一. AWK 说明

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：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。

二. awk命令格式和选项

2.1 awk的语法有两种形式

1. 命令行方式

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

其中，commands是真正awk命令，[-F域分隔符]是可选的。input-file(s)是待处理的文件。

在awk中，文件的每一行中，由域分隔符分开的每一项称为一个域。通常，在不指名-F域分隔符的情况下，默认的域分隔符是空格。

2. 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-file(s)

其中，-f选项加载awk-script-file中的awk脚本，input-file(s)跟上面的是一样的。

2.2 命令选项

（1）-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:。

（2）-v var=value or --asign var=value ：赋值一个用户定义变量。

（3）-f scripfile or --file scriptfile ：从脚本文件中读取awk命令。

（4）-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。

（5）-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。

（6）-W copyleft or --copyleft, -W copyright or --copyright ：打印简短的版权信息。

（7）-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。

（8）-W lint or --lint ：打印不能向传统unix平台移植的结构的警告。

（9）-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。

（10）-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。

（11）-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:]]。

（12）-W source program-text or --source program-text ：使用program-text作为源代码，可与-f命令混用。

（13）-W version or --version ：打印bug报告信息的版本。

三. 使用方法

```
#awk '{pattern + action}' {filenames}
```

尽管操作可能会很复杂，但语法总是这样，其中 pattern 表示 AWK 在数据中查找的内容，而 action 是在找到匹配内容时所执行的一系列命令。花括号（{}）不需要在程序中始终出现，但它们用于根据特定的模式对一系列指令进行分组。 pattern就是要表示的正则表达式，用斜杠括起来。

awk语言的最基本功能是在文件或者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。

通常，awk是以文件的一行为处理单位的。awk每接收文件的一行，然后执行相应的命令，来处理文本。

四. 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。

4.1. 模式

模式可以是以下任意一个：

（1）正则表达式：使用通配符的扩展集。

（2）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（3）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。

（4）模式匹配表达式：用运算符~(匹配)和~!(不匹配)。

（5）模式，模式：指定一个行的范围。该语法不能包括BEGIN和END模式。

（6）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。

（7）END：让用户在最后一条输入记录被读取之后发生的动作。

4.2. 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：

（1）变量或数组赋值

（2）输出命令

（3）内置函数

（4）控制流命令

五. 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)。

六. awk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
+ -	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

七. 记录和域

7.1. 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。

7.2. 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)。

7.3. 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。

八. 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。

九. 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。

十. 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。

十一. 示例

1. 入门实例

1.1 显示最近登录的5个帐号:

```
#last -n 5 | awk '{print $1}'
root
root
root
dmtsai
root
```

1.2 如果只是显示/etc/passwd的账户:

```
#cat /etc/passwd |awk -F ':' '{print $1}'
```

```
root
daemon
bin
sys
```

1.3 如果只是显示/etc/passwd的账户和账户对应的shell,而账户与shell之间以tab键分割:

```
#cat /etc/passwd |awk -F ':' '{print $1"\t"$7}'
root /bin/bash
daemon /bin/sh
bin /bin/sh
sys /bin/sh
```

1.4 如果只是显示/etc/passwd的账户和账户对应的shell,而账户与shell之间以逗号分割,而且在所有行添加列名name,shell,在最后一行添加"blue,/bin/nosh":

```
#cat /etc/passwd |awk -F ':' 'BEGIN {print "name,shell"}
{print $1","$7} END {print "blue,/bin/nosh"}' name,shell
root,/bin/bash
daemon,/bin/sh
bin,/bin/sh
sys,/bin/sh
....
blue,/bin/nosh
```

1.5 搜索/etc/passwd有root关键字的所有行:

```
#awk -F: '/root/' /etc/passwd
root:x:0:0:root:/root:/bin/bash
```

这种是pattern的使用示例，匹配了pattern(这里是root)的行才会执行action(没有指定action，默认输出每行的内容)。

搜索支持正则，例如找root开头的: awk -F: '/^root/' /etc/passwd

1.6 搜索/etc/passwd有root关键字的所有行，并显示对应的shell

```
# awk -F: '/root/{print $7}' /etc/passwd
/bin/bash
```

1.7 其他小示例:

$ 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。

2. awk内置变量示例

统计/etc/passwd:文件名，每行的行号，每行的列数，对应的完整行内容:

```
#awk -F ':' '{print "filename:" FILENAME ",linenumber:" NR ",columns:" NF ",linecontent:"$0}' /etc/passwd
filename:/etc/passwd,linenumber:1,columns:7,linecontent:root:x:0:0:root:/root:/bin/bash
filename:/etc/passwd,linenumber:2,columns:7,linecontent:daemon:x:1:1:daemon:/usr/sbin:/bin/sh
filename:/etc/passwd,linenumber:3,columns:7,linecontent:bin:x:2:2:bin:/bin:/bin/sh
filename:/etc/passwd,linenumber:4,columns:7,linecontent:sys:x:3:3:sys:/dev:/bin/sh
```

使用printf替代print,可以让代码更加简洁，易读

```
 #awk -F ':' '{printf("filename:%10s,linenumber:%s,columns:%s,linecontent:%s\n",FILENAME,NR,NF,$0)}' /etc/passwd
```

awk中同时提供了print和printf两种打印输出的函数。

其中print函数的参数可以是变量、数值或者字符串。字符串必须用双引号引用，参数用逗号分隔。如果没有逗号，参数就串联在一起而无法区分。这里，逗号的作用与输出文件的分隔符的作用是一样的，只是后者是空格而已。

printf函数，其用法和c语言中printf基本相似,可以格式化字符串,输出复杂时，printf更加好用，代码更易懂。

3. awk自定义变量

3.1. 下面统计/etc/passwd的账户人数:

```
#awk '{count++;print $0;} END{print "user count is ", count}' /etc/passwd root:x:0:0:root:/root:/bin/bash
......
user count is  40
```

count是自定义变量。之前的action{}里都是只有一个print,其实print只是一个语句，而action{}可以有多个语句，以;号隔开。

3.2. 这里没有初始化count，虽然默认是0，但是妥当的做法还是初始化为0:

```
#awk 'BEGIN {count=0;print "[start]user count is ", count} {count=count+1;print $0;} END{print "[end]user count is ", count}' /etc/passwd
[start]user count is 0 root:x:0:0:root:/root:/bin/bash
...
[end]user count is 40
```

3.3. 统计某个文件夹下的文件占用的字节数:

```
#ls -l |awk 'BEGIN {size=0;} {size=size+$5;} END{print "[end]size is ", size}'
[end]size is  8657198
```

3.4 如果以M为单位显示:

```
#ls -l |awk 'BEGIN {size=0;} {size=size+$5;} END{print "[end]size is ", size/1024/1024,"M"}' [end]size is
 8.25889 M
```

注意，统计不包括文件夹的子目录。

4. 条件语句

统计某个文件夹下的文件占用的字节数,过滤4096大小的文件(一般都是文件夹):

```
#ls -l |awk 'BEGIN {size=0;print "[start]size is ", size} {if($5!=4096){size=size+$5;}} END{print "[end]size is ", size/1024/1024,"M"}'
[end]size is  8.22339 M
```

5. 循环语句

显示/etc/passwd的账户:

```
#awk -F ':' 'BEGIN {count=0;} {name[count] = $1;count++;};
END{for (i = 0; i < NR; i++) print i, name[i]}' /etc/passwd

0 root

1 daemon

2 bin

3 sys

4 sync

5 games
......
```