任务二　sed 流编辑器

(一)　sed 简介

sed（stream editor）是一种在线编辑器，它一次处理一行内容。处理时，把当前处理的行存储在临时缓冲区中，称为“模式空间”（pattern space），接着用 sed 命令处理缓冲区中的内容，处理完成后，把缓冲区的内容送往屏幕。接着处理下一行，这样不断重复，直到文件末尾。文件内容并没有改变，除非你使用重定向存储输出。Sed 主要用来自动编辑一个或多个文件；简化对文件的反复操作；编写转换程序等。

用法：sed [选项]... {脚本(如果没有其他脚本)} [输入文件]...

简单演示

```
[root@teacher ~]# sed 'p' /etc/passwd
sed 的逐行操作。
[demo@teacher sedtest]$ cat /etc/passwd |sed 'p'

[root@teacher ~]# sed '/root/p' /etc/passwd

[root@teacher ~]# sed -n '/root/p' /etc/passwd
```

-n 只打印满足条件的行

-r 使 sed 支持扩展正则表达式

(二)　常用选项

-n：取消默认的回显（只回显符合条件的行）。

-e：将- e 后面的字符看作是操作脚本，如果只有一个操作脚本，- e 可以省略。

-f : sed 的脚本来自于文件

-i : 直接修改文件

-r : 支持扩展的正则表达式

--follow-symlinks 跟踪符号链接

示例：

```
[root@teacher ~]# sed -n -e '/root/p' -e '/demo/p' /etc/passwd
[root@teacher ~]# sed -n -e '/root/p;/demo/p' /etc/passwd


[root@teacher ~]# sed -i   '/^$/d' grep.txt
```

（三） sed 操作命令

字符	描述
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，如果标记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：

字符	描述

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）

（四） 地址定位

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 。

字符	描述
first~step	从 first 行开始，循环时每次步进为 step ，当 first 为零时，等于 step。
$	最后一行。
/regexp/	匹配该正则表达式的行。
\cregexpc	匹配该正则表达式的行，c 可以是任意字符。
1，addr2	从匹配第一个地址开始，到 addr2。
addr1,+N	匹配 addr1 和 addr1 后面的 N 行。
addr1,~N	匹配 addr1 和 addr1 后面的直到行号等于 N 的行。

（五） sed 使用示例

```
[demo@teacher sedtest]$ cp /etc/passwd /tmp/sedtest/passwdbk
[demo@teacher sedtest]$ cat -n /tmp/sedtest/passwdbk|sed    '2~3a test00'
[demo@teacher sedtest]$ cat -n /tmp/sedtest/passwdbk|sed    '$c test00'
[demo@teacher sedtest]$ cat -n /tmp/sedtest/passwdbk|sed    '/root/d'
[demo@teacher sedtest]$ cat -n /tmp/sedtest/passwdbk|sed    '1i \\t\t passwd
document'
[demo@teacher sedtest]$ cat -n /tmp/sedtest/passwdbk|sed -n '\a[0-9]\{5\}a
p'
[demo@teacher sedtest]$ cat -n /tmp/sedtest/passwdbk|sed -n '\#[0-9]\{5\}#
p'
[demo@teacher sedtest]$ cat -n /tmp/sedtest/passwdbk|sed -n '1,5 p'
[demo@teacher sedtest]$ cat -n /tmp/sedtest/passwdbk|sed -n '2,+5 p'
[demo@teacher sedtest]$ cat -n /tmp/sedtest/passwdbk|sed -n '2,~5 p'
[demo@teacher sedtest]$ cat -n /tmp/sedtest/passwdbk|sed -n '2,+5 !p'
[demo@teacher sedtest]$ cat -n /tmp/sedtest/passwdbk|sed 's/root/ADMIN/g'
[demo@teacher sedtest]$ cat -n /tmp/sedtest/passwdbk|sed -n
's/root/ADMIN/gp'
[demo@teacher sedtest]$ cat -n /tmp/sedtest/passwdbk|sed      '30,$s/[0-
9]/X/g'
[demo@teacher sedtest]$ cat -n /tmp/sedtest/passwdbk|sed
'/:107:/,$s/sbin/SBIN/g'
[demo@teacher sedtest]$ sed    '/root/=' passwdbk
[demo@teacher sedtest]$ sed
'y/abcdefghijklmnopkrstuvwxyz/ABCDEFGHIJKLMNOPKRSTUVWXYZ/'
passwdbk
[demo@teacher sedtest]$ sed '/root/w 1.txt' passwdbk
```

（六） 使用技巧

```
[demo@teacher sedtest]$ sed 's/^.//' passwdbk
[demo@teacher sedtest]$ sed 's/.$//' passwdbk
[demo@teacher sedtest]$ sed 's/[a-zA-Z]\{1,\}$//' passwdbk

[demo@teacher sedtest]$ sed 's/ntp/(&)/'    passwdbk
[demo@teacher sedtest]$ sed 's/^./# &/' passwdbk

[demo@teacher sedtest]$ sed -r 's/^[0-9a-zA-Z]+:x:[0-9]+//' passwdbk

[demo@teacher sedtest]$ sed 's/\(^[0-9a-zA-Z]\{1,\}\)\(:x:\)\([0-
9]\{1,\}\)/\3\2\1/' passwdbk

 [demo@365linux sedtest]$ sed      's/\(^[0-Z]\{1,\}\)\(:x:\)\([0-
9]\{1,\}\)/\3\2\1/'    passwdbk

或者：  sed    -r 's/(^[0-9a-zA-Z]+)(:x:)([0-9]+)/\3\2\1/'    passwdbk
        sed    -r 's/(^[0-Z]+)(:x:)([0-9]+)/\3\2\1/'    passwdbk

[demo@teacher sedtest]$ sed    's/\(^[0-9a-zA-
Z]\{1,\}\)\(:.*:\)\(\/.*$\)/\3\2\1/    ' passwdbk

[demo@teacher sedtest]$ sed -n '/^demo/s/\/bin\/bash/\/sbin\/nologin/gp'
passwdbk
[demo@teacher sedtest]$ sed -n '/^demo/s#/bin/bash#/sbin/nologin#gp'
passwdbk
#可以是任意字符，来替换/。
```

[demo@teacher sedtest]$ sort -t":" -k3 -nr passwdbk |sed '/:48:/,$s#/.*$#/bin/csh#'

[demo@teacher sedtest]$ sed '/[a-z]:[1-4]\{0,1\}[0-9]\{1\}:/s/\/[a-z]\{1,\}\/[a-z]\{1,\}$/\/bin\/csh/' passwdbk

[demo@teacher sedtest]$ name=demo; sed -n "/$name/p" passwdbk

sed 使用变量匹配时，要使用双引号。

任务三　awk 编程语言

(一)　awk 简介

awk 是一种编程语言，用于在 linux/unix 下对文本和数据进行处理。数据可以来自标准输入、一个或多个文件，或其它命令的输出。它支持用户自定义函数和动态正则表达式等先进功能，是 linux/unix 下的一个强大编程工具。它在命令行中使用，但更多是作为脚本来使用。awk 的处理文本和数据的方式是这样的，它逐行扫描文件，从第一行到最后一行，寻找匹配的特定模式的行，并在这些行上进行你想要的操作。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)，如果没有指定模式，则所有被操作所指定的行都被处理。awk 分别代表其作者姓氏的第一个字母。因为它的作者是三个人，分别是 Alfred Aho、Peter Weinberger、Brian　Kernighan。gawk 是 awk 的 GNU 版本，它提供了 Bell 实验室和 GNU 的一些扩展。下面介绍的 awk 是以 GUN 的 gawk 为例的，在 linux 系统中已把 awk 链接到 gawk，所以下面全部以 awk 进行介绍。

AWK 的功能是什么？与 sed 和 grep 很相似，awk 是一种样式扫描与处理工具。但其功能却大大强于 sed 和 grep。 awk 提供了极其强大的功能：它几乎可以完成 grep 和 sed 所能完成的全部工作，同时，它还可以可以进行样式装入、流控制、数学运算符、进程控制语句甚至于内置的变量和函数。它具备了一个完整

的语言所应具有的几乎所有精美特性。实际上，awk 的确拥有自己的语言：awk 程序设计语言，awk 的三位创建者已将它正式定义为：样式扫描和处理语言。

（二） awk 用法

awk [POSIX or GNU 风格选项] [--] '程序文本' 文件 ...

awk [POSIX or GNU 风格选项] -f '程序文件' [--] 文件 ...

示例：

```
[root@localhost ~]# awk   '/root/{print $0}' /etc/passwd
```

（三） awk 选项

-F fs or --field-separator fs

指定输入文件折分隔符，fs 是一个字符串或者是一个正则表达式，如-F:。（FS 与定义变量的值） *

-v var=value or --asign var=value

在程序开始执行前赋值一个用户定义变量。在 BEGIN 块中可用。 *

-f program-file or --file program-file

从脚本文件中读取 awk 程序。

-mf NNN and -mr NNN

设置各种内在限制为 NNN，-mf 选项限制最大的字段数；-mr 选项限制最大记录数。这两个功能是 Bell 实验室版 awk 的扩展功能，在标准 awk 中不适用。（gawk 忽略）

-O or --optimize

启用优化后的程序内部表示。

-W compat or --compat, -W traditional or --traditional

在兼容模式下运行 awk。所以 gawk 的行为和标准的 awk 完全一样，所有的 awk 扩展都被忽略。 *

-W copyleft or --copyleft, -W copyright or --copyright

打印简短的版权信息。

-W dump-variables[=file] or --dump-variables[=file]

打印全局变量，以及它们的类型、值的排序列表到一个文件。（在用比较长的 awk 程序是避免变量冲突）

-W exec file or --exec file

类似于-f，不过它是最后被处理的。它应该用在#！脚本，尤其是 CGI 应用，避免在命令行从 URL 传递选项和源码（！）。这个选项禁止命令行变量赋值。

-W gen-po or --gen-po

扫描和分析 awk 程序，并在标准输出生成一个 GNU .po 格式的文件。程序本身不执行。

-W help or --help, -W usage or --usage

打印全部 awk 选项和每个选项的简短说明。

-W lint[=value] or --lint[=value]

提供关于可疑或不能移植到其他 AWK 实现的结构的警告。

-W lint-old or --lint-old

打印关于不能向传统 unix awk 移植的结构的警告。

-W non-decimal-data or --non-decimal-data

认识八进制和十六进制值输入数据。使用此选项时非常谨慎！

-W posix or --posix

打开兼容模式。但有以下限制：

不识别：\x 转义序列。

当 FS 被设置为一个空格时只有空格和制表符作为分隔符，换行符不行。

在？和：后不能继续行。

不识别关键字 function 的代名词 func。

操作符**和**=不能代替^和^=。

fflush（）函数无效。 *

-W profile[=prof file] or --profile[=prof file]

-W re-interval or --re-interval

允许间隔正则表达式的使用，参考(grep 中的 Posix 字符类)，如 r{n,m}。 *

-W source program-text or --source program-text

使用 program-text 作为源代码，可与-f 命令混用。

-W use-lc-numeric or --use-lc-numeric

这迫使 awk 来解析输入数据时使用的本地环境小数点字符。

-W version or --version

打印 bug 报告信息的版本。

--

选项结尾的信号。在 AWK 程序以-开始时有用。

（四） awk 程序执行

pattern { action statements }

function name(parameter list) { statements }

AWK 程序由一个序列的模式操作语句和可选的函数定义组成。

VARIABLES, RECORDS AND FIELDS

AWK 的变量是动态的，当他们在第一次使用时被定义。变量值可以是浮点数或（和）字符串，AWK 也有一维数组，模拟多维数组。在程序运行时有一些预定义变量被设置。

通常，记录被换行符分隔。可以使用内建的变量 RS 来控制记录的分隔符。

每读取一个记录，gawk 使用 FS 变量定义的分隔符来分隔字段。

（五） 内建变量(Built-in Variables)

变量	描述
ARGC	命令行参数的数目。
ARGIND	当前被处理的文件的 ARGV 的 index(从 0 开始算)。
ARGV	包含命令行参数的数组。
BINMODE	在 non-POSIX 系统中，文件的 I/O 指定使用"binary"模式。
CONVFMT	数字转换格式(默认值为%.6g)。
ENVIRON	当前环境变量关联数组。
ERRNO	包含一个描述错误信息的字符串。
FIELDWIDTHS	字段宽度列表(用空格键分隔)。设置以后匹配字段宽度而不是以 FS 分隔。 *

FILENAME	当前输入的文件名。 *
FNR	当前输入文件的记录号数值。 *
FS	字段分隔符(默认是任何空格)。 *
IGNORECASE	控制所有正则表达式和字符串操作的大小写敏感性（若值为非零，则忽略大小写匹配，初始值为 0）。
LINT	从 AWK 程序内部提供--lint 选项的动态控制。（若为真，打印 lint 警告）
NF	当前输入记录中的字段数。 *
NR	到目前为止看到的输入记录的总数。 *
OFMT	数字的输出格式(默认值是%.6g)。
OFS	输出字段分隔符(默认值是一个空格)。 *
ORS	输出记录分隔符(默认值是一个换行符)。 *
PROCINFO	这个数组的元素提供正在运行的 AWK 程序信息。可用（PROCINFO["egid"] PROCINFO["euid"] PROCINFO["FS"] PROCINFO["gid"] PROCINFO["pgrpid"] PROCINFO["pid"] PROCINFO["ppid"] PROCINFO["uid"] PROCINFO["version"] ）
RS	记录分隔符(默认是一个换行符)。 *
RT	记录终止符（gawk 使用 RS 的值）
RSTART	由 match()函数匹配到的第一个字符索引。（若无匹配则为 0）
RLENGTH	由 match（）函数所匹配的字符串的长度。（若无匹配为-1）
SUBSEP	多个数组元素下标分隔符(默认值是\034)。
TEXTDOMAIN	AWK 程序的文本域。用来查找程序的字符串本地化翻译。

示例：

```
[demo@teacher ~]$ echo "123456" |awk -v FIELDWIDTHS="3 2" '{print $1,$2}'
123 45
[demo@teacher ~]$ echo "123456" |awk -v FIELDWIDTHS="3 3" '{print $1,$2}'
123 456
[demo@teacher ~]$ echo "123456" |awk -v FIELDWIDTHS="2 99999" '{print $1,$2}'
12 3456
```

（六） 字符串常量

字符串常量在 AWK 中使用双引号。

符号	描述
\\	反斜杠
\a	警报
\b	退格
\f	换页符
\n	换行符
\r	回车符
\t	水平制表符
\v	垂直制表符
\xhex digits	\x 后面的字符代表十六进制数字 （如\x1B 是 ASCII ESC 字符）

\ddd	代表八进制数字（如\033 是 ASCII ESC 字符）
\c	代表字面字符 c

（七） 模式和行为

AWK 是一个行处理程序，匹配到 PATTERNS，则进行 ACTIONS，ACTIONS 永久在 { } 中。可以缺少 PATTERNS，也可以缺少 ACTIONS，但不能都没有。如果 PATTERNS 丢失，则 ACTIONS 处理每一行，若 ACTIONS 丢失，则 ACTIONS 等同于 {print}，即打印整个记录。

以＃开头注释一行。空白行可用作单独的语句。通常，一个语句以换行符结束，然而，以", { ? : && ||" 结束的行并非如此。还有 do ,else 有他们自己的语法自动到下一行。其他情况下，可以使用\来忽略换行符。

多个语句可以放在一行用；号隔开。

PATTERNS

符号	描述
BEGIN END	BEGIN 和 END 是两个特殊的模式，它不是用来针对输入的测试。如果所有的语句写在一个 BEGIN 块里面，则所有 BEGIN 模式的 ACTION 会被合并，并在输入被读取之前执行。同样的，所有的 END 块也会被合并，在所有输入结束时执行（或者在有 exit 语句执行时）。BEGIN 和 END 模式不能和模式表达式里面的模式联合。BEGIN 和 END 模式必须有 ACTION 部分。
/regular expression/	对每一行匹配正则表达式，类似于 egrep 的用法。

relational expression	关系表达式，可以使用运算符进行操作，通常用来比较正则表达式或（和）字符串（数字）之间的比较。
pattern && pattern	逻辑与 AND
pattern || pattern	逻辑或 OR
pattern ? pattern : pattern	如果第一个 pattern 为真，则使用第二个 pattern 来匹配，否则使用第三个。
(pattern)	边界界定。
! pattern	逻辑取反 NOT
pattern1, pattern2	定义一个范围。

（八） 正则表达式

符号	描述
c	匹配元字符 c
\c	匹配字面字符 c
.	匹配任何单个字符串(包括换行符)
^	匹配以一个字符串开头
$	匹配以一个字符串结尾
[abc...]	字符列表，匹配 abc...中的任一字符
[^abc...]	取反字符列表，匹配除了 abc...的任一字符
r1|r2	交替：匹配 r1 或者 r2

r1r2	串联：匹配 r1 以及 r2
r+	匹配一个以上的 r
r*	匹配零个以上的 r
r?	匹配零个或一个 r
(r)	组：匹配 r
r{n} r{n,} r{n,m}	间隔型表达式，表示 r 重复的次数，只有在--posix 或--re一interval 在命令行指定时有效。
\y	匹配一个单词开头或者末尾的空字符串。
\B	匹配单词内的空字符串。
\<	匹配一个单词的开头的空字符串，锚定开始。
\>	匹配一个单词的末尾的空字符串，锚定末尾。
\w	匹配由字母,数字或下划线组成的字符串。
\W	匹配不是由字母，数字或下划线组成的字符串。
\‘	匹配缓冲区（字符串）开头的一个空字符串。
\'	匹配缓冲区末尾的一个空字符串。
[:alnum:]	字母和数字
[:alpha:]	字母
[:blank:]	空格和 tab
[:cntrl:]	控制字符
[:digit:]	数字
[:graph:]	可打印且可见字符

[:lower:]	小写字母
[:print:]	可打印字符（不是指控制字符）
[:punct:]	标点符号（非字母，数字，控制字符或空白字符）
[:space:]	空白字符（比如空格，tab，换页符等）
[:upper:]	大写字母
[:xdigit:]	十六进制数

字符类只在中括号中才有效。

/[A-Za-z0-9]/==>/[[:alnum:]]/

整理符号（Collating Symbols）

等价类 （Equivalence Classes）

使用在非英语环境，gawk 不识别（只识别 POSIX 字符类）

gawk 通过各种命令行选项控制怎样在正则表达式中解释字符。

符号	描述
没有选项	默认，gawk 提供所有 POSIX 正则表达式和 GNU 正则表达式，但不支持间格表达式。 *
--posixabrt 173 usbmuxd 113 rtkit 172 qemu 107 avahi-autoipd 170 pulse 171	只有 POSIX 正则表达式被支持，GNU 则不支持（\w 被解释为 w），间格表达式被允许。
--traditional	传统的 UNIX AWK 正则表达式匹配。GNU，间隔，POSIX 字符类不支持，八进制和十六进制被转义成字面字符。

--re-interval	允许间隔字符，即使使用了--traditional。 *

（九） 操作行为（ACTIONS）

操作语句位于｛ ｝内。操作语句包含常见的任务，条件，循环等语句。

（十） 运算符

符号	描述
（...）	组
$	字段引用
++ --	自加和自减
^	求幂 （** **=）
+ - !	加号，减号，逻辑否
* / %	乘，除，取余
+ -	加法，减法
space	字符串连接
| |&	getline,print,printf 的 I/O 管道
< > <= >= != ==	常规关系运算符
~ !~	正则表达式匹配，取反匹配。（不要在其左边使用常量正则表达式（/foo/））,只能是右边。/foo/ ~ exp 等同于(($0 ~ /foo/) ~ exp)

in	数组成员
&&	逻辑与
||	逻辑或
?:	C 语言条件表达式。来自于 expr1?expr2:expr3
= += -= *= /= %= ^=	运算形式缩写规则

(十一) 控制语句

if (condition) statement [else statement]

while (condition) statement

do statement while (condition)

for (expr1; expr2; expr3) statement

for (var in array) statement

break

continue

delete array[index]

delete array

exit [expression]

{ statements }

(十二) 输入/输出语句

语句	描述
close(file [, how])	关闭文件，管道或者是 co-process。how 选项只能用在当关闭一个 co-process 的双向管道，值为“to”或者

	“from”。
getline	设置$0 从下一个输入记录;设置 NF，NR，FNR。*
getline <file	设置$0 从 file 的下一个记录;设置 NF。
getline var	设置 var 从下一个输入记录;设置 NR，FNR。
getline var <file	设置 var 从 file 的下一个记录。
command | getline [var]	从管道(命令的标准输出)获取$0 或 var。
command |& getline [var]	从管道(命令以 co-process 方式的标准输出)获取$0 或 var。
next	停止处理当前的输入记录。读取下一个输入记录和开始处理第一个模式。如果输入记录读取完成，则执行 END 块（如果有） *
nextfile	停止处理当前的输入文件。下一个输入记录来自下一个输入文件。FILENMAE 和 ARGIND 被更新，FNR 被重设为 1，AWK 开始处理第一个模式。。如果输入记录读取完成，则执行 END 块（如果有）
print	打印当前的记录。打印终止符为 ORS 变量的值。*
print expr-list	打印表达式。每个表达式迎 OFS 的值分隔。ORS 结束。
print expr-list >file	打印表达式到文件 file。
printf fmt, expr-list	格式化并打印。
printf fmt, expr-list >file	格式化并打印到文件。
system(cmd-line)	执行命令行命令并返回退出状态（在 no-POSIX 系统可能不能用）。 *

flush([file])	刷新任何输出文件或管道文件相关的缓存。

print 和 printf 允许额外的输出重定向

语句	描述
print ... >> file	追加到文件
print ... | command	写入管道
print ... |& command	发送数据到 co-process 或 socket

getline 命令成功返回 1，文件结束返回 0，错误返回－1.ERRNO 包含描述错误问题的字符串。

> NOTE
>
> 如果在一个循环里到 getline, 或者从 print,printf 使用管道，co-process,socket ，必须使用 close()创建一个命令和 socket 新的实例。AWK 不能自动关闭 pipes,sockets,co-process,当他们返回 EOF 时。

(十三) 特殊文件名

/dev/stdin

/dev/stdout

/dev/stderr

/dev/fd/n

/inet/tcp/lport/rhost/rport

/inet/udp/lport/rhost/rport

/inet/raw/lport/rhost/rport

/dev/pid

/dev/ppid

/dev/pgrpid

/dev/user

（十四）数值函数

atan2(y,x)

cos(expr)

exp(expr)

int(expr)

log(expr)

rand()

sin()

sqrt()

srand([expr])

（十五）字符串函数

asort(s [,d])

asorti(s [,d])

gensub(r,s,h [,t])

gsub(r,s [,t])

index(s,t)

length([S])

match(s,r[,a])

split(s, a [, r])

sprintf(fmt, expr-list)

strtonum(str)

sub(r, s [, t])

substr(s, i [, n])

tolower(str)

toupper(str)

（十六）时间函数

mktime(datespec)

strftime([format [, timestamp[, utc-flag]]])

systime()

（十七）位操作函数

and(v1, v2)

compl(val)

lshift(val, count)

or(v1, v2)

rshift(val, count)

xor(v1, v2)

（十八）国际化函数

bindtextdomain(directory [, domain])

dcgettext(string [, domain [, category]])

dcngettext(string1 , string2 , number [, domain [, category]])

（十九）用户定义函数

function name(parameter list) { statements }

(二十) 动态加载新函数

extension(object, function)

(二十一) 信号

pgawk 接受两种信号。

SIGUSR1

SIGHUP

(二十二) 示例

打印和排序所有用户的登录名:

```
BEGIN         { FS = ":" }
      { print $1 | "sort" }
```

统计文件的行数:

```
        { nlines++ }
    END    { print nlines }
```

在文件的每行前面添加行号：

```
          { print FNR, $0 }
```

连接和行号 (上一个例子的变化):

```
          { print NR, $0 }
```

从数据的特定行运行一个特定的命令:

```
           tail -f access_log |
           awk '/myhome.html/ { system("nmap " $1 ">> logdir/myhome.html") }'
```