第14章 正则表达式

学习Linux请加QQ群： 群1(163262181) 群2(148412746) 群3(246401509) 群4(173884211)

跟阿铭学Linux邀请函 (http://www.aminglinux.com)，猿课已上线，请加微信aminglinux84索要配套视频教程。

这部分内容可以说是学习shell脚本之前必学的内容。如果你这部分内容学的越好，那么你的shell脚本编写能力就会越强。所以不要嫌这部分内容啰嗦，也不要怕麻烦，要用心学习。一定要多加练习，练习多了就能熟练掌握了。

在计算机科学中，正则表达式是这样解释的：它是指一个用来描述或者匹配一系列符合某个句法规则的字符串的单个字符串。在很多文本编辑器或其他工具里，正则表达式通常被用来检索和/或替换那些符合某个模式的文本内容。许多程序设计语言都支持利用正则表达式进行字符串操作。对于系统管理员来讲，正则表达式贯穿在我们的日常运维工作中，无论是查找某个文档，抑或查询某个日志文件分析其内容，都会用到正则表达式。

其实正则表达式，只是一种思想，一种表示方法。只要我们使用的工具支持表示这种思想那么这个工具就可以处理正则表达式的字符串。常用的工具有grep, sed, awk 等，下面阿铭就分别介绍一下这三种工具的使用方法。

grep / egrep

阿铭在前面的内容中多次提到并用到grep命令，可见它的重要性。所以好好学习一下这个重要的命令吧。你要知道的是grep连同下面讲的sed, awk都是针对文本的行才操作的。

语法： `grep [-cinvABC] 'word' filename`

-c ： 打印符合要求的行数

-i ： 忽略大小写

-n ： 在输出符合要求的行的同时连同行号一起输出

-v ： 打印不符合要求的行

-A ： 后跟一个数字（有无空格都可以），例如 -A2则表示打印符合要求的行以及下面两行

-B ： 后跟一个数字，例如 -B2 则表示打印符合要求的行以及上面两行

-C ： 后跟一个数字，例如 -C2 则表示打印符合要求的行以及上下各两行

```
[root@localhost ~]# grep -A2 'halt' /etc/passwd
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
uucp:x:10:14:uucp:/var/spool/uucp:/sbin/nologin
```

把包含 'halt' 的行以及这行下面的两行都打印出。

```
[root@localhost ~]# grep -B2 'halt' /etc/passwd
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
```

把包含 'halt' 的行以及这行上面的两行都打印出。

```
[root@localhost ~]# grep -C2 'halt' /etc/passwd
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
uucp:x:10:14:uucp:/var/spool/uucp:/sbin/nologin
```

把包含 'halt' 的行以及这行上面和下面的各两行都打印出。

下面阿铭举几个典型实例帮你更深刻的理解grep.

1. 过滤出带有某个关键词的行并输出行号

```
[root@localhost ~]# grep -n 'root' /etc/passwd
1:root:x:0:0:root:/root:/bin/bash
11:operator:x:11:0:operator:/root:/sbin/nologin
```

2. 过滤不带有某个关键词的行，并输出行号

```
[root@localhost ~]# grep -nv 'nologin' /etc/passwd
1:root:x:0:0:root:/root:/bin/bash
```

目录列表

```
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
26:test:x:511:511::/home/test:/bin/bash
27:test1:x:512:511::/home/test1:/bin/bash
```

3. 过滤出所有包含数字的行

```
[root@localhost ~]# grep '[0-9]' /etc/inittab
# upstart works, see init(5), init(8), and initctl(8).
#   0 - halt (Do NOT set initdefault to this)
#   1 - Single user mode
#   2 - Multiuser, without NFS (The same as 3, if you do not have networking)
#   3 - Full multiuser mode
#   4 - unused
#   5 - X11
#   6 - reboot (Do NOT set initdefault to this)
id:3:initdefault:
```

4. 过滤出所有不包含数字的行

```
[root@localhost ~]# grep -v '[0-9]' /etc/inittab
# inittab is only used by upstart for the default runlevel.
#
# ADDING OTHER CONFIGURATION HERE WILL HAVE NO EFFECT ON YOUR SYSTEM.
#
# System initialization is started by /etc/init/rcS.conf
#
# Individual runlevels are started by /etc/init/rc.conf
#
# Ctrl-Alt-Delete is handled by /etc/init/control-alt-delete.conf
#
# Terminal gettys are handled by /etc/init/tty.conf and /etc/init/serial.conf,
# with configuration in /etc/sysconfig/init.
#
# For information on how to write upstart event handlers, or how
#
# Default runlevel. The runlevels used are:
#
```

5. 把所有以‘#’开头的行去除

```
[root@localhost ~]# grep -v '^#' /etc/inittab
id:3:initdefault:
```

5. 去除所有空行和以‘#’开头的行

```
[root@localhost ~]# grep -v '^#' /etc/crontab |grep -v '^$'
SHELL=/bin/bash
PATH=/sbin:/bin:/usr/sbin:/usr/bin
MAILTO=root
HOME=/
```

在正则表达式中，“^”表示行的开始，“$”表示行的结尾，那么空行则可以用“^$”表示，如何打印出不以英文字母开头的行呢？

```
[root@localhost ~]# vim test.txt
[root@localhost ~]# cat test.txt
123
abc
456

abc2323
#laksdjf
Allllllll
```

阿铭先在test.txt中写几行字符串，用来做实验。

```
[root@localhost ~]# grep '^[^a-zA-Z]' test.txt
123
456
#laksdjf
```

```
[root@localhost ~]# grep '[^a-zA-Z]' test.txt
123
456
abc2323
#laksdjf
```

在前面阿铭也提到过这个 '[ ]' 的应用，如果是数字的话就用[0-9]这样的形式，当然有时候也可以用这样的形式[15]即只含有1或者5，注意，它不会认为是15。如果要过滤出数字以及大小写字母则要这样写[0-9a-zA-Z]。另外[ ]还有一种形式，就是[^字符] 表示除[ ]内的字符之外的字符。

7. 过滤任意一个字符与重复字符

```
[root@localhost ~]# grep 'r..o' /etc/passwd
operator:x:11:0:operator:/root:/sbin/nologin
gopher:x:13:30:gopher:/var/gopher:/sbin/nologin
vcsa:x:69:69:virtual console memory owner:/dev:/sbin/nologin
```

`.` 表示任意一个字符，上例中，就是把符合r与o之间有两个任意字符的行过滤出来， `*` 表示零个或多个前面的字符。

```
[root@localhost ~]# grep 'ooo*' /etc/passwd
root:x:0:0:root:/root:/bin/bash
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
uucp:x:10:14:uucp:/var/spool/uucp:/sbin/nologin
operator:x:11:0:operator:/root:/sbin/nologin
postfix:x:89:89::/var/spool/postfix:/sbin/nologin
```

'ooo*' 表示oo, ooo, oooo ... 或者更多的 'o' 现在你是否想到了 '.*' 这个组合表示什么意义？

```
[root@localhost ~]# grep '.*' /etc/passwd |wc -l
27
[root@localhost ~]# wc -l /etc/passwd
27 /etc/passwd
```

'.*' 表示零个或多个任意字符，空行也包含在内。

8. 指定要过滤字符出现的次数

```
[root@localhost ~]# grep 'o\{2\}' /etc/passwd
root:x:0:0:root:/root:/bin/bash
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
uucp:x:10:14:uucp:/var/spool/uucp:/sbin/nologin
operator:x:11:0:operator:/root:/sbin/nologin
postfix:x:89:89::/var/spool/postfix:/sbin/nologin
```

这里用到了{ }，其内部为数字，表示前面的字符要重复的次数。上例中表示包含有两个o 即 'oo' 的行。注意，{ }左右都需要加上脱意字符 '\'，另外，使用{ }我们还可以表示一个范围的，具体格式是 '{n1,n2}' 其中n1<n2，表示重复n1到n2次前面的字符，n2还可以为空，则表示大于等于n1次。

上面部分讲的grep，另外阿铭常常用到egrep这个工具，简单点讲，后者是前者的扩展版本，我们可以用egrep完成grep不能完成的工作，当然了grep能完成的egrep完全可以完成。如果你嫌麻烦，egrep了解一下即可，因为grep的功能已经足够可以胜任你的日常工作了。下面阿铭介绍egrep不用于grep的几个用法。为了试验方便，阿铭把test.txt 编辑成如下内容：

```
rot:x:0:0:/rot:/bin/bash
operator:x:11:0:operator:/root:/sbin/nologin
operator:x:11:0:operator:/rooot:/sbin/nologin
roooot:x:0:0:/roooooot:/bin/bash
11111111111111111111111111111
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

1. 筛选一个或一个以上前面的字符

```
[root@localhost ~]# egrep 'o+' test.txt
rot:x:0:0:/rot:/bin/bash
operator:x:11:0:operator:/root:/sbin/nologin
operator:x:11:0:operator:/rooot:/sbin/nologin
roooot:x:0:0:/roooooot:/bin/bash
[root@localhost ~]# egrep 'oo+' test.txt
operator:x:11:0:operator:/root:/sbin/nologin
operator:x:11:0:operator:/rooot:/sbin/nologin
roooot:x:0:0:/roooooot:/bin/bash
[root@localhost ~]# egrep 'ooo+' test.txt
```

```
operator:x:11:0:operator:/rooot:/sbin/nologin
roooot:x:0:0:/roooooot:/bin/bash
```

和grep 不同的是，egrep这里是使用'+'的。

2. 筛选零个或一个前面的字符

```
[root@localhost ~]# egrep 'o?' test.txt
rot:x:0:0:/rot:/bin/bash
operator:x:11:0:operator:/root:/sbin/nologin
operator:x:11:0:operator:/rooot:/sbin/nologin
roooot:x:0:0:/roooooot:/bin/bash
11111111111111111111111111111111
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
[root@localhost ~]# egrep 'ooo?' test.txt
operator:x:11:0:operator:/root:/sbin/nologin
operator:x:11:0:operator:/rooot:/sbin/nologin
roooot:x:0:0:/roooooot:/bin/bash
[root@localhost ~]# egrep 'oooo?' test.txt
operator:x:11:0:operator:/rooot:/sbin/nologin
roooot:x:0:0:/roooooot:/bin/bash
```

3. 筛选字符串1或者字符串2

```
[root@localhost ~]# egrep 'aaa|111|ooo' test.txt
operator:x:11:0:operator:/rooot:/sbin/nologin
roooot:x:0:0:/roooooot:/bin/bash
11111111111111111111111111111111
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

4. egrep中( )的应用

```
[root@localhost ~]# egrep 'r(oo|at)o' test.txt
operator:x:11:0:operator:/root:/sbin/nologin
operator:x:11:0:operator:/rooot:/sbin/nologin
roooot:x:0:0:/roooooot:/bin/bash
```

用( )表示一个整体，例如(oo)+就表示1个 'oo' 或者多个 'oo'

```
[root@localhost ~]# egrep '(oo)+' test.txt
operator:x:11:0:operator:/root:/sbin/nologin
operator:x:11:0:operator:/rooot:/sbin/nologin
roooot:x:0:0:/roooooot:/bin/bash
```

sed工具的使用

grep工具的功能其实还不够强大，grep实现的只是查找功能，而它却不能实现把查找的内容替换掉。以前用vim的时候，可以查找也可以替换，但是只局限于在文本内部来操作，而不能输出到屏幕上。sed工具以及下面要讲的awk工具就能实现把替换的文本输出到屏幕上的功能了，而且还有其他更丰富的功能。sed和awk都是流式编辑器，是针对文档的行来操作的。

1. 打印某行

`sed -n 'n'p filename` 单引号内的n是一个数字，表示第几行：

```
[root@localhost ~]# sed -n '2'p /etc/passwd
bin:x:1:1:bin:/bin:/sbin/nologin
```

要想把所有行都打印出来可以使用 `sed -n '1,$'p filename`

```
[root@localhost ~]# sed -n '1,$'p test.txt
rot:x:0:0:/rot:/bin/bash
operator:x:11:0:operator:/root:/sbin/nologin
operator:x:11:0:operator:/rooot:/sbin/nologin
roooot:x:0:0:/roooooot:/bin/bash
11111111111111111111111111111111
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

也可以指定一个区间：

```
[root@localhost ~]# sed -n '1,3'p test.txt
rot:x:0:0:/rot:/bin/bash
operator:x:11:0:operator:/root:/sbin/nologin
```

```
operator:x:11:0:operator:/rooot:/sbin/nologin
```

2. 打印包含某个字符串的行

```
[root@localhost ~]# sed -n '/root/'p test.txt
operator:x:11:0:operator:/root:/sbin/nologin
```

grep中使用的特殊字符，如 `^ $ . *` 等同样也能在sed中使用

```
[root@localhost ~]# sed -n '/^1/'p test.txt
11111111111111111111111111111111
[root@localhost ~]# sed -n '/in$/'p test.txt
operator:x:11:0:operator:/root:/sbin/nologin
operator:x:11:0:operator:/rooot:/sbin/nologin
[root@localhost ~]# sed -n '/r..o/'p test.txt
operator:x:11:0:operator:/root:/sbin/nologin
operator:x:11:0:operator:/rooot:/sbin/nologin
roooot:x:0:0:/roooooot:/bin/bash
[root@localhost ~]# sed -n '/ooo*/'p test.txt
operator:x:11:0:operator:/root:/sbin/nologin
operator:x:11:0:operator:/rooot:/sbin/nologin
roooot:x:0:0:/roooooot:/bin/bash
```

3. -e可以实现多个行为

```
[root@localhost ~]# sed -e '1'p -e '/111/'p -n test.txt
rot:x:0:0:/rot:/bin/bash
11111111111111111111111111111111
```

4. 删除某行或者多行

```
[root@localhost ~]# sed '1'd test.txt
operator:x:11:0:operator:/root:/sbin/nologin
operator:x:11:0:operator:/rooot:/sbin/nologin
roooot:x:0:0:/roooooot:/bin/bash
11111111111111111111111111111111
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
[root@localhost ~]# sed '1,3'd test.txt
roooot:x:0:0:/roooooot:/bin/bash
11111111111111111111111111111111
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
[root@localhost ~]# sed '/oot/'d test.txt
rot:x:0:0:/rot:/bin/bash
11111111111111111111111111111111
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

‘d’这个字符就是删除的动作了，不仅可以删除指定的单行以及多行，而且还可以删除匹配某个字符的行，另外还可以删除从某一行一直到文档末行。

5. 替换字符或字符串

```
[root@localhost ~]# sed '1,2s/ot/to/g' test.txt
rto:x:0:0:/rto:/bin/bash
operator:x:11:0:operator:/roto:/sbin/nologin
operator:x:11:0:operator:/rooot:/sbin/nologin
roooot:x:0:0:/roooooot:/bin/bash
11111111111111111111111111111111
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

上例中的‘s’就是替换的命令，‘g’为本行中全局替换，如果不加‘g’只换该行中出现的第一个。除了可以使用‘/’作为分隔符外，还可以使用其他特殊字符例如‘#’或者‘@’都没有问题。

```
[root@localhost ~]# sed 's#ot#to#g' test.txt
rto:x:0:0:/rto:/bin/bash
operator:x:11:0:operator:/roto:/sbin/nologin
operator:x:11:0:operator:/rooto:/sbin/nologin
roooto:x:0:0:/rooooto:/bin/bash
11111111111111111111111111111111
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
[root@localhost ~]# sed 's@ot@to@g' test.txt
rto:x:0:0:/rto:/bin/bash
operator:x:11:0:operator:/roto:/sbin/nologin
operator:x:11:0:operator:/rooto:/sbin/nologin
```

```
roooto:x:0:0:/rooooto:/bin/bash
11111111111111111111111111111111
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

现在思考一下，如何删除文档中的所有数字或者字母？

```
[root@localhost ~]# sed 's/[0-9]//g' test.txt
rot:x:::/rot:/bin/bash
operator:x:::operator:/root:/sbin/nologin
operator:x:::operator:/rooot:/sbin/nologin
roooot:x:::/roooooot:/bin/bash

aaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

[0-9]表示任意的数字。这里你也可以写成[a-zA-Z]甚至[0-9a-zA-Z]

```
[root@localhost ~]# sed 's/[a-zA-Z]//g' test.txt
::0:0:/:://
::11:0::/://
::11:0::/://
::0:0:/:://
11111111111111111111111111111111
```

5. 调换两个字符串的位置

```
[root@localhost ~]# sed 's/\(rot\)\(.*\)\(bash\)/\3\2\1/' test.txt
bash:x:0:0:/rot:/bin/rot
operator:x:11:0:operator:/root:/sbin/nologin
operator:x:11:0:operator:/rooot:/sbin/nologin
roooot:x:0:0:/roooooot:/bin/bash
11111111111111111111111111111111
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

这个就需要解释一下了，上例中用 () 把所想要替换的字符括起来成为一个整体，因为括号在sed中属于特殊符号，所以需要在前面加脱意字符 ‘\’, 替换时则写成 ‘\1’, ‘\2’, ‘\3’ 的形式。除了调换两个字符串的位置外，阿铭还常常用到在某一行前或者后增加指定内容。

```
[root@localhost ~]# sed 's/^.*$/123&/' test.txt
123rot:x:0:0:/rot:/bin/bash
123operator:x:11:0:operator:/root:/sbin/nologin
123operator:x:11:0:operator:/rooot:/sbin/nologin
123roooot:x:0:0:/roooooot:/bin/bash
12311111111111111111111111111111
123aaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

7. 直接修改文件的内容

```
[root@localhost ~]# sed -i 's/ot/to/g' test.txt
[root@localhost ~]# cat test.txt
rto:x:0:0:/rto:/bin/bash
operator:x:11:0:operator:/roto:/sbin/nologin
operator:x:11:0:operator:/rooto:/sbin/nologin
roooto:x:0:0:/rooooto:/bin/bash
11111111111111111111111111111111
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

这样就可以直接更改test.txt文件中的内容了。由于这个命令可以直接把文件修改，所以在修改前最好先复制一下文件以免改错。

sed常用到的也就上面这些了，只要你多加练习就能熟悉它了。为了能让你更加牢固的掌握sed的应用，阿铭留几个练习题，希望你能认真完成。

1. 把/etc/passwd 复制到/root/test.txt，用sed打印所有行
2. 打印test.txt的3到10行
3. 打印test.txt 中包含 ‘root’ 的行
4. 删除test.txt 的15行以及以后所有行
5. 删除test.txt中包含 ‘bash’ 的行
6. 替换test.txt 中 ‘root’ 为 ‘toor’
7. 替换test.txt中 ‘/sbin/nologin’ 为 ‘/bin/login’
8. 删除test.txt中5到10行中所有的数字
9. 删除test.txt 中所有特殊字符（除了数字以及大小写字母）
0. 把test.txt中第一个单词和最后一个单词调换位置
1. 把test.txt中出现的第一个数字和最后一个单词替换位置
2. 把test.txt 中第一个数字移动到行末尾
3. 在test.txt 20行到末行最前面加 ‘aaa:’

阿铭希望你能尽量多想一想，答案只是用来作为参考的。

sed习题答案

```
1.  /bin/cp /etc/passwd  /root/test.txt ;  sed -n '1,$'p test.txt
2.  sed -n '3,10'p test.txt
3.  sed -n '/root/'p test.txt
4.  sed '15,$'d  test.txt
5.  sed '/bash/'d test.txt
6.  sed 's/root/toor/g' test.txt
7.  sed 's#sbin/nologin#bin/login#g' test.txt
8.  sed '5,10s/[0-9]//g' test.txt
9.  sed 's/[^0-9a-zA-Z]//g' test.txt
10.  sed 's/\(^[a-zA-Z][a-zA-Z]*\)\([^a-zA-Z].*\)\([^a-zA-Z]\)\([a-zA-Z][a-zA-Z]*$\)/\4\2\3\1/' test.txt
11.  sed 's#\([^0-9][^0-9]*\)\([0-9][0-9]*\)\([^0-9].*\)\([^a-zA-Z]\)\([a-zA-Z][a-zA-Z]*$\)#\1\5\3\4\2#' test.txt
12.  sed 's#\([^0-9][^0-9]*\)\([0-9][0-9]*\)\([^0-9].*$\)#\1\3\2#' test.txt
13.  sed '20,$s/^.*$/aaa:&/' test.txt
```

awk工具的使用

上面也提到了awk和sed一样是流式编辑器，它也是针对文档中的行来操作的，一行一行的去执行。awk比sed更加强大，它能做到sed能做到的，同样也能做到sed不能做到的。awk工具其实是很复杂的，有专门的书籍来介绍它的应用，但是阿铭认为学那么复杂没有必要，只要能处理日常管理工作中的问题即可。何必让自己的脑袋装那么东西来为难自己？毕竟用的也不多，即使现在教会了你很多，你也学会了，如果很久不用肯定就忘记了。鉴于此，阿铭仅介绍比较常见的awk应用，如果你感兴趣的话，再去深入研究吧。

1. 截取文档中的某个段

```
[root@localhost ~]# head -n2 /etc/passwd |awk -F ':' '{print $1}'
root
bin
```

解释一下，-F 选项的作用是指定分隔符，如果不加-F指定，则以空格或者tab为分隔符。 Print为打印的动作，用来打印出某个字段。$1为第一个字段，$2为第二个字段，依次类推，有一个特殊的那就是$0，它表示整行。

```
[root@localhost ~]# head -n2 test.txt |awk -F':' '{print $0}'
rto:x:0:0:/rto:/bin/bash
operator:x:11:0:operator:/roto:/sbin/nologin
```

注意awk的格式，-F后紧跟单引号，然后里面为分隔符，print的动作要用 { } 括起来，否则会报错。print还可以打印自定义的内容，但是自定义的内容要用双引号括起来。

```
[root@localhost ~]# head -n2 test.txt |awk -F':' '{print $1"#"$2"#"$3"#"$4}'
rto#x#0#0
operator#x#11#0
```

2. 匹配字符或字符串

```
[root@localhost ~]# awk '/oo/' test.txt
operator:x:11:0:operator:/rooto:/sbin/nologin
roooto:x:0:0:/rooooto:/bin/bash
```

跟sed很类似吧，不过还有比sed更强大的匹配。

```
[root@localhost ~]# awk -F ':' '$1 ~/oo/' test.txt
roooto:x:0:0:/rooooto:/bin/bash
```

可以让某个段去匹配，这里的'~'就是匹配的意思，继续往下看

```
[root@localhost ~]# awk -F ':' '/root/ {print $1,$3} /test/ {print $1,$3}' /etc/passwd
root 0
operator 11
test 511
test1 512
```

awk还可以多次匹配，如上例中匹配完root，再匹配test，它还可以只打印所匹配的段。

3. 条件操作符

```
[root@localhost ~]# awk -F ':' '$3=="0"' /etc/passwd
root:x:0:0:root:/root:/bin/bash
```

awk中是可以用逻辑符号判断的，比如 '==' 就是等于，也可以理解为 '精确匹配' 另外也有 >, '>=, '<, '<=, '!= 等等，值得注意的是，在和数字比较时，若把比较的数字用双引号引起来后，那么awk不会认为是

数字，而认为是字符，不加双引号则认为是数字。

```
[root@localhost ~]# awk -F ':' '$3>="500"' /etc/passwd
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
nobody:x:99:99:Nobody:/:/sbin/nologin
dbus:x:81:81:System message bus:/:/sbin/nologin
vcsa:x:69:69:virtual console memory owner:/dev:/sbin/nologin
haldaemon:x:68:68:HAL daemon:/:/sbin/nologin
postfix:x:89:89::/var/spool/postfix:/sbin/nologin
sshd:x:74:74:Privilege-separated SSH:/var/empty/sshd:/sbin/nologin
tcpdump:x:72:72::/:/sbin/nologin
user11:x:510:502:user11,user11's office,12345678,123456789:/home/user11:/sbin/nologin
test:x:511:511::/home/test:/bin/bash
test1:x:512:511::/home/test1:/bin/bash
```

在上面的例子中，阿铭本想把uid大于等于500的行打印出，但是结果并不是我们的预期，这是因为awk把所有的数字当作字符来对待了，就跟上一章中提到的 sort 排序原理一样。

```
[root@localhost ~]# awk -F ':' '$7!="/sbin/nologin"' /etc/passwd
root:x:0:0:root:/root:/bin/bash
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
test:x:511:511::/home/test:/bin/bash
test1:x:512:511::/home/test1:/bin/bash
```

!= 为不匹配，除了针对某一个段的字符进行逻辑比较外，还可以两个段之间进行逻辑比较。

```
[root@localhost ~]# awk -F ':' '$3<$4' /etc/passwd
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
uucp:x:10:14:uucp:/var/spool/uucp:/sbin/nologin
games:x:12:100:games:/usr/games:/sbin/nologin
gopher:x:13:30:gopher:/var/gopher:/sbin/nologin
ftp:x:14:50:FTP User:/var/ftp:/sbin/nologin
```

另外还可以使用 && 和 || 表示“并且”和“或者”的意思。

```
[root@localhost ~]# awk -F ':' '$3>"5" && $3<"7"' /etc/passwd
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
vcsa:x:69:69:virtual console memory owner:/dev:/sbin/nologin
haldaemon:x:68:68:HAL daemon:/:/sbin/nologin
user11:x:510:502:user11,user11's office,12345678,123456789:/home/user11:/sbin/nologin
test:x:511:511::/home/test:/bin/bash
test1:x:512:511::/home/test1:/bin/bash
```

也可以是或者

```
[root@localhost ~]# awk -F ':' '$3>"5" || $7=="/bin/bash"' /etc/passwd
root:x:0:0:root:/root:/bin/bash
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
nobody:x:99:99:Nobody:/:/sbin/nologin
dbus:x:81:81:System message bus:/:/sbin/nologin
vcsa:x:69:69:virtual console memory owner:/dev:/sbin/nologin
haldaemon:x:68:68:HAL daemon:/:/sbin/nologin
postfix:x:89:89::/var/spool/postfix:/sbin/nologin
sshd:x:74:74:Privilege-separated SSH:/var/empty/sshd:/sbin/nologin
tcpdump:x:72:72::/:/sbin/nologin
user11:x:510:502:user11,user11's office,12345678,123456789:/home/user11:/sbin/nologin
test:x:511:511::/home/test:/bin/bash
test1:x:512:511::/home/test1:/bin/bash
```

4. awk的内置变量

awk常用的变量有：

NF ： 用分隔符分隔后一共有多少段

NR ： 行数

```
[root@localhost ~]# head -n3 /etc/passwd | awk -F ':' '{print NF}'
7
```

```
7
7
[root@localhost ~]# head -n3 /etc/passwd | awk -F ':' '{print $NF}'
/bin/bash
/sbin/nologin
/sbin/nologin
```

NF 是多少段，而$NF是最后一段的值，而NR则是行号。

```
[root@localhost ~]# head -n3 /etc/passwd | awk -F ':' '{print NR}'
1
2
3
```

我们可以使用行号作为判断条件：

```
[root@localhost ~]# awk 'NR>20' /etc/passwd
postfix:x:89:89::/var/spool/postfix:/sbin/nologin
abrt:x:173:173::/etc/abrt:/sbin/nologin
sshd:x:74:74:Privilege-separated SSH:/var/empty/sshd:/sbin/nologin
tcpdump:x:72:72::/:/sbin/nologin
user11:x:510:502:user11,user11's office,12345678,123456789:/home/user11:/sbin/nologin
test:x:511:511::/home/test:/bin/bash
test1:x:512:511::/home/test1:/bin/bash
```

也可以配合段匹配一起使用：

```
[root@localhost ~]# awk -F ':' 'NR>20 && $1 ~ /ssh/' /etc/passwd
sshd:x:74:74:Privilege-separated SSH:/var/empty/sshd:/sbin/nologin
```

5. awk中的数学运算

awk可以把段值更改：

```
[root@localhost ~]# head -n 3 /etc/passwd |awk -F ':' '$1="root"'
root x 0 0 root /root /bin/bash
root x 1 1 bin /bin /sbin/nologin
root x 2 2 daemon /sbin /sbin/nologin
```

awk还可以对各个段的值进行数学运算：

```
[root@localhost ~]# head -n2 /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
[root@localhost ~]# head -n2 /etc/passwd |awk -F ':' '{$7=$3+$4}'
[root@localhost ~]# head -n2 /etc/passwd |awk -F ':' '{$7=$3+$4; print $0}'
root x 0 0 root /root 0
bin x 1 1 bin /bin 2
```

当然还可以计算某个段的总和

```
[root@localhost ~]# awk -F ':' '{(tot=tot+$3)}; END {print tot}' /etc/passwd
2891
```

这里的END要注意一下，表示所有的行都已经执行，这是awk特有的语法，其实awk连同sed都可以写成一个脚本文件，而且有他们特有的语法，在awk中使用if判断、for循环都是可以的，只是阿铭认为日常管理工作中没有必要使用那么复杂的语句而已。

```
[root@localhost ~]# awk -F ':' '{if ($1=="root") print $0}' /etc/passwd
root:x:0:0:root:/root:/bin/bash
```

基本上，正则表达的内容就这些了。但是阿铭要提醒你一下，阿铭介绍的这些仅仅是最基本的东西，并没有提啊深入的去讲sed和awk，但是完全可以满足日常工作的需要，有时候也许你会碰到比较复杂的需求，如果真遇到了就去请教一下google吧。下面出几道关于awk的练习题，希望你要认真完成。

1. 用awk 打印整个test.txt （以下操作都是用awk工具实现，针对test.txt）
2. 查找所有包含 'bash' 的行
3. 用 ':' 作为分隔符，查找第三段等于0的行
4. 用 ':' 作为分隔符，查找第一段为 'root' 的行，并把该段的 'root' 换成 'toor' (可以连同sed一起使用)
5. 用 ':' 作为分隔符，打印最后一段
6. 打印行数大于20的所有行
7. 用 ':' 作为分隔符，打印所有第三段小于第四段的行
8. 用 ':' 作为分隔符，打印第一段以及最后一段，并且中间用 '@' 连接 （例如，第一行应该是这样的形式 'root@/bin/bash' ）
9. 用 ':' 作为分隔符，把整个文档的第四段相加，求和

awk习题答案

```
1. awk '{print $0}' test.txt
2. awk '/bash/' test.txt
3. awk -F':' '$3=="0"' test.txt
4. awk -F':' '$1=="root"' test.txt |sed 's/root/toor/'
5. awk -F':' '{print $NF}' test.txt
6. awk -F':' 'NR>20' test.txt
7. awk -F':' '$3<$4' test.txt
8. awk -F':' '{print $1"@"$NF}' test.txt
9. awk -F':' '{(sum+=$4)}; END {print sum}' test.txt
```

阿铭建议你最好再扩展学习一下： http://www.aminglinux.com/bbs/thread-5438-1-1.html

教程答疑：请移步这里.

欢迎你加入 阿铭学院 和阿铭一起学习Linux，让阿铭成为你Linux生涯中永远的朋友吧！