第二部分　文本过滤

第7章　正则表达式介绍

随着对UNIX和LINUX熟悉程度的不断加深，需要经常接触到正则表达式这个领域。使用shell时，从一个文件中抽取多于一个字符串将会很麻烦。例如，在一个文本中抽取一个词，它的头两个字符是大写的，后面紧跟四个数字。如果不使用某种正则表达式，在 shell中将不能实现这个操作。

本章内容包括：

- 匹配行首与行尾。
- 匹配数据集。
- 只匹配字母和数字。
- 匹配一定范围内的字符串集。

当从一个文件或命令输出中抽取或过滤文本时，可以使用正则表达式（ RE），正则表达式是一些特殊或不很特殊的字符串模式的集合。

为了抽取或获得信息，我们给出抽取操作应遵守的一些规则。这些规则由一些特殊字符或进行模式匹配操作时使用的元字符组成。也可以使用规则字符作为模式中的一部分进行搜寻。例如， A将查询A，x将查找字母x。

系统自带的所有大的文本过滤工具在某种模式下都支持正则表达式的使用，并且还包括一些扩展的元字符集。这里只涉及其中之一，即以字符出现情况进行匹配的表达式，原因是一些系统将这类模式划分为一组形成基本元字符的集合。这是一个好想法，本书也采用这种方式。

本章设计的基本元字符使用在grep和sed命令中，同时结合{\\}（以字符出现情况进行匹配的元字符）使用在awk语言中。

表7-1　基本元字符集及其含义

^	只匹配行首
$	只匹配行尾
*	一个单字符后紧跟*，匹配0个或多个此单字符
[ ]	匹配[]内字符。可以是一个单字符，也可以是字符序列。可以使用 -表示[]内字符序列范围，如用[1-5]代替[12345]
\	用来屏蔽一个元字符的特殊含义。因为有时在 shell中一些元字符有特殊含义。\可以使其失去应有意义
.	匹配任意单字符
pattern\{n\}	用来匹配前面pattern出现次数。n为次数
pattern\{n，\}m	含义同上，但次数最少为n
pattern\{n，m\}	含义同上，但pattern出现次数在n与m之间

现在详细讲解其中特殊含义。

7.1 使用句点匹配单字符

句点“.”可以匹配任意单字符。例如，如果要匹配一个字符串，以beg开头，中间夹一个任意字符，那么可以表示为beg.n，“.”可以匹配字符串头，也可以是中间任意字符。

在ls -l命令中，可以匹配一定权限：

```
...x..x..x
```

此格式匹配用户本身，用户组及其他组成员的执行权限。

```
drwxrwxrw-      – no match
-rw-rw-rw-      – no match
-rwx-rwxr-x     – match
-rwx-r-x-r-x    – match
```

假定正在过滤一个文本文件，对于一个有10个字符的脚本集，要求前4个字符之后为XC，匹配操作如下：

```
....xc....
```

以上例子解释为前4个字符任意，5，6字符为XC，后4个字符也任意，按下例运行：

```
1234XC9088      – match
4523XX9001      – no match
0011XA9912      – no match
9931XC3445      – match
```

注意，“. ”允许匹配ASCII集中任意字符，或为字母，或为数字。

7.2 在行首以^匹配字符串或字符序列

^只允许在一行的开始匹配字符或单词。例如，使用 ls -l命令，并匹配目录。之所以可以这样做是因为ls -l命令结果每行第一个字符是d，即代表一个目录。

```
^d
drwxrwxrw-      – match
-rw-rw-rw-      – no match
drwx-rwxr-x     – match
-rwx-r-x-r-x    – no match
```

回到脚本（1），使用^001，结果将匹配每行开始为001的字符串或单词：

```
1234XC9088      – no match
4523XX9001      – no match
0011XA9912      – match
9931XC3445      – no match
```

可以将各种模式结合使用，例如：

```
^...4xc....
```

结果为：

```
1234XC9088      – match
4523XX9001      – no match
0011XA9912      – no match
9931XC3445      – no match
3224XC193       – no match
```

以上模式表示，在每行开始，匹配任意 3个字符，后跟4XC，最后为任意4个字符。^在正则表达式中使用频繁，因为大量的抽取操作通常在行首。

在行首第4个字符为1，匹配操作表示为：

```
^...1
```

结果为：

1234XC9088 – no match
4523XX9001 – no match
0011XA9912 – match
9931XC3445 – match

行首前4个字符为comp，匹配操作表示为：

```
^comp
```

假定重新定义匹配模式，行首前 4个字符为comp，后面紧跟两个任意字符，并以 ing结尾，一种方法为：

```
^comp..ing
```

以上例子太明显了，不是很有用，但仍讲述了混合使用正则模式的基本概念。

7.3 在行尾以$匹配字符串或字符

可以说$与^正相反，它在行尾匹配字符串或字符， $符号放在匹配单词后。假定要匹配以单词trouble结尾的所有行，操作为：

```
trouble$
```

类似的，使用1d$返回每行以1d结尾的所有字符串。

如果要匹配所有空行，执行以下操作：

```
^$
```

具体分析为匹配行首，又匹配行尾，中间没有任何模式，因此为空行。

如果只返回包含一个字符的行，操作如下：

```
^.$
```

不像空白行，在行首与行尾之间有一个模式，代表任意单字符。

如果在行尾匹配单词jet01，操作如下：

```
jet01$
```

7.4 使用*匹配字符串中的单字符或其重复序列

使用此特殊字符匹配任意字符或字符串的重复多次表达式。例如：

```
compu*t
```

将匹配字符u一次或多次：

computer
computing
compuuuuute

另一个例子：

```
10133*
```

匹配

```
101333
10133
101344444
```

7.5 使用\屏蔽一个特殊字符的含义

有时需要查找一些字符或字符串，而它们包含了系统指定为特殊字符的一个字符。什么是特殊字符？一般意义上讲，下列字符可以认为是特殊字符：

```
$.'"*[]^|()\+?
```

假定要匹配包含字符“.”的各行而“,”代表匹配任意单字符的特殊字符，因此需要屏蔽其含义。操作如下：

```
\.
```

上述模式不认为反斜杠后面的字符是特殊字符，而是一个普通字符，即句点。

假定要匹配包含^的各行，将反斜杠放在它前面就可以屏蔽其特殊含义。如下：

```
\^
```

如果要在正则表达式中匹配以*.pas结尾的所有文件，可做如下操作：

```
\*\.pas
```

即可屏蔽字符*的特定含义。

7.6 使用[]匹配一个范围或集合

使用[]匹配特定字符串或字符串集，可以用逗号将括弧内要匹配的不同字符串分开，但并不强制要求这样做（一些系统提倡在复杂的表达式中使用逗号），这样做可以增加模式的可读性。

使用“-”表示一个字符串范围，表明字符串范围从“-”左边字符开始，到“-”右边字符结束。

如果熟知一个字符串匹配操作，应经常使用[]模式。

假定要匹配任意一个数字，可以使用：

```
[0123456789]
```

然而，通过使用“-”符号可以简化操作：

```
[0-9]
```

或任意小写字母

```
[a-z]
```

要匹配任意字母，则使用：

```
[A-Za-z]
```

表明从A-Z、a-z的字母范围。

如要匹配任意字母或数字，模式如下：

```
[A-Za-z0-9]
```

在字符序列结合使用中，可以用[]指出字符范围。假定要匹配一单词，以s开头，中间有一任意字母，以t结尾，那么操作如下：

```
s[a-z A-Z]t
```

上述过程返回大写或小写字母混合的单词，如仅匹配小写字母，可使用：

```
s[a-z]t
```

如要匹配Computer或computer两个单词，可做如下操作：

```
[Cc]omputer
```

为抽取诸如Scout、shout、bought等单词，使用下列表达式：

```
[ou] .*t
```

匹配以字母o或u开头，后跟任意一个字符任意次，并以t结尾的任意字母。

也许要匹配所有包含system后跟句点的所有单词，这里S可大写或小写。使用如下操作：

```
[S,s]ystem\.
```

[]在指定模式匹配的范围或限制方面很有用。结合使用 *与[]更是有益，例如[A-Za-Z]*将匹配所有单词。

```
[A-Za-z]*
```

注意^符号的使用，当直接用在第一个括号里，意指否定或不匹配括号里内容。

```
[^a-zA-Z]
```

匹配任一非字母型字符，而

```
[^0-9]
```

匹配任一非数字型字符。

通过最后一个例子，应可猜知除了使用^，还有一些方法用来搜索任意一个特殊字符。

7.7 使用\{\}匹配模式结果出现的次数

使用*可匹配所有匹配结果任意次，但如果只要指定次数，就应使用 \{\}，此模式有三种形式，即：

pattern\{n\} 匹配模式出现n次。

pattern\{n,\} 匹配模式出现最少n次。

pattern\{n,m} 匹配模式出现n到m次之间，n,m为0-255中任意整数。

请看第一个例子，匹配字母A出现两次，并以B结尾，操作如下：

```
A\{2\}B
```

匹配值为AAB

匹配A至少4次，使用：

```
A\{4,\}B
```

可以得结果AAAAB或AAAAAAAB，但不能为AAAB。

如给出出现次数范围，例如A出现2次到4次之间：

```
A\{2,4\}B
```

则结果为AAB、AAAB、AAAAB，而不是AB或AAAAAB等。

假定从下述列表中抽取代码：

1234XC9088
4523XX9001
0011XA9912
9931XC3445

格式如下：前4个字符是数字，接下来是xx，最后4个也是数字，操作如下：

```
[0-9]\{4\}XX[0-9]\{4\}
```

具体含义如下：

1) 匹配数字出现4次。

2) 后跟代码xx。

3) 最后是数字出现4次。

结果为：

1234XC9088 – no match
4523XX9001 – match
0011XA9912 – no match
9931XC3445 – no match

在写正则表达式时，可能会有点难度或达不到预期效果，一个好习惯是在写真正的正则表达式前先写下预期的输出结果。这样做，当写错时，可以逐渐修改，以消除意外结果，直至返回正确值。为节省设计基本模式的时间，表 7-2给出一些例子，这些例子并无特别顺序。

表7-2 经常使用的正则表达式举例

^	行首
$	行尾
^[the]	以the开头行
[Ss]igna[lL]	匹配单词signal、signaL、Signal、SignaL
[Ss]igna[lL]\.	同上，但加一句点
[mayMAY]	包含may大写或小写字母的行
^USER$	只包含USER的行
[tty]$	以tty结尾的行
\.	带句点的行
^d..x..x..x	对用户、用户组及其他用户组成员有可执行权限的目录
^[^l]	排除关联目录的目录列表
[.*0]	0之前或之后加任意字符
[000*]	000或更多个
[iI]	大写或小写I
[iI][nN]	大写或小写i或n
[^$]	空行
[^.*$]	匹配行中任意字符串
^.....$	包括6个字符的行
[a- zA-Z]	任意单字符
[a-z][a-z]*	至少一个小写字母
[^0-9\$]	非数字或美元标识
[^0-0A-Za-z]	非数字或字母
[123]	1到3中一个数字
[Dd]evice	单词device或Device
De..ce	前两个字母为De，后跟两个任意字符，最后为ce

（续）

\^q	以^q开始行
^.$	仅有一个字符的行
^\.[0-9][0-9]	以一个句点和两个数字开始的行
'"Device"'	单词device
De[Vv]ice\.	单词Device或device
[0-9]\{2\}-[0-9]\{2\}-[0-9]\{4\}	日期格式dd-mm-yyyy
[0-9]\{3\}\.[0-9]\{3\}\.[0-9]\{3\}\.[0-9]\{3\}	IP地址格式nnn. nnn.nnn.nnn
[^.*$]	匹配任意行

7.8 小结

在shell编程中，一段好的脚本与完美的脚本间的差别之一，就是要熟知正则表达式并学会使用它们。相比较起来，用一个命令抽取一段文本比用三四个命令得出同样的结果要节省许多时间。

既然已经学会了正则表达式中经常使用的基本特殊字符，又通过一些例子简化了其复杂操作，那么现在可以看一些真正的例程了。

好，下面将讲述大量的grep,sed和awk例程。