LINUX 下 grep/sed/gawk 指令详解

目　录

grep

1. grep 正则表达式元字符集（基本集）

^ 锚定行的开始 如：'^grep'匹配所有以 grep 开头的行。

$ 锚定行的结束 如：'grep$'匹配所有以 grep 结尾的行。

. 匹配一个非换行符的字符 如：'gr.p'匹配 gr 后接一个任意字符，然后是 p。

* 匹配零个或多个先前字符 如：'*grep'匹配所有一个或多个空格后紧跟 grep 的行。 .*一起用代表任意字符。

[] 匹配一个指定范围内的字符，如'[Gg]rep'匹配 Grep 和 grep。

[^] 匹配一个不在指定范围内的字符，如：'[^A-FH-Z]rep'匹配不包含 A-R 和 T-Z 的一个字母开头，紧跟 rep 的行。

\(..\) 标记匹配字符，如'\(love\)'，love 被标记为 1。

\< 锚定单词的开始，如:'\<grep'匹配包含以 grep 开头的单词的行。

\> 锚定单词的结束，如'grep\>'匹配包含以 grep 结尾的单词的行。

x\{m\} 重复字符 x，m 次，如：'0\{5\}'匹配包含 5 个 o 的行。

x\{m,\} 重复字符 x,至少 m 次，如：'o\{5,\}'匹配至少有 5 个 o 的行。

x\{m,n\} 重复字符 x，至少 m 次，不多于 n 次，如：'o\{5,10\}'匹配 5--10 个 o 的行。

\w 匹配文字和数字字符，也就是[A-Za-z0-9]，如：'G\w*p'匹配以 G 后跟零个或多个文字或数字字符，然后是 p。

\W \w 的反置形式，匹配一个或多个非单词字符，如点号句号等。

\b 单词锁定符，如: '\bgrepb\'只匹配 grep。

2. 用于 egrep 和 grep -E 的元字符扩展集

+ 匹配一个或多个先前的字符。如：'[a-z]+able'，匹配一个或多个小写字母后跟 able 的串，如 loveable,enable,disable 等。

? 匹配零个或多个先前的字符。如：'gr?p'匹配 gr 后跟一个或没有字符，然后是 p 的行。

a|b|c 匹配 a 或 b 或 c。如：grep|sed 匹配 grep 或 sed

() 分组符号，如：love(able|rs)ov+匹配 loveable 或 lovers，匹配一个或多个 ov。

x{m},x{m,},x{m,n} 作用同 x\{m\},x\{m,\},x\{m,n\}

3. POSIX 字符类

为了在不同国家的字符编码中保持一至，POSIX(The Portable Operating System Interface)增加了特殊的字符类，如[:alnum:]是 A-Za-z0-9 的另一个写法。要把它们放到[]号内才能成为正则表达式，如[A- Za-z0-9]或[[:alnum:]]。在 linux 下的 grep 除 fgrep 外，都支持 POSIX 的字符类。

[:alnum:] 文字数字字符

[:alpha:]文字字符

[:digit:] 数字字符

[:graph:] 非空字符（非空格、控制字符）

[:lower:] 小写字符

[:cntrl:] 控制字符

[:print:] 非空字符（包括空格）

[:punct:] 标点符号

[:space:] 所有空白字符（新行，空格，制表符）

[:upper:] 大写字符

[:xdigit:] 十六进制数字（0-9，a-f，A-F）

4. Grep 命令选项

-? 同时显示匹配行上下的? 行，如：grep -2 pattern filename 同时显示匹配行的上下 2 行。

-b，--byte-offset 打印匹配行前面打印该行所在的块号码。

-c,--count 只打印匹配的行数，不显示匹配的内容。

-f File，--file=File 从文件中提取模板。空文件中包含 0 个模板，所以什么都不匹配。

-h，--no-filename 当搜索多个文件时，不显示匹配文件名前缀。

-i，--ignore-case 忽略大小写差别。

-q，--quiet 取消显示，只返回退出状态。0 则表示找到了匹配的行。

-l，--files-with-matches 打印匹配模板的文件清单。

-L，--files-without-match 打印不匹配模板的文件清单。

-n，--line-number 在匹配的行前面打印行号。

-s，--silent 不显示关于不存在或者无法读取文件的错误信息。

-v，--revert-match 反检索，只显示不匹配的行。

-w，--word-regexp 如果被\<和\>引用，就把表达式做为一个单词搜索。

-V，--version 显示软件版本信息。

5. 实例

要用好 grep 这个工具，其实就是要写好正则表达式，所以这里不对 grep 的所有功能进行实例讲解，只列几个例子，讲解一个正则表达式的写法。

$ ls -l | grep '^a'

通过管道过滤 ls -l 输出的内容，只显示以 a 开头的行。

$ grep 'test' d*

显示所有以 d 开头的文件中包含 test 的行。

$ grep 'test' aa bb cc

显示在 aa，bb，cc 文件中匹配 test 的行。

$ grep '[a-z]\{5\}' aa

显示所有包含每个字符串至少有 5 个连续小写字符的字符串的行。

$ grep 'w\(es\)t.*\1' aa

如果 west 被匹配，则 es 就被存储到内存中，并标记为 1，然后搜索任意个字符（.*），这些字符后面紧跟着另外一个 es（\1），找到就显示该行。如果用 egrep 或 grep -E，就不用"\"号进行转义，直接写成'w(es)t.*\1'就可以了。

sed

1. 定址功能

可以通过定址来定位你所希望编辑的行，该地址用数字构成，用逗号分隔的两个行数表示以这两行为起止的行的范围（包括行数表示的那两行）。如 1，3 表示 1，2，3 行，美元符号($)表示最后一行。范围可以通过数据，正则表达式或者二者结合的方式确定 。

2.sed 命令调用格式

调用 sed 命令有两种形式：

* sed [options] 'command' file(s)

* sed [options] -f scrīptfile file(s)

a\ 在当前行后面加入一行文本。

b lable 分支到脚本中带有标记的地方，如果分支不存在则分支到脚本的末尾。

c\ 用新的文本改变本行的文本。

d 从模板块（Pattern space）位置删除行。

D 删除模板块的第一行。

i\ 在当前行上面插入文本。

h 拷贝模板块的内容到内存中的缓冲区。

H 追加模板块的内容到内存中的缓冲区。

g 获得内存缓冲区的内容，并替代当前模板块中的文本。

G 获得内存缓冲区的内容，并追加到当前模板块文本的后面。

l 列表不能打印字符的清单。

n 读取下一个输入行，用下一个命令处理新的行而不是用第一个命令。

N 追加下一个输入行到模板块后面并在二者间嵌入一个新行，改变当前行号码。

p 打印模板块的行。

P（大写） 打印模板块的第一行。

q 退出 Sed。

r file 从 file 中读行。

t label if 分支，从最后一行开始，条件一旦满足或者 T，t 命令，将导致分支到带有标号的命令处，或者到脚本的末尾。

T label 错误分支，从最后一行开始，一旦发生错误或者 T，t 命令，将导致分支到带有标号的命令处，或者到脚本的末尾。

w file 写并追加模板块到 file 末尾。

W file 写并追加模板块的第一行到 file 末尾。

! 表示后面的命令对所有没有被选定的行发生作用。

s/re/string 用 string 替换正则表达式 re。

= 打印当前行号码。

*把注释扩展到下一个换行符以前。

以下的是替换标记

- g 表示行内全面替换。
- p 表示打印行。
- w 表示把行写入一个文件。
- x 表示互换模板块中的文本和缓冲区中的文本。
- y 表示把一个字符翻译为另外的字符（但是不用于正则表达式）

3. 选项 -e command, --expression=command

允许多台编辑。

-h, --help 打印帮助，并显示 bug 列表的地址。

-n, --quiet, --silent 取消默认输出。

-f, --filer=scrīpt-file 引导 sed 脚本文件名。

-V, --version 打印版本和版权信息。

4. 元字符集

^ 锚定行的开始 如：/^sed/匹配所有以 sed 开头的行。

$ 锚定行的结束 如：/sed$/匹配所有以 sed 结尾的行。

. 匹配一个非换行符的字符 如：/s.d/匹配 s 后接一个任意字符，然后是 d。

* 匹配零或多个字符 如：/*sed/匹配所有模板是一个或多个空格后紧跟 sed 的行。

[] 匹配一个指定范围内的字符，如/[Ss]ed/匹配 sed 和 Sed。

[^] 匹配一个不在指定范围内的字符，如：/[^A-RT-Z]ed/匹配不包含 A-R 和 T-Z 的一个字母开头，紧跟 ed 的行。

\(..\) 保存匹配的字符，如 s/\(love\)able/\1rs，loveable 被替换成 lovers。

& 保存搜索字符用来替换其他字符，如 s/love/**&**/，love 这成**love**。

\< 锚定单词的开始，如:/\<love/匹配包含以 love 开头的单词的行。

\> 锚定单词的结束，如/love\>/匹配包含以 love 结尾的单词的行。 x\{m\} 重复字符 x，m 次，如：/0\{5\}/匹配包含 5 个 o 的行。

x\{m,\} 重复字符 x,至少 m 次，如：/o\{5,\}/匹配至少有 5 个 o 的行。 x\{m,n\} 重复字符 x，至少 m 次，不多于 n 次，如：/o\{5,10\}/匹配 5--10 个 o 的行。

5. 实例

5.1 删除：d 命令

* $ sed '2d' example-----删除 example 文件的第二行。
* $ sed '2,$d' example-----删除 example 文件的第二行到末尾所有行。
* $ sed '$d' example-----删除 example 文件的最后一行。
* $ sed '/test/'d example-----删除 example 文件所有包含 test 的行。

5.2 替换：s 命令

* $ sed 's/test/mytest/g' example-----在整行范围内把 test 替换为 mytest。如果没有 g 标记，则只有每行第一个匹配的 test 被替换成 mytest。
* $ sed -n 's/^test/mytest/p' example-----(-n)选项和 p 标志一起使用表示只打印那些发生替换的行。也就是说，如果某一行开头的 test 被替换成 mytest，就打印它。
* $ sed 's/^192.168.0.1/&localhost/' example-----&符号表示替换换字符串中被找到的部份。所有以 192.168.0.1 开头的行都会被替换成它自己加 localhost，变成 192.168.0.1localhost。
* $ sed -n 's/\(love\)able/\1rs/p' example-----love 被标记为 1,所有 loveable

会被替换成 lovers，而且替换的行会被打印出来。

* $ sed 's#10#100#g' example-----不论什么字符，紧跟着 s 命令的都被认为是新的分隔符，所以，“#”在这里是分隔符，代替了默认的“/”分隔符。表示把所有 10 替换成 100。

5.3 选定行的范围：逗号

* $ sed -n '/test/,/check/p' example-----所有在模板 test 和 check 所确定的范围内的行都被打印。

* $ sed -n '5,/^test/p' example-----打印从第五行开始到第一个包含以 test 开始的行之间的所有行。

* $ sed '/test/,/check/s/$/sed test/' example-----对于模板 test 和 west 之间的行，每行的末尾用字符串 sed test 替换。

5.4 多点编辑：e 命令

* $ sed -e '1,5d' -e 's/test/check/' example-----(-e)选项允许在同一行里执行多条命令。如例子所示，第一条命令删除 1 至 5 行，第二条命令用 check 替换 test。命令的执行顺序对结果有影响。如果两个命令都是替换命令，那么第一个替换命令将影响第二个替换命令的结果。

* $ sed --expression='s/test/check/' --expression='/love/d' example-----一个比-e 更好的命令是--expression。它能给 sed 表达式赋值。

5.5 从文件读入：r 命令

* $ sed '/test/r file' example-----file 里的内容被读进来，显示在与 test 匹配的行后面，如果匹配多行，则 file 的内容将显示在所有匹配行的下面。

5.6 写入文件：w 命令

* $ sed -n '/test/w file' example-----在 example 中所有包含 test 的行都被写入 file 里。

5.7 追加命令：a 命令

* $ sed '/^test/a\\--->this is a example' example<-----'this is a example' 被追加到以 test 开头的行后面，sed 要求命令 a 后面有一个反斜杠。

5.8 插入：i 命令

```
$ sed '/test/i\\
new line
-------------------------' example
```

如果 test 被匹配，则把反斜杠后面的文本插入到匹配行的前面。

下一个：n 命令

* $ sed '/test/{ n; s/aa/bb/; }' example-----如果 test 被匹配，则移动到匹配行的下一行，替换这一行的 aa，变为 bb，并打印该行，然后继续。

5.9 变形：y 命令

* $ sed '1,10y/abcde/ABCDE/' example-----把 1--10 行内所有 abcde 转变为大写，注意，正则表达式元字符不能使用这个命令。

5.10 退出：q 命令

* $ sed '10q' example-----打印完第 10 行后，退出 sed。

5.11 保持和获取：h 命令和 G 命令

* $ sed -e '/test/h' -e '$G example-----在 sed 处理文件的时候，每一行都被保存在一个叫模式空间的临时缓冲区中，除非行被删除或者输出被取消，否则所有被处理的行都将打印在屏幕上。接着模式空间被清空，并存入新的一行等待处理。在这个例子里，匹配 test 的行被找到后，将存入模式空间，h 命令将其复制并存入一个称为保持缓存区的特殊缓冲区内。第二条语句的意思是，当到达最后一行后，G 命令取出保持缓冲区的行，然后把它放回模式空间中，且追加到现在已经存在于模式空间中的行的末尾。在这个例子中就是追加到最后一行。简单来说，任何包含 test 的行都被复制并追加到该文件的末尾。

5.12 保持和互换：h 命令和 x 命令

* $ sed -e '/test/h' -e '/check/x' example -----互换模式空间和保持缓冲区的内容。也就是把包含 test 与 check 的行互换。

6. 脚本

Sed 脚本是一个 sed 的命令清单，启动 Sed 时以-f 选项引导脚本文件名。Sed 对于脚本中输入的命令非常挑剔，在命令的末尾不能有任何空白或文本，如果在一行中有多个命令，要用分号分隔。以#开头的行为注释行，且不能跨行。 7. 小技巧

* 在 sed 的命令行中引用 shell 变量时要使用双引号，而不是通常所用的单引号。下面是一个根据 name 变量的内容来删除 named.conf 文件中 zone 段的脚本：

```
name='zone\ "localhost"'
sed "/$name/,/};/d" named.conf
```

gawk

1.gawk 的主要功能

gawk 的主要功能是针对文件的每一行(l i n e)，也就是每一条记录，搜寻指定的格式。当某一行符合指定的格式时，gawk 就会在此行执行被指定的动作。gawk 依此方式自动处理输入文件的每一行直到输入文件档案结束。

gawk 经常用在如下的几个方面：

- 根据要求选择文件的某几行，几列或部分字段以供显示输出。
- 分析文档中的某一个字出现的频率、位置等。
- 根据某一个文档的信息准备格式化输出。
- 以一个功能十分强大的方式过滤输出文档。
- 根据文档中的数值进行计算。

2.如何执行 gawk 程序

基本上有两种方法可以执行 gawk 程序。

如果 gawk 程序很短，则可以将 gawk 直接写在命令行，如下所示：

gawk 'program' input-file1 input-file2 ...

其中 program 包括一些 pattern 和 action。

如果 gawk 程序较长，较为方便的做法是将 gawk 程序存在一个文件中，

gawk 的格式如下所示：

gawk -f program-file input-file1 input-file2 ...

gawk 程序的文件不止一个时，执行 gawk 的格式如下所示：

gawk -f program-file1 -f program-file2 ... input-file1 input-file2 ...

3.文件、记录和字段

一般情况下，gawk 可以处理文件中的数值数据，但也可以处理字符串信息。如果数据没有存储在文件中，可以通过管道命令和其他的重定向方法给 gawk 提供输入。当然， gawk 只能处理文本文件（A S C I I 码文件)。

电话号码本就是一个 gawk 可以处理的文件的简单例子。电话号码本由很多条目组成，每一个条目都有同样的格式：姓、名、地址、电话号码。每一个条目都是按字母顺序排列。

在 gawk 中，每一个这样的条目叫做一个记录。它是一个完整的数据的集合。例如，电话号码本中的 Smith John 这个条目，包括他的地址和电话号码，就是一条记录。

记录中的每一项叫做一个字段。在 gawk 中，字段是最基本的单位。多个记录的集合组成了一个文件。

大多数情况下，字段之间由一个特殊的字符分开，像空格、TAB、分号等。这些字符叫做字段分隔符。请看下面这个/etc/passwd 文件：

tparker;t36s62hs;501;101;TimParker;/home/tparker;/bin/bash

etreijs;2ys639dj3;502;101;EdTreijs;/home/etreijs;/bin/tcsh

ychow;1h27sj;503;101;Yvonne Chow;/home/ychow;/bin/bash

你可以看出/etc/passwd 文件使用分号作为字段分隔符。/etc/passwd 文件中的每一行都包括七个字段：用户名；口令；用户 I D；工作组 I D；注释； h o m e 目录；启始的外壳。如果你想要查找第六个字段，只需数过五个分号即可。

但考虑到以下电话号码本的例子，你就会发现一些问题：

Smith John 13 Wilson St. 555-1283

Smith John 2736 Artside Dr Apt 123 555-2736

Smith John 125 Westmount Cr 555-1726

虽 然我们能够分辨出每个记录包括四个字段，但 g a w k 却无能为力。电话号码本使用空格作

为分隔符，所以 gawk 认为 Smith 是第一个字段，John 是第二个字段，13 是第三个字段，依次类推。就 gawk 而言，如果用空格作为字段分隔符的话，则第一个记录有六个字段，而第二个记录有八个字段。

所以，我们必须找出一个更好的字段分隔符。例如，像下面一样使用斜杠作为字段分隔符：

Smith/John/13 Wilson St./555-1283

Smith/John/2736 Artside Dr/Apt/123/555-2736

Smith/John/125 Westmount Cr/555-1726

如果你没有指定其他的字符作为字段分隔符,那么 gawk 将缺省地使用空格或 TAB 作为字段分隔符。

4.模式和动作

在 gawk 语言中每一个命令都由两部分组成：一个模式（pattern）和一个相应的动作（action)。只要模式符合，gawk 就会执行相应的动作。其中模式部分用两个斜杠括起来，而动

作部分用一对花括号括起来。例如：

/pattern1/{action1}

/pattern2/{action2}

/pattern3/{action3}

所有的 gawk 程序都是由这样的一对对的模式和动作组成的。其中模式或动作都能够被省略，但是两个不能同时被省略。如果模式被省略，则对于作为输入的文件里面的每一行，动作都会被执行。如果动作被省略，则缺省的动作被执行，既显示出所有符合模式的输入行而不做任何的改动。

下面是一个简单的例子，因为 gawk 程序很短，所以将 gawk 程序直接写在外壳命令行：

gawk ’/tparker/’ /etc/passwd

此程序在上面提到的/etc/passwd 文件中寻找符合 tparker 模式的记录并显示（此例中没有动作，所以缺省的动作被执行）。

让我们再看一个例子：

gawk '/UNIX/{print $2}' file2.data

此命令将逐行查找 file2.data 文件中包含 UNIX 的记录，并打印这些记录的第二个字段。

你也可以在一个命令中使用多个模式和动作对，例如：

gawk '/scandal/{print $1} /rumor/{print $2}' gossip_file

此命令搜索文件 gossip_file 中包括 scandal 的记录，并打印第一个字段。然后再从头搜索 gossip_file 中包括 rumor 的记录，并打印第二个字段。

5.比较运算和数值运算

gawk 有很多比较运算符，下面列出重要的几个：

= = 相等

! = 不相等

> 大于

< 小于

> = 大于等于

< = 小于等于

例如：

gawk '$4 > 100' testfile

将会显示文件 testfile 中那些第四个字段大于 1 0 0 的记录。

下表列出了 gawk 中基本的数值运算符。

运算符说明示例

+ 加法运算 2+6

- 减法运算 6-3

* 乘法运算 2*5

/ 除法运算 8/4

^ 乘方运算 3^2 (=9)

% 求余数 9%4 (=1)

例如：

{print $3/2}

显示第三个字段被 2 除的结果。

在 gawk 中，运算符的优先权和一般的数学运算的优先权一样。例如：

{print $1+$2*$3}

显示第二个字段和第三个字段相乘，然后和第一个字段相加的结果。

你也可以用括号改变优先次序。例如：

{print ($1+$2)*$3}

显示第一个字段和第二个字段相加，然后和第三个字段相乘的结果。

6.内部函数

gawk 中有各种的内部函数，现在介绍如下：

6.1 随机数和数学函数

sqrt(x) 求 x 的平方根

sin(x) 求 x 的正弦函数

cos(x) 求 x 的余弦函数

atan2(x, y) 求 x / y 的余切函数

log(x) 求 x 的自然对数

exp(x) 求 x 的 e 次方

int(x) 求 x 的整数部分

rand() 求 0 和 1 之间的随机数

srand(x) 将 x 设置为 rand()的种子数

6.2 字符串的内部函数

• index(in, find) 在字符串 in 中寻找字符串 find 第一次出现的地方，返回值是字符串 find 出现在字符串 in 里面的位置。如果在字符串 in 里面找不到字符串 find，则返回值为 0。

例如：

print index("peanut", " a n ")

显示结果 3。

• length(string) 求出 string 有几个字符。

例如：

length("abcde")

显示结果 5。

• match(string,regexp) 在字符串 string 中寻找符合 regexp 的最长、最靠左边的子字符串。返回值是 regexp 在 string 的开始位置，即 index 值。match 函数将会设置系统变量 RSTART 等于 index 的值，系统变量 RLENGTH 等于符合的字符个数。如果不符合，则会设置 RSTART 为 0、RLENGTH 为- 1。

• sprintf(format, expression1, . . .) 和 printf 类似，但是 sprintf 并不显示，而是返回字符串。

例如：

sprintf("pi = %.2f (approx.)", 2 2 / 7)

返回的字符串为 pi = 3.14 (approx.)

• s u b (r e g e x p，r e p l a c e m e n t，t a rg e t) 在字符串 t a rget 中寻找符合 regexp 的最长、最靠左的

地方，以字串 replacement 代替最左边的 r e g e x p。

例如：

str = "water，water，everywhere"

sub(/at/， "ith"，str)

结果字符串 str 会变成 wither，water，everywhere

• gsub(regexp,replacement,target) 与前面的 sub 类似。在字符串 target 中寻找符合 regexp 的所有地方，以字符串 replacement 代替所有的 regexp。

例如：

str = "water，water，everywhere"

gsub(/at/， "ith"，str)

结果字符串 str 会变成 wither，wither，everywhere

• substr(string，start，length)返回字符串 string 的子字符串，这个子字符串的长度为 length，从第 start 个位置开始。

例如：

substr("washington"，5，3)

返回值为 ing

如果没有 length，则返回的子字符串是从第 start 个位置开始至结束。

例如：

substr("washington"，5)

返回值为 ington。

• tolower(string) 将字符串 s t r i n g 的大写字母改为小写字母。

例如：

tolower("MiXeD cAsE 123")

返回值为 mixed case 123。

• toupper(string) 将字符串 s t r i n g 的小写字母改为大写字母。

例如：

toupper("MiXeD cAsE 123")

返回值为 MIXED CASE 123。

6.3 输入输出的内部函数

• close(filename) 将输入或输出的文件 filename 关闭。

• system(command) 此函数允许用户执行操作系统的指令，执行完毕后将回到 gawk 程序。

例如：

BEGIN {system("ls")}

7.字符串和数字

字符串就是一连串的字符，它可以被 gawk 逐字地翻译。字符串用双引号括起来。数字不能用双引号括起来，并且 gawk 将它当作一个数值。例如：

gawk '$1 != "Tim" {print}' testfile

此命令将显示第一个字段和 Tim 不相同的所有记录。如果命令中 Tim 两边不用双引号，gawk 将不能正确执行。

再如：

gawk '$1 == "50" {print}' testfile

此命令将显示所有第一个字段和 5 0 这个字符串相同的记录。g a w k 不管第一字段中的数值的大小，而只是逐字地比较。这时，字符串 5 0 和数值 5 0 并不相等。

8.格式化输出

我们可以让动作显示一些比较复杂的结果。例如：

gawk '$1 != "Tim" {print $1, $ 5, $ 6, $2}' testfile

将显示 testfile 文件中所有第一个字段和 Ti m 不相同的记录的第一、第五、第六和第二个字段。

进一步，你可以在 p r i n t 动作中加入字符串，例如：

gawk '$1 != "Tim" {print "The entry for ", $ 1, "is not Tim. ", $2}' testfile

print 动作的每一部分用逗号隔开。

借用 C 语言的格式化输出指令，可以让 gawk 的输出形式更为多样。这时，应该用 printf 而不是 print。例如：

{printf "%5s likes this language\n", $ 2 }

printf 中的%5s 部分告诉 gawk 如何格式化输出字符串，也就是输出 5 个字符长。它的值由 printf 的最后部分指出，在此是第二个字段。\n 是回车换行符。如果第二个字段中存储的是人名，则输出结果大致如下：

Tim likes this language

Geoff likes this language

Mike likes this language

Joe likes this language

gawk 语言支持的其他格式控制符号如下：

• c 如果是字符串，则显示第一个字符；如果是整数，则将数字以 ASCII 字符的形式显示。例如：

printf “% c”, 65

结果将显示字母 A。

• d 显示十进制的整数。

• i 显示十进制的整数。

• e 将浮点数以科学记数法的形式显示。

例如：

print “$ 4 . 3 e”, 1950

结果将显示1.950e+03。

• f 将数字以浮点的形式显示。

• g 将数字以科学记数法的形式或浮点的形式显示。数字的绝对值如果大于等于 0 . 0 0 0 1 则

以浮点的形式显示，否则以科学记数法的形式显示。

• o 显示无符号的八进制整数。

• s 显示一个字符串。

• x 显示无符号的十六进制整数。1 0至1 5以a至f表示。

• X 显示无符号的十六进制整数。1 0至1 5以A至F表示。

• % 它并不是真正的格式控制字符，% %将显示%。

当你使用这些格式控制字符时，你可以在控制字符前给出数字，以表示你将用的几位或几个字符。例如，6 d表示一个整数有6位。再请看下面的例子：

{printf "%5s works for %5s and earns %2d an hour", $1, $2, $3}

将会产生类似如下的输出：

Joe works for Mike and earns 12 an hour

当处理数据时，你可以指定数据的精确位数

{printf "%5s earns $%.2f an hour", $ 3, $ 6 }

其输出将类似于：

Joe earns $12.17 an hour

你也可以使用一些换码控制符格式化整行的输出。之所以叫做换码控制符，是因为 gawk 对这些符号有特殊的解释。下面列出常用的换码控制符：

\a 警告或响铃字符。

\b 后退一格。

\f 换页。

\n 换行。

\r 回车。

\t Ta b。

\v 垂直的 t a b。

9.改变字段分隔符

在 g a w k 中，缺省的字段分隔符一般是空格符或 TA B。但你可以在命令行使用- F 选项改变字符分隔符，只需在- F 后面跟着你想用的分隔符即可。

gawk -F″ ;″’/tparker/{print}’ /etc/passwd

在此例中，你将字符分隔符设置成分号。注意： - F 必须是大写的，而且必须在第一个引号之前。

10.元字符

gawk 语言在格式匹配时有其特殊的规则。例如， cat 能够和记录中任何位置有这三个字符的字段匹配。但有时你需要一些更为特殊的匹配。如果你想让 cat 只和 concatenate 匹配，则需要在格式两端加上空格：

/ cat / {print}

再例如，你希望既和 cat 又和 CAT 匹配，则可以使用或（|）：

/ cat | CAT / {print}

在 gawk 中，有几个字符有特殊意义。下面列出可以用在 gawk 格式中的这些字符：

• ^ 表示字段的开始。

例如：

$3 ~ /^b/

如果第三个字段以字符 b 开始，则匹配。

• $ 表示字段的结束。

例如：

$3 ~ /b$/

如果第三个字段以字符 b 结束，则匹配。

• . 表示和任何单字符 m 匹配。

例如：

$3 ~ /i.m/

如果第三个字段有字符 i，则匹配。

• | 表示“或”。

例如：

/ c a t | C AT/

和 cat 或 C AT 字符匹配。

• * 表示字符的零到多次重复。

例如：

/UNI*X/

和 U N X、U N I X、U N I I X、U N I I I X 等匹配。

• + 表示字符的一次到多次重复。

例如：

/UNI+X/

和ＵＮＩＸ、ＵＮＩＩＸ等匹配。

• \{a，b\} 表示字符ａ次到ｂ次之间的重复。

例如：

/ＵＮＩ\｛1，3\｝Ｘ

和ＵＮＩＸ、ＵＮＩＩＸ和ＵＮＩＩＩＸ匹配。

• ？表示字符零次和一次的重复。

例如：

/UNI?X/

和UNX 和ＵＮＩＸ匹配。

• [] 表示字符的范围。

例如：

/I[BDG]M/

和ＩＢＭ、ＩＤＭ和ＩＧＭ匹配

• [^] 表示不在[ ]中的字符。

例如：

/I[^DE]M/

和所有的以Ｉ开始、Ｍ结束的包括三个字符的字符串匹配，除了 IDM 和 IEM 之外。

11.调用 gawk 程序

当需要很多对模式和动作时，你可以编写一个 gawk 程序（也叫做 gawk 脚本）。在 gawk 程序中，你可以省略模式和动作两边的引号，因为在 gawk 程序中，模式和动作从哪开始和从哪结束时是很显然的。

你可以使用如下命令调用ｇａｗｋ程序：

gawk -f scrīpt filename

此命令使 gawk 对文件 filename 执行名为 scrīpt 的 gawk 程序。

如果你不希望使用缺省的字段分隔符,你可以在 f 选项后面跟着 F 选项指定新的字段分隔符(当然你也可以在 gawk 程序中指定),例如,使用分号作为字段分隔符:

```
gawk -f scrīpt -F";" filename
```

如果希望 gawk 程序处理多个文件,则把各个文件名罗列其后:

```
gawk -f scrīpt filename1 filename2 filename3 ...
```

缺省情况下, gawk 的输出将送往屏幕。但你可以使用 Linux 的重定向命令使 gawk 的输出送往一个文件:

```
gawk -f scrīpt filename > save_file
```

12.BEGIN 和 END

有两个特殊的模式在 gawk 中非常有用。BEGIN 模式用来指明 gawk 开始处理一个文件之前执行一些动作。BEGIN 经常用来初始化数值,设置参数等。END 模式用来在文件处理完成后执行一些指令,一般用作总结或注释。

BEGIN 和 END 中所有要执行的指令都应该用花括号括起来。BEGIN 和 END 必须使用大写。

请看下面的例子:

```
BEGIN { print "Starting the process the file" }
$1 == "UNIX" {print}
$2 > 10 {printf "This line has a value of %d", $ 2 }
END { print "Finished processing the file. Bye!"}
```

此程序中,先显示一条信息: Starting the process the file,然后将所有第一个字段等于 UNIX 的整条记录显示出来,然后再显示第二个字段大于 10 的记录,最后显示信息: Finished processing the file. Bye!

13.变量

在 gawk 中，可以用等号(=)给一个变量赋值：

var1=10

在 gawk 中，你不必事先声明变量类型。

请看下面的例子：

$1 == "Plastic" { count = count + 1 }

如果第一个字段是 Plastic，则 count 的值加 1。在此之前，我们应当给 count 赋予过初值，一般是在 BEGIN 部分。

下面是比较完整的例子：

BEGIN { count = 0 }

$5 == "UNIX" { count = count + 1 }

END { printf "%d occurrences of UNIX were found", count }

变量可以和字段和数值一起使用，所以，下面的表达式均为合法：

count = count + $6

count = $5 - 8

count = $5 + var1

变量也可以是格式的一部分，例如：

$2 > max_value {print "Max value exceeded by ", $2 - max_value}

$4 - var1 < min_value {print "Illegal value of ", $ 4 }

14.内置变量

gawk 语言中有几个十分有用的内置变量，现在列于下面：

NR 已经读取过的记录数。

FNR 从当前文件中读出的记录数。

FILENAME 输入文件的名字。

FS 字段分隔符（缺省为空格）。

RS 记录分隔符（缺省为换行）。

OFMT 数字的输出格式（缺省为% g）。

OFS 输出字段分隔符。

ORS 输出记录分隔符。

NF 当前记录中的字段数。

如果你只处理一个文件，则 NR 和 FNR 的值是一样的。但如果是多个文件， NR 是对所有的文件来说的，而 FNR 则只是针对当前文件而言。

例如：

NR <= 5 {print "Not enough fields in the record"}

检查记录数是否小于 5，如果小于 5，则显示出错信息。

FS 十分有用，因为 FS 控制输入文件的字段分隔符。例如，在 BEGIN 格式中，使用如下的命令：

F S = " : "

15. 控制结构

15.1 if 表达式

if 表达式的语法如下：

```
if (expression){
c o m m a n d s
}
e l s e {
```

```
c o m m a n d s
}
```

例如：

```
# a simple if loop
(if ($1 == 0){
print "This cell has a value of zero"
}
else {
printf "The value is %d\n", $ 1
} )
```

再看下一个例子：

```
# a nicely formatted if loop
(if ($1 > $2){
print "The first column is larger"
}
else {
print "The second column is larger"
} )
```

15.2 while 循环

while 循环的语法如下：

```
while (expression){
c o m m a n d s
}
```

例如：

```
# interest calculation computes compound interest
# inputs from a file are the amount, interest_rateand years
{var = 1
while (var <= $3) {
printf(" %f\n", $1*(1+$2)^var)
var++}
}
```

15.3 for 循环

for 循环的语法如下：

```
for (initialization; expression; increment) {
c o m m a n d
}
```

例如：

```
# interest calculation computes compound interest
# inputs from a file are the amount, interest_rateand years
{for (var=1; var <= $3; var++) {
printf(" %f\n", $1*(1+$2)^var)
}
}
```

15.4 next 和 exit

next 指令用来告诉 gawk 处理文件中的下一个记录， 而不管现在正在做什么。语法如下：

```
{ command1
c o m m a n d 2
c o m m a n d 3
next
c o m m a n d 4
}
```

程序只要执行到 n e x t 指令，就跳到下一个记录从头执行命令。因此，本例中， c o m m a n d 4 指令永远不会被执行。

程序遇到 exit 指令后，就转到程序的末尾去执行 END，如果有 END 的话。

16.数组

gawk 语言支持数组结构。数组不必事先初始化。声明一个数组的方法如下：

```
a r r a y n a m e [ n u m ] = v a l u e
```

请看下面的例子：

```
# reverse lines in a file
{line[NR] = $0 } # remember each line
END {var=NR # output lines in reverse order
while (var > 0){
print line[var]
v a r - -
}
}
```

此段程序读取一个文件的每一行，并用相反的顺序显示出来。我们使用 NR 作为数组的下标来存储文件的每一条记录，然后在从最后一条记录开始，将文件逐条地显示出来。

17.用户自定义函数

复杂的 gawk 程序常常可以使用自己定义的函数来简化。调用用户自定义函数与调用内部函数的方法一样。函数的定义可以放在 gawk 程序的任何地方。

用户自定义函数的格式如下：

```
function name (parameter-list) {
b o d y - o f - f u n c t i o n
}
```

name 是所定义的函数的名称。一个正确的函数名称可包括一序列的字母、数字、下标线(underscores)，但是不可用数字做开头。parameter- list 是函数的全部参数的列表，各个参数之间以逗点隔开。body-of-function 包含 gawk 的表达式，它是函数定义里最重要的部分，它决定函数实际要做的事情。

下面这个例子，会将每个记录的第一个字段的值的平方与第二个字段的值的平方加起来。

```
{print "sum =", S q u a r e S u m ( $ 1, $ 2 ) }
function SquareSum(x, y) {
s u m = x * x + y * y
return sum
}
```

到此，我们已经知道了 gawk 的基本用法。gawk 语言十分易学好用，例如，你可以用 gawk 编写一段小程序来计算一个目录中所有文件的个数和容量。如果用其他的语言，如 C 语言，则会十分的麻烦，相反，gawk 只需要几行就可以完成此工作。

18.几个实例

最后，再举几个 gawk 的例子：

```
gawk '{if (NF > max) max = NF}
END {print max}'
```

此程序会显示所有输入行之中字段的最大个数。

```
gawk 'length($0) > 80'
```

此程序会显示出超过 80 个字符的每一行。此处只有模式被列出，动作是采用缺省值显示整个记录。

```
gawk 'NF > 0'
```

显示拥有至少一个字段的所有行。这是一个简单的方法，将一个文件里的所有空白行删除。

```
gawk 'BEGIN {for (i = 1; i <= 7; i++)
print int(101 * rand())}'
```

此程序会显示出范围是 0 到 100 之间的 7 个随机数。

```
ls -l files | gawk '{x += $4}; END {print "total bytes: " x}'
```

此程序会显示出所有指定的文件的总字节数。

```
expand file | gawk '{if (x < length()) x = length()}
END {print "maximum line length is " x}'
```

此程序会将指定文件里最长一行的长度显示出来。expand 会将 tab 改成 space，所以是用实际的右边界来做长度的比较。

```
gawk 'BEGIN {FS = ":"}
{print $1 | "sort"}' /etc/passwd
```

此程序会将所有用户的登录名称，依照字母的顺序显示出来。

```
gawk '{nlines++}
END {print nlines}'
```

此程序会将一个文件的总行数显示出来。

```
gawk 'END {print NR}'
```

此程序也会将一个文件的总行数显示出来，但计算行数的工作由 gawk 来做。

gawk '{print NR, $ 0 }'

此程序显示出文件的内容时，会在每行的最前面显示出行号，它的函数与'cat -n'类似。