**awk命令应用**

有三种方式调用awk

**一,命令行方式**

awk [ -F field-separator] 'commands' input-file(s)

 注意：awk -F 域分隔符 ‘命令’,commands 是真正的命令， [ - F域分隔符] 是可选的，因为 awk使用空格作为缺省的域分隔符， 如果以:分割，则为  **awk -F: 'commands' input-file**

示例：  
    1、检测系统中UID为0的用户  
    awk -F: '$3==0 {print $1}' /etc/passwd  
    2、检测系统中密码为空的用户  
    awk -F: 'length($2)==0 {print $1}' /etc/shadow

**二，将所有 a w k命令插入一个文件，并使 a w k程序可执行，然后用 a w k命令解释器作为脚本的首行，以便通过键入脚本名称来调用它。**

**三，将所有的 a w k命令插入一个单独文件，然后调用**

**awk -f awk -script-file input-files**

- f 选项指明在文件 awk\_script\_file中的a w k脚本，input\_file(s)是使用 a w k进行浏览的文件名。

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

awk脚本

在命令中调用a w k时，a w k脚本由各种操作和模式组成。

如果设置了 - F选项，则 a w k每次读一条记录或一行，并使用指定的分隔符分隔指定域，但

如果未设置- F选项，a w k假定空格为域分隔符，并保持这个设置直到发现一新行。当新行出现时，a w k命令获悉已读完整条记录，然后在下一个记录启动读命令，这个读进程将持续到文件

尾或文件不再存在。

模式和动作

任何a w k语句都由模式和动作组成。在一个 a w k脚本中可能有许多语句。模式部分决定动作语句何时触发及触发事件。处理即对数据进行的操作。如果省略模式部分，动作将时刻保持执行状态。模式可以是任何条件语句或复合语句或正则表达式。模式包括两个特殊字段 B E G I N和

E N D。使用B E G I N语句设置计数和打印头。 B E G I N语句使用在任何文本浏览动作之前，之后文本浏览动作依据输入文件开始执行。 E N D语句用来在 a w k完成文本浏览动作后打印输出文本总数和结尾状态标志。如果不特别指明模式， a w k总是匹配或打印行数。

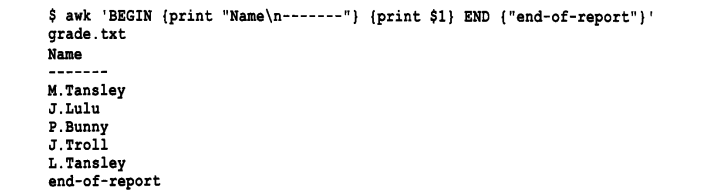
实际动作在大括号 { }内指明。动作大多数用来打印，但是还有些更长的代码诸如 i f 和循环（l o o p i n g）语句及循环退出结构。如果不指明采取动作， a w k将打印出所有浏览出来的记录。

域和记录

a w k执行时，其浏览域标记为 $ 1，$ 2 . . . $ n。这种方法称为域标识。使用这些域标识将更容 易对域进行进一步处理。

使用$ 1 , $ 3表示参照第 1和第3域，注意这里用逗号做域分隔。如果希望打印一个有 5个域 的记录的所有域，不必指明 $ 1 , $ 2 , $ 3 , $ 4 , $ 5，可使用 $ 0，意即所有域。 Aw k浏览时，到达一新行，即假定到达包含域的记录末尾，然后执行新记录下一行的读动作，并重新设置域分隔。注意执行时不要混淆符号 $和s h e l l提示符$，它们是不同的。为打印一个域或所有域，使用 p r i n t命令。这是一个a w k动作（动作语法用圆括号括起来） 。

eg:打印头和尾：



awk错误信息提示：

确保整个a w k命令用单引号括起来。

确保命令内所有引号成对出现。

确保用花括号括起动作语句，用圆括号括起条件语句。

可能忘记使用花括号，也许你认为没有必要，但 a w k不这样认为，将按之解释语法

元字符

这里是 a w k中正则表达式匹配操作中经常用到的字符

\ ^ $ . [] | () \* + ?

这里有两个字符第 7章没有讲到，因为它们只适用于 a w k而不适用于 g r e p或s e d。它们是：

+ 使用+匹配一个或多个字符。

？ 匹配模式出现频率。例如使用 /X Y?Z/ 匹配X Y Z或Y Z

awk条件操作符

操作符   描述

<         小于

> =     大于等于

< =     小于等于

= =     等于

!=       不等于

~        匹配正则表达式

! ~       不匹配正则表达式

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

**AWK内置命令**

ARGC         命令行参数个数

ARGV         命令行参数排列

ENVIRON   支持队列中系统环境变量的使用

FILENAME     awk浏览的文件名

FNR     浏览文件的记录数

FS     设置输入域分隔符，等价于命令行 - F选项

NF     浏览记录的域个数

NR     已读的记录数

OFS     输出域分隔符

ORS     输出记录分隔符

RS     控制记录分隔符

解释：

ARGC支持命令行中传入 awk脚本的参数个数。 ARGV是ARGC的参数排列数组，其中每 一元素表示为ARGV[ n ]，n为期望访问的命令行参数。

ENVIRON 支持系统设置的环境变量，要访问单独变量，使用实际变量名，例如 ENVIRON[“EDITOR”] =“Vi” 。

FILENAME支持a w k脚本实际操作的输入文件。因为 a w k可以同时处理许多文件，因此如 果访问了这个变量，将告之系统目前正在浏览的实际文件。

FNR支持 awk目前操作的记录数。其变量值小于等于 NR。如果脚本正在访问许多文件， 每一新输入文件都将重新设置此变量。

FS用来在a w k中设置域分隔符，与命令行中 - F选项功能相同。缺省情况下为空格。如果用 逗号来作域分隔符，设置 FS = "，"。

NF支持记录域个数，在记录被读之后再设置。

OFS允许指定输出域分隔符，缺省为空格。如果想设置为 #，写入 OFS = " # "。

ORS为输出记录分隔符，缺省为新行（ \ n） 。

RS是记录分隔符，缺省为新行 ( \ n )。

NF的一个强大功能是将变量 $PWD的返回值传入a w k并显示其目录。这里需要指定域分隔 符/。

eg:

$ pwd

/home/unionmon/caohui

$ echo $PWD |awk -F/ '{print $NF}'

caohui

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

只显示修改的记录：（如果是大文件，只显示修改记录方便查看） testfile.log 里的数据：msmd 04 23 sdfrwt 100 23 45

$ awk '{ if($1=="msmd") {$1="haha";print $1}}' testfile.log

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

**内置字符串函数**

gsub(r, s) 在整个$ 0中用s替代r

gsub(r, s，t) 在整个t中用s替代r

index( s , t ) 返回s中字符串t的第一位置

length ( s ) 返回s长度

match(s,r) 测试s是否包含匹配 r的字符串

split( s , a , f s ) 在f s上将s 分成序列a

sprint (fmt,exp) 返回经f m t格式化后的exp

sub( r, s ) 用$ 0中最左边最长的子串代替 s

substr(s ,p) 返回字符串s中从p开始的后缀部分

substr (s,p,n) 返回字符串s 中从p开始长度为n的后缀部分

gsub函数有点类似于sed查找和替换。它允许替换一个字符串或字符为另一个字符串或字符，并以正则表达式的形式执行。第一个函数作用于记录 $ 0，第二个 gsub函数允许指定目标， 然而，如果未指定目标，缺省为 $ 0。

index( s , t ) 函数返回目标字符串 s中查询字符串 t的首位置。

length 函数返回字符串 s字符 长度。

match函数测试字符串 s 是否包含一个正则表达式 r 定义的匹配。

split使用域分隔符 f s将 字符串s 划分为指定序列 a。

sprint函数类似于printf函数（以后涉及） ，返回基本输出格式fmt的结果字符串exp。

sub( r, s )函数将用 s 替代 $ 0 中最左边最长的子串，该子串被（ r ）匹配。

substr(s ,p) 返回字符串s 在位置p后的后缀。

substr (s,p,n) 同上，并指定子串长度为 n。

eg:

$ awk 'gsub(/99/,1212) {print $0}' testfile.log

msmd 04 23 sdfrwt 1212 23 45

$ awk 'BEGIN {print index("msmd","m")}' testfile.log

1

$ awk '$1=="msmd" {print length($1)" " $1}' testfile.log

4 msmd

$ awk 'BEGIN {print length("A FEW GOOD　MEN")}'

15

$ awk 'BEGIN {print match("ANCD",/d/)}'

0

$ awk 'BEGIN {print match("ANCD",/C/)}'

3

$ awk '$1=="msmd" {print match($1,"s")}' testfile.log

2

$ awk 'BEGIN {print split("12324#35#566",myarray,"#")}'

3

s u b s t r是一个很有用的函数。它按照起始位置及长度返回字符串的一部分

$ awk '$1=="msmd" {print substr($1,1,4)}' testfile.log

msmd

这个例子中，指定在域 1的第一个字符开始，返回其前面 4个字符。

s u b s t r的另一种形式是返回字符串后缀或指定位置后面字符。这里需要给出指定字符串及 其返回字串的起始位置。例如，从文本文件中抽取姓氏，需操作域 1，并从第三个字符开始

$ awk '{print substr($1,2)}' testfile.log

smd

注：testfile.log里只有一行记录

还有一个例子，在 B E G I N部分定义字符串，在 E N D 部分返回从第 t 个字符开始抽取的子 串。

$ awk 'BEGIN {STR="a few good men"}END {print substr(STR,7)}' testfile.log

good men

 从shell中向awk传入字符串

本章开始已经提到过，awk脚本大多只有一行，其中很少是字符串表示的。本书大多要求在一行内完成awk脚本，这一点通过将变量传入awk命令行会变得很容易。现就其基本原理讲述一些例子。

eg:使用管道将字符串 stand -by传入awk，返回其长度。

$ echo "Stand-by"|awk '{print length($0)}'

8

eg:文件名为一变量，管道输出到awk ，返回不带扩展名的文件名。

$ STR="mydoc.txt"

$ echo $STR | awk '{print substr($STR,1,5)}'

mydoc

eg:设置文件名为一变量，管道输出到awk ，只返回其扩展名。

$ STR="mydoc.txt"

$ echo $STR | awk '{print substr($STR,7)}'

txt

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

**字符串屏蔽序列**

使用字符串或正则表达式时，有时需要在输出中加入一新行或查询一元字符。

打印一新行时， （新行为字符 \ n ） ，给出其屏蔽序列，以不失其特殊含义，用法为在字符 串前加入反斜线。例如使用 \ n强迫打印一新行。

如果使用正则表达式，查询花括号（ { }） ，在字符前加反斜线，如 / \ { / ，将在a w k中失掉其 特殊含义。

awk中使用的屏蔽序列:

\b     退格键

\t     tab键

\f     走纸换页

\n     新行

\c     任意其他特殊字符，例如 \ \为反斜线符号

\r     回车键

\ddd     八进制值

eg:

使用上述符号，打印 May Day，中间夹 t a b键，后跟两个新行，再打印 May Day，但这次 使用八进制数104、141、171、分别代表 D、a、y。

$ awk 'BEGIN {print "\n\May\tDay\n\nMay \104\141\171"}'

May Day

May Day

注意:\104为D的八进制ASCII码，\141为a的八进制ASCII码，等等。

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

awk 输出函数printf

目前为止，所有例子的输出都是直接到屏幕，除了 t a b 键以外没有任何格式。 a w k提供函 数printf ，拥有几种不同的格式化输出功能。例如按列输出、左对齐或右对齐方式.

每一种printf 函数（格式控制字符）都以一个 %符号开始，以一个决定转换的字符结束。转换包含三种修饰符。

printf函数基本语法是

 printf （[格式控制符 ]，参数） ，格式控制字符通常在引号里。

**awk printf 修饰符:**

-         左对齐

width  域的步长，用 0表示0步长

.prec    最大字符串长度，或小数点的位数

%c      ASCII字符

%d     整数

%e     浮点数，科学记数法

%f      浮点数，例如（1 2 3 . 4 4）

%g     awk决定使用哪种浮点数转换 e或者f

%o     八进制数

%s     字符串

%x     十六进制数

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

**字符转换**

观察A S C I I码中6 5的等价值。管道输出 6 5到a w k。p r i n t f进行A S C I I码字符转换。这里也加 入换行，因为缺省情况下 p r i n t f不做换行动作。

$ echo "65" |awk '{printf "%c\n",$0 }'

A

当然也可以按同样方式使用 a w k得到同样结果。

$ awk 'BEGIN {printf "%c\n",65}'

A

所有的字符转换都是一样的，下面的例子表示进行浮点数转换后‘ 9 9 9’的输出结果。整 数传入后被加了六个小数点。

$ awk 'BEGIN {printf "%f\n",999}'

999.000000

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

**格式化输出：**

打印所有的学生名字和序列号，要求名字左对齐， 1 5个字符长度，后跟序列号。注意 \ n 换行符放在最后一个指示符后面。输出将自动分成两列。

$ awk '{printf "%-15s %s\n",$1,$3}' testfile.log

msmd 23

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

**向一行awk命令传值：**

awk 命令变量=输入文件值

eg:

下面的例子在命令行中设置变量 A G E等于1 0，然后传入 a w k中，查询年龄在 1 0岁以上的所有学生。

$ awk '{ if ($5>AGE) print $0}' AGE=10 testfile.log

msmd 04 23 sdfrwt 99 23 45

awk也允许传入环境变量。下面的例子使用环境变量 LOGNAME支持当前用户名。可从 who命令管道输出到awk中获得相应信息

$ who | awk '{ if ($1==user) print $1 " ,you are connected to" $2 }' user=$LOGNAME

unionmon ,you are connected topts/8

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

**awk数组 :**

数组使用前，不必定义，也不必指定数组元素个数。经常使用循环来访问数组。下面是一种循环类型的基本结构：

For (element in array ) print array[element]

对于记录“ 1 2 3 # 4 5 6 # 6 7 8” ，先使用 s p l i t 函数划分它，再使用循环打印各数组元素。