AWK

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

awk [options] 'script' var=value file(s)

awk [options] -f scriptfile var=value file(s)

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. 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。

3.1. 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。

3.2. 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

四. 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)。

五. 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

运算符	描述
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

六. 记录和域

6.1. 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。

6.2. 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)。

6.3. 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。

七. gawk 专用正则表达式元字符

以下几个是 gawk 专用的，不适合 unix 版本的 awk。

（1）/Y：匹配一个单词开头或者末尾的空字符串。
（2）/B：匹配单词内的空字符串。
（3）/<：匹配一个单词的开头的空字符串，锚定开始。
（4）/>：匹配一个单词的末尾的空字符串，锚定末尾。
（5）/w：匹配一个字母数字组成的单词。
（6）/W：匹配一个非字母数字组成的单词。
（7）/‘：匹配字符串开头的一个空字符串。
（8）/'：匹配字符串末尾的一个空字符串。

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、awk '/101/' file 显示文件 file 中包含 101 的匹配行。
awk '/101/,/105/' file
awk '$1 == 5' file
awk '$1 == "CT"' file 注意必须带双引号
awk '$1 * $2 >100 ' file
awk '$2 >5 && $2<=15' file

2、awk '{print NR,NF,$1,$NF,}' file 显示文件 file 的当前记录号、域数和每一行的第一个和最后一个域。
awk '/101/ {print $1,$2 + 10}' file 显示文件 file 的匹配行的第一、二个域加 10。
awk '/101/ {print $1$2}' file
awk '/101/ {print $1 $2}' file 显示文件 file 的匹配行的第一、二个域，但显示时域中间没有分隔符。

3、df | awk '$4>1000000 ' 通过管道符获得输入，如：显示第 4 个域满足条件的行。

4、awk -F "|" '{print $1}' file 按照新的分隔符“|”进行操作。
awk 'BEGIN { FS="[: /t|]" }
{print $1,$2,$3}' file 通过设置输入分隔符（FS="[: /t|]"）修改输入分隔符。
Sep="|"
awk -F $Sep '{print $1}' file 按照环境变量 Sep 的值做为分隔符。
awk -F '[:/t|]' '{print $1}' file 按照正则表达式的值做为分隔符，这里代表空格、:、TAB、|同时做为分隔符。
awk -F '[][]' '{print $1}' file 按照正则表达式的值做为分隔符，这里代表[、]

5、awk -f awkfile file 通过文件 awkfile 的内容依次进行控制。
cat awkfile
/101/{print "/047 Hello! /047"} --遇到匹配行以后打印 ' Hello! './047 代表单引号。
{print $1,$2} --因为没有模式控制，打印每一行的前两个域。
6、awk '$1 ~ /101/ {print $1}' file 显示文件中第一个域匹配 101 的行（记录）。
7、awk 'BEGIN { OFS="%"}
{print $1,$2}' file 通过设置输出分隔符（OFS="%"）修改输出格式。
8、awk 'BEGIN { max=100 ;print "max=" max}
BEGIN 表示在处理任意行之前进行的操作。
{max=($1 >max ?$1:max); print $1,"Now max is "max}' file 取得文件第一个域的最大值。

9、awk '$1 * $2 >100 {print $1}' file 显示文件中第一个域匹配 101 的行（记录）。
10、awk '{$1 == 'Chi' {$3 = 'China'; print}' file 找到匹配行后先将第 3 个域替换后再显示该行（记录）。
awk '{$7 %= 3; print $7}' file 将第 7 域被 3 除，并将余数赋给第 7 域再打印。
11、awk '/tom/ {wage=$2+$3; printf wage}' file 找到匹配行后为变量 wage 赋值并打印该变量。
12、awk '/tom/ {count++;}
END {print "tom was found "count" times"}' file

END 表示在所有输入行处理完后进行处理。

13、awk 'gsub(//$/,"");gsub(/,/,""); cost+=$4;
END {print "The total is $" cost>"filename"}' file
gsub 函数用空串替换$和,再将结果输出到 filename 中。
1 2 3 $1,200.00
1 2 3 $2,300.00
1 2 3 $4,000.00

awk '{gsub(//$/,"");gsub(/,/,"");
if ($4>1000&&$4<2000) c1+=$4;
else if ($4>2000&&$4<3000) c2+=$4;
else if ($4>3000&&$4<4000) c3+=$4;
else c4+=$4; }
END {printf "c1=[%d];c2=[%d];c3=[%d];c4=[%d]/n",c1,c2,c3,c4}"' file
通过 if 和 else if 完成条件语句

awk '{gsub(//$/,"");gsub(/,/,"");
if ($4>3000&&$4<4000) exit;
else c4+=$4; }
END {printf "c1=[%d];c2=[%d];c3=[%d];c4=[%d]/n",c1,c2,c3,c4}"' file
通过 exit 在某条件时退出，但是仍执行 END 操作。
awk '{gsub(//$/,"");gsub(/,/,"");
if ($4>3000) next;
else c4+=$4; }
END {printf "c4=[%d]/n",c4}"' file
通过 next 在某条件时跳过该行，对下一行执行操作。

14、awk '{ print FILENAME,$0 }' file1 file2 file3>fileall
把 file1、file2、file3 的文件内容全部写到 fileall 中，格式为打印文件并前置文件名。

15、awk ' $1!=previous { close(previous); previous=$1 }
{print substr($0,index($0," ") +1)>$1}' fileall
把合并后的文件重新分拆为 3 个文件。并与原文件一致。

16、awk 'BEGIN {"date"|getline d; print d}'
通过管道把 date 的执行结果送给 getline，并赋给变量 d，然后打印。

17、awk 'BEGIN {system("echo "Input your name://c""); getline d;print "/nYour name is",d,"/b!/n"}'
通过 getline 命令交互输入 name，并显示出来。
awk 'BEGIN {FS=":"; while(getline< "/etc/passwd" >0) { if($1~"050[0-9]_") print $1}}'
打印/etc/passwd 文件中用户名包含 050x_的用户名。

18、awk '{ i=1;while(i<NF) {print NF,$i;i++}}' file 通过 while 语句实现循环。
awk '{ for(i=1;i<NF;i++) {print NF,$i}}' file 通过 for 语句实现循环。

type file|awk -F "/" '
{ for(i=1;i<NF;i++)
{ if(i==NF-1) { printf "%s",$i }
else { printf "%s/",$i } }}'
显示一个文件的全路径。

用 for 和 if 显示日期
awk 'BEGIN {
for(j=1;j<=12;j++)
{ flag=0;

```
printf "/n%d 月份/n",j;
for(i=1;i<=31;i++)
{
if (j==2&&i>28) flag=1;
if ((j==4||j==6||j==9||j==11)&&i>30) flag=1;
if (flag==0) {printf "%02d%02d ",j,i}
}
}
}'
```

19、在 awk 中调用系统变量必须用单引号，如果是双引号，则表示字符串

Flag=abcd

awk '{print '$Flag'}' 结果为 abcd

awk '{print "$Flag"}' 结果为$Flag

20. 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。