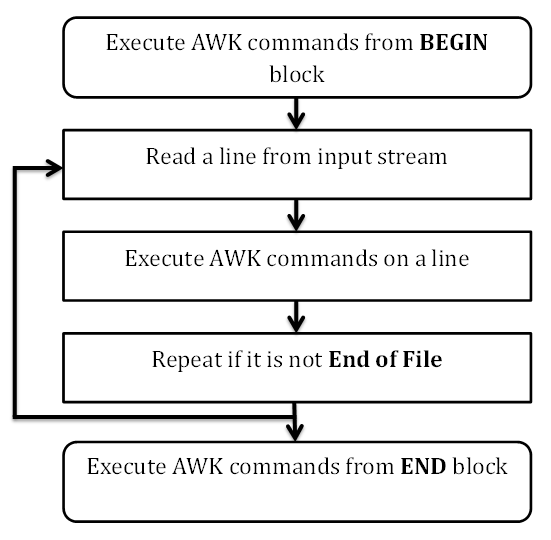
# Awk教程

## 一、awk 简介

awk是一种解释执行的编程语言，专门用来处理文本数据的。

## 二、awk工作流程：

Awk内部工作原理；AWK执行的流程非常简单：读( Read )、执 行（ Execute )与重复( Repeat )；下面是流程图：

Awk工作流程图

读( Read )：AWK 从输入流（文件、管道或者标准输入）中读入一行然后将其存入内存中。

执行(Execute)：对于每一行输入，所有的 AWK 命令按顺执行。 默认情况下，AWK 命令是针对于每一行输入，但是我们可以将其限制在指定的模式中。

重复（Repeate）：一直重复上述两个过程直到文件结束。

## awk程序结构

Awk程序结构主要包括三块：BEGIN（开始块）、主体块、END（结束块）；

开始块（BEGIN BLOCK）：

语法：

BEGIN{awk-commands}

开始块就是awk程序启动时执行的代码部分（在处理输入流之前执行），并且在整个过程中只执行一次；一般情况下，我们在开始块中初始化一些变量。BEGIN是awk的关键字，因此必须要大写。【注：开始块部分是可选，即你的awk程序可以没有开始块部分】

主体块（Body Block）：

语法：

/pattern/{awk-commands}

针对每一个输入的行都会执行一次主体部分的命令，默认情况下，对于输入的每一行，awk都会执行主体部分的命令，但是我可以使用/pattern/限制其在指定模式下。

结束块（END BLOCK）：

语法：

END{awk-commands}

结束块是awk程序结束时执行的代码（在处理完输入流之后执行），END也是awk的关键字，必须大写，与开始块类似，结束块也是可选的。

### 示例1：

题目：有一个文本a.txt，内容如下：

1 zhengchengcheng math 100

2 wangbin chinese 80

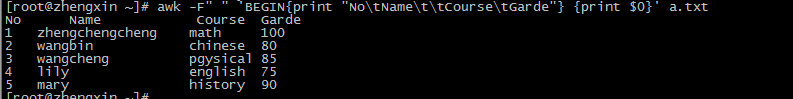
3 wangcheng pgysical 85

4 lily english 75

5 mary history 90

使用awk将其标准输出并附带表头信息？

解答：



## awk基本语法

### Awk命令行模式：

一般情况下，我们使用awk命令行模式比较多，即直接使用awk命令处理文本数据；

Awk命令行格式如下：

awk [options] ‘{awk-commands}’ file

### Awk程序文件：

当然，除了直接使用awk命令来处理文本之外，我们还可以将awk命令写成脚本来执行，即awk程序文件。

Awk程序文件执行格式如下：

awk [options] -f file

### Awk标准选项：

常用的awk选项如下：

-v：这个选项可以为变量赋值，它允许在程序执行之前为变量赋值【注：一般用法即在程序执行前为变量赋值】

例如： awk -v name='xin' 'BEGIN{printf "name is: %s\n",name}'

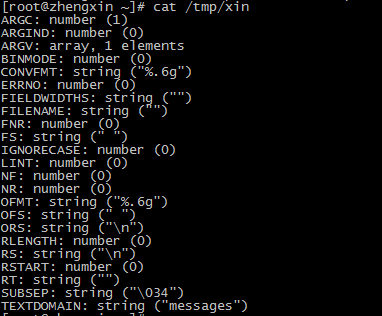
执行结果如下：

IMG_256

--dump-variables[=file]：此选项会将awk全局变量及相应值按序输出到指定文件中，不指定文件，默认输出文件名是awkvars.out

执行结果如下：

awk --dump-variables=/tmp/xin '' 【注：’’引号不能少，不然会报错，代表awk主体部分；--dump-variables=/tmp/xin表示将内容输出到/tmp/xin文件中去】



--lint[=fatal]：这个选项用于检测程序的可移植情况以及代码中的可以部分，如果提供了参数fatal，awk会将所有的警告信息当错误信息输出。

例如：awk --lint '' /tmp/xin 【注：同样，强调一遍，主题部分不能省略，没处理命令使用’’代替即可，检测文件为/tmp/xin】

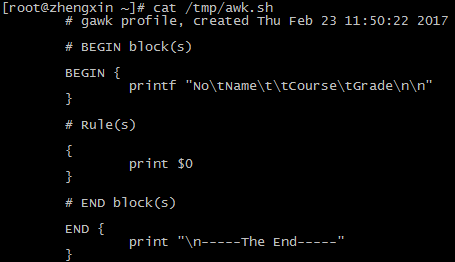
执行结果如下：

IMG_256

--profile[=file]：此选项会将程序文件以一种很优美的格式输出（常用来格式化awk脚本文件），默认输出文件是awkprof.out，[=file]表示将格式化后的脚本内容输出到指定文件。

例如：awk -F" " --profile=/tmp/awk.sh 'BEGIN{printf "No\tName\t\tCourse\tGrade\n\n"} {print $0} END{print"\n-----The End-----"}' a.txt 【注：使用awk格式化输出a.txt内容，并将命令以一种优美的方式输出到/tmp/awk.sh文件中去】

执行结果如下：



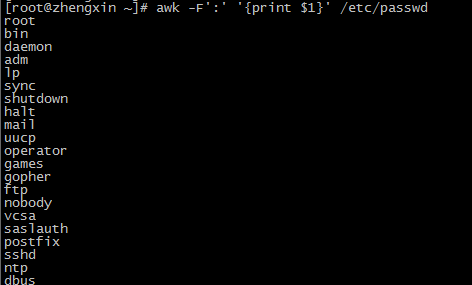
--version：显示awk版本信息。

命令：awk --version

-F：指定域分隔符。【用来分隔每条记录中各个域的】

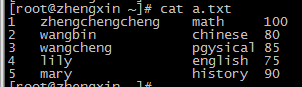
例如：awk -F':' '{print $1}' /etc/passwd 【注：打印系统所有的用户；$1代表第一个域】

执行结果如下：



## awk基本示例

我们以a.txt文件作为示例文件，a.txt文件内容如下：



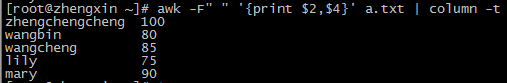
### 打印列或域：

在awk中，对于输入流，输入的每行，叫做一条记录；每一条记录中以分隔符分隔的各个部分叫做域，awk每次载入一条记录进行处理，默认的分隔符为空格。

例如：打印出a.txt文件中姓名和成绩这两列，使用awk进行操作；

命令：awk -F" " '{print $2,$4}' a.txt | column -t 【注：column -t是按比较美观的格式显示输出】

执行结果如下：



### 输出整行：

$0代表输出整条记录。

例如：输出姓名为wangbin的那行，用awk去实现？

命令：awk -F" " '/wangbin/{print $0}' a.txt 【这条语句先用/pattern/去匹配，匹配到后在输出整行内容】

执行结果如下：

IMG_256

【注：如果没有主体块（即：{print $0}），则默认是输出记录（行）】

### 匹配模式输出指定的列（域）：

例如：输出姓名为wangbin那行中的第1,2,4列，用awk实现？

命令：awk -F" " '/wangbin/{print $1,$2,$4}' a.txt

执行结果如下：

IMG_256

### 计算匹配次数并输出：

例如：计算姓名以w开头的行的个数，用awk实现？

命令： awk -F" " '/w/{count=1;count+=1} END{print "count=",count}' a.txt

或

awk -F' ' '/w/{++count} END {printf "count is: %d\n",count}' a.txt

执行结果如下：

IMG_256

### 输出字符数多于33的行：

例如：打印输出字符串个数多于33的行，用awk实现？

命令：awk -F" " 'length($0) > 33{print $0}' a.txt

或者

awk -F" " 'length($0) > 33' a.txt

执行结果如下：

IMG_256

【注：length函数是awk的内置函数，该函数返回字符串的长度，变量$0表示整行，缺失的主体块会执行默认动作（打印输出）；如果一行中的字符数超过33，则length($0) > 33为真，则打印输出，否则不打印】

## awk内置变量

Awk提供了一些内置变量，在写脚本或实际运用中起到很重要的作用。

### 1、标准awk变量：

（1）ARGC

ARGC表示在命令行提供的参数的个数；【注：实际参数个数=ARGC -1,因为第一个参数永远是awk】

例如：awk 'BEGIN{print "arguments = ",ARGC -1}' xin ss ll w

执行结果如下：

IMG_256

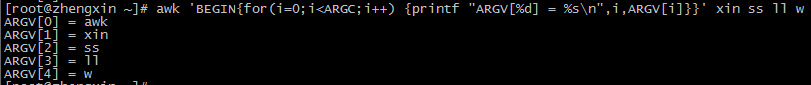
【注：从上面可以看出，ARGC = 实际参数个数+1】

（2）ARGV

ARGV表示将命令行输出的参数存储为一个数组，ARGV即是这个数组的名称；数组的有效索引从0到ARGC-1；

例如：awk 'BEGIN{for(i=0;i<ARGC;i++) {printf "ARGV[%d] = %s\n",i,ARGV[i]}}' xin ss ll w 【这条命令是循环打印出所有参数】

执行结果如下：



【注：第一个参数永远是awk，索引号为0】

（3）ENVIRON

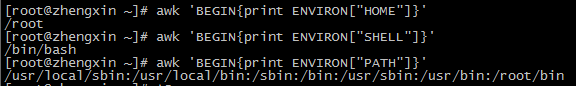
ENVIRON表示与环境变量相关的关联数组变量；【注：即使用这个内置变量可以查看当前系统环境下的环境变量设置，用法为ENVIRON[“环境变量名”]】

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

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

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

执行结果如下：

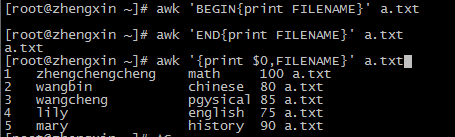


（4）FILENAME

FILENAME表示当前文件名称；

例如：awk 'END{print FILENAME}' a.txt

执行结果如下：



【注：开始块中FILENAME未定义，因为BEGIN开始块是在文本处理操作之前执行的，所以FILENAME还没定义，该变量的值为空；一般用于END结束块，打印出当前文件名称】

（5）FS

FS表示输入时的域分隔符，其默认值是空格，可以使用-F参数指定与分隔符；

例如：awk 'BEGIN{print "FS=" FS}' | cat -A

执行结果如下：

IMG_256

【注：cat -A是显示行结束标识符，linux默认是$；可以看出FS默认是空格】

也可以使用-F参数指定域分隔符；

例如：awk -F: 'BEGIN{print "FS=" FS}' | cat -A

执行结果如下：

IMG_256

【注：上图显示FS变成了:】

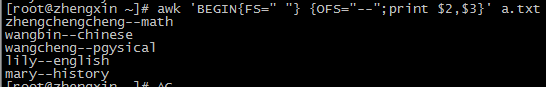
（6）OFS

OFS表示输出时的域分隔符，其默认值也是空格；

例如：打印输出a.txt文件中的第2,3列，并改变与分隔符为”--”,使用awk实现；

命令： awk 'BEGIN{FS=" "} {OFS="--";print $2,$3}' a.txt

执行结果如下：



（7）NF

NF代表当前输入记录中域的数量；

例如：匹配输出姓名为wangbin的行，并显示这行的域总数；

命令：awk -F" " '/wangbin/{printf "context: (%s)\nNF: (%s)\n",$0,NF}' a.txt | column -t

执行结果如下：

IMG_256

（8）NR

NR表示当前输入记录数（即行号）；

例如：打印输出a.txt文件第2-4行（即1<NR<=4）的内容，用awk实现；

命令：awk -F" " '{if(NR<=4 && NR>1) print NR,$0}' a.txt | column -t

执行结果如下：

IMG_256

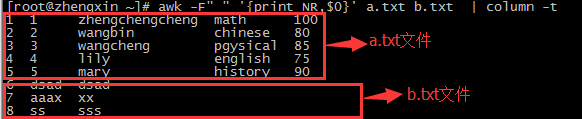
（9）FNR

FNR也表示记录数（行号），与NR类似，但是不同在于它是相对于文件而言的，此变量在处理多个文件时有重要的作用，每当从新的文件读入时，FNR都会被重新置为0；

NR与FNR比较：

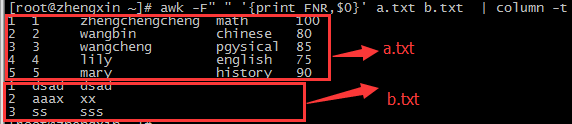
例如：现在有两个文件a.txt，b.txt，现在要求打印输出并添加行号，分别使用NR和FNR对比效果；

NR显示结果：



【注：如上图执行结果显示，NR是相对于整个输入流而言，行号不断从1开始递增，所以无法通过NR来区分多个文件】

FNR显示结果：



【注：如上图结果显示，FNR是相对于文件而言的，当读入新文件时，FNR会被重新置为0，即当一个文件结束，开始读入第二个文件时，行号重新从1开始递增，所以可以通过FNR来区分多个文件】

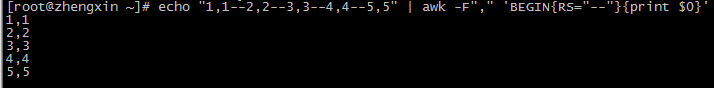
（10）RS

RS表示输入记录（行）分隔符【即输入时的记录分隔符】，默认是换行符，可以使用RS指定输入记录分隔符；

例如：echo "1,1--2,2--3,3--4,4--5,5" | awk -F"," 'BEGIN{RS="--"}{print $0}'

上面的语句：输入文本内容为"1,1--2,2--3,3--4,4--5,5"，默认的记录分隔符为换行符，我们使用RS="--"指定记录分隔符为--后，awk主体程序就需要处理五条记录【这五条记录分别为：1,1、2,2、3,3、4,4、5,5】

执行结果如下：



【注：以上就是显示的结果，当然大家可能会发现为什么结尾会多一个空包行呢，print最后默认就会输出一个换行，这个可以不用理会】

（11）ORS

ORS表示输出记录（行）分隔符【即输出时的记录分隔符】，默认是换行符，可以使用ORS指定输出记录分隔符；

例如：将a.txt文件不换行输出，并指定输出记录分隔符为||，用来区分原来的每一条记录，用awk实现；

命令：awk '{ORS=" || ";print $0}' a.txt

执行结果如下：

IMG_256

【结果显示，已将换行符换成了||】

（12）RLENGTH

RLENGTH表示match函数匹配的字符串长度，awk的match函数用于在输入的字符串中搜索指定字符串；

例如：awk 'BEGIN{if(match("this is good","good"));print RLENGTH}' 【表示match函数在输入字符串中匹配good字符串，RLENGTH返回匹配到的good字符串的长度】

执行结果如下：

IMG_256

（13）RSTART

RSTART表示由match函数匹配的字符串的第一个字符的位置【注：是匹配字符串的第一个字符在所有输入的内容中的位置】；

例如：awk 'BEGIN{if(match("this is good","good"));print RSTART}' 【这表示match匹配到的good字符串中的第一个字符”g”在输入内容中”this is good”的位置】

执行结果如下：

IMG_256

1. $0

$0表示输入的整个输入记录；

（15）$n

$n表示当前输入记录的第n个域，这些域之间由FS分隔，n的取值为1,2,3......；

### awk特定变量（即专有变量）

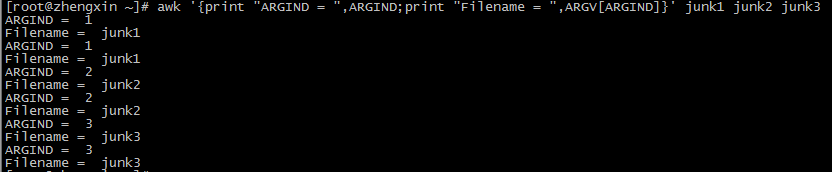
#### （1）ARGIND

ARGIND表示当前文件中正在处理的ARGV数组的索引值；

例子：有三个文件分别是junk1、junk2、junk3，每个文件中都有两行；

命令： awk '{print "ARGIND = ",ARGIND;print "Filename = ",ARGV[ARGIND]}' junk1 junk2 junk3

执行结果如下：



【注：ARGIND其实就是每个文件的序列号，例如第一个文件是junk1，那么这时ARGIND=1，且会遍历该文件中所有行，有多少行就会输出多少个ARGIND=1（这变量生产中我一次也没用到过）】

#### ERRNO

ERRNO用于存储当getline重定向失败或者close函数调用失败时的失败信息；

例子：awk 'BEGIN{ret=getline<"xin.t";if(ret==-1) print "error: ",ERRNO}'

执行结果如下：

IMG_256

【注：因为xin.t这个文件并不存在，报错，错误信息保存到了ERROR变量中】

#### IGNORECASE

IGNORECASE表示忽略大小写；

例如：找出a.txt文件中name=wangbin这一行；使用awk这一行

首先不开启，即IGNORECASE=0的情况（区分大小写）：

执行结果如下：

IMG_256

开启IGNORECASE，即IGNORECASE=1的情况（忽略大小写）：

执行结果如下：

IMG_256

#### TEXTDOMAIN

TEXTDOMAIN表示awk程序当前文本域，它主要是来寻找程序中字符串的本地翻译，用于程序的国际化；默认值是messages；

例如： awk 'BEGIN{print TEXTDOMAIN}'

执行结果如下：

IMG_256

## awk操作符

### 算数运算符

#### 加法（+）

例子：awk 'BEGIN{a=5;b=3;printf "a + b = %d\n",a+b}'

或者

awk 'BEGIN{a=5;b=3;print "a + b =",a+b}'

执行结果如下：

IMG_256

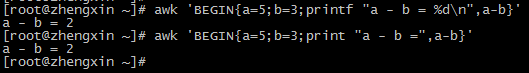
#### 减法（-）

例子：awk 'BEGIN{a=5;b=3;printf "a - b = %d\n",a-b}'

或者

awk 'BEGIN{a=5;b=3;print "a - b =",a-b}'

执行结果如下：



#### 乘法（\*）

例子：awk 'BEGIN{a=5;b=3;printf "a \* b = %d\n",a\*b}'

或者

awk 'BEGIN{a=5;b=3;print "a \* b =",a\*b}'

执行结果如下：

IMG_256

#### 除法（/）【awk除法可以保留精度】

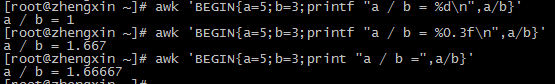
例子：awk 'BEGIN{a=5;b=3;printf "a / b = %d\n",a/b}' 【结果只保留整数】

awk 'BEGIN{a=5;b=3;printf "a / b = %0.3f\n",a/b}' 【结果保留3位精度】

或者

awk 'BEGIN{a=5;b=3;print "a / b =",a/b}' 【默认print输出保留5位精度】

执行结果如下：



#### （5）模运算符（%）【即取余运算】

例子：awk 'BEGIN{a=5;b=3;printf "a % b = %d\n",a%b}'

或者

awk 'BEGIN{a=5;b=3;print "a % b =",a%b}'

执行结果如下：

IMG_256

#### 递增运算符之前置递增运算（如++a）

例子：awk 'BEGIN{a=5;b=++a;printf "a = %d, b = %d\n",a,b}'

执行结果如下：

IMG_256

【注：b=++a表示先执行递增操作，后执行赋值操作，即先执行a=a+1，在执行b=a，所以结果a和b都为6】

#### 递增运算符之后置递增运算（如a++）

例子：awk 'BEGIN{a=5;b=a++;printf "a = %d, b = %d\n",a,b}'

执行结果如下：

IMG_256

【注：b=a++表示先执行赋值操作，后执行递增操作，即先执行b=a，再执行a=a+1，所以结果a=6，b=5】

#### 递减运算符之前置递减运算（如--a）

例子：awk 'BEGIN{a=5;b=--a;printf "a = %d, b = %d\n",a,b}'

执行结果如下：

IMG_256

【注：b=--a表示先执行递减操作，后执行赋值操作，即先执行a=a-1，再执行b=a，所以结果a=4，b=4】

#### 递减运算符之后置递减运算（如a--）

例子：awk 'BEGIN{a=5;b=a--;printf "a = %d, b = %d\n",a,b}'

执行结果如下：

IMG_256

【注：b=a--表示先执行赋值操作，后执行递减操作，即先执行b=a，再执行a=a-1，所以结果a=4，b=5】

#### 赋值运算符

##### 简单赋值（=）

例子：awk 'BEGIN{name = "zx";printf "name = %s\n",name}'

执行结果如下：

IMG_256

##### 加法赋值（+=）

例子：awk 'BEGIN{cnt=10;cnt+=10;printf "cnt = %d\n",cnt}'

执行结果如下：

IMG_256

【注：首先给cnt变量赋值为10，在使用加法赋值将cnt值增加10；cnt+=10等价于cnt=cnt+10】

##### 减法赋值（-=）

例子：awk 'BEGIN{cnt=90;cnt-=10;printf "cnt = %d\n",cnt}'

执行结果如下：

IMG_256

【注：首先给cnt变量赋值为90，再使用减法赋值将cnt的值减少10；cnt-=10等价于

cnt=cnt-10】

##### 乘法赋值（\*=）

例子：awk 'BEGIN{cnt=90;cnt\*=10;printf "cnt = %d\n",cnt}'

执行结果如下：

IMG_256

【注：首先给cnt变量赋值为90，再使用乘法赋值将cnt乘以10；cnt\*=10登记于cnt=cnt\*10】

##### 除法赋值（/=）

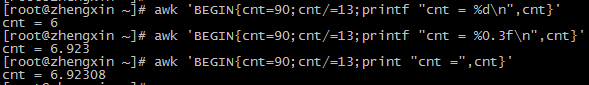
例子：awk 'BEGIN{cnt=90;cnt/=13;printf "cnt = %d\n",cnt}' 【结果只保留整数部分】

awk 'BEGIN{cnt=90;cnt/=13;printf "cnt = %0.3f\n",cnt}' 【结果保留3位精度】

或者

awk 'BEGIN{cnt=90;cnt/=13;print "cnt =",cnt}' 【默认print输出保留5位精度】

执行结果如下：



【注：首先给cnt变量赋值为90，再使用除法赋值将cnt除以13；cnt/=13登记于cnt=cnt/13】

##### 模运算赋值（%=）

例子：awk 'BEGIN{cnt=90;cnt%=13;print "cnt =",cnt}'

执行结果如下：

IMG_256

【注：首先给cnt变量赋值为90，再使用模运算赋值将cnt除以13取余数；cnt%=13等价于cnt=cnt%13】

##### 指数赋值（^=）

例子：awk 'BEGIN{cnt=3;cnt^=3;print "cnt =",cnt}'

执行结果：

IMG_256

【注：首先给cnt变量赋值为3，再使用指数赋值计算cnt的次幂；cnt^=3等价于cnt=cnt^3】

### 2、关系运算符

#### 等于（==）

例子：awk 'BEGIN{a=5;b=6;if(a == b) print "a == b";else print "a!=b"}'

执行结果如下：

IMG_256

#### 不等于（!=）

例子：awk 'BEGIN{a=5;b=6;if(a != b) print "true";else print "false"}'

执行结果如下：

IMG_256

#### 小于（<）

例子：awk 'BEGIN{a=5;b=6;if(a < b) print "true";else print "false"}'

执行结果如下：

IMG_256

#### 大于（>）

例子：awk 'BEGIN{a=5;b=6;if(a > b) print "true";else print "false"}'

执行结果如下：

IMG_256

#### 大于或等于（>=）

例子：awk 'BEGIN{a=5;b=6;if(a >= b) print "true";else print "false"}'

执行结果如下：

IMG_256

#### 小于或等于（<=）

例子：awk 'BEGIN{a=5;b=6;if(a <= b) print "true";else print "false"}'

执行结果如下：

IMG_256

### 逻辑运算符

#### 逻辑与（&&）

语法：expr1 && expr2

解释说明：如果expr1和expr2均为真，则最终结果为真，否则为假；【注：如果expr1为真时才会执行expr2表达式，若expr1为假，则直接返回结果为假】

例子：awk 'BEGIN{a=6;if(a > 0 && a < 6) print "true";else print "false"}'

或者

awk 'BEGIN{a=6;if(a > 0 && a <= 6) print "true";else print "false"}'

执行结果如下：

IMG_256

#### 逻辑或（||）

语法：expr1 || expr2

解释说明：如果expr1和expr2至少其中一个为真，则最终结果为真，二者均为假时结果才为假；【注：如果expr1为假时才会执行expr2表达式，若expr1为真，则不会再执行expr表达式，直接返回结果为真】

例子：awk 'BEGIN{a=6;if(a > 0 || a < 6) print "true";else print "false"}'

或者

awk 'BEGIN{a=6;if(a < 0 || a < 6) print "true";else print "false"}'

执行结果如下：

IMG_256

#### （3）逻辑非（!）

语法：!expr1

解释说明：逻辑非将expr1的值取反，如果expr1为真，则返回0，否则返回1；

例子：

执行结果如下：

IMG_256

【注：length(name)是用来判断name变量是否为空，不为空为真（0），为空为假（1）】

### 三元运算符

语法：condition expression?statement1:statement2

解释说明：当条件表达式（condition expression）为真时，statement1执行，否则statement2执行。

例子：以下例子是打印最大值；

awk 'BEGIN{a=20;b=30;c=40;(a>b)?max=a:max=b;(c>max)?max=c:max=max;print "Max is: " max}'

执行结果如下：

IMG_256

### 一元运算符

#### （1）一元加运算

一元加运算符表示为+，它将操作数乘以+1；

例子： awk 'BEGIN { a = -10; a = +a; print "a =", a }'

执行结果如下：

IMG_256

【注：首先给a变量赋值-10，a=+a表示a=a\*1，最终结果为-10】

#### （2）一元减运算

一元减运算符表示为-，它将操作数乘以-1；

例子：awk 'BEGIN { a = -10; a = -a; print "a =", a }'

执行结果如下：

IMG_256

【注：首先给a变量赋值-10，a=-a表示a=a\*(-1)，最终结果为10】

### 字符串连接操作符（即空格符）

空格操作符（space）可以完成两个字符串的连接操作；

例子：awk 'BEGIN { str1="Hello,"; str2="World"; str3 = str1 str2; print str3 }'

执行结果如下：

IMG_256

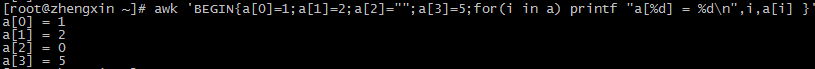
【注：str3 = str1 str2表示将str1和str2的值拼接在一起赋值给str3，然后打印输出str3】

### 数组成员操作符（in）

数组成员操作符为in，该操作符用于访问数组元素。

例子：

执行结果如下：



【注：这个例子中i代表a数组中的元素的下标，打印输出数组中所有元素】

### 8、正则表达式操作符

#### （1）匹配（~）

匹配操作符是~，它用于搜索包含匹配模式字符串的域；

例如：打印输出a.txt文件中姓名包含w字符的行；用awk实现。

awk -F" " '$2~"w"{print $0}' a.txt

执行结果如下：

IMG_256

#### 不匹配（!~）

匹配操作符是!~，它用于搜索不包含匹配模式字符串的域；

例如：打印输出a.txt文件中姓名不包含w字符的行；用awk实现。

awk -F" " '$2!~"w"{print $0}' a.txt

执行结果如下：

IMG_256

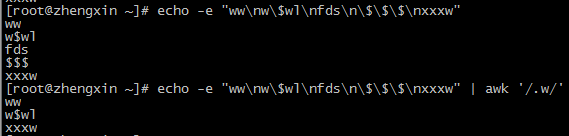
## awk正则表达式

### 点（.）

点字符（.）可以匹配除了行结束符的任意单个字符；【如果要匹配行结束符$，需要用\转义】

例如：echo -e "ww\nw\$wl\nfds\n\$\$\$\nxxxw" | awk '/.w/'

执行结果如下：

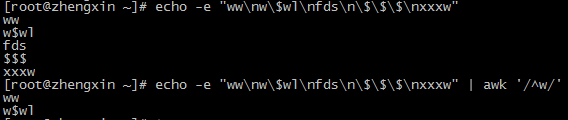


### 行开始（^）

行开始符（^）匹配一行的开始；

例如：echo -e "ww\nw\$wl\nfds\n\$\$\$\nxxxw" | awk '/^w/'

执行结果如下：



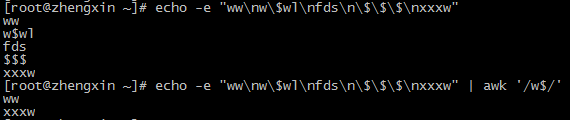
【注：匹配以w字符开头的行】

### 行结束（$）

行结束符（$）匹配一行的结束；

例如：echo -e "ww\nw\$wl\nfds\n\$\$\$\nxxxw" | awk '/w$/'

执行结果如下：

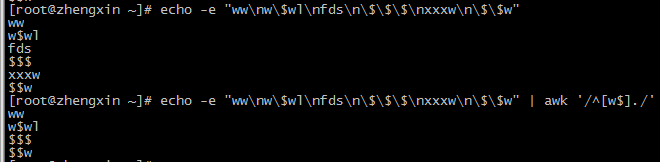


### 匹配字符集（[str1str2...]）

匹配字符集用于匹配集合（由方括号表示）中的一个字符；

例如：echo -e "ww\nw\$wl\nfds\n\$\$\$\nxxxw" | awk '/^[w$]./' 【匹配行开始为w或$的行】

执行结果如下：

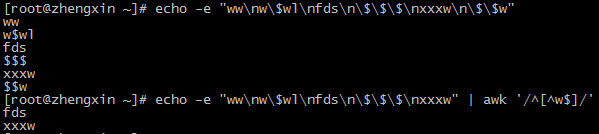


### 排除集（[^str1str2...]）

正则匹配时会排除集合中的字符；

例如：echo -e "ww\nw\$wl\nfds\n\$\$\$\nxxxw" | awk '/^[^w$]/' 【排除以w或$开始的行】

执行结果如下：

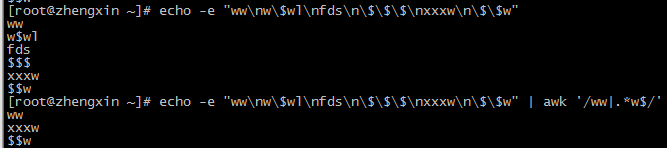


### 或（|）

竖线（|）允许正则表达式实现逻辑或运算；

例如：echo -e "ww\nw\$wl\nfds\n\$\$\$\nxxxw\n\$\$w" | awk '/ww|.\*w$/'

执行结果如下：

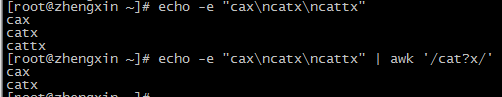


### 最多出现一次（?）

?表示前面的字符不出现或者出现一次；【?前面的字符作为参考字符，不算在初始内容内】

例子：echo -e "cax\ncatx\ncattx" | awk '/cat?x/'

执行结果如下：



【注：上述例子表示打印输出匹配前面的t 0次或1次的行】

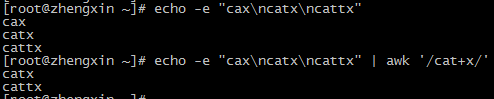
【注：参考字符为t，得出的结果分别匹配t：0次、1次】

### 至少出现一次（+）

+表示匹配前面的字符至少一次；

例子：echo -e "cax\ncatx\ncattx" | awk '/cat+x/'

执行结果如下：



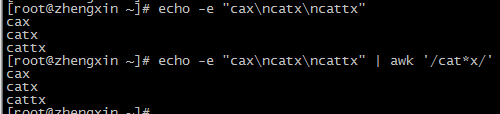
【注：参考字符为t，得出的结果分别匹配t：1次、2次】

### 出现0次或多次（\*）

\*表示匹配前面的字符任意个；

例子：echo -e "cax\ncatx\ncattx" | awk '/cat\*x/'

执行结果如下：



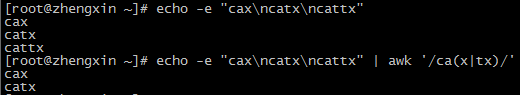
【注：参考字符为t，得出的结果分别匹配t：0次、1次、2次】

### 分组（()）

()用于分组，|用来提供多种选择；

例子：echo -e "cax\ncatx\ncattx" | awk '/ca(x|tx)/'

执行结果如下：



## awk数组

AWK 有关联数组这种数据结构，而这种数据结构最好的一个特点就是它的索引值不需要是连续的整数值。我们既可以使用数字也可以使用字符串作为数组的索引。除此之外，关联数组也不需要提前声明其大小，因为它在运行时可以自动的增大或减小。

awk数组语法：

array\_name[index]=value

【其中array\_name是数组名称，index是数组索引，value是数组元素所赋予的值】

### 创建数组

例如：创建一个数组age；使用awk实现；

awk 'BEGIN{age["zx"]=25;age["wc"]=26;age["wb"]=27;for(i in age) printf "age[%s]\t= %d\n",i,age[i]}'

执行结果如下：

IMG_256

【注：上面的例子表示的是定义一个数组age，并打印age数组中所有的元素】

访问数组中元素：arrary\_name[index]

### 删除数组元素

删除数组元素语法：delete array\_name[index]

例如：awk 'BEGIN{age["zx"]=25;age["wc"]=26;age["wb"]=27;delete age["zx"];for(i in age) printf "age[%s]\t= %d\n",i,age[i]}'

执行结果如下：

IMG_256

### 多维数组

Awk本身只支持多维数组，不过我们可以很容易的使用一维数组模拟多维数组；

例如：以下是一个二维数组：

100 200 300

400 500 600

700 800 900

【上面的实例中，array[0][0] 存储 100，array[0][1] 存储 200 ，依次类推】

在awk中，为了在array[0][0]处存储100，我们可以使用如下语法：

array[“0,0”] = 100 【该实例中我们使用0,0作为索引，但是这并不是两个索引，事实上，它是一个字符串索引0,0】

例子：打印输出上述二维数组，使用awk实现；

awk 'BEGIN{a["0,0"] = 100;a["0,1"] = 200;\

a["0,2"] = 300;a["1,0"] = 400;\

a["1,1"] = 500;a["1,2"] = 600;\

a["2,0"] = 700;a["2,1"] = 800;\

a["2,2"] = 900;printf "a[0][0] = %s\n",a["0,0"];\

printf "a[0][1] = %s\n",a["0,1"];printf "a[0][2] = %s\n",a["0,2"];\

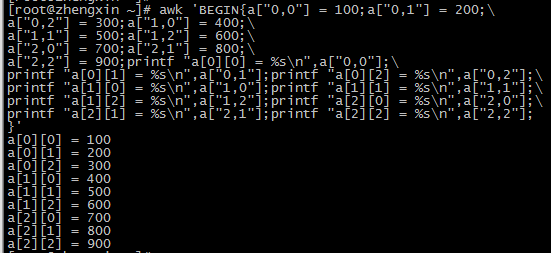
printf "a[1][0] = %s\n",a["1,0"];printf "a[1][1] = %s\n",a["1,1"];\

printf "a[1][2] = %s\n",a["1,2"];printf "a[2][0] = %s\n",a["2,0"];\

printf "a[2][1] = %s\n",a["2,1"];printf "a[2][2] = %s\n",a["2,2"];

}'

执行结果如下：



## awk控制流（if else）

与其实的编程语言一样，AWK 同样提供了条件语句控制程序的执行流程；

### if语句

条件语句测试条件然后根据条件选择执行相应的动作。下面是条件语句的语法：

第一种写法：

if(condition)

action

第二种写法：

if(condition)

{

action 1

action 2

......

action n

}

例子：判断数字为奇数还是偶数？

awk 'BEGIN{a = 10;if(a%2==0) print "a is oushu";else print "a is jishu"}'

或者

awk -v a=21 'BEGIN{if(a%2==0) print "a is oushu";else print "a is jishu"}'

执行结果如下：

IMG_256

### if-else语句

if-else语句中允许在条件为假时执行另外一组的动作。下面为 if-else 的语法格式：

if (condition)

action-1

else

action-2

例子：awk 'BEGIN{a = 30;b = 20;if(a==b) print "a = b";else print "a != b"}'

执行结果如下：

IMG_256

### if-else-if语句梯

使用多个 if-else 语句构造 if-else-if 梯从而实现多个条件的判断【类似于shell的if-elif-else语法】；

if-else-if语法：

if(condition-1)

action-1

else if(condition-2)

action2

else if(condition-3)

action3

else

action4

例子：awk 'BEGIN{a = 30;b = 20;if(a==b) print "a = b";else if(a>b) print "a > b";else print "a < b"}'

执行结果如下：

IMG_256

## awk循环（for、while）

除了前面介绍的条件语句，AWK 还提供了循环语句。该语句的作用就是当条件为真时重复执行一系列的命令。

### For循环

for 循环的语法如下：

for (initialisation; condition; increment/decrement)

{

action

}

【For 语句首先执行初始化动作( initialisation )，然后再检查条件( condition )。如果条件为真，则执行动作( action )，然后执行递增( increment )或者递减( decrement )操作。只要条件为真循环就会一直执行。每次循环结束都会进条件检查，若条件为假则结束　循环】

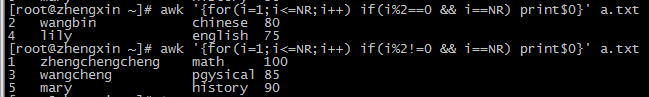
例子：分别打印输出a.txt文件中的奇数行与偶数行？

awk '{for(i=1;i<=NR;i++) {if(i%2==0 && i==NR) print$0}}' a.txt

或者

awk '{for(i=1;i<=NR;i++) {if(i%2!=0 && i==NR) print$0}}' a.txt

执行结果如下：



【注：for循环语句循环体可以不加{}，但是为了规范还是加上比较好，以防出错，】

### while循环

While 循环会一直执行动作直到逻辑条件为假为止。其语法如下：

while (condition)

{

action1

action2

......

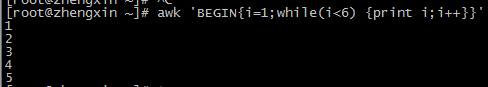
}

【AWK 首先检查条件是否为真，若条件为真则执行动作。此过程一直重复直到条件为假时，则停止。】

例子：循环打印输出1到5；

awk 'BEGIN{i=1;while(i<6) {print i;i++}}' 【while后面执行的action一定要加{}，不然无限循环打印1】

执行结果如下：



【注：这里while循环循环体要加{}，不然会不断输出1，所以为了程序不出现错误，循环体统一放在{}中去】

### do-while循环

Do-While 循环与 While 循环相似，但是 Do-While 的条件测试放到了循环的尾部。下面是 do-while 的语法：

do

{

action

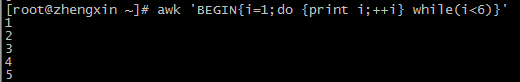
}

while (condition)

【注：在 do-while 循环中，无论条件是真是假，循环语句至少执行一次，执行后检查条件真假；而while循环是先判断条件在执行】

例子：awk 'BEGIN{i=1;do {print i;++i} while(i<6)}'

执行结果如下：



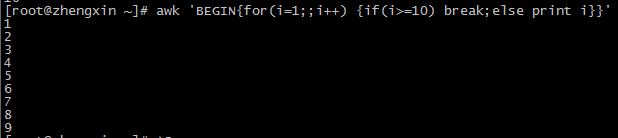
### Break

break用于结束整个循环，同其他编程语言的用法；

例子：打印输出1-9 九个整数；

awk 'BEGIN{for(i=1;;i++) {if(i>=10) break;else print i}}'

执行结果如下：



【当i>=10时，执行break，就跳出整个循环】

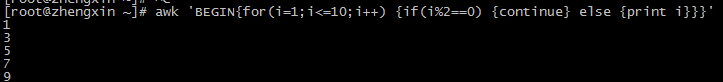
### Continue

continue 语句用于在循环体内部结束本次循环，从而直接进入下一次循环迭代；

例子：打印输出1-10之间的奇数；

awk 'BEGIN{for(i=1;i<=10;i++) {if(i%2==0) continue;else print i}}'

执行结果如下：



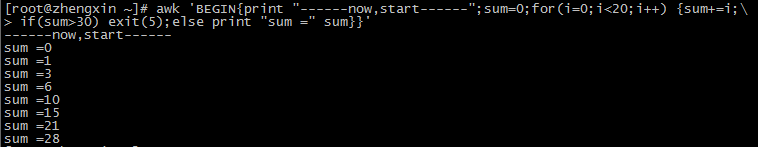
【当i%2==0时，执行continue，就跳出本次循环，进入下一次循环】

### Exit

Exit 用于结束脚本程序的执行。该函数接受一个整数作为参数表示 AWK 进程结束状态。 如果没有提供该参数，其默认状态为 0 ；

例子：awk 'BEGIN{print "------now,start------";sum=0;for(i=0;i<20;i++) {sum+=i;if(sum>30) exit(5);else print "sum =" sum}}'

执行结果如下：



【注：执行完毕后，使用echo $?查看退出码是否是exit()中定义的退出码】

## 十二、内置函数

AWK 为程序开发者提供了丰富的内置函数。这一章节会讲解 AWK 提供的算术函数、字符串操作函数、时间操作相关的函数、位操作函数以及其它各种各样的函数。

### 算数函数

#### int(expr)

此函数返回数值 expr 的整数部；

例子：awk 'BEGIN{a=100.24567;result=int(a);printf "a = %0.5f\nresult = %s\n" ,a,result}'

执行结果如下：

IMG_256

#### Rand()

Rand() 函数返回一个大于等于 0 小于 1 的随机数 N（0<= N < 1）；

例子： awk 'BEGIN{a=rand();b=rand();printf "a = %f\nb = %f\n",a,b}'

执行结果如下：

IMG_256

#### sqrt(expr)

此函数计算 expr 的平方根；

例子：awk 'BEGIN{a=256;printf "sqrt(%f) = %f\n",a,sqrt(a)}'

执行结果如下：

IMG_256

### 字符串函数

#### asort(arr,[, d [,how] ])

asort 函数使用 GAWK 值比较的一般规则排序 arr 中的内容，然后用以 1 开始的有序整数替换排序内容的索引。【asort按ASCII码的顺序排序arr中元素的内容，然后以从1开始的有序整数替换排序内容的索引】

例子：awk 'BEGIN {

arr[0] = "Three"

arr[1] = "One"

arr[2] = "Two"

print "Array elements before sorting:"

for (i in arr) {

printf "arr[%d] = %s\n",i,arr[i]

}

asort(arr)

print "Array elements after sorting:"

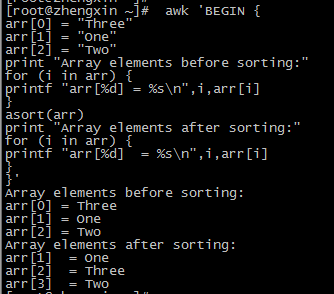
for (i in arr) {

printf "arr[%d] = %s\n",i,arr[i]

}

}'

执行结果如下：



【注：使用asort()函数前后对比，asort()是将arr数组元素的内容先进行排序，按照ASCII码的顺序进行排序，排序完内容后，然后针对内容以从1开始的有序整数排序索引】

#### asorti(arr,[, d [,how] ])

asorti 函数的行为与 asort 函数的行为很相似，二者的差别在于 aosrt 对数组的值排序，而 asorti 对数组的索引排序；【即arr数组中元素的值变成了索引，然后以从1开始的有序整数作为排序索引】

例子：awk 'BEGIN {

arr["The"] = 1

arr["One"] = 2

arr["Two"] = 3

print "Array elements before sorting:"

for (i in arr) {

printf "arr[%s] = %s\n",i,arr[i]

}

asorti(arr)

print "Array elements after sorting:"

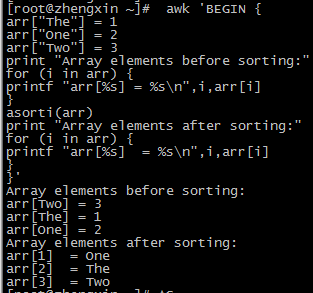
for (i in arr) {

printf "arr[%s] = %s\n",i,arr[i]

}

}'

执行结果如下：



【注：使用asorti()函数后，arr数组元素的值编程原来的索引，然后以1开始的有序整数作为新的索引打印输出】

#### gsub(regx,sub, string)

gsub 是全局替换( global substitution )的缩写。它将出现的子串（regx）替换为 sub。第三个参数 string 是可选的，默认值为 $0，表示在整个输入记录中搜索子串，Regx代表被替换的内容，可以用正则表达式匹配，sub表示要替换成的内容。

例子：awk 'BEGIN {

str = "Hello, World! The World"

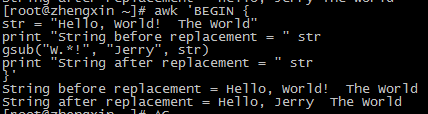
print "String before replacement = " str

gsub("W.\*!", "Jerry", str)

print "String after replacement = " str

}'

执行结果如下：



#### index(str,sub)

index 函数用于检测字符串 sub 是否是 str 的子串。如果 sub 是 str 的子串，则返回子串 sub 在字符串 str 的开始位置；若不是其子串，则返回 0。str 的字符位置索引从 1 开始计数。

例子：awk 'BEGIN{a="this is good";b="good";ret=index(a,b);print ret}'

或者

awk 'BEGIN{a="this is good";b="xx";ret=index(a,b);print ret}'

执行结果如下：

IMG_256

【若b中所有的字符串不在a中，则返回0，否则返回b中最开始字符的索引位置】

#### length(str)

length 函数返回字符串的长度；

例子：awk 'BEGIN{a="abcdref ed";print "length(a) = " length(a)}'

执行结果：

IMG_256

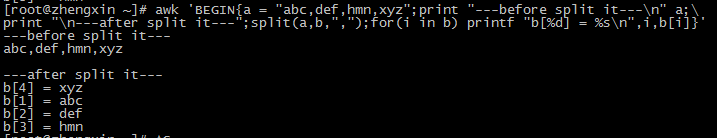
#### split(str, arr,regex)

split 函数使用正则表达式 regex 分割字符串 str，分割后的所有结果存储在数组 arr 中。如果没有指定 regex 则使用 FS 切分。【这里的regex表示分隔符，如果不指定的话按照FS来切割】

例子：awk 'BEGIN{a = "abc,def,hmn,xyz";print "---before split it---\n" a;\

print "\n---after split it---";split(a,b,",");for(i in b) printf "b[%d] = %s\n",i,b[i]}'

执行结果如下：



#### strtonum(str)

strtonum 将字符串 str 转换为数值。 如果字符串以 0 开始，则将其当作十进制数；如果字符串以 0x 或 0X 开始，则将其当作十六进制数；否则，将其当作浮点数。

例子：awk 'BEGIN {

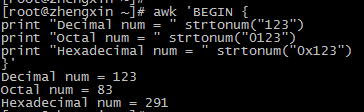
print "Decimal num = " strtonum("123")

print "Octal num = " strtonum("0123")

print "Hexadecimal num = " strtonum("0x123")

}'

执行结果如下：



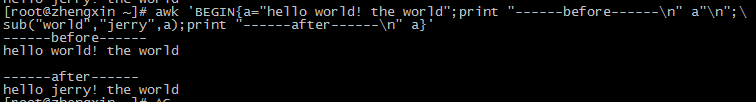
#### sub(regex,sub,string)

sub 函数执行一次子串替换。它将第一次出现的子串regex用 sub 替换。第三个参数是可选的，默认为 $0。

例子：awk 'BEGIN{a="hello world! the world";print "------before------\n" a"\n";\

sub("world","jerry",a);print "------after------\n" a}'

执行结果如下：



#### substr(str, start, length)

substr 函数返回 str 字符串中从第 start 个字符开始长度为 length 的子串。如果没有指定 length的值，返回 str 从第 start个字符开始的后缀子串。

例子：awk 'BEGIN{a="hello world!";b=substr(a,1,5);print b}'

执行结果如下：

IMG_256

【注：返回a字符串中从第一个字符开始，长度为5的所有字符】

#### tolower（str）

此函数将字符串 str 中所有大写字母转换为小写字母然后返回。注意，字符串 str 本身并不被改变。

例子：awk 'BEGIN{a="THIS IS MY NAME!";print tolower(a)}'

执行结果如下：

IMG_256

#### （11）toupper（str）

此函数将字符串 str 中所有小写字母转换为大写字母然后返回。注意，字符串 str 本身不被改变。

例子：awk 'BEGIN{a="this is my name!";print toupper(a)}'

执行结果如下：

IMG_256

### 时间函数

systime

mktime（dataspec）

strftime([format [, timestamp[, utc-flag]]])

### 位操作函数

#### and

执行位与操作【即两个都为真才为真，1为真，0为假】；

例子：awk 'BEGIN{a=9;b=3;printf "(%d and %d) = %d\n",a,b,and(a,b)}'

执行结果如下：

IMG_256

【注：9用二进制表示为00001001，3用二进制表示为00000011，然后执行and操作，相与得出结果为00000001，用十进制表示为1】

#### lshirt(num,step)

左移位操作；

例子：awk 'BEGIN{a=10;printf "lshift(%d)by 1 = %d\n",a,lshift(a,1)}'

执行结果如下：

IMG_256

【注：10用二进制表示为00001010，执行左移操作后变为00010100，十进制表示即变为20】

#### rshirt(num,step)

右移位操作；

例子：awk 'BEGIN{a=10;printf "rshift(%d)by 1 = %d\n",a,rshift(a,1)}'

执行结果如下：

IMG_256

【注：10用二进制表示为00001010，执行右移位操作后变为00000101，十进制表示即变为5】

#### or

按位或操作【即两个都为假才为假，1为真，0为假】；

例子：awk 'BEGIN{a=9;b=3;printf "(%d or %d) = %d\n",a,b,or(a,b)}'

执行结果如下：

IMG_256

【注：9用二进制表示为00001001，3用二进制表示为00000011，然后执行or操作，得出结果为00001011，用十进制表示为11】

#### xor

按位异或运算【即两个不同为真，相同则为假,1为真，0为假】

例子：awk 'BEGIN{a=9;b=3;printf "(%d xor %d) = %d\n",a,b,xor(a,b)}'

执行结果如下：

IMG_256

【注：9用二进制表示为00001001，3用二进制表示为00000011，然后执行xor操作，得出结果为00001010，用十进制表示为10】

### 5、其他函数

#### （1）close(expr)

关闭管道的文件。

例子：awk 'BEGIN {

cmd = "tr [a-z] [A-Z]"

print "hello, world !!!" |& cmd

close(cmd, "to")

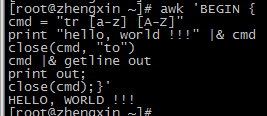
cmd |& getline out

print out;

close(cmd);

}'

执行结果如下：



解释说明：

• 第一条语句 cmd = "tr [a-z] [A-Z]" 在　AWK 中建立了一个双向的通信通道。

• 第二条语句 print 为 tr 命令提供输入。&| 表示双向通信。

• 第三条语句 close(cmd, "to") 完成执行后关闭 to 进程。

• 第四条语句 cmd |& getline out 使用 getline 函数将输出存储到 out 变量中。

• 接下来的输出语句打印输出的内容，最后 close 函数关闭 cmd。

#### （2）delete

delete 被用于从数组中删除元素。下面的例子演示了如何使用 delete：

例子：

awk 'BEGIN {

arr[0] = "One"

arr[1] = "Two"

arr[2] = "Three"

arr[3] = "Four"

print "Array elements before delete operation:"

for (i in arr) {

print arr[i]

}

delete arr[0]

delete arr[1]

print "Array elements after delete operation:"

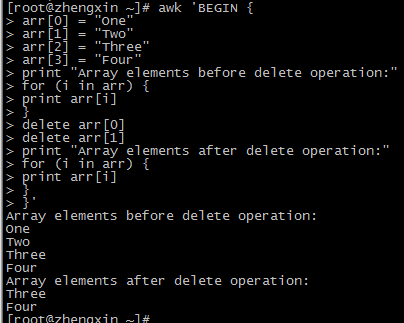
for (i in arr) {

print arr[i]

}

}'

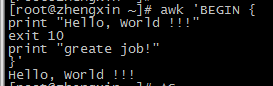
执行结果如下：



#### exit

该函数终止脚本执行。它可以接受可选的参数 expr 传递 AWK 返回状态；

例子：



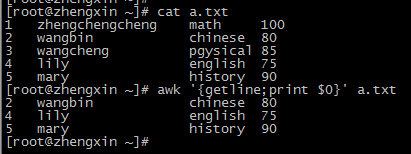
【注：exit语句执行完毕后，直接退出程序，不执行print操作】

#### getline

getline 函数读入下一行；

例子： awk '{getline;print $0}' a.txt

执行结果如下：



脚本看似工作正常，但是第一行去哪儿了呢？让我们理一下整个过程。刚启动时，AWK 从文件 marks.txt 中读入一行存储到变量 $0 中。在下一条语句中，我们使用 getline 读入下一行。 因此 AWK 读入第二行并存储到 $0中。最后，AWK 使用 print 输出第二行的内容。这个过程一直到文件结束。

#### next

next 停止处理当前记录，并且进入到下一条记录的处理过程。下面的例子中，当模式串匹配成功后程序并不执行任何操作；

例子：awk '{if($2~/wang/) next;print $0}' a.txt

执行结果如下：

IMG_256

【当$2中包含wang字符串时，就停止处理当前记录，直接进入下一条记录的处理过程】

#### （6）nextfile

nextfile 停止处理当前文件，从下一个文件第一个记录开始处理。

#### return

return 用于从用户自定义的函数中返回值。请注意，如果没有指定返回值，那么的返回值是未定义的。

例子：

awk 'function add(a,b){c = a + b;return c} BEGIN{res = add(11,20);print "11 + 20 = " res}'

或者

awk 'function add(a,b){c = a + b;return c} BEGIN{print "11 + 20 = ",add(11,20)}'

执行结果如下：

IMG_256

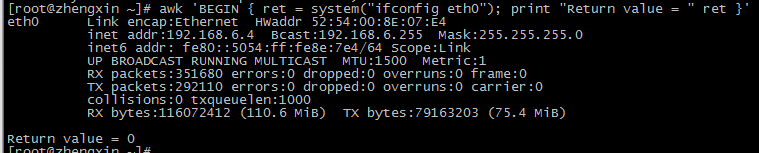
【注：add是用户自定义的函数，可以在后面通过调用】

#### system

system 函数可以执行特定的命令然后返回其退出状态。返回值为 0 表示命令执行成功；非 0 表示命令执行失败。

例子：awk 'BEGIN { ret = system("ifconfig eth0"); print "Return value = " ret }'

执行结果如下：



## 用户自定义函数

函数是程序的基本构造部分。AWK 允许我们自定义函数。事实上，大部分的程序功能都可以被切分成多个函数，这样每个函数可以独立的编写与测试。

下面是用户自定义函数的一般形式【类似于shell】：

function function\_name(argument1, argument2, ...)

{

Function body

}

上述定义函数的语法中：

function\_name 是用户自定义函数的名称。函数名称应该以字母开头，其后可以是数字、字母或下划线的自由组合。

AWK 保留的关键字不能作为用户自定义函数的名称。

自定义函数可以接受多个输入参数，这些参数之间通过逗号分隔。参数并不是必须的。我们也可以定义没有任何输入参数的函数。

function body 是函数体部分，它包含 AWK 程序代码。

例子：比较两个数的大小，打印输出最大值和最小值；

awk 'function MAX\_MIN\_FIND(a,b,c) \

{

max=a;min=a;if(c > max) max=c;if(c < min) min=c;\

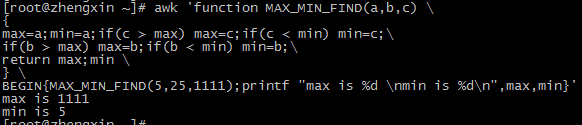
if(b > max) max=b;if(b < min) min=b;\

return max;min \

} \

BEGIN{MAX\_MIN\_FIND(5,25,1111);printf "max is %d \nmin is %d\n",max,min}'

执行结果如下：



【注：把一个数作为标准，然后另外两个数与之对比】

# 附录：习题集锦

知识点：

1）数组

数组是用来存储一系列值的变量，可通过索引来访问数组的值。

Awk中数组称为关联数组，因为它的下标（索引）可以是数字也可以是字符串。

下标通常称为键，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

**1、Nginx日志分析**

    日志格式：'$remote\_addr - $remote\_user [$time\_local] "$request" $status $body\_bytes\_sent "$http\_referer" "$http\_user\_agent" "$http\_x\_forwarded\_for"'

    日志记录：27.189.231.39 - - [09/Apr/2016:17:21:23 +0800] "GET /Public/index/images/icon\_pre.png HTTP/1.1" 200 44668 "http://www.test.com/Public/index/css/global.css" "Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/44.0.2403.157 Safari/537.36" "-"

**1）统计日志中访问最多的10个IP**

        思路：对第一列进行去重，并输出出现的次数

        方法1：$ awk '{a[$1]++}END{for(i in a)print a[i],i|"sort -k1 -nr|head -n10"}' access.log

        方法2：$ awk '{print $1}' access.log |sort |uniq -c |sort -k1 -nr |head -n10

        说明：a[$1]++ 创建数组a，以第一列作为下标，使用运算符++作为数组元素，元素初始值为0。处理一个IP时，下标是IP，元素加1，处理第二个IP时，下标是IP，元素加1，如果这个IP已经存在，则元素再加1，也就是这个IP出现了两次，元素结果是2，以此类推。因此可以实现去重，统计出现次数。

**2）统计日志中访问大于100次的IP**

        方法1：$ awk '{a[$1]++}END{for(i in a){if(a[i]>100)print i,a[i]}}' access.log

        方法2：$ awk '{a[$1]++;if(a[$1]>100){b[$1]++}}END{for(i in b){print i,a[i]}}' access.log

        说明：方法1是将结果保存a数组后，输出时判断符合要求的IP。方法2是将结果保存a数组时，并判断符合要求的IP放到b数组，最后打印b数组的IP。

**3）统计2016年4月9日一天内访问最多的10个IP**

        思路：先过滤出这个时间段的日志，然后去重，统计出现次数

        方法1：$ awk '$4>="[9/Apr/2016:00:00:01" && $4<="[9/Apr/2016:23:59:59" {a[$1]++}END{for(i in a)print a[i],i|"sort -k1 -nr|head -n10"}' access.log

        方法2：$ sed -n '/\[9\/Apr\/2016:00:00:01/,/\[9\/Apr\/2016:23:59:59/p' access.log |sort |uniq -c |sort -k1 -nr |head -n10  #前提开始时间与结束时间日志中必须存在

**4）统计当前时间前一分钟的访问数**

        思路：先获取当前时间前一分钟对应日志格式的时间，再匹配统计

        $ date=$(date -d '-1 minute' +%d/%b/%Y:%H:%M);awk -vdate=$date '$0~date{c++}END{print c}' access.log

        $ date=$(date -d '-1 minute' +%d/%b/%Y:%H:%M);awk -vdate=$date '$4>="["date":00" && $4<="["date":59"{c++}END{print c}' access.log

        $ grep -c $(date -d '-1 minute' +%d/%b/%Y:%H:%M) access.log

        说明：date +%d/%b/%Y:%H:%M --> 09/Apr/2016:01:55

**5）统计访问最多的前10个页面（$request）**

        $ awk '{a[$7]++}END{for(i in a)print a[i],i|"sort -k1 -nr|head -n10"}' access.log

**6）统计每个URL访问内容的总大小（$body\_bytes\_sent）**

        $ awk '{a[$7]++;size[$7]+=$10}END{for(i in a)print a[i],size[i],i}' access.log

**7）统计每个IP访问状态码数量（$status）**

        $ awk '{a[$1" "$9]++}END{for(i in a)print i,a[i]}' access.log

**8）统计访问状态码为404的IP及出现次数**

        $ awk '{if($9~/404/)a[$1" "$9]++}END{for(i in a)print i,a[i]}' access.log

1. 查看某时间段的日志信息；

awk '$2>="09:13" && $2<="10:20"{print $0}' /home/admin/tomcat/logs/mercury.log 【这是表示查看当天时间段在9:13至10:20之间的所有日志信息】

# 附录二、awk如何使用shell变量

有很多种方法，最简单的一种记录如下：

2 line=XXX

3 awk '$1== " '$line' " {print $0}' inputfile

4 注意的是，awk命令本身需要将模式和动作部分用单引号引用，所以里面引用shell变量是双引号+单引号，双引号是保证正确处理变量值的空格，放大一下：

5 " '$line' "

6 另外：

7 awk '$1="win"{$"'$curcls'"=$"'$curcls'"+1;print $0}' nwf

8 则是另一种间接引用模式，因为通常shell的间接引用是使用

9 eval echo \$$var

10 更高级的

11 a=10

12 b=a

13 c=${!b}

14 则c=10

例子如下：脚本awk引用shell变量

cat rjsms\_check.sh

#!/bin/bash

#description: Intercept sms-logs for the last three minutes to check rjsms\_channel...

TIME\_START=$(date -d "-5min" +%H:%M)

TIME\_END=$(date +%H:%M)

awk '$2>"'"$TIME\_START"'" && $2<"'"$TIME\_END"'" {print $0}' /home/admin/sms/logs/message-center.log | grep "save 1 sms SmsRecord" | grep "channelId='OLDRUJIA'" 【注意的是，awk命令本身需要将模式和动作部分用单引号引用，所以里面引用shell变量是双引号+单引号，双引号是保证正确处理变量值的空格，但是因为时间是字符串外面需要再套一层双引号】