



马哥教育

IT 人的高薪职业学院

文本处理三客之AWK



讲师：王晓春

本章内容



- ◆ awk介绍
- ◆ awk基本用法
- ◆ awk变量
- ◆ awk格式化
- ◆ awk操作符
- ◆ awk条件判断
- ◆ awk循环
- ◆ awk数组
- ◆ awk函数
- ◆ 调用系统命令

◆ awk : Aho, Weinberger, Kernighan , 报告生成器 , 格式化文本输出

◆ 有多种版本 : New awk (nawk) , GNU awk (gawk)

◆ gawk : 模式扫描和处理语言

◆ 基本用法 :

awk [options] 'program' var=value file...

awk [options] -f programfile var=value file...

awk [options] 'BEGIN{action;... }pattern{action;... }END{action;... }' file ...

awk 程序可由 : BEGIN语句块、能够使用模式匹配的通用语句块、END语句块 , 共3部分组成

program 通常是被放在单引号中

◆ 选项 :

-F “分隔符” 指明输入时用到的字段分隔符

-v var=value 变量赋值

- ◆ 基本格式：awk [options] 'program' file...
- ◆ Program：pattern{action statements;..}
- ◆ pattern和action
 - pattern部分决定动作语句何时触发及触发事件
BEGIN,END
 - action statements对数据进行处理，放在{}内指明
print, printf
- ◆ 分割符、域和记录
 - awk执行时，由分隔符分隔的字段（域）标记\$1,\$2...\$n称为域标识。\$0为所有域，注意：此时和shell中变量\$符含义不同
 - 文件的每一行称为记录
 - 省略action，则默认执行 print \$0 的操作

- ◆ 第一步：执行BEGIN{action;... }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{ action;... }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{action;...}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，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

◆ print格式：print item1, item2, ...

◆ 要点：

(1) 逗号分隔符

(2) 输出item可以字符串，也可是数值；当前记录的字段、变量或awk的表达式

(3) 如省略item，相当于print \$0

◆ 示例：

```
awk '{print "hello,awk"}'
```

```
awk -F: '{print}' /etc/passwd
```

```
awk -F: '{print "wang" }' /etc/passwd
```

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

```
awk -F: '{print $0}' /etc/passwd
```

```
awk -F: '{print $1" \t" $3}' /etc/passwd
```

```
grep "^UUID" /etc/fstab | awk '{print $2,$4}'
```

◆ 变量：内置和自定义变量

◆ FS：输入字段分隔符，默认为空白字符

```
awk -v FS=':' '{print $1,FS,$3}' /etc/passwd
```

```
awk -F: '{print $1,$3,$7}' /etc/passwd
```

◆ OFS：输出字段分隔符，默认为空白字符

```
awk -v FS=':' -v OFS=':' '{print $1,$3,$7}' /etc/passwd
```

◆ RS：输入记录分隔符，指定输入时的换行符

```
awk -v RS=' ' '{print }' /etc/passwd
```

◆ ORS：输出记录分隔符，输出时用指定符号代替换行符

```
awk -v RS=' ' -v ORS='###' '{print $0}' /etc/passwd
```

◆ NF：字段数量

```
awk -F: '{print NF}' /etc/fstab
```

 引用变量时，变量前不需加\$

```
awk -F: '{print $(NF-1)}' /etc/passwd
```

◆ NR：记录号

```
awk '{print NR}' /etc/fstab ; awk END '{print NR}' /etc/fstab
```

- ◆ FNR : 各文件分别计数,记录号

```
awk '{print FNR}' /etc/fstab /etc/inittab
```

- ◆ FILENAME : 当前文件名

```
awk '{print FILENAME}' /etc/fstab
```

- ◆ ARGV : 命令行参数的个数

```
awk '{print ARGV}' /etc/fstab /etc/inittab
```

```
awk 'BEGIN {print ARGV}' /etc/fstab /etc/inittab
```

- ◆ ARGV : 数组, 保存的是命令行所给定的各参数

```
awk 'BEGIN {print ARGV[0]}' /etc/fstab /etc/inittab
```

```
awk 'BEGIN {print ARGV[1]}' /etc/fstab /etc/inittab
```


◆ 自定义变量(区分字符大小写)

(1) -v var=value

(2) 在program中直接定义

◆ 示例：

```
awk -v test='hello gawk' '{print test}' /etc/fstab
```

```
awk -v test='hello gawk' 'BEGIN{print test}'
```

```
awk 'BEGIN{test="hello,gawk";print test}'
```

```
awk -F: '{sex= "male" ;print $1,sex,age;age=18}' /etc/passwd
```

```
cat awkscript
```

```
{print script,$1,$2}
```

```
awk -F: -f awkscript script="awk" /etc/passwd
```

printf命令



- ◆ 格式化输出：printf “FORMAT” , item1, item2, ...
 - (1) 必须指定FORMAT
 - (2) 不会自动换行，需要显式给出换行控制符，\n
 - (3) FORMAT中需要分别为后面每个item指定格式符
- ◆ 格式符：与item一一对应
 - %c：显示字符的ASCII码
 - %d, %i：显示十进制整数
 - %e, %E：显示科学计数法数值
 - %f：显示为浮点数
 - %g, %G：以科学计数法或浮点形式显示数值
 - %s：显示字符串
 - %u：无符号整数
 - %%：显示%自身
- ◆ 修饰符
 - #[.#] 第一个数字控制显示的宽度；第二个#表示小数点后精度，%3.1f
 - 左对齐（默认右对齐）%-15s
 - + 显示数值的正负符号 %+d

printf示例



```
awk -F: '{printf "%s", $1}' /etc/passwd
```

```
awk -F: '{printf "%s\n", $1}' /etc/passwd
```

```
awk -F: '{printf "%-20s %10d\n", $1, $3}' /etc/passwd
```

```
awk -F: '{printf "Username: %s\n", $1}' /etc/passwd
```

```
awk -F: '{printf "Username: %s, UID: %d\n", $1, $3}' /etc/passwd
```

```
awk -F: '{printf "Username: %15s, UID: %d\n", $1, $3}' /etc/passwd
```

```
awk -F: '{printf "Username: %-15s, UID: %d\n", $1, $3}' /etc/passwd
```

◆ 算术操作符：

$x+y$, $x-y$, $x*y$, x/y , x^y , $x\%y$

- x：转换为负数

+x：将字符串转换为数值

◆ 字符串操作符：没有符号的操作符，字符串连接

◆ 赋值操作符：

$=$, $+=$, $-=$, $*=$, $/=$, $\%=$, $\wedge=$, $++$, $--$

➤ 下面两语句有何不同

- `awk 'BEGIN{i=0;print ++i,i}'`
- `awk 'BEGIN{i=0;print i++,i}'`

◆ 比较操作符：

`==, !=, >, >=, <, <=`

◆ 模式匹配符：

`~`：左边是否和右边匹配，包含

`!~`：是否不匹配

➤ 示例：

```
awk -F: '$0 ~ /root/{print $1}' /etc/passwd
```

```
awk '$0~ "^root"' /etc/passwd
```

```
awk '$0 !~ /root/' /etc/passwd
```

```
awk -F: '$3==0' /etc/passwd
```

◆ 逻辑操作符：与&&，或||，非!

◆ 示例：

- `awk -F: '$3 >= 0 && $3 <= 1000 {print $1}' /etc/passwd`
- `awk -F: '$3 == 0 || $3 >= 1000 {print $1}' /etc/passwd`
- `awk -F: '!($3 == 0) {print $1}' /etc/passwd`
- `awk -F: '!($3 >= 500) {print $3}' /etc/passwd`

◆ 条件表达式（三目表达式）

`selector?if-true-expression:if-false-expression`

• 示例：

```
awk -F: '{ $3 >= 1000 ? usertype = "Common User" : usertype = "SysUser";  
printf "%15s: %-s\n", $1, usertype }' /etc/passwd
```

- ◆ PATTERN:根据pattern条件，过滤匹配的行，再做处理

(1)如果未指定：空模式，匹配每一行

(2) /regular expression/：仅处理能够模式匹配到的行，需要用/ /括起来

```
awk '/^UUID/{print $1}' /etc/fstab
```

```
awk '!/^UUID/{print $1}' /etc/fstab
```

(3) relational expression: 关系表达式，结果为“真”才会被处理

真：结果为非0值，非空字符串

假：结果为空字符串或0值

- ◆ 示例：

```
awk -F: 'i=1;j=1{print i,j}' /etc/passwd
```

```
awk '!0' /etc/passwd;    awk '!1' /etc/passwd
```

```
Awk -F: '$3>=1000{print $1,$3}' /etc/passwd
```

```
awk -F: '$3<1000{print $1,$3}' /etc/passwd
```

```
awk -F: '$NF==" /bin/bash"{print $1,$NF}' /etc/passwd
```

```
awk -F: '$NF ~ /bash$/{print $1,$NF}' /etc/passwd
```

◆ 4) line ranges : 行范围

startline,endline : /pat1/,/pat2/ 不支持直接给出数字格式

```
awk -F: '/^root\>/,/^nobody\>/{print $1}' /etc/passwd
```

```
awk -F: '(NR>=10&&NR<=20){print NR,$1}' /etc/passwd
```

◆ (5) BEGIN/END模式

BEGIN{} : 仅在开始处理文件中的文本之前执行一次

END{} : 仅在文本处理完成之后执行一次

示例



```
awk -F : 'BEGIN {print "USER USERID"} {print $1:""$3}
```

```
END{print "END FILE"}' /etc/passwd
```

```
awk -F: '{print "USER USERID";print $1:""$3} END{print "END FILE"}' /etc/passwd
```

```
awk -F: 'BEGIN{print "USER UID \n-----"}{print $1,$3}' /etc/passwd
```

```
awk -F: 'BEGIN{print "USER UID \n-----"}{print $1,$3}'
```

```
END{print "===== "}' /etc/passwd
```

```
seq 10 | awk 'i=0'
```

```
seq 10 | awk 'i=1'
```

```
seq 10 | awk 'i=!i'
```

```
seq 10 | awk '{i=!i;print i}'
```

```
seq 10 | awk '!(i=!i)'
```

```
seq 10 | awk -v i=1 'i=!i'
```

◆ 常用的action分类

- (1) Expressions : 算术 , 比较表达式等
- (2) Control statements : if, while等
- (3) Compound statements : 组合语句
- (4) input statements
- (5) output statements : print等

awk控制语句

- ◆ { statements;... } 组合语句
- ◆ if(condition) {statements;...}
- ◆ if(condition) {statements;...} else {statements;...}
- ◆ while(conditon) {statments;...}
- ◆ do {statements;...} while(condition)
- ◆ for(expr1;expr2;expr3) {statements;...}
- ◆ break
- ◆ continue
- ◆ delete array[index]
- ◆ delete array
- ◆ exit

awk控制语句if-else



- ◆ 语法：if(condition){statement;...}[else statement]
if(condition1){statement1}else if(condition2){statement2}else{statement3}
- ◆ 使用场景：对awk取得的整行或某个字段做条件判断
- ◆ 示例：
awk -F: '{if(\$3>=1000)print \$1,\$3}' /etc/passwd
awk -F: '{if(\$NF==" /bin/bash") print \$1}' /etc/passwd
awk '{if(NF>5) print \$0}' /etc/fstab
awk -F: '{if(\$3>=1000) {printf "Common user: %s\n",\$1} else {printf "root or Sysuser: %s\n",\$1}}' /etc/passwd
awk -F: '{if(\$3>=1000) printf "Common user: %s\n",\$1; else printf "root or Sysuser: %s\n",\$1}' /etc/passwd
df -h|awk -F% '/^\\dev/{print \$1}|awk '\$NF>=80{print \$1,\$5}'
awk 'BEGIN{ test=100;if(test>90){print "very good"}
else if(test>60){ print "good"}else{print "no pass"}}'

- ◆ while循环

- ◆ 语法：while(condition){statement;...}

- ◆ 条件“真”，进入循环；条件“假”，退出循环

- ◆ 使用场景：

 - 对一行内的多个字段逐一类似处理时使用

 - 对数组中的各元素逐一处理时使用

- ◆ 示例：

 - awk '/^[[:space:]]*linux16/{i=1;while(i<=NF)
 {print \$i,length(\$i); i++}}' /etc/grub2.cfg

 - awk '/^[[:space:]]*linux16/{i=1;while(i<=NF)
 {if(length(\$i)>=10){print \$i,length(\$i); i++}}' /etc/grub2.cfg

- ◆ do-while循环
- ◆ 语法：do {statement;...}while(condition)
- ◆ 意义：无论真假，至少执行一次循环体

◆ 示例：

```
awk 'BEGIN{ total=0;i=0;do{ total+=i;i++;}while(i<=100);print total}'
```

◆ for循环

◆ 语法：for(expr1;expr2;expr3) {statement;...}

◆ 常见用法：

```
for(variable assignment;condition;iteration process)
    {for-body}
```

◆ 特殊用法：能够遍历数组中的元素

语法：for(var in array) {for-body}

◆ 示例：

```
awk '/^[[:space:]]*linux16/{for(i=1;i<=NF;i++) {print $i,length($i)}}'
/etc/grub2.cfg
```

- ◆ `time (awk 'BEGIN{ total=0;for(i=0;i<=10000;i++){total+=i;};print total;})'`
- ◆ `time (total=0;for i in {1..10000};do total=$((total+i));done;echo $total)`
- ◆ `time (for ((i=0;i<=10000;i++));do let total+=i;done;echo $total)`
- ◆ `time (seq -s " +" 10000|bc)`

◆ switch语句

◆ 语法：switch(expression) {case VALUE1 or /REGEXP/: statement1; case VALUE2 or /REGEXP2/: statement2; ...; default: statementn}

◆ break和continue

```
awk 'BEGIN{sum=0;for(i=1;i<=100;i++)
    {if(i%2==0)continue;sum+=i}print sum}'
```

```
awk 'BEGIN{sum=0;for(i=1;i<=100;i++)
    {if(i==66)break;sum+=i}print sum}'
```

◆ break [n]

◆ continue [n]

◆ next:

提前结束对本行处理而直接进入下一行处理（awk自身循环）

示例：

```
awk -F: '{if($3%2!=0) next; print $1,$3}' /etc/passwd
```

- ◆ 关联数组：array[index-expression]
- ◆ index-expression:
 - (1) 可使用任意字符串；字符串要使用双引号括起来
 - (2) 如果某数组元素事先不存在，在引用时，awk会自动创建此元素，并将其值初始化为“空串”
 - (3) 若要判断数组中是否存在某元素，要使用“index in array”格式进行遍历
- ◆ 示例：

```
weekdays["mon"]="Monday"
awk 'BEGIN{weekdays["mon"]="Monday";weekdays["tue"]="Tuesday";
      print weekdays["mon"]}'
awk '!line[$0]++' dupfile
awk '{!line[$0]++;print $0, line[$0]}' dupfile
```

- ◆ 若要遍历数组中的每个元素，要使用for循环

- ◆ for(var in array) {for-body}

- ◆ 注意：var会遍历array的每个索引

- ◆ 示例：

```
awk 'BEGIN{weekdays["mon"]="Monday";weekdays["tue"]="Tuesday";
```

```
    for(i in weekdays) {print weekdays[i]}}'
```

```
netstat -tan | awk '/^tcp/{state[$NF]++}
```

```
    END{for(i in state) { print i,state[i]}}'
```

```
awk '{ip[$1]++}END{for(i in ip) {print i,ip[i]}}' /var/log/httpd/access_log
```

◆ 数值处理：

rand()：返回0和1之间一个随机数

```
awk 'BEGIN{srand(); for (i=1;i<=10;i++)print int(rand()*100) }'
```

◆ 字符串处理：

- length([s])：返回指定字符串的长度

- sub(r,s,[t])：对t字符串搜索r表示模式匹配的内容，并将第一个匹配内容替换为s

```
echo "2008:08:08 08:08:08" | awk 'sub(/:/,"-", $1)'
```

```
echo "2008:08:08 08:08:08" | awk '{sub(/:/,"-", $1);print $0}'
```

- gsub(r,s,[t])：对t字符串进行搜索r表示的模式匹配的内容，并全部替换为s所表示的内容

```
echo "2008:08:08 08:08:08" | awk 'gsub(/:/,"-", $0)'
```

```
echo "2008:08:08 08:08:08" | awk '{gsub(/:/,"-", $0);print $0}'
```

- split(s,array,[r])：以r为分隔符，切割字符串s，并将切割后的结果保存至array所表示的数组中，第一个索引值为1,第二个索引值为2,...

```
netstat -tn | awk '/^tcp\>/{split($5,ip,":");count[ip[1]]++}'
```

```
END{for (i in count) {print i,count[i]}}
```

◆ 自定义函数格式：

```
function name ( parameter, parameter, ... ) {  
    statements  
    return expression  
}
```

◆ 示例：

```
cat fun.awk  
function max(x,y) {  
    x>y?var=x:var=y  
    return var  
}  
BEGIN{a=3;b=2;print max(a,b)}  
awk -f fun.awk
```

◆ system命令

- ◆ 空格是awk中的字符串连接符，如果system中需要使用awk中的变量可以使用空格分隔，或者说除了awk的变量外其他一律用"引用起来

```
awk 'BEGIN{system("hostname")}'
```

```
awk 'BEGIN{score=100; system("echo your score is " score)}'
```

◆ 将awk程序写成脚本，直接调用或执行

◆ 示例：

```
cat f1.awk
```

```
{if($3>=1000)print $1,$3}
```

```
awk -F: -f f1.awk /etc/passwd
```

```
cat f2.awk
```

```
#!/bin/awk -f
```

```
#this is a awk script
```

```
{if($3>=1000)print $1,$3}
```

```
chmod +x f2.awk
```

```
f2.awk -F: /etc/passwd
```


向awk脚本传递参数

◆ 格式：

awkfile var=value var2=value2... Inputfile

◆ 注意：在BEGIN过程中不可用。直到首行输入完成以后，变量才可用。可以通过-v 参数，让awk在执行BEGIN之前得到变量的值。命令行中每一个指定的变量都需要一个-v参数

◆ 示例：

```
cat test.awk
```

```
#!/bin/awk -f
```

```
{if($3 >=min && $3<=max)print $1,$3}
```

```
chmod +x test.awk
```

```
test.awk -F: min=100 max=200 /etc/passwd
```

- ◆ 1、文件ip_list.txt如下格式，请提取“.magedu.com”前面的主机名部分并写入到回到该文件中
 - 1 blog.magedu.com
 - 2 www.magedu.com
 - ...
 - 999 study.magedu.com
- ◆ 2、统计/etc/fstab文件中每个文件系统类型出现的次数
- ◆ 3、统计/etc/fstab文件中每个单词出现的次数
- ◆ 4、提取出字符串Yd\$C@M05MB%9&Bdh7dq+YVixp3vpw中的所有数字
- ◆ 5、有一文件记录了1-100000之间随机的整数共5000个，存储的格式100,50,35,89...请取出其中最大和最小的整数
- ◆ 6、解决DOS攻击生产案例：根据web日志或者或者网络连接数，监控当某个IP并发连接数或者短时内PV达到100，即调用防火墙命令封掉对应的IP，监控频率每隔5分钟。防火墙命令为：iptables -A INPUT -s IP -j REJECT

- ◆ 7、将以下文件内容中FQDN取出并根据其进行计数从高到低排序

<http://mail.magedu.com/index.html>

<http://www.magedu.com/test.html>

<http://study.magedu.com/index.html>

<http://blog.magedu.com/index.html>

<http://www.magedu.com/images/logo.jpg>

<http://blog.magedu.com/20080102.html>

- ◆ 8、将以下文本以inode为标记，对inode相同的counts进行累加，并且统计出同一inode中，beginnumber的最小值和endnumber的最大值

inode|beginnumber|endnumber|counts|

106|3363120000|3363129999|10000|

106|3368560000|3368579999|20000|

310|3337000000|3337000100|101|

310|3342950000|3342959999|10000|

310|3362120960|3362120961|2|

311|3313460102|3313469999|9898|

311|3313470000|3313499999|30000|

311|3362120962|3362120963|2|

输出的结果格式为：

310|3337000000|3362120961|10103|

311|3313460102|3362120963|39900|

106|3363120000|3368579999|30000|

- ◆ 博客 : <http://mageedu.blog.51cto.com>
- ◆ 主页 : <http://www.magedu.com>
- ◆ QQ : 1661815153, 113228115
- ◆ QQ群 : 203585050, 279599283

祝大家学业有成

谢 谢

咨询热线 400-080-6560