

5_Shell06 高级 sed 应用 awk 基本用法

一 awk

1.1 awk 特性

是一种编程语言/数据处理引擎

精确定位查找数据,并可再次处理

基于模式匹配检查输入文本,逐行处理并输出

通常用在 shell 脚本中,获取指定的数据

单独用时,可对文本数据做统计

1.2 awk 命令

格式 1: 前置命令 | awk [选项] '[条件]{指令}'

格式 2: awk [选项] '[条件]{指令}' 文件...

选项: -F 指定分隔符,可省略(默认空格或 tab)

指令: print 输出内容到屏幕

默认下以空格或 tab 分隔列

\$n 表示列号,\$0 表示所有列;NR 显示行号,NF 显示列号

```
[root@server0 opt]# cat test1
```

```
hello the world
```

```
welcome to beijing
```

1.2.1 选项默认,无条件案例

```
[root@server0 opt]# awk '{print}' test1 #输出所有内容
```

```
hello the world
```

```
welcome to beijing
```

```
[root@server0 opt]# awk '{print $2}' test1 #输出第 2 列
```

```
the
```

```
to
```

```
[root@server0 opt]# awk '{print $2}' test1 #输出第 3 2 列
```

```
world the
```

```
beijing to
```

```
[root@server0 opt]# awk '{print $0,$3}' test1
```

```
#输出所有列及第 3 列
```

```
hello the world world
```

```
welcome to beijing beijing
```

1.2.2 有选项,有条件案例

```
[root@server0 opt]# head -6 /etc/passwd > user
```

```
[root@server0 opt]# awk -F: '{print $6}' user
```

```
#指定:为列分隔符,输出第 6 列,即家目录
```

```
[root@server0 opt]# awk -F[:/] '{print $9}' user
```

```
#以:或/为分隔符,输出第 9 列.文件中有:/时,:与/中间仍算 1 列
```

```
[root@server0 opt]# awk -Fo '{print $2}' test1
```

```
#以小写 o 为分隔符,输出中的第 2 列
```

```
[root@server0 opt]# df -h / | awk '{print $4}'
```

#查看系统/分区可用空间大小

```
可用
```

```
7.1G
```

```
[root@server0 opt]# awk '{print NR}' test1 #输出行号
```

```
1
```

```
2
```

```
[root@server0 opt]# awk '{print NF}' test1 #输出列号
```

```
3
```

```
3
```

```
[root@server0 opt]# awk '{print NR,NF}' test1 #输出行列号
```

```
1 3
```

```
2 3
```

```
[root@server0 opt]# awk -F[:] '{print NR,NF}' /etc/passwd
```

#以:为分隔符,输出行列号

```
[root@server0 opt]# awk -F[:/] '{print NR,NF}' /etc/passwd
```

#以:或/为分隔符,输出行列号,:/之间无内容,也算1列

```
[root@server0 opt]# awk -Fbin/ '{print $2}' user
```

#以 bin/ 为分隔符,输出第2列

1.2.3 df -h 结合 awk 查看/分区信息详细步骤:

```
[root@server0 opt]# df -h / #查看/分区信息
```

| 文件系统 | 容量 | 已用 | 可用 | 已用% | 挂载点 |
|------|----|----|----|-----|-----|
|------|----|----|----|-----|-----|

| | | | | | |
|-----------|-----|------|------|-----|--|
| /dev/vda1 | 10G | 3.0G | 7.1G | 30% | |
|-----------|-----|------|------|-----|--|

```
[root@server0 opt]# df -h / | awk '{print $4}'
```

可用

7.1G #以空格为分隔符查看/分区信息的第 4 列

```
[root@server0 opt]# df -h / | awk '/vda1/{print $4}'
```

7.1G #以空格为分隔符,以正则表达式/vda1/作为条件筛选行,查看该行的第 4 列

```
[root@server0 opt]# df -h / | awk '/vda1/{print "根分区剩余容
```

```
量是:"$4}" #以空格为分隔符,以正则表达式/vda1/作为条件筛选行,查看该行
```

的第 4 列,并添加提示信息

根分区剩余容量是:7.1G

1.2.4 ifconfig eth0 命令结合 awk 查看网卡 eth0 信息详细:

```
[root@server0 opt]# ifconfig eth0 | awk '/RX p/{print $5}'
```

#输出网卡 eth0 接收的流量字节数

#以空格为分隔符,以正则表达式/RX p/作为条件筛选行,查看该行的第 5 列

```
[root@server0 opt]# ifconfig eth0 | awk '/RX p/{print "eth0 网
```

```
卡接收的流量是:"$5"字节}"
```

eth0 网卡接收的流量是:888469 字节

#以空格为分隔符,以正则表达式/RX p/作为条件筛选行,查看该行的第 5 列,并添

加提示信息

```
[root@server0 opt]# ifconfig eth0 | awk '/TX p/{print "eth0 网卡发送的流量是:"$5"字节"}'
```

eth0 网卡发送的流量是:431674 字节

#以空格为分隔符,以正则表达式/TX p/作为条件筛选行,查看该行的第 5 列,并添加提示信息

1.2.5 awk 刷选系统安全日志

```
[root@desktop0 ~]# tail -20 /var/log/secure | awk '/Failed/{print $11}'
```

#以空格为分隔符,以正则表达式/Failed/作为条件筛选行,查看该行第 11 列

```
[root@desktop0 ~]# cat /var/log/secure | awk '/Failed/{print "地址为:"$11"的主机尝试登录本机,因输入错误密码而失败!"}'
```

#以空格为分隔符,以正则表达式/Failed/作为条件筛选行,查看该行的第 11 列,并添加提示信息

#/var/log/secure 文件记录本机的安全信息

```
[root@desktop0 ~]# awk '/Failed/{print "地址为:"$11"的主机尝试登录本机,因输入错误密码而失败!"}' /var/log/secure
```

#此命令与上一条命令效果相同

1.2.6 awk 输出用户及解释器

```
[root@server0 ~]# head -5 /etc/passwd | awk -F: '{print "用户
```

"\$1"的解释器为"\$7}"'

#以:为分隔符,筛选/etc/passwd 文件前 5 行的第 1 列和第 5 列,并添加提示信息

```
[root@server0 ~]# head -5 /etc/passwd | awk -F: '{print "用户
```

"\$1"的解释器为"\$7",家目录为"\$6}"'

#以:为分隔符,筛选/etc/passwd 文件前 5 行的第 1 列和第 7 6 列,及提示信息

1.3 awk 条件

条件的表现形式

正则表达式

/正则表达式/,直接支持扩展正则表达式

~匹配(包含);!~不匹配(不包含); \转义

数值/字符串比较

== != > < >= <=;被比较的字符串为常量,需加""

逻辑比较 && 逻辑与 || 逻辑或

运算符 + - * / % ++ -- += -= *= /= %= 直接支持小数运算

1.4 控制 awk 的工作流程

1.4.1 格式

BEGIN{指令} 执行 1 次

{指令} 执行 n 次

END{指令} 执行 1 次,若涉及到行,只处理最后一行

\t tab 制表符,为常量,使用时加""

案例：格式化输出/etc/passwd 文件

要求：格式化输出 passwd 文件内容时，要求第一行为列表标题，中间打印用户的名称、UID、家目录信息，最后一行提示一共已处理文本的总行数，如图所示。

```
User      UID      Home
root      0        /root
bin       1        /bin
daemon    2        /sbin
adm       3        /var/adm
.. ..
Total 59 lines.
```

```
[root@server0 ~]# awk -F: 'BEGIN{print"User\t\tUID\t\tHome"}
{print$1"\t\t"$3"\t\t"$6}END{print"Total",NR,"lines."}'
```

/opt/user

| User | UID | Home |
|------|-----|------|
|------|-----|------|

| | | |
|------|---|-------|
| root | 0 | /root |
|------|---|-------|

| | | |
|-----|---|------|
| bin | 1 | /bin |
|-----|---|------|

| | | |
|--------|---|-------|
| daemon | 2 | /sbin |
|--------|---|-------|

| | | |
|-----|---|----------|
| adm | 3 | /var/adm |
|-----|---|----------|

| | | |
|----|---|----------------|
| lp | 4 | /var/spool/lpd |
|----|---|----------------|

| | | |
|------|---|-------|
| sync | 5 | /sbin |
|------|---|-------|

Total 6 lines.

1.4.2 awk 正则表达式+包含(不包含)案例

```
[root@server0 opt]# awk -F: '$6~/\bin/{print}' user
```

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

#以:为分隔符,输出文件 user 内第 6 列包含/bin 的行,其中{print}可省略,

\为转义

```
[root@server0 opt]# awk -F: '$6!~/\bin/{print}' user
```

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

```
daemon:x:2:2:daemon:/sbin:/sbin/nologin
```

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

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

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

#以:为分隔符,输出文件 user 内第 6 列不包含/bin 的行,其中{print}可省略

```
[root@server0 opt]# awk -F: '$1~/root/' user
```

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

#以:为分隔符,输出文件 user 内第 1 列包含/root/的行,其中{print}已省略

```
[root@server0 opt]# awk -F: '/bin/' user #输出有 bin 的行
```

```
[root@server0 opt]# awk -F: '$6~/bin/' user
```

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

```
daemon:x:2:2:daemon:/sbin:/sbin/nologin
```



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

#输出第 6 列包含 bin 的行

1.4.3 正则结合数值/字符串比较案例

```
[root@server0 opt]# awk -F: '$3>=1000{print$1,$3}'
```

```
/etc/passwd
```

```
nfsnobody 65534
```

```
student 1000
```

```
abc 1001
```

#以:为分隔符,输出/etc/passwd 内普通用户的用户名称和 UID 信息

#普通用户的 UID 大于等于 1000

```
[root@server0 opt]# awk -F: '$1=="root"{print}' /etc/passwd
```

#以:为分隔符,输出/etc/passwd 内第 1 列字符为 root 的行,在字符串比较中,字符串为常量,需要加""

```
[root@server0 opt]# awk -F: '$3==0{print}' /etc/passwd
```

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

#以:为分隔符,输出/etc/passwd 内第 3 列字符为 0 的行 (UID 为 0 的行)

```
[root@server0 opt]# awk -F: '$3<10{print}' /etc/passwd
```

#以:为分隔符,输出/etc/passwd 内 UID 小于 10 的行

```
[root@server0 opt]# awk -F: '$3<10{print $1}' /etc/passwd
```

#以:为分隔符,输出/etc/passwd 内 UID 小于 10 的行的用户名

1.4.4 awk 结合逻辑比较案例

```
[root@server0 opt]# awk 'NR>=3 && NR<=5' user
```

#输出 user 内行号大于等于 3, 小于等于 5 的行,{print}已省略

```
[root@server0 opt]# awk 'NR>3 && NR<5' user
```

#输出 user 内行号大于 3 且小于 5 的行,{print}已省略

```
[root@server0 opt]# awk 'NR>3 || NR<5' user
```

#输出 user 内行号大于 3, 或行号小于 5 的行[输出结果为所有],{print}已省略

```
[root@server0 opt]# awk 'NR>5 && NR<3' user
```

#输出 user 内行号大于 5 且小于 3 的行,逻辑错误

```
[root@server0 opt]# awk -F: '$3>=100 && $3<=1000 {print  
t}' /etc/passwd
```

#以:为分隔符,输出/etc/passwd 内 UID 在 100-1000 范围内的行

1.4.5 awk 结合数值运算

```
[root@svr5 ~]# awk 'BEGIN{x++;print x}'
```

1 #x++,x--只能放 print 前,若 x 未赋值,默认为 0,BEGIN 内以;做分隔

```
[root@svr5 ~]# awk 'BEGIN{x=8;print x+=2}'
```

10

```
[root@svr5 ~]# awk 'BEGIN{x=8;x--;print x}'
```

7

```
[root@server0 opt]# awk 'BEGIN{x=8;x++;print x+=2}'
```

11

```
[root@server0 opt]# awk 'BEGIN{x=8;x--;print x-=2}'
```

5

```
[root@server0 opt]# awk 'BEGIN{print 2.2+3}'
```

5.2 #直接支持小数运算

```
[root@server0 opt]# awk 'BEGIN{print 2.2-3}'
```

-0.8 #直接支持小数运算

```
[root@server0 opt]# awk 'BEGIN{print 2.2*3}'
```

6.6 #直接支持小数运算

```
[root@server0 opt]# awk 'BEGIN{print 2.2/3}'
```

0.733333 #直接支持小数运算

```
[root@server0 opt]# awk 'BEGIN{print 2.2%3}'
```

2.2 #直接支持小数运算

```
[root@svr5 ~]# seq 200 | awk '$1%3==0'
```

#输出 200 以内 3 的倍数,{print}已省略

#seq 200 的输出为 1 列,\$1 表示每行第 1 列

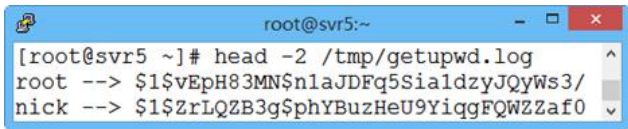
1.4.6 综合练习:sed 综合脚本应用

本案例要求编写脚本 `getupwd.sh`, 实现以下需求:

找到使用 `bash` 作登录 Shell 的本地用户

列出这些用户的 `shadow` 密码记录

按每行“用户名 --> 密码记录”保存到 getupwd.log



```
root@svr5:~  
[root@svr5 ~]# head -2 /tmp/getupwd.log  
root --> $1$vEpH83MN$nlajDFq5SialdzyJQyWs3/  
nick --> $1$ZrLQZB3g$phYBuzHeU9YiqgFQWZZaf0
```

基本思路:

先用 **sed** 工具取出登录 Shell 为/bin/bash 的用户记录，保存为临时文件/tmp/urec.tmp，并计算记录数量

再结合 **while** 循环遍历取得的账号记录，逐行进行处理

针对每一行用户记录，采用掐头去尾的方式获得用户名、密码字符串

按照指定格式追加到/tmp/getuupwd.log 文件

结束循环后删除临时文件，报告分析结果

编写 getupwd.sh 脚本

```
[root@svr5 ~]# vim ./getupwd.sh  
  
#/bin/bash  
  
A=$(sed -n '/bash$/s/:.*/p' /etc/passwd)  
  
#在/etc/passwd 文件中以 bash 结尾的每一行中,把第 1 个:及:后的所有  
内容替换为空,并输出给变量 A(提取符合条件的账号记录)  
  
for i in $A          #遍历账号记录  
do  
  
    pass1=$(grep $i /etc/shadow) #在 shadow 中找到用户所在行
```

```
pass2=${pass1#*:} #删除用户所在行第一个:及:左边的内容,掐头
pass=${pass2%*:} #删除用户所在行最后一个:及:右边的内容,去尾
echo "$i --> $pass" #输出结果

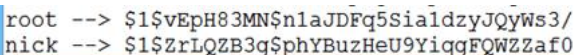
done
```

1.4.7 综合练习:awk 综合脚本应用

编写脚本,实现以下需求:

找到使用 **bash** 作登录 **Shell** 的本地用户

列出这些用户的 **shadow** 密码记录,如图-2 所示



```
root --> $1$vEpH83MN$nlajDFq5SialdzyJQyWs3/
nick --> $1$ZrLQZB3g$phYBuzHeU9YiqgFQWZZaf0
```

任务需求及思路分析

编写脚本的任务要求如下:

分析出使用 **bash** 作登录 **Shell** 的本地用户

列出这些用户的 **shadow** 密码记录

按每行“用户名 --> 密码记录”保存结果

根据实现思路编写脚本

```
[root@svr5 ~]# vim getupwd-awk.sh
```

```
#!/bin/bash
```

```
A=$(awk -F: '/bash$/{print $1}' /etc/passwd)
```

```
for i in $A
```

```
do
```

```
    grep $i: /etc/shadow | awk -F: '{print$1,"-->",$2}'
```

```
done
```

```
#!/bin/bash
```

```
a=$(awk -F: '/bash$/{print $1}' /etc/passwd)
```

```
for i in $a
```

```
do
```

```
    b=$(grep $i: /etc/shadow | awk -F: '{print $2}')
```

```
    echo "$a --> $b" >> /opt/2.txt
```

```
done
```

#在 shadow 文件中,以:为分隔符,grep \$i:消除筛选到 root1 类似的用户