AWK

1. 格式

1). 命令形式

`awk [-F|-f|-v] ‘BEGIN{} //{command1; command2} END{}’ file`

> BEGIN{ 这里面放的是执行前的语句 }，其中的语句只会运行一次
> END {这里面放的是处理完所有的行后要执行的语句 }。其中的语句也只会运行一次
> {这里面放的是处理每一行时要执行的语句}，这里的语句会读文件的每行都运行一次

[-F|-f|-v] 大参数，-F指定分隔符，-f调用脚本，-v定义变量 var=value

' ' 引用代码块

BEGIN 初始化代码块，在对每一行进行处理之前，初始化代码，主要是引用全局变量，设置FS分隔符

// 匹配代码块，可以是字符串或正则表达式

{} 命令代码块，包含一条或多条命令

; 多条命令使用分号分隔

END 结尾代码块，在对每一行进行处理之后再执行的代码块，主要是进行最终计算或输出结尾摘要信息

2). 特殊要点

linux awk 内置变量使用介绍

特殊符号	含义
$0	表示整个当前行
$1	每行第一个字段
NF	字段数量变量
NR	每行的记录号，多文件记录递增
FNR	与NR类似，不过多文件记录不递增，每个文件都从1开始
\t	制表符
\n	换行符
FS	BEGIN时定义分隔符，输入字段分隔符 默认是空格
RS	输入的记录分隔符， 默认为换行符(即文本是按一行一行输入)
~	匹配，与==相比不是精确比较
!~	不匹配，不精确比较
==	等于，必须全部相等，精确比较
!=	不等于，精确比较
&&	逻辑与
||	逻辑或
+	匹配时表示1个或1个以上
/[0-9][0-9]+/	两个或两个以上数字
/[0-9][0-9]*/	一个或一个以上数字
FILENAME	文件名
OFS	输出字段分隔符， 默认也是空格，可以改为制表符等
ORS	输出的记录分隔符，默认为换行符,即处理结果也是一行一行输出到屏幕
-F'[:#/]'	定义三个分隔符

2. 内置函数

linux awk 内置函数详细介绍

> 这篇博客的函数都写在begin{}中

1). 算术函数

函数名	说明
atan2(y, x)	返回 y/x 的反正切。
cos(x)	返回 x 的余弦；x 是弧度。
sin(x)	返回 x 的正弦；x 是弧度。
exp(x)	返回 x 幂函数。
log(x)	返回 x 的自然对数。
sqrt(x)	返回 x 平方根。
int(x)	返回 x 的截断至整数的值。
rand( )	返回任意数字 n，其中 0 <= n < 1。
srand([Expr])	将 rand 函数的种子值设置为 Expr 参数的值。返回先前的种子值。

```
# OFMT 设置输出数据格式是保留3位小数
awk 'BEGIN{OFMT="%.3f";fs=sin(1);fe=exp(10);fl=log(10);fi=int(3.1415);print
fs,fe,fl,fi;}'
# 0.841 22026.466 2.303 3

#获得随机数:
awk 'BEGIN{srand();fr=int(100*rand());print fr;}'
# 78
```

2). 字符串函数

函数	说明
gsub(Ere, Repl, [In])	除了正则表达式所有具体值被替代这点，它和 sub 函数完全一样地执行，。
sub(Ere, Repl, [In])	用 Repl 参数指定的字符串替换 In 参数指定的字符串中的由 Ere 参数指定的扩展正则表达式的第一个具体值。sub 函数返回替换的数量。出现在 Repl 参数指定的字符串中的 &（和符号）由 In 参数指定的与 Ere 参数的指定的扩展正则表达式匹配的字符串替换。如果未指定 In 参数，缺省值是整个记录（$0 记录变量）。
index(String1, String2)	在由 String1 参数指定的字符串（其中有出现 String2 指定的参数）中，返回位置，从 1 开始编号。如果 String2 参数不在 String1 参数中出现，则返回 0（零）。
length [(String)]	返回 String 参数指定的字符串的长度（字符形式）。如果未给出 String 参数，则返回整个记录的长度（$0 记录变量）。
blength [(String)]	返回 String 参数指定的字符串的长度（以字节为单位）。如果未给出 String 参数，则返回整个记录的长度（$0 记录变量）。
substr(String, M, [N])	返回具有 N 参数指定的字符数量子串。子串从 String 参数指定的字符串取得，其字符以 M 参数指定的位置开始。M 参数指定为将 String 参数中的第一个字符作为编号 1。如果未指定 N 参数，则子串的长度将是 M 参数指定的位置到 String 参数的末尾 的长度。
match(String, Ere)	在 String 参数指定的字符串（Ere 参数指定的扩展正则表达式出现在其中）中返回位置（字符形式），从 1 开始编号，或如果 Ere 参数不出现，则返回 0（零）。RSTART 特殊变量设置为返回值。RLENGTH 特殊变量设置为匹配的字符串的长度，或如果未找到任何匹配，则设置为 -1（负一）。
split(String, A, [Ere])	将 String 参数指定的参数分割为数组元素 A[1], A[2], . . ., A[n]，并返回 n 变量的值。此分隔可以通过 Ere 参数指定的扩展正则表达式进行，或用当前字段分隔符（FS 特殊变量）来进行（如果没有给出 Ere 参数）。除非上下文指明特定的元素还应具有一个数字值，否则 A 数组中的元素用字符串值来创建。
tolower(String)	返回 String 参数指定的字符串，字符串中每个大写字符将更改为小写。大写和小写的映射由当前语言环境的 LC_CTYPE 范畴定义。
toupper(String)	返回 String 参数指定的字符串，字符串中每个小写字符将更改为大写。大写和小写的映射由当前语言环境的 LC_CTYPE 范畴定义。
sprintf(Format, Expr, Expr, . . .)	根据 Format 参数指定的 printf 子例程格式字符串来格式化 Expr 参数指定的表达式并返回最后生成的字符串。

> Ere都可以是正则表达式

```
# 在 info中查找满足正则表达式，/[0-9]+/ 用""替换，并且替换后的值，赋值给info 未给info值，默认是$0
awk 'BEGIN{info="this is a test2010test!";gsub(/[0-9]+/,"!",info);print info}'
# this is a test!test!

# 查找字符串（index使用）
awk 'BEGIN{info="this is a test2010test!";print index(info,"test")?"ok":"no found";}'
```

```
# ok
# 未找到，返回0

# 正则表达式匹配查找(match使用）
awk 'BEGIN{info="this is a test2010test!";print match(info,/[0-9]+/)?"ok":"no
found";}'    # ok

# 截取字符串(substr使用）
awk 'BEGIN{info="this is a test2010test!";print substr(info,4,10);}'
# s is a tes
# 从第 4个 字符开始，截取10个长度字符串

# 字符串分割（split使用）
awk 'BEGIN{info="this is a test";split(info,tA," ");print length(tA);for(k in
tA){print k,tA[k];}}'
# 4
# 4 test
# 1 this
# 2 is
# 3 a
# 分割info,动态创建数组tA,这里比较有意思，awk for …in 循环，是一个无序的循环。 并不是从数组
下标1…n ，因此使用时候需要注意。
```

> awk for …in 循环，是一个无序的循环。 并不是从数组下标1…n ，因此使用时候需要注意
>
> 应该是因为awk中数组其实是字段。

格式化字符串输出（sprintf使用）

格式化字符串包括两部分内容: 一部分是正常字符, 这些字符将按原样输出; 另一部分是格式化规定字符, 以"%"开始, 后跟一个或几个规定字符,用来确定输出内容格式。

格式符	说明
%d	十进制有符号整数
%u	十进制无符号整数
%f	浮点数
%s	字符串
%c	单个字符
%p	指针的值
%e	指数形式的浮点数
%x	%X 无符号以十六进制表示的整数
%o	无符号以八进制表示的整数
%g	自动选择合适的表示法

```
awk 'BEGIN{n1=124.113;n2=-1.224;n3=1.2345;
printf("%.2f,%.2u,%.2g,%X,%o\n",n1,n2,n3,n1,n1);}'
# 124.11,18446744073709551615,1.2,7C,174
```

3). 时间函数

函数名	说明
mktime(YYYY MM DD HH MM SS[DST])	生成时间格式
strftime([format [, timestamp]])	格式化时间输出，将时间戳转为时间字符串 具体格式，见下表.
systime()	得到时间戳,返回从1970年1月1日开始到当前时间(不计闰年)的整秒数

```
# 创建指定时间(mktime使用）
awk 'BEGIN{tstamp=mktime("2001 01 01 12 12 12");print strftime("%c",tstamp);}'
# 2001年01月01日 星期一 12时12分12秒

# 求2个时间段中间时间差,介绍了strftime使用方法
awk 'BEGIN{tstamp1=mktime("2001 01 01 12 12 12");tstamp2=mktime("2001 02 01 0 0 0");print tstamp2-tstamp1;}'
# 2634468

awk 'BEGIN{tstamp1=mktime("2001 01 01 12 12 12");tstamp2=systime();print tstamp2-tstamp1;}'
# 308201392
```

strftime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(%)

4). 一般函数

函数	说明
close(Expression)	用同一个带字符串值的 Expression 参数来关闭由 print 或 printf 语句打开的或调用 getline 函数打开的文件或管道。如果文件或管道成功关闭，则返回 0；其它情况下返回非零值。如果打算写一个文件，并稍后在同一个程序中读取文件，则 close 语句是必需的。
system(Command)	执行 Command 参数指定的命令，并返回退出状态。等同于 system 子例程。
Expression | getline [Variable]	从来自 Expression 参数指定的命令的输出中通过管道传送的流中读取一个输入记录，并将该记录的值指定给 Variable 参数指定的变量。如果当前未打开将 Expression 参数的值作为其命令名称的流，则创建流。创建的流等同于调用 popen 子例程，此时 Command 参数取 Expression 参数的值且 Mode 参数设置为一个是 r 的值。只要流保留打开且 Expression 参数求得同一个字符串，则对 getline 函数的每次后续调用读取另一个记录。如果未指定 Variable 参数，则 $0 记录变量和 NF 特殊变量设置为从流读取的记录。
getline [Variable] < Expression	从 Expression 参数指定的文件读取输入的下一个记录，并将 Variable 参数指定的变量设置为该记录的值。只要流保留打开且 Expression 参数对同一个字符串求值，则对 getline 函数的每次后续调用读取另一个记录。如果未指定 Variable 参数，则 $0 记录变量和 NF 特殊变量设置为从流读取的记录。
getline [Variable]	将 Variable 参数指定的变量设置为从当前输入文件读取的下一个输入记录。如果未指定 Variable 参数，则 $0 记录变量设置为该记录的值，还将设置 NF、NR 和 FNR 特殊变量。

```
# 打开外部文件（close用法）
awk 'BEGIN{while("cat /etc/passwd"|getline){print $0;};close("/etc/passwd");}'
# root:x:0:0:root:/root:/bin/bash
# bin:x:1:1:bin:/bin:/sbin/nologin
# daemon:x:2:2:daemon:/sbin:/sbin/nologin

# 逐行读取外部文件(getline使用方法）
awk 'BEGIN{while(getline < "/etc/passwd"){print $0;};close("/etc/passwd");}'
# root:x:0:0:root:/root:/bin/bash
# bin:x:1:1:bin:/bin:/sbin/nologin
# daemon:x:2:2:daemon:/sbin:/sbin/nologin

awk 'BEGIN{print "Enter your name:";getline name;print name;}'
# Enter your name:
# wangdongdong
# wangdongdong

# 调用外部应用程序(system使用方法）
awk 'BEGIN{b=system("python helloworld.py");print b;}'
# helloworld
# 0
```

3. 数组

linux awk数组操作详细介绍

4. 案例

1). print

print 是awk打印指定内容的主要命令

```
awk '{print}' /etc/passwd == awk '{print $0}' /etc/passwd
awk '{print " "}' /etc/passwd #不输出passwd的内容，而是输出相同个数的空行，进一步解释了awk是一行一行处理文本
awk '{print "a"}' /etc/passwd #输出相同个数的a行，一行只有一个a字母
awk -F":" '{print $1}' /etc/passwd
awk -F: '{print $1; print $2}' /etc/passwd #将每一行的前二个字段，分行输出，进一步理解一行一行处理文本
awk -F: '{print $1,$3,$6}' OFS="\t" /etc/passwd #输出字段1,3,6，以制表符作为分隔符
```

2). -f指定脚本文件

```
awk -f script.awk  file

# script.awk内容如下：
BEGIN{
FS=":"
}
{print $1}   #效果与awk -F":" '{print $1}'相同,只是分隔符使用FS在代码自身中指定

awk 'BEGIN{X=0} /^$/{ X+=1 } END{print "I find",X,"blank lines."}' test
# I find 4 blank lines.

ls -l|awk 'BEGIN{sum=0} !/^d/{sum+=$5} END{print "total size is",sum}' #计算文件大小
#total size is 17487
```

3). -F指定分隔符

$1 指指定分隔符后，第一个字段，$3第三个字段， \t是制表符

一个或多个连续的空格或制表符看做一个定界符，即多个空格看做一个空格

```
awk -F":" '{print $1}'  /etc/passwd
awk -F":" '{print $1 $3}'  /etc/passwd #$1与$3相连输出，不分隔
awk -F":" '{print $1,$3}'  /etc/passwd #多了一个逗号，$1与$3使用空格分隔
awk -F":" '{print $1 " " $3}'  /etc/passwd #$1与$3之间手动添加空格分隔
awk -F":" '{print "Username:" $1 "\t\t Uid:" $3 }' /etc/passwd #自定义输出
awk -F: '{print NF}' /etc/passwd #显示每行有多少字段
awk -F: '{print $NF}' /etc/passwd #将每行第NF个字段的值打印出来
awk -F: 'NF==4 {print }' /etc/passwd #显示只有4个字段的行
```

```
awk -F: 'NF>2{print $0}' /etc/passwd #显示每行字段数量大于2的行
awk '{print NR,$0}' /etc/passwd #输出每行的行号
awk -F: '{print NR,NF,$NF,"\t",$0}' /etc/passwd #依次打印行号，字段数，最后字段值，制表符，每行内容
awk -F: 'NR==5{print}'  /etc/passwd #显示第5行
awk -F: 'NR==5 || NR==6{print}'  /etc/passwd #显示第5行和第6行
route -n|awk 'NR!=1{print}' #不显示第一行
```

4). IF语句

必须用在{}中，且比较内容用()扩起来

```
awk -F: '{if($1~/mail/) print $1}' /etc/passwd #简写
awk -F: '{if($1~/mail/) {print $1}}' /etc/passwd #全写
awk -F: '{if($1~/mail/) {print $1} else {print $2}}' /etc/passwd #if...else...

awk -F: '{if($3>100) print "large"; else print "small"}' /etc/passwd
awk -F: 'BEGIN{A=0;B=0} {if($3>100) {A++; print "large"} else {B++; print "small"}} END{print A,"\t",B}' /etc/passwd  #ID大于100,A加1，否则B加1
awk -F: '{if($3<100) next; else print}' /etc/passwd #小于100跳过，否则显示
awk -F: '{print ($3>100 ? "yes":"no")}'  /etc/passwd
awk -F: '{print ($3>100 ? $3":\tyes":$3":\tno")}'  /etc/passwd
```

5). 条件表达式

== != > >=

```
awk -F":" '$1=="mysql"{print $3}' /etc/passwd
awk -F":" '{if($1=="mysql") print $3}' /etc/passwd #与上面相同
awk -F":" '$1!="mysql"{print $3}' /etc/passwd #不等于
awk -F":" '$3>1000{print $3}' /etc/passwd #大于
awk -F":" '$3>=100{print $3}' /etc/passwd #大于等于
awk -F":" '$3<1{print $3}' /etc/passwd #小于
awk -F":" '$3<=1{print $3}' /etc/passwd #小于等于
```

6). 逻辑运算符

&& ||

```
awk -F: '$1~/mail/ && $3>8 {print }' /etc/passwd #逻辑与，$1匹配mail，并且$3>8
awk -F: '{if($1~/mail/ && $3>8) print }' /etc/passwd
awk -F: '$1~/mail/ || $3>1000 {print }' /etc/passwd #逻辑或
awk -F: '{if($1~/mail/ || $3>1000) print }' /etc/passwd
```

7). 数值运算

```
awk -F: '$3 > 100' /etc/passwd
awk -F: '$3 > 100 || $3 < 5' /etc/passwd
awk -F: '$3+$4 > 200' /etc/passwd
awk -F: '/mysql|mail/{print $3+10}' /etc/passwd #第三个字段加10打印
awk -F: '/mysql/{print $3-$4}' /etc/passwd #减法
awk -F: '/mysql/{print $3*$4}' /etc/passwd #求乘积
awk '/MemFree/{print $2/1024}' /proc/meminfo #除法
awk '/MemFree/{print int($2/1024)}' /proc/meminfo #取整
```

8). 输出分隔符OFS

```
awk '$6 ~ /FIN/ || NR==1 {print NR,$4,$5,$6}' OFS="\t" netstat.txt
awk '$6 ~ /WAIT/ || NR==1 {print NR,$4,$5,$6}' OFS="\t" netstat.txt
#输出字段6匹配WAIT的行，其中输出每行行号，字段4，5,6，并使用制表符分割字段
```

9). 输出处理结果到文件

①在命令代码块中直接输出 route -n|awk 'NR!=1{print > "./fs"}'

②使用重定向进行输出 route -n|awk 'NR!=1{print}' > ./fs

10). 格式化输出

```
netstat -anp|awk '{printf "%-8s %-8s %-10s\n",$1,$2,$3}'
```

printf表示格式输出

%格式化输出分隔符

-8长度为8个字符

s表示字符串类型

打印每行前三个字段，指定第一个字段输出字符串类型(长度为8)，第二个字段输出字符串类型(长度为8),

第三个字段输出字符串类型(长度为10)

```
netstat -anp|awk '$6=="LISTEN" || NR==1 {printf "%-10s %-10s %-10s
\n",$1,$2,$3}'
netstat -anp|awk '$6=="LISTEN" || NR==1 {printf "%-3s %-10s %-10s %-10s
\n",NR,$1,$2,$3}'
```

11). 数组

```
netstat -anp|awk 'NR!=1{a[$6]++} END{for (i in a) print i,"\t",a[i]}'
netstat -anp|awk 'NR!=1{a[$6]++} END{for (i in a) \
  printf "%-20s %-10s %-5s \n", i,"\t",a[i]}'
```

12). 其他应用

```
awk -F: '{print NF}' helloworld.sh #输出文件每行有多少字段
awk -F: '{print $1,$2,$3,$4,$5}' helloworld.sh #输出前5个字段
awk -F: '{print $1,$2,$3,$4,$5}' OFS='\t' helloworld.sh #输出前5个字段并使用制表符分隔输出
awk -F: '{print NR,$1,$2,$3,$4,$5}' OFS='\t' helloworld.sh #制表符分隔输出前5个字段，并打印行号
awk -F'[:#]' '{print NF}'  helloworld.sh #指定多个分隔符: #，输出每行多少字段
awk -F'[:#]' '{print $1,$2,$3,$4,$5,$6,$7}' OFS='\t' helloworld.sh #制表符分隔输出多字段
awk -F'[:#/]' '{print NF}' helloworld.sh #指定三个分隔符，并输出每行字段数
awk -F'[:#/]' '{print $1,$2,$3,$4,$5,$6,$7,$8,$9,$10,$11,$12}' hi.sh #制表符分隔输出多字段

#计算/home目录下，普通文件的大小，使用KB作为单位
ls -l|awk 'BEGIN{sum=0} !/^d/{sum+=$5} END{print "total size
is:",sum/1024,"KB"}'
ls -l|awk 'BEGIN{sum=0} !/^d/{sum+=$5} END{print "total size
is:",int(sum/1024),"KB"}' #int是取整的意思

#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]}'

#统计/home目录下不同用户的普通文件的总数是多少？
ls -l|awk 'NR!=1 && !/^d/{sum[$3]++} END{for (i in sum) printf "%-6s %-5s %-3s
\n",i," ",sum[i]}'

#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"}'

#输出成绩表
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}' test
```

13). Vlookup

awk实现excel vlookup

```
# 数据
$ cat a.txt
1       abc
2       def
3       ghi
4       jlm
$ cat b.txt
3       shit
1       rubb
# 方法
awk 'NR==FNR{a[$1]=$2;next}NR>FNR{if($1 in a)print $0"\t"a[$1]}' a.txt b.txt
# 3       shit    ghi
# 1       rubb    abc
```

> 注：a是一个字典，`NR==FNR`时在读一个文件，`NR>FNR`时在读b.txt
>
> `NR==FNR{a[$1]=$2;next}`是使用第一个文件`a.txt`初始化字典，将其每行`$1`作为key，`$2`作为value
>
> `if($1 in a)`指序号`$1`是否在a的key中，如果在打印的`a[$1]`为key：`$1`对于的value
>
> 查看字典：`awk 'NR==FNR{a[$1]=$2;next}END{for(k in a){print k,a[k]}}' a.txt b.txt`

14). group by

[总结一下awk的group by功能]

通过数组的key作为作为on的条件，value进行聚合运算

```
# 数据
cat c.txt
06 01 06 30      2.700         81.000
06 01 06 45      3.900        175.500
06 01 07 00      2.400          0.000
06 01 07 15      0.160          2.400
06 01 08 00      0.380          0.000
06 01 08 15      0.300          4.500
06 01 08 30      3.900        117.000
06 01 08 45      5.520        248.400
06 01 09 00      6.600          0.000
06 01 09 15      9.600        144.000
06 01 09 30      3.300         99.000
06 01 09 45      2.300        103.500
06 01 10 15      7.880        118.200
06 01 10 30     10.820        324.600
06 01 10 45      7.360        331.200
06 01 11 00     11.940          0.000
06 01 11 15      4.200         63.000
06 01 11 30      3.180         95.400
06 01 11 45      1.800         81.000
06 01 12 00     30.970          0.000
```

```
# 实现
# 单列group by单列聚合
awk '{a[$2]+=$5}END{for(i in a) printf "%s %10.3f\n",i,a[i]}' c.txt

# 单列group by多列聚合
awk '{a[$2]+=$5;b[$2]+=$6}END{for(i in a) printf "%s %10.3f
%14.3f\n",i,a[i],b[i]}' c.txt

# 多列group by单列聚合
awk '{a[$2" "$3]+=$5}END{for(i in a) printf "%s %10.3f\n",i,a[i]}' c.txt # 求和
awk '{a[$2" "$3]+=1}END{for(i in a) printf "%s %10.3f\n",i,a[i]}' c.txt # 计数
awk '{s[$2" "$3]+=$5;n[$2" "$3]+=1}END{for(i in s){avg_v=s[i]/n[i];printf "%s
%10.3f\n",i,avg_v}}' c.txt # 平均

# 多列group by多列聚合：多来一个数组
awk '{a[$2" "$3]+=$5;b[$2" "$3]+=$6}END{for(i in a) printf "%s %14.3f
%14.3f\n",i,a[i],b[i]}' c.txt
```

awk -F' '/from/{print}'

```
awk '{for(i=0;++i<=NF;)a[i]=a[i]?a[i] FS $i:$i}END{for(i=0;i++<NF;)printf
a[i]"\b \n"}'
```

15). 取最新分区

```
export HIVE_SKIP_SPARK_ASSEMBLY=true
partition_tbl="cf_model.bt_score_bt_merge_score"
partition_cmd="
  set hive.cli.print.header=flase;
  show partitions $partition_tbl;"
v_partition=`hive -S -e  "$partition_cmd" | sort | tail -n 1`
last_dt=${v_partition:3:10}
echo "last dt of $ascore_tbl is "$last_dt
```

16). 统计词频

```
#egrep -o "\b[[:alpha:]]+\b" a.log
egrep -o ",[[:alpha:]]+" test.data
awk '{++count[$0]} END{for (word in count){
printf("%-20s%d\n",word,count[word]);}}'
sort -n -r -k2,2
egrep -o ",[[:alpha:]]+" test.data|awk -F',' '{++count[$2]} END{for (word in
count){ print(word,count[word]);}}' OFS="\t"|sort -n -r -k2,2


egrep -o ",([[:alpha:]]|（|）)+" test.data|awk -F',' '{++count[$2]} END{for (word
in count){ print(word,count[word]);}}' OFS="\t"|sort -n -r -k2,2
```

5. 正则表达式

awk 正则表达式、正则运算符详细介绍

linux shell 正则表达式(BREs,EREs,PREs)差异比较

1). +

指定如果**一个或多个字符**或扩展正则表达式的具体值（在 +（加号）前）在这个字符串中，则字符串匹配。

命令行：`awk '/smith+ern/' testfile`

将包含字符 smit，后跟一个或多个 h 字符，并以字符 ern 结束的字符串的任何记录打印至标准输出。

此示例中的输出是：smithern, harry smithhern, anne

2). ?

指定如果**零个或一个字符**或扩展正则表达式的具体值（在？（问号）之前）在字符串中，则字符串匹配。

命令行：`awk '/smith?/' testfile`

将包含字符 `smit`，后跟零个或一个 `h` 字符的实例的所有记录打印至标准输出。

此示例中的输出是：smith, alan smithern, harry smithhern, anne smitters, alexis

3). |

指定如果以 |（垂直线）隔开的字符串的**任何一个在字符串中**，则字符串匹配。

命令行：`awk '/allen | alan /' testfile`

将包含字符串 allen 或 alan 的所有记录打印至标准输出。

此示例中的输出是：smiley, allen smith, alan

4). ()

在正则表达式中将字符串组合在一起。

命令行：`awk '/a(ll)?(nn)?e/' testfile`

将具有字符串 `ae` 或 `alle` 或 `anne` 或 `allnne` 的所有记录打印至标准输出。

此示例中的输出是：smiley, allen smithhern, anne

5). {m}

指定如果**正好有 m 个模式的具体值位于字符串中**，则字符串匹配。

命令行：awk '/l{2}/' testfile

打印至标准输出：smiley, allen

6). {m,}

指定如果**至少 m 个模式的具体值在字符串中**，则字符串匹配。

命令行：`awk '/t{2,}/' testfile`

打印至标准输出：smitters, alexis

7). {m, n}

指定如果 **m 和 n 之间（包含的 m 和 n）个模式**的具体值在字符串中（其中m<= n），则字符串匹配。

命令行：`awk '/er{1, 2}/' testfile`

打印至标准输出：smithern, harry smithern, anne smitters, alexis

8). [String]

指定正则表达式与方括号内 **String 变量指定的任何字符匹配。**

命令行：`awk '/sm[a-h]/' testfile`

将具有 `sm` 后跟以字母顺序从 `a` 到 `h` 排列的任何字符的所有记录打印至标准输出。

此示例的输出是：smawley, andy

9). [^ String]

在 [ ]（方括号）和在指定字符串开头的 ^ (插入记号) 指明**正则表达式与方括号内的任何字符不匹配。**

命令行：`awk '/sm\[^a-h]/' testfile`

打印至标准输出：smiley, allen smith, alan smithern, harry smithhern, anne smitters, alexis

10). ~, !~

表示**指定变量与正则表达式****匹配（~）或**不匹配**（!~）的条件语句。

命令行：`awk '$1 ~ /n/' testfile`

将第一个字段包含字符 `n` 的所有记录打印至标准输出。

此示例中的输出是：smithern, harry smithhern, anne

11). ^

指定字段或记录的**开头。**

命令行： awk '$2 ~ /^h/' testfile

将把字符 `h` 作为第二个字段的第一个字符的所有记录打印至标准输出。

此示例中的输出是：smithern, harry

12). $

指定字段或记录的**末尾**。

命令行： awk '$2 ~ /y$/' testfile

将把字符 `y` 作为第二个字段的最后一个字符的所有记录打印至标准输出。

此示例中的输出是：smawley, andy smithern, harry

13). .（句号）

表示除了在空白末尾的终端换行字符以外的**任何一个字符**。

命令行： awk '/a..e/' testfile

将具有以两个字符隔开的字符 `a` 和 e 的所有记录打印至标准输出。

此示例中的输出是：smawley, andy smiley, allen smithhern, anne

14). *（星号）

表示**零个或更多的任意字符**。

命令行： awk '/a.*e/' testfile

将具有以零个或更多字符隔开的字符 `a` 和 e 的所有记录打印至标准输出。

此示例中的输出是：smawley, andy smiley, allen smithhern, anne smitters, alexis

15). \ (反斜杠)

转义字符。当位于在扩展正则表达式中具有特殊含义的任何字符之前时，转义字符除去该字符的任何特殊含义。

例如，命令行： /a\/\//

将与模式 a // 匹配，因为反斜杠否定斜杠作为正则表达式定界符的通常含义。要将反斜杠本身指定为字符，则使用双反斜杠。有关反斜杠及其使用的更多信息，请参阅以下关于转义序列的内容。