链滴

Shell awk 命令详解（格式 + 使用方法）

作者：Leif160519
原文链接：https://ld246.com/article/1587373553488
来源网站：链滴

awk 命令的基本格式如下：

awk '条件1 {动作 1} 条件 2 {动作 2} ...' 文件名

1.条件（Pattern）：

一般使用关系表达式作为条件。这些关系表达式非常多，具体参考表1。

表 1 awk支持的主要条件类型

条件类型	条 件	说 明
awk保留字	BEGIN	在 awk 程序开始，尚未读取任何数据之前执行。BEGIN 后的动作只在程序开始时执行一次
awk保留字	END	在 awk 程序处完所有数据，即将结束时执行?END 后的动作只在程序结束时执行一次
关系运算符	>	大于
关系运算符	<	小于
关系运算符	>=	大于等于
关系运算符	<=	小于等于
关系运算符	==	等于。用于判断个值是否相等。如果是给变童赋值，则使用"="”
关系运算符	!=	不等于
关系运算符	A~B	判断字符串 A 是否包含能匹配 B 表达式的子字符串
关系运算符	A!~B	判断字符串 A 是否不包含能匹配 B 表达式的子字符串

正则表达式 /正则/ 如果在“//”
可以写入字符，则也可以支持正则表达式

例如：

x>10：判断变量 x 是否大于10；

x == y：判断变量 x 是否等于变量 y；

A~B：判断字符串 A 中是否包含能匹配 B 表达式的子字符串；

A!~B：判断字符串 A 中是否不包含能匹配 B 表达式的子字符串；

2.动作（Action）：

- 格式化输出；
- 流程控制语句；

我们先来学习 awk 的基本用法，也就是只看看格式化输出动作是干什么的。看看这个例子：

```
awk '{printf $2 "\t" $6 "\n"}' student.txt
#输出第二列和第六列的内容
Name    Average
Liming    87.66
Sc    85.66
Gao    91.66
```

在这个例子中没有设定任何的条件类型，所以这个文件中的所有内容都符合条件，动作会无条件执行动作是格式化输出 printf，"$2"和"$6"分别代表第二个字段和第六个字段，所以这条 awk 命令会列出tudent.txt 文件的第二个字段和第六个字段。

虽然都是截取列的命令，但是 awk 命令比 cut 命令智能多了，cut 命令是不能很好地识别空格作为分符的；而对于 awk 命令来说，只要分隔开，不管是空格还是制表符，都可以识别。比如刚刚截取 df 令的结果时，cut 命令已经力不从心了，我们来看看 awk 命令，命令如下：

```
df -h | awk '{print $1 "\t" $3}'
文件系统    已用
/dev/mapper/centos_192-root    12G
devtmpfs    0
tmpfs    0
tmpfs    269M
tmpfs    0
/dev/sda1    173M
/dev/mapper/centos_192-home    33M
tmpfs    0
overlay    12G
overlay    12G
overlay    12G
shm    0
shm    0
shm    0
overlay    12G
shm    0
```

在这两个例子中，我们分别使用了 printf 动作和 print 动作。发现了吗？如果使用 printf 动作，就必

在最后加入"\n"，因为 printf 只能识别标准输出格式；如果我们不使用"\n"，它就不会换行。而 print 动作则会在每次输出后自动换行，所以不用在最后加入"\n"。

3.awk的条件

我们来看看 awk 可以支持什么样的条件类型吧。awk 支持的主要条件类型如表 1 所示。

1) BEGIN

BEGIN 是 awk 的保留字，是一种特殊的条件类型。BEGIN 的执行时机是"在 awk 程序一开始，尚未取任何数据之前"。

一旦 BEGIN 后的动作执行一次，当 awk 开始从文件中读入数据时，BEGIN 的条件就不再成立，所以 EGIN 定义的动作只能被执行一次。例如：

```
awk 'BEGIN{printf "This is a transcript\n"} {printf $2 "\t" $6 "\n"}' student.txt
This is a transcript
Name    Average
Liming    87.66
Sc    85.66
Gao    91.66
```

解释：

- #awk命令只要检测不到完整的单引号就不会执行，所以这条命令的换行不用加入""

就是一行命令

- #这里定义了两个动作
- #第一个动作使用BEGIN条件，所以会在读入文件数据前打印"这是一张成绩单"（只会执行一次）
- #第二个动作会打印文件的第二个字段和第六个字段

2) END

END 也是 awk 的保留字，不过刚好和 BEGIN 相反。END 是在 awk 程序处理完所有数据，即将结时执行的。END 后的动作只在程序结束时执行一次。例如：

```
awk 'END{printf "The End \n"}{printf $2 "\t" $6 "\n"}' student.txt
Name    Average
Liming    87.66
Sc    85.66
Gao    91.66
The End
```

3)关系运算符

举几个例子看看关系运算符。假设我想看看平均成绩大于等于 87 分的学员是谁，就可以这样输入命：

【例 1】

```
cat student.txt | grep -v Name |awk '$4 >= 87 {printf $2 "\n"}'
Liming
```

Sc

解释：

- #使用cat输出文件内容，用grep取反包含"Name"的行
- #判断第四个字段（平均成绩）大于等于87分的行，如果判断式成立，则打印第六列（学员名）

在加入了条件之后，只有条件成立，动作才会执行；如果条件不满足，则动作不执行。通过这个实验大家可以发现，虽然 awk 是列提取命令，但是也要按行来读入。

这条命令的执行过程是这样的：

- 如果有 BEGIN 条件，则先执行 BEGIN 定义的动作。
- 如果没有 BEGIN 条件，则读入第一行，把第一行的数据依次赋予 $0、$1、$2 等变量。其中，$0 表此- 行的整体数据，$1 代表第一个字段，$2 代表第二个字段。
- 依据条件类型判断动作是否执行。如果条件符合，则执行动作；否则读入下一行激据。如果没有条，则每行都执行动作。
- 读入下一行数据，重复执行社步骤。

如果我想看看 Sc 用户的平均成绩呢？

【例 2】

```
awk '$2 -/Sc/ {printf $4 "\n"}' student.txt
#如果第二个字段中包含"Sc"字符，则打印第六个字段
96
```

这里要注意，在 awk 中，只有使用"//"包含的字符串，awk 命令才会查找。也就是说，字符串必须用"/"包含，awk 命令才能正确识别。

4) 正则表达式

如果想让 awk 识别字符串，则必须使用"//"包含，例如：

```
awk '/Liming/ {print}' student.txt
#打印Liming的成绩
1 Liming 82 95 86 87.66
```

当使用 df 命令查看分区的使用情况时，如果我只想查看真正的系统分区的使用情况，而不想查看光和临时分区的使用情况，则可以这样做：

```
df -h | awk '/sda[0-9]/ {printf $1 "\t" $5 "\n"}'
#查询包含"sda数字"的行，并打印第一个字段和第五个字段
/dev/sda1    18%
```

4.awk流程制

之所以称为 awk 编程，是因为在 awk 中允许定义变量，允许使用运算符，允许使用流程控制语句和义函数。这样就使得 awk 编程成了一门完整的程序语言，当然难度也比普通的命令要大得多。

所有语言的流程控制都非常类似，在这里只举一些例子，用来演示 awk 流程控制的作用。如果你现看不懂这些例子，则可以等学习完 Bash 流程控制之后，回过头来学习。

我们再利用 studert.txt 文件做一个练习，后面的使用比较复杂，我们再看看这个文件的内容，如下：

```
cat student.txt
ID Name PHP Linux MySQL Average
1 Liming 82 95 86 87.66
2 Sc 74 96 87 85.66
3 Gao 99 83 93 91.66
```

先来看看如何在 awk 中定义变量与调用变量的值。假设我想统计 PHP 成绩的总分，就应该这样做：

```
awk 'NR==2{php1 =$3} NR==3{php2=$3} NR==4{php3= $3;totle=php1+php2+php3;print "
otle php is" totle}' student.txt
totle php is255
```

这条命令有点复杂了，我们解释一下:

- "NR==2{php1=$3}"(条件是NR==2,动作是php=$3) 是指如果输入数据是第二行（第一行是标题），就把第二行的第三个字段的值赋予变量"php1"。
- "NR==3{php2=$3}"是指如果输入数据是第三行，就把第三行的第三个字段的值赋予变量"php2" NR==4{php3=$3;totle=php1+php2+php3;print"totle php is"totle}"("NR==4"是条件，后面{}的都是动作）是指如果输入数是第四行，就把第四行的第三个字段的值赋予变量"php3"；然后定义变 totle 的值是"php1+php2+php3";最后输出"totle php is"关键字，后面加变量 totle 的值。

在awk编程中，因为命令语句非常长，所以在输入格式时需要注意以下内容:

- 多个条件{动作}可以用空格分隔，也可以用回车分隔。
- 在一个动作中，如果需要执行多条命令，则需要用分隔，或用回车分隔。
- 在awk中，变量的赋值与调用都不需要加入"$"符号。
- 在条件中判断两个值是否相同，请使用"==",以便和变量赋值进行区分。

再看看如何实现流程控制。假设 Linux 成绩大于 90 分，就非常棒，命令如下：

```
awk '{if (NR>=2){if ($4>90) printf $2" is a good man!\n"}}' student.txt
#程序中有两个if判断，第一个判断行号大于2，第二个判断Linux成绩大于90分
Liming is a good man!
Sc is a good man!
```

其实在 awk 中，if 判断语句完全可以直接利用 awk 自带的条件来取代，刚刚的脚本可以改写成这样：

```
awk 'NR>=2 {test=$4} test>90 {printf $2" is a good man!\n"}' student.txt
Liming is a good man!
Sc is a good man!
```

解释:

- #先判断行号,如果大于2,就把第四个字段的值赋予变量test
- #再判断成绩，如果test的值大于90分，就打印好男人

5.区分

awk、sed、grep更适合的方向：

- grep 更适合单纯的查找或匹配文本
- sed 更适合编辑匹配到的文本
- awk 更适合格式化文本，对文本进行较复杂格式处理

参考：

- Shell awk命令详解（格式+使用方法）
- Linux awk 命令