第9章 AWK 介 绍

如果要格式化报文或从一个大的文本文件中抽取数据包，那么 awk可以完成这些任务。它在文本浏览和数据的熟练使用上性能优异。

整体来说， awk是所有shell过滤工具中最难掌握的，不知道为什么，也许是其复杂的语法或含义不明确的错误提示信息。在学习 awk语言过程中，就会慢慢掌握诸如 Bailing out 和 awk:cmd.Line:等错误信息。可以说awk是一种自解释的编程语言，之所以要在 shell中使用awk是因为awk本身是学习的好例子，但结合 awk与其他工具诸如grep和sed，将会使shell编程更加容易。

本章没有讲述awk的全部特性，也不涉及 awk的深层次编程，（这些可以在专门讲述 awk的书籍中找到）。本章仅注重于讲述使用 awk执行行操作及怎样从文本文件和字符串中抽取信息。

本章内容有：

- 抽取域。
- 匹配正则表达式。
- 比较域。
- 向awk传递参数。
- 基本的awk行操作和脚本。

本书几乎所有包含 awk命令的脚本都结合了 sed和grep，以从文本文件和字符串中抽取信息。为获得所需信息，文本必须格式化，意即用域分隔符划分抽取域，分隔符可能是任意字符，在以后讲述awk时再详细讨论。

awk以发展这种语言的人 Aho.Weninberger和Kernigham命名。还有 nawk和gawk，它们扩展了文本特性，但本章不予讨论。

awk语言的最基本功能是在文件或字符串中基于指定规则浏览和抽取信息。 awk抽取信息后，才能进行其他文本操作。完整的 awk脚本通常用来格式化文本文件中的信息。

9.1 调用awk

有三种方式调用awk，第一种是命令行方式，如：

awk [-F field-separator] 'commands' input-file(s)

这里， commands是真正的awk命令。本章将经常使用这种方法。

上面例子中，[-F域分隔符]是可选的，因为awk使用空格作为缺省的域分隔符，因此如果要浏览域间有空格的文本，不必指定这个选项，但如果要浏览诸如 passwd文件，此文件各域以冒号作为分隔符，则必须指明 -F选项，如：

awk -F: 'commands' input-file

第二种方法是将所有 awk命令插入一个文件，并使 awk程序可执行，然后用 awk命令解释器作为脚本的首行，以便通过键入脚本名称来调用它。

第三种方式是将所有的awk命令插入一个单独文件，然后调用：

awk -f awk-script-file input-files(s)

-f选项指明在文件awk_script_file中的awk脚本，input_file(s)是使用awk进行浏览的文件名。

9.2 awk脚本

在命令中调用awk时，awk脚本由各种操作和模式组成。

如果设置了-F选项，则awk每次读一条记录或一行，并使用指定的分隔符分隔指定域，但如果未设置-F选项，awk假定空格为域分隔符，并保持这个设置直到发现一新行。当新行出现时，awk命令获悉已读完整条记录，然后在下一个记录启动读命令，这个读进程将持续到文件尾或文件不再存在。

参照表9-1，awk每次在文件中读一行，找到域分隔符（这里是符号 #），设置其为域n，直至一新行（这里是缺省记录分隔符），然后，划分这一行作为一条记录，接着 awk再次启动下一行读进程。

表9-1 awk读文件记录的方式

域1	分隔符	域2	分隔符	域3	分隔符	域4及换行
P.Bunny(记录1)	#	02/99	#	48	#	Yellow \n
J.Troll(记录2)	#	07/99	#	4842	#	Brown-3 \n

9.2.1 模式和动作

任何awk语句都由模式和动作组成。在一个 awk脚本中可能有许多语句。模式部分决定动作语句何时触发及触发事件。处理即对数据进行的操作。如果省略模式部分，动作将时刻保持执行状态。

模式可以是任何条件语句或复合语句或正则表达式。模式包括两个特殊字段 BEGIN和END。使用BEGIN语句设置计数和打印头。BEGIN语句使用在任何文本浏览动作之前，之后文本浏览动作依据输入文件开始执行。 END语句用来在awk完成文本浏览动作后打印输出文本总数和结尾状态标志。如果不特别指明模式， awk总是匹配或打印行数。

实际动作在大括号{}内指明。动作大多数用来打印，但是还有些更长的代码诸如 if和循环（looping）语句及循环退出结构。如果不指明采取动作， awk将打印出所有浏览出来的记录。

下面将深入讲解这些模式和动作。

9.2.2 域和记录

awk执行时，其浏览域标记为 $1，$2...$n。这种方法称为域标识。使用这些域标识将更容易对域进行进一步处理。

使用$1,$3表示参照第1和第3域，注意这里用逗号做域分隔。如果希望打印一个有 5个域的记录的所有域，不必指明 $1,$2,$3,$4,$5，可使用$0，意即所有域。Awk浏览时，到达一新行，即假定到达包含域的记录末尾，然后执行新记录下一行的读动作，并重新设置域分隔。注意执行时不要混淆符号$和shell提示符$，它们是不同的。

为打印一个域或所有域，使用print命令。这是一个awk动作（动作语法用圆括号括起来）。

1. 抽取域

真正执行前看几个例子，现有一文本文件 grade.txt，记录了一个称为柔道数据库的行信息。

```
$ pg grade.txt
M.Tansley    05/99     48311   Green     8    40    44
J.Lulu       06/99     48317   green     9    24    26
P.Bunny      02/99     48      Yellow    12   35    28
J.Troll      07/99     4842    Brown-3   12   26    26
L.Tansley    05/99     4712    Brown-2   12   30    28
```

此文本文件有7个域，即（1）名字、（2）升段日期、（3）学生序号、（4）腰带级别、（5）年龄、（6）目前比赛积分、（7）比赛最高分。

因为域间使用空格作为域分隔符，故不必用 -F选项划分域，现浏览文件并导出一些数据。

在例子中为了利于显示，将空格加宽使各域看得更清晰。

2. 保存awk输出

有两种方式保存 shell提示符下awk脚本的输出。最简单的方式是使用输出重定向符号 >文件名，下面的例子重定向输出到文件 wow。

```
$ awk '{print $0}' grade.txt >wow
```

使用这种方法要注意，显示屏上不会显示输出结果。因为它直接输出到文件。只有在保证输出结果正确时才会使用这种方法。它也会重写硬盘上同名数据。

第二种方法是使用 tee命令，在输出到文件的同时输出到屏幕。在测试输出结果正确与否时多使用这种方法。例如输出重定向到文件 delete_me_and_die，同时输出到屏幕。使用这种方法，在awk命令结尾写入 | tee delete_me_and_die。

```
$ awk '{print $0}' grade.txt | tee delete_me_and_die
```

3. 使用标准输入

在深入讲解这一章之前，先对 awk脚本的输入方法简要介绍一下。实际上任何脚本都是从标准输入中接受输入的。为运行本章脚本，使用 awk脚本输入文件格式，例如：

```
$ belts.awk grade_student.txt
```

也可替代使用下述格式：

使用重定向方法：

```
$ belts.awk < grade2.txt
```

或管道方法：

```
$ grade2.txt|belts.awk
```

4. 打印所有记录

```
$ awk '{print $0}' grade.txt
```

awk读每一条记录。因为没有模式部分，只有动作部分 {print $0}(打印所有记录)，这个动作必须用花括号括起来。上述命令打印整个文件。

```
M.Tansley    05/99     48311   Green     8    40    44
J.Lulu       06/99     48317   green     9    24    26
P.Bunny      02/99     48      Yellow    12   35    28
J.Troll      07/99     4842    Brown-3   12   26    26
L.Tansley    05/99     4712    Brown-2   12   30    28
```

5. 打印单独记录

假定只打印学生名字和腰带级别，通过查看域所在列，可知为 field-1和field-4，因此可以使用$1和$4，但不要忘了加逗号以分隔域。

```
$ awk  '{print $1,$4}' grade.txt
M.Tansley    Green
J.Lulu       green
P.Bunny      Yellow
J.Troll      Brown-3
L.Tansley    Brown-2
```

6. 打印报告头

上述命令输出在名字和腰带级别之间用一些空格使之更容易划分，也可以在域间使用 tab 键加以划分。为加入tab键，使用tab键速记引用符 \t，后面将对速记引用加以详细讨论。也可以为输出文本加入信息头。本例中加入 name和belt及下划线。下划线使用 \n，强迫启动新行，并在\n下一行启动打印文本操作。打印信息头放置在 BEGIN模式部分，因为打印信息头被界定为一个动作，必须用大括号括起来。在 awk查看第一条记录前，信息头被打印。

```
$ awk  'BEGIN {print "Name     Belt\n-------------------------------"}
{print $1"\t"$4}' grade.txt

Name             Belt
---------------------------------------
M.Tansley        Green
J.Lulu           green
P.Bunny          Yellow
J.Troll          Brown-3
L.Tansley        Brown-3
```

7. 打印信息尾

如果在末行加入end of report信息，可使用END语句。END语句在所有文本处理动作执行完之后才被执行。 END语句在脚本中的位置放置在主要动作之后。下面简单打印头信息并告之查询动作完成。

```
$ awk 'BEGIN {print "Name\n-------"} {print $1} END {"end-of-report"}'
grade.txt
Name
-------
M.Tansley
J.Lulu
P.Bunny
J.Troll
L.Tansley
end-of-report
```

8. awk错误信息提示

几乎可以肯定，在使用 awk时，将会在命令中碰到一些错误。 awk将试图打印错误行，但由于大部分命令都只在一行，因此帮助不大。

系统给出的显示错误信息提示可读性不好。使用上述例子，如果丢了一个双引号， awk将返回：

```
$ awk 'BEGIN {print "Name\n-------} {print $1} END {"end-of-report"}''
grade.txt

awk: cmd. line:1: BEGIN {print "Name\n-------} {print $1} END
```

```
{"end-of -report"}
awk: cmd. line:1: ^ unterminated string
```

当第一次使用awk时，可能被错误信息搅得不知所措，但通过长时间和不断的学习，可总结出以下规则。在碰到awk错误时，可相应查找：

- 确保整个awk命令用单引号括起来。
- 确保命令内所有引号成对出现。
- 确保用花括号括起动作语句，用圆括号括起条件语句。
- 可能忘记使用花括号，也许你认为没有必要，但 awk不这样认为，将按之解释语法。

如果查询文件不存在，将得到下述错误信息：

```
$ awk 'END {print NR}' grades.txt

awk: cmd. line:2: fatal: cannot open file 'grades.txt' for
reading (No such file or directory)
```

9. awk键盘输入

如果在命令行并没有输入文件grade.txt，将会怎样？

```
$ awk 'BEGIN {print "Name  Belt\n--------------------------------"}
      {print $1"\t"$4}'

Name              Belt
--------------------------------
>
```

BEGIN部分打印了文件头，但awk最终停止操作并等待，并没有返回shell提示符。这是因为awk期望获得键盘输入。因为没有给出输入文件， awk假定下面将会给出。如果愿意，顺序输入相关文本，并在输入完成后敲<Ctrl-D>键。如果敲入了正确的域分隔符， awk会像第一个例子一样正常处理文本。这种处理并不常用，因为它大多应用于大量的打印稿。

9.2.3 awk中正则表达式及其操作

在grep一章中，有许多例子用到正则表达式，这里将不使用同样的例子，但可以使用条件操作讲述awk中正则表达式的用法。

这里正则表达式用斜线括起来。例如，在文本文件中查询字符串 Green，使用/Green/可以查出单词Green的出现情况。

9.2.4 元字符

这里是awk中正则表达式匹配操作中经常用到的字符，详细情况请参阅本书第 7章正则表达式概述。

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

这里有两个字符第7章没有讲到，因为它们只适用于 awk而不适用于grep或sed。它们是：

+ 使用+匹配一个或多个字符。

？ 匹配模式出现频率。例如使用 /XY?Z/匹配XYZ或YZ。

9.2.5 条件操作符

表9-2给出awk条件操作符，后面将给出其用法。

表9-2 awk条件操作符

操 作 符	描 述	操 作 符	描 述
<	小于	>=	大于等于
<=	小于等于	~	匹配正则表达式
==	等于	!~	不匹配正则表达式
!=	不等于		

1. 匹配

为使一域号匹配正则表达式，使用符号‘～’后紧跟正则表达式，也可以用 if语句。awk中if后面的条件用（）括起来。

观察文件grade.txt，如果只要打印brown腰带级别可知其所在域为 field-4，这样可以写出表达式{if($4~/brown/) print }意即如果field-4包含brown，打印它。如果条件满足，则打印匹配记录行。可以编写下面脚本，因为这是一个动作，必须用花括号 {}括起来。

```
$ awk  {if($4~/Brown/) print $0}' grade.txt
J.Troll     07/99   4842    Brown-3   12    26      26
L.Tansley   05/99   4712    Brown-2   12    30      28
```

匹配记录找到时，如果不特别声明， awk缺省打印整条记录。使用 if语句开始有点难，但不要着急，因为有许多方法可以跳过它，并仍保持同样结果。下面例子意即如果记录包含模式brown，就打印它：

```
$ awk  '$0 ~ /Brown/' grade.txt
J.Troll     07/99     4842   Brown-3  12    26      26
L.Tansley   05/99     4712   Brown-2  12    30      28
```

2. 精确匹配

假定要使字符串精确匹配，比如说查看学生序号 48，文件中有许多学生序号包含 48，如果在field-3中查询序号48，awk将返回所有序号带48的记录：

```
$ awk  '{if($3~/48/) print $0}' grade.txt
M.Tansley   05/99     48311  Green    8     40    44
J.Lulu      06/99     48317  green    9     24    26
P.Bunny     02/99     48     Yellow   12    35    28
J.Troll     07/99     4842   Brown-3  12    26    26
```

为精确匹配48，使用等号==，并用单引号括起条件。例如 $3==“48”，这样确保只有48序号得以匹配，其余则不行。

```
$ awk '$3=="48" {print $0}' grade.txt
P.Bunny     02/99    48      Yellow   12    35    28
```

3. 不匹配

有时要浏览信息并抽取不匹配操作的记录，与 ~相反的符号是!~，意即不匹配。像原来使用查询brown腰带级别的匹配操作一样，现在看看不匹配情况。表达式 $0 !~/brown/，意即查询不包含模式brown腰带级别的记录并打印它。

注意，缺省情况下， awk将打印所有匹配记录，因此这里不必加入动作部分。

```
$ awk  '$0 !~ /Brown/' grade.txt
M.Tansley   05/99     48311  Green    8     40    44
J.Lulu      06/99     48317  green    9     24    26
P.Bunny     02/99     48     Yellow   12    35    28
```

可以只对field-4进行不匹配操作，方法如下：

```
$ awk '{if($4!~/Brown/) print $0}' grade.txt
M.Tansley   05/99     48311  Green    8     40    44
J.Lulu      06/99     48317  green    9     24    26
P.Bunny     02/99     48     Yellow   12    35    28
```

如果只使用命令awk$4 !="brown"{print $0} grade.txt，将返回错误结果，因为用引号括起了brown，将只匹配‘ brown而不匹配brown-2和brown-3，当然，如果想要查询非brown-2的腰带级别，可做如下操作：

```
$ awk '$4 != "Brown-2" {print $0}' grade.txt
```

4. 小于

看看哪些学生可以获得升段机会。测试这一点即判断目前级别分 field-6是否小于最高分field-7，在输出结果中，加入这一改动很容易。

```
$ awk '{if ($6 < $7) print $0 "$1 Try better at the next comp"}'
grade.txt
M.Tansley Try better at the next comp
J.Lulu Try better at the next comp
```

5. 小于等于

对比小于，小于等于只在操作符上做些小改动，满足此条件的记录也包括上面例子中的输出情况。

```
$ awk '{if ($6 <= $7) print $1}' grade.txt
M.Tansley
J.Lulu
J.Troll
```

6. 大于

大于符号大家都熟知，请看例子：

```
$ awk '{if ($6 > $7) print $1}' grade.txt
L.Tansley
P.Bunny
```

希望读者已经掌握了操作符的基本用法。

7. 设置大小写

为查询大小写信息，可使用[]符号。在测试正则表达式时提到可匹配[]内任意字符或单词，因此若查询文件中级别为green的所有记录，不论其大小写，表达式应为‘ /[Gg]reen/：’

```
$ awk '/[Gg]reen/' grade.txt
M.Tansley   05/99     48311  Green    8     40    44
J.Lulu      06/99     48317  green    9     24    26
```

8. 任意字符

抽取名字，其记录第一域的第四个字符是 a，使用句点.。表达式/^...a/意为行首前三个字符任意，第四个是a，尖角符号代表行首。

```
$ awk '$1 ~/^ ... a/' grade.txt
M.Tansley   05/99     48311  Green    8     40    44
L.Tansley   05/99     4712   Brown-2  12    30    28
```

9. 或关系匹配

为抽取级别为yellow或brown的记录，使用竖线符 |。意为匹配 | 两边模式之一。注意，使用竖线符时，语句必须用圆括号括起来。

```
$ awk '$0~/(Yellow|Brown)/' grade.txt
P.Bunny     02/99     48      Yellow   12   35    28
J.Troll     07/99     4842    Brown-3  12   26    26
L.Tansley   05/99     4712    Brown-2  12   30    28
```

上面例子输出所有级别为Yellow或Brown的记录。

使用这种方法在查询级别为Green或green时，可以得到与使用[]表达式相同的结果。

```
$ awk '$0~/(Green|green)/' grade.txt
M.Tansley   05/99     48311   Green    8     40    44
J.Lulu      06/99     48317   green    9     24    26
```

10. 行首

不必总是使用域号。如果查询文本文件行首包含48的代码，可简单使用下面^符号：

```
$ awk '/^48/' input-file
```

这里讲述了在awk中怎样使用第7章中涉及的表达式。像第7章的开头提到的，所有表达式（除字符重复出现外）在awk中都是合法的。

复合模式或复合操作符用于形成复杂的逻辑操作，复杂程度取决于编程者本人。有必要了解的是，复合表达式即为模式间通过使用下述各表达式互相结合起来的表达式：

&& AND：语句两边必须同时匹配为真。

|| OR：语句两边同时或其中一边匹配为真。

! 非 求逆

11. AND

打印记录，使其名字为‘ P.Bunny且级别为Yellow，使用表达式($1=="P.Bunny"&&$4=="Yellow")，意为&&两边匹配均为真。完整命令如下：

```
$ awk '{if ($1=="P.Bunny" && $4=="Yellow")print $0}' grade.txt
P.Bunny     02/99     48      Yellow   12   35    28
```

12. Or

如果查询级别为Yellow或Brown，使用或命令。意为“ || ”符号两边的匹配模式之一或全部为真。

```
$ awk '{if ($4=="Yellow" || $4~/Brown/) print $0}' grade.txt
P.Bunny     02/99     48      Yellow   12   35    28
J.Troll     07/99     4842    Brown-3  12   26    26
L.Tansley   05/99     4712    Brown-2  12   30    28
```

9.2.6 awk内置变量

awk有许多内置变量用来设置环境信息。这些变量可以被改变。表 9-3显示了最常使用的一些变量，并给出其基本含义。

表9-3 awk内置变量

ARGC	命令行参数个数
ARGV	命令行参数排列
ENVIRON	支持队列中系统环境变量的使用
FILENAME	awk浏览的文件名
FNR	浏览文件的记录数
FS	设置输入域分隔符，等价于命令行-F选项

(续)

NF	浏览记录的域个数
NR	已读的记录数
OFS	输出域分隔符
ORS	输出记录分隔符
RS	控制记录分隔符

ARGC支持命令行中传入awk脚本的参数个数。ARGV是ARGC的参数排列数组，其中每一元素表示为ARGV[n]，n为期望访问的命令行参数。

ENVIRON支持系统设置的环境变量，要访问单独变量，使用实际变量名，例如ENVIRON[“EDITOR”]=“Vi”。

FILENAME支持awk脚本实际操作的输入文件。因为awk可以同时处理许多文件，因此如果访问了这个变量，将告之系统目前正在浏览的实际文件。

FNR支持awk目前操作的记录数。其变量值小于等于 NR。如果脚本正在访问许多文件，每一新输入文件都将重新设置此变量。

FS用来在awk中设置域分隔符，与命令行中-F选项功能相同。缺省情况下为空格。如果用逗号来作域分隔符，设置FS=", "。

NF支持记录域个数，在记录被读之后再设置。

OFS允许指定输出域分隔符，缺省为空格。如果想设置为 #，写入OFS="#"。

ORS为输出记录分隔符，缺省为新行（\n）。

RS是记录分隔符，缺省为新行(\n)。

9.2.7 NF、NR和FILENAME

下面看一看awk内置变量的例子。

要快速查看记录个数，应使用 NR。比如说导出一个数据库文件后，如果想快速浏览记录个数，以便对比于其初始状态，查出导出过程中出现的错误。使用 NR将打印输入文件的记录个数。print NR放在END语法中。

```
$ awk 'END {print NR}' grade.txt
```

以下例子中，所有学生记录被打印，并带有其记录号。使用 NF变量显示每一条读记录中有多少个域，并在END部分打印输入文件名。

```
$ awk '{print NF,NR,$0}END{print FILENAME}' grade.txt
7  1 M.Tansley  05/99   48311   Green    8       40      44
7  2 J.Lulu     06/99   48317   green    9       24      26
7  3 P.Bunny    02/99   48      Yellow   12      35      28
7  4 J.Troll    07/99   4842    Brown-3  12      26      26
7  5 L.Tansley  05/99   4712    Brown-2  12      30      28
grade.txt
```

在从文件中抽取信息时，最好首先检查文件中是否有记录。下面的例子只有在文件中至少有一个记录时才查询 Brown级别记录。使用 AND复合语句实现这一功能。意即至少存在一个记录后，查询字符串Brown，最后打印结果。

```
$ awk '{if (NR >0 && $4~/Brown/)print $0}' grade.txt
J.Troll      07/99    4842   Brown-3  12   26      26
L.Tansley    05/99    4712   Brown-2  12   30      28
```

NF的一个强大功能是将变量$PWD的返回值传入awk并显示其目录。这里需要指定域分隔符/。

```
$ pwd
/usr/local/etc
$ echo $PWD | awk -F/ '{print $NF}'
etc
```

另一个例子是显示文件名。

```
$ echo "/usr/local/etc/rc.sybase" | awk -F/ '{print $NF}'
rc.sybase
```

9.2.8 awk操作符

在awk中使用操作符，基本表达式可以划分为数字型、字符串型、变量型、域及数组元素，前面已经讲过一些。下面列出其完整列表。

在表达式中可以使用下述任何一种操作符。

= += *= / = %= ^ =	赋值操作符
?	条件表达操作符
|| && !	并、与、非（上一节已讲到）
~!~	匹配操作符，包括匹配和不匹配
< <= == != >>	关系操作符
+ - * / % ^	算术操作符
+ + --	前缀和后缀

前面已经讲到了其中几种操作，下面继续讲述未涉及的部分。

1. 设置输入域到域变量名

在awk中，设置有意义的域名是一种好习惯，在进行模式匹配或关系操作时更容易理解。一般的变量名设置方式为name=$n，这里name为调用的域变量名，n为实际域号。例如设置学生域名为name，级别域名为belt，操作为name=$1;belts=$4。注意分号的使用，它分隔awk命令。下面例子中，重新赋值学生名域为 name，级别域为belts。查询级别为Yellow的记录，并最终打印名称和级别。

```
$ awk '{name=$1;belts=$4; if(belts ~/Yellow/)print name" is belt
"belts}' grade.txt
P.Bunny is belt Yellow
```

2. 域值比较操作

有两种方式测试一数值域是否小于另一数值域。

1) 在BEGIN中给变量名赋值。

2) 在关系操作中使用实际数值。

通常在BEGIN部分赋值是很有益的，可以在awk表达式进行改动时减少很多麻烦。

使用关系操作必须用圆括号括起来。

下面的例子查询所有比赛中得分在27点以下的学生。

用引号将数字引用起来是可选的，“27”、27产生同样的结果。

```
$ awk '{if($6 < 27)print$0}' grade.txt
J.Lulu      06/99   48317   green   9     24      26
J.Troll     07/99   4842    Brown-3 12    26      26
```

第二个例子中给数字赋以变量名BASELINE和在BEGIN部分给变量赋值，两者意义相同。

```
$ awk 'BEGIN {BASELINE="27  "}{if($6 < BASELINE)print$0}'  grade.txt
J.Lulu      06/99   48317   green   9     24      26
J.Troll     07/99   4842    Brown-3 12    26      26
```

3. 修改数值域取值

当在awk中修改任何域时，重要的一点是要记住实际输入文件是不可修改的，修改的只是保存在缓存里的awk复本。awk会在变量NR或NF变量中反映出修改痕迹。

为修改数值域，简单的给域标识重赋新值，如： $1=$1+5，会将域1数值加5，但要确保赋值域其子集为数值型。

修改M.Tansley的目前级别分域，使其数值从 40减为39，使用赋值语句$6=$6-1，当然在实施修改前首先要匹配域名。

```
$ awk '{if($1=="M.Tansley") $6=$6-1; print $1, $6, $7}' grade.txt
M.Tansley   39  44
J.Lulu      24  26
P.Bunny     35  28
J.Troll     26  26
L.Tansley   30  28
```

4. 修改文本域

修改文本域即对其重新赋值。需要做的就是赋给一个新的字符串。在 J.Troll中加入字母，使其成为J.L.Troll，表达式为$1="J.L.Troll"，记住字符串要使用双秒号（""），并用圆括号括起整个语法。

```
$ awk '{if($1=="J.Troll") ($1="J.L.Troll"); print $1}' grade.txt
M.Tansley
J.Lulu
P.Bunny
J.L.Troll
L.Tansley
```

5. 只显示修改记录

上述例子均是对一个小文件的域进行修改，因此打印出所有记录查看修改部分不成问题，但如果文件很大，记录甚至超过 100，打印所有记录只为查看修改部分显然不合情理。在模式后面使用花括号将只打印修改部分。取得模式，再根据模式结果实施操作，可能有些抽象，现举一例，只打印修改部分。注意花括号的位置。

```
$ awk '{if($1=="J.Troll") {$1="J.L.Troll";print $1}}' grade.txt
J.L.Troll
```

6. 创建新的输出域

在awk中处理数据时，基于各域进行计算时创建新域是一种好习惯。创建新域要通过其他域赋予新域标识符。如创建一个基于其他域的加法新域{$4=$2+$3}，这里假定记录包含3个域，则域4为新建域，保存域2和域3相加结果。

在文件grade.txt中创建新域 8保存域目前级别分与域最高级别分的减法值。表达式为‘ {$8=$7-$6} ’，语法首先测试域目前级别分小于域最高级别分。新域因此只打印其值大于零的学生名称及其新域值。在BEGIN部分加入tab键以对齐报告头。

```
$ awk 'BEGIN{ print "Name\t Difference"}{if($6 <$7) {$8=$7--$6; print
$1,$8}}' grade.txt
Name        Difference
```

```
M.Tansley    4
J.Lulu       2
```

当然可以创建新域，并赋给其更有意义的变量名。例如：

```
$ awk 'BEGIN{ print "Name\t Difference"}{if($6 <$7) {diff=$7-$6; print
$1,diff}}' grade.txt
M.Tansley    4
J.Lulu       2
```

7. 增加列值

为增加列数或进行运行结果统计，使用符号 +=。增加的结果赋给符号左边变量值，增加到变量的域在符号右边。例如将 $1加入变量total，表达式为total+=$1。列值增加很有用。许多文件都要求统计总数，但输出其统计结果十分繁琐。在 awk中这很简单，请看下面的例子。

将所有学生的‘目前级别分’加在一起，方法是 tot+=$6，tot即为awk浏览的整个文件的域6结果总和。所有记录读完后，在 END部分加入一些提示信息及域 6总和。不必在 awk中显示说明打印所有记录，每一个操作匹配时，这是缺省动作。

```
$ awk '(tot+=$6); END{print "Club student total points :" tot}'
grade.txt
M.Tansley    05/99      48311   Green     8      40    44
J.Lulu       06/99      48317   green     9      24    26
P.Bunny      02/99      48      Yellow    12     35    28
J.Troll      07/99      4842    Brown-3   12     26    26
L.Tansley    05/99      4712    Brown-2   12     30    28
Club student total points :155
```

如果文件很大，你只想打印结果部分而不是所有记录，在语句的外面加上圆括号（）即可。

```
$ awk '{(tot+=$6)}; END{print "Club student total points :" tot}'
grade.txt
Club student total points :155
```

8. 文件长度相加

在目录中查看文件时，如果想快速查看所有文件的长度及其总和，但要排除子目录，使用ls -l命令，然后管道输出到awk，awk首先剔除首字符为d（使用正则表达式）的记录，然后将文件长度列相加，并输出每一文件长度及在 END部分输出所有文件的长度。

本例中，首先用 ls -l命令查看一下文件属性。注意第二个文件属性首字符为 d，说明它是一个目录，文件长度是第 5列，文件名是第 9列。如果系统不是这样排列文件名及其长度，应适时加以改变。

```
          -rw-r--r--   1 root   root      80   Apr  11  18:56  acc.txt
          drwx------   2 root   root      1024 Mar  26  20:53  nsmail
Columns   1            2 3      4         5    6    7   8      9
```

下面的正则表达式表明必须匹配行首，并排除字符 d，表达式为^[^d]。

使用此模式打印文件名及其长度，然后将各长度相加放入变量 tot中。

```
$ ls -l | awk ' /^[^d]/ {print $9"\t"$5} {tot+=$5} END
{print "total KB:"tot}'
dev_pkg.fail     345
failedlogin      12416
messages         4260
sulog            12810
```

```
utmp            1856
wtmp            7104
total KB:41351
```

9.2.9 内置的字符串函数

awk有许多强大的字符串函数，见表9-4。

表9-4 awk内置字符串函数

gsub(r,s)	在整个$0中用s替代r
gsub(r,s,t)	在整个t中用s替代r
index(s,t)	返回s中字符串t的第一位置
length(s)	返回s长度
match(s,r)	测试s是否包含匹配r的字符串
split(s,a,fs)	在fs上将s分成序列a
sprint(fmt,exp)	返回经fmt格式化后的exp
sub(r,s)	用$0中最左边最长的子串代替s
substr(s,p)	返回字符串s中从p开始的后缀部分
substr(s,p,n)	返回字符串s中从p开始长度为n的后缀部分

gsub函数有点类似于sed查找和替换。它允许替换一个字符串或字符为另一个字符串或字符，并以正则表达式的形式执行。第一个函数作用于记录 $0，第二个gsub函数允许指定目标，然而，如果未指定目标，缺省为$0。

index（s，t）函数返回目标字符串s中查询字符串t的首位置。length函数返回字符串s字符长度。match函数测试字符串s是否包含一个正则表达式r定义的匹配。split使用域分隔符fs将字符串s划分为指定序列a。sprint函数类似于printf函数（以后涉及），返回基本输出格式fmt的结果字符串exp。sub（r，s）函数将用s替代$0中最左边最长的子串，该子串被（ r）匹配。sub（s，p）返回字符串s在位置p后的后缀。substr（s，p，n）同上，并指定子串长度为n。

现在看一看awk中这些字符串函数的功能。

1. gsub

要在整个记录中替换一个字符串为另一个，使用正则表达式格式， /目标模式/，替换模式/。例如改变学生序号4842到4899：

```
$ awk 'gsub(/4842/,4899) {print $0}' grade.txt
J.Troll     07/99     4899   Brown-3  12    26    26
```

2. index

查询字符串s中t出现的第一位置。必须用双引号将字符串括起来。例如返回目标字符串Bunny中ny出现的第一位置，即字符个数。

```
$ awk 'BEGIN {print index("Bunny","ny")}' grade.txt
4
```

3. length

返回所需字符串长度，例如检验字符串 J.Troll返回名字及其长度，即人名构成的字符个数。

```
$ awk '$1=="J.Troll" {print length($1)" "$1}' grade.txt
7 J.Troll
```

还有一种方法，这里字符串加双引号。

```
$ awk 'BEGIN {print length("A FEW GOOD MEN")}'
14
```

4. match

match测试目标字符串是否包含查找字符的一部分。可以对查找部分使用正则表达式，返回值为成功出现的字符排列数。如果未找到，返回 0，第一个例子在ANCD中查找d。因其不存在，所以返回0。第二个例子在ANCD中查找D。因其存在，所以返回ANCD中D出现的首位置字符数。第三个例子在学生J.Lulu中查找u。

```
$ awk 'BEGIN {print match("ANCD",/d/)}'
0
$ awk 'BEGIN {print match("ANCD",/C/)}'
3
$ awk '$1=="J.Lulu" {print match($1,"u")}' grade.txt
4
```

5. split

使用split返回字符串数组元素个数。工作方式如下：如果有一字符串，包含一指定分隔符 - ，例如AD2-KP9-JU2-LP-1，将之划分成一个数组。使用 split，指定分隔符及数组名。此例中，命令格式为("AD2-KP9-JU2-LP-1"，parts_array，"-")，split然后返回数组下标数，这里结果为4。

还有一个例子使用不同的分隔符。

```
$ awk 'BEGIN {print split("123#456#678", myarray, "#")}'
3
```

这个例子中，split返回数组myarray的下标数。数组myarray取值如下：

Myarray[1]="123"
Myarray[2]="456"
Myarray[3]="678"

本章结尾部分讲述数组概念。

6. sub

使用sub发现并替换模式的第一次出现位置。字符串 STR包含 ‘ poped popo pill ’，执行下列sub命令sub（/op/，"op"，STR)。模式op第一次出现时，进行替换操作，返回结果如下：‘ pOPed pope pill ’。

本章文本文件中，学生J.Troll的记录有两个值一样，“ 目前级别分 ”与“ 最高级别分 ”。只改变第一个为29，第二个仍为24不动，操作命令为sub（/26/，"29"，$0)，只替换第一个出现24的位置。注意J.Troll记录需存在。

```
$ awk '$1=="J.Troll" sub(/26/,"29",$0)' grade.txt
M.Tansley   05/99     48311  Green    8     40    44
J.Lulu      06/99     48317  green    9     24    29
P.Bunny     02/99     48     Yellow   12    35    28
J.Troll     07/99     4842   Brown-3  12    29    26
L.Tansley   05/99     4712   Brown-2  12    30    28
```

7. substr

substr是一个很有用的函数。它按照起始位置及长度返回字符串的一部分。例子如下：

```
$ awk '$1=="L.Tansley" {print substr($1,1,5)}' grade.txt
L.Tan
```

上面例子中，指定在域1的第一个字符开始，返回其前面5个字符。

如果给定长度值远大于字符串长度， awk将从起始位置返回所有字符，要抽取 L Tansl-ey的姓，只需从第3个字符开始返回长度为7。可以输入长度99，awk返回结果相同。

```
$ awk '$1=="L.Tansley" {print substr($1,3,99)}' grade.txt
Tansley
```

substr的另一种形式是返回字符串后缀或指定位置后面字符。这里需要给出指定字符串及其返回字串的起始位置。例如，从文本文件中抽取姓氏，需操作域 1，并从第三个字符开始：

```
$ awk '{print substr($1,3)}' grade.txt
Tansley
Lulu
Bunny
Troll
Tansley
```

还有一个例子，在 BEGIN部分定义字符串，在 END部分返回从第 t个字符开始抽取的子串。

```
$ awk 'BEGIN {STR="A FEW GOOD MEN"}END{print substr(STR,7)}' grade.txt
GOOD MEN
```

8. 从shell中向awk传入字符串

本章开始已经提到过， awk脚本大多只有一行，其中很少是字符串表示的。本书大多要求在一行内完成awk脚本，这一点通过将变量传入 awk命令行会变得很容易。现就其基本原理讲述一些例子。

使用管道将字符串stand-by传入awk，返回其长度。

```
$ echo "Stand-by" |awk '{print length($0)}'
8
```

设置文件名为一变量，管道输出到 awk，返回不带扩展名的文件名。

```
$ STR="mydoc.txt"
$ echo $STR| awk '{print substr($STR,1,5)}'
mydoc
```

设置文件名为一变量，管道输出到 awk，只返回其扩展名。

```
$ STR="mydoc.txt"
$ echo $STR| awk '{print substr($STR,7)}'
txt
```

9.2.10 字符串屏蔽序列

使用字符串或正则表达式时，有时需要在输出中加入一新行或查询一元字符。

打印一新行时，（新行为字符\n），给出其屏蔽序列，以不失其特殊含义，用法为在字符串前加入反斜线。例如使用\n强迫打印一新行。

如果使用正则表达式，查询花括号（ {}），在字符前加反斜线，如/\{/，将在awk中失掉其特殊含义。

表9-5列出awk识别的另外一些屏蔽序列

表9-5 awk中使用的屏蔽序列

\b	退格键	\t	tab键
\f	走纸换页	\ddd	八进制值
\n	新行	\c	任意其他特殊字符，例如\\为反斜线符号
\r	回车键		

使用上述符号，打印May Day，中间夹tab键，后跟两个新行，再打印May Day，但这次使用八进制数104、141、171、分别代表D、a、y。

```
$ awk 'BEGIN {print"\n\May\tDay\n\nMay \104\141\171"}'
May     Day

May     Day
```

注意，\104为D的八进制ASCII码，\141为a的八进制ASCII码，等等。

9.2.11 awk输出函数printf

目前为止，所有例子的输出都是直接到屏幕，除了 tab键以外没有任何格式。 awk提供函数printf，拥有几种不同的格式化输出功能。例如按列输出、左对齐或右对齐方式。

每一种printf函数（格式控制字符）都以一个 %符号开始，以一个决定转换的字符结束。转换包含三种修饰符。

printf函数基本语法是printf（[格式控制符]，参数），格式控制字符通常在引号里。

9.2.12 printf修饰符

表9-6 awk printf修饰符

-	左对齐
Width	域的步长，用0表示0步长
.prec	最大字符串长度，或小数点右边的位数

表9-7 awk printf格式

%c	ASCII字符
%d	整数
%e	浮点数，科学记数法
%f	浮点数，例如（123.44）
%g	awk决定使用哪种浮点数转换e或者f
%o	八进制数
%s	字符串
%x	十六进制数

1. 字符转换

观察ASCII码中65的等价值。管道输出65到awk。printf进行ASCII码字符转换。这里也加入换行，因为缺省情况下printf不做换行动作。

```
$ echo "65" | awk '{printf "%c\n",$0}'
A
```

当然也可以按同样方式使用awk得到同样结果。

```
$ awk 'BEGIN {printf "%c\n",65}'
A
```

所有的字符转换都是一样的，下面的例子表示进行浮点数转换后‘ 999 ’的输出结果。整数传入后被加了六个小数点。

```
$ awk 'BEGIN {printf "%f\n",999}
999.000000
```

2. 格式化输出

打印所有的学生名字和序列号，要求名字左对齐， 15个字符长度，后跟序列号。注意 \n 换行符放在最后一个指示符后面。输出将自动分成两列。

```
$ awk {printf "%-15s %s\n", $1,$3}' grade.txt
M.Tansley       48311
J.Lulu          48317
P.Bunny         48
J.Troll         4842
L.Tansley       4712
```

最好加入一些文本注释帮助理解报文含义。可在正文前嵌入头信息。注意这里使用 print 加入头信息。如果愿意，也可使用 printf。

```
$ awk 'BEGIN {print "Name \t\tS.Number"}{printf "%-15s %s\n", $1,$3}'
grade.txt
```

3. 向一行awk命令传值

在查看awk脚本前，先来查看怎样在awk命令行中传递变量。

在awk执行前将值传入awk变量，需要将变量放在命令行中，格式如下：

```
awk 命令变量=输入文件值
```

（后面会讲到怎样传递变量到awk脚本中）。

下面的例子在命令行中设置变量 AGE等于10，然后传入awk中，查询年龄在10岁以下的所有学生。

```
$ awk '{if ($5 < AGE) print $0}' AGE=10 grade.txt
M.Tansley   05/99     48311  Green    8     40    44
J.Lulu      06/99     48317  green    9     24    26
```

要快速查看文件系统空间容量，观察其是否达到一定水平，可使用下面 awk一行脚本。因为要监视的已使用空间容量不断在变化，可以在命令行指定一个触发值。首先用管道命令将 df -k传入awk，然后抽出第 4列，即剩余可利用空间容量。使用 $4~/^[0-9]/取得容量数值（1024块）而不是df的文件头，然后对命令行与‘ if($4<TRIGGER) ’上变量TRIGGER中指定的值进行查询测试。

```
$ df -k | awk '($4 ~/^[0-9]/) {if($4 < TRIGGER) print $6"\t"$4}'
TRIGGER=56000
/dos    55808
/apps   51022
```

在系统中使用df -k命令，产生下列信息：

	Filesystem	1024-blocks	Used	Free	%Used	Mounted on
Column	1	2	3	4	5	6

如果系统中df输出格式不同，必须相应改变列号以适应工作系统。

当然可以使用管道将值传入 awk。本例使用 who命令， who命令第一列包含注册用户名，这里打印注册用户，并加入一定信息。

```
$ who | awk '{print $1 " is logged on"}'
louisel is logged on
papam is logged on
```

awk也允许传入环境变量。下面的例子使用环境变量 LOGNAME支持当前用户名。可从who命令管道输出到awk中获得相应信息。

```
$ who | awk '{if ($1 == user) print $1" you are connected to
"$2}'user=$LOGNAME
```

如果root为当前登录用户，输出如下：

```
root you are connected to ttyp1
```

4. awk脚本文件

可以将awk脚本写入一个文件再执行它。命令不必很长（尽管这是写入一个脚本文件的主要原因），甚至可以接受一行命令。这样可以保存 awk命令，以使不必每次使用时都需要重新输入。使用文件的另一个好处是可以增加注释，以便于理解脚本的真正用途和功能。

使用前面的几个例子，将之转换成 awk可执行文件。像原来做的一样，将学生目前级别分相加awk ‘（tot+=$6） END{print "club student total points："tot}’ grade.txt。

创建新文件student_tot.awk，给所有awk程序加入awk扩展名是一种好习惯，这样通过查看文件名就知道这是一个awk程序。文本如下：

```
!/bin/awk -f
# all comment lines must start with a hash '#'
# name: student_tot.awk
# to call: student_tot.awk grade.txt
# prints total and average of club student points


#print a header first
BEGIN{
print "Student    Date  Member No.  Grade  Age   Points   Max"
print "Name       Joined                          Gained   Point Available"
print "=================================================================="


}
# let's add the scores of points gained
(tot+=$6)

# finished processing now let's print the total and average point
END{print "Club student total points  :" tot
  print "Average Club Student points:" tot/NR}
```

第一行是！/bin/awk -f。这很重要，没有它自包含脚本将不能执行。这一行告之脚本系统中awk的位置。通过将命令分开，脚本可读性提高，还可以在命令之间加入注释。这里加入头信息和结尾的平均值。基本上这是一个一行脚本文件。

执行时，在脚本文件后键入输入文件名，但是首先要对脚本文件加入可执行权限。

```
$ chmod u+x student_tot.awk
$ student_tot.awk grade.txt
Student    Date     Member No. Grade  Age  Points   Max
Name       Joined                         Gained   Point Available
==================================================================
M.Tansley   05/99     48311  Green    8     40     44
J.Lulu      06/99     48317  green    9     24     26
P.Bunny     02/99     48     Yellow   12    35     28
J.Troll     07/99     4842   Brown-3  12    26     26
L.Tansley   05/99     4712   Brown-2  12    30     28
Club student total points  :155
Average Club Student points:31
```

系统中运用的帐号核实程序检验数据操作人的数据输入，不幸的是这个程序有一点错误，或者应该说是“非文本特征”。如果一个记录被发现包含一个错误，它应该一次只打印一行

“ERROR*”，但实际上打印了许多这样的错误行。这会给帐号管理员造成误解，因此需要用awk脚本过滤出错误行的出现频率，使得每一个失败记录只对应一个错误行。

在awk实施过滤前先看看部分文件：

```
...
...
INVALID LCSD 98GJ23
ERROR*
ERROR*
CAUTION LPSS ERROR ON ACC NO.
ERROR*
ERROR*
ERROR*
ERROR*
ERROR*
PASS FIELD INVALID ON LDPS
ERROR*
ERROR*
PASS FIELD INVALID ON GHSI
ERROR*
CAUTION LPSS ERROR ON ACC NO.
ERROR*
ERROR*
```

awk脚本如下：

```
#!/bin/awk -f
# error_strip.awk
# to call: error_strip.awk <filename>
# strips out the ERROR* lines if there are more than one
# ERROR* lines after each failed record.

BEGIN { error_line="" }
# tell awk the whole is "ERROR*"
{ if ($0 == "ERROR*" && error_line == "ERROR*")

# go to next line
next;
  error_line = $0; print }
```

awk过滤结果如下：

```
$ strip.awk strip
INVALID LCSD 98GJ23
ERROR*
CAUTION LPSS ERROR ON ACC NO.
ERROR*
PASS FIELD INVALID ON LDPS
ERROR*
PASS FIELD INVALID ON GHSI
ERROR*
CAUTION LPSS ERROR ON ACC NO.
ERROR*
```

5. 在awk中使用FS变量

如果使用非空格符做域分隔符（FS）浏览文件，例如# 或 ：，编写这样的一行命令很容易，因为使用FS选项可以在命令行中指定域分隔符。

```
$ awk -F: 'awk {print $0}' input-file
```

使用awk脚本时，记住设置FS变量是在BEGIN部分。如果不这样做， awk将会发生混淆，不知道域分隔符是什么。

下述脚本指定FS变量。脚本从/etc/passwd文件中抽取第1和第5域，通过分号“;”分隔passwd文件域。第1域是帐号名，第5域是帐号所有者。

```
$ pg passwd.awk
!/bin/awk -f
# to call: passwd.awk /etc/passwd
# print out the first and fifth fields
BEGIN{
FS=":"}
{print $1,"\t",$5}


$ passwd.awk /etc/passwd
root      Special Admin login
xdm       Restart xdm Login
sysadm    Regular Admin login
daemon    Daemon Login for daemons needing permissions
```

6. 向awk脚本传值

向awk脚本传值与向awk一行命令传值方式大体相同，格式为：

```
awk script_file var=value input_file
```

下述脚本对比检查文件中域号和指定数字。这里使用了 NF变量MAX，表示指定检查的域号，使用双引号将域分隔符括起来，即使它是一个空格。

```
$ pg fieldcheck.awk
#!/bin/awk -f
# check on how many fields in a file
# name:fieldcheck.awk
# to call: fieldcheck MAX=n FS=<separator> filename
#
NF!=MAX{
print("line " NR " does not have" MAX " fields")}
```

如果以/etc/passwd作输入文件（passwd文件有7个域），运行上述脚本。参数格式如下：

```
$ fieldcheck.awk MAX=7 FS=":" /etc/passwd
```

使用前面一行脚本的例子，将之转换成awk脚本如下：

```
$ pg name.awk
#!/bin/awk -f
# name: age.awk
# to call: age.awk AGE=n grade.txt
# prints ages that are lower than the age supplied on the comand line
{if ($5 < AGE)
  print $0}
```

文本包括了比实际命令更多的信息，没关系，仔细研读文本后，就可以精确知道其功能及如何调用它。

不要忘了增加脚本的可执行权限，然后将变量和赋值放在命令行脚本名字后、输入文件前执行。

```
$ age.awk AGE=10 grade.txt
M.Tansley   05/99     48311  Green    8     40    44
J.Lulu      06/99     48317  green    9     24    26
```

同样可以使用前面提到的管道命令传值，下述 awk脚本从du命令获得输入，并输出块和字节数。

```
$ pg duawk.awk
#!/bin/awk -f
# to call: du | duawk.awk
# prints file/direc's in bytes and blocks
BEGIN{
OFS="\t" ;
print "name" "\t\t","bytes","blocks\n"
print "============================="}
{print $2,"\t\t",$1*512,$1}
```

为运行这段脚本，使用du命令，并管道输出至awk脚本。

```
$ du | awkdu.awk
name                    bytes    blocks
=========================================
./profile.d             2048     4
./X11                   135680   265
./rc.d/init.d           27136    53
./rc.d/rc0.d            512      1
./rc.d/rc1.d            512      1
```

9.2.13 awk数组

前面讲述split函数时，提到怎样使用它将元素划分进一个数组。这里还有一个例子：

```
$ awk 'BEGIN {print split("123#456#678", myarray, "#")}'
3
```

在上面的例子中，split返回数组myarray下标数。实际上myarray数组为：

Myarray[1]＝"123"
Myarray[2]＝"456"
Myarray[3]＝"678"

数组使用前，不必定义，也不必指定数组元素个数。经常使用循环来访问数组。下面是一种循环类型的基本结构：

```
For (element in array ) print array[element]
```

对于记录“ 123#456#678 ”，先使用split函数划分它，再使用循环打印各数组元素。操作脚本如下：

```
$ pg arraytest.awk
#!/bin/awk -f
# name: arraytest.awk
# prints out an array
BEGIN{
record="123#456#789";
split(record, myarray,"#") }
  END { for (i in myarray) {print myarray[i]}}
```

要运行脚本，使用/dev/null作为输入文件。

```
$ arraytest.awk  /dev/null
123
456
789
```

数组和记录

上面的例子讲述怎样通过split函数使用数组。也可以预先定义数组，并使用它与域进行比较测试，下面的例子中将使用更多的数组。

下面是从空手道数据库卸载的一部分数据，包含了学生级别及是否是成人或未成年人的信息，有两个域，分隔符为（#），文件如下：

```
$ pg grade_student.txt
Yellow#Junior
Orange#Senior
Yellow#Junior
Purple#Junior
Brown-2#Junior
White#Senior
Orange#Senior
Red#Junior
Brown-2#Senior
Yellow#Senior
Red#Junior
Blue#Senior
Green#Senior
Purple#Junior
White#Junior
```

脚本功能是读文件并输出下列信息。

1) 俱乐部中Yellow、Orange和Red级别的人各是多少。

2) 俱乐部中有多少成年人和未成年人。

查看文件，也许20秒内就会猜出答案，但是如果记录超过 60个又怎么办呢？这不会很容易就看出来，必须使用awk脚本。

首先看看awk脚本，然后做进一步讲解。

```
$ pg belts.awk
!/bin/awk -f
# name: belts.awk
# to call: belts.awk grade2.txt
# loops through the grade2.txt file and counts how many
# belts we have in (yellow, orange, red)
# also count how many adults and juniors we have
#
# start of BEGIN
# set FS and load the arrays with our values
BEGIN{FS="#"
# load the belt colours we are interested in only
belt["Yellow"]
belt["Orange"]
belt["Red"]
# end of BEGIN
# load the student type
student["Junior"]
student["Senior"]
}
# loop thru array that holds the belt colours against field-1
# if we have a match, keep a running total
  {for (colour in belt)
    {if ($1==colour)
    belt[colour]++}}
# loop thru array that holds the student type against
```

```
# field-2 if we have a match, keep a running total
  {for (senior_or_junior in student)
    {if ($2==senior_or_junior)
    student[senior_or_junior]++}}

# finished processing so print out the matches..for each array
END{ for (colour in belt) print "The club has", belt[colour], colour,
"Belts"

for (senior_or_junior in student) print "The club
has",student[senior_or_junior]\
,senior_or_junior, "students"}
```

BEGIN部分设置FS为符号#，即域分隔符，因为要查找 Yellow、Orange和Red三个级别。然后在脚本中手工建立数组下标对学生做同样的操作。注意，脚本到此只有下标或元素，并没有给数组名本身加任何注释。初始化完成后， BEGIN部分结束。记住BEGIN部分并没有文件处理操作。

现在可以处理文件了。首先给数组命名为 color，使用循环语句测试域1级别列是否等于数组元素之一（Yellow、Orange或Red），如果匹配，依照匹配元素将运行总数保存进数组。

同样处理数组‘ Senior_or_junior ’，浏览域2时匹配操作满足，运行总数存入 junior或senior的匹配数组元素。

END部分打印浏览结果，对每一个数组使用循环语句并打印它。

注意在打印语句末尾有一个 \符号，用来通知awk（或相关脚本）命令持续到下一行，当输入一个很长的命令，并且想分行输入时可使用这种方法。运行脚本前记住要加入可执行权限。

```
$ belts.awk grade_student.txt
The club has 2 Red Belts
The club has 2 Orange Belts
The club has 3 Yellow Belts
The club has 7 Senior students
The club has 8 Junior students
```

9.3 小结

awk语言学起来可能有些复杂，但使用它来编写一行命令或小脚本并不太难。本章讲述了awk的最基本功能，相信 大家已经掌握了awk的基本用法。awk是shell编程的一个重要工具。在shell命令或编程中，虽然可以使用 awk强大的文本处理能力，但是并不要求你成为这方面的专家。