《Unix Shell 实例精解》学习笔记

By

Turner

第一章 关于 UNIX Shell 的介绍

1. 定义

   shell 是一种特殊的程序，被用作用户与内核(kernel)的 UNIX 操作系统核心通讯。常见的 shell 有 C shell、B shell 和 Korn shell（B shell 的超集）。

2. shell 主要功能

   - 解释交互运行时，在命令行提示下敲入的命令
   - 制订用户环境，通常在 shell 初始化文件中作这种工作。例如：设置终端键及窗口特征；设置搜索路径、权限、提示等
   - shell 可以用作解释编程语言。Shell 程序也叫命令表，由再文件中列出的命令组成。

3. 主要 shell 命令：

   who mv rm wc ls cat date at
   lpr rsh more awk pwd bc vi finger w
   pg find cc cp dd grep ksh sh
   ps sed cal lp

第 2 章 UNIX 工具箱

1．正则表达式

一个正则表达式就是用来在一次搜索中匹配相同字符的一个字符模式。在大多数程序中，把一个正则表达式封装在正斜杠(/)里。

2．正则表达式元字符

元字符	功能	实例	解释
^	行开头定位	/^love/	与所有 love 开头的行匹配
$	行末尾定位	/love$/	与所有 love 结尾的行匹配
.	匹配单个字符	/l..e/	与包含一个 l，后跟两个字符，然后跟一个 e 的行相匹配
*	跟前驱的 0 个或多个字符相匹配	/ *love/	跟 0 个或多个空格后面的 love 模式的行相匹配
[]	与其中的一个相匹配	/[Ll]ove/	与包含 love 或者 Love 的行匹配
[x-z]	与集中一个范围内的一个字符相匹配	/[A-Z]ove/	与后面跟 ove 的从 A 到 Z 的字相匹配
[^]	与不在集里的字符匹配	/[^A-Z]ove/	不包括 A 到 Z，后面跟 ove 的字相匹配
\	给一个元字符转移	/love\./	匹配行包括 love，跟一个句点
许多用 RE 元字符的 UNIX 程序支持的附加元字符(vi 和 grep 支持)			
\<	词开头定位	/\<love/	匹配行包含用 love 开头的词
\>	词结尾定位	/love\>/	匹配行包含 love 结尾的词
\(..\)	标志与以后用的字符相匹配	/(love\)able\ler/	Able 可达 9 个标志，模式最左边用第一个标志开始。例如，模式 love 保存作标志 1，以后引用作\1;在这个例子中，搜索模式包括后面跟 lover 的 lovable
x\{m\} x\{m,\} x\{m,n\}	字符 x 重复 m 次 至少 m 次 m 到 n 次	0\{5,10\}	如果行包含 5—10 个连续的 o 则匹配

表(2.1)

3．举例

查找文件中的含有 love 的词：

```
% vi picnic
    I had a lovely time on our little picnic. Lovers were all around
  us, oh love
  :/love/
```

4．组合正则表达式元字符

文件内容：(数字是行号，竖线标明行的左右边界，不属于文件内容。这个文

件我写在 shell/exam/exam_2.9 中)

```
1  |Christian Scott lives here and will put on a Christmas party.|
2  |There are around 30 to 35 people invited.|
3  |They are: |
4  |                              Tom|
5  |Dan|
6  |     Rhonda Savage|
7  |Nicky and Kimerly.|
8  |Steve, Suzanne, Ginger and Larry.|
```

组合举例：

- /^[A-Z]..$/
  搜索行以 A 至 Z 的一个字母开头，然后跟两个任意字母，然后跟一个换行符的行。将找到第 5 行。
- /^[A-Z][a-z]*3[0-5]/
  搜索以一个大写字母开头，后跟 0 个或多个小写字母，再跟数字 3，再跟 0—5 之间的一个数字。
- /^ *[A-Z][a-z][a-z]$/
  搜索以 0 个或多个空格开头，跟一个大写字母，两个小写字母和一个换车符。将找到第 4 行的 TOM（整行匹配）和第 5 行。注意，*前面有一个空格。
- /^[A-Za-z]*[^,][A-Za-z]*$/
  将查找以 0 个或多个大写或小写字母开头，不跟逗号，然后跟 0 个或多个大写或小写字母，然后跟一个换车符。将找到第 5 行。书中解释有误。

5. 更多的正则表达式元字符
这里讨论的元字符不一定可以移植到所有的正则表达式中，但一般可以用在 vi、sed 和 grep 中。

```
% vi textfile
```

```
Patty won fourth place in the 50 yard dash square and fair.
Occurences like this are rare.
Haha, what you want is just fourth.
~
~
```

```
:/\<fourth\>/
```

将查找词 fourth

第 3 章 grep 家族

1. grep 的含义是“全局搜索正则表达式(RE)并打印该行”
2. grep 支持的正则表达式
   与在文件中搜索基本一样。可以参考表 2.1。
3. grep 的选项

选项	功能
-b	在各行之前放置它发现的块号。有时在根据上下文定位磁盘字块时有用
-c	显示匹配行数而不是内容
-h	不显示文件名
-I	在座比较时忽略字母大小写
-n	文件中每行之前给出它的相关行号
-s	无声操作。即除了错误消息外不做任何显示。用于检查退出状态
-v	把搜索翻转为只显示不匹配的行
-w	把表达式当作一个次来搜索，相当于用\<和\>括起来

表 3.1

4. grep 命令的退出状态
   如果 grep 操作成功，则状态是 0，如果模式没找到，状态是 1，如果文件没找到，状态是 2。如果操作被取消，则状态是 130。查看状态的方法：在 csh 中用 echo $status。在 sh 和 ksh 中用 echo $? 。例如
   $ echo $?
   0
5. 带正则表达式的 grep 举例：
   用于这些例子的文件叫 datafile，位于 chap03 目录。内容如下：

```
northwest       NW      Charles Main        3.0     .98     3       34
western         WE      Sharon Gray         5.3     .97     5       23
southwest       SW      Lewis Dalsass       2.7     .8      2       18
southern        SO      Suan Chin           5.1     .95     4       15
southeast       SE      Patricia Hemenway   4.0     .7      4       17
eastern         EA      TB Savage           4.4     .84     5       20
northeast       NE      AM Main Jr.         5.1     .94     3       13
north           NO      Margot Weber        4.5     .89     5        9
central         CT      Ann Stephens        5.7     .94     5       13
```

   1) grep NW datafile
      解释：打印 datafile 中包含 NW 的行
   2) grep NW d*
      解释：打印所有以 d 开头的文件中含有 NW 的文件。
   3) grep ’^n’ datafile
      解释：打印文件 datafile 中所有以字母 n 开头的行
   4) grep TB Savage datafile
      解释：在 Savage 和 datafile 文件中查找有 TB 的行
   5) grep ’TB Savage’ datafile
      解释：在 datafile 文件中查找含有 TB Savage 的行并打印。这个例子在

书中有误。

6) grep '^[we]' datafile

打印 datafile 中以 w 或者 e 开头的行

7) grep 'ss* ' datafile

```
northwest       NW      Charles Main            3.0     .98     3       34
southwest       SW      Lewis Dalsass           2.7     .8      2       18
```

解释：打印所有包含一个 s 并跟 0 个或者多个 s，然后跟一个空格的行

6. 用管道的 grep

grep 可以从管道得到输入。

```
% ls -l
drwxr-xr-x   6 oracle   dba          512  4 月  3 21:49 chap10
drwxr-xr-x   2 oracle   dba          512  4 月 10 22:23 exam
-rwxr--r--   1 oracle   dba         1842  4 月  3 21:51 readme.txt
-rwxr--r--   1 oracle   dba      1801  4 月  3 21:51 unix_readme.txt
% ls -l | grep '^d'
drwxr-xr-x   6 oracle   dba          512  4 月  3 21:49 chap10
drwxr-xr-x   2 oracle   dba          512  4 月 10 22:23 exam
% ls -l |grep '^[^d]'
-rwxr--r--   1 oracle   dba         1842  4 月  3 21:51 readme.txt
-rwxr--r--   1 oracle   dba      1801  4 月  3 21:51 unix_readme.txt
```

7. 带选项的 grep 举例：

```
grep -c 'west' datafile
3
```

解释计算 datafile 中含有 west 的总数。

8. egrep（扩展的 grep）

egrep 可以使用额外的正则表达式，如下表。

元字符	功能	例子	解释
+	匹配一个或多个前驱字符	'[a-z]+ove'	匹配一个或多个小写字母，后跟 ove
?	匹配 0 个或者 1 个前驱字符	'lo?ve'	将找到 love 或 love
a|b	匹配 a 或者 b	'love|hate'	与 love 或 hate 匹配
()	组字符	'lov(ely|able)	与 lovely 或 lovable 匹配

表 3.2

9. egrep 举例：

```
egrep '2\.?[0-9]' datafile
```

解释：打印所有这样的行：它包含一个 2，后跟 0 个或者一个句号，然后跟一个数字。

10. fgrep

fgrep 把所有的元字符都当作字符本身，只代表自己。

11. UNIX 工具试验参考答案（内容参考 datebook）

✓ 打印包含 San 的行

```
grep 'San' datebook
```

- ✓ 打印所有以 J 开头的人名所在的行
  grep '^J' datebook
- ✓ 打印以 700 结尾的行
  grep '700$' datebook
- ✓ 打印所有不包含 834 的行
  grep -v '834' datebook
- ✓ 打印出生在 12 月(December)的行
  grep '/12' datebook
- ✓ 打印工资是 6 位数的行，并给出行号
  grep -n '[0-9]\{6,\}$' datebook

第 4 章 流编辑器（sed）

1. sed 命令简介

sed 是流线型、非交互式编辑器。它允许你执行与 vi 和 ex 编辑器里一样的编辑任务。Sed 程序不是与编辑器交互式工作的，而是让你在命令行里敲入编辑的命令，给文件命名，然后在屏幕上查看命令输出结果。

2. sed 如何工作

sed 编辑器按一次处理一行的方式来处理文件，并把输出送到屏幕上。

3. sed 可以用寻址的方式来决定想要编辑哪一行。

4. sed 命令和选项

命令	功能
a\	在当前行上添加一个文本行或者多个文本行
c\	用新闻本改变（取代）当前行里的文本
d	删除行
i\	在当前行之前插入文本
h	把模式空间内容复制到一个固定缓存
H	把模式空间内容添加到一个固定缓存
g	得到固定缓存里所有的禀复制到模式缓存，重写其内容
G	得到固定缓存的内容并复制到模式缓存，添加到里面
I	列出不打印的字符
p	打印行
n	读下一输入行，并开始用下一个命令处理换行符，而不是用第一个命令
q	结束或退出 sed
r	从一个文件读如行
!	把命令应用到除了选出的行以外的其他所有行
s	把一个字串替换成另一个
替换标志	
g	在一行上进行全局替换
p	打印行
w	把行写到一个文件中
x	用模式空间的内容交换固定缓存的内容
y	把一个字符转换成另一个（不能和整则表达式元字符一起使用）

5. sed 元字符

基本上，grep 和 vi 使用的元字符都可以用在 sed 中。具体可参照第二章。

下表列出了一些特别的 sed 元字符：

元字符	功能	例子	解释
&	保存搜索串以便可以记在替换串里	s/love/**&**/	&号代表搜索串。串 love 将被星号包围的自身所替代；即 love 变成**love**

6. sed 的实例(使用 datafile)

1> 打印：p 命令

```
sed '/north/p' datafile
```

默认输出所有行，找到 north 的行重复打印

```
sed –n '/north/p' datafile
```

禁止默认输出，只打印找到 north 的行

2> 删除：d 命令

```
sed '3d' datafile
```

删除第三行，其余行输出到屏幕

```
sed '3,$d' datafile
```

从第 3 行到最后一行都删除，将剩余部分输出到屏幕

```
sed '/north/d' datafile
```

将含有 north 的行删除，其余输出到屏幕

3> 替换：s 命令

```
sed 's/west/north/g' datafile
```

解释：找到 datafile 中的所有 west 并替换成 north，将替换后的内容输出到屏幕。

```
sed 's/[0-9][0-9]$/&.5/' datafile
```

解释：在替代串里的&字符代表在搜索串中真正找到的。每个以两个数字结尾的行都被它自己取代，且要在后面加上.5

```
sed -n 's/Hemenway/Jones/gp' datafile
```

解释：所有的 Hemenway 所在的位置都用 Jones 来取代，而且只有改变的行被打印。-n 与 p 命令选项相结合来禁止默认输出。g 代表全局替换

```
sed -n 's/\(Mar\)got/\1ianne/p' datafile
```

解释：模式 Mar 被封装在括弧里且在一个专用寄存器里存为标记 1。在替换串里它将被引用做\1。然后用 Marianne 替代 Margot。

```
sed 's#3#88#g' datafile
```

s 命令后面的字符是搜索串和替换串之间的分界符。默认的分界符是一个正斜杠，但也可以改变（只有使用 s 命令时）。无论 s 命令后面跟什么字符，它都是新的串分界符。当搜索包含一个正斜杠的模式，如路径或生日时，这种技巧可能有用

4> 被选中的行的范围：逗号

```
sed -n '/west/,/east/p' datafile
```

打印在 west 和 east 之间的模式范围内所有行。如果 west 出现在 east 之后，则打印从 west 到下一个 east 或者到文件末尾的行，无论哪种情况先出现都可以。

```
sed '/west/,/east/s/$/**VACA**/' datafile
```

解释：对于在模式 west 到 east 范围内的行，行末尾将用**VACA**来取代。

5> 多次编辑 -e 选项

```
sed -e '1,3d' -e 's/Hemenway/Jones/' datafile
```

-e 选项允许多次编辑。不同的编辑顺序可能导致不同的结果。例如，如果两个命令都执行了替换，第一次替换可能影响第二次替换。

6> 从文件中读取：r 命令

```
sed '/Suan/r newfile' datafile
```

解释：r 命令从 newfile 中读取内容，将内容输出到 Suan 的后面。如果

datafile 中 Suan 出现的次数不只一次，则分别放到 Suan 的后面。

7> 写入文件：w 命令

sed -n '/north/w newfile' datafile

解释：w 命令把指定的行写入到一个文件。本例中所有的包含 north 的行写入到 newfile 中。等同于 sed -n '/north/p' datafile >newfile

8> 添加：a 命令

```
$ sed '/north/a\
> ---->THE NORTH SALES DISTRICT HAS MOVED<-----' datafile
northwest       NW      Charles Main          3.0     .98     3       34
---->THE NORTH SALES DISTRICT HAS MOVED<-----
western         WE      Sharon Gray           5.3     .97     5       23
southwest       SW      Lewis Dalsass         2.7     .8      2       18
southern        SO      Suan Chin             5.1     .95     4       15
southeast       SE      Patricia Hemenway     4.0     .7      4       17
eastern         EA      TB Savage             4.4     .84     5       20
northeast       NE      AM Main Jr.           5.1     .94     3       13
---->THE NORTH SALES DISTRICT HAS MOVED<-----
north           NO      Margot Weber          4.5     .89     5        9
---->THE NORTH SALES DISTRICT HAS MOVED<-----
central         CT      Ann Stephens          5.7     .94     5       13
```

解释：红颜色的内容是要输入的内容。a\命令后面跟要添加的内容。奇怪的是 a\后面必须另起一行，在输入要添加的内容，否则会提示命令错乱，真是搞不懂。

9> 插入：i 命令

```
$ sed '/north/i\
> ---->THE NORTH SALES DISTRICT HAS MOVED<-----' datafile
---->THE NORTH SALES DISTRICT HAS MOVED<-----
northwest       NW      Charles Main          3.0     .98     3       34
western         WE      Sharon Gray           5.3     .97     5       23
southwest       SW      Lewis Dalsass         2.7     .8      2       18
southern        SO      Suan Chin             5.1     .95     4       15
southeast       SE      Patricia Hemenway     4.0     .7      4       17
eastern         EA      TB Savage             4.4     .84     5       20
---->THE NORTH SALES DISTRICT HAS MOVED<-----
northeast       NE      AM Main Jr.           5.1     .94     3       13
---->THE NORTH SALES DISTRICT HAS MOVED<-----
north           NO      Margot Weber          4.5     .89     5        9
central         CT      Ann Stephens          5.7     .94     5       13
```

解释：在符合模式的行前面插入内容。其余和 a\命令相同。

10>下一个：n 命令

```
$ sed '/eastern/{n;s/AM/Archie/;}' datafile
northwest       NW      Charles Main          3.0     .98     3       34
western         WE      Sharon Gray           5.3     .97     5       23
southwest       SW      Lewis Dalsass         2.7     .8      2       18
southern        SO      Suan Chin             5.1     .95     4       15
```

```
southeast       SE      Patricia Hemenway       4.0     .7      4       17
eastern         EA      TB Savage               4.4     .84     5       20
northeast       NE      Archie Main Jr.         5.1     .94     3       13
.....
```

解释：如果在某一行里模式 eastern 被匹配，n 命令使 sed 区的下一行，用该行带换模式空间，用 Archie 替换 AM，打印并继续。

11>变换：y 命令

```
sed '1,3y/abcdefghijklmnopqrst/ABCDEFGHIJKLMNOPQRST/' datafile
```

解释将对应字母进行转换。

12>退出：q 命令

```
sed '5q' datafile
```

解释：在打印了 5 行之后，用 q 命令退出 sed 程序。

13>保存和取得：h 和 G 命令

```
$ sed -e '/southeast/h' -e '$G' datafile
northwest       NW      Charles Main            3.0     .98     3       34
western         WE      Sharon Gray             5.3     .97     5       23
southwest       SW      Lewis Dalsass           2.7     .8      2       18
southern        SO      Suan Chin               5.1     .95     4       15
southeast       SE      Patricia Hemenway       4.0     .7      4       17
eastern         EA      TB Savage               4.4     .84     5       20
northeast       NE      AM Main Jr.             5.1     .94     3       13
north           NO      Margot Weber            4.5     .89     5       9
central         CT      Ann Stephens            5.7     .94     5       13
southeast       SE      Patricia Hemenway       4.0     .7      4       17
```

解释：当 sed 处理文件时，每行都存在模式空间(pattern space)的临时缓存中。除非行被禁止打印或删除,否则行将在处理完后被打印到屏幕,然后请模式空间并把下一输入行保存在那里等待处理。在这个例子中，在找到模式之后，把它放在模式空间里，而且 h 命令复制它并把它存到另一个叫做保存缓存(holding buffer)中。第二个 sed 指令里，当读入最后一行($)时,G 命令告诉 sed 从包存缓存中取得该行并放回模式空间缓存，添加到当前存在那里的行中。本例子就是最后一行。

```
$ sed -e '/WE/{h;d;}' -e '/CT/G' datafile
northwest       NW      Charles Main            3.0     .98     3       34
southwest       SW      Lewis Dalsass           2.7     .8      2       18
southern        SO      Suan Chin               5.1     .95     4       15
southeast       SE      Patricia Hemenway       4.0     .7      4       17
eastern         EA      TB Savage               4.4     .84     5       20
northeast       NE      AM Main Jr.             5.1     .94     3       13
north           NO      Margot Weber            4.5     .89     5       9
central         CT      Ann Stephens            5.7     .94     5       13
western         WE      Sharon Gray             5.3     .97     5       23
```

解释：第一个命令 h 将找到了 WE 的行放到保存缓存中，然后删除该行；第二个命令 /CT/G 就是在找到了 CT 的行的后面加入保存缓存的内容。

14>G 和 g 的区别

G 命令在符合的条件行后面添加保存缓存中的内容；g 命令用保存缓存中

的内容覆盖符合条件的行。

15>sed 命令的花括号{}的作用

花括号{}中可以放入多个命令，每个命令后面要用分号;。

16>保存和交换：h 和 x 命令。

```
$ sed -e '/Patricia/h' -e '/Margot/x' datafile
northwest       NW      Charles Main            3.0     .98     3       34
western         WE      Sharon Gray             5.3     .97     5       23
southwest       SW      Lewis Dalsass           2.7     .8      2       18
southern        SO      Suan Chin               5.1     .95     4       15
southeast       SE      Patricia Hemenway       4.0     .7      4       17
eastern         EA      TB Savage               4.4     .84     5       20
northeast       NE      AM Main Jr.             5.1     .94     3       13
southeast       SE      Patricia Hemenway       4.0     .7      4       17
central         CT      Ann Stephens            5.7     .94     5       13
```

解释：x 命令将找到的行用保存缓存中的内容替换。

7. 用 sed 来编写命令表

sed 命令表(script)是文件里的一个 sed 命令列表。用-f 选项来引用一个命令表文件。编辑 sed 命令表有特殊要求：命令末尾不能有任何为岁的空白符或者文本。如果命令不是自成一行，就必须用分号结束。在源代码 chap4 目录下有两个编辑好的命令表文件(sedding1 和 sedding2)可以参考。下面是使用 sed 命令表的例子。

```
$ sed -f sedding1 datafile
EMPLOYEE DATABASE
---------------------
northwest       NW      Charles Main            3.0     .98     3       34
western         WE      Sharon Gray             5.3     .97     5       23
southwest       SW      Lewis Dalsass           2.7     .8      2       18
        Lewis is the TOP Salesperson for April!!
        Lewis is moving to the southern district next month.
        CONGRATULATIONS!
southern        SO      Suan Chin               5.1     .95     4       15
southeast       SE      Patricia Hemenway       4.0     .7      4       17
eastern         EA      TB Savage               4.4     .84     5       20
northeast       NE      AM Main Jr.             5.1     .94     3       13
        ********************
        MARGOT HAS RETIRED
        ********************
```

8. Sed 练习参考答案

练习内容参考 databook 文件

1〉 把 Jon 改成 Jonathan

```
sed 's/Jon/Jonathan/' datebook
```

2〉 删除头 3 行

```
sed '1,3d' datebook
```

3〉 打印 5—10 行

```
sed -n '5,10p' datebook
```

4〉 删除包含 Lane 的行

```
sed '/Lane/d' datebook
```

5〉 打印所有生日是在 Noverber 到 December 之间的行

```
sed -n '/:1[12]\//p' datebook
```

6〉 把三个星添加到以 Fred 开头的行尾

```
sed '/^Fred/s/$/***/' datebook
```

7〉 用 JOSE HAS RETIRED 取代包含 Jose 的行

```
sed 's/^Jose[0-9]*[a-z]*[A-Z]* *.*$/JOSE HAS LEFT/' datebook
```

8〉 把 Popeye 的生日改成 11/14/46

```
sed '/Popeye/s/:[0-9]*[0-9]*\/[0-9]*[0-9]*\/[0-9]*[0-9]*/:11\/14\/46/' datebook
```

9〉 删除所有空白行

```
sed '/^$/d' datebook
```

10〉 写一个 sed 命令表，将:

a. 在第 1 行之前插入标题 PERSONNEL FILE
b. 删除以 500 结尾的工资
c. 打印文件内容，把姓和名颠倒
d. 在文件末尾添加 THE END

答案放在 chap04/a10 文件中，内容如下：

```
# My first sed script by Wangzhh.
1i\
        PERSONNEL FILE
/500/d
s/\([A-Z][a-z]*\) \([A-Z][a-z]*\):/\2 \1:/
$a\
    THE END
```

第 5 章 awk 实用程序：awk 作为一种 UNIX 工具

1. awk 简介

   awk 是用来操作数据和产生报表的一种编程语言。数据可能来自标准输入、一个或者多个文件或者是一个进程的输出。awk 可以用在命令行里用于简单操作，或者可以为了较大的应用而写到程序中。常见的 awk 命令有 awk、nawk 等。awk 从第 1 行到最后一行逐行扫描文件（或输入），并执行选定的操作（封装在花括号里）。本文章所有的例子使用的文件未经说明都在 chap05 目录下。

2. awk 的格式

   awk 程序由 awk 命令，封装在引号里（或在一个文件里）的程序指令，和输入文件组成。如果没有指定一个输入文件，则输入来自标准输入(stin)键盘。

   employees 文件中的内容：

   ```
   $ cat employees
   Tom Jones       4424    5/12/66 543354
   Mary Adams      5346    11/4/63 28765
   Sally Chang     1654    7/22/54 650000
   Billy Black     1683    9/23/44 336500
   ```

   1〉从文件输入

   ```
   $ awk '/Mary/' employees
   ```

   解释：打印 employees 文件中含有 Mary 的行

   ```
   $ awk '{print $1}' employees
   Tom
   Mary
   Sally
   Billy
   ```

   解释：打印文件 employees 中的第一个域。域的分隔符是空格。

   ```
   nawk '/Sally/{print $1,$2}' employees
   ```

   解释：只有找到了 Sally 的行才打印第一个域和第二个域。

   2〉管道输入

   ```
   $ df -k|awk '$4>1024000'
   ```

   解释：报告剩余空间大于 1024000k 的盘。

3. 格式化输出

   1〉print 函数

   例子 1：

   ```
   $ date
   2005 年 04 月 30 日 星期六 19 时 29 分 25 秒 CST
   $ date|awk '{print "Date:" $1 "\nTime:" $3}'
   Date:2005 年 04 月 30 日
   Time:19 时 34 分 24 秒
   ```

   注意，用 date 命令查看时间格式。不同的语言可能格式也不一样，因此 awk 也要随之而变。\n 是转义序列，表示换行符。常见转义序列如下表：

转义序列	含义
\b	退格
\f	换页
\n	换行
\r	回车
\t	跳格
\047	八进制值 47，一个单引号
\c	C 代表任意其它字符，例如\''

表 5.1 转义序列

例子 2：

```
$ nawk '/Sally/{print "\t\tHave a nice day, "$1,$2 "\!"}' employees
        Have a nice day, Sally Chang!
```

解释：如果包含模式 Sally，则 print 函数打印两个跳格，串 Have a nice day,第一个域和第二个域，然后跟叹号。

OFMT 变量：当打印数字时，如果使用 print 函数，想控制精度时，可以用 OFMT 变量。默认设置是“%.6g”,也就是打印精度到小数点后 6 位。下面的例子可以改变精度。

```
$ nawk 'BEGIN{OFMT="%.2f"; print 1.23456789,12E-2}'
1.23 0.12
```

2〉printf 函数

printf 函数提供了强大的格式化输出功能，如果对 c 比较熟悉，就不会感觉太陌生。

转换字符	定义
c	字符
s	串
d	十进制数
ld	长十进制数
u	无符号十进制数
lu	无符号长十进制数
x	十六进制数
lx	长十六进制数
o	八进制数
lo	长八进制数
e	以科学技术发记的符点数
f	浮点数
g	从 e 或 f 转换中选择一种占用空间最少的记符点数

表 5.2 printf 的转换字符

字符	定义
-	左对齐修饰符
#	整数在用八进制显示前面有个 0；整数在用十六进制显示时前面有个 0x
+	对于用 d、e、f 和 g 的转换，带一个数字符号+或-显示整数
0	显示的值用 0 补而不是用空格

表 5.3 修饰符

格式符	功能
假设 x='A' y=15 z=2.3 $1=Bob Smith	
%c	打印一个单 ASCII 字符 $ printf "The character is %c\n" $x The character is A
%d	打印一个十进制数 $ printf "The boy is %d years old\n" $y The boy is 15 years old
%e	打印用科学计数法记的一个数 $ printf "z is %e\n" $z z is 2.300000e+00
%f	打印一个符点数 $ printf "z is %f\n" $z z is 2.300000
%o	打印一个数的八进制值 $ printf "y is %o\n" $y y is 17
%s	打印一个字符串 $ printf "The name of the culprit is %s\n" $1 The name of the culprit is Bob Smith
%x	打印一个数的十六进制值 $ printf "y is %x\n" $y y is f

表 5.4 printf 格式说明符

例子：

```
$ echo UNIX |awk '{printf "|%-15s|\n",$1}'
|UNIX           |
```

解释：echo 命令输出从管道中送给 awk。格式说明符说明将会打印一个占 15 个空格，左对齐，封装在竖杠里而且有换行的串。

```
$ awk '{printf "The name is %-15s ID is %8d\n",$1,$3}' employees
The name is Tom             ID is     4424
The name is Mary            ID is     5346
The name is Sally           ID is     1654
The name is Billy           ID is     1683
```

4. 在一个文件里的 awk 命令

如果 awk 命令放在文件里，就使用-f 选项和 awk 文件名结合使用。处理过程：把一条记录读到 awk 的缓存里而且对该记录测试并执行 awk 文件里的每个命令。在 awk 完成对第一条记录的操作后，删除该记录并把下一条记录读入缓存，依此类推。如果操作不受模式控制，则默认行为是打印整个记录。

例子：

```
$ cat employees
Tom Jones       4424    5/12/66 543354
Mary Adams      5346    11/4/63 28765
Sally Chang     1654    7/22/54 650000
Billy Black     1683    9/23/44 336500
$ cat awkfile
/^Mary/{print "Hello Mary!"}
{print $1, $2, $3}
$ nawk -f awkfile employees
Tom Jones 4424
Hello Mary!
Mary Adams 5346
Sally Chang 1654
Billy Black 1683
```

5. 记录和域

1〉记录：awk 不把输入数据看作一个无穷的字符串，而是把它看作一种格式或结构。默认情况下把每行叫做一个记录(record)，并以一个换行符终止。输入和输出的记录分隔符默认是回车符，保存在内置 awk 变量 ORS 和 RS 中。ORS 和 RS 可以改变，但是方式有限。

例子:

```
$ nawk '{print $0}' employees
Tom Jones       4424    5/12/66 543354
Mary Adams      5346    11/4/63 28765
Sally Chang     1654    7/22/54 650000
Billy Black     1683    9/23/44 336500
```

解释：变量$0 保存当前记录，这条命令相当于 nawk'{print}' employees

例子：

```
$ awk '{print NR,$0}' employees
1 Tom Jones     4424    5/12/66 543354
2 Mary Adams    5346    11/4/63 28765
3 Sally Chang   1654    7/22/54 650000
4 Billy Black   1683    9/23/44 336500
```

解释：NR 代表行号。

2〉域

类似表中的字段，是指记录中的一个词条。默认域分隔符是空白区域，也就是空格或制表符(TAB)。Awk 中用 NF 来记录每条记录的域数量。

3〉域分隔符

输入域分隔符：awk 的内置变量 FS 保存输入域分隔符的值。当时用 FS 的默认值时，awk 用空格或制表符分隔域，删除前导空白区和制表符。FS

可以改变，可以在 BEGIN 语句中改变，也可以在命令中改变。要想在命令中改变需要用-F 选项。

举例：

```
$ cat employees2
Tom Jones:4424:5/12/66:543354
Mary Adams:5346:11/4/63:28765
Sally Chang:1654:7/22/54:650000
Billy Black:1683:9/23/44:336500
$ awk '{print $1,$2}' employees2
Tom Jones:4424:5/12/66:543354
Mary Adams:5346:11/4/63:28765
Sally Chang:1654:7/22/54:650000
Billy Black:1683:9/23/44:336500
$ awk -F: '{print $1,$2}' employees2
Tom Jones 4424
Mary Adams 5346
Sally Chang 1654
Billy Black 1683
```

举例 2：多个域分隔符。如果使用多个域分隔符，要将其封装在方括号里。

```
$ nawk -F'[ :\t]' '{print $1,$2,$3}' employees2
Tom Jones 4424
Mary Adams 5346
Sally Chang 1654
Billy Black 1683
```

解释：把空格、制表符、冒号当成输入域分隔符。

输出域分隔符：默认的输出域分隔符用逗号来分隔，被分隔的域之间打印一个空格，如果域之间没有逗号，则打印时各域将挤在一起。

6. 模式和操作

awk 模式(patterns)控制 awk 将对一行输入作什么样的操作。一个模式包括一个正则表达式，一个产生正确或者错误条件的表达式，或者它们的组合。默认操作时打印模式中符合表达式条件的各行。当读入一个模式时，有一条隐含的 if 语句。如果 if 语句是隐含的，周围可以没有花括号。

例子：

```
$ awk '$3<4000' employees
Sally Chang     1654    7/22/54 650000
Billy Black     1683    9/23/44 336500
```

解释：如果第 3 个域小于 4000，则打印该域。

操作：操作(actions)是封装在花括号里且由分号分隔的语句。如果一个模式在一个操作之前，则该模式规定了何时执行该操作。

举例：

```
$ awk '/Tom/{print "Hello there," $1}' employees
Hello there,Tom
```

解释：如果记录中包含 Tom，则打印 Hello there,Tom

7. 正则表达式

一个正则表达式对 awk 来说是一个由封装在正斜杠里的字符组成的模式。Awk 支持的正则表达式与 egrep 基本一样，可以参考表 3.1 和表 3.2。

举例：

```
$ nawk '/^Mary/' employees
Mary Adams      5346    11/4/63 28765
```

解释：显示 employees 文件中以 Mary 开头的行

```
$ nawk '/^[A-Z][a-z]* /' employees
Tom Jones       4424    5/12/66 543354
Mary Adams      5346    11/4/63 28765
Sally Chang     1654    7/22/54 650000
Billy Black     1683    9/23/44 336500
```

解释：显示行开头是一个大写字母，然后跟一个或多个小写字母，再跟一个空格。

匹配操作符(~)、否定号(!)用来与以条记录或域里的表达式相匹配。

例子：

```
$ awk '$1 ~/[Bb]ill/' employees
Billy Black     1683    9/23/44 336500
```

解释：awk 将显示第一个域里与 Bill 或者 bill 相匹配的行

```
$ awk '$1 !~/ly$/' employees
Tom Jones       4424    5/12/66 543354
Mary Adams      5346    11/4/63 28765
```

解释：显示第一个域中末尾不是 ly 的所有行。

8. 写在命令表文件中的 awk 命令

写 awk 命令表的方法可以参考 chap05/info 文件。该文件内容如下：

```
# Here is a sample script:
#*****************************************************************
*
# My first awk script by Jack Sprat
# Script name: info; Date: February 28, 1999
/Tom/{print  "Tom's birthday is " $3}
/Mary/{print NR, $0}
/^Sally/{print "Hi Sally, " $1 " has a salary of $" $4 "."}
#End of script
```

9. 练习题参考答案

习题文件是 chap05/donors,部分内容如下：

```
Mike Harrington:(510) 548-1278:250:100:175
Christian Dobbins:(408) 538-2358:155:90:201
Susan Dalsass:(206) 654-6279:250:60:50
Archie McNichol:(206) 548-1348:250:100:175
```

上面的数据库包含名字、电话号码和过去三个月的竞选捐款。

1〉打印所有的电话号码

```
$ awk -F: '{print $2}' donors
```

2〉打印 Dan 的电话号码

```
$ awk -F: '/^Dan/{print $2}' donors
```

3〉打印 Susan 的名字和电话号码

```
$ nawk -F'[ :\t]' '/^Susan/{print $1,$3,$4}' donors
```

4〉打印所有以 D 开头的姓

```
$ awk '$2 ~/^D/{print $2}' donors
```

5〉打印所有以一个 C 或 E 开头的名

```
$ awk '/^[CE]/{print $1}' donors
```

6〉打印所有只有四个字符的名

```
$ nawk '/^[A-Z][a-z][a-z][a-z] /{print $1}' donors
```

7〉打印所有那些区号是 916 的名

```
$ awk -F: '$2 ~/916/{print $1}' donors
```

8〉打印 Mike 的活动捐款。打印每个值时都要以一个美元符号打头；例如：$500$200$300

```
$ awk -F: '/^Mike/{printf "$%d$%d$%d\n",$3,$4,$5}' donors
```

9〉打印姓，其后跟一个逗号和名

```
$ nawk -F'[ :\t]' '{printf "%s,%s\n",$2,$1}' donors
```

10〉 写一个叫 facts 的 awk 命令表，它能：

a. 打印 Savages 的全名和电话号码
b. 打印 Chet 的捐款
c. 打印所有第一个月捐款$250 的人

文件内容如下：

```
$ cat facts
#My first awk scripts by wangzhh.
/Savage/{print $1,$2}
/^Chet/{printf "Beneficence of Chet: $%d$%d$%d\n",$3,$4,$5}
$3 ~/250/{printf "Person who's beneficence is 250:%s\n",$1}
```

调用方法：

```
nawk -F: -f facts donors
```

第 6 章 awk 实用程序：awk 编程结构

比较表达式比较行，如果行里的条件为真，就执行操作。如果给表达式求值为真则值等于 1，反之等于 0。

1. 关系操作符

下表列出了关系操作符。

操作符	含义	例子
<	小于	x<y
<=	小于等于	x<=y
==	等于	x==y
!=	不等于	x!=y
>=	大于等于	x>=y
>	大于	x>y
~	与正则表达式相匹配	x~/y/
!~	不与正则表达式相匹配	x!~/y/

表 6.1 关系操作符

关系操作符举例：

```
$ cat employees
Tom Jones       4423    5/12/66    543354
Mary Adams      5346    11/4/63    28765
Sally Chang     1654    7/22/54    650000
Billy Black     1683    9/23/44    336500
$ awk '$3==4423' employees
Tom Jones       4423    5/12/66    543354
$ nawk '$3 >5000{print $1}' employees
Mary
$ nawk '$2 !~/Adam/' employees
Tom Jones       4423    5/12/66    543354
Sally Chang     1654    7/22/54    650000
Billy Black     1683    9/23/44    336500
```

条件表达式：一个条件表达式用两个符号，问号和冒号来给表达式求值。与一个 if/else 语句相比更简介。

2. 条件表达式

格式：

```
conditional expression1?expression2:expression3
```

它等同于

```
{if(expression1)
   expression2
 else
   expression3
}
```

条件表达式举例：

```
nawk '{max=($1>$2)?$1:$2;print max}' employees
```

解释：如果第一个域大于第二个域，就把第一个域的内容赋给 max,反之把第二个域的内容赋给 max，然后打印 max

3. 计算

可以在模式里执行计算。awk 以浮点方式执行所有的算术运算。

操作符	含义	例子
+	加	x+y
-	减	x-y
*	乘	x*y
/	除	x/y
%	求模	x%y
^	幂运算	x^y

表 6.2 算术运算操作符

4. 复合模式

复合模式（compound patterns）是把模式和逻辑操作符相结合的表达式，给一个表达式从左到右求值。

操作符	含义	例子
&&	逻辑与	a&&b
||	逻辑或	a||b
!	逻辑非	!a

表 6.3 逻辑操作符

复合模式举例：

```
$ nawk '$3>4000 && $3<=6000' employees
Tom Jones       4423    5/12/66    543354
Mary Adams      5346    11/4/63    28765
$ nawk '!($3>4000 && $3<=6000)' employees
Sally Chang     1654    7/22/54    650000
Billy Black     1683    9/23/44    336500
```

5. 范围模式

范围模式从一个模式的第一次出现匹配到第二个模式的第一次出现，然后从第一个模式的第二次出现匹配到第二个模式的第二次出现等等。如果第一个模式被匹配，而第二个模式没有找到，则 awk 将显示直到末尾的所有行。

例子：

```
$ cat datafile
northwest       NW      Joel Craig      3.0     .98     3       4
western         WE      Sharon Kelly    5.3     .97     5       23
southwest       SW      Chris Foster    2.7     .8      2       18
southern        SO      May Chin        5.1     .95     4       15
southeast       SE      Derek Johnson   4.0     .7      4       17
eastern         EA      Susan Beal      4.4     .84     5       20
northeast       NE      TJ Nichols      5.1     .94     3       13
north           NO      Val Shultz      4.5     .89     5       9
central         CT      Sheri Watson    5.7     .94     5       13
$ nawk '/^north/,/^west/' datafile
northwest       NW      Joel Craig      3.0     .98     3       4
western         WE      Sharon Kelly    5.3     .97     5       23
northeast       NE      TJ Nichols      5.1     .94     3       13
```

```
north           NO      Val Shultz      4.5     .89     5       9
central         CT      Sheri Watson    5.7     .94     5       13
```

6. 一个数据有效性检查程序

本例子使用 chap06/passwd 文件，内容如下：

```
tooth:pwHfudo.eC9sM:476:40:Contract Admin.:/home/rickenbacker/tooth:/bin/csh
lisam:9JY7OuS2f3lHY:4467:40:Lisa M. Spencer:/home/fortune1/lisam:/bin/csh
goode:v7Ww.nWJCeSIQ:32555:60:Goodwill Guest User:/usr/goodwill:/bin/csh
bonzo:eTZbu6M2jM7VA:5101:911: SSTOOL Log account :/home/sun4/bonzo:/bin/csh
info:mKZsrioPtW9hA:611:41:Terri Stern:/home/chewie/info:/bin/csh
cnc:IN1IVqVj1bVv2:10209:41:Charles Carnell:/home/christine/cnc:/bin/csh
bee:*:347:40:Contract Temp.:/home/chanel5/bee:/bin/csh
friedman:oyuIiKoFTVOTE:3561:50:Jay Friedman:/home/ibanez/friedman:/bin/csh
chambers:Rw7R1k77yUY4.:592:40:Carol Chambers:/usr/callisto2/chambers:/bin/csh
gregc:nkLulOg:7777:30:Greg Champlin FE Chicago
ramona:gbDQLdDBeRc46:16660:68:RamonaLeininge MWA CustomerService Rep:/home/forsh:/bin/csh
```

要求程序能够显示域个数不等于 7 的行、第一个域不包含数字字母的行、第二个域等于星号的行。程序写在 ex_6.8.nawk 文件中，内容如下：

```
# nawk script (covered later in the chapter)
#   To demonstrate example 6.8 using this script, run
#   % nawk  -f ex_6.8.nawk  passwd
BEGIN {FS = ":"}
NF != 7 { printf("line %d, does not have 7 fields: %s\n",NR,$0)}
$1 !~ /[A-Za-z0-9]/{printf("line %d, nonalphanumeric user id: %s\n"NR,$0)}
$2 == "*" {printf("line %d, no password: %s\n",NR,$0)}
```

下面是调用方法和执行结果：

```
$ awk -f ex_6.8.nawk passwd
line 7, no password: bee:*:347:40:Contract Temp.:/home/chanel5/bee:/bin/csh
line 10, does not have 7 fields: gregc:nkLulOg:7777:30:Greg Champlin FE Chicago
```

7. 练习题参考答案

联系文件是 chap06 下的 lab4.data，内容如下：

```
Mike Harrington:(510) 548-1278:250:100:175
Christian Dobbins:(408) 538-2358:155:90:201
Susan Dalsass:(206) 654-6279:250:60:50
Archie McNichol:(206) 548-1348:250:100:175
Jody Savage:(206) 548-1278:15:188:150
Guy Quigley:(916) 343-6410:250:100:175
Dan Savage:(406) 298-7744:450:300:275
Nancy McNeil:(206) 548-1278:250:80:75
John Goldenrod:(916) 348-4278:250:100:175
Chet Main:(510) 548-5258:50:95:135
Tom Savage:(408) 926-3456:250:168:200
Elizabeth Stachelin:(916) 440-1763:175:75:300
```

1>打印那些第一个月捐款超过 100 的人的姓名

```
$ awk -F: '$3>100 {print $1}' lab4.data
```

2> 打印那些第一个月捐款少于$60 的人的姓名和电话号码

```
$ awk -F: '$3<60{print $1,$2}' lab4.data
```

3> 打印那些第三个月捐款在 90 到 150 之间的人

```
$ awk -F: '$5>=90 && $5<=150 {print $0}' lab4.data
```

4> 打印那些三个月捐款在 800 以上的人

```
$ awk -F: '$3+$4+$5>800 {print $0}' lab4.data
```

5> 打印那些平均每月捐款大于$150 的人名和点话号码

```
$ nawk -F'[ :\t]' '($5+$6+$7)/3>150{print $1,$3,$4}' lab4.data
```

6> 打印那些区号不是 916 的人名

```
$ nawk -F'[ :\t]' '$3 !~/916/{print $1}' lab4.data
```

7> 打印每条记录，记录号在前面

```
$ awk '{print NR,$0}' lab4.data
```

8> 打印每个人的名字和捐款总额

```
$ nawk -F'[ :\t]' 'sum=($5+$6+$7) {print $1,sum}' lab4.data
```

9> 把 Elizabeth 的第二次捐款加上$10

```
$ nawk -F'[ :\t]' 'juankuan=($1 ~/Elizabeth/)?$6+10:$6{print $1,juankuan}' lab4.data
```

10>把 Nancy McNeil 的名字改成 Louise McInnes

```
$ nawk 'name=($1 ~/Nancy/)?"Louise McInnes":$1{print name}' lab4.data
```

第 7 章 awk 实用程序：awk 编程

1. 变量

- 数值和串常数

  数值常数可以表示成整数、浮点、科学计数等。含有空格的串要封装在双引号里。例如"Hello world"。如果一个域或这数组元素为空，则串值为空。一个空行也被当作空串。

- 用户自定义变量

  用户自定义变量包括字母、数字和下划线，且不能以数字开头。在 awk 中变量不用声明，awk 通过表达式的上下文推断变量类型，并且如果需要，还可以在不同类型的变量中相互转换。

- 递加和递减操作符(++和--)

  与 c++语言中一样。x++是后递加，++x 是先递加。

- 内置变量。

  内值变量要大写，他们可以用在表达式里而且可以被重置。如下表

变量名	含义
ARGC	命令行变元个数
ARGV	命令行变元组数
FILENAME	当前输入文件名
FNR	当前文件里的记录号
FS	输入域分隔符，默认是空格
NF	当前域里的域个数
NR	到目前为止的记录数
OFMT	数值输出格式
OFS	输出域分隔符
ORS	输出记录分隔符
RLENGTH	由 match 函数匹配的串的长度
RS	输入记录分隔符
RSTART	由 match 函数匹配的串偏移量
SUBSEP	下标分隔符

表 7.2 nawk 内置变量

  内置变量举例：

```
$ nawk -F: '$1=="Mary Adams"{print NR,$1,$2,$NF}' employees2
2 Mary Adams 5346 28765
```

  解释：-F 选项把分隔符改成冒号。如果域 1 等于 Mary Adams,则打印记录号、第一个域、第二个域和最后一个域($NF)

- BEGIN 模式

  BEGIN 模式后面跟一个操作模块,在 awk 处理文件之前执行该模块。BEGIN 模式主要用来设置 OFS、RS、FS 等内置变量的值。

```
$ awk 'BEGIN{FS=":";OFS="\t";ORS="\n\n"}{print $1,$2,$3}' employees2
Tom Jones       4423    5/12/66

Mary Adams      5346    11/4/63
```

```
Sally Chang      1654     7/22/54

Mary Black       1683     9/23/44
```

解释：在处理输入文件之前，把域分隔符置成一个冒号，输出域分隔符制成跳格符，并把输出记录分隔符(ORS)设置成两个换行符。

- END 模式

  END 模式不与任何输入行相匹配，但是执行任何与 END 模式相关的操作。在所有行处理完毕之后再来处理 END 模式。

```
$ awk '/Mary/{count++}END{print "Mary was found " count " times."}' employees
Mary was found 1 times.
```

解释：对于包含 Mary 的每一行，count 变量值都递增 1。awk 处理完毕之后，END 模块打印结果。

2. 重定向和管道

- 输出重定向：当把输出从 awk 重定向到一个 UNIX 文件时，使用 shell 重定向操作符。如果重定向操作符用在 awk 命令里面则必须将重定向文件用双引号括起来。例如：

  ```
  $ awk '$4>70 {print $1,$2 >"passing_film"}' datafile
  ```

- 输入重定向(getline)

  getline 函数用来从标准输入，例如管道或者文件来读入数据，而不是从正被处理的当前文件。getline 取得输入的下一行，并更新 NF、NR 和 FNR 等内置变量的值。如果 getline 找到记录则返回 1，达到 EOF（文件结束）则返回 0。如果有错误则返回-1。

  - 举例 1：

    ```
    $ nawk 'BEGIN{"date"|getline d;print d}' datafile
    2005 年 05 月 06 日 星期五 15 时 05 分 41 秒 CST
    ```

    解释：执行 date 命令，然后把输出从管道送到 getline，并赋给自定义变量 d，然后打印 d。

  - 举例 2：

    ```
    $ nawk 'BEGIN{while("ls"|getline) print}'
    awk.sc2
    datafile
    datafile2
    employees
    employees2
    lab5.data
    names
    passwd
    ```

    解释：将把 ls 的输出送到 getline。对于每次循环，getline 都从 ls 读取一个以上输出，然后打印到屏幕上

  - 举例 3：

    ```
    $ nawk 'BEGIN{printf "What is your name?";\
    > getline name<"/dev/tty"}\
    > $1 ~ name {print "Found " name " on line ",NR "."}\
    > END{print "See ya, " name "."}' employees
    ```

What is your name?Wangzhonghai
See ya, Wangzhonghai.
解释：将在屏幕上打印 What is your name?，并且等待用户响应。getline 函数将从终端(/dev/tty)接受输入，一直到输入一个换行符为止，然后把输入存到用户自定义变量 name 中。如果递一个域和 name 的值相匹配，则执行 print 函数。在 END 模式中打印"See ya，"然后跟 name 的值。

- 举例 4

$ nawk 'BEGIN{while (getline<"/etc/passwd">0)lc++;print lc}'
16
解释：awk 将从“/etc/passwd”读取各行，lc 递增，直到 EOF，然后打印 lc 的值。

3. 管道

如果在一个 awk 程序中打开一个管道，则必须在打开另一个之前先关闭它。在管道符号由变的命令封装在双引号里。一次只能打开一个管道。下面的例子是将 names 文件中的姓名按照姓作为第一关键字，名作为第二关键字来进行倒排序：

```
$ awk '{print $1,$2|"sort -r +1 -2 +0 -1 "}' names
tony tram
john smith
dan savage
barbara nguyen
elizabeth lone
john goldenrod
susan goldberg
george goldberg
eliza goldberg
alice cheba
```

4. 关闭文件和管道

如果在 awk 中再次使用一个文件或者管道来读取或写入，则需要首先关闭管道。打开和关闭管道的例子可以参考/chap07/awk.sc3 文件。（书中源代码没有这个文件，而书上错误很多。）

5. 回顾（略）

6. 条件语句

1) if 语句

以 if 结构开头的语句是操作语句。在条件模式(conditional patterns)里，if 是隐含的；在一个条件操作语句中，if 要显式说明，并且后面跟一个封装在括弧里的表达式。如果跟在条件表达式后面的语句不止一条，则语句组必须放在花括号里，并且用分号或者换行符来分开。

格式：

```
if (expression){statement;statement;...}
```

举例 1：

```
$ nawk '{if($8>15) print $1 " To high"}' datafile
```

举例 2：

```
$ nawk '{if($8>15 && $8<=23){safe++; print $1 " OK " safe}}' datafile
```

2) if/else 语句

if/else 语句允许一个二路判断。

格式：

```
{if(expression) {
   statement;statement;...
   }
else{
   statement;statement;...
   }
}
```

3) if/else else if 语句

允许多路判断。

格式：

```
{if (expression) {
   statement;statement;...
   }
else if (expression){
   statement;statement;...
   }
else if (expression){
   statement;statement;...
   }
else{
   statement;statement;...
   }
}
```

7. 循环

循环的作用是，如果一个条件为真，则重复执行测试表达式后的语句。循环通常用来迭代一条记录里的域，及循环操作 END 模块里的一个数组的元素。awk 有 3 种循环：while 循环、for 循环和特殊 for 循环。

1) while 循环

使用规则与 c 中一样。

举例：

```
$ nawk '{i=1;while(i<=NF){print NF,$i;i++}}' datafile
```

解释：变量 i 初始化为 1；当 I 小于或等于记录中的域个数(NF)时，执行 print 并让 i 递增，然后再测试表达式，直到 i>NF。读取下一条的记录的时候 i 被重新初始化。

2) for 循环

使用规则与 c 中一样。

举例：

```
$ nawk '{ for (i=1;i<=NF;i++) print NF,$i}' datafile
```

3) 循环控制：break 和 continue

break 和 continue。break 让你在某个条件为真时调出循环。continue

使循环在某个条件为真时条过后面的任何语句，并把控制返回循环顶部，开始下一次迭代。

举例：

```
{for (x=3;x<=NF;x++)
   if($x<0){print "Bottomed out!";break}
#breaks out of for loop
}
```

举例 2：

```
{for (x=3;x<=NF;x++)
   if($x==0){print "Get next item!";continue}
#starts next item of for loop
}
```

8. 程序控制语句

1) next 语句

next 语句从输入文件取得下一个输入行，在 awk 命令表顶部重新开始执行。

举例：

```
$ awk '{if($1 ~/Tom/){next} else {print}}' db
```

解释：如果第一个域包含 Tom，则略过。

2) exit 语句

exit 语句用来终止 awk 程序。它停止处理记录，但是不跳过 END 语句。

3)

9. 数组

在 awk 中数组叫做关联数组(associative arrays)，因为下标记可以是数也可以是串。awk 中的数组不必提前声明，也不必声明大小。数组元素用 0 或空串来初始化，这根据上下文而定。

1) 关联数组的下标

- 把变量用作数组索引。

  举例：

  ```
  $ nawk '{name[x++]=$2};END{for(i=0;i<NR;i++)\
  > print i,name[i]}' employees
  0 Jones
  1 Adams
  2 Chang
  3 Black
  ```

  解释：在数组 name 里的下标是一个用户自定义变量 x。++显示了一个数值型的上下文。

- 特殊 for 循环

  在 for 循环无效的情况下，即当串被用作下标或者下标不是连续的数时，用特殊 for 循环来读取一个关联数组。特殊 for 循环把下标当作一个键来找到它的相关的值。

  格式：

  ```
  {for(item in arrayname){
       print arrayname[item]}}
  ```

举例：

```
$ nawk '/^Tom/{name[NR]=$1};\
> END{for(i=1;i<=NR;i++) print name[i]}' db
Tom



Tom
Tom

Tommy
$ nawk '/^Tom/{name[NR]=$1};\
> END{for(i in name){print name[i]}}' db
Tom
Tom
Tommy
Tom
```

解释：这两个例子用 NR（行号）来做下标，因此匹配 Tom 的模式的行不连续，数组下标不连续。如果用传统 for 来循环打印会在数组没有值的地方打印空值。通过使用特殊 for 循环，只打印数组中有值的内容。

□

2)

10.

附录 B 比较三种 shell

特性		C shell	Bounne shell	Korn shell
变量	给局域变量赋值	set x=5	s=5	s=5
	赋变量属性			typeset
	给环境变量赋值	setenv NAME Bob	NAME=Bob	export NAME=Bob
	存取变量	echo $NAME set var=net echo ${var}work network	echo $NAME var=net echo ${var}work network	echo $NAME 或 print $NAME var=net print ${var}work network
专用变量	该进程的 PID	$$	$$	$$
	退出状态	$status	$?	$?
	上一个后台作业		$!	$!
数组	给数组赋值	Set x=(a b c)	N/A	Y[0]=a;y[1]=b;y[2]=c Set –A fruit apples pears plums