null 字符被定义在 ASCII 码中，由数字 零来表示（相反的，例如，空格字符在 ASCII 码中由数字32表示）。find 命令提供的 -print0 行为， 则会产生由 null 字符分离的输出，并且 xargs 命令有一个 –null 选项，这个选项会接受由 null 字符 分离的输入。这里有一个例子：

|  |  |
| --- | --- |
| find ~ -iname ‘\*.jpg’ -print0 | xargs –null ls -l |

Using this technique, we can ensure that all files, even those containing embedded spaces in their names, are handled correctly.

使用这项技术，我们可以保证所有文件，甚至那些文件名中包含空格的文件，都能被正确地处理。

**返回操练场**

It’s time to put find to some (almost) practical use. We’ll create a playground and try out some of what we have learned.

到实际使用 find 命令的时候了。我们将会创建一个操练场，来实践一些我们所学到的知识。

First, let’s create a playground with lots of subdirectories and files:

首先，让我们创建一个包含许多子目录和文件的操练场：

[me@linuxbox ~]$ mkdir -p playground/dir-{00{1..9},0{10..99},100}

[me@linuxbox ~]$ touch playground/dir-{00{1..9},0{10..99},100}/file-{A..Z}

Marvel in the power of the command line! With these two lines, we created a playground directory containing one hundred subdirectories each containing twenty-six empty files. Try that with the GUI!

惊叹于命令行的强大功能！只用这两行，我们就创建了一个包含一百个子目录，每个子目录中 包含了26个空文件的操练场。试试用 GUI 来创建它！

The method we employed to accomplish this magic involved a familiar command (mkdir), an exotic shell expansion (braces) and a new command, touch. By combining mkdir with the -p option (which causes mkdir to create the parent directories of the specified paths) with brace expansion, we were able to create one hundred directories.

我们用来创造这个奇迹的方法中包含一个熟悉的命令（mkdir），一个奇异的 shell 扩展（大括号） 和一个新命令，touch。通过结合 mkdir 命令和-p 选项（导致 mkdir 命令创建指定路径的父目录），以及 大括号展开，我们能够创建一百个目录。

The touch command is usually used to set or update the access, change, and modify times of files. However, if a filename argument is that of a nonexistent file, an empty file is created.

这个 touch 命令通常被用来设置或更新文件的访问，更改，和修改时间。然而，如果一个文件名参数是一个 不存在的文件，则会创建一个空文件。

In our playground, we created one hundred instances of a file named file-A. Let’s find them:

在我们的操练场中，我们创建了一百个名为 file-A 的文件实例。让我们找到它们：

[me@linuxbox ~]$ find playground -type f -name 'file-A'

Note that unlike ls, find does not produce results in sorted order. Its order is determined by the layout of the storage device. To confirm that we actually have one hundred instances of the file we can confirm it this way:

注意不同于 ls 命令，find 命令的输出结果是无序的。其顺序由存储设备的布局决定。为了确定实际上 我们拥有一百个此文件的实例，我们可以用这种方式来确认：

[me@linuxbox ~]$ find playground -type f -name 'file-A' | wc -l

Next, let’s look at finding files based on their modification times. This will be helpful when creating backups or organizing files in chronological order. To do this, we will first create a reference file against which we will compare modification time:

下一步，让我们看一下基于文件的修改时间来查找文件。当创建备份文件或者以年代顺序来 组织文件的时候，这会很有帮助。为此，首先我们将创建一个参考文件，我们将与其比较修改时间：

[me@linuxbox ~]$ touch playground/timestamp

This creates an empty file named timestamp and sets its modification time to the current time. We can verify this by using another handy command, stat, which is a kind of souped-up version of ls. The stat command reveals all that the system understands about a file and its attributes:

这个创建了一个空文件，名为 timestamp，并且把它的修改时间设置为当前时间。我们能够验证 它通过使用另一个方便的命令，stat，是一款加大马力的 ls 命令版本。这个 stat 命令会展示系统对 某个文件及其属性所知道的所有信息：

[me@linuxbox ~]$ stat playground/timestamp

File: 'playground/timestamp'

Size: 0 Blocks: 0 IO Block: 4096 regular empty file

Device: 803h/2051d Inode: 14265061 Links: 1

Access: (0644/-rw-r--r--) Uid: ( 1001/ me) Gid: ( 1001/ me)

Access: 2008-10-08 15:15:39.000000000 -0400

Modify: 2008-10-08 15:15:39.000000000 -0400

Change: 2008-10-08 15:15:39.000000000 -0400

If we touch the file again and then examine it with stat, we will see that the file’s times have been updated.

如果我们再次 touch 这个文件，然后用 stat 命令检测它，我们会发现所有文件的时间已经更新了。

[me@linuxbox ~]$ touch playground/timestamp

[me@linuxbox ~]$ stat playground/timestamp

File: 'playground/timestamp'

Size: 0 Blocks: 0 IO Block: 4096 regular empty file

Device: 803h/2051d Inode: 14265061 Links: 1

Access: (0644/-rw-r--r--) Uid: ( 1001/ me) Gid: ( 1001/ me)

Access: 2008-10-08 15:23:33.000000000 -0400

Modify: 2008-10-08 15:23:33.000000000 -0400

Change: 2008-10-08 15:23:33.000000000 -0400

Next, let’s use find to update some of our playground files:

下一步，让我们使用 find 命令来更新一些操练场中的文件：

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

This updates all files in the playground named file-B. Next we’ll use find to identify the updated files by comparing all the files to the reference file timestamp:

这会更新操练场中所有名为 file-B 的文件。接下来我们会使用 find 命令来识别已更新的文件， 通过把所有文件与参考文件 timestamp 做比较：

[me@linuxbox ~]$ find playground -type f -newer playground/timestamp

The results contain all one hundred instances of file-B. Since we performed a touch on all the files in the playground named file-B after we updated timestamp, they are now “newer” than timestamp and thus can be identified with the -newer test.

搜索结果包含所有一百个文件 file-B 的实例。因为我们在更新了文件 timestamp 之后， touch 了操练场中名为 file-B 的所有文件，所以现在它们“新于”timestamp 文件，因此能被用 -newer 测试条件识别出来。

Finally, let’s go back to the bad permissions test we performed earlier and apply it to playground:

最后，让我们回到之前那个错误权限的例子中，把它应用于操练场里：

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

This command lists all one hundred directories and twenty-six hundred files in playground (as well as timestamp and playground itself, for a total of 2702) because none of them meets our definition of “good permissions.” With our knowledge of operators and actions, we can add actions to this command to apply new permissions to the files and directories in our playground:

这个命令列出了操练场中所有一百个目录和二百六十个文件（还有 timestamp 和操练场本身，共 2702 个） ，因为没有一个符合我们“正确权限”的定义。通过对运算符和行为知识的了解，我们可以给这个命令 添加行为，对实战场中的文件和目录应用新的权限。

[me@linuxbox ~]$ find playground \( -type f -not -perm 0600 -exec chmod 0600 '{}' ';' \)

-or \( -type d -not -perm 0711 -exec chmod 0700 '{}' ';' \)

On a day-to-day basis, we might find it easier to issue two commands, one for the directories and one for the files, rather than this one large compound command, but it’s nice to know that we can do it this way. The important point here is to understand how the operators and actions can be used together to perform useful tasks.

在日常的基础上，我们可能发现运行两个命令会比较容易一些，一个操作目录，另一个操作文件， 而不是这一个长长的复合命令，但是很高兴知道，我们能这样执行命令。这里最重要的一点是要 理解怎样把操作符和行为结合起来使用，来执行有用的任务。

**选项**

Finally, we have the options. The options are used to control the scope of a find search. They may be included with other tests and actions when constructing find expressions. Here is a list of the most commonly used ones:

最后，我们有这些选项。这些选项被用来控制 find 命令的搜索范围。当构建 find 表达式的时候， 它们可能被其它的测试条件和行为包含：

|  |  |
| --- | --- |
| *Table 18-7: find Options* | |
| Option | Description |
| -depth | Direct find to process a directory’s files before the directory itself. This option is automatically applied when the -delete action is specified. |
| -maxdepth levels | Set the maximum number of levels that find will descend into a directory tree when performing tests and actions. |
| -mindepth levels | Set the minimum number of levels that find will descend into a directory tree before applying tests and actions. |
| -mount | Direct find not to traverse directories that are mounted on other file systems. |
| -noleaf | Direct find not to optimize its search based on the assumption that it is searching a Unix-like file system. This is needed when scanning DOS/Windows file systems and CD-ROMs. |

|  |  |
| --- | --- |
| *表 18-7: find 命令选项* | |
| 选项 | 描述 |
| -depth | 指导 find 程序先处理目录中的文件，再处理目录自身。当指定-delete 行为时，会自动 应用这个选项。 |
| -maxdepth levels | 当执行测试条件和行为的时候，设置 find 程序陷入目录树的最大级别数 |
| -mindepth levels | 在应用测试条件和行为之前，设置 find 程序陷入目录数的最小级别数。 |
| -mount | 指导 find 程序不要搜索挂载到其它文件系统上的目录。 |
| -noleaf | 指导 find 程序不要基于搜索类 Unix 的文件系统做出的假设，来优化它的搜索。 |