AWK

AWK是一种优良的文本处理工具。它不仅是 [Linux](http://baike.baidu.com/view/1634.htm) 中也是任何环境中现有的功能最强大的数据处理引擎之一。这种编程及[数据操作语言](http://baike.baidu.com/view/3426309.htm" \t "_blank)（其名称得自于它的创始人 Alfred Aho 、Peter Weinberger 和 Brian Kernighan 姓氏的首个字母）的最大功能取决于一个人所拥有的知识。AWK 提供了极其强大的功能：可以进行样式装入、[流控制](http://baike.baidu.com/view/1292763.htm" \t "_blank)、数学[运算符](http://baike.baidu.com/view/425996.htm)、进程[控制语句](http://baike.baidu.com/view/1359886.htm)甚至于内置的[变量](http://baike.baidu.com/view/296689.htm)和函数。它具备了一个完整的语言所应具有的几乎所有精美特性。实际上 AWK 的确拥有自己的语言：AWK [程序设计语言](http://baike.baidu.com/view/128511.htm)， 三位创建者已将它正式定义为“样式扫描和处理语言”。它允许您创建简短的程序，这些程序读取输入文件、为[数据排序](http://baike.baidu.com/view/2586460.htm" \t "_blank)、处理数据、对输入执行计算以及生成报表，还有无数其他的功能。

最简单地说， AWK 是一种用于处理文本的编程语言工具。AWK 在很多方面类似于 shell 编程语言，尽管 AWK 具有完全属于其本身的语法。它的设计思想来源于 SNOBOL4 、sed 、Marc Rochkind设计的有效性语言、语言工具 yacc 和 lex ，当然还从 C 语言中获取了一些优秀的思想。在最初创造 AWK 时，其目的是用于文本处理，并且这种语言的基础是，只要在输入数据中有模式匹配，就执行一系列指令。该实用工具扫描文件中的每一行，查找与命令行中所给定内容相匹配的模式。如果发现匹配内容，则进行下一个编程步骤。如果找不到匹配内容，则继续处理下一行。

我们常用sed和awk相结合的方式处理文本sed|awk的方式能让我们的处理命令更快捷,提高工作效率

**awk语法**

awk [options]‘commands’files

**option**

**-F** 定义字段分隔符**,**默认的分隔符是连续的空格或制表符

如:

aaaa bbbb

aaa bb

ccc dd

awk ‘{print $1}’a.txt

awk –F: ‘{print $1}’/etc/passwd

用**$1,$2,$3**等的顺序表示**files**中每行以间隔符号分隔的各列不同域.$0代表整行

如:

awk ‘{print $1,$2,$0}’a.txt

**NF**变量表示当前记录的字段数

如:

awk –F: ‘{print NF}’/etc/passwd

awk –F: ‘{print $NF}’/etc/passwd

**-v**可以借用次方式从**shell**变量中引入

**command**

读前处理 行处理 读后处理

1. 读前处理**BEGIN{awk\_cmd1;awk\_cmd2}**

如:

awk 'BEGIN {print "hello,World"}'

awk 'BEGIN{print "start handle file"}{print}END{print"stop handle file"}' /etc/passwd

1. 行处理**:**定址命令

定址方法**:** 正则**,**变量**,**比较和关系运算

正则需要用**//**包围起来

**awk –F: '/root/{print}' /etc/passwd**

**awk –F: '$1~/root/{print}' /etc/passwd**

**awk –F: '$1!~/root/{print}' /etc/passwd**

**^** 行首

**$** 行尾

**.** 除了换行符以外的任意单个字符

**\*** 前导字符的零个或多个

**.\*** 所有字符

**[]** 字符组内的任一字符

**[^]**对字符组内的每个字符取反**(**不匹配字符组内的每个字符**)**

**^[^]** 非字符组内的字符开头的行

**[a-z]** 小写字母

**[A-Z]** 大写字母

**[a-Z]** 小写和大写字母

**[0-9]** 数字

**\<** 单词头单词一般以空格或特殊字符做分隔**,**连续的字符串被当做单词

**\>** 单词尾

**NR**变量定址

**NR** 表示**AWK**读入的行数

**FNR**表示读入行所在文件中的行数

**# awk '{print NR,FNR,$1}' file1 file2**

**1 1 aaaaa**

**2 2 bbbbb**

**3 3 ccccc**

**4 1 dddddd**

**5 2 eeeeee**

**6 3 ffffff**

逻辑运算可直接引用域进行运算

**== >= <= != > <**

**# awk 'NR==1 {print}' /etc/passwd**

**root:x:0:0:root:/root:/bin/bash**

**3.**命令**{print $0}**

**4.**读后处理**END {awk\_cmd1;awk\_cmd2;}**

例子:

cat 2.txt | head -2

root:x:0:0:root:/root:/bin/bash

bin:x:1:1:bin:/bin:/sbin/nologin

awk –F”:”‘{print $1,$6}’2.txt | head -2

root /root

bin /bin

awk –F“:” ‘{print $1”\t\t”$6}’ 2.txt|head -2

root /root

bin /bin

awk –F”:” ‘BEGIN{print “用户名\t\t家目录”}{print $1”\t\t”$6}’2.txt| head -2

例2:cat 1.txt

mona 70 77 85 83 70 89

john 85 92 78 94 88 91

andrea 89 90 85 94 90 95

jasper 84 88 80 92 84 82

dunce 6 80 60 60 61 62

ellis 5 98 89 96 96 92

awk ‘{print $0}’1.txt

awk ‘$2~/6/{print $0}’1.txt

awk ‘$2>=6{print $0}’1.txt

**操作符**

赋值**= ++ -- += -= /= \*=**

**++ Add 1 to variable.**

awk 'BEGIN{a=1;a++;print a}' 打印值为2

awk 'BEGIN{a=1;++a;print a}' 打印值为2 先加1在执行命令

awk 'BEGIN{a=1;print a++;print a}' 先执行命令然后a+1

a=1命令先执行a还=1,最后在打印a=2

**-- Subtract 1 from variable.** awk '

BEGIN{a=1;a--;print a}' 打印值为0

**+= Assign result of addition.**

awk 'BEGIN{a=1;b=a+5;print b}' 给b赋值

awk 'BEGIN{a=1;a=a+5;print a}' 给自己赋值

awk 'BEGIN{a=1;a+=5;print a}' 上条命令简化写法

**-= Assign result of subtraction.**

awk 'BEGIN{a=1;a-=5;print a}' a-5在赋值给a 以下相同

awk 'BEGIN{a=1;a-=5.2;print a}' 支持小数运算

**\*= Assign result of multiplication.**

**/= Assign result of division.**

**%= Assign result of modulo.**

**^= Assign result of exponentiation**

|| Logical OR 逻辑 两边任意一边成立

&& Logical AND 逻辑与 两边成立 前边如果假值后边就不做了

! Logical NOT 逻辑取反 原本真值变成假值

例如:

awk '$1~/root/||NR>40{print}' /etc/passwd

awk '$1~/root/&&NR<2{print}' /etc/passwd

匹配正则或不匹配**,**正则需要用**/**正则**/** 包围住

**~ !~**

关系比较字符串时要把字符串用双引号引起来

**< <= > >= != ==**

字段引用

**$** 字段引用需要加**$,**而变量引用直接用变量名取

运算符**+ - \* / % ++ --**

转义序列

**\\ \**自身

**\$** 转义**$**

**\t** 制表符

**\b** 退格符

**\r** 回车符

**\n** 换行符

**\c** 取消换行

**awk**引用**shell**变量

a=root

awk –v var=$a –F“:” ’$1==$var{print $0}’ /etc/passwd

**内置变量**

FS 定义字段分隔符,默认为一个空格

OFS 输出的字段分隔符，默认为一个空格

awk 'BEGIN{ FS=":";OFS="-" }{ print $1,$3,$5 }' /etc/passwd | head -2

awk 'BEGIN{ FS=":";OFS="-" }{ print $1”@@”$3”##”$5 }' /etc/passwd | head -2 自己定义

RS 记录分隔符，默认为一个换行符

head -2 /etc/passwd | awk 'BEGIN{ RS=":"}{print}'

ORS 输出的记录分隔符，默认为一个换行符

head -2 /etc/passwd | awk 'BEGIN{ ORS="-"}{print}'

NR 行数(如两个文件一个5行 1个10行 输出为15行)

awk '{print NR}' aa.txt cc.txt

FNR 行数，多文件操作时会重新排序(如两个文件一个5行,一个10行 输出为5 10) awk '{print FNR}' aa.txt cc.txt

NF 输出当前输入记录的编号（字段的个数）

aaaa bbb ccc

cccc bbb

aaa ccc bb nn

awk '{ print NF}' aa.txt

FILENAME 文件名

**ARGC** 命令行参数个数

**ARGV** 命令行参数排列

**ENVIRON** 输出系统环境变量

例子:

cat awk.txt

aaa#123

asdf:234

haha:456

awk –F’#’ ’{print NR,$0,NF,FILENAME }’awk.txt

1 aaaa#123 2 awk.sh

2 asdf:234 1 awk.sh

3 haha:456 1 awk.sh

4 0 awk.sh

awk -F"#" '{ print ARGC,ARGV[0],ARGV[1]}' awk.sh.txt

2 awk awk.sh.txt

2 awk awk.sh.txt

2 awk awk.sh.txt

2 awk awk.sh.txt

awk -F"#" '{ print NR,$0,NF,$1,$2 }' awk.sh.txt

1 aaaa#123 2 aaaa 123

2 asdf:234 1 asdf:234

3 haha:456 1 haha:456

4 0

练习题:

**1**打印**uid**在**30~40**范围内的用户名

**2**打印第**5-10**行的行号和用户名

**3** 打印奇数行

**4** 打印偶数行

**5** 打印字段数大于**5** 的行

**6**打印**UID**不等于**GID**的用户名

**7**打印**1..100**以内的**7**的倍数和包含**7**的数

8计算UID相加的总和;计算GID相加的总和

9. 计算VSZ和RSS各自的和 并以M单位显示

**流程控制:**

**分支结构**

1. **if (条件) 动作**

若有多个动作**,**则要用大括号将动作体包含起来**if (**条件**) {**动作**1;**动作**2}**

# awk -F : '{if ($1 == "root") print $1}' /etc/passwd

# awk -F: '{if ($1 == "root") {print $1;print $6} }' /etc/passwd

1. **if(条件1)**

**动作1**

**else**

**动作2**

# awk -F: '{if ($1 == "root"){print $1}else print $6}' /etc/passwd

# awk -F: '{if ($1 == "root") print $1;else print $6}' /etc/passwd

上面两个命令是等价的,要么用分号隔开,表示第一个动作体的结束,要么将动作体用大括号定位范围

三.

**if (条件1)**

**动作1**

**else if(条件2)**

**动作2**

**else if(条件3)**

**动作3**

**else**

**动作**

awk -F: '{if ($1 == "root") print $1;else if ($1 == "seker") print $6;else if ($1 == "zorro") print $7;else printNR}' /etc/passwd

读前处理和读后处理

# awk -F: 'BEGIN {print NR,NF}' /etc/passwd

0 0

# awk -F: 'END {print NR,NF}' /etc/passwd

46 7

# awk -F: 'BEGIN{i=1} {i++} END {print i}' /etc/passwd

47

练习:找出普通用户的用户名并统计数量

将系统用户按UID分组标记 0 admin; 1-499 sysuser; 500+ users

**循环语句**

**while(条件) {**

**动作**

**条件运算**

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

**BEGIN**块可以独立使用**,**不需要引入文件

# awk 'BEGIN{i=1;while(i<100) {print i;i++}}'

练习 打印**100**以内的偶数

**x=1**

**do {**

**动作1**

**x++**

**} while (x<5)**

# awk 'BEGIN{i=5;do{print i;i++}while(i<10)}'

# awk 'BEGIN{i=5;do{print i;i++}while(i<1)}'

for(预置;条件;递增) {

动作

}

# awk 'BEGIN {for (x=1;x<=4;x++) print x }'

练习:

逆序输出每个字段

达到这样既可

/bin/bash

/root

root

0

0

x

root

跳转语句

break 跳出循环

# awk 'BEGIN {for(x=1;x<5;x++) {if (x==3) break;print x }}'

1

2

continue 在达到循环底部之前终止当前循环从新开始下一次循环

# awk 'BEGIN {for(x=1;x<5;x++) {if (x==3) continue;print x }}'

1

2

4

next 读入下一行同时返回脚本顶部这样可以避免对当前行执行其他操作

# awk -F: 'NR > 5 {next} {print $1} END {print NR}' /etc/passwd

root

bin

daemon

adm

lp

cat -b u.txt

1 root:x:0:0:root:/root:/bin/bash

2 bin:x:1:1:bin:/bin:/sbin/nologin

3 daemon:x:2:2:daemon:/sbin:/sbin/nologin

4 adm:x:3:4:adm:/var/adm:/sbin/nologin

5 lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin

6 sync:x:5:0:sync:/sbin:/bin/sync

7 shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown

8 halt:x:7:0:halt:/sbin:/sbin/halt

9 mail:x:8:12:mail:/var/spool/mail:/sbin/nologin

10 news:x:9:13:news:/etc/news:

awk -F: 'BEGIN{i=1}NR>5 {next}{print i++}END{print NR}' u.txt

1

2

3

4

5

10

awk -F: 'BEGIN{i=1}NR>5 {exit}{print i++}END{print NR}' u.txt

1

2

3

4

5

数组

自定义数组

# awk 'BEGIN {ary[1]="seker";ary[2]="zorro";print ary[1],ary[2]}'

seker zorro

# awk 'BEGIN {ary[1]="seker";ary[2]="zorro";for(i in ary) print ary[i]}'

seker

zorro

循环产生数组和取出数组

# awk 'BEGIN{n=5;for (i=1;i<=n;i++) ary[i]=i+100;for(m in ary) print m,ary[m]}'

4 104

5 105

1 101

2 102

3 103

awk -F: '{ary[NR]=$1} END {for(i in ary) print i,ary[i]}' /etc/passwd

1 root

2 bin

3 daemon

4 adm

5 lp

6 sync

7 shutdown

8 halt

9 mail

awk -F: '{ary[$3]=$1} END {for(i in ary) print i,ary[i]}' /etc/passwd

10 uucp

11 operator

12 games

13 gopher

14 ftp

32 rpc

37 rpm

练习1答案

1. awk -F: '$3>=30&&$3<=40 {print NR, $1}' passwd
2. awk -F: 'NR>=5&&NR<=10 {print NR, $1}' passwd
3. awk -F: 'NR%2!=0 {print}' passwd
4. awk -F: 'NR%2==0 {print}' passwd
5. awk -F: 'NF>5 {print NR}' passwd
6. awk -F: '$3!=$4 {print $3,$4}' passwd
7. **s**eq 100 | awk '$0%7==0 || $0 ~/7/ {print $0}' | head -3
8. awk -F: 'BEGIN{i=0}{sum+=$3;gsum+=$4;i++}END{print i;print sum;print gsum}' /etc/passwd
9. ps aux | awk 'BEGIN{i=0}NR>=2{vsum+=$5;rsum+=$6;i++}END{print vsum/1024"M";print rsum/1024"M";print i}'

练习2:

练习找出普通用户的用户名并统计数量

awk -F: 'BEGIN{i=0} $3>=500 {print $1;i++} END {print i}' /etc/passwd

awk -F: '{if($3==0) print $1"\t"$3"\t""admin";else if($3>=1&&$3<500) print $1,$3,"sysuser";else print $1,$3,"user"}' /etc/passwd

练习打印100以内的偶数

# awk 'BEGIN{i=1;while(i<100) {if (i%2==0) print i;i++}}'

逆序输出每个字段

达到这样既可

/bin/bash

/root

root

0

0

x

root

# awk -F: '{for (x=NF;x>0;x--) print $x}' /etc/passwd

输出样式

%s是字符类型,%d数值类型

printf默认是不输出换行的所以要加\n

10和7是偏移量

默认是右对齐,所有加个- 就是左对齐,就是把不足的位数用空格填充

注意:格式与输出列之间要有逗号

# awk -F: '{printf "%-10s %-10d %s\n",$1,$3,$7}' /etc/passwd

继续解决上一个试做题的格式问题

# awk -F: '/bash$/{for (x=NF;x>0;x--) printf "%-13s",$x;printf "\n"}' /etc/passwd

使用嵌套的for循环,打印100-999之间的数,个十百位分别用一个for来打印

# awk 'BEGIN{OFS="";for (i=1;i<=9;i++) {for (j=0;j<=9;j++) {for (n=0;n<=9;n++) print i,j,n}}}'

打印乘法口诀表

# cat 99.sh

#!/bin/bash

awk 'BEGIN{

for(i=1;i<10;i++)

{

for(j=1;j<=i;j++)

printf "%d\*%d=%d ",j,i,j\*i

print

}

}'

#

打印金字塔

# cat jin.sh

#!/bin/bash

awk 'BEGIN{

num=5

for(i=1;i<=num;i++)

{

for (n=1;n<=num-i;n++)

printf "%s"," "

for (j=1;j<=2\*i-1;j++)

printf "%s","\*"

print

}

}'