# 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(stdin)、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有很多内建的功能，比如数组、函数等，这是它和C语言的相同之处，灵活性是awk最大的优势。

# 用法

## 语法

awk option 'pattern {action}' file

awk 'BEGIN{ commands } pattern{ commands } END{ commands }'

第一步：执行BEGIN{ commands }语句块中的语句；

第二步：从文件或标准输入(stdin)读取一行，然后执行pattern{ commands }语句块，它逐行扫描文件，从第一行到最后一行重复这个过程，直到文件全部被读取完毕。

第三步：当读至输入流末尾时，执行END{ commands }语句块。

BEGIN语句块在awk开始从输入流中读取行之前被执行，这是一个可选的语句块，比如变量初始化、打印输出表格的表头等语句通常可以写在BEGIN语句块中。

END语句块在awk从输入流中读取完所有的行之后即被执行，比如打印所有行的分析结果这类信息汇总都是在END语句块中完成，它也是一个可选语句块。

pattern语句块中的通用命令是最重要的部分，它也是可选的。如果没有提供pattern语句块，则默认执行{ print }，即打印每一个读取到的行，awk读取的每一行都会执行该语句块。

## 选项描述

-f program-file 从文件中读取 awk 程序源文件

-F fs 指定 fs 为输入字段分隔符

-v var=value 变量赋值

--posix 兼容 POSIX 正则表达式

--dump-variables=[file] 把 awk 命令时的全局变量写入文件，默认文件是 awkvars.out

--profile=[file] 格式化 awk 语句到文件，默认是 awkprof.out

## 常用模式

BEGIN{ } 给程序赋予初始状态，先执行的工作

END{ } 程序结束之后执行的一些扫尾工作

/regular expression/ 为每个输入记录匹配正则表达式

pattern && pattern 逻辑 and，满足两个模式

pattern || pattern 逻辑 or，满足其中一个模式

! pattern 逻辑 not，不满足模式

pattern1, pattern2 范围模式，匹配所有模式 1 的记录，直到匹配到模式 2

## 变量

FS 输入字段分隔符，默认是空格或制表符

OFS 输出字段分隔符，默认是空格

RS 输入记录分隔符，默认是换行符\n

ORS 输出记录分隔符，默认是换行符\n

NF 统计当前记录中字段个数

NR 统计记录编号，每处理一行记录，编号就会+1

FNR 统计记录编号，每处理一行记录，编号也会+1，与 NR 不同的是，处理文件时，编号会重新计数。

ARGC 命令行参数数量

ARGV 命令行参数数组序列数组，下标从 0 开始，ARGV[0]是 awk

ARGIND 当前正在处理的文件索引值。第一个文件是 1，第二个文件是 2，以此

ENVIRON 当前系统的环境变量

FILENAME 输出当前处理的文件名

IGNORECASE 忽略大小写

SUBSEP 数组中下标的分隔符，默认为"\034"

$0 表示整个当前行

$1 每行第一个字段

-F'[,;/]' 支持多个分隔符

## 运算符

(....) 分组

$ 字段引用

++ -- 递增和递减

+ - ! 加号，减号，和逻辑否定

\* / % 乘，除和取余

+ - 加法，减法

| |& 管道，用于 getline，print 和 printf

< > <= >= != == 关系运算符

~ !~ 正则表达式匹配，否定正则表达式匹配

in 数组成员

&& || 逻辑 and，逻辑 or

?: 简写条件表达式：

expr1 ? expr2 : expr3

第一个表达式为真，执行 expr2，否则执行 expr3

= += -= \*= /= %= ^= 变量赋值运算符

须知：

在 awk 中，有 3 种情况表达式为假：数字是 0，空字符串和未定义的值。

数值运算，未定义变量初始值为 0。字符运算，未定义变量初始值为空。

举例测试：

# awk 'BEGIN{n=0;if(n)print "true";else print "false"}'

false

# awk 'BEGIN{s="";if(s)print "true";else print "false"}'

false

# awk 'BEGIN{if(s)print "true";else print "false"}'

false

## 动作

print、流程控制、I/O 语句等。

# 样例

## 指定分隔符，打印指定字段

### 默认空格分隔

$ awk '{print $2}' test.txt

cat's

is

dog's

### 指定分隔符 -F

$ awk -F " " '{print $2}' test.txt

cat's

is

dog's

awk -F, '{print $1,$2}' awk.txt

## 变量使用

### 变量赋值

$ awk -v a=123 'BEGIN{print a}'

123

### 系统变量使用单引号

a=hello

awk 'BEGIN{print '$a'}'

### FS 和 OFS

在程序开始前重新赋值 FS 变量，改变默认分隔符为冒号，与-F 一样，OFS指定输出连接符

$ awk 'BEGIN{FS=",";OFS=":"}{print $1,$2}' awk.txt

a:b

1:2

b:c

### NF 字段个数

$ awk -F, '{print NF}' awk.txt

6

5

5

#### 打印最后一个字段

$ awk -F, '{print $NF}' awk.txt

c

5

q

#### 打印倒数第二个字段

$ awk -F, '{print $(NF-1)}' awk.txt

bb

4

w

### NR 和 FNR

NR 统计记录编号，每处理一行记录，编号就会+1，FNR 不同的是在统计第二个文件时会重新计数。

#### 打印行数

$ awk -F, '{print NR,$0}' awk.txt

1 a,b,c,a,bb,c

2 1,2,a,4,5

3 b,c,a,w,q

#### 打印总行数

$ awk -F, 'END{print NR}' awk.txt

3

也可以用

$ awk '{sum+=1}END{print sum}' awk.txt

3

#### 打印第三行

$ awk -F, 'NR==3{print NR,$0}' awk.txt

3 b,c,a,w,q

#### 打印1到3行

$ awk -F, 'NR>=1&&NR<=3{print NR,$0}' awk.txt

1 a,b,c,a,bb,c

2 1,2,a,4,5

3 b,c,a,w,q

#### NR和FNR区别

$ awk -F, '{print NR,FNR,$0}' awk.txt awk.txt

1 1 a,b,c,a,bb,c

2 2 1,2,a,4,5

3 3 b,c,a,w,q

4 1 a,b,c,a,bb,c

5 2 1,2,a,4,5

6 3 b,c,a,w,q

### ARGC 和 ARGV

ARGC 是命令行参数数量

ARGV 是将命令行参数存到数组，元素由 ARGC 指定，数组下标从 0 开始

$ awk 'BEGIN{print ARGC}' 1 2 3

4

$ awk 'BEGIN{print ARGV[2]}' 1 2

2

### ARGIND

ARGIND 是当前正在处理的文件索引值，第一个文件是 1，第二个文件是 2，以此类推，从而可以通过这种方式判断正在处理哪个文件。

$ awk '{print ARGIND,$0}' awk.txt test.txt

1 a,b,c,a,bb,c

1 1,2,a,4,5

1 b,c,a,w,q

2 my cat's name is betty

2 This is your dog

2 my dog's name is frank

2 This is your fish

2 my fish's name is george

2 This is your goat

2 my goat's name is adam

### ENVIRON

ENVIRON 调用系统变量

awk 'BEGIN{print ENVIRON["HOME"]}'

/root

### FILENAME

FILENAME 是当前处理文件的文件名。

$ awk '{print FILENAME"->"$0}' awk.txt test.txt

awk.txt->a,b,c,a,bb,c

awk.txt->1,2,a,4,5

awk.txt->b,c,a,w,q

test.txt->my cat's name is betty

test.txt->This is your dog

test.txt->my dog's name is frank

test.txt->This is your fish

test.txt->my fish's name is george

test.txt->This is your goat

test.txt->my goat's name is adam

### IGNORECASE=1 忽略大小写

$ awk 'BEGIN{IGNORECASE=1} /MY/ {print $0}' test.txt

my cat's name is betty

my dog's name is frank

my fish's name is george

my goat's name is adam

## 格式化输出

$ awk 'BEGIN{math=0;eng=0;com=0;printf "Lineno. Name No. Math English Computer Total\n";printf "------------------------------------------------------------\n"}{math+=$3; eng+=$4; com+=$5;printf "%-8s %-7s %-7s %-7s %-9s %-10s %-7s \n",NR,$1,$2,$3,$4,$5,$3+$4+$5} END{printf "------------------------------------------------------------\n";printf "%-24s %-7s %-9s %-20s \n","Total:",math,eng,com;printf "%-24s %-7s %-9s %-20s \n","Avg:",math/NR,eng/NR,com/NR}' score.txt

Lineno. Name No. Math English Computer Total

------------------------------------------------------------

1 Marry 2143 78 84 77 239

2 Jack 2321 66 78 45 189

3 Tom 2122 48 77 71 196

4 Mike 2537 87 97 95 279

5 Bob 2415 40 57 62 159

------------------------------------------------------------

Total: 319 393 350

Avg: 63.8 78.6 70

## /re/正则匹配

### 匹配特定行

$ awk '/my/{print $0}' test.txt

my cat's name is betty

my dog's name is frank

my fish's name is george

my goat's name is adam

### 逻辑 and、or 和 not

$ awk '/my/ && /cat/ {print $0}' test.txt

my cat's name is betty

$ awk '/my/ || /This/ {print $0}' test.txt

my cat's name is betty

This is your dog

my dog's name is frank

This is your fish

my fish's name is george

This is your goat

my goat's name is adam

$ awk '!/my/ {print $0}' test.txt

This is your dog

This is your fish

This is your goat

$ awk '/^[^my]/ {print $0}' test.txt

This is your dog

This is your fish

This is your goat

## 运算符

### 基础运算

+ - \* / %

$ echo 1|awk '{print $0+1}'

2

### 打印奇数行

$ echo 1|awk 'NF%2==1{print $0+1}'

2

### 三目运算符

三目运算作为一个表达式，里面不允许写 print

awk 'BEGIN{print 1==1?"yes":"no"}'

yes

## 数组

存储一系列相同类型的元素，键/值方式存储，通过下标（键）来访问值。

awk 中数组称为关联数组，不仅可以使用数字作为下标，还可以使用字符串作为下标。

数组元素的键和值存储在 awk 程序内部的一个表中，该表采用散列算法，因此数组元素是随机排序。

数组格式：array[index]=value

### 自定义数组

awk 'BEGIN{a[0]="test";print a[0]}'

test

$ awk 'BEGIN{a["b"]="test";print a["b"]}'

test

### 统计单词次数

$ awk -F, '{for (i=1;i<=NF;i++){a[$i]++}}END{for (v in a)print v,a[v]}' awk.txt

4 1

5 1

bb 1

w 1

a 4

b 2

c 3

q 1

1 1

2 1

## 流程控制

### BEGIN 和 END

BEGIN 模式是在处理文件之前执行该操作，常用于修改内置变量、变量赋值和打印输出的页眉或标题。END模式是在程序处理完才会执行。BEGIN和END可以单独使用或同时使用。

#### 输出指定行首

$ tail /etc/services |awk 'BEGIN{print "==="}{print $1}'

===

directplay8

directplay8

man

#### 指定行尾

$ tail /etc/services |awk '{print $0}END{print "===\nEND......"}'

directplay8 6073/tcp #DirectPlay8

directplay8 6073/udp #DirectPlay8

man 9535/tcp #Remote Man Server

rasadv 9753/tcp

rasadv 9753/udp

imip-channels 11320/tcp #IMIP Channels Port

imip-channels 11320/udp #IMIP Channels Port

directplaysrvr 47624/tcp #Direct Play Server

directplaysrvr 47624/udp #Direct Play Server

===

END......

### if 语句

#### 单分支

格式：if (condition) statement [ else statement ]

$ awk -F, '{if($1=="a") print $0;}' awk.txt

a,b,c,a,bb,c

#### 多分支

$ awk -F, '{if($1=="a") {print $0}else if($1 == "b") {print $0} else {print "not matched"}}' awk.txt

a,b,c,a,bb,c

not matched

b,c,a,w,q

#### 语句换行

awk -F, '{if($1=="a") {print $0}

else if($1 == "b") {print $0}

else {print "not matched"}}' awk.txt

a,b,c,a,bb,c

not matched

b,c,a,w,q

#### 变量控制

$ awk -F, 'BEGIN{a=0;b=0;}{if($1=="a") {a+=1;print $0}

else if($1 == "b") {b+=1;print $0}

else {b+=1;print "not matched"}}END{print a,b}' awk.txt

a,b,c,a,bb,c

not matched

b,c,a,w,q

1 2

### while 语句

格式：while (condition) statement

$ awk '{i=1;while(i<=NF){print $i;i++}}' awk.txt

a,b,c,a,bb,c

1,2,a,4,5

b,c,a,w,q

还支持do{}while()

### for 语句 C 语言风格

格式：for (expr1; expr2; expr3) statement

$ awk '{for(i=1;i<=NF;i++)print $i}' awk.txt

a,b,c,a,bb,c

1,2,a,4,5

b,c,a,w,q

#### 倒序打印文本

$ awk '{for(i=NF;i>=1;i--)print $i}' awk.txt

a,b,c,a,bb,c

1,2,a,4,5

b,c,a,w,q

### for 语句遍历数组

格式：for (var in array) statement

$ awk '{a[NR]=$0}END{for(v in a)print v,a[v]}' awk.txt

1 a,b,c,a,bb,c

2 1,2,a,4,5

3 b,c,a,w,q

### break 和 continue 语句

break 跳过所有循环，continue 跳过当前循环

$ awk 'BEGIN{for(i=1;i<=5;i++){if(i==3){break};print i}}'

1

2

### 删除数组和元素

delete array[index] 删除数组元素

delete array 删除数组

$ awk '{a[NR]=$0}END{delete a;for(v in a)print v,a[v]}' awk.txt

## 其他

### 统计/home目录下不同用户的普通文件的大小总size是多少？

ls -l|awk 'NR!=1 && !/^d/{sum[$3]+=$5} END{for (i in sum) printf "%-6s %-5s %-3s %-2s \n",i," ",sum[i]/1024/1024,"MB"}'

### 统计netstat -anp 状态为LISTEN和CONNECT的连接数量分别是多少

netstat -anp|awk '$6~/LISTEN|CONNECTED/{sum[$6]++} END{for (i in sum) printf "%-10s %-6s %-3s \n", i," ",sum[i]}'