AWK常用技巧

惨绿少年 Linux运维, Shell编程, 运维基本功 0评论 来源：本站原创 62℃ 字体： 小 中 大

1.1 介绍

awk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。实际上 AWK 的确拥有自己的语言： AWK 程序设计语言 ， 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、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。

awk 是一种很棒的语言，它适合文本处理和报表生成，其语法较为常见，借鉴了某些语言的一些精华，如 C 语言等。在 linux 系统日常处理工作中，发挥很重要的作用，掌握了 awk将会使你的工作变的高大上。

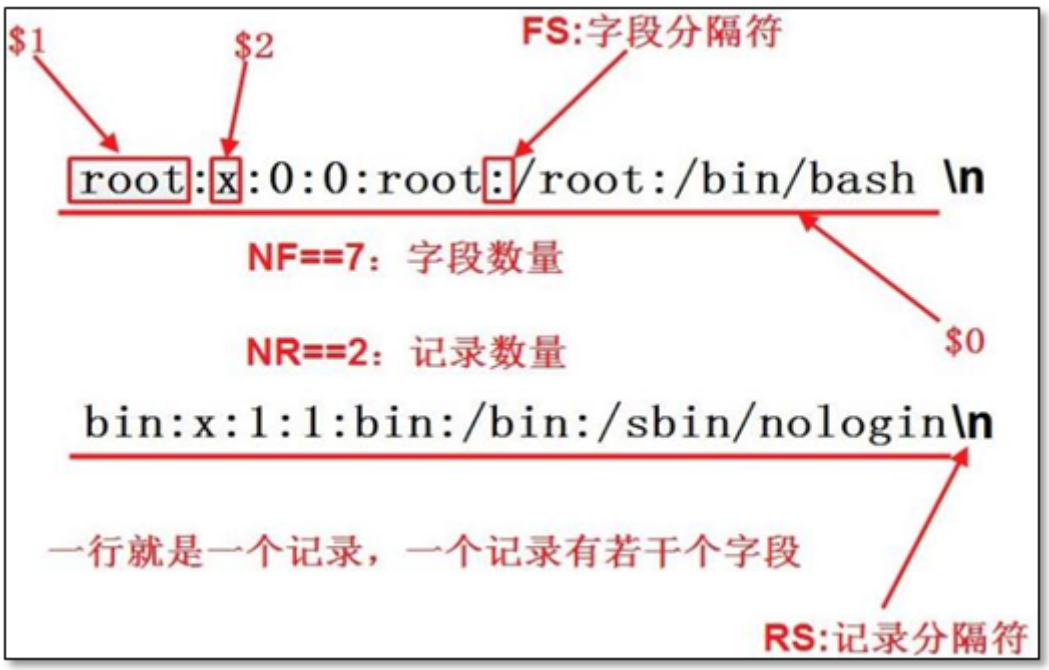
1.1.1 AWK原理

这需要一个例子来说明，你将会见到/etc/passwd 文件的内容出现在眼前。

```
awk '{print $0}' /etc/passwd
echo clsn|awk '{print "hello,world"}'
awk '{ print "clsn" }' /etc/passwd
```

现在，解释 awk 做了些什么。调用 awk时，我们指定/etc/passwd 作为输入文件。执行 awk 时，它依次对/etc/passwd 中的每一行执行 print 命令。所有输出都发送到 stdout，所得到的结果与执行 cat /etc/passwd 完全相同。现在，解释{ print }代码块。在awk 中，花括号用于将几块代码组合到一起，这一点类似于C 语言。在代码块中只有一条 print 命令。在awk 中，如果只出现 print 命令，那么将打印当前行的全部内容。

再次说明，awk 对输入文件中的每一行都执行这个脚本。



1.2 AWK常用速查表

1.2.1 AWK运算符

运算符	说明
赋值运算符	
= += -= *= /= %= ^= **=	赋值语句
逻辑运算符	
||	逻辑或
&&	逻辑与
正则运算符	
~ !~	匹配正则表达式和不匹配正则表达式
关系运算符	
< <= > >= != ==	关系运算符
算术运算符	
+ –	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
其他运算符	
$	字段引用
空格	字符串链接符
?:	三目运算符
In	数组中是否存在某键值

1.2.2 常用AWK内置变量

变量名	属性

$0	当前记录
$1~$n	当前记录的第 n 个字段
FS	输入字段分隔符 默认是空格
RS	输入记录分割符 默认为换行符
NF	当前记录中的字段个数，就是有多少列
NR	已经读出的记录数，就是行号，从 1 开始
OFS	输出字段分隔符 默认也是空格
ORS	输出的记录分隔符 默认为换行符

1.2.3 awk中的正则

元字符	功能	示例	解释
^	行首定位符	/^root/	匹配所有以 root 开头的行
$	行尾定位符	/root$/	匹配所有以 root 结尾的行
.	匹配任意单个字符	/r..t/	匹配字母 r,然后两个任意字符，再以 l 结尾的行，比如 root,r33l 等
*****	匹配 0 个或多个前导字符 (包括回车)	/a*ool/	匹配 0 个或多个 a 之后紧跟着 ool 的行，比如 ool，aaaaool 等
+	匹配 1 个或多个前导字符	/a+b/	匹配 1 个或多个 a 加 b 的行，比如 ab,aab 等
?	匹配 0 个或 1 个前导字符	/a?b/	匹配 b 或 ab 的行

[]	匹配指定字符组内的任意一个字符	/^[abc]	匹配以字母 a 或b 或 c 开头的行
[^]	匹配不在指定字符组内任意一个字符	/^[^abc]/	匹配不以字母 a 或 b 或 c 开头的行
()	子表达式组合	/(rool)+/	表示一个或多个 rool 组合，当有一些字符需要组合时，使用括号括起来
|	或者的意思	/(root)|B/	匹配root 或者 B 的行
\	转义字符	/a\W/	匹配 a//
~,!~	匹配，不匹配的条件语句	$1~/root/	匹配第一个字段包含字符root 的所有记录
x{m}	x 重复m 次	/(root){3}/	需要注意一点的是，root 加括号和不
x{m,}	x 重复至少m 次	/(root){3,}/	加括号的区别，x 可以表示字符串也
X{m,n}	x 重复至少 m 次， 但不超过 n 次	/(root){5,6}/	可以只是一个字符，所以/root\{5\}/ 表示匹配roo 再加上5 个t，及roottttt
	需要指定参数: - posix 或者	cat rex.txt smierth,harry	/\(root\)\{2,\}/ 则表示匹配 rootrootrootroot 等

	–re-interval 没有该参数不能使用该模式	smierth,reru robin,tom	awk -posix '/er\{1,2\}/' rex.text smierth,harry smierth,reru

1.2.4 awk 常用函数表

函数	说明
gsub(Ere, Repl, [In])	除了正则表达式所有具体值被替代这点，它和sub 函数完全一样地执行，。
sub(Ere, Repl, [In])	用 Repl 参数指定的字符串替换 In 参数指定的字符串中的由 Ere参数指定的扩展正则表达式的第一个具体值。sub 函数返回替换的数量。出现在Repl 参数指定的字符串中的 &（和符号）由 In 参数指定的与 Ere 参数的指定的扩展正则表达式匹配的字符串替换。如果未指定 In 参数，缺省值是整个记录（$0 记录变量）。
index(String1, String2)	在由 String1 参数指定的字符串（其中有出现String2 指定的参数）中，返回位置，从 1 开始编号。如果 String2 参数不在 String1 参数中出现，则返回 0（零）。
length [(String)]	返回 String 参数指定的字符串的长度（字符形式）。如果未给出 String 参数，则返回整个记录的长度（$0 记录变量）。
blength [(String)]	返回 String 参数指定的字符串的长度（以字节为单位）。如果未给出 String 参数，则返回整个记录的长度（$0 记录变量）。
	返回具有 N 参数指定的字符数量子串。子串从String 参数指定的字符串取得，其字

substr(String, M, [N])	符以 M 参数指定的位置开始。M 参数指定为将 String 参数中的第一个字符作为编号 1。如果未指定 N 参数，则子串的长度将是 M 参数指定的位置到 String 参数的末尾 的长度。
match(String, Ere)	在 String 参数指定的字符串（Ere 参数指定的扩展正则表达式出现在其中）中返回位置（字符形式），从 1 开始编号，或如果 Ere 参数不出现，则返回 0（零）。RSTART 特殊变量设置为返回值。RLENGTH特殊变量设置为匹配的字符串的长度，或如果未找到任何匹配，则设置为-1（负一）。
split(String, A, [Ere])	将 String 参数指定的参数分割为数组元素 A[1], A[2], . . ., A[n]，并返回 n 变量的值。此分隔可以通过 Ere 参数指定的扩展正则表达式进行，或用当前字段分隔符（FS 特殊变量）来进行（如果没有给出 Ere参数）。除非上下文指明特定的元素还应具有一个数字值，否则 A 数

1.3 AWK实践

1.3.1 函数的简单使用

```
[root@clsn6 ~]# awk '{gsub(/[0-9]+/,"");print}' /tmp/passwd
root:x:::root:/root:/bin/bash
bin:x:::bin:/bin:/sbin/nologin
daemon:x:::daemon:/sbin:/sbin/nologin
adm:x:::adm:/var/adm:/sbin/nologin
lp:x:::lp:/var/spool/lpd:/sbin/nologin
sync:x:::sync:/sbin:/bin/sync
...
```

1.3.2 统计系统中的secure文件中谁在破解你的密码

1、找到谁在破解密码

```
[root@clsn6 awk]# awk '/Failed/{print $(NF-3)}' secure-20161219 |sort |uniq -c  |sort -nk1
      1 103.237.144.68
      1 122.228.238.66
```

```
      1 195.20.3.210
      1 85.93.5.71
...
```

2、过滤出包含Failed password的行 并统计每个ip地址出现的次数

```
[root@clsn6 awk]# awk '/Failed/{fa[$(NF-3)]++}END{for(pol in fa)print pol,fa[pol]}' secure-20161
218.65.30.25 68652
218.65.30.53 34326
218.87.109.154 21201
112.85.42.103 18065
112.85.42.99 17164
218.87.109.151 17163
```

1.3.3 统计secure文件中每个用户，被同一ip破解多少次

1、进破解使用的用户及地址定向到文件中

```
[root@clsn6 awk]# awk '/Failed/{print $(NF-5),$(NF-3)}' secure-20161219 >user-ip.log
```

2、对文件进行去重排序测

```
[root@clsn6 awk]# awk '{h[$1" "$2]++}END{for(p in h) print p,h[p]}' user-ip.log |head
export 209.126.122.70 3
cvsadmin 209.126.122.70 3
user1 209.126.122.70 1
dasusr1 209.126.122.70 4
1 118.100.251.170 1
git 209.126.122.70 5
boss 195.154.50.61 1
user1 46.139.219.84 2
www 123.31.34.165 2
webmaster 195.154.50.61 1
```

3、操作日志文件，取出数据

```
[root@clsn6 awk]# awk '/Failed/{h[$(NF-5)" "$(NF-3)]++}END{for(p in h) print p,h[p]}'  secure-20
export 209.126.122.70 3
cvsadmin 209.126.122.70 3
user1 209.126.122.70 1
dasusr1 209.126.122.70 4
1 118.100.251.170 1
```

1.3.4 统计nginx access.log文件中对ip地址去重并统计重复数

```
[root@clsn6 awk]# awk '{h[$1]++}END{for(i in h) print i,h[i]}' access.log |sort -nrk2 |head
58.220.223.62 12049
112.64.171.98 10856
114.83.184.139 1982
```

1.3.5 统计access.log文件中每个ip地址使用了多少流量

1、流量总计

```
[root@clsn6 awk]# awk '{sum=sum+$10}END{print sum}' access.log
2478496663
[root@clsn6 awk]# awk '{sum=sum+$10}END{print sum/1024^3}' access.log
2.30828
```

2、每个ip使用的流量

```
[root@clsn6 awk]# awk '{h[$1]=h[$1]+$10}END{for(p in h) print p,h[p]/1024^2 }' access.log |sort
114.83.184.139   29.91
117.136.66.10    21.3922
116.216.30.47    20.4716
223.104.5.197    20.4705
116.216.0.60     18.2584
```

1.3.6 统计access.log文件中每个ip地址使用了多少流量和每个ip地址的出现次数

```
[root@clsn6 awk]# awk '{count[$1]++;sum[$1]=sum[$1]+$10}END{for(pol in sum)print pol,count[pol],
58.220.223.62    12049  12603075
112.64.171.98    10856  15255013
114.83.184.139   1982   31362956
117.136.66.10    1662   22431302
115.29.245.13    1318   1161468
223.104.5.197    961    21464856
116.216.0.60     957    19145329
```

格式化命令

```
awk '{
  count[$1]++
  sum[$1]+=$10
}END{
  for(pol in sum)
  print pol,count[pol],sum[pol]
}' access.log |sort -nrk3 |head
```

格式化后执行

```
[root@clsn6 awk]# awk '{
  count[$1]++
  sum[$1]+=$10
}END{
  for(pol in sum)
  print pol,count[pol],sum[pol]
}' access.log |column -t|  sort -nrk3 |head
114.83.184.139   1982   31362956
117.136.66.10    1662   22431302
116.216.30.47    506    21466000
223.104.5.197    961    21464856
```

```
116.216.0.60     957    19145329
114.141.164.180  695    17219553
114.111.166.22   753    17121524
223.104.5.202    871    16911512
116.228.21.187   596    15969887
112.64.171.98    10856  15255013
```

1.3.7 按要求得到最后面的格式的结果

文件内容

```
cat >next2.txg<< EOF
web01[192.168.2.100]
httpd            ok
tomcat            ok
sendmail          ok
web02[192.168.2.101]
httpd            ok
postfix           ok
web03[192.168.2.102]
mysqld           ok
httpd            ok
EOF
```

想要的结果:

```
web01[192.168.2.100] httpd           ok
web01[192.168.2.100] tomcat          ok
web01[192.168.2.100] sendmail         ok
web02[192.168.2.101] httpd           ok
web02[192.168.2.101] postfix          ok
web03[192.168.2.102] mysqld          ok
web03[192.168.2.102] httpd           ok
```

方法一:

```
[root@clsn6 awk]# awk '/^web/{tmp=$0} !/^web/{print tmp,$0}' next.txt
web01[192.168.2.100] httpd           ok
web01[192.168.2.100] tomcat          ok
web01[192.168.2.100] sendmail        ok
web02[192.168.2.101] httpd           ok
web02[192.168.2.101] postfix         ok
web03[192.168.2.102] mysqld          ok
web03[192.168.2.102] httpd           ok
```

方法二:

```
[root@clsn6 awk]# awk '/^web/{tmp=$0;next}{print tmp,$0}' next.txt
web01[192.168.2.100] httpd           ok
web01[192.168.2.100] tomcat          ok
web01[192.168.2.100] sendmail        ok
web02[192.168.2.101] httpd           ok
web02[192.168.2.101] postfix         ok
web03[192.168.2.102] mysqld          ok
web03[192.168.2.102] httpd           ok
```

next:停止处理当前行，从头开始处理下一行

```
[root@clsn6 awk]# seq 5 |awk 'NR==3{next}{print NR,$0}'
1 1
2 2
4 4
5 5
[root@clsn6 awk]# seq 5 |awk 'NR==1{next}{print NR,$0}'
2 2
3 3
4 4
5 5
```

跳过偶数行

```
[root@clsn6 awk]# seq 5 |awk 'NR%2==0{next}{print NR,$0}'
1 1
3 3
5 5
```

跳过奇数行

```
[root@clsn6 awk]# seq 5 |awk 'NR%2==1{next}{print NR,$0}'
2 2
4 4
```

1.3.8 统计每个学生的总成绩和平均成绩

成绩文件

```
cat > chengji.txt <<EOF
cc 90 98 98 96 96 92
ll 70 77 85 83 70 89
ss 85 92 78 94 88 91
nn 89 90 85 94 90 95
bb 84 88 80 92 84 82
gg 64 80 60 60 61 62
EOF
```

算出总和

```
[root@clsn6 awk]#  awk '{sum=0;for(i=2;i<=NF;i++)sum+=$i;print $1,sum}' chengji.txt
cc 570
ll 474
ss 528
nn 543
bb 510
gg 387
```

算出平均数

```
[root@clsn6 awk]#  awk '{sum=0;for(i=2;i<=NF;i++)sum+=$i;avg=sum/(NF-1);print $1,sum,avg}' cheng
cc 570 95
ll 474 79
ss 528 88
nn 543 90.5
bb 510 85
gg 387 64.5
```

格式化命令

```
awk '{
    sum=0
    for(i=2;i<=NF;i++)
    sum+=$i
    avg=sum/(NF-1)
    print $1,sum,avg
}' chengji.txt
```

1.3.9 打印下面语句中字符数小于6的单词

文件内容

```
echo "I am clsn ops,I very like linux hahahaha." > text.txt
```

shell方法

```
[root@clsn6 awk]# for i in `sed  's#,# #g' text.txt`
> do
>    [ ${#i} -lt 6 ] && echo $i
> done
I
am
clsn
ops
I
very
like
linux
```

grep 方法

```
[root@clsn6 awk]# egrep -wo '[a-Z]{,6}' text.txt
I
am
clsn
ops
I
very
like
linux
```

awk方法

```
[root@clsn6 awk]# echo clsn |awk '{print length($1)}'
4
[root@clsn6 awk]# awk -F "[, ]" '{for(i=1;i<=NF;i++) if (length($i)<6) print $i}'   text.txt
I
am
clsn
ops
I
very
like
linux
```

1.4 附录

1.4.1 sort命令的使用

```
[root@clsn6 awk]# du -sh /* 2>/dev/null|sort -hr
1.1G    /usr
214M    /lib
120M    /var
109M    /root
38M    /boot
...
```

1.4.2 统计流量使用的命令

```
[root@clsn6 ~]# yum install htop iotop iftop atop nethogs -y
# iftop 总体流量使用情况
# nethogs 显示进行级别的流量使用
```

1.4.3 其他日志分析方法

```
awstat，piwiki，elk
```

1.5 参考文献

[1] https://www.gnu.org/software/gawk/manual/gawk.html

[2] https://www.cnblogs.com/ginvip/p/6352157.html

惨绿少年Linux www.nmtui.com