第二十 正则表达式

In the next few chapters, we are going to look at tools used to manipulate text. As we have seen, text data plays an important role on all Unix-like systems, such as Linux. But before we can fully appreciate all of the features offered by these tools, we have to first examine a technology that is frequently associated with the most sophisticated uses of these tools — regular expressions.

接下来的几章中，我们将会看一下一些用来操作文本的工具。正如我们所见到的，在类 Unix 的 操作系统中，比如 Linux 中，文本数据起着举足轻重的作用。但是在我们能完全理解这些工具提供的 所有功能之前，我们不得不先看看，经常与这些工具的高级使用相关联的一门技术——正则表达式。

As we have navigated the many features and facilities offered by the command line, we have encountered some truly arcane shell features and commands, such as shell expansion and quoting, keyboard shortcuts, and command history, not to mention the vi editor. Regular expressions continue this “tradition” and may be (arguably) the most arcane feature of them all. This is not to suggest that the time it takes to learn about them is not worth the effort. Quite the contrary. A good understanding will enable us to perform amazing feats, though their full value may not be immediately apparent. What Are Regular Expressions?

我们已经浏览了许多由命令行提供的功能和工具，我们遇到了一些真正神秘的 shell 功能和命令， 比如 shell 展开和引用，键盘快捷键，和命令历史，更不用说 vi 编辑器了。正则表达式延续了 这种“传统”，而且有可能（备受争议地）是其中最神秘的功能。这并不是说花费时间来学习它们 是不值得的，而是恰恰相反。虽然它们的全部价值可能不能立即显现，但是较强理解这些功能 使我们能够表演令人惊奇的技艺。什么是正则表达式？

Simply put, regular expressions are symbolic notations used to identify patterns in text. In some ways, they resemble the shell’s wildcard method of matching file and pathnames, but on a much grander scale. Regular expressions are supported by many command line tools and by most programming languages to facilitate the solution of text manipulation problems. However, to further confuse things, not all regular expressions are the same; they vary slightly from tool to tool and from programming language to language. For our discussion, we will limit ourselves to regular expressions as described in the POSIX standard (which will cover most of the command line tools), as opposed to many programming languages (most notably Perl), which use slightly larger and richer sets of notations.

简而言之，正则表达式是一种符号表示法，被用来识别文本模式。在某种程度上，它们与匹配 文件和路径名的 shell 通配符比较相似，但其规模更庞大。许多命令行工具和大多数的编程语言 都支持正则表达式，以此来帮助解决文本操作问题。然而，并不是所有的正则表达式都是一样的， 这就进一步混淆了事情；不同工具以及不同语言之间的正则表达式都略有差异。我们将会限定 POSIX 标准中描述的正则表达式（其包括了大多数的命令行工具），供我们讨论， 与许多编程语言（最著名的 Perl 语言）相反，它们使用了更多和更丰富的符号集。

**grep**

The main program we will use to work with regular expressions is our old pal, grep. The name “grep” is actually derived from the phrase “global regular expression print,” so we can see that grep has something to do with regular expressions. In essence, grep searches text files for the occurrence of a specified regular expression and outputs any line containing a match to standard output.

我们将使用的主要程序是我们的老朋友，grep 程序，它会用到正则表达式。实际上，“grep”这个名字 来自于短语“global regular expression print”，所以我们能看出 grep 程序和正则表达式有关联。 本质上，grep 程序会在文本文件中查找一个指定的正则表达式，并把匹配行输出到标准输出。

So far, we have used grep with fixed strings, like so:

到目前为止，我们已经使用 grep 程序查找了固定的字符串，就像这样:

[me@linuxbox ~]$ ls /usr/bin | grep zip

This will list all the files in the /usr/bin directory whose names contain the substring “zip”.

这个命令会列出，位于目录 /usr/bin 中，文件名中包含子字符串“zip”的所有文件。

The grep program accepts options and arguments this way:

这个 grep 程序以这样的方式来接受选项和参数：

grep [options] regex [file...]

where regex is a regular expression.

这里的 regx 是指一个正则表达式。

Here is a list of the commonly used grep options:

这是一个常用的 grep 选项列表：

|  |  |
| --- | --- |
| *Table20-1: grep Options* | |
| Option | Description |
| -i | Ignore case. Do not distinguish between upper and lower case characters. May also be specified --ignore-case. |
| -v | Invert match. Normally, grep prints lines that contain a match. This option causes grep to print every line that does not contain a match. May also be specified --invert-match. |
| -c | Print the number of matches (or non-matches if the -v option is also specified) instead of the lines themselves. May also be specified --count. |
| -l | Print the name of each file that contains a match instead of the lines themselves. May also be specified --files-with-matches. |
| -L | Like the -l option, but print only the names of files that do not contain matches. May also be specified --files-without-match. |
| -n | Prefix each matching line with the number of the line within the file. May also be specified --line-number. |
| -h | For multi-file searches, suppress the output of filenames. May also be specified --no-filename. |

|  |  |
| --- | --- |
| *表20-1: grep 选项* | |
| 选项 | 描述 |
| -i | 忽略大小写。不会区分大小写字符。也可用--ignore-case 来指定。 |
| -v | 不匹配。通常，grep 程序会打印包含匹配项的文本行。这个选项导致 grep 程序 只会不包含匹配项的文本行。也可用--invert-match 来指定。 |
| -c | 打印匹配的数量（或者是不匹配的数目，若指定了-v 选项），而不是文本行本身。 也可用--count 选项来指定。 |
| -l | 打印包含匹配项的文件名，而不是文本行本身，也可用--files-with-matches 选项来指定。 |
| -L | 相似于-l 选项，但是只是打印不包含匹配项的文件名。也可用--files-without-match 来指定。 |
| -n | 在每个匹配行之前打印出其位于文件中的相应行号。也可用--line-number 选项来指定。 |
| -h | 应用于多文件搜索，不输出文件名。也可用--no-filename 选项来指定。 |

In order to more fully explore grep, let’s create some text files to search:

为了更好的探究 grep 程序，让我们创建一些文本文件来搜寻：

[me@linuxbox ~]$ ls /bin > dirlist-bin.txt

[me@linuxbox ~]$ ls /usr/bin > dirlist-usr-bin.txt

[me@linuxbox ~]$ ls /sbin > dirlist-sbin.txt

[me@linuxbox ~]$ ls /usr/sbin > dirlist-usr-sbin.txt

[me@linuxbox ~]$ ls dirlist\*.txt

dirlist-bin.txt dirlist-sbin.txt dirlist-usr-sbin.txt

dirlist-usr-bin.txt

We can perform a simple search of our list of files like this:

我们能够对我们的文件列表执行简单的搜索，像这样：

[me@linuxbox ~]$ grep bzip dirlist\*.txt

dirlist-bin.txt:bzip2

dirlist-bin.txt:bzip2recover

In this example, grep searches all of the listed files for the string bzip and finds two matches, both in the file dirlist-bin.txt. If we were only interested in the list of files that contained matches rather than the matches themselves, we could specify the -l option:

在这个例子里，grep 程序在所有列出的文件中搜索字符串 bzip，然后找到两个匹配项，其都在 文件 dirlist-bin.txt 中。如果我们只是对包含匹配项的文件列表，而不是对匹配项本身感兴趣 的话，我们可以指定-l 选项：

[me@linuxbox ~]$ grep -l bzip dirlist\*.txt

dirlist-bin.txt

Conversely, if we wanted only to see a list of the files that did not contain a match, we could do this:

相反地，如果我们只想查看不包含匹配项的文件列表，我们可以这样操作：

[me@linuxbox ~]$ grep -L bzip dirlist\*.txt

dirlist-sbin.txt

dirlist-usr-bin.txt

dirlist-usr-sbin.txt

**元字符和文本**

While it may not seem apparent, our grep searches have been using regular expressions all along, albeit very simple ones. The regular expression “bzip” is taken to mean that a match will occur only if the line in the file contains at least four characters and that somewhere in the line the characters “b”, “z”, “i”, and “p” are found in that order, with no other characters in between. The characters in the string “bzip” are all literal characters, in that they match themselves. In addition to literals, regular expressions may also include metacharacters that are used to specify more complex matches. Regular expression metacharacters consist of the following:

它可能看起来不明显，但是我们的 grep 程序一直使用了正则表达式，虽然是非常简单的例子。 这个正则表达式“bzip”意味着，匹配项所在行至少包含4个字符，并且按照字符 “b”, “z”, “i”, 和 “p”的顺序 出现在匹配行的某处，字符之间没有其它的字符。字符串“bzip”中的所有字符都是原义字符，因为 它们匹配本身。除了原义字符之外，正则表达式也可能包含元字符，其被用来指定更复杂的匹配项。 正则表达式元字符由以下字符组成：

^ $ . [ ] { } - ? \* + ( ) | \

All other characters are considered literals, though the backslash character is used in a few cases to create meta sequences, as well as allowing the metacharacters to be escaped and treated as literals instead of being interpreted as metacharacters.

然后其它所有字符都被认为是原义字符，虽然在个别情况下，反斜杠会被用来创建元序列， 也允许元字符被转义为原义字符，而不是被解释为元字符。

Note: As we can see, many of the regular expression metacharacters are also characters that have meaning to the shell when expansion is performed. When we pass regular expressions containing metacharacters on the command line, it is vital that they be enclosed in quotes to prevent the shell from attempting to expand them.

注意：正如我们所见到的，当 shell 执行展开的时候，许多正则表达式元字符，也是对 shell 有特殊 含义的字符。当我们在命令行中传递包含元字符的正则表达式的时候，把元字符用引号引起来至关重要， 这样可以阻止 shell 试图展开它们。

**任何字符**

The first metacharacter we will look at is the dot or period character, which is used to match any character. If we include it in a regular expression, it will match any character in that character position. Here’s an example:

我们将要查看的第一个元字符是圆点字符，其被用来匹配任意字符。如果我们在正则表达式中包含它， 它将会匹配在此位置的任意一个字符。这里有个例子：

[me@linuxbox ~]$ grep -h '.zip' dirlist\*.txt

bunzip2

bzip2

bzip2recover

gunzip

gzip

funzip

gpg-zip

preunzip

prezip

prezip-bin

unzip

unzipsfx

We searched for any line in our files that matches the regular expression “.zip”. There are a couple of interesting things to note about the results. Notice that the zip program was not found. This is because the inclusion of the dot metacharacter in our regular expression increased the length of the required match to four characters, and because the name “zip” only contains three, it does not match. Also, if there had been any files in our lists that contained the file extension .zip, they would have also been matched as well, because the period character in the file extension is treated as “any character,” too.

我们在文件中查找包含正则表达式“.zip”的文本行。对于搜索结果，有几点需要注意一下。 注意没有找到这个 zip 程序。这是因为在我们的正则表达式中包含的圆点字符把所要求的匹配项的长度 增加到四个字符，并且字符串“zip”只包含三个字符，所以这个 zip 程序不匹配。另外，如果我们的文件列表 中有一些文件的扩展名是.zip，则它们也会成为匹配项，因为文件扩展名中的圆点符号也会被看作是 “任意字符”。

**锚点**

The caret and dollar sign characters are treated as anchors in regular expressions. This means that they cause the match to occur only if the regular expression is found at the beginning of the line or at the end of the line:

在正则表达式中，插入符号和美元符号被看作是锚点。这意味着正则表达式 只有在文本行的开头或末尾被找到时，才算发生一次匹配。

[me@linuxbox ~]$ grep -h '^zip' dirlist\*.txt

zip

zipcloak

zipgrep

zipinfo

zipnote

zipsplit

[me@linuxbox ~]$ grep -h 'zip$' dirlist\*.txt

gunzip

gzip

funzip

gpg-zip

preunzip

prezip

unzip

zip

[me@linuxbox ~]$ grep -h '^zip$' dirlist\*.txt

zip

Here we searched the list of files for the string “zip” located at the beginning of the line, the end of the line, and on a line where it is at both the beginning and the end of the line (i.e., by itself on the line.) Note that the regular expression ‘^$’ (a beginning and an end with nothing in between) will match blank lines.

这里我们分别在文件列表中搜索行首，行尾以及行首和行尾同时包含字符串“zip”（例如，zip 独占一行）的匹配行。 注意正则表达式‘^$’（行首和行尾之间没有字符）会匹配空行。

A Crossword Puzzle Helper

字谜助手

Even with our limited knowledge of regular expressions at this point, we can do something useful.

到目前为止，甚至凭借我们有限的正则表达式知识，我们已经能做些有意义的事情了。

My wife loves crossword puzzles and she will sometimes ask me for help with a particular question. Something like, “what’s a five letter word whose third letter is ‘j’ and last letter is ‘r’ that means…?” This kind of question got me thinking.

我妻子喜欢玩字谜游戏，有时候她会因为一个特殊的问题，而向我求助。类似这样的问题，“一个 有五个字母的单词，它的第三个字母是‘j’，最后一个字母是‘r’，是哪个单词？”这类问题会 让我动脑筋想想。

Did you know that your Linux system contains a dictionary? It does. Take a look in the /usr/share/dict directory and you might find one, or several. The dictionary files located there are just long lists of words, one per line, arranged in alphabetical order. On my system, the words file contains just over 98,500 words. To find possible answers to the crossword puzzle question above, we could do this:

你知道你的 Linux 系统中带有一本英文字典吗？千真万确。看一下 /usr/share/dict 目录，你就能找到一本， 或几本。存储在此目录下的字典文件，其内容仅仅是一个长长的单词列表，每行一个单词，按照字母顺序排列。在我的 系统中，这个文件仅包含98,000个单词。为了找到可能的上述字谜的答案，我们可以这样做：

[me@linuxbox ~]$ grep -i '^..j.r$' /usr/share/dict/words

Major

major

Using this regular expression, we can find all the words in our dictionary file that are five letters long and have a “j” in the third position and an “r” in the last position.

使用这个正则表达式，我们能在我们的字典文件中查找到包含五个字母，且第三个字母 是“j”，最后一个字母是“r”的所有单词。

**中括号表达式和字符类**

In addition to matching any character at a given position in our regular expression, we can also match a single character from a specified set of characters by using bracket expressions. With bracket expressions, we can specify a set of characters (including characters that would otherwise be interpreted as metacharacters) to be matched. In this example, using a two character set:

除了能够在正则表达式中的给定位置匹配任意字符之外，通过使用中括号表达式， 我们也能够从一个指定的字符集合中匹配一个单个的字符。通过中括号表达式，我们能够指定 一个字符集合（包含在不加中括号的情况下会被解释为元字符的字符）来被匹配。在这个例子里，使用了一个两个字符的集合：

[me@linuxbox ~]$ grep -h '[bg]zip' dirlist\*.txt

bzip2

bzip2recover

gzip

we match any line that contains the string “bzip” or “gzip”.

我们匹配包含字符串“bzip”或者“gzip”的任意行。

A set may contain any number of characters, and metacharacters lose their special meaning when placed within brackets. However, there are two cases in which metacharacters are used within bracket expressions, and have different meanings. The first is the caret (^), which is used to indicate negation; the second is the dash (-), which is used to indicate a character range.

一个字符集合可能包含任意多个字符，并且元字符被放置到中括号里面后会失去了它们的特殊含义。 然而，在两种情况下，会在中括号表达式中使用元字符，并且有着不同的含义。第一个元字符 是插入字符，其被用来表示否定；第二个是连字符字符，其被用来表示一个字符区域。

**否定**

If the first character in a bracket expression is a caret (^), the remaining characters are taken to be a set of characters that must not be present at the given character position. We do this by modifying our previous example:

如果在正则表示式中的第一个字符是一个插入字符，则剩余的字符被看作是不会在给定的字符位置出现的 字符集合。通过修改之前的例子，我们试验一下：

[me@linuxbox ~]$ grep -h '[^bg]zip' dirlist\*.txt

bunzip2

gunzip

funzip

gpg-zip

preunzip

prezip

prezip-bin

unzip

unzipsfx

With negation activated, we get a list of files that contain the string “zip” preceded by any character except “b” or “g”. Notice that the file zip was not found. A negated character set still requires a character at the given position, but the character must not be a member of the negated set.

通过激活否定操作，我们得到一个文件列表，它们的文件名都包含字符串“zip”，并且“zip”的前一个字符 是除了“b”和“g”之外的任意字符。注意文件 zip 没有被发现。一个否定的字符集仍然在给定位置要求一个字符， 但是这个字符必须不是否定字符集的成员。

The caret character only invokes negation if it is the first character within a bracket expression; otherwise, it loses its special meaning and becomes an ordinary character in the set.

这个插入字符如果是中括号表达式中的第一个字符的时候，才会唤醒否定功能；否则，它会失去 它的特殊含义，变成字符集中的一个普通字符。

**传统的字符区域**

If we wanted to construct a regular expression that would find every file in our lists beginning with an upper case letter, we could do this:

如果我们想要构建一个正则表达式，它可以在我们的列表中找到每个以大写字母开头的文件，我们 可以这样做：

[me@linuxbox ~]$ grep -h '^[ABCDEFGHIJKLMNOPQRSTUVWXZY]' dirlist\*.txt

It’s just a matter of putting all twenty-six upper case letters in a bracket expression. But the idea of all that typing is deeply troubling, so there is another way:

这只是一个在正则表达式中输入26个大写字母的问题。但是输入所有字母非常令人烦恼，所以有另外一种方式：

[me@linuxbox ~]$ grep -h '^[A-Z]' dirlist\*.txt

MAKEDEV

ControlPanel

GET

HEAD

POST

X

X11

Xorg

MAKEFLOPPIES

NetworkManager

NetworkManagerDispatcher

By using a three character range, we can abbreviate the twenty-six letters. Any range of characters can be expressed this way including multiple ranges, such as this expression that matches all filenames starting with letters and numbers:

通过使用一个三字符区域，我们能够缩写26个字母。任意字符的区域都能按照这种方式表达，包括多个区域， 比如下面这个表达式就匹配了所有以字母和数字开头的文件名：

[me@linuxbox ~]$ grep -h '^[A-Za-z0-9]' dirlist\*.txt

In character ranges, we see that the dash character is treated specially, so how do we actually include a dash character in a bracket expression? By making it the first character in the expression. Consider these two examples:

在字符区域中，我们看到这个连字符被特殊对待，所以我们怎样在一个正则表达式中包含一个连字符呢？ 方法就是使连字符成为表达式中的第一个字符。考虑一下这两个例子：

[me@linuxbox ~]$ grep -h '[A-Z]' dirlist\*.txt

This will match every filename containing an upper case letter. While:

这会匹配包含一个大写字母的文件名。然而：

[me@linuxbox ~]$ grep -h '[-AZ]' dirlist\*.txt

will match every filename containing a dash, or a upper case “A” or an uppercase “Z”.

上面的表达式会匹配包含一个连字符，或一个大写字母“A”，或一个大写字母“Z”的文件名。

**POSIX 字符集**

The traditional character ranges are an easily understood and effective way to handle the problem of quickly specifying sets of characters. Unfortunately, they don’t always work. While we have not encountered any problems with our use of grep so far, we might run into problems using other programs.

这个传统的字符区域在处理快速地指定字符集合的问题方面，是一个易于理解的和有效的方式。 不幸地是，它们不总是工作。到目前为止，虽然我们在使用 grep 程序的时候没有遇到任何问题， 但是我们可能在使用其它程序的时候会遭遇困难。

Back in Chapter 5, we looked at how wildcards are used to perform pathname expansion. In that discussion, we said that character ranges could be used in a manner almost identical to the way they are used in regular expressions, but here’s the problem:

回到第5章，我们看看通配符怎样被用来完成路径名展开操作。在那次讨论中，我们说过在 某种程度上，那个字符区域被使用的方式几乎与在正则表达式中的用法一样，但是有一个问题：

[me@linuxbox ~]$ ls /usr/sbin/[ABCDEFGHIJKLMNOPQRSTUVWXYZ]\*

/usr/sbin/MAKEFLOPPIES

/usr/sbin/NetworkManagerDispatcher

/usr/sbin/NetworkManager

(Depending on the Linux distribution, we will get a different list of files, possibly an empty list. This example is from Ubuntu) This command produces the expected result — a list of only the files whose names begin with an uppercase letter, but:

（依赖于不同的 Linux 发行版，我们将得到不同的文件列表，有可能是一个空列表。这个例子来自于 Ubuntu） 这个命令产生了期望的结果——只有以大写字母开头的文件名，但是：

[me@linuxbox ~]$ ls /usr/sbin/[A-Z]\*

/usr/sbin/biosdecode

/usr/sbin/chat

/usr/sbin/chgpasswd

/usr/sbin/chpasswd

/usr/sbin/chroot

/usr/sbin/cleanup-info

/usr/sbin/complain

/usr/sbin/console-kit-daemon

with this command we get an entirely different result (only a partial listing of the results is shown). Why is that? It’s a long story, but here’s the short version:

通过这个命令我们得到整个不同的结果（只显示了一部分结果列表）。为什么会是那样？ 说来话长，但是这个版本比较简短：

Back when Unix was first developed, it only knew about ASCII characters, and this feature reflects that fact. In ASCII, the first thirty-two characters (numbers 0-31) are control codes (things like tabs, backspaces, and carriage returns). The next thirty-two (32-63) contain printable characters, including most punctuation characters and the numerals zero through nine. The next thirty-two (numbers 64-95) contain the uppercase letters and a few more punctuation symbols. The final thirty-one (numbers 96-127) contain the lowercase letters and yet more punctuation symbols. Based on this arrangement, systems using ASCII used a collation order that looked like this:

追溯到 Unix 刚刚开发的时候，它只知道 ASCII 字符，并且这个特性反映了事实。在 ASCII 中，前32个字符 （数字0－31）都是控制码（如 tabs，backspaces，和回车）。随后的32个字符（32－63）包含可打印的字符， 包括大多数的标点符号和数字0到9。再随后的32个字符（64－95）包含大写字符和一些更多的标点符号。 最后的31个字符（96－127）包含小写字母和更多的标点符号。基于这种安排方式，系统使用这种排序规则 的 ASCII：

ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz

This differs from proper dictionary order, which is like this:

这个不同于正常的字典顺序，其像这样：

aAbBcCdDeEfFgGhHiIjJkKlLmMnNoOpPqQrRsStTuUvVwWxXyYzZ

As the popularity of Unix spread beyond the United States, there grew a need to support characters not found in U.S. English. The ASCII table was expanded to use a full eight bits, adding characters numbers 128-255, which accommodated many more languages.

随着 Unix 系统的知名度在美国之外的国家传播开来，就需要支持不在 U.S.英语范围内的字符。 于是就扩展了这个 ASCII 字符表，使用了整个8位，添加了字符（数字128－255），这样就 容纳了更多的语言。

To support this ability, the POSIX standards introduced a concept called a locale, which could be adjusted to select the character set needed for a particular location. We can see the language setting of our system using this command:

为了支持这种能力，POSIX 标准介绍了一种叫做 locale 的概念，其可以被调整，来为某个特殊的区域， 选择所需的字符集。通过使用下面这个命令，我们能够查看到我们系统的语言设置：

[me@linuxbox ~]$ echo $LANG

en\_US.UTF-8

With this setting, POSIX compliant applications will use a dictionary collation order rather than ASCII order. This explains the behavior of the commands above. A character range of [A-Z] when interpreted in dictionary order includes all of the alphabetic characters except the lowercase “a”, hence our results.

通过这个设置，POSIX 相容的应用程序将会使用字典排列顺序而不是 ASCII 顺序。这就解释了上述命令的行为。 当[A-Z]字符区域按照字典顺序解释的时候，包含除了小写字母“a”之外的所有字母，因此得到这样的结果。

To partially work around this problem, the POSIX standard includes a number of character classes which provide useful ranges of characters. They are described in the table below:

为了部分地解决这个问题，POSIX 标准包含了大量的字符集，其提供了有用的字符区域。 下表中描述了它们：

|  |  |
| --- | --- |
| *Table 20-2: POSIX Character Classes* | |
| Character Class | Description |
| [:alnum:] | The alphanumeric characters. In ASCII, equivalent to: [A-Za-z0-9] |
| [:word:] | The same as [:alnum:], with the addition of the underscore (\\_) character. |
| [:alpha:] | The alphabetic characters. In ASCII, equivalent to: [A-Za-z] |
| [:blank:] | Includes the space and tab characters. |
| [:cntrl:] | The ASCII control codes. Includes the ASCII characters zero through thirty-one and 127. |
| [:digit:] | The numerals zero through nine. |
| [:graph:] | The visible characters. In ASCII, it includes characters 33 through 126. |
| [:lower:] | The lowercase letters. |
| [:punct:] | The punctuation characters. In ASCII, equivalent to: |
| [:print:] | The printable characters. All the characters in [:graph:] plus the space character. |
| [:space:] | The whitespace characters including space, tab, carriage return, newline, vertical tab, and form feed. In ASCII, equivalent to: [ \t\r\n\v\f] |
| [:upper:] | The upper case characters. |
| [:xdigit:] | Characters used to express hexadecimal numbers. In ASCII, equivalent to: [0-9A-Fa-f] |

|  |  |
| --- | --- |
| *表20-2: POSIX 字符集* | |
| 字符集 | 说明 |
| [:alnum:] | 字母数字字符。在 ASCII 中，等价于：[A-Za-z0-9] |
| [:word:] | 与[:alnum:]相同, 但增加了下划线字符。 |
| [:alpha:] | 字母字符。在 ASCII 中，等价于：[A-Za-z] |
| [:blank:] | 包含空格和 tab 字符。 |
| [:cntrl:] | ASCII 的控制码。包含了0到31，和127的 ASCII 字符。 |
| [:digit:] | 数字0到9 |
| [:graph:] | 可视字符。在 ASCII 中，它包含33到126的字符。 |
| [:lower:] | 小写字母。 |
| [:punct:] | 标点符号字符。在 ASCII 中，等价于： |
| [:print:] | 可打印的字符。在[:graph:]中的所有字符，再加上空格字符。 |
| [:space:] | 空白字符，包括空格，tab，回车，换行，vertical tab, 和 form feed.在 ASCII 中， 等价于：[ \t\r\n\v\f] |
| [:upper:] | 大写字母。 |
| [:xdigit:] | 用来表示十六进制数字的字符。在 ASCII 中，等价于：[0-9A-Fa-f] |

Even with the character classes, there is still no convenient way to express partial ranges, such as [A-M].

甚至通过字符集，仍然没有便捷的方法来表达部分区域，比如[A-M]。

Using character classes, we can repeat our directory listing and see an improved result:

通过使用字符集，我们重做上述的例题，看到一个改进的结果：

[me@linuxbox ~]$ ls /usr/sbin/[[:upper:]]\*

/usr/sbin/MAKEFLOPPIES

/usr/sbin/NetworkManagerDispatcher

/usr/sbin/NetworkManager

Remember, however, that this is not an example of a regular expression, rather it is the shell performing pathname expansion. We show it here because POSIX character classes can be used for both.

记住，然而，这不是一个正则表达式的例子，而是 shell 正在执行路径名展开操作。我们在这里展示这个例子， 是因为 POSIX 规范的字符集适用于二者。

Reverting To Traditional Collation Order

恢复到传统的排列顺序

You can opt to have your system use the traditional (ASCII) collation order by changing the value of the LANG environment variable. As we saw above, the LANG variable contains the name of the language and character set used in your locale. This value was originally determined when you selected an installation language as your Linux was installed.

通过改变环境变量 LANG 的值，你可以选择让你的系统使用传统的（ASCII）排列规则。如上所示，这个 LANG 变量包含了语种和字符集。这个值最初由你安装 Linux 系统时所选择的安装语言决定。

To see the locale settings, use the locale command:

使用 locale 命令，来查看 locale 的设置。

[me@linuxbox ~]$ locale

LANG=en\_US.UTF-8

LC\_CTYPE="en\_US.UTF-8"

LC\_NUMERIC="en\_US.UTF-8"

LC\_TIME="en\_US.UTF-8"

LC\_COLLATE="en\_US.UTF-8"

LC\_MONETARY="en\_US.UTF-8"

LC\_MESSAGES="en\_US.UTF-8"

LC\_PAPER="en\_US.UTF-8"

LC\_NAME="en\_US.UTF-8"

LC\_ADDRESS="en\_US.UTF-8"

LC\_TELEPHONE="en\_US.UTF-8"

LC\_MEASUREMENT="en\_US.UTF-8"

LC\_IDENTIFICATION="en\_US.UTF-8"

LC\_ALL=

To change the locale to use the traditional Unix behaviors, set the LANG variable to POSIX:

把这个 LANG 变量设置为 POSIX，来更改 locale，使其使用传统的 Unix 行为。

**[me@linuxbox ~]$ export LANG=POSIX**

Note that this change converts the system to use U.S. English (more specifically, ASCII) for its character set, so be sure if this is really what you want.

You can make this change permanent by adding this line to you your .bashrc file:

注意这个改动使系统为它的字符集使用 U.S.英语（更准确地说，ASCII），所以要确认一下这 是否是你真正想要的效果。通过把这条语句添加到你的.bashrc 文件中，你可以使这个更改永久有效。

**export LANG=POSIX**

**POSIX 基本的 Vs.扩展的正则表达式**

Just when we thought this couldn’t get any more confusing, we discover that POSIX also splits regular expression implementations into two kinds: basic regular expressions (BRE) and extended regular expressions (ERE). The features we have covered so far are supported by any application that is POSIX-compliant and implements BRE. Our grep program is one such program.

就在我们认为这已经非常令人困惑了，我们却发现 POSIX 把正则表达式的实现分成了两类： 基本正则表达式（BRE）和扩展的正则表达式（ERE）。既服从 POSIX 规范又实现了 BRE 的任意应用程序，都支持我们目前研究的所有正则表达式特性。我们的 grep 程序就是其中一个。

What’s the difference between BRE and ERE? It’s a matter of metacharacters. With BRE, the following metacharacters are recognized:

BRE 和 ERE 之间有什么区别呢？这是关于元字符的问题。BRE 可以辨别以下元字符：

^ $ . [ ] \*

All other characters are considered literals. With ERE, the following metacharacters (and their associated functions) are added:

其它的所有字符被认为是文本字符。ERE 添加了以下元字符（以及与其相关的功能）:

( ) { } ? + |

However (and this is the fun part), the “(”, “)”, “{”, and “}” characters are treated as metacharacters in BRE if they are escaped with a backslash, whereas with ERE, preceding any metacharacter with a backslash causes it to be treated as a literal. Any weirdness that comes along will be covered in the discussions that follow.

然而（这也是有趣的地方），在 BRE 中，字符“(”，“)”，“{”，和 “}”用反斜杠转义后，被看作是元字符, 相反在 ERE 中，在任意元字符之前加上反斜杠会导致其被看作是一个文本字符。在随后的讨论中将会涵盖 很多奇异的特性。

Since the features we are going to discuss next are part of ERE, we are going to need to use a different grep. Traditionally, this has been performed by the egrep program, but the GNU version of grep also supports extended regular expressions when the -E option is used.

因为我们将要讨论的下一个特性是 ERE 的一部分，我们将要使用一个不同的 grep 程序。照惯例， 一直由 egrep 程序来执行这项操作，但是 GUN 版本的 grep 程序也支持扩展的正则表达式，当使用了-E 选项之后。

POSIX

During the 1980’s, Unix became a very popular commercial operating system, but by 1988, the Unix world was in turmoil. Many computer manufacturers had licensed the Unix source code from its creators, AT&T, and were supplying various versions of the operating system with their systems. However, in their efforts to create product differentiation, each manufacturer added proprietary changes and extensions. This started to limit the compatibility of the software.

在 20 世纪 80 年代，Unix 成为一款非常流行的商业操作系统，但是到了1988年，Unix 世界 一片混乱。许多计算机制造商从 Unix 的创建者 AT&T 那里得到了许可的 Unix 源码，并且 供应各种版本的操作系统。然而，在他们努力创造产品差异化的同时，每个制造商都增加了 专用的更改和扩展。这就开始限制了软件的兼容性。

As always with proprietary vendors, each was trying to play a winning game of “lock-in” with their customers. This dark time in the history of Unix is known today as “the Balkanization.”

专有软件供应商一如既往，每个供应商都试图玩嬴游戏“锁定”他们的客户。这个 Unix 历史上 的黑暗时代，就是今天众所周知的 “the Balkanization”。

Enter the IEEE (Institute of Electrical and Electronics Engineers). In the mid-1980s, the IEEE began developing a set of standards that would define how Unix (and Unix-like) systems would perform. These standards, formally known as IEEE 1003, define the application programming interfaces (APIs), shell and utilities that are to be found on a standard Unix-like system. The name “POSIX,” which stands for Portable Operating System Interface (with the “X” added to the end for extra snappiness), was suggested by Richard Stallman (yes, that Richard Stallman), and was adopted by the IEEE.

然后进入 IEEE（ 电气与电子工程师协会 ）时代。在上世纪 80 年代中叶，IEEE 开始制定一套标准， 其将会定义 Unix 系统（ 以及类 Unix 的系统 ）如何执行。这些标准，正式成为 IEEE 1003， 定义了应用程序编程接口（ APIs ），shell 和一些实用程序，其将会在标准的类 Unix 操作系统中找到。“POSIX” 这个名字，象征着可移植的操作系统接口（为了额外的，添加末尾的 “X” ）， 是由 Richard Stallman 建议的（ 是的，的确是 Richard Stallman ），后来被 IEEE 采纳。

**Alternation**

The first of the extended regular expression features we will discuss is called alternation, which is the facility that allows a match to occur from among a set of expressions. Just as a bracket expression allows a single character to match from a set of specified characters, alternation allows matches from a set of strings or other regular expressions. To demonstrate, we’ll use grep in conjunction with echo. First, let’s try a plain old string match:

我们将要讨论的扩展表达式的第一个特性叫做 alternation（交替），其是一款允许从一系列表达式 之间选择匹配项的实用程序。就像中括号表达式允许从一系列指定的字符之间匹配单个字符那样， alternation 允许从一系列字符串或者是其它的正则表达式中选择匹配项。为了说明问题， 我们将会结合 echo 程序来使用 grep 命令。首先，让我们试一个普通的字符串匹配：

[me@linuxbox ~]$ echo "AAA" | grep AAA

AAA

[me@linuxbox ~]$ echo "BBB" | grep AAA

[me@linuxbox ~]$

A pretty straightforward example, in which we pipe the output of echo into grep and see the results. When a match occurs, we see it printed out; when no match occurs, we see no results.

一个相当直截了当的例子，我们把 echo 的输出管道给 grep，然后看到输出结果。当出现 一个匹配项时，我们看到它会打印出来；当没有匹配项时，我们看到没有输出结果。

Now we’ll add alternation, signified by the vertical bar metacharacter:

现在我们将添加 alternation，以竖杠线元字符为标记：

[me@linuxbox ~]$ echo "AAA" | grep -E 'AAA|BBB'

AAA

[me@linuxbox ~]$ echo "BBB" | grep -E 'AAA|BBB'

BBB

[me@linuxbox ~]$ echo "CCC" | grep -E 'AAA|BBB'

[me@linuxbox ~]$

Here we see the regular expression ‘AAA|BBB’ which means “match either the string AAA or the string BBB.” Notice that since this is an extended feature, we added the -E option to grep (though we could have just used the egrep program instead), and we enclosed the regular expression in quotes to prevent the shell from interpreting the vertical bar metacharacter as a pipe operator. Alternation is not limited to two choices:

这里我们看到正则表达式’AAA|BBB’，这意味着“匹配字符串 AAA 或者是字符串 BBB”。注意因为这是 一个扩展的特性，我们给 grep 命令（虽然我们能以 egrep 程序来代替）添加了-E 选项，并且我们 把这个正则表达式用单引号引起来，为的是阻止 shell 把竖杠线元字符解释为一个 pipe 操作符。 Alternation 并不局限于两种选择：

[me@linuxbox ~]$ echo "AAA" | grep -E 'AAA|BBB|CCC'

AAA

To combine alternation with other regular expression elements, we can use () to separate the alternation:

为了把 alternation 和其它正则表达式元素结合起来，我们可以使用()来分离 alternation。

[me@linuxbox ~]$ grep -Eh '^(bz|gz|zip)' dirlist\*.txt

This expression will match the filenames in our lists that start with either “bz”, “gz”, or “zip”. Had we left off the parentheses, the meaning of this regular expression :

这个表达式将会在我们的列表中匹配以“bz”，或“gz”，或“zip”开头的文件名。如果我们删除了圆括号， 这个表达式的意思：

[me@linuxbox ~]$ grep -Eh '^bz|gz|zip' dirlist\*.txt

changes to match any filename that begins with “bz” or contains “gz” or contains “zip”.

会变成匹配任意以“bz”开头，或包含“gz”，或包含“zip”的文件名。

**限定符**

Extended regular expressions support several ways to specify the number of times an element is matched.

扩展的正则表达式支持几种方法，来指定一个元素被匹配的次数。

**? - 匹配零个或一个元素**

This quantifier means, in effect, “make the preceding element optional.” Let’s say we wanted to check a phone number for validity and we considered a phone number to be valid if it matched either of these two forms:

这个限定符意味着，实际上，“使前面的元素可有可无。”比方说我们想要查看一个电话号码的真实性， 如果它匹配下面两种格式的任意一种，我们就认为这个电话号码是真实的：

(nnn) nnn-nnnn

nnn nnn-nnnn

where “n” is a numeral. We could construct a regular expression like this:

这里的“n”是一个数字。我们可以构建一个像这样的正则表达式：

^\(?[0-9][0-9][0-9]\)? [0-9][0-9][0-9]-[0-9][0-9][0-9][0-9]$

In this expression, we follow the parentheses characters with question marks to indicate that they are to be matched zero or one time. Again, since the parentheses are normally metacharacters (in ERE), we precede them with backslashes to cause them to be treated as literals instead.

在这个表达式中，我们在圆括号之后加上一个问号，来表示它们将被匹配零次或一次。再一次，因为 通常圆括号都是元字符（在 ERE 中），所以我们在圆括号之前加上了反斜杠，使它们成为文本字符。

Let’s try it:

让我们试一下：

[me@linuxbox ~]$ echo "(555) 123-4567" | grep -E '^\(?[0-9][0-9][0-9]

\)? [0-9][0-9][0-9]-[0-9][0-9][0-9][0-9]$'

(555) 123-4567

[me@linuxbox ~]$ echo "555 123-4567" | grep -E '^\(?[0-9][0-9][0-9]\)

? [0-9][0-9][0-9]-[0-9][0-9][0-9][0-9]$'

555 123-4567

[me@linuxbox ~]$ echo "AAA 123-4567" | grep -E '^\(?[0-9][0-9][0-9]\)

? [0-9][0-9][0-9]-[0-9][0-9][0-9][0-9]$'

[me@linuxbox ~]$

Here we see that the expression matches both forms of the phone number, but does not match one containing non-numeric characters.

这里我们看到这个表达式匹配这个电话号码的两种形式，但是不匹配包含非数字字符的号码。

**\* - 匹配零个或多个元素**

Like the ? metacharacter, the \* is used to denote an optional item; however, unlike the ?, the item may occur any number of times, not just once. Let’s say we wanted to see if a string was a sentence; that is, it starts with an uppercase letter, then contains any number of upper and lowercase letters and spaces, and ends with a period. To match this (very crude) definition of a sentence, we could use a regular expression like this:

像 ? 元字符一样，这个 \* 被用来表示一个可选的字符；然而，又与 ? 不同，匹配的字符可以出现 任意多次，不仅是一次。比方说我们想要知道是否一个字符串是一句话；也就是说，字符串开始于 一个大写字母，然后包含任意多个大写和小写的字母和空格，最后以句号收尾。为了匹配这个（非常粗略的） 语句的定义，我们能够使用一个像这样的正则表达式：

[[:upper:]][[:upper:][:lower:] ]\*.

The expression consists of three items: a bracket expression containing the [:upper:] character class, a bracket expression containing both the [:upper:] and [:lower:] character classes and a space, and a period escaped with a backslash. The second element is trailed with an \* metacharacter, so that after the leading uppercase letter in our sentence, any number of upper and lowercase letters and spaces may follow it and still match:

这个表达式由三个元素组成：一个包含[:upper:]字符集的中括号表达式，一个包含[:upper:]和[:lower:] 两个字符集以及一个空格的中括号表达式，和一个被反斜杠字符转义过的圆点。第二个元素末尾带有一个 \*元字符，所以在开头的大写字母之后，可能会跟随着任意数目的大写和小写字母和空格，并且匹配：

[me@linuxbox ~]$ echo "This works." | grep -E '[[:upper:]][[:upper:][:lower:] ]\*\.'

This works.

[me@linuxbox ~]$ echo "This Works." | grep -E '[[:upper:]][[:upper:][:lower:] ]\*\.'

This Works.

[me@linuxbox ~]$ echo "this does not" | grep -E '[[:upper:]][[:upper:][:lower:] ]\*\.'

[me@linuxbox ~]$

The expression matches the first two tests, but not the third, since it lacks the required leading uppercase character and trailing period.

这个表达式匹配前两个测试语句，但不匹配第三个，因为第三个句子缺少开头的大写字母和末尾的句号。

**+ - 匹配一个或多个元素**

The + metacharacter works much like the \*, except it requires at least one instance of the preceding element to cause a match. Here is a regular expression that will only match lines consisting of groups of one or more alphabetic characters separated by single spaces:

这个 + 元字符的作用与 \* 非常相似，除了它要求前面的元素至少出现一次匹配。这个正则表达式只匹配 那些由一个或多个字母字符组构成的文本行，字母字符之间由单个空格分开：

^([[:alpha:]]+ ?)+$

[me@linuxbox ~]$ echo "This that" | grep -E '^([[:alpha:]]+ ?)+$'

This that

[me@linuxbox ~]$ echo "a b c" | grep -E '^([[:alpha:]]+ ?)+$'

a b c

[me@linuxbox ~]$ echo "a b 9" | grep -E '^([[:alpha:]]+ ?)+$'

[me@linuxbox ~]$ echo "abc d" | grep -E '^([[:alpha:]]+ ?)+$'

[me@linuxbox ~]$

We see that this expression does not match the line “a b 9” because it contains a non- alphabetic character; nor does it match “abc d” because more than one space character separates the characters “c” and “d”.

我们看到这个正则表达式不匹配“a b 9”这一行，因为它包含了一个非字母的字符；它也不匹配 “abc d” ，因为在字符“c”和“d”之间不止一个空格。

**{ } - 匹配特定个数的元素**

The { and } metacharacters are used to express minimum and maximum numbers of required matches. They may be specified in four possible ways:

这个 { 和 } 元字符都被用来表达要求匹配的最小和最大数目。它们可以通过四种方法来指定：

|  |  |
| --- | --- |
| *Table 20-3: Specifying The Number Of Matches* | |
| Specifier | Meaning |
| {n} | Match the preceding element if it occurs exactly n times. |
| {n,m} | Match the preceding element if it occurs at least n times, but no more than m times. |
| {n,} | Match the preceding element if it occurs n or more times. |
| {,m} | Match the preceding element if it occurs no more than m times. |

|  |  |
| --- | --- |
| *表20-3: 指定匹配的数目* | |
| 限定符 | 意思 |
| {n} | 匹配前面的元素，如果它确切地出现了 n 次。 |
| {n,m} | 匹配前面的元素，如果它至少出现了 n 次，但是不多于 m 次。 |
| {n,} | 匹配前面的元素，如果它出现了 n 次或多于 n 次。 |
| {,m} | 匹配前面的元素，如果它出现的次数不多于 m 次。 |

Going back to our earlier example with the phone numbers, we can use this method of specifying repetitions to simplify our original regular expression from:

回到之前处理电话号码的例子，我们能够使用这种指定重复次数的方法来简化我们最初的正则表达式：

^\(?[0-9][0-9][0-9]\)? [0-9][0-9][0-9]-[0-9][0-9][0-9][0-9]$

简化为：

^\(?[0-9]{3}\)? [0-9]{3}-[0-9]{4}$

Let’s try it:

让我们试一下：

[me@linuxbox ~]$ echo "(555) 123-4567" | grep -E '^\(?[0-9]{3}\)? [0-9]{3}-[0-9]{4}$'

(555) 123-4567

[me@linuxbox ~]$ echo "555 123-4567" | grep -E '^\(?[0-9]{3}\)? [0-9]{3}-[0-9]{4}$'

555 123-4567

[me@linuxbox ~]$ echo "5555 123-4567" | grep -E '^\(?[0-9]{3}\)? [0-9]{3}-[0-9]{4}$'

[me@linuxbox ~]$

As we can see, our revised expression can successfully validate numbers both with and without the parentheses, while rejecting those numbers that are not properly formatted.

我们可以看到，我们修订的表达式能成功地验证带有和不带有圆括号的数字，而拒绝那些格式 不正确的数字。

**让正则表达式工作起来**

Let’s look at some of the commands we already know and see how they can be used with regular expressions.

让我们看看一些我们已经知道的命令，然后看一下它们怎样使用正则表达式。

**通过 grep 命令来验证一个电话簿**

In our earlier example, we looked at single phone numbers and checked them for proper formatting. A more realistic scenario would be checking a list of numbers instead, so let’s make a list. We’ll do this by reciting a magical incantation to the command line. It will be magic because we have not covered most of the commands involved, but worry not. We will get there in future chapters. Here is the incantation:

在我们先前的例子中，我们查看过单个电话号码，并且检查了它们的格式。一个更现实的 情形是检查一个数字列表，所以我们先创建一个列表。我们将背诵一个神奇的咒语到命令行中。 它会很神奇，因为我们还没有涵盖所涉及的大部分命令，但是不要担心。我们将在后面的章节里面 讨论那些命令。这里是这个咒语：

[me@linuxbox ~]$ for i in {1..10}; do echo "(${RANDOM:0:3}) ${RANDO

M:0:3}-${RANDOM:0:4}" >> phonelist.txt; done

This command will produce a file named phonelist.txt containing ten phone numbers. Each time the command is repeated, another ten numbers are added to the list. We can also change the value 10 near the beginning of the command to produce more or fewer phone numbers. If we examine the contents of the file, however, we see we have a problem:

这个命令会创建一个包含10个电话号码的名为 phonelist.txt 的文件。每次重复这个命令的时候， 另外10个号码会被添加到这个列表中。我们也能够更改命令开头附近的数值10，来生成或多或少的 电话号码。如果我们查看这个文件的内容，然而我们会发现一个问题：

[me@linuxbox ~]$ cat phonelist.txt

(232) 298-2265

(624) 381-1078

(540) 126-1980

(874) 163-2885

(286) 254-2860

(292) 108-518

(129) 44-1379

(458) 273-1642

(686) 299-8268

(198) 307-2440

Some of the numbers are malformed, which is perfect for our purposes, since we will use grep to validate them.

一些号码是残缺不全的，但是它们很适合我们的需求，因为我们将使用 grep 命令来验证它们。

One useful method of validation would be to scan the file for invalid numbers and display the resulting list on the display:

一个有用的验证方法是扫描这个文件，查找无效的号码，并把搜索结果显示到屏幕上：

[me@linuxbox ~]$ grep -Ev '^\([0-9]{3}\) [0-9]{3}-[0-9]{4}$'

phonelist.txt

(292) 108-518

(129) 44-1379

[me@linuxbox ~]$

Here we use the -v option to produce an inverse match so that we will only output the lines in the list that do not match the specified expression. The expression itself includes the anchor metacharacters at each end to ensure that the number has no extra characters at either end. This expression also requires that the parentheses be present in a valid number, unlike our earlier phone number example.

这里我们使用-v 选项来产生相反的匹配，因此我们将只输出不匹配指定表达式的文本行。这个 表达式自身的两端都包含定位点（锚）元字符，是为了确保这个号码的两端没有多余的字符。 这个表达式也要求圆括号出现在一个有效的号码中，不同于我们先前电话号码的实例。

**用 find 查找丑陋的文件名**

The find command supports a test based on a regular expression. There is an important consideration to keep in mind when using regular expressions in find versus grep. Whereas grep will print a line when the line contains a string that matches an expression, find requires that the pathname exactly match the regular expression. In the following example, we will use find with a regular expression to find every pathname that contains any character that is not a member of the following set:

这个 find 命令支持一个基于正则表达式的测试。当在使用正则表达式方面比较 find 和 grep 命令的时候， 还有一个重要问题要牢记在心。当某一行包含的字符串匹配上了一个表达式的时候，grep 命令会打印出这一行， 然而 find 命令要求路径名精确地匹配这个正则表达式。在下面的例子里面，我们将使用带有一个正则 表达式的 find 命令，来查找每个路径名，其包含的任意字符都不是以下字符集中的一员。

[-\\_./0-9a-zA-Z]

Such a scan would reveal pathnames that contain embedded spaces and other potentially offensive characters:

这样一种扫描会发现包含空格和其它潜在不规范字符的路径名：

[me@linuxbox ~]$ find . -regex '.\*[^-\\_./0-9a-zA-Z].\*'

Due to the requirement for an exact match of the entire pathname, we use .\* at both ends of the expression to match zero or more instances of any character. In the middle of the expression, we use a negated bracket expression containing our set of acceptable pathname characters.

由于要精确地匹配整个路径名，所以我们在表达式的两端使用了.\*，来匹配零个或多个字符。 在表达式中间，我们使用了否定的中括号表达式，其包含了我们一系列可接受的路径名字符。

**用 locate 查找文件**

The locate program supports both basic (the --regexp option) and extended (the -- regex option) regular expressions. With it, we can perform many of the same operations that we performed earlier with our dirlist files:

这个 locate 程序支持基本的（--regexp 选项）和扩展的（--regex 选项）正则表达式。通过 locate 命令，我们能够执行许多与先前操作 dirlist 文件时相同的操作：

[me@linuxbox ~]$ locate --regex 'bin/(bz|gz|zip)'

/bin/bzcat

/bin/bzcmp

/bin/bzdiff

/bin/bzegrep

/bin/bzexe

/bin/bzfgrep

/bin/bzgrep

/bin/bzip2

/bin/bzip2recover

/bin/bzless

/bin/bzmore

/bin/gzexe

/bin/gzip

/usr/bin/zip

/usr/bin/zipcloak

/usr/bin/zipgrep

/usr/bin/zipinfo

/usr/bin/zipnote

/usr/bin/zipsplit

Using alternation, we perform a search for pathnames that contain either bin/bz, bin/gz, or /bin/zip.

通过使用 alternation，我们搜索包含 bin/bz，bin/gz，或/bin/zip 字符串的路径名。

**在 less 和 vim 中查找文本**

less and vim both share the same method of searching for text. Pressing the / key followed by a regular expression will perform a search. If we use less to view our phonelist.txt file:

less 和 vim 两者享有相同的文本查找方法。按下/按键，然后输入正则表达式，来执行搜索任务。 如果我们使用 less 程序来浏览我们的 phonelist.txt 文件：

[me@linuxbox ~]$ less phonelist.txt

Then search for our validation expression:

然后查找我们有效的表达式：

(232) 298-2265

(624) 381-1078

(540) 126-1980

(874) 163-2885

(286) 254-2860

(292) 108-518

(129) 44-1379

(458) 273-1642

(686) 299-8268

(198) 307-2440

~

~

~

/^\([0-9]{3}\) [0-9]{3}-[0-9]{4}$

less will highlight the strings that match, leaving the invalid ones easy to spot:

less 将会高亮匹配到的字符串，这样就很容易看到无效的电话号码：

(232) 298-2265

(624) 381-1078

(540) 126-1980

(874) 163-2885

(286) 254-2860

(292) 108-518

(129) 44-1379

(458) 273-1642

(686) 299-8268

(198) 307-2440

~

~

~

(END)

vim, on the other hand, supports basic regular expressions, so our search expression would look like this:

另一方面，vim 支持基本的正则表达式，所以我们用于搜索的表达式看起来像这样：

/([0-9]\{3\}) [0-9]\{3\}-[0-9]\{4\}

We can see that the expression is mostly the same; however, many of the characters that are considered metacharacters in extended expressions are considered literals in basic expressions. They are only treated as metacharacters when escaped with a backslash.

我们看到表达式几乎一样；然而，在扩展表达式中，许多被认为是元字符的字符在基本的表达式 中被看作是文本字符。只有用反斜杠把它们转义之后，它们才被看作是元字符。

Depending on the particular configuration of vim on our system, the matching will be highlighted. If not, try this command mode command:

依赖于系统中 vim 的特殊配置，匹配项将会被高亮。如若不是，试试这个命令模式：

:hlsearch

to activate search highlighting.

来激活搜索高亮功能。

Note: Depending on your distribution, vim may or may not support text search highlighting. Ubuntu, in particular, supplies a very stripped-down version of vim by default. On such systems, you may want to use your package manager to install a more complete version of vim.

注意：依赖于你的发行版，vim 有可能支持或不支持文本搜索高亮功能。尤其是 Ubuntu 自带了 一款非常简化的 vim 版本。在这样的系统中，你可能要使用你的软件包管理器来安装一个功能 更完备的 vim 版本。

**总结归纳**

In this chapter, we’ve seen a few of the many uses of regular expressions. We can find even more if we use regular expressions to search for additional applications that use them. We can do that by searching the man pages:

在这章中，我们已经看到几个使用正则表达式例子。如果我们使用正则表达式来搜索那些使用正则表达式的应用程序， 我们可以找到更多的使用实例。通过查找手册页，我们就能找到：

[me@linuxbox ~]$ cd /usr/share/man/man1

[me@linuxbox man1]$ zgrep -El 'regex|regular expression' \*.gz

The zgrep program provides a front end for grep, allowing it to read compressed files. In our example, we search the compressed section one man page files located in their usual location. The result of this command is a list of files containing either the string “regex” or “regular expression”. As we can see, regular expressions show up in a lot of programs.

这个 zgrep 程序是 grep 的前端，允许 grep 来读取压缩文件。在我们的例子中，我们在手册文件所在的 目录中，搜索压缩文件中的内容。这个命令的结果是一个包含字符串“regex”或者“regular expression”的文件列表。正如我们所看到的，正则表达式会出现在大量程序中。

There is one feature found in basic regular expressions that we did not cover. Called back references, this feature will be discussed in the next chapter.

基本正则表达式中有一个特性，我们没有涵盖。叫做反引用，这个特性在下一章中会被讨论到。

**拓展阅读**

There are many online resources for learning regular expressions, including various tutorials and cheat sheets.

有许多在线学习正则表达式的资源，包括各种各样的教材和速记表。

In addition, the Wikipedia has good articles on the following background topics:

另外，关于下面的背景话题，Wikipedia 有不错的文章。

* POSIX: <http://en.wikipedia.org/wiki/Posix>
* ASCII: <http://en.wikipedia.org/wiki/Ascii>