18 文本处理工具awk基础

文本处理工具awk

在Linux文本处理三剑客中，相对于 `grep` 的**查找**、 `sed` 的**编辑**、 `awk` 尤为擅长**数据分析及生成报告**。

一、 awk概述

1.1 awk简介

`awk` 三个字母分别代表其创建者姓氏的第一个字母。因为它的创建者是三个人，分别是Alfred Aho、Peter Weinberger、Brian Kernighan。

`awk` 是不单一个强大的**文本分析工具**，还是一个**编程工具**。 `awk` 拥有自己的语言——**awk程序设计语言**，三位创建者将它定义为“**样式扫描和处理语言**”，支持**正则表达式、条件判断、数组、循环、自定义函数**等功能。

因此， `awk` 可以在命令行中作为命令使用，也可以以文件形式作为脚本来使用。

通过 `man awk` 可以获取相关功能说明。

1.2 awk工作原理

`awk` 由一个**主输入循环**维持，主输入循环反复执行，直到终止条件被触发。**主输入循环无须编写**， `awk` 已经搭好主输入循环的框架。主输入循环自动**逐行**读取输入文件行进行**处理，处理单元由模式和动作构成**，首先寻找**匹配的特定模式的行**，然后**在这些行上**执行**动作**。

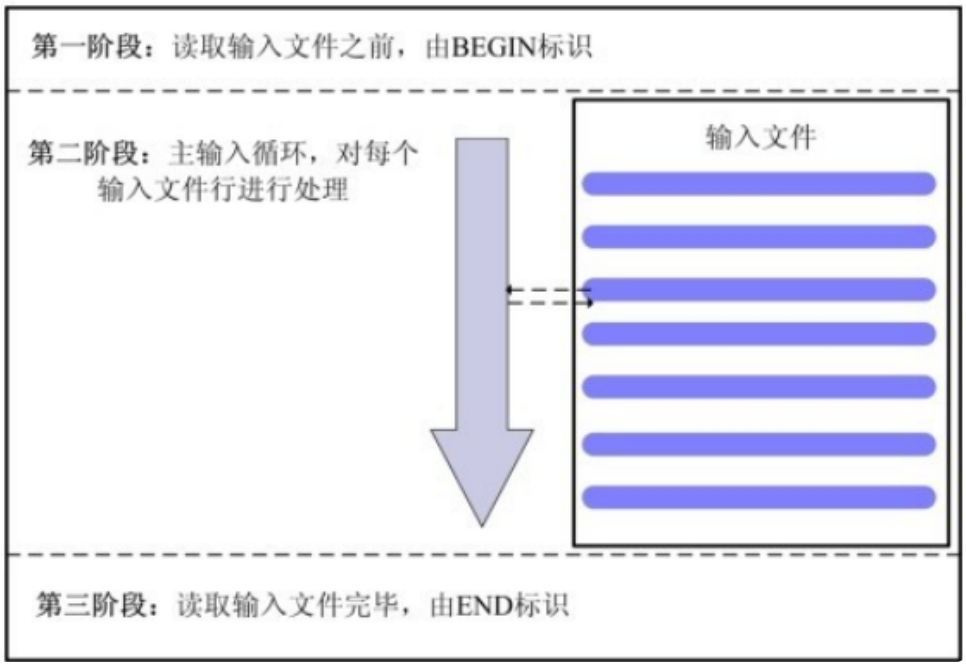
如果**没有指定模式**，则所有被操作**所指定的行都被处理**。

如果**没有指定处理动作**，则把匹配的行显示到**标准输出**（屏幕）。

`awk` 有两个特殊的模式： `BEGIN` 和 `END` 。

- `BEGIN` 模式在 `awk` 开始**从输入流中读取之前**被执行，这是一个**可选的语句块**，一般应用于变量初始化、打印输出表格的表头等情况。
- `END` 模式在 `awk` 在**处理完所有的文本之后**被执行（如打印所有行后）被执行，它也是一个**可选语句块**。一般应用于打印出分析结果等操作。

因此，`awk`的工作流程可表示为：`awk 'BEGIN{动作} 模式{动作} END {动作}' 文件`



二、awk基本语法

2.1 awk的语法格式

`awk`命令的格式为：`awk [选项] '[模式] {动作} ...' 文件...`

`awk`命令的选项是可选的，常用选项如下所示：

命令选项	描述
`-F`	指定作为输入行的分隔符，默认分隔符为空格或tab键。
`-v`	定义变量 `var=value` 。
`-f`	指定 `awk` 命令文件。

任何`awk`语句的主要**处理单元**都由**模式（pattern）**和**动作（action）**组成。

模式是一组用于测试输入行是否需要执行动作的**规则**。

动作是包含**语句、函数和表达式**的**执行过程**。

简言之，**模式决定动作何时触发和触发事件，动作执行对输入行的处理**。

模式是可选的，如果省略意味着对文件中**每一行均执行一次动作**。

最常用的动作是 `print` ，它可以输出特定数据。

数据可以来自**标准输入**、一个或多个**文件**，或其他**命令的输出**。

案例1：awk命令格式

在下面的例子中， `-F":"` 为选项， `/root/` 为模式， `{print}` 为动作。

Shell 复制代码

```shell
# 以:为分隔符分割字段，输出包含root的所有行中的第1个字段
[root@shell ~]# awk -F":" '/root/ {print $1}' /etc/passwd
root
operator
```

案例2：print动作

最常用的动作是 `print` ，它可以输出特定数据，既可以输出常量，也可以输出变量。

Shell 复制代码

```shell
# 输出/etc/hosts的全部内容
[root@shell ~]# awk '{print}' /etc/hosts
127.0.0.1   localhost localhost.localdomain localhost4 localhost4.localdomain4
::1         localhost localhost.localdomain localhost6 localhost6.localdomain6
# print输出常量，由于/etc/hosts有2行内容，所以输出2次
[root@shell ~]# awk '{print 123}' /etc/hosts
123
123
# 输出多个值时，值用,分隔
[root@shell ~]# awk '{print "abc",123}' /etc/hosts
abc 123
abc 123
# 输出常量和变量
[root@shell ~]# awk '{print "abc",123,$1}' /etc/hosts
abc 123 127.0.0.1
abc 123 ::1
```

2.2 记录和字段

`awk` 认为输入文件是**结构化**的， `awk` 将每个输入文件**行**定义为**记录**。

行中的每一**列**定义为**字段**，字段之间用**空格、Tab键或其他符号**进行分隔，分隔字段的符号叫做**分隔符**。**默认分隔符为空格**。

`awk` 定义字段操作符 `$` 来指定执行动作的字段，字段操作符 `$` 后面跟数字或变量来标识字段的位置，每条记录的**字段从1开始编号**，如 `$1` 表示第1个字段、 `$2` 表示第2个字段、 **`$0` 表示所有的字段**。

案例3：选择字段

```shell
# 不选择字段即输出所有字段
[root@shell ~]# awk '{print}' /etc/hosts
127.0.0.1   localhost localhost.localdomain localhost4 localhost4.localdomain4
::1         localhost localhost.localdomain localhost6 localhost6.localdomain6
# $0表示所有字段
[root@shell ~]# awk '{print $0}' /etc/hosts
127.0.0.1   localhost localhost.localdomain localhost4 localhost4.localdomain4
::1         localhost localhost.localdomain localhost6 localhost6.localdomain6
# $1表示第1个字段
[root@shell ~]# awk '{print $1}' /etc/hosts
127.0.0.1
::1
# $1 $3表示第1、3字段，中间无分隔符
[root@shell ~]# awk '{print $1 $3}' /etc/hosts
127.0.0.1localhost.localdomain
::1localhost.localdomain
# $1,$3表示第1、2、4字段，中间有分隔符
[root@shell ~]# awk '{print $1,$2,$4}' /etc/hosts
127.0.0.1 localhost localhost4
::1 localhost localhost6
# 每行分别输出第1、3字段
[root@shell ~]# awk '{print $1} {print $3}' /etc/hosts
127.0.0.1
localhost.localdomain
::1
localhost.localdomain
[root@shell ~]# awk '{print $1;print $3}' /etc/hosts
127.0.0.1
localhost.localdomain
::1
localhost.localdomain
```

2.3 分隔符

`awk` 的默认分隔符是空格键，Tab键被看做是**连续的空格键**来处理，可以使用 `awk` 的 `-F` 选项自定义分隔符。

案例4：自定义分隔符

Shell 复制代码

```
# 以:为分隔符，输出第1、7字段
[root@shell ~]# awk -F":" '{print $1,$7}' /etc/passwd
root /bin/bash
bin /sbin/nologin
daemon /sbin/nologin
adm /sbin/nologin
lp /sbin/nologin
sync /bin/sync
shutdown /sbin/shutdown
halt /sbin/halt
mail /sbin/nologin
operator /sbin/nologin
games /sbin/nologin
ftp /sbin/nologin
nobody /sbin/nologin
systemd-network /sbin/nologin
dbus /sbin/nologin
polkitd /sbin/nologin
sshd /sbin/nologin
postfix /sbin/nologin
# 以:为分隔符，输出第1个字段
[root@shell ~]# awk -F":" '{print $1}' /etc/passwd
root
bin
daemon
adm
lp
sync
shutdown
halt
mail
operator
games
ftp
nobody
systemd-network
dbus
polkitd
sshd
postfix
# 以:或-为分隔符，输出第1个字段
[root@shell ~]# awk -F "[:-]" '{print $1}' /etc/passwd
root
bin
daemon
```

```
adm
lp
sync
shutdown
halt
mail
operator
games
ftp
nobody
systemd
dbus
polkitd
sshd
postfix
```

2.4 内置变量

尽管 `-F` 选项可以改变分隔符，但是 `awk` 还提供了另一种方法来改变分隔符，这就是使用 `awk` 内置环境变量 `FS` ，可以通过**在 `BEGIN` 字段中设置 `FS` 的值**来改变分隔符。

案例5：通过内置变量设置分隔符

```shell
# 以:为分隔符分割字段，输出包含root的所有行中的第1个字段
[root@shell ~]# awk 'BEGIN {FS=":"} {print $1}' /etc/passwd
root
bin
daemon
adm
lp
sync
shutdown
halt
mail
operator
games
ftp
nobody
systemd-network
dbus
polkitd
sshd
postfix
```

`awk` 提供了有很多有用内置变量，常见的内置变量如下表所示。

变量	描述
`$0`	完整的输入记录。
`$n`	当前记录的第 `n` 个字段，字段间由 `FS` 分隔。
`NF`	浏览记录的**字段个数**。
`$NF`	**最后一个字段**的值。
`NR`	已读的记录数，理解为**行号**，多文件行号递增。
`FNR`	与 `NR` 类似，不过多文件记录数**不递增**，每个文件都从1开始。
`FS`	设置输入字段分隔符，同 `-F` 选项。
`RS`	记录分隔符（**默认是一个换行符**）。

`OFS`	输出数据时，每个字段间以 `OFS` 制定的字符作为分隔符。
`ORS`	输出数据时，每行记录间以 `ORS` 制定的字符作为分隔符。
`FILENAME`	awk浏览的文件名。
`ARGC`	命令行参数的数目。
`ARGIND`	命令行中当前文件的位置（从0开始算）。
`ARGV`	包含命令行参数的数组。
`FIELDWIDTHS`	字段宽度列表（用空格键分隔）。
`IGNORECASE`	如果为真，则进行忽略大小写的匹配。

案例6：内置变量

Shell | 复制代码

```
# NF表示每行的字段数量
[root@shell ~]# awk 'BEGIN {FS=":"}{print NF}' /etc/passwd
7
7
7
7
7
7
7
7
7
7
7
7
7
7
7
7
7
7
7
# $NF表示最后一个字段
[root@shell ~]# awk 'BEGIN {FS=":"}{print $NF}' /etc/passwd
/bin/bash
/sbin/nologin
/sbin/nologin
/sbin/nologin
/sbin/nologin
/bin/sync
/sbin/shutdown
/sbin/halt
/sbin/nologin
/sbin/nologin
/sbin/nologin
/sbin/nologin
/sbin/nologin
/sbin/nologin
/sbin/nologin
/sbin/nologin
/sbin/nologin
/sbin/nologin
```

Shell | 复制代码

```
# NR表示记录数（行号）
[root@shell ~]# awk '{print NR}' /etc/hosts
1
2
[root@shell ~]# awk '{print NR,$1}' /etc/hosts
1 127.0.0.1
2 ::1
[root@shell ~]# awk '{print NR,$1}' /etc/hosts /etc/passwd
1 127.0.0.1
2 ::1
3 root:x:0:0:root:/root:/bin/bash
4 bin:x:1:1:bin:/bin:/sbin/nologin
5 daemon:x:2:2:daemon:/sbin:/sbin/nologin
6 adm:x:3:4:adm:/var/adm:/sbin/nologin
7 lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
8 sync:x:5:0:sync:/sbin:/bin/sync
9 shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
10 halt:x:7:0:halt:/sbin:/sbin/halt
11 mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
12 operator:x:11:0:operator:/root:/sbin/nologin
13 games:x:12:100:games:/usr/games:/sbin/nologin
14 ftp:x:14:50:FTP
15 nobody:x:99:99:Nobody:/:/sbin/nologin
16 systemd-network:x:192:192:systemd
17 dbus:x:81:81:System
18 polkitd:x:999:998:User
19 sshd:x:74:74:Privilege-separated
20 postfix:x:89:89::/var/spool/postfix:/sbin/nologin
# 对于单个文件FNR功能与NR相同，对于多个文件每个文件单独计数
[root@shell ~]# awk '{print FNR,$1}' /etc/hosts
1 127.0.0.1
2 ::1
[root@shell ~]# awk '{print FNR,$1}' /etc/hosts /etc/passwd
1 127.0.0.1
2 ::1
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 operator:x:11:0:operator:/root:/sbin/nologin
```

```
11 games:x:12:100:games:/usr/games:/sbin/nologin
12 ftp:x:14:50:FTP
13 nobody:x:99:99:Nobody:/:/sbin/nologin
14 systemd-network:x:192:192:systemd
15 dbus:x:81:81:System
16 polkitd:x:999:998:User
17 sshd:x:74:74:Privilege-separated
18 postfix:x:89:89::/var/spool/postfix:/sbin/nologin
```

Shell | 复制代码

```
# RS表示行分隔符
[root@shell ~]# awk 'BEGIN {RS=" "} {print $1}' /etc/hosts
[root@shell ~]# awk 'BEGIN {RS=" "} {print}' /etc/hosts
127.0.0.1



localhost
localhost.localdomain
localhost4
localhost4.localdomain4
::1









localhost
localhost.localdomain
localhost6
localhost6.localdomain6

# ORS表示输出行分隔符
[root@shell ~]# awk 'BEGIN {ORS="-"} {print $1}' /etc/hosts
127.0.0.1-::1-[root@shell ~]#
# OFS表示输出字段分隔符
[root@shell ~]# awk 'BEGIN {OFS="-"} {print $1,$3}' /etc/hosts
127.0.0.1-localhost.localdomain
::1-localhost.localdomain
```

```shell
# FIELDWIDTHS表示按宽度分割字段
[root@shell ~]# awk 'BEGIN {FIELDWIDTHS="5"}{print $1,$2}' /etc/passwd
root:
bin:x
daemo
adm:x
lp:x:
sync:
shutd
halt:
mail:
opera
games
ftp:x
nobod
syste
dbus:
polki
sshd:
postf
# FIELDWIDTHS分割字段必须指定输出的每个字段的宽度
[root@shell ~]# awk 'BEGIN {FIELDWIDTHS="5 3"}{print $1,$2}' /etc/passwd
root: x:0
bin:x :1:
daemo n:x
adm:x :3:
lp:x: 4:7
sync: x:5
shutd own
halt: x:7
mail: x:8
opera tor
games :x:
ftp:x :14
nobod y:x
syste md-
dbus: x:8
polki td:
sshd: x:7
postf ix:
```

2.5 自定义变量

`awk` 可以通过 `-v` 选项**定义新的变量**，也可以使用 `-v` 选项**修改内置变量的值**。

案例7：通过-v定义新变量

```shell
[root@shell ~]# awk -v x=1 -v y="b" '{print x,y}' /etc/hosts
1 b
1 b
# 注意$1 $2是脚本参数
[root@shell ~]# vi test_awk_v.sh
awk -v a=$2 '{print $a}' $1
[root@shell ~]# . test_awk_v.sh /etc/hosts 1
127.0.0.1
::1
```

案例8：通过-v修改内置变量

```shell
[root@shell ~]# awk -v FS=":" '{print $1}' /etc/passwd
root
bin
daemon
adm
lp
sync
shutdown
halt
mail
operator
games
ftp
nobody
systemd-network
dbus
polkitd
sshd
postfix
```

`awk` 除了可以使用自己的内置变量之外，还可以使用系统定义的变量。

案例9：awk使用系统定义的变量

```shell
[root@shell ~]# x=1
[root@shell ~]# awk -v a=$x  '{print a}' /etc/hosts
1
1
[root@shell ~]# awk '{print "'$x'"}' /etc/hosts
1
1
```

2.6 模式匹配

`awk` 既支持使用**正则表达式进行模糊匹配**，也支持使用**关系运算符**进行比较操作，并且支持使用**逻辑运算符**进行逻辑操作。

2.6.1 正则表达式匹配

`awk` 支持的正则表达式比较符号如下。

符号	含义
`/正则表达式/`	对全部数据进行正则匹配
`!/正则表达式/`	对全部数据进行正则匹配后取反
`~/正则表达式/`	对特定数据进行正则匹配
`!~/正则表达式/`	对特点数据进行正则匹配后取反

案例10：正则表达式模式

Shell | 复制代码

```
# 输出包含root的所有行
[root@shell ~]# awk -F":" '/root/ {print}' /etc/passwd
root:x:0:0:root:/root:/bin/bash
operator:x:11:0:operator:/root:/sbin/nologin
[root@shell ~]# awk -F":" '/root/ {print $1}' /etc/passwd
root
operator
# 输出不包含root的所有行
[root@shell ~]# awk -F":" '!/root/ {print}' /etc/passwd
bin:x:1:1:bin:/bin:/sbin/nologin
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
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
games:x:12:100:games:/usr/games:/sbin/nologin
ftp:x:14:50:FTP User:/var/ftp:/sbin/nologin
nobody:x:99:99:Nobody:/:/sbin/nologin
systemd-network:x:192:192:systemd Network Management:/:/sbin/nologin
dbus:x:81:81:System message bus:/:/sbin/nologin
polkitd:x:999:998:User for polkitd:/:/sbin/nologin
sshd:x:74:74:Privilege-separated SSH:/var/empty/sshd:/sbin/nologin
postfix:x:89:89::/var/spool/postfix:/sbin/nologin
# 输出第一列包含root的行
[root@shell ~]# awk -F":" '$1~/root/ {print $0}' /etc/passwd
root:x:0:0:root:/root:/bin/bash
# 输出第一列不包含root的行
[root@shell ~]# awk -F":" '$1!~/root/ {print}' /etc/passwd
bin:x:1:1:bin:/bin:/sbin/nologin
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
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
operator:x:11:0:operator:/root:/sbin/nologin
games:x:12:100:games:/usr/games:/sbin/nologin
ftp:x:14:50:FTP User:/var/ftp:/sbin/nologin
nobody:x:99:99:Nobody:/:/sbin/nologin
systemd-network:x:192:192:systemd Network Management:/:/sbin/nologin
dbus:x:81:81:System message bus:/:/sbin/nologin
polkitd:x:999:998:User for polkitd:/:/sbin/nologin
```

```
sshd:x:74:74:Privilege-separated SSH:/var/empty/sshd:/sbin/nologin
postfix:x:89:89::/var/spool/postfix:/sbin/nologin
```

`awk` 支持 `?` 和 `+` 两个扩展元字符，而 `grep` 和 `sed` 并不支持。 `awk` 支持的正则表达式元字符如下：

元字符	解释
`^`	行首定位符。 例如：/^root/ 表示匹配所有以root开头的行
`$`	行尾定位符。 例如：/root$/ 表示匹配所有以root结尾的行
`.`	匹配任意单个字符。 例如：/r..t 表示字段3数值减10
`*`	匹配0个或多个前导字符（包括回车）。 例如：/a*ool/ 表示匹配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。 例如：/a+b/ 表示匹配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。 例如：/a?b/ 表示匹配b或ab的行
`[]`	匹配指定字符组内的任意一个字符。 例如：/^[abc] 表示以字母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。 例如：/^[^abc] 表示匹配不以字母a或b或c开头的行
`()`	子表达式组合。 例如：/(root)+/ 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。 例如：/(root)|B/ 表示匹配root或B的行
`\`	转义字符。 例如：/a\/\// 表示匹配a//

`x{m}`
`x{m,}`
`x{m,n}`
	x重复m次，x重复至少m次，x重复至少m次但不超过n次，需要指定参数–posix或者—re–interval没有该参数不能使用该模式。
	例如：/(root){3}/，/(root){3}/，/(root){5,6}/。需要注意一点的是，root加括号和不加括号的区别，x可以表示字符串也可以是一个字符，所以/root\{5\}/表示匹配roo再加上5个t，及roottttt，/\(root\)\{2,\}/则表示匹配rootrootrootroot等。

Shell | 复制代码

```
[root@shell ~]# awk -F":" '/^root/ {print}' /etc/passwd
root:x:0:0:root:/root:/bin/bash
```

2.6.2 关系表达式匹配

`awk` 支持的关系表达式比较符号如下。

运算符	描述
`==`	等于，精确比较。 例如：awk '$3= ="48" {print $0}' file 只打印第3个字段等于 "48"的记录
`!=`	不等于，精确比较。 例如：awk '$1 !="abc" file' 表示提取第一个字段不是abc的行
`>`	大于。 例如：awk '$1>500 {print $2}' file表示如果字段1的值大于500，则打印字段2
`>=`	大于等于。 例如：awk '$1>=400 {print $2}' file 表示如果字段1的值大于等于400，则打印字段2

<	小于。 例如：awk '$1<200{print $2}' file 表示如果字段1的值小于200，则打印字段2
<=	小于等于。 例如：awk '$1<=100 {print $2}' file表示如果字段1的值小于等于100，则打印字段2

案例11：比较运算符支持

Shell | 复制代码

```
# 输出第三列值等于99的行
[root@shell ~]# awk -F":" '$3==99' /etc/passwd
nobody:x:99:99:Nobody:/:/sbin/nologin
# 输出第一列值等于bin的行
[root@shell ~]# awk -F":" '$1=="bin"' /etc/passwd
bin:x:1:1:bin:/bin:/sbin/nologin
# 输出第三列值大于100的行
[root@shell ~]# awk -F":" '$3>100' /etc/passwd
systemd-network:x:192:192:systemd Network Management:/:/sbin/nologin
polkitd:x:999:998:User for polkitd:/:/sbin/nologin
# 输出第三列值大于100的行的第一列
[root@shell ~]# awk -F":" '$3>100 {print $1}' /etc/passwd
systemd-network
polkitd
# 输出第一行内容
[root@shell ~]# awk -F":" 'NR==1' /etc/passwd
root:x:0:0:root:/root:/bin/bash
```

2.6.3 逻辑运算符

逻辑符号	含义
&&	逻辑与，如果A&&B，要求满足A并且满足B
||	逻辑或，如果A||B，要求满足A或者满足B

案例12：逻辑运算符

```shell
# 逻辑与
[root@shell ~]# awk -F":" '$3>1&&$3<5 {print $1}' /etc/passwd
daemon
adm
lp
# 逻辑或
[root@shell ~]# awk -F":" '$3==1||$3==5 {print $1}' /etc/passwd
bin
```

小结

- awk工作原理：`BEGIN 主循环逐行处理（模式 动作） END`
- awk基本语法：`awk [选项] '[模式] {动作} ...' 文件...`

课程目标

- 知识目标：熟练掌握`awk`命令的基本语法。
- 技能目标：能够利用`awk`命令完成实战场景的处理。

课外拓展

- 进一步了解`awk`命令的应用场景。

参考资料

- `awk --help`或`man awk`
- 《Linux Shell核心编程指南》，丁明一，电子工业出版社