# 1 awk介绍

AWK是一个处理行的流编辑器，匹配文件中的信息然后转换成一定的格式而后显示。

他工作在贪婪模式下。会尽可能多的匹配到pattern所匹配到的行。

特性：

## 1 awk存在内置变量

**记录变量：**

FS: field separator，读取文件本时，所使用字段分隔符；

RS: Record separator，输入文本信息所使用的换行符；

OFS: Output Filed Separator: 输出文件时，所使用的字段分隔符

例子：awk ‘BEGIN{OFS=”:”}{print $1,$2,$3,$4}’ test.txt

ORS：Output Row Separator：

**数据变量：**

NR: The number of input records，awk命令所处理的记录数；如果有多个文件，这 个数目会把处理的多个文件中行统一计数；

NF：Number of Field，当前记录的field个数；

FNR: 与NR不同的是，FNR用于记录正处理的行是当前这一文件中被总共处理的 行数；

ARGV: 数组，保存命令行本身这个字符串，如awk '{print $0}' a.txt b.txt这个命令 中，ARGV[0]保存awk，ARGV[1]保存a.txt；

ARGC: awk命令的参数的个数；

FILENAME: awk命令所处理的文件的名称；

ENVIRON：当前shell环境变量及其值的关联数组；

如：awk 'BEGIN{print ENVIRON["PATH"]}'

## 2 用户自定义变量

gawk允许用户自定义自己的变量以便在程序代码中使用，变量名命名规则与大多数编程语言相同，只能使用字母、数字和下划线，且不能以数字开头。gawk变量名称区分字符大小写。

在脚本中赋值变量

awk 'BEGIN{var="variable testing";print var}' **#注意：同一个花括号的两个不同语 #句，要用；隔开**

在命令行中使用赋值变量

awk -v var="variable testing" 'BEGIN{print var}'

## **3 awk的操作符**：

**算术操作符：**

-x: 负值

+x: 转换为数值；

x^y:

x\*\*y: 次方

x\*y: 乘法

x/y：除法

x+y:

x-y:

x%y:

**字符串操作符**：

只有一个，而且不用写出来，用于实现字符串连接；

**赋值操作符：**

=

+=

-=

\*=

/=

%=

^=

\*\*=

++

--

**需要注意的是，如果某模式为=号，此时使用/=/可能会有语法错误，应以/[=]/替代；**

**布尔值**

awk中，任何非0值或非空字符串都为真，反之就为假；

**比较操作符：**

x < y True if x is less than y.

x <= y True if x is less than or equal to y.

x > y True if x is greater than y.

x >= y True if x is greater than or equal to y.

x == y True if x is equal to y.

x != y True if x is not equal to y.

x ~ y True if the string x matches the regexp denoted by y.

x !~ y True if the string x does not match the regexp denoted by y.

subscript in array True if the array array has an element with the subscript subscript.

**表达式间的逻辑关系符：**

&&

||

**条件表达式：**

selector?if-true-exp:if-false-exp

if selector; then

if-true-exp

else

if-false-exp

fi

a=3

b=4

a>b?a is max:b ia max

**函数调用：**

function\_name (para1,para2)

## 4 控制语句：

**if-else**

语法：if (condition) {then-body} else {[ else-body ]}  **#else可以不用**

例子：

awk -F: '{if ($1=="root") print $1, "Admin"; else print $1, "Common User"}' /etc/passwd

awk -F: '{if ($1=="root") printf "%-15s: %s\n", $1,"Admin"; else printf "%-15s: %s\n", $1, "Common User"}' /etc/passwd

awk -F: -v sum=0 '{if ($3>=500) sum++}END{print sum}' /etc/passwd

**while**

**while是对每行每个字段做循环。因为awk本身作行循环**

语法： while (condition){statement1; statment2; ...}

awk -F: '{i=1;while (i<=3) {print $i;i++}}' /etc/passwd

awk -F: '{i=1;while (i<=NF) { if (length($i)>=4) {print $i}; i++ }}' /etc/passwd

**do-while**

语法： do {statement1, statement2, ...} while (condition)

awk -F: '{i=1;do {print $i;i++}while(i<=3)}' /etc/passwd

**for**

语法： for ( variable assignment; condition; iteration process) { statement1, statement2, ...}

awk -F: '{for(i=1;i<=3;i++) print $i}' /etc/passwd

awk -F: '{for(i=1;i<=NF;i++) { if (length($i)>=4) {print $i}}}' /etc/passwd

for循环还可以用来遍历数组元素：

语法： for (i in array) {statement1, statement2, ...}

awk -F: '$NF!~/^$/{BASH[$NF]++}END{for(A in BASH){printf "%15s:%i\n",A,BASH[A]}}' /etc/passwd

**case**

语法：switch (expression) { case VALUE or /REGEXP/: statement1, statement2,... default: statement1, ...}

**break 和 continue**

常用于循环或case语句中

**next**

提前结束对本行文本的处理，并接着处理下一行；例如，下面的命令将显示其ID号为 奇数的用户：

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

## 5 数组

array[index-expression]

index-expression可以使用任意字符串；需要注意的是，如果某数据组元素事先不 存在，那么在引用其时，awk会自动创建此元素并初始化为空串；因此，要判断 某数据组中是否存在某元素，需要使用index in array的方式。

要遍历数组中的每一个元素，需要使用如下的特殊结构：

for (var in array) { statement1, ... }

其中，var用于引用数组下标，而不是元素值；

例子：

netstat -ant | awk '/^tcp/ {++S[$NF]} END {for(a in S) print a, S[a]}'

每出现一被/^tcp/模式匹配到的行，数组S[$NF]就加1，NF为当前匹配到的 行的最后一个字段，此处用其值做为数组S的元素索引；

awk '{counts[$1]++}; END {for(url in counts) print counts[url], url}' /var/log/httpd/access\_log

用法与上一个例子相同，用于统计某日志文件中IP地的访问量

删除数组变量

从关系数组中删除数组索引需要使用delete命令。使用格式为：

delete array[index]

## 6 awk的内置函数

split(string, array [, fieldsep [, seps ] ])

功能：将string表示的字符串以fieldsep为分隔符进行分隔，并将分隔后的结果保 存array为名的数组中；数组下标为从0开始的序列；

netstat -ant | awk '/:80\>/{split($5,clients,":");IP[clients[1]]++}END{for(i in IP){print IP[i],i}}' | sort -rn | head -50

length([string])

功能：返回string字符串中字符的个数；

substr(string, start [, length])

功能：取string字符串中的子串，从start开始，取length个；start从1开始计 数；

system(command)

功能：执行系统command并将结果返回至awk命令

systime()

功能：取系统当前时间

tolower(s)

功能：将s中的所有字母转为小写

toupper(s)

功能：将s中的所有字母转为大写

## 7 用户自定义函数

自定义函数使用function关键字。格式如下：

function F\_NAME([variable])

{

statements

}

函数还可以使用return语句返回值，格式为“return value”

# 2 awk语法

awk [options] ‘[PATTERN]{action1}{action2}……’ file1 file2,……

**options:**

-F 指定输入分隔符

-F: 指定输入分隔符为冒号

-f 指定文件

-v Variable\_NAME 赋值变量

## Pattern

常见的模式类型：

1、Regexp: 正则表达式，格式为/regular expression/

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

2、expresssion： 表达式，其值非0或为非空字符时满足条件，如：$1 ~ /foo/ 或 $1 == "magedu"，用运算符~(匹配)和!~(不匹配)。

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

3、Ranges： 指定的匹配范围，格式为pat1,pat2

awk -F: '/^roo/,/^m/{print $1}' /etc/passwd

4、BEGIN/END：特殊模式，仅在awk命令执行前运行一次或结束前运行一次

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

END：让用户在最后一条输入记录被读取之后发生的动作。

awk -F: ‘{shell[$NF]++}END{for(a in shell) {print a,shell[a]}}’ /etc/passwd

5、Empty(空模式)：匹配任意输入行，即没有PATTERN。

awk -F: ‘{shell[$NF]++}END{for(a in shell) {print a,shell[a]}}’ /etc/passwd

## Action

**注意：同一个花括号的两个不同语句，要用；隔开。**

**例如：awk 'BEGIN{var="variable testing";print var}'**

1、Expressions:

**2、Control statements控制语句**

if/for/while……

3、Compound statements

4、Input statements

getline 不常用

**5、Output statements**

**print**

print的使用格式：

print item1, item2, ... **#这时用逗号隔开！！**

1、各项目之间使用逗号隔开，而输出时则以空白字符分隔；

2、输出的item可以为字符串或数值、当前记录的字段(如$1)、变量或awk的表 达式；数值会先转换为字符串，而后再输出；

3、**print命令后面的item可以省略，此时其功能相当于print $0, 因此，如果想输 出空白行，则需要使用print ""；**

例子：

# awk 'BEGIN { print "line one\nline two\nline three" }'

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

输出重定向：

print items > output-file

print items >> output-file

print items | command

特殊文件描述符：

/dev/stdin：标准输入

/dev/sdtout: 标准输出

/dev/stderr: 错误输出

/dev/fd/N: 某特定文件描述符，如/dev/stdin就相当于/dev/fd/0；

例子：

# awk -F: '{printf "%-15s %i\n",$1,$3 > "/dev/stderr" }' /etc/passwd

**printf**

printf命令的使用格式：

printf format, item1, item2, ...

要点：

1、其与print命令的最大不同是，printf需要指定format；

2、format用于指定后面的每个item的输出格式；

3、printf语句不会自动打印换行符；\n

format格式的指示符都以%开头，后跟一个字符；如下：

%c: 显示字符的ASCII码；

%d, %i：十进制整数；

%e, %E：科学计数法显示数值；

%f: 显示浮点数；

%g, %G: 以科学计数法的格式或浮点数的格式显示数值；

%s: 显示字符串；

%u: 无符号整数；

%%: 显示%自身；

修饰符：

N: 显示宽度；

-: 左对齐；

+：显示数值符号；

例子：

# awk -F: '{printf "%-15s%i\n",$1,$3}' /etc/passwd