Awk学习笔记

整理：**Jims of** 肥肥世家

<jims.yang@gmail.com>

第一次发布时间:2004年8月6日

Table of Contents

1. awk简介
2. awk命令格式和选项
    2.1. awk的语法有两种形式
    2.2. 命令选项
3. 模式和操作
    3.1. 模式
    3.2. 操作
4. awk的环境变量
5. awk运算符
6. 记录和域
    6.1. 记录
    6.2. 域
    6.3. 域分隔符
7. gawk专用正则表达式元字符
8. POSIX字符集
9. 匹配操作符(~)
10. 比较表达式
11. 范围模板
12. 一个验证passwd文件有效性的例子
13. 几个实例
14. awk编程
    14.1. 变量
    14.2. BEGIN模块
    14.3. END模块
    14.4. 重定向和管道
    14.5. 条件语句
    14.6. 循环
    14.7. 数组
    14.8. awk的内建函数
15. How-to

1. awk简介

awk是一种编程语言，用于在linux/unix下对文本和数据进行处理。数据可以来自标准输入、一个或多个文件，或其它命令的输出。它支持用户自定义函数和动态正则表达式等先进功能，是linux/unix下的一个强大编程工具。它在命令行中使用，但更多是作为脚本来使用。awk的处理文本和数据的方式是这样的，它逐行扫描文件，从第一行到最后一行，寻找匹配的特定模式的行，并在这些行上进行你想要的操作。如果没有指定处理动作，则把匹配的行显示到标准输出(屏幕)，如果没有指定模式，则所有被操作所指定的行都被处理。awk分别代表其作者姓氏的第一个字母。因为它的作者是三个人，分别是Alfred Aho、Brian Kernighan、Peter Weinberger。gawk是awk的GNU版本，它提供了Bell实验室和GNU的一些扩展。下面介绍的awk是以GUN的gawk为例的，在linux系统中已把awk链接到gawk，所以下面全部以awk进行介绍。

2. awk命令格式和选项

2.1. awk的语法有两种形式

- awk [options] 'script' var=value file(s)
- awk [options] -f scriptfile var=value file(s)

2.2. 命令选项

-F fs or --field-separator fs

指定输入文件折分隔符，fs是一个字符串或者是一个正则表达式，如-F:。

-v var=value or --asign var=value

赋值一个用户定义变量。

-f scripfile or --file scriptfile

从脚本文件中读取awk命令。

-mf nnn and -mr nnn

对nnn值设置内在限制，-mf选项限制分配给nnn的最大块数目；-mr选项限制记录的最大数目。这两个功能是Bell实验室版awk的扩展功能，在标准awk中不适用。

-W compact or --compat, -W traditional or --traditional

在兼容模式下运行awk。所以gawk的行为和标准的awk完全一样，所有的awk扩展都被忽略。

-W copyleft or --copyleft, -W copyright or --copyright

打印简短的版权信息。

-W help or --help, -W usage or --usage

打印全部awk选项和每个选项的简短说明。

-W lint or --lint

打印不能向传统unix平台移植的结构的警告。

-W lint-old or --lint-old

打印关于不能向传统unix平台移植的结构的警告。

-W posix

打开兼容模式。但有以下限制，不识别：\x、函数关键字、func、换码序列以及当fs是一个空格时，将新行作为一个域分隔符；操作符**和**=不能代替^和^=；fflush无效。

-W re-interval or --re-inerval

允许间隔正则表达式的使用，参考(grep中的Posix字符类)，如括号表达式[[:alpha:]]。

-W source program-text or --source program-text

使用program-text作为源代码，可与-f命令混用。

-W version or --version

打印bug报告信息的版本。

3. 模式和操作

awk脚本是由模式和操作组成的：

pattern {action} 如$ awk '/root/' test，或$ awk '$3 < 100' test。

两者是可选的，如果没有模式，则action应用到全部记录，如果没有action，则输出匹配全部记录。默认情况下，每一个输入行都是一条记录，但用户可通过RS变量指定不同的分隔符进行分隔。

3.1. 模式

模式可以是以下任意一个：

- /正则表达式/：使用通配符的扩展集。
- 关系表达式：可以用下面运算符表中的关系运算符进行操作，可以是字符串或数字的比较，如$2>%1选择第二个字段比第一个字段长的行。
- 模式匹配表达式：用运算符~(匹配)和~!(不匹配)。
- 模式，模式：指定一个行的范围。该语法不能包括BEGIN和END模式。

- BEGIN：让用户指定在第一条输入记录被处理之前所发生的动作，通常可在这里设置全局变量。
- END：让用户在最后一条输入记录被读取之后发生的动作。

3.2. 操作

操作由一人或多个命令、函数、表达式组成，之间由换行符或分号隔开，并位于大括号内。主要有四部份：

- 变量或数组赋值
- 输出命令
- 内置函数
- 控制流命令

4. awk的环境变量

Table 1. awk的环境变量

变量	描述
$n	当前记录的第n个字段，字段间由FS分隔。
$0	完整的输入记录。
ARGC	命令行参数的数目。
ARGIND	命令行中当前文件的位置(从0开始算)。
ARGV	包含命令行参数的数组。
CONVFMT	数字转换格式(默认值为%.6g)
ENVIRON	环境变量关联数组。
ERRNO	最后一个系统错误的描述。
FIELDWIDTHS	字段宽度列表(用空格键分隔)。
FILENAME	当前文件名。
FNR	同NR，但相对于当前文件。
FS	字段分隔符(默认是任何空格)。
IGNORECASE	如果为真，则进行忽略大小写的匹配。
NF	当前记录中的字段数。
NR	当前记录数。
OFMT	数字的输出格式(默认值是%.6g)。
OFS	输出字段分隔符(默认值是一个空格)。
ORS	输出记录分隔符(默认值是一个换行符)。
RLENGTH	由match函数所匹配的字符串的长度。
RS	记录分隔符(默认是一个换行符)。
RSTART	由match函数所匹配的字符串的第一个位置。
SUBSEP	数组下标分隔符(默认值是\034)。

5. awk运算符

Table 2. 运算符

运算符	描述
= += -= *= /= %= ^= **=	赋值
?:	C条件表达式
||	逻辑或
&&	逻辑与
~ ~!	匹配正则表达式和不匹配正则表达式
< <= > >= != ==	关系运算符
空格	连接

运算符	描述
+ -	加，减
* / &	乘，除与求余
+ - !	一元加，减和逻辑非
^ ***	求幂
++ --	增加或减少，作为前缀或后缀
$	字段引用
in	数组成员

6. 记录和域

6.1. 记录

awk把每一个以换行符结束的行称为一个记录。

记录分隔符：默认的输入和输出的分隔符都是回车，保存在内建变量ORS和RS中。

$0变量：它指的是整条记录。如$ awk '{print $0}' test将输出test文件中的所有记录。

变量NR：一个计数器，每处理完一条记录，NR的值就增加1。如$ awk '{print NR,$0}' test将输出test文件中所有记录，并在记录前显示记录号。

6.2. 域

记录中每个单词称做“域”，默认情况下以空格或tab分隔。awk可跟踪域的个数，并在内建变量NF中保存该值。如$ awk '{print $1,$3}' test将打印test文件中第一和第三个以空格分开的列(域)。

6.3. 域分隔符

内建变量FS保存输入域分隔符的值，默认是空格或tab。我们可以通过-F命令行选项修改FS的值。如$ awk -F: '{print $1,$5}' test将打印以冒号为分隔符的第一，第五列的内容。

可以同时使用多个域分隔符，这时应该把分隔符写成放到方括号中，如$awk -F'[:\t]' '{print $1,$3}' test，表示以空格、冒号和tab作为分隔符。

输出域的分隔符默认是一个空格，保存在OFS中。如$ awk -F: '{print $1,$5}' test，$1和$5间的逗号就是OFS的值。

7. gawk专用正则表达式元字符

一般通用的元字符集就不讲了，可参考我的Sed和Grep学习笔记。以下几个是gawk专用的，不适合unix版本的awk。

\Y

匹配一个单词开头或者末尾的空字符串。

\B

匹配单词内的空字符串。

\<

匹配一个单词的开头的空字符串，锚定开始。

\>

匹配一个单词的末尾的空字符串，锚定末尾。

\w

匹配一个字母数字组成的单词。

\W

匹配一个非字母数字组成的单词。

\‘

匹配字符串开头的一个空字符串。

\'

匹配字符串末尾的一个空字符串。

8. POSIX字符集

可参考我的Grep学习笔记

9. 匹配操作符(~)

用来在记录或者域内匹配正则表达式。如$ awk '$1 ~/^root/' test将显示test文件第一列中以root开头的行。

10. 比较表达式

conditional expression1 ? expression2: expression3，例如：$ awk '{max = {$1 > $3} ? $1: $3: print max}' test。如果第一个域大于第三个域，$1就赋值给max，否则$3就赋值给max。

$ awk '$1 + $2 < 100' test。如果第一和第二个域相加大于100，则打印这些行。

$ awk '$1 > 5 && $2 < 10' test,如果第一个域大于5，并且第二个域小于10，则打印这些行。

11. 范围模板

范围模板匹配从第一个模板的第一次出现到第二个模板的第一次出现之间所有行。如果有一个模板没出现，则匹配到开头或末尾。如$ awk '/root/,/mysql/' test将显示root第一次出现到mysql第一次出现之间的所有行。

12. 一个验证passwd文件有效性的例子

```
1$ cat /etc/passwd | awk -F: '\
2NF != 7{\
3printf("line %d,does not have 7 fields:%s\n",NR,$0)}\
4$1 !~ /[A-Za-z0-9]/{printf("line %d,non alpha and numeric user id:%d: %s\n,NR,$0)}\
5$2 == "*" {printf("line %d, no password: %s\n",NR,$0)}'
```

1 cat把结果输出给awk，awk把域之间的分隔符设为冒号。
2 如果域的数量(NF)不等于7，就执行下面的程序。
3 printf打印字符串"line ?? does not have 7 fields"，并显示该条记录。
4 如果第一个域没有包含任何字母和数字，printf打印“no alpha and numeric user id" ，并显示记录数和记录。
5 如果第二个域是一个星号，就打印字符串“no passwd”，紧跟着显示记录数和记录本身。

13. 几个实例

- $ awk '/^(no|so)/' test-----打印所有以模式no或so开头的行。
- $ awk '/^[ns]/{print $1}' test-----如果记录以n或s开头，就打印这个记录。
- $ awk '$1 ~/[0-9][0-9]$/(print $1}' test-----如果第一个域以两个数字结束就打印这个记录。
- $ awk '$1 == 100 || $2 < 50' test-----如果第一个或等于100或者第二个域小于50，则打印该行。
- $ awk '$1 != 10' test-----如果第一个域不等于10就打印该行。
- $ awk '/test/{print $1 + 10}' test-----如果记录包含正则表达式test，则第一个域加10并打印出来。
- $ awk '{print ($1 > 5 ? "ok "$1: "error"$1)}' test-----如果第一个域大于5则打印问号后面的表达式值，否则打印冒号后面的表达式值。
- $ awk '/^root/,/^mysql/' test----打印以正则表达式root开头的记录到以正则表达式mysql开头的记录范围内的所有记录。如果找到一个新的正则表达式root开头的记录，则继续打印直到下一个以正则表达式mysql开头的记录为止，或到文件末尾。

14. awk编程

14.1. 变量

- 在awk中，变量不需要定义就可以直接使用，变量类型可以是数字或字符串。
- 赋值格式：Variable = expression，如$ awk '$1 ~/test/{count = $2 + $3; print count}' test,上式的作用是,awk先扫描第一个域，一旦test匹配，就把第二个域的值加上第三个域的值，并把结果赋值给变量count，最后打印出来。
- awk可以在命令行中给变量赋值，然后将这个变量传输给awk脚本。如$ awk -F: -f awkscript month=4 year=2004 test，上式的month和year都是自定义变量，分别被赋值为4和2004。在awk脚本中，这些变量使用起来就象是在脚本中建立的一样。注意，如果参数前面出现test，那么在BEGIN语句中的变量就不能被使用。
- 域变量也可被赋值和修改，如$ awk '{$2 = 100 + $1; print }' test,上式表示，如果第二个域不存在，awk将计算表达式100加$1的值，并将其赋值给$2，如果第二个域存在，则用表达式的值覆盖$2原来的值。再例如：$ awk '$1 == "root"{$1 ="test";print}' test，如果第一个域的值是“root”，则把它赋值为“test”，注意，字符串一定要用双引号。
- 内建变量的使用。变量列表在前面已列出，现在举个例子说明一下。$ awk -F: '{IGNORECASE=1; $1 == "MARY"{print NR,$1,$2,$NF}'test，把IGNORECASE设为1代表忽略大小写，打印第一个域是mary的记录数、第一个域、第二个域和最后一个域。

14.2. BEGIN模块

BEGIN模块后紧跟着动作块，这个动作块在awk处理任何输入文件之前执行。所以它可以在没有任何输入的情况下进行测试。它通常用来改变内建变量的值，如OFS,RS和FS等，以及打印标题。如：$ awk 'BEGIN{FS=":"; OFS="\t"; ORS="\n\n"}{print $1,$2,$3} test。上式表示，在处理输入文件以前，域分隔符(FS)被设为冒号，输出文件分隔符(OFS)被设置为制表符，输出记录分隔符(ORS)被设置为两个换行符。$ awk 'BEGIN{print "TITLE TEST"}只打印标题。

14.3. END模块

END不匹配任何的输入文件，但是执行动作块中的所有动作，它在整个输入文件处理完成后被执行。如$ awk 'END{print "The number of records is" NR}' test，上式将打印所有被处理的记录数。

14.4. 重定向和管道

- awk可使用shell的重定向符进行重定向输出，如：$ awk '$1 = 100 {print $1 > "output_file" }' test。上式表示如果第一个域的值等于100，则把它输出到output_file中。也可以用>>来重定向输出，但不清空文件，只做追加操作。
- 输出重定向需用到getline函数。getline从标准输入、管道或者当前正在处理的文件之外的其他输入文件获得输入。它负责从输入获得下一行的内容，并给NF,NR和FNR等内建变量赋值。如果得到一条记录，getline函数返回1，如果到达文件的末尾就返回0，如果出现错误，例如打开文件失败，就返回-1。如：

  $ awk 'BEGIN{ "date" | getline d; print d}' test。执行linux的date命令，并通过管道输出给getline，然后再把输出赋值给自定义变量d，并打印它。

  $ awk 'BEGIN{"date" | getline d; split(d,mon); print mon[2]}' test。执行shell的date命令，并通过管道输出给getline，然后getline从管道中读取并将输入赋值给d，split函数把变量d转化成数组mon，然后打印数组mon的第二个元素。

  $ awk 'BEGIN{while("ls" | getline) print}'，命令ls的输出传递给geline作为输入，循环使getline从ls的输出中读取一行，并把它打印到屏幕。这里没有输入文件，因为BEGIN块在打开输入文件前执行，所以可以忽略输入文件。

  $ awk 'BEGIN{printf "What is your name?"; getline name < "/dev/tty" } $1 ~name {print "Found" name on line ", NR "."} END{print "See you," name "."} test。在屏幕上打印”What is your name?",并等待用户应答。当一行输入完毕后，getline函数从终端接收该行输入，并把它储存在自定义变量name中。如果第一个域匹配变量name的值，print函数就被执行，END块打印See you和name的值。

  $ awk 'BEGIN{while (getline < "/etc/passwd" > 0) lc++; print lc}'。awk将逐行读取文件/etc/passwd的内容，在到达文件末尾前，计数器lc一直增加，当到末尾时，打印lc的值。注意，如果文件不存在，getline返回-1，如果到达文件的末尾就返回0，如果读到一行，就返回1，所以命令 while (getline < "/etc/passwd")在文件不存在的情况下将陷入无限循环，因为返回-1表示逻辑真。
- 可以在awk中打开一个管道，且同一时刻只能有一个管道存在。通过close()可关闭管道。如：$ awk '{print $1, $2 | "sort" }' test END {close("sort")}。awd把print语句的输出通过管道作为linux命令sort的输入,END块执行关闭管道操作。
- system函数可以在awk中执行linux的命令。如：$ awk 'BEGIN{system("clear")}'。
- fflush函数用以刷新输出缓冲区，如果没有参数，就刷新标准输出的缓冲区，如果以空字符串为参数，如fflush("")，则刷新所有文件和管道的输出缓冲区。

14.5. 条件语句

awk中的条件语句是从C语言中借鉴过来的，可控制程序的流程。

14.5.1. if语句

格式：

```
        {if (expression){
                    statement; statement; ...
                      }
        }
```

$ awk '{if ($1 <$2) print $2 "too high"}' test。如果第一个域小于第二个域则打印。

$ awk '{if ($1 < $2) {count++; print "ok"}}' test.如果第一个域小于第二个域，则count加一，并打印ok。

14.5.2. if/else语句，用于双重判断。

格式：

```
        {if (expression){
                    statement; statement; ...
                        }
        else{
                    statement; statement; ...
                        }
        }
```

$ awk '{if ($1 > 100) print $1 "bad" ; else print "ok"}' test。如果$1大于100则打印$1 bad,否则打印ok。

$ awk '{if ($1 > 100){ count++; print $1} else {count--; print $2}' test。如果$1大于100，则count加一，并打印$1，否则count减一，并打印$1。

14.5.3. if/else else if语句，用于多重判断。

格式：

```
        {if (expression){
                     statement; statement; ...
                    }
        else if (expression){
                     statement; statement; ...
                    }
        else if (expression){
                     statement; statement; ...
                    }
        else {
                    statement; statement; ...
             }
        }
```

14.6. 循环

- awk有三种循环:while循环；for循环；special for循环。
- $ awk '{ i = 1; while (i <= NF) { print NF,$i; i++}}' test。变量的初始值为1，若i小于可等于NF(记录中域的个数),则执行打印语句，且i增加1。直到i的值大于NF.
- $ awk '{for (i = 1; i<NF; i++) print NF,$i}' test。作用同上。
- breadkcontinue语句。break用于在满足条件的情况下跳出循环；continue用于在满足条件的情况下忽略后面的语句，直接返回循环的顶端。如：

  ```
  {for ( x=3; x<=NF; x++)
              if ($x<0){print "Bottomed out!"; break}}
  {for ( x=3; x<=NF; x++)
              if ($x==0){print "Get next item"; continue}}
  ```

- next语句从输入文件中读取一行，然后从头开始执行awk脚本。如：

  ```
  {if ($1 ~/test/){next}
      else {print}
  }
  ```

- exit语句用于结束awk程序，但不会略过END块。退出状态为0代表成功，非零值表示出错。

14.7. 数组

awk中的数组的下标可以是数字和字母，称为关联数组。

14.7.1. 下标与关联数组

- 用变量作为数组下标。如：$ awk {name[x++]=$2};END{for(i=0;i<NR;i++) print i,name[i]}' test。数组name中的下标是一个自定义变量x，awk初始化x的值为0，在每次使用后增加1。第二个域的值被赋给name数组的各个元素。在END模块中，for循环被用于循环整个数组，从下标为0的元素开始，打印那些存储在数组中的值。因为下标是关健字，所以它不一定从0开始，可以从任何值开始。
- special for循环用于读取关联数组中的元素。格式如下：

  ```
  {for (item in arrayname){
          print arrayname[item]
          }
  }
  ```

  $ awk '/^tom/{name[NR]=$1}; END{for(i in name){print name[i]}}' test。打印有值的数组元素。打印的顺序是随机的。
- 用字符串作为下标。如：count["test"]
- 用域值作为数组的下标。一种新的for循环方式，for (index_value in array) statement。如:$ awk '{count[$1]++} END{for(name in count) print name,count[name]}' test。该语句将打印$1中字符串出现的次数。它首先以第一个域作数组count的下标，第一个域变化，索引就变化。
- delete函数用于删除数组元素。如：$ awk '{line[x++]=$1} END{for(x in line) delete(line[x])}' test。分配给数组line的是第一个域的值，所有记录处理完成后，special for循环将删除每一个元素。

14.8. awk的内建函数

14.8.1. 字符串函数

- sub函数匹配记录中最大、最靠左边的子字符串的正则表达式，并用替换字符串替换这些字符串。如果没有指定目标字符串就默认使用整个记录。替换只发生在第一次匹配的时候。格式如下：

  ```
  sub (regular expression, substitution string):
  sub (regular expression, substitution string, target string)
  ```

  实例：

  ```
  $ awk '{ sub(/test/, "mytest"); print }' testfile
  $ awk '{ sub(/test/, "mytest"); $1}; print }' testfile
  ```

  第一个例子在整个记录中匹配，替换只发生在第一次匹配发生的时候。如要在整个文件中进行匹配需要用到gsub

  第二个例子在整个记录的第一个域中进行匹配，替换只发生在第一次匹配发生的时候。
- gsub函数作用如sub，但它在整个文档中进行匹配。格式如下：

  ```
  gsub (regular expression, substitution string)
  gsub (regular expression, substitution string, target string)
  ```

  实例：

  ```
  $ awk '{ gsub(/test/, "mytest"); print }' testfile
  $ awk '{ gsub(/test/, "mytest"), $1 }; print }' testfile
  ```

  第一个例子在整个文档中匹配test，匹配的都被替换成mytest。

  第二个例子在整个文档的第一个域中匹配，所有匹配的都被替换成mytest。
- index函数返回子字符串第一次被匹配的位置，偏移量从位置1开始。格式如下：

  ```
  index(string, substring)
  ```

  实例：

  ```
  $ awk '{ print index("test", "mytest") }' testfile
  ```

  实例返回test在mytest的位置，结果应该是3。

- length函数返回记录的字符数。格式如下：

```
length( string )
length
```

实例：

```
$ awk '{ print length( "test" ) }'
$ awk '{ print length }' testfile
```

第一个实例返回test字符串的长度。

第二个实例返回testfile文件中第条记录的字符数。

- substr函数返回从位置1开始的子字符串，如果指定长度超过实际长度，就返回整个字符串。格式如下：

```
substr( string, starting position )
substr( string, starting position, length of string )
```

实例：

```
$ awk '{ print substr( "hello world", 7,11 ) }'
```

上例截取了world子字符串。

- match函数返回在字符串中正则表达式位置的索引，如果找不到指定的正则表达式则返回0。match函数会设置内建变量RSTART为字符串中子字符串的开始位置，RLENGTH为到子字符串末尾的字符个数。substr可利于这些变量来截取字符串。函数格式如下：

```
match( string, regular expression )
```

实例：

```
$ awk '{start=match("this is a test",/[a-z]+$/); print start}'
$ awk '{start=match("this is a test",/[a-z]+$/); print start, RSTART, RLENGTH }'
```

第一个实例打印以连续小写字符结尾的开始位置，这里是11。

第二个实例还打印RSTART和RLENGTH变量，这里是11(start)，11(RSTART)，4(RLENGTH)。

- toupper和tolower函数可用于字符串大小间的转换，该功能只在gawk中有效。格式如下：

```
toupper( string )
tolower( string )
```

实例：

```
$ awk '{ print toupper("test"), tolower("TEST") }'
```

- split函数可按给定的分隔符把字符串分割为一个数组。如果分隔符没提供，则按当前FS值进行分割。格式如下：

```
split( string, array, field separator )
split( string, array )
```

实例：

```
$ awk '{ split( "20:18:00", time, ":" ); print time[2] }'
```

上例把时间按冒号分割到time数组内，并显示第二个数组元素18。

14.8.2. 时间函数

- systime函数返回从1970年1月1日开始到当前时间(不计闰年)的整秒数。格式如下：

```
systime()
```

实例：

```
$ awk '{ now = systime(); print now }'
```

- strftime函数使用C库中的strftime函数格式化时间。格式如下：

```
systime( [format specification][,timestamp] )
```

Table 3. 日期和时间格式说明符

格式	描述
%a	星期几的缩写(Sun)
%A	星期几的完整写法(Sunday)
%b	月名的缩写(Oct)
%B	月名的完整写法(October)
%c	本地日期和时间
%d	十进制日期
%D	日期 08/20/99
%e	日期，如果只有一位会补上一个空格
%H	用十进制表示24小时格式的小时
%I	用十进制表示12小时格式的小时
%j	从1月1日起一年中的第几天
%m	十进制表示的月份
%M	十进制表示的分钟
%p	12小时表示法(AM/PM)
%S	十进制表示的秒
%U	十进制表示的一年中的第几个星期(星期天作为一个星期的开始)
%w	十进制表示的星期几(星期天是0)
%W	十进制表示的一年中的第几个星期(星期一作为一个星期的开始)
%x	重新设置本地日期(08/20/99)
%X	重新设置本地时间(12：00：00)
%y	两位数字表示的年(99)
%Y	当前月份
%Z	时区(PDT)
%%	百分号(%)

实例：

```
$ awk '{ now=strftime( "%D", systime() ); print now }'
$ awk '{ now=strftime("%m/%d/%y"); print now }'
```

14.8.3. 内建数学函数

Table 4.

函数名称	返回值
atan2(x,y)	y,x范围内的余切
cos(x)	余弦函数
exp(x)	求幂
int(x)	取整
log(x)	自然对数
rand()	随机数
sin(x)	正弦
sqrt(x)	平方根
srand(x)	x是rand()函数的种子
int(x)	取整，过程没有舍入
rand()	产生一个大于等于0而小于1的随机数

14.8.4. 自定义函数

在awk中还可自定义函数，格式如下：

```
function name ( parameter, parameter, parameter, ... ) {
```

```
                    statements
                    return expression                # the return statement and expression are optional
    }
```

15. How-to

- 如何把一行竖排的数据转换成横排？

  awk '{printf("%s,",$1)}' filename