第 4 讲　Linux 基础 (3)

王晓庆

wangxiaoqing@outlook.com

March 27, 2016

Outline

正则表达式

- 正则表达式由字符与元字符组成，整个表达式用于描述符合某些特定特征的一类字符串。
- 最基本的正则表达式匹配单个字符，大多数字符都与其自身匹配，少数具有特殊含义的元字符可以在其前面加反斜杠进行引用。
- Linux 中许多工具都支持正则表达式，各种编程语言也包含了处理正则表达式的库。

基本正则表达式 (BRE) 元字符

. 表示单个任意字符

```
grep 'a.b' file
grep 'a\.b' file
```

[xyz] 表示单个括号中列出的字符

```
grep '[0-9]\.[0-9]' file
grep '[^0-9][0-9][^0-9]' file
```

x* 表示其左边的项 x 出现 0 次以上

```
grep '[0-9][0-9]*' file
```

基本正则表达式 (BRE) 元字符

- 位置锚定

^ x 匹配位于行首的 x

```
grep '^#' file
grep -v '^#' file
grep '^[^#]' file
```

y$ 匹配位于行尾的 y

```
grep '[0-9]$' file
grep -v '^$' file | wc -l
```

扩展正则表达式 (ERE) 元字符

- 扩展正则表达式元字符
  - ? + | { } ( )
- ERE 元字符被视为普通字符，除非在其前面加上反斜杠！

扩展正则表达式 (ERE) 元字符 (2)

x+ 表示其左边的项 x 出现 1 次以上

```
grep '[0-9]\+' file
grep -E '[0-9]+' file
egrep '[0-9]+' file
```

x? 表示其左边的项 x 出现 0 次或 1 次

```
grep '[0-9]*\.?[0-9]\+' file
```

x|y 表示 x 或 y

```
grep 'html|HTML' file
```

扩展正则表达式 (ERE) 元字符 (3)

x{m} 表示 x 出现 m 次

```
grep '\$[0-9]\{2\}\.[0-9]{\2\}[^0-9]' file
```

x{m,} 表示 x 出现 m 次以上

```
grep '\$[0-9]\{5,\}\.[0-9]{\2\}[^0-9]' file
```

x{m,n} 表示 x 出现 m 到 n 次 (m<n)

```
grep '\$[0-9]\{3,4\}\.[0-9]{\2\}[^0-9]' file
```

扩展正则表达式 (ERE) 元字符 (4)

() 与组合与反向引用

```
grep '\([0-9]\+\)\.\1' file
grep '^\(.\).*\1$' file
```

试一试

1. 如何在/usr/share/dict/words 中查找长度为 5 的回文单词？
2. 如何滤出类似 warning 这样的行？
3. 如何查找包含相邻重复词的行？

字符类

- 为应更多语言环境，POSIX 定义了若干字符类

[:alnum:] 数字、字母
[:alpha:] 字母
[:blank:] 空格符、制表符
[:cntrl:] 控制字符
[:digit:] 数字0-9
[:graph:] 数字、字母、标点符号
[:lower:] 小写字母
[:print:] 数字、字母、标点符号、空格符
[:punct:] 标点符号
[:space:] 制表符、换行符、回车符、空格符
[:upper:] 大写字母
[:xdigit:] 十六进制数字0-9a-fA-F

反斜杠字符

- 有些字符前加上反斜杠后具有特殊含义

\b 匹配单词边缘的空字符串
\B 匹配非单词边缘的空字符串
\< 匹配单词开头的空字符串
\> 匹配单词结尾的空字符串
\w 匹配单词字符，即[_[:alnum:]]
\W 匹配非单词字符，即[^_[:alnum:]]
\s 匹配空白符，即[[:space:]]
\S 匹配非空白符，即[^[:space:]]

字符类和反斜杠字符举例

字符类示例

```
tr -d '[:punct:]' <file
grep '[[:digit:]]\+$' file # 注意两层中括号的区别
```

反斜杠字符示例

```
grep \bconfident\b /usr/share/dict/words
grep \Bconfident\B /usr/share/dict/words
```

fgrep(fixed fgrep)

- 等同于 grep -F
- fgrep 将所有字符都看作普通字符，搜索速度快！
- fgrep 可以同时搜索以换行符隔开的多个字符串

示例

```
fgrep 'normal mode
insert mode
command-line mode' learn-vim
```

在 find 命令中使用正则表达式

- -regex 匹配正则表达式
- -iregex 同上且忽略大小写

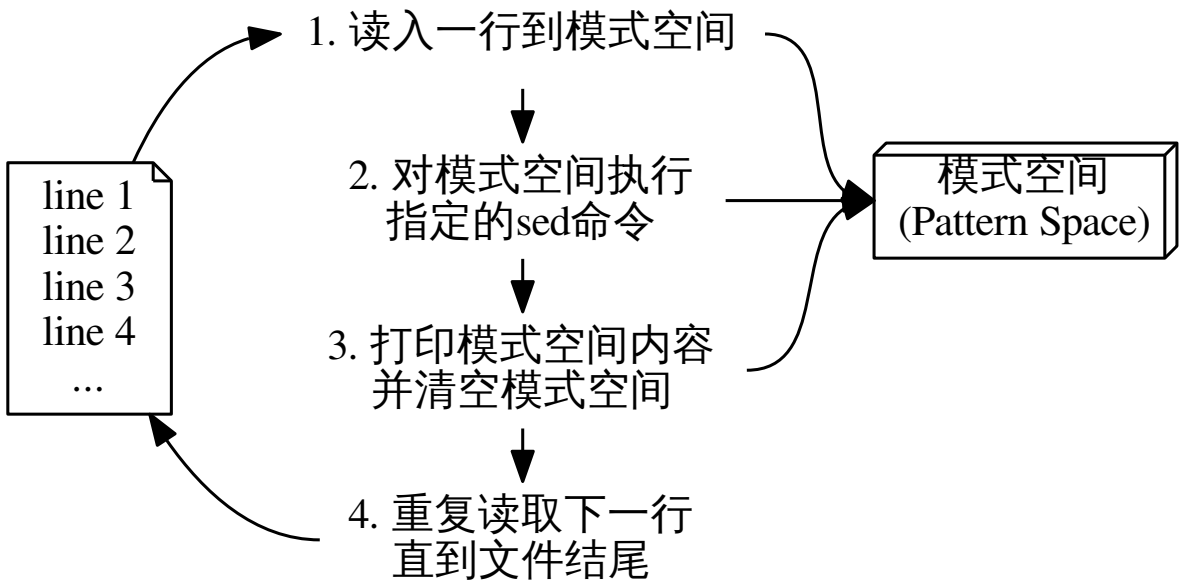
示例

```
find /usr/bin -regex 'mk'      #:-(
find /usr/bin -regex '.*mk.*'  #:-)
```

sed 简介

- sed(stream editor)

  sed 提供非交互式批量文本编辑功能，例如在 100 个文件中，处理 20 个不同的编辑操作。
- sed 的工作原理



运行 sed

- sed options... [script] [file]...

常用选项

```
-n   安静模式，不打印模式空间内容
-e script 添加处理脚本
-f script-file 添加保存在文件中的脚本
-i 直接修改原始输入文件
```

注意

script 内的命令可以用分号 (;) 或换行分隔

sed 工作原理

- sed 按以下方式循环处理每一行（sed 从前至后仅处理一遍）：
  1. 从输入流中读取一行，去掉尾部的换行符后放入模式空间
  2. 执行脚本，脚本中的每条命令可以与一个地址关联，地址可认为是某种条件码，命令仅在条件满足时才被执行。
  3. 当执行至脚本结尾时，除非指定了-n 选项，否则模式空间的内容将被打印到输出流，并且如果前面被删除了换行符则将其添加回去，
  4. 回到步骤 1，开始下一次循环。

注意

除非使用了类似 D 这样的特殊命令，每次循环的最后，模式空间都将被清空。

地址

行选择

n 第n行
m~n 从第m行开始，每n行，如1~2表示所有奇数行
$ 末行
/regexp/ 与正则表达式regexp匹配的行
\%regexp% 同上，但把默认的/替换成其他字符，如%

范围选择

m,n 从第m行至第n行
n,/regexp/ 从第n行往后至与regexp匹配的第一行
/regexp1/,/regexp2/
m,n! 除m~n行外
/regexp/! 除与regexp匹配的行外

常用命令

- # 注释
- q 退出
- d 删除模式空间，立即进入下一循环
- p 打印模式空间
- n 打印模式空间 (若无-n 选项)，然后将其内容替换为下一行或者退出
- { command1; command2; ... } 命令组，命令之间用分号隔开

sed 简单示例

示例

```
sed '' file
sed -n '' file
sed -n '1,$p' file
sed '10q' file
sed -n '$p' file
sed '/^$/d' file
sed '\|^#|d' file
sed -n 'n;p' file
sed -n -e 'n' -e 'p' file   # 同上
sed '1~2d' file             # 同上
sed '2~2!d' file            # 同上
```

s 命令 (搜索替换)

示例

```
cat well.txt
Sam reads well, sam writes well, sam sings well.
sed 's/sam/tom/' well.txt
sed 's/sam/tom/i' well.txt
sed 's#sam#tom#gi' well.txt
sed 's|sam|tom|2i' well.txt
sed 's$sam$tom$2gi' well.txt
```

试一试

```
echo 'this costs 23, and that costs 35' >costs.txt
```

1. 在所有价格前面加上美元符$
2. 若把23改成.23，把35改成3.5，怎么加美元符$?

a,i,c 命令

示例

```
sed '3a\
line 1\
line 2' file   # 在第 3 行后追加 (append)2 行内容

sed '10i\
line 1\
line 2' file   # 在第 10 行前插入 (insert)2 行内容

sed '/regex/c
line 1
line 2' file   # 将与 regex 匹配的行修改 (change) 为 2 行内容
```

r,w,y,= 命令和-f 选项

示例

```
sed '3r file2' file1 # 将 file2 的内容插入 file1 第 3 行之后
sed -n '/:\/\//w file2' file1 # 匹配://的行号保存至 file2
sed -n '\#://#w file2' file1  # 同上
sed 'y/aeiou/xxxxx/' file # 逐字符替换，前后长度需一致！
sed -n '$=' file          # 打印最后一行的行号，即 wc -l

cat sedscript             # 准备好 sed 脚本文件
1,3d                      # 每行包含一条 sed 命令
s/old/new/g
y/abc/xyz/
sed -n -f sedscript file  # 利用-f 让 sed 根据脚本处理 file
```

e 命令

- 将 sed 处理得到的结果提交给 shell 执行。

示例

```
# 复制目录结构
find teach/2014-linux/ -type d \
| sed 's/2014/2016/' \
| sed -n 's/^/mkdir -p /e'
```

sed 应用举例

- 实现 basename 命令

```
find /usr/bin -name 'mk*' -exec basename {} \;
```

- 实现 dirname 命令

```
find . -name '*.ppt' -exec dirname {} \; | sort | uniq
```

- 抽取网页文本内容
- unix 文本和 windows 文本转换

sed 高级命令

- D
  如果模式空间不含换行符，则与 d 相同。否则，删除模式空间内容至第一个换行符 (包含该换行符)，然后对模式空间重新执行一遍所有命令 (不读入新行)。
- N
  为模式空间追加换行符和一新行，如果没有新行可读，则直接退出 sed，不再执行后续命令。
- P
  打印模式空间内容至第一个换行符 (包括该换行符)

sed 高级命令

示例

```
sed -n 'N;D' students.db   # 打印末行
sed -n 'N;P' students.db   # 打印奇数行
sed -n '=' books.db | sed 'N;s/\n/ /'   # 加行号
```

想一想：下面哪些命令可以打印出输入的最末 2 行？

1. sed ‘N;N;D’ students.db
2. sed -n ‘$-1,$p’ students.db
3. head -4 students.db | sed -n ‘N;$p’
4. head -3 students.db | sed -n ‘N;$p’
5. sed ‘N;$!D’ students.db

暂存空间 (hold space)

- sed 运行时可以使用两个缓存空间
  - 模式空间 (pattern space)
  - 暂存空间 (hold space)

模式空间

1. 不断从输入获取新行
2. 可对其内容执行 sed 命令
3. 一般每次执行完 sed 命令后会被清空

暂存空间

1. 默认无内容，但可以从模式空间获得内容
2. 不能直接对其内容执行 sed 命令
3. 不会自动清空其内容

在模式空间和暂存空间之间传递数据

- x – exchange
  交换模式空间和暂存空间的内容。
- h – hold pattern space
  把模式空间的内容复制到暂存空间 (覆盖)
- H – Hold pattern space
  把模式空间的内容追加到暂存空间尾部 (用换行符分隔)
- g – get contents of hold area
  把暂存空间的内容复制到模式空间 (覆盖)
- G – Get contents of hold area
  把暂存空间的内容追加到模式空间尾部 (用换行符分隔)

暂存空间使用示例

示例

```
# 在行间加空行
sed 'G' students.db          # 每行前加空行
sed 'x;p;x' students.db      # 每行后加空行
# 实现 tac file 命令
sed -n '1!G;$!h;$p' students.db
sed '1!G;$!h;$!d' students.db
```

流程控制

标签

```
:label      #设置标签
```

分支 (branch)

```
b label     #跳转到标签位置
b           #跳转到脚本结尾
```

测试 (test)

```
t label     #如果成功执行了s命令，则跳转到标签
t           #如果成功执行了s命令，则跳转到脚本结尾
```

流程控制示例

示例

```
sed ':a;N;6,$D;ba' students.db      # 实现 tail -5 命令
sed ':m;s/^.\{1,79\}$/ &/;tm' students.db # 实现文本右对齐
```

试一试

1. 如何实现文本居中对齐？
2. 如何在/usr/share/dict/words 中搜索任意长度的回文单词？

awk 简介

- awk 是由 Al Aho，Peter Weinberger 和 Brian Kernighan 设计与实现的一种模式扫描与处理语言。
- awk 最早的设计目的是针对报表生成的一种小巧且具表达力的语言,awk 对于处理格式化结构的文本文件特别强大。

awk 入门

- 引例

```
cat emp.data
Beth   4.00 0
Dan    3.75 0
Kathy  4.00 10
Mark   5.00 20
Mary   5.50 22
Susie  4.25 18
```

注：第 1 列为员工姓名，第 2 列为时薪，第 3 列为工作时间

1. 要求打印出所有工作时间大于 0 的员工应发薪水

```
awk '$3>0 {print $1, $2*$3}' emp.data
```

1. 要求打印出工作时间为 0 的员工的姓名

```
awk '$3==0 {print $1}' emp.data
```

awk 工作原理

- awk 程序结构
  - 引例中位于引号内的部分就是 awk 程序，awk 程序由如下形式的语句构成：

```
pattern {action}
pattern {action}
......
```

  - awk 程序依次扫描每行输入，每一行都会与每个 pattern 比较，若匹配则执行相应的 action。

如果省略 {action} 部分，则打印与 pattern 匹配的整行

```
awk '$3==0' emp.data  #打印工作时间为0的员工的整条记录
```

如果省略 pattern 部分，则每一行都执行对应的 action

```
awk '{print $1}' emp.data  #打印所有员工的姓名
```

运行 awk 程序

方式 1：awk ‘program’ input files

```
awk '$3==0 {print $1}' file1 file2
```

方式 2：awk ‘program’

```
awk '$3==0 {print $1}'   # 输入来自标准输入
Beth  4.00 10
Kathy 3.58 0
Kathy
```

方式 3：awk -f progfile input files

```
cat prog.awk
$3==0 {print $1}
awk -f prog.awk file1 file2
```

字段 (field) 和内置变量

- awk 默认每行为一条记录，记录内的字段分隔符为空格符或 tab 符
- 特定字段：$1,$2,...
- 整条记录：$0
- 当前记录字段个数：NF
- 当前记录最后一个字段：$NF
- 当前记录号：NR

示例

```
1. who | awk '{print NF,$1,$NF}'
2. awk '{print NR, $0}' emp.data
3. awk '{print "total pay for",$1,"is",$2*$3}' emp.data
```

更好的输出

示例

```
1. {printf("pay for %s is %2.2f\n", $1, $2*$3)}
2. {printf("%-8s $%6.2f\n", $1, $2*$3)}
3. awk '{printf("%6.2f  %s\n", $2*$3, $0)}' emp.data \
| sort
```

模式

- 模式匹配结果为真，则执行相应动作

1. `/regex/`
2. `!/regex/`
3. `$0~/regex/`
4. `$1!~/regex/`
5. `NF==0`
6. `NR%2!=0`
7. `(NR>5)&&(length($3)<30)`
8. `(NR==3),(NR==5)`
9. `/<[Hh][Tt][Mm][Ll]>/,/<\/[Hh][Tt][Mm][Ll]>/`
10. `1`

模式

示例

```
$2 >= 5
$2*$3 > 50{printf("$%.2f for %s\n", $2*$3, $1)}
$1 == "Susie"
/Susie/
$2 >=4 || $3 >=20
!($2 <4 && $3 <20)
```

BEGIN 和 END

- BEGIN 模式所对应的动作在 awk 处理第一行之前执行
- END 模式所对应的动作在 awk 处理完末行之后执行

示例

```
BEGIN {print "NAME    RATE  HOURS"; print ""}
      {print}
  END {print "Total:", NR, "records"}
```

计算

示例

```
# 计数：累计工作时间超过 15 小时的员工人数
$3>15 {emp = emp +1}
  END {print emp, "employees worked more than 15 hours"}

# 统计：计算员工总工资和平均工资
    {pay += $2*$3}
END {print NR, "employees"
     print "total pay is", pay
     print "average pay is", pay/NR
    }
```

处理文本

示例

```
# 打印时薪最高的员工姓名及其时薪
$2 > maxrate {maxrate = $2; maxemp = $1}
        END {print "highest pay rate:", maxemp, maxrate}

# 字符串连接：紧凑打印所有员工姓名
    {names = names $1 " "}
END {print names}

# 打印末行
    {last = $0}
END {print last}
```

内置函数

示例

```
# 打印每个员工的姓名长度
{print $1, length($1)}

# 统计行数、单词数和字符数
    {nc += length($0) + 1
     nw += NF
    }
END {print NR,"lines,",nw,"words,",nc,"chars"}
```

内置函数 (部分数值函数)

函数名	返回值
atan2(y,x)	y/x(-pi 到 pi) 的反正切
cos(x)	x 的余弦，x 为弧度值
exp(x)	e 的 x 次幂
int(x)	x 向 0 取整
log(x)	求 x 的自然对数 (以 e 为底)
rand()	[0,1) 之间的随机数
sin(x)	x 的正弦，x 为弧度值
sqrt(x)	x 的平方根
srand(x)	x 为 rand() 的新随机种子

数值函数使用举例

示例

```
1. randint = int(n*rand())+1 # 获得 1 到 n 之间的随机整数
2. x=int(x+0.5)  # 四舍五入取整
3. awk 'BEGIN{cos(60*3.1415926/180)}'
4. echo 1000 | awk '{print log($1)/log(10)}'
```

内置函数 (部分字符串函数)

函数名	函数描述
gsub(r,s[,t])	在 $0/t 中将 r 全局替换为 s, 返回替换次数
index(s,t)	返回 t 在 s 中首次出现的位置，返回 0 表示未找到
length(s)	返回 s 的长度
match(s,r)	测试 s 是否包含 r，返回 index 或 0
split(s,a[,fs])	将 s 拆分为数组 a，分隔符为 FS/fs，返回字段个数
sprintf(fmt,list)	返回按指定格式控制的字符串
sub(r,s[,t])	类似于 gsub(r,s[,t])，但仅替换 1 次
substr(s,p[,n])	返回 s 从位置 p 开始 [的长度为 n] 的子串

字符串函数使用举例

示例

```
1. awk 'BEGIN{print index("banana","an")}'
2. echo banana | awk '{gsub(/an/,"ok");print}'
3. echo banana | awk '{sub(/an/,"&d&");print}'
4. echo banana | awk '{print substr($1,2,2)}'
5. echo banana | awk '{print substr($1,3)}'
6. echo '10/01/2016' \
| awk '{split($0,date,"/");print date[3]}'
```

自定义函数

示例

```
{ print max($1, max($2,$3)) }
function max(m,n){
  return m>n ? m : n
}
```

控制流语句

- if-else

示例

```
$2 > 6 {n++; pay += $2*$3}
END    { if (n>0)
           print n, "employees, total pay is",pay,
                    "average pay is",pay/n
         else
           print "no employees are paid more than $6/h"
       }
```

控制流语句

- while

示例：计算银行复利

```
# interest1 - compute compound interest
#  input: amount rate years
# output: compounded value at the end of each year
{ i=1
  while (i <= $3) {
    printf("\t%.2f\n", $1*(1+$2)^i)
    i++
  }
}

awk -f interest1
1000 .06 5
```

控制流语句

- for

示例

```
# interest2 - compute compound interest
#  input: amount rate years
# output: compounded value at the end of each year
{ for (i=1; i<=$3; i++)
    printf("\t%.2f\n", $1*(1+$2)^i)
}
```

控制流语句应用举例

示例：查找相邻重复词 (包括行首与前一行末的重复)

```
cat double
NF>0 {
if ($1 == lastword)
  printf "%s\t%d: %s\n" , FILENAME, FNR, $1
for ( i=1;i<NF;i++ )
  if ( $i == $(i+1) )
    printf "%s\t%d: %s\n" , FILENAME, FNR, $i
lastword = $NF
}

awk -f double file1 file2 file3
```

数组

示例

```
# 反向输出每一行 (类似 tac 命令)   while 版
    { line[NR] = $0 }
END { i = NR
      while (i>0) {
        print line[i]
        i--
      }
    }
# 反向输出每一行 (类似 tac 命令)   for 版
    { line[NR] = $0 }
END { for (i=NR; i>0; i--)
        print line[i]
    }
```

其他简单示例

awk 简单程序示例

```
1. END {print NR} # 打印行数
2. NR==10         # 打印第 10 行
3. {print $NF}    # 打印每行最后一个字段
4. {field=$NF}END{print field} # 打印末行末字段
5. NF>4  # 打印多于 4 个字段的行
6. $NF>4 # 打印最后一个字段的值大于 4 的行
7. {nf=nf+NF}END{print nf} # 打印所有行字段个数之和
8. /Beth/{n++}END{print n} # 打印包含 Beth 的行数
9. $1>max{max=$1;maxline=$0}END{print max,maxline}
10. NF>0
11. length($0) > 80
12. {print NF, $0}
```

其他简单示例

awk 简单程序示例 (2)

```
13. {print $2, $1}
14. {temp=$1;$1=$2;$2=temp;print}
15. {$1=NR;print}
16. {$2="";print}
17. {for(i=NF;i>0;i--)printf("%s ", $i)
 printf("\n")
}
18. {sum = 0
 for (i=1;i<=NF;i++) sum+=$i
 print sum
}
19. {for(i=1;i<=NF;i++)sum+=$i}END{print sum}
20. {for(i=1;i<=NF;i++)if($i<0)$i=-$i}print}
```

指定字段分隔符

方法 1：-F 选项

```
awk -F: '/bash/{print $1,$2}' /etc/passwd
```

方法 2：在 BEGIN 中为 FS 内置变量赋值

```
BEGIN  {FS=':'}
/bash/ {print $1,$2}
```

awk 内置变量

变量名	含义
FILENAME	当前文件名
FS	输入的字段分隔符 (默认为空格符和 tab 符)
RS	输入的记录分隔符 (默认为换行符)
NF	当前记录字段个数
NR	当前记录的序号
FNR	当前文件的当前记录号
OFMT	数字的输出格式 (默认为%.6g)
OFS	输出的字段分隔符 (默认为空格符)
ORS	输出的记录分隔符 (默认为换行符)

awk 内置变量使用举例

示例

```
cat stu.db
tom
19
male

mary
18
female
```

请将stu.db的内容转换为如下格式：

```
tom:19:male
mary:18:female
```

关联数组

- 问题：现有客户订货记录 order.db 如下：

```
cat order.db
Susie 400
John 100
Mary 200
Mary 300
John 100
Susie 100
Mary 100
John 200
Mary 600
Susie 500
```

请统计每位客户的订货总数。

关联数组

统计每位客户的订货总数

```
awk '{sum[$1]+=$2}
  END{for(u in sum)print n,sum[n]}' order.db
```

试一试

请用 awk 进行词频统计，并打印出每个单词出现的次数。

综合实例

- 问题：现有学生成绩清单 score.list 如下：

```
jasper: 80 82 84 84 88 92
andrea: 85 89 90 90 94 95
ellis: 89 90 92 96 96 98
mona: 70 70 77 83 85 89
john: 78 85 88 91 92 94
dunce: 60 60 61 62 64 80
```

请统计：
(1)每位同学的平均分及等级(A,B,C,D,F)
(2)班平均成绩
(3)平均成绩高于或等与班平均的人数
(4)平均成绩低于班平均的人数
(5)每个等级的人数

综合实例

```
grep -v '^ *#' grade.awk
BEGIN { OFS = "\t" }
{
        total = 0
        for (i = 2; i <= NF; ++i)
          total += $i
        avg = total / (NF - 1)
        student_avg[NR] = avg
        if (avg >= 90)  grade = "A"
        else if (avg >= 80) grade = "B"
        else if (avg >= 70) grade = "C"
        else if (avg >= 60) grade = "D"
        else grade = "F"
        ++class_grade[grade]
        print $1, avg, grade
}
```

综合实例

```
END {
for (x = 1; x <= NR; x++)
        class_avg_total += student_avg[x]
class_average = class_avg_total / NR
for (x = 1; x <= NR; x++)
        if (student_avg[x] >= class_average)
          ++above_average
        else
          ++below_average
print ""
print "Class Average: ", class_average
print "At or Above Average: ", above_average
print "Below Average: ", below_average
for (letter_grade in class_grade)
        print letter_grade ":", class_grade[letter_grade]
  | "sort" }
```

综合实例

- awk -f grade.awk score.list 的输出结果如下：

```
jasper: 85      B
andrea: 90.5    A
ellis:  93.5    A
mona:   79      C
john:   88      B
dunce:  64.5    D

Class Average:  83.4167
At or Above Average:    4
Below Average:  2
A:      2
B:      2
C:      1
D:      1
```