第二章　UNIX的软件工具

- 2.1 Unix软件开发工具简介
- 2.2 BACKUS系统
- 2.3 vi
- 2.4 sed
- 2.5 awk
- 2.6 make
- 2.7 SVN

➢2.5 awk

➢awk概述

awk(A Pattern Scanning and Process Language)称为字符扫描和处理语言。由贝尔实验室的Alfred V. Aho，Peter J. Weinberger和Brian W. Kernighan三人发明并命名。

awk也是一个非交互式的文本编辑程序， awk的作用是用awk命令对输入文件逐行进行处理。其适用情况与sed类似。

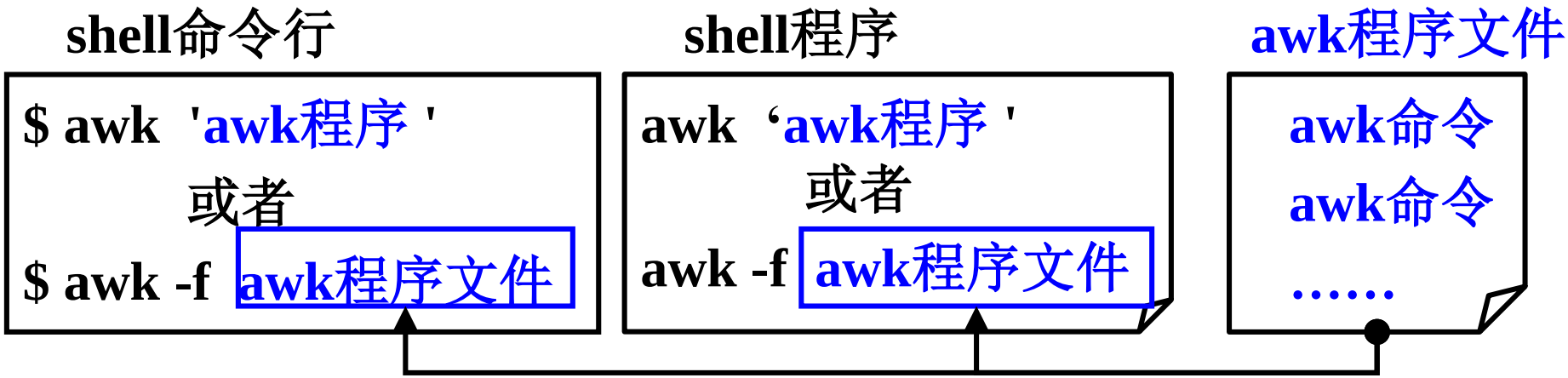
➢2.5 awk

➢awk的执行方式

- 方式1　awk '*awk程序*' [*输入文件*]　　或者
- 方式2　awk -f *awk程序文件* [*输入文件*]
- 功能

用*awk程序*对*输入文件*逐行进行扫描并且处理。

*输入文件*缺省时为stdin。输出为stdout。



➢2.5 awk

➢记录、字符段和特殊变量

awk定义文件由若干个记录（record）组成。在缺省的情况下，每一行为一个记录，因此可以简单地对记录和行不予区分。每个记录由若干个字符段（field）组成。字符段通常以非空字符组成，即缺省的字符段分隔符是空白符（空格<SP>和制表符<TAB>）。

以下是awk提供的一些常用的特殊变量：

RS： 记录分隔符(record separator)。缺省值为换行符。

FS： 字符段分隔符(field separator)。缺省值为空白符。

NR： 每个记录的序号(number of record)。如果以行作为记录，则最后一行的行号就代表文件的行数。

NF： 每个记录中字符段的数目(number of field)。

各个字符段依次用变量$1，$2，…表示。

$0表示整个记录。

➢2.5 awk

➢变量及运算

awk程序中的变量可以是数字型的，或者是字符型的。数字型变量的计算按浮点类型处理。

变量名可以用除了特殊变量之外的任何字符来定义，并可随时引用变量。变量的初值为零或空字符。

字符段变量的引用可以用数字型表达式表示，例如当i和n都是已知的数字（数字型变量）时，可以使用$(i+1)，$(i-n)等等。

除字符段变量的引用外，变量的引用不需加$

awk提供各种运算操作：

包括算术运算、关系运算和逻辑运算等。

awk运算符与C语言的运算符是一样的。

算术运算符有+、-、*./、%(取模)、++和--。

关系运算符有>、<、>=、<=、==和!=。

逻辑运算符有&&、||和!。

赋值运算符有=、+=，-=、*=、/=和%=。

➢2.5 awk

➢awk的程序结构

awk程序由若干个命令组成，每一个命令都可以采用以下任意一种方式：

pattern **{*action*}**	对匹配***pattern***的行执行***action***。
pattern	***action***缺省，输出与***pattern***匹配的行
{*action*}	***pattern***缺省，对所有行都执行***action***操作

其中，***pattern***称为匹配模式，***action***称为操作。

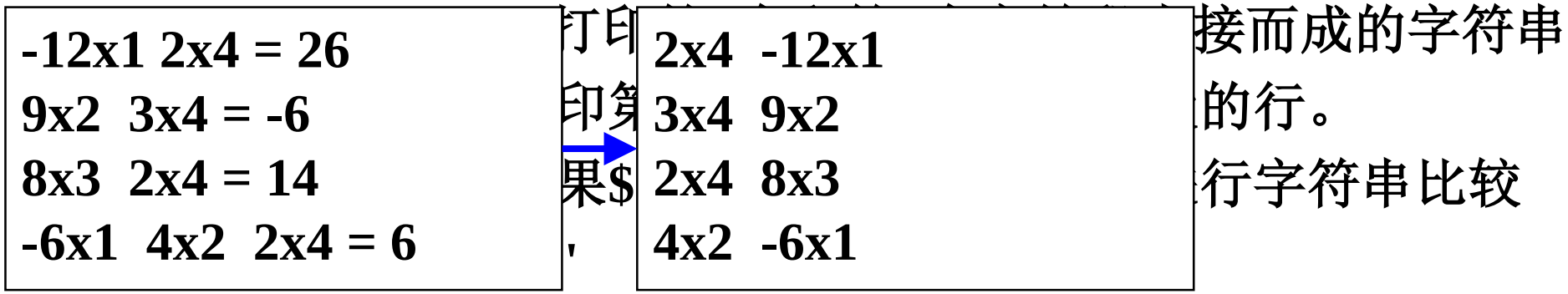
两者都可以缺省，为此给***action***加上花括号以示区别。

※变量及运算

awk提供将字符段和常数串接起来的功能。在打印语句或函数调用的实在参数表达式中，通常是用逗号分隔各个参数。否则，仅用空白格分隔的参数，awk就认为这是参数串接而成的一个参数。

【例2-18】awk简单程序示例。

'{print□$2, $1}'　　打印第2个字段和第1个字段



接而成的字符串

的行。

行字符串比较

※变量及运算

awk提供将字符段和常数串接起来的功能。在打印语句或函数调用的实在参数表达式中，通常是用逗号分隔各个参数。否则，仅用空白格分隔的参数，awk就认为这是参数串接而成的一个参数。

【例2-18】awk简单程序示例。

'{print□$2, $1}'　　打印第2个字段和第1个字段

'{print□$1 $2}'　　打印第1个和第2个字符段串接而成的字符串

```
-12x1 2x4 = 26
9x2  3x4 = -6
8x3  2x4 = 14
-6x1  4x2  2x4 = 6
```

印第 … 的行。

果$ … 行字符串比较

'

```
-12x12x4
9x23x4
8x32x4
-6x14x2
```

※变量及运算

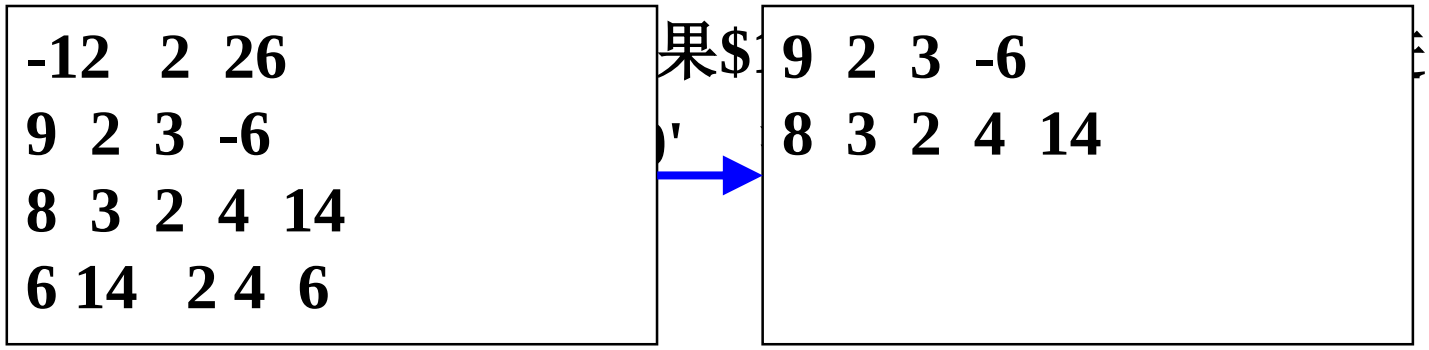
awk提供将字符段和常数串接起来的功能。在打印语句或函数调用的实在参数表达式中，通常是用逗号分隔各个参数。否则，仅用空白格分隔的参数，awk就认为这是参数串接而成的一个参数。

【例2-18】awk简单程序示例。

'{print□$2, $1}'　　打印第2个字段和第1个字段

'{print□$1 $2}'　　打印第1个和第2个字符段串接而成的字符串

'$1 > $2'　　打印第1个字段大于第2个字段的行。



※变量及运算

awk提供将字符段和常数串接起来的功能。在打印语句或函数调用的实在参数表达式中，通常是用逗号分隔各个参数。否则，仅用空白格分隔的参数，awk就认为这是参数串接而成的一个参数。

【例2-18】awk简单程序示例。

'{print□$2, $1}'	**打印第2个字段和第1个字段**
'{print□$1 $2}'	**打印第1个和第2个字符段串接而成的字符串**
'$1 > $2'	**打印第1个字段大于第2个字段的行。如果$1和$2都不是数值，将进行字符串比较**
'NR == 100, NR == 200'	**打印第100行到第200行**

➢2.5 awk

➢awk程序的匹配模式

pattern {*action*} 对匹配*pattern*的行执行*action*。

*pattern*的表达形式为：

pattern1 [, *pattern2*]]

表示由*pattern1*和*pattern2*确定匹配行。

当*pattern2*缺省时，表示由*pattern1*确定匹配行。

*pattern1*和*pattern2*可以表达为以下几种匹配模式：

- 字符串匹配模式
- 组合模式
- 专用模式

➢2.5 awk

➢awk程序的匹配模式

➢ 字符串匹配模式

/string/

表示寻找含有字符串***string***的匹配行。

var **~ /*string*/**

表示在变量***var***中查找与***string***匹配的子字符串（包括变量***var***等于字符串***string***）。

var **!~ /*string*/**

表示在变量***var***中不存在与***string***匹配的子字符串。

string的用法除了与**vi**和**ex**中对字符串的定义相同之外，还可以使用以下方式：

x*|*y	表示匹配字符串***x***或者匹配字符串***y***。
***x*+**	表示对字符***x***的一次或多次重复。
***x*?**	表示对字符***x***的零次或多次重复。

➢2.5 awk

➢awk程序的匹配模式

➢ 字符串匹配模式

【例2-19】字符串匹配模式示例。

$1 ~ /John/　　选择第1个字符段$1中含有子字符串John的行

$1 ! ~/John/　　选择$1中不含有子字符串John的行

/ab|cd/　　选择匹配字符串ab或cd的行

/[Aa]b|[Cc]d/　　选择匹配字符串Ab、ab、Cd或cd的行

/start/, /stop/　　选择含有字符串start到含有stop之间的所有行

➢2.5 awk

➢awk程序的匹配模式

➢ 组合模式

组合模式是指各种运算（算术运算、关系运算和逻辑运算）的组合。

【例2-20】组合模式示例。

$2 > $1 +100

选择$2大于$1加100的行

NR%2 == 0 && $1 !~ /John/

选择$1中不含有子字符串John的偶数行

NR%2 != 0 && $1 > "s"

选择首字符大于s（t，u等）的奇数行

➢2.5 awk

➢awk程序的匹配模式

➢ 专用模式

专用模式有BEGIN和END两个，它们与文件内容无关，格式为：

BEGIN {*action*} 在处理所有行之前执行*action*。

END {*action*} 在处理所有行之后执行*action*。

例如，打印文件中字符段的总数，awk程序为；

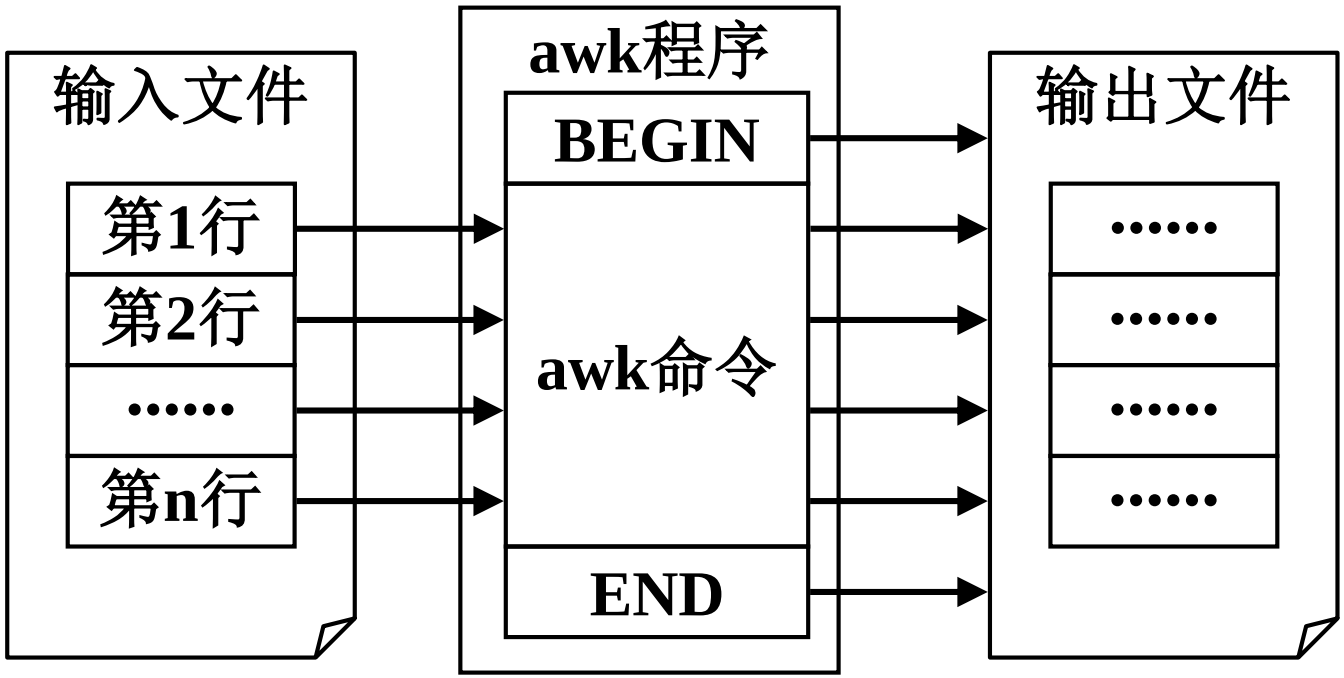
‘{ words +=NF }

END {print words}’

由于初始时words的值为0，因而最后将打印出文件中的字符段总数。

➢2.5 awk

➢awk程序的匹配模式



➢2.5 awk

➢awk程序的操作(action)

在一对大括号内可以有多个awk语句，组成一个操作，各个语句之间用分号隔开。

只有一个语句或者多个语句中的最后一个语句可以省略分号。

- 打印语句

awk提供的打印语句有print和printf。格式为：

print□[*expr*, …]

printf□*format*, [*expr*, …]

- print语句的作用是按自由格式打印若干个参量*expr*
  - 各参量之间自动加空格，并且自动加换行符。省略参量时，打印整行。
- printf语句的作用与C语言中的printf作用相同。
  - 每一个表达式*expr*的格式由*format*中用百分号%引起的格式说明符表示，例如%d表示打印整数，%s表示打印字符串等等。
  - *format*是用双引号括起的字符串。

➢2.5 awk

➢awk程序的操作(action)

- ➢ 打印语句

例如，以下程序都表示将每一行加上行号后打印：

{print□NR, $0}

{printf□"%d\t", NR; print}

例如，用浮点格式%8.2f、整型格式%5d和字符型格式%s分别打印第2个字符段、第3个字符段以及变量prev，可采用以下的printf语句：

{printf□"%8.2f%5d%s\n", $2, $3, prev}

例如，每隔10行输出第5个字符段(如果有第5个字符段的话)，而且不产生换行符的命令为：

NR % 10 == 1 && NF >= 5　　{printf□"%d\t", $5}

➢2.5 awk

➢awk程序的操作(action)

➢ 赋值语句

赋值语句的形式为：

var **=** ***expr***

其中，***var***是任何一种变量或数组元素。***expr***是常数或算术表达式，或者是下页将介绍的函数。

例如：

{ x = 5 }	**x将获得整型值5**
{ y = "smith" }	**y将获得字符串"smith"**
{ x = "3" + "4" }	**将使x获得的值为"7"**
{ $3 = $1 + $2 }	**将$1和$2按数值相加后赋给$3**

➢2.5 awk

➢常用内部函数

length[(*var*)]	**相当于C语言的函数strlen()**
sqrt(*var*)	
log(*var*)	
exp(*var*)	

例如：

length($1□$2)	**计算$1和$2串接后的字符数**
length($1) > length($2)	**比较$1与$2中的字符数**
{ print□length($1 $2) }	**打印$1与$2的字符数之和**
{ print□length }	**打印一行中的字符总数。等价于**
{ print□length($0) }	

length的参量可以省略，因此length等价于length($0)。

➢2.5 awk

➢控制语句

awk提供一些基本的控制结构，如if-else，while和for。

if (*expr*) *statement*

else *statement*

for (*expr1*; *expr2*; *expr3*) *statement*

while (*expr*) *statement*

break、continue和exit语句的作用与C语言相同。

如果awk程序由若干行命令组成，next语句的作用是结束对文件中当前行的awk命令操作，即对当前行不进行余下的awk命令操作，而直接处理文件中的下一行。

➢2.5 awk

➢控制语句（续）

在控制结构内的语句组如果是几个语句的话要用一对大括号括起，它们的用法与C语言中的控制结构作用相同。

例如： **{if($3 > 1000) $3 = “too big”; print}**

表示当$3大于1000时，将$3改为字符串“too big”，然后打印。

例如： **{if($3 > 1000) {$3 = “too big”; print}}**

表示当$3大于1000时，执行替换并且打印，否则不打印。

例如：反向打印记录中所有的字符段，每个字符段占一行，可采用以下程序：

{i = NF; while(i >= 1) {print $i; i--}}

注意：$i为第i个字符段，而不是对变量i的引用。

➢awk程序示例

【例2-21】先打印一段字符表示开始，再打印全文时加上行号，最后打印文件的行数。

编写文件awk4的内容为：

```
awk ‘
      BEGIN {print “Starting>>>”}
      {printf  “%d\t”, NR; print }
      END {print “Total lines: ”, NR}’  <in4 >out4
```

执行命令为

```
$ awk4
```

输入文件in4为：

```
#include <stdio.h>
main()
{
    printf(“Hello!\n”);
}
```

输出文件out4为：

```
Starting>>>                    由BEGIN产生
1    #include <stdio.h>
2    main()
3    {
4        printf(“Hello!\n”);
5    }
Total lines: 5                 由END产生
```

【例2-22】awk程序示例：文件排版整理。

已知输入文件的字符/数据没有整齐排版，例如文件random内容如下：

```
R20  24    20
     C10   20   50
M21          63        13      4000
```

要使文件random整齐排列为

```
R20   24     20
C10   20     50
M21   63     13     4000
```

可以用三种方式完成文件的排版整理。

(1) 编写awk程序awk5为

```
awk ‘
   {for (i=1; i<=NF; i++) printf “%s \t”, $i;
     printf “\n”}  ’ $1
```

$i指第i个字段

$1指命令行的第1个参数

运行：

```
$ awk5 random
```

【例2-22】awk程序示例：文件排版整理。（续）

文件random内容如下：

```
  R20  24    20
       C10   20   50
  M21          63        13      4000
```

要使文件random整齐排列为

```
  R20   24     20
  C10   20     50
  M21   63     13     4000
```

(2) 将awk程序写在awk命令文件script中：

```
    {for (i=1; i<=NF; i++) printf “%s\t”, $i; printf “\n”}
```

在shell中执行awk命令：

```
  $ awk -f script  <random
```

(3) 在shell中运行awk程序：

```
  $ awk '
  > {for (i=1; i<=NF; i++) printf “%s\t”, $i;
  > printf “\n”} ' random
```

注意：大于号>是shell的二级待命符

【例2-23】处理文件，要求第1个字段重复的行只输出一次。

假定**in6**的内容为：

```
aa abc
aa def
bb
cc 123
cc 45
```

则**out6**为：

```
aa abc
bb
cc 123
```

【例2-23】处理文件，要求第1个字段重复的行只输出一次。

含awk程序的文件awk6：

```
awk '$1 != prev {print; prev=$1} '
```

执行awk6：

```
$ awk6  <in6  >out6
```

假定in6的内容为：

```
aa abc
aa def
bb
cc 123
cc 45
```

则out6为：

```
aa abc
bb
cc 123
```

【例2-23】处理文件，要求第1个字段重复的行只输出一次。（续）

含awk程序的文件awk6：

```
awk '$1 != prev {print; prev=$1} '
```

执行awk6：

```
$ awk6  <in6  >out6
```

awk6的解析：

行	in6	$1	prev	测试情况		prev=$1	out
1	aa abc	aa	空	不同	=>	aa	aa abc
2	aa def	aa	aa	相同	=>	aa(不变)	
3	bb	bb	aa	不同	=>	bb	bb
4	cc 123	cc	bb	不同	=>	cc	cc 123
5	cc 45	cc	cc	相同	=>	cc(不变)	

【例2-24】修改例2-23

如果当前行的$1与上一行的$1相同，只输出$2，如果当前行的$1与上一行的$1不相同，则输出$1和$2。

假定仍以in6为输入：

```
aa abc
aa def
bb
cc 123
cc 45
```

则out7为：

```
aa    abc   def
bb
cc 123   45
```

【例2-24】修改例2-23

如果当前行的$1与上一行的$1相同，只输出$2，如果当前行的$1与上一行的$1不相同，则输出$1和$2。编写awk程序awk7为：

```
awk ‘
      $1 == prev  {printf “\t%s”, $2;}
      $1 != prev   {printf “\n%s\t%s”, $1, $2;  prev=$1} ’
```

其中需要注意的是以上两行程序的顺序不能颠倒。

执行awk7：

```
$ awk7 <in6  >out7
```

假定仍以in6为输入：

```
aa abc
aa def
bb
cc 123
cc 45
```

则out7为：

```
⊕                                  多输出一个空行
aa    abc   def⊕
bb⊕
cc 123   45          最后一行没有回车
```

【例2-25】awk程序

在例2-24中awk7产生了一个空行，而且最后一行没有回车。修改题意为：删除可能出现的空行，并且在最后一行产生换行。

编写shell程序awk8为：

```
sort | \
awk ‘
   $1 == prev  {printf “\t%s”, $2;}
   $1 != prev   {printf “\n%s\t%s”, $1, $2;  prev=$1}
   END {printf “\n”} ’ | \                在最后一行产生换行
awk ‘NR != 1’                          不输出第1行(空行)
```

执行awk8：

```
$ awk8  <in8  >out8
```

假定in8的内容为：

```
cc 123
aa def
bb
aa abc
cc 45
```

则out8为：

```
aa    abc    def⊕
bb⊕
cc    123    45⊕      最后一行有回车
```

➢sed和awk程序的比较及联合应用

【例2-27】可以用sed或者awk实现相同功能的程序示例

(1) 删除空行

```
sed '/^$/d'                    awk 'NF >0'
```

(2) 删除第1行到第10行

```
sed '1,10d'                    awk 'NR >10'
```

(3) 输出第1行到第4行

```
sed '5,$d'                     awk 'NR <5'
```

(4) 在每个行尾都加上反斜杠 \

```
sed ‘s/$/\\/’                  awk '{print $0 "\\"}'
```

(5) 在每行前插入一个制表符

```
sed 's/^/\t/'                  awk '{print "\t" $0}'
```

【例2-28】文件整理程序awk11

假定输入文件in11为

620396
720393
300048
720392
300687
720359
510597

希望整理后输出文件out11为：

620396,□720393,□300048
720392,□300687,□720359
510597

整理文件的要求是每行输出三个数据，并且用“,□”作为分隔符。

【例2-28】文件整理程序awk11

假定输入文件in11为	希望整理后输出文件out11为： 620396,□720393,□300048 720392,□300687,□720359 510597 整理文件的要求是每行输出三个数据，并且用“,□”作为分隔符。
620396 720393 300048 720392 300687 720359 510597	

执行命令为：

```
$awk11 <in11 >out11
```

shell程序awk11的内容为：

```
awk '
   NR%3 == 0 {print}
   NR%3 != 0 {printf "%d,□", $1}          每行输出三个数据
   END {if (NR%3 != 0) printf "\n"} ' | \
sed 's/,□$//'                             删除最后一行行尾的“,□”
```

➢作　业

- 上机及习题　　**2-9, 2-10, 2-20, 2-21**