第一节

UNIX 简介

一、 UNIX 常见命令

1. 日期与时间命令（date）

2. 日历命令（cal）

3. 谁在线命令（who）

Who 命令显示当前登录到系统的所有用户。

1. 不用参数，简单显示在线用户的情况

```
# who
root      pts/2        11 月 20 日 13:05 (192.168.50.22)
```

2. 使用-u 参数，显示在线用户 的使用情况

```
#who -u
root     pts/2        11 月 20 日 13:05   .     5330    (192.168.50.22)
```

3. 使用 H 参数，显示每一列内容的标题

```
# who -uH
名称      线           时间         闲置     PID    注释
root    pts/2     11 月 20 日 13:05   .     5330    (192.168.50.22)
```

4. 如果希望察看自己的信息，使用 am I 参数

```
# who am I
root      pts/2        11 月 20 日 13:05 (192.168.50.22)
```

4. 修改密码命令（passwd）

```
Passwd 用户名
```

一般用户有自我修改密码的权限。

5. 显示消息命令（echo）

6. 在线文档命令（man）

7. 打印命令（lpr）

```
Lpr options file-list
```

使用-P 选项指定打印机

```
#lpr file1                          //打印 file1 文件
#lpr file1 file2                    //打印 file1、file2 文件
#lpr –Plp0 file1 file2 file3        //使用 lp0 打印机打印 file1、file2、file3
```

8. 终端命令（tty）

实用程序 tty 用于显示正在使用的终端名称。

```
# tty
/dev/pts/2
```

9. 终端设置命令（stty）

1. 在不使用选项或参数的情况下设置终端

不使用选项或参数的情况下使用 stty 命令，显示终端的当前常用设置。

```
# stty
speed 38400 baud; -parity
```

```
行数=24, 列数=80; ypixels = 0; xpixels = 0;
swtch = <undef>;
brkint -inpck -istrip icrnl -ixany imaxbel onlcr tab3
echo echoe echok echoctl echoke iexten
```

2. 在只使用选项的情况下设置终端

终端设置命令包含两个选项（-a 和-g），这两个选项度不允许使用参数。使用-a 选项，该命令显示当前终端的选项设置。使用-g 选项，显示所选设置的格式为一种可用作另一个终端设置命令的参数格式。

```
# stty -a
speed 38400 baud;
行数=24, 列数=80; ypixels = 0; xpixels = 0;
csdata ?
eucw 1:0:0:0, scrw 1:0:0:0
intr = ^c; quit = ^\; erase = ^?; kill = ^u;
eof = ^d; eol = <undef>; eol2 = <undef>; swtch = <undef>;
start = ^q; stop = ^s; susp = ^z; dsusp = ^y;
rprnt = ^r; flush = ^o; werase = ^w; lnext = ^v;
-parenb -parodd cs8 -cstopb -hupcl cread -clocal -loblk -crtscts -crtsxoff -par
-ignbrk brkint ignpar -parmrk -inpck -istrip -inlcr -igncr icrnl -iuclc
ixon -ixany -ixoff imaxbel
isig icanon -xcase echo echoe echok -echonl -noflsh
-tostop echoctl -echoprt echoke -defecho -flusho -pendin iexten
opost -olcuc onlcr -ocrnl -onocr -onlret -ofill -ofdel tab3

# stty -g
2506:1805:bf:8a3b:3:1c:7f:15:4:0:0:0:11:13:1a:19:12:f:17:16:0:0:1:1:0:00:0:0:0
:0
```

3. 使用参数情况下设置终端

- 设置清除与取消（ek） ek 参数使删除和取消设置返回(删除:Ctrl+h 取消:Ctrl+u）
- 将终端设置通用配置（sane） sane 参数江终端配置设置为适用于大多数终端的合理配置

```
# stty sane
```

- 设置清除键（earse） 默认情况下终端清除键为 Ctrl+h，可以使用此参数进行更改。

```
# stty earse^e
```

- 设置取消键（kill） 取消键用来删除整行的，默认情况下为 Ctrl+u，可以使用此参数修改。

```
# stty kill9
```

- 设置中断键（intr） 中断键中断或挂起一个命令。默认情况下为 Ctrl+c，可以使用此参数修改。

```
# stty intr^9
```

10. 记录会话命令（script）

Script 用于记录一个交互式会话，默认将记录保存为文件 typescript,也可以指定文件，追加记录使用-a 参数。

11. 系统名称命令（uname）
12. 计算器命令（bc）
    1. 简单数学运算

       支持加（+）、减（-）、乘（*）、除（/）、取模（%）、幂（^）

       结果默认四舍五入取整
    2. 浮点运算

       使用 scale 设置小数位

       34-57/2

       6

       scale=5

       34-57/2

       5.50000
    3. 算术数制运算

       Ibase 更改输入数制

       Obase 更改输出数制

       在不进行更改的情况下默认输出都为十进制

       退出使用 quit

       ibase=2

       111

       7

       1111111

       127

第二节

Vi 编辑器基础

一、 模式

vi 编辑器使用两种基本模式：命令模式和文本模式。

1. 命令模式

当 vi 处于命令模式下，用户按下的任何键都被看作是一个命令。命令一经输入就被执行，不需要回车。

2. 文本模式

当 vi 处于文本模式下，用户按下的任何键都被看作是文本。

3. 变更模式

- 当激活 vi 时，vi 处于命令模式。在会话期间可以在命令模式和文本模式之间来回切换
- 要退出 vi，则必须处于命令模式
- 可以使用命令 ***（a、A、i、I、o、O）*** 六个命令将 vi 切换至文本模式
- 使用 Esc 键将 vi 从文本模式切换成命令模式

二、 命令

1. 添加命令

a. 插入命令（***i*** 和 ***I***）

可以在当前位置之前或者当前行的开始处插入文本。

小写命令 ***i*** 光标当前位置插入文本。

大写命令 ***I*** 在当前行开始处插入文本

b. 追加命令（***a*** 和 ***A***）

小写命令 ***a*** 在当前字符之后追加文本

大写命令 ***A*** 在当前行行尾之后追加文本

c. 换行命令

小写命令 ***o*** 在当前行下面打开一个新行

大写命令 ***O*** 在当前行上面打开一个新行

2. 光标移动命令

光标移动命令只有在命令模式下才生效，执行完光标移动命令 vi 仍处于命令模式下

a. 水平移动

左移： ***h；←；Backspace***

右移： ***l；→；Spacebar***

b. 移动至行首（***0***）或行尾（***$***）

移动至行首：使用数字键 ***0***

移动至行尾：使用符号 ***$***

c. 垂直移动命令

上移： ***k；*** ↑

下移； ***j；*** ↓

d. 上行或下行

上行命令，直接移动到当前行的上行行首： ***-***

下行命令，直接移动到当前行的下行行尾； ***+***

3. 删除命令

删除当前字符： ***x***

删除光标前一个字符： ***X***

删除当前行： ***dd***

4. 合并命令

使用命令（***J*** ）合并两行，可以在第一行的任意位置使用该命令。

5. 滚动命令

命令	功能
Ctrl+y	向上滚动一行
Ctrl+e	向下滚动一行
Ctrl+u	向上滚动半屏（回行）
Ctrl+d	向下滚动半屏（回行）
Ctrl+b	向上滚动整屏（24 行）
Ctrl+f	向下滚动整屏（24 行）

6. 撤销命令

u: 只撤销上一次编辑

U: 撤销当前行的全部修改

7. 保存和退出命令

a. 保存并继续（***: w***）

b. 保存至新文件（***: w filename***）

c. 保存并退出（***ZZ***）

d. 写并退出（***: wq***）

e. 退出（***: q***）

f. 退出但不保存（***: q!*** ）

第三节

文件系统

一、 文件名

在 UNIX 中，命名文件时受到的限制较少，一般情况下文件名可达 255 个字符，文件名可以是任何 ASCII 字符的组合，但最好不要使用有特殊意义的字符，比如 <;>;.等。

建议遵循以下简单规则：

- 文件名开头使用字母字符
- 用分割符分割文件名不同的部分。有效的分割符包括下划线、句点和连字符。
- 在文件名尾使用一个扩展名，例如文本文件使用.txt，数据文件使用.dat，二进制文件使用.bin，c 和 C++使用.c 和.c++。
- 不要使用句点作为文件名的开头，以句点开头的文件名表示隐含文件。

通配符

匹配任意单一字符 ?

匹配集合中的单一字符 [.....]

匹配 0 个或多个字符 *

二、 文件类型

- 常规文件：包含可供用户未来处理的用户数据。
- 目录文件：包含所有存储于物理设备的文件名称和位置的文件。
- 字符特殊文件：字符特殊文件代表一种物理设备，如终端。
- 块特殊文件：块特殊文件代表一种物理设备，如磁盘
- 符号链接文件：符号链接文件是一种逻辑文件，它定义另一个位于系统其他位置文件位置
- FIFO 文件：一种先进先出文件，也称命名管道，用于进程间通讯的文件。
- 套接字（Socket）：套接字是一种用于网络通讯的特殊文件。

三、 常规文件

1. 文本文件
2. 二进制文件

四、 目录

1. 特殊目录
   a. 根目录
   b. 用户根目录
   c. 工作目录
   d. 父目录
2. 路径和路径名
   a. 绝对路径名

   绝对路径名指定从根至预期目录或文件的完整路径。以斜线开始的路径名都是绝对路径名

   b. 相对路径名
3. 相对路径名缩写

   a. 用户根目录（~）

   b. 工作目录（.）

   c. 父目录（..）

d. 创建相对路径

五、 文件系统实现

1. 文件系统

文件系统包括 4 个称为块的结构化区：启动块、超级块、索引节点（inode）块和数据块。启动块、超级块、索引节点块固定位于磁盘的开始处。

a. 启动块

位于磁盘开始处，包含一个称为启动程序的小程序。

b. 超级块

磁盘中的第 2 个块，包含文件系统信息，信息包括：磁盘总容量、空闲块数量和坏磁盘块的位置。

c. 索引节点块

超级块之后就是索引节点块，它包含关于数据块中各个文件的信息。如文件的拥有者、权限，对应地址等。

d. 数据块

首先，包含用户的所有文件和用户的数据。其次，包含与用户文件相关的特殊文件：常规文件、目录文件、符号链接文件和 FIFO 文件。最后，包括字符特殊、块特殊和套接子系统文件。

2. 目录内容

3. 链接

链接是索引节点和文件之间的一种逻辑关系，它将文件名与物理位置联系起来。

a. 硬链接

目录中的索引节点将文件名直接链接之物理文件。这种方式为多文件链接提供了基础。

b. 符号链接

符号链接（软链接）是索引节点通过一种称为符号链接的特殊文件与物理文件相关联。符号链接不如物理链接有效。符号链接为两种特殊情况而设计的：链接至目录，以及链接职位于其他文件系统的文件。

c. 多链接

索引节点设计的一个优点是可以链接两个或多个不同文件名至一个物理文件。多链接结构是一种方便有效的文件共享方法。

d. 当前目录和父目录

六、 目录特有操作

1. 定位目录命令（pwd）

pwd 命令可以确定当前目录所在目录结构中的位置，此命令不包含选项和属性。结果是显示当前目录的绝对路径。

```
# pwd
/etc/vg
```

2. 目录列表命令（ls）

目录列表命令列出某个目录的内容。

a. 长列表

```
# ls -l
total 1576
-rw-r--r--   1 root     system          2 Jan 03 07:53 .init.state
```

```
-rw-rw-r--   1 root     system        2721 Jan 01 08:33 3270.keys
-rw-rw-r--   1 root     system        3968 Jan 01 08:33 3270_arab_kyb.map
-rw-rw-r--   1 root     system        5754 Jan 01 08:33 3270keys.dtterm
-rw-rw-r--   1 root     system        5528 Jan 01 08:33 3270keys.hft
drwxr-xr-x   6 imnadm   imnadm         512 Jan 01 08:56 IMNSearch
lrwxrwxrwx   1 root     system          17 Jan 01 08:33 aliases -> /etc/mail/as
-rw-r--r--   1 root     system        6397 Jan 01 09:05 basecust
-rw-rw-r--   1 root     system       23874 Jan 01 08:43 binld.cnf
```

文件类型标记

文件标记	文件类型
-	常规文件
d	目录
c	字符特殊
b	块特殊
l	符号链接
p	FIFO
s	套接字

b. 列表选项

- 显示全部（a）
- 工作目录（d）

  与长列表选项（l）一同使用显示当前目录属性

  ```
  # ls -ld
  drwxrwxr-x   2 root     system         512 Jan 01 08:32 .
  ```

- 显示用户和用户组 ID（n）

  与工作目录选项一起使用显示用户 ID 和用户组 ID

  ```
  # ls -nd
  drwxrwxr-x   2 0        0              512 Jan 01 08:32 .
  ```

- 逆序（r）

  逆序选项（-r）按降序排列显示文件。

```
# ls
HTTPServer  bin         include     lost+found  sbin        sysv
IMNSearch   ccs         java130     lpd         share       tmp
TT_DB       doc         lbin        lpp         spool       ucb
X11R6       docsearch   lib         man         src         usg
adm         dt          linux       opt         swlag       websm
aix         etc         local       samples     sys
```

```
# ls -r
websm       swlag       opt         linux       dt          adm
usg         src         man         lib         docsearch   X11R6
```

ucb	***spool***	***lpp***	***lbin***	***doc***	***TT_DB***
tmp	***share***	***lpd***	***java130***	***ccs***	***IMNSearch***
sysv	***sbin***	***lost+found***	***include***	***bin***	***HTTPServer***
sys	***samples***	***local***	***etc***	***aix***	

- 时间排序

  按时间排序有三种方法：基本时间排序（-lt）；按最近访问时间排序；按索引节点日期变更排序（-lc）。
- 标示目录（-p）
- 递归列表（-Rp）
- 显示索引节点号（-i）
- 单列显示（-1（此为数字 1））

3. 目录创建命令（mkdir）

   包含两个选项参数，一个（-m）控制新目录的权限，另一个父目录选项（-p）
4. 改变目录命令（cd）
5. 目录删除命令（rmdir）

   此命令只有在目录为空的时候才能删除目录。

七、 常规文件特有操作

1. 创建文件
2. 编辑文件

   如 vi，sed 等
3. 文件显示命令（more）

   More 命令的现实选项

选项	解释
-c	显示之前清屏
-d	显示出错信息
-f	对长行不做屏幕换行
-l	忽略换页符
-r	以格式^C 显示控制字符
-s	对多个空行进行压缩
-u	取消文本下划线
-w	在输出结束处等待，以便用户输入任何键来继续
-line	设置一屏所显示的行数（默认值为屏幕大小-2）
+number	从指定行号开始输出
+/ptrn	定位第一个模式出现的位置，并从该位置前两行处开始输出

more 命令的继续显示选项

选项	解释
Space	显示下一屏输出
n+space	显示下一屏并设置屏幕大小为 n 行
return	向下多显示一行
d	显示半屏

nf	跳过 n 屏且显示一屏
nb	后移 n 屏且显示一屏
q	推出 more 命令
=	显示当前行号
: f	显示当前文件名和行号
v	在当前位置转换到 vi 编辑器
。	重复前一个命令

4. 打印文件（lpr）

八、 目录和文件的共有操作

1. 复制命令（cp）
   a. 保持属性（-p）
   b. 交互式选项（-i）
   c. 递归选项（-r）
2. 移动命令（mv）
   a. 交互式选项（-i）
   b. 强制式选项（-f）
3. 链接命令（ln）
   a. 符号选项(-s)
      默认链接类型是硬链接。要创建一个符号链接，须使用符号连接选项（-s）
   b. 交互式选项（-i）
   c. 强制式选项（-f）
   d. 硬链接
      i. 至文件的硬链接
         为创建一个至文件的硬链接，需要制定源文件和目标文件。如果目标文件不存在，则创建它。如果存在，则首先删除它，然后重新创建一个链接文件，
         #ln file1 lnDir/linkedFile
      ii. 硬链接到目录
         Unix 不允许硬链接到目录
   e. 符号链接
      符号链接物理上并不存在，只是指向真实的目录或文件。如果物理文件被删除，则该文件不再以原来的名称出现，但其符号链接的名义下依然可见，但无法存取它
      i. 至文件的符号链接
         #ln –s file2 lnDir
      ii. 至目录的符号连接
         创建至目录的符号链接的代码与创建至文件的代码相同
4. 删除命令（rm）
   删除实用程序（rm）通过删除条目到文件的链接，但可能存在一个文件有多个链接的情况，只有当删除文件所有的相关链接后，该文件才被物理删除。
5. 文件查找命令（find）

查找条件完整列表

条件	匹配与……
-name file	文件名
-perm nnn	权限 nnn，nnn 必须是八进制的数字
-perm –nnn	位屏蔽权限 nnn，
-type c	文件类型。有效的文件类型有：块（b）、字符（c）、目录（d）、链接（l）、管道（p）、文件（f）和套接字（s）
-link n	某个文件的链接数
-user name	用户名
-nouser	/etc/passwd 文件中不包含的用户名
-group gname	用户组名
-nogroup	/etc/group 不包含的用户组名
-size n|nc	以块（n）或字符（nc）为单位的文件精确大小
-atime +n|-n	n 天或 n 天前（+n），或者最近 n 天（-n）已被存取过的文件。注意 find 命令更改文件的存取时间
-mtime +n|-n	n 天或 n 天前（+n），或者最近 n 天（-n）已被修改过的文件
-ctime +n|-n	n 天或 n 天前（+n），或者最近 n 天（-n）已被更改的文件
-newer file	文件修改日期迟于文件日期
-print	在标准输出中显示绝对路径
-exec	执行制定的命令
-ok	执行指定的命令，并在执行前请求 yes/no 确认
-prune	一旦查找到一个匹配的文件，则终止对当前目录及其子目录的文件检查

第四节

安全性与文件权限

一、 用户与用户组

用户组命令：

Unix 提供一个 groups 命令来确定一个用户的用户组。如果输入不带用户 id 的 groups 命令，则系统一您的用户组作响应；如果输入带用户 id 的 groups 命令，则它返回该用户的用户组。如果一个用户属于多个用户组，则列出所有用户组。

二、 安全性等级

Unix 系统包含三种安全性等级：系统、目录和文件。

1. 系统安全性

在创建用户的时候，会将登录名、密码、用户 id 等重要信息写入文件/etc/passwd

2. 权限码

权限码分为三组，第 1 组包含目录或文件拥有者的权限，第 2 组包含由用户 id 标识的用户组成员的用户组权限，第三组包含其它任何用户的权限。权限分为读、写、执行。

a. 目录等级的权限

读权限：用户可以拥有某个目录的读权限

写权限：可以添加或删除该目录的条目。

执行权限：又称为搜索权限，为了引用某一目录下的一个子目录或文件，必须拥有该目录或文件的绝对路径名中全部目录的执行权限。因此，目录的用户权限通常包括读权限和执行权限。

b. 文件等级的权限

读权限：拥有文件读权限的用户可以读或复制文件。

写权限：可以修改和删除文件

执行权限：可以执行程序、脚本。

3. 权限检查

使用长列表命令（ls -l）查看文件和目录的权限。

三、 修改权限（chmod）

1. 符号代码
2. 八进制代码

r:4

w:2

x:1

3. 选项

四、 用户掩码

1. 基本概念
2. 用户掩码命令（umask）

Umask 命令用于显示并设置用户掩码。

显示用户掩码设置，使用不带参数的 umask 命令。设置，使用带新掩码设置的 umask 命令。

```
# umask
0022
```

在会话中设置用户掩码，则只在该会话期间暂时有效。退出系统时，则设置返回至默认值。

五、 改变拥有者和用户组

1. 改变拥有者（chown）

2. 改变用户组命令（chgrp）

第五节

Shell 简介

unix 操作系统包括四种不同的部分：内核、shell、实用程序和应用程序。

Shell 包括两个主要部分：第一部分是解释程序，解释程序读取用户的命令，并操作内核执行。第二部分，开发 shell 脚本的程序设计功能。

一、 Unix 会话

1. 登录 shell 验证]

   Unix 中包含一个系统变量 SHELL,它标识登录 shell 的路径。使用如下命令验证：

   # echo $SHELL

   /sbin/sh

2. 当前 shell 验证

   验证当前 shell 可以使用如下命令，但此命令只适用于 Korn 和 Bash shell，不适用于 C shell。

   # echo $0

   ksh

3. shell 关系

   从子 shell 退回到父 shell 使用 exit 命令

   # exit

4. 退出系统

   # logout

二、 标准流

Unix 定义三种由命令使用的标准流，每种标准流分配一个描述符。标准输入描述符为 0；标准输出描述符为 1；标准错误描述符为 2。

三、 重定向

1. 输入重定向

   输入重定向运算符为小于号（<）。

   表示输入重定向有两种方法：

   第一种方法显示指定，借助描述符 0 将重定向应用于标准输入。如：***command 0<file1***

   第二种方法省略该描述符，由于只存在一种标准输入，所以可以省略。如：***command <file1***

   注：在描述符和重定向符号之间不包含空格

2. 输出重定向

   输出重定向运算符为大于号（>）。

   输出重定向有三种方式：

   直接使用输出重定向符(>)指向输出文件名，当输出指向的是一个不存在的文件时，则系统创建并写入输出；如果该文件已存在，则系统的行为取决于 noclobber 参数的设置。当 noclobber 被置位时，它将禁止重定向破坏现有文件。此方式的格式如：***command 1 > file1*** 或 ***command > file1***

   如果想要覆盖该参数并用新的输出替代当前文件，则必须使用重定向覆盖运算符，即大于号和竖线（>|），在这种情况下系统首先清空当前文件，然后将新的输出写入该文件。格式如下：

   command >| file1

   如果想要将输出结果追加至该文件，则重定向符号是两个大于号（>>）。格式如下：***command >> file1***

3. 错误重定向

四、 管道

管道是一种运算符，它将一个命令的输出临时保存在缓冲区，并作为下一个命令的输入。左端明亮必须能够发送数据至标准输出，而右端命令必须能够从标准输入接收数据。管道符号是一个垂直竖线（|）。

```
# who |more
```

管道是一个运算符，而不是一个命令。它告知 shell 立即接收第一个命令的标准输出，该输出必须被发送到标准输出，并且让其进入第二个命令的输入。

五、 tee 命令

tee 命令复制标准输入至标准输出，并且同时将其复制一个或多个文件。如果输出文件不存在，tee 命令创建它，如果他们存在，则覆盖他们。为了防止文件被覆盖，可以使用-a 选项，使输出追加到现有文件，而不是删除它们当前的内容。但是，注意追加选项不适用于标准输出，因为标准输出总是自动追加的。

```
# tee teeout
this is a sample of the tee command.You see each line repeated.
this is a sample of the tee command.You see each line repeated.
on the standard output as soon I key enter.Each line is also.
on the standard output as soon I key enter.Each line is also.
written to teeout
written to teeout
# more teeout
this is a sample of the tee command.You see each line repeated.
on the standard output as soon I key enter.Each line is also.
written to teeout
```

六、 命令执行

在执行一个命令前，shell 什么也不做。当用户在命令行输入一个命令，解释程序分析该命令并引导他执行某个实用程序或其它应用程序。

1. 顺序命令

可以在单行中输入一个命令顺序。必须使用一个分号将每一个命令与其前一个分隔。命令之间不存在直接的关系：也就是说，一个命令不与其它命令通信。他们只不过被组合为一行并执行。

```
# echo "\n Goblins & Ghosts \n     Month" >Oct2000;cal 10 2000 >> Oct2000
# more Oct2000
 Goblins & Ghosts
     Month
   October 2000
 S  M Tu  W Th  F  S
 1  2  3  4  5  6  7
 8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30 31
```

2. 分组命令

以上事例也可以使用分组命令实现

```
# (echo "\n Gobling & Ghosts \n   Month";cal 10 2000)>Oct2000_1
# more Oct2000_1
 Gobling & Ghosts
    Month
   October 2000
 S  M Tu  W Th  F  S
 1  2  3  4  5  6  7
 8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30 31
```

3. 链式命令

链式命令是命令之间使用管道连接，命令之间存在直接关系。第一个命令的输出成为第二个命令的输入。

4. 条件命令

可以使用条件关系组合两个或多个命令。存在两种 shell 逻辑运算符：和（&&）与或（||）。通常，当以逻辑和组合命令时，则只在第一个命令成功的条件下才执行的 2 个命令。而，如果使用逻辑或组合命令时，则只用在第一个命令失败的条件下才执行第 2 个命令。

```
# ls
Oct2000    Oct2000_1   newtext    substr.scr  teeout     typescript
# cp newtext newtext1 && echo "Copy successful"
Copy successful
# ls
Oct2000    newtext    substr.scr  typescript
Oct2000_1  newtext1   teeout
# cp Oct2001 Oct2002 || echo "Copy failed"
cp: cannot access Oct2001
Copy failed
```

七、 命令行编辑

1. 命令行编辑概念
2. 编辑器选择

可以通过编辑/etc/profile 文件为系统设置默认的命令行编辑器。也可以在使用命令行时设置，但他只作用于当前会话。

使用 set 命令设置命令行编辑器。

```
# set -o vi
```

或

```
# set -o emacs
```

以上两种方式只可以选用其中之一。

3. 行编辑器 vi
4. 使用命令行编辑器

八、 引号

Shell 在命令中使用一个经选择的元字符集。所谓元字符是拥有特殊解释的字符。例如：管道（|）。除了用于与 shell 通讯，元字符被广泛用作文本。因此，当想要将他们用作元字符时，以及当想要将他们用作文本时，需要某种特殊的方法告知 shell 解释程序。三种被称作引号的元字符发挥了这种作用：反斜杠、双引号和单引号。

1. 反斜杠

   元字符反斜杠改变紧跟其后的字符解释——他将文字字符转化为特殊字符，以及将特殊字符转化为文字字符。反斜杠之改变一个字符——紧跟其后的一个字符。

2. 双引号

   当需要改变几个字符的含义，可以使用双引号。双引号去掉大多数元字符的特殊解释。在一个变量名前的美元符号以及单引号是例外。双引号保留诸如空格、制表符和换行符等空白字符。

3. 单引号

   单引号类似于双引号，但他们的效果更强。他们只保留但引号的含义，任何被包围的元字符都被视作文字字符。单引号和双引号必须成对出现。

九、 命令替代

命令替代提供了将某个命令结果转化为字符串的功能。转化某个命令的结果为字符串的命令替代运算符是一格美元符合一对括号。将命令置于一对括号内，并且以美元符为先导。当使用命令替代时，命令被执行，并且使输出杯创建并且转化至字符串。例如：

```
# echo date
date
# echo $(date)
Wed Jan 7 09:46:56 BEIST 1970
```

C shell 中的命令替代是将命令包围在一对反引号中。如下例所示：

```
# echo `date`
Wed Jan 7 09:49:53 BEIST 1970
```

十、 作业控制

Shell 的重要特性之一就是作业控制。

1. 作业

   通常，作业就是计算机中运行的一项用户任务。UNIX 中包含一种特殊的作业定义：作业是在命令行中输入的一个或一组命令。

2. 前台与后台作业

   a. 前台作业

      前台作业时运行在用户的有效监控之下的任何作业。它是由用户启动并且可能经由标准输入和标准输出与用户交互。在它运行的过程中，无法启动其他的作业。

      - 挂起一个前台作业

        在前台作业运行的过程中，它可以被挂起。

        挂起一个前台作业按下 Ctrl+z，恢复前台作业命令 fg。

      - 终止一个前台作业

        终止一个正在运行的前台作业使用组合键 Ctrl+c，在终止后，键入回车键激活命令行提示符。

   b. 后台作业

      启动一个后台作业，在命令的后面紧跟&，如果该作业需要参数，则&紧跟在最后一个参数之后。

      - 挂起、重启和终止后台作业

      挂起后台作业使用 stop 命令

重启一个挂起的后台作业使用 bg 命令。

终止后台作业使用 kill 命令。

以上三种命令需要以一个百分号（%）打头的作业号。如：

```
$longjob.scr&
[1]     1795841
$stop %1
[1]+ 1795841        stop(SIGSTOP)   logjob.scr&
$bg 51
[1]     logjob.scr&
$kill %1
[1]+     Terminated
```

- 在前后台之间移动作业

为了在前后台之间移动一个作业，则该作业必须被挂起。一旦该作业被挂起，可以使用 bg 命令将从挂起状态它移动到后台。用于作业位于前台所以不需要作业号。为了移动一个后台作业至前台，使用 fg 命令。

3. 作业命令

使用 jobs 命令能够显示系统中所有作业，无论它是处于运行还是挂起状态。它将显示每个作业的作业号、当前值和状态。

```
$jobs
[4]+Stopped(SIGTSTP)        longJob.scr
[3]-Running                 bgCount200.scr&
[2] Running                 bgCount200.scr&
[1] Running                 bgCount200.scr&
```

a.当前值标志

作业 4 在第二列包含一个加号，作业 3 在第二列包含也各减号。这些符号成为当前值标志。如果某个命令在输入时没有给出作业号，加号就表示该作业为默认作业。减号表示如果第一个作业结束那一个作业将成为默认作业。被挂起的作业被自动给予默认当前值标志。如果两个作业被挂起，则最近被挂起的作业为加号，较早的为减号。如果不存在被挂起的作业，则较好被赋予最近启动的作业。

b.使用作业号

执行 stop 和 kill 命令需要一个作业号，而 fg 和 bg 命令只有当存在多个作业时才需要作业号。除了适用作业号外，默认作业也可以表示为%+或%%,包含减号当前值的作业可以表示为%-。

c.作业状态

如何时候，一个作业可能处于三种状态之一：前台、后台或挂起。

4. 进程 ID

作业号与用户会话和终端相关，他们不是全局的。进程标示符（PID），是 UNIX 系统给定的全局性的标示符。

十一、 别名

通过将一个名称分配给命令，别名提供了一种创建定制命令的方法。

1. Korn shell 和 Bash shell 中的别名

别名通过 alias 命令创建的。其格式为：

```
alias  name=command-definition
```

其中，alias 是命令的关键字，name 是要创建的别名的名称，而 command-definition 是代码。

例如：

```
# alias dir=ls
# alias dir='ls -l'
# alias dir="ls -l |more"
# alias dir=ls
# alias lndir='dir -l|more'
```

a. 别名的参数

只要不存在歧义，参数可以被传递给一个别名。参数被添加到命令之后，参数可能包含合适的通配符。

b. 列表显示别名

为了列出全部的别名，使用不带参数的 alias 命令。

c. 删除别名

别名通过 unalias 命令删除的。它包含一个参数，即一个要删除的别名列表。当使用-a 参数时，则删除全部别名。

2. C shell 中的别名

C shell 与 K shell 的区别在于格式，而不是功能。其格式为：

```
alias name definition
```

3. 别名小结

功能	Korn shell 和 Bash shell	C shell
定义	$alias x=command	%alias x command
参数	至位于末尾	任何位置
列表显示	$alias	%alias
删除	$unalias x y z	%unalias x y z
全部删除	$unalias –a	%unalias *

十二、 变量

变量是一个可以保存值得内存地址。每种 shell 都允许创建、保存和访问位于变量中的值。每个 shell 变量都必须拥有一个名字。变量名必须以一个英文字母或下划线字符开头，后接一个或多个字母或者下划线字符。存在两大变量类型：用户定义变量和预定义变量。

1. 用户定义变量

用户变量并非专门定义的，在对变量第一次引用时创建它。

2. 预定义变量

预定义变量用于配置一个用户的 shell 环境。

3. 在变量中保存数据

操作	Korn shell 和 Bash shell	C shell
赋值	variable=value	set variable=value
引用	$variable	$variable

4. 访问变量

十三、 预定义变量

与定义变量用于配置一个用户的 shell 环境。预定义变量可以分为两类：shell 变量和环境变量。Shell 变量用于定制 shell，环境变量用户控制用户环境，并且可以导出至子 shell。

Korn shell和Bash shell	C shell	解释
CDPATH	Cdpath	当目录参数是一个相对路径名时，包含 cd 命令搜索路径
EDITOR	EDITOR	命令行编辑器的路径名
ENV		环境文件的路径名
HOME	home(HOME)	用户根目录路径名
PATH	path(PATH)	命令的搜索路径
PS1	prompt	主要提示符，如$和%
SHELL	shell(SHELL)	登录 shell 的路径名
TERM	term（TERM）	终端类型
TMOUT	autologout	定义自动注销之前等待的时间，单位为秒
VISAUL	VISUAL	命令黄编辑器的路径，同 EDITOR

1. CDPATH

   变量 CDPATH 包含一个有冒号分隔的路径名列表。如下所示：

   :$HOME:/bin/usr/files

2. HOME

   HOME 变量包含至用户根目录的 PATH。默认值是登录目录。

3. PATH

   PATH 变量用于搜索一个命令的目录。位于 PATH 变量的条目必须以冒号分隔。如下所示：

   $PATH=/bin:/usr:/bin::

4. 主提示符（PS1 Prompt）
5. SHELL

   登录 SHELL 保存您的登录 shell 的路径

6. TERM

   变量 TERM 保存对您正在使用终端的描述。改变量的值可被交互式命令使用。

操作	Korn shell 和 Bash shell	C shell
设置	var=value	set var=value (setenv var value)
取消设置	unset var	unset var (unsetenv var)
显示一个	echo $var	echo $var
显示全部	set	set(setenv)

十四、 选项

Korn shell 和 Bash shell	C shell	解释
noglob	noglob	禁止通配符展开

verbose	verbose	在命令执行前显示他们
xtrace		在命令执行前显示命令和参数
emacs		将 emacs 用于命令行编辑
ignoreeof	ifnoreeof	不允许使用 Ctrl+d 推出 shell
noclobber	noclobber	不允许重新定向损坏的现有文件
vi		将 vi 用于命令行编辑

处理选项：

用于显示、设置和取消设置选项的命令

操作	Korn shell 和 Bash shell	C shell
设置	set –o option	set option
取消设置	set +o option	unset option
显示全部	set –o	set

#set –o verbose 设置

#set +o verbose 取消设置

十五、 Shell 环境定制

1. 暂时定制
2. 永久定制

第六节

过滤器

在 unix 中，过滤器是一个命令，他从标准输入流中取得输入，对输入进行操作，然后再把结果发送到标准输出流中。

过滤器	动作
more	把所有输入数据传递到输出，并且在每一个数据屏的结尾处停顿
cat	把所有的输入数据传递到输出
cmp	比较两个文件
comm.	标记两个文件中的公用行
cut	只传递特定行
diff	标记两个文件之间的差别或两个目录下公用文件的差别
head	传递从数据开始处开始值得的行数
paste	结合列
sort	将数据排序
tail	传递从数据结尾处开始指定的行数
tr	转换一个或多个字符为特定字符
uniq	删除重复行
wc	对字符、单词或行计数
grep	只传递指定行
sed	传递被编辑行
awk	传递被编辑行，并解析行

一、 过滤器和管道

过滤器本质上可以结合管道工作。换句话讲，过滤器可以在管道的左边、两个管道之间以及管道的右边。

二、 连接文件

UNIX 提供用于连接名的强大工具，也可以成为连接命令，缩写为 cat。它按照命令列出的顺序把一个文件或多个文件连接起来。

1. 连接命令（cat）

给定一个或多个输入文件时，cat 命令将他们逐个写到标准输出。结果所有输入文件被结合起来，成为一个输出。

```
# ls
file1  file2  test1  test2
# more file1
This is file1
# more file2
This is file2
# cat file1 file2
This is file1
This is file2
```

a. 用 cat 显示文件

一个基本用法是使 cat 成为显示文件的可用工具。当只给出一个输入文件时，结果是输入文件被显示在屏幕上。

b. 用 cat 创建文件

第二个使用 cat 的特殊应用是创建文件。虽然还只是只有一个输入，但这次输入来自键盘，因为要保存文件，所以要重新定向标准输出到一个文件而不是显示器。输入完成后，使用键盘命令 Ctrl+d 键标识。注意，该标记必须是最后一行中的第一个字符，否则，将变成输出的一部分。

```
# cat >file3
This is file3
goodbye
```

2. cat 选项

cat 有 6 个选项，可被分为 4 组：可视字符、缓存输出、缺失文件和编号行。

a. 可视字符

有时显示输出需要看到所有的字符，如果文件包含非打印字符，或行尾有空字符，我们将不能发现。使用-v 选项帮助我们看到控制字符，但依然不能看到制表符、换行符和换页符。不过，使用-ve 选项，则可以在每行结尾处打印一个美元符号（$），这是可以很容易的看到制表符和结尾的空格。如果使用-vt 选项将可以看到制表符显示为^I。

```
# cat file3
This is file3
goodbye
1      2      3      4      5
good good study!
yes
# cat -vet file3
This is file3$
goodbye$
1^I2^I3^I4^I5$
good good study!   $
yes $
```

b. 缓存输出

当输出被缓存时，在系统将其写入一个文件前，她一直被保存在计算机中。正常情况下 cat 输出都被缓存，可以通过指定不进行缓存的-u 选项，他会强制将输出写入一个文件中。

c. 缺失文件

当把几个文件连接到一起时，如果其中一个文件缺失，系统将提示信息。如果不想在输出重包含给信息，可以使用-s 选项。

d. 编号行

当文件被写入标准输出时，编号行选项（-n）将每一个文件中的每一行进行编号。如果写入的是多个文件，则编号将从每个文件重新开始。

3. 显示文件的开始和结尾

UNIX 提供两个命令，head 和 tail，显示文件的各个部分。

a. head 命令

head 命令把一个或多个文件从开始到指定行数的内容复制到标准输出流中。使用选项指定输出的行数。如果行数缺失，head 假定为 10 行。如果 head 命令中包含多个文件，head 会在文件输出前显示他的名字。

```
# more teeout
this is a sample of the tee command.You see each line repeated.
on the standard output as soon I key enter.Each line is also.
written to teeout
# head -2 teeout
this is a sample of the tee command.You see each line repeated.
on the standard output as soon I key enter.Each line is also.
```

b. tail 命令

tail 命令从文件的结尾向上计算。tail 命令的一般格式如下：

```
tail  options  input files
```

tail 中可用的选项

选项	代码	描述
从开始计数	+N	跳过 N-1 行，复制其余部分到文件结尾
从结尾计数	-N	复制最后 N 行
按行计数	-l	按行计数
按字符计数	-c	按字符计数
按块计数	-b	按磁盘块计数
逆向计数	-r	以逆向顺序（从底向上）输出

4. cut 和 paste

虽然 cut 和 paste 命令在 Unix 中与文本编辑器的剪切和黏贴操作是不同的概念。但是，这个两个命令对文件执行类似的操作，cut 从文件中删除数据列，paste 则连接数据列。

Chicago	IL	2783726	300572	1434029
Houston	TX	1630553	1595138	1049300
Los Angeless	CA	3485398	2968528	1791011
New York	NY	7322564	7071639	3314000
Philadelphia	PA	1585577	1688210	1736895

a. cut 命令

cut 命令的基本意图是从标准输入或一个或多个文件中抽取一个或多个数据列。

i. 指定字符位置

当数据在固定列中排列时，可以通过字符指定列的位置。表明文件带有固定列格式，应使用-c，后跟一个或多个列规范。（下例中分隔符使用空格）

```
# more censusFixed
Chicago         IL      2783786 3005072 1434029
Houston         TX      1630553 1595138 1049300
Los Angeless    CA       3485398 2968528 1791011
New York        NY      7322564 7071639 3314000
# cut -c 1-14,25-31 censusFixed
```

```
Chicago      2783786
Houston      1630553
Los Angeless  3485398
New York     7322564
```

ii. 字段说明

字段用一个称为分隔符的终止符相互分隔。任何字符都可以是分隔符。要指定一个字段，使用 field 选项（-f），字段的编号的行开始算起，第一个字段为字段编号 1。Cut 命令假定分割符是一个制表符，如果不是，必须在分割符选项（-d）里指定。Cut 选项中抑制选项（-s），该选项告诉 cut 不显示没有分隔符的行。

```
# more censusFixed2
Chicago        IL      2783786 3005072 1434029
Houston        TX      1630553 1595138 1049300
Los Angeless    CA      3485398 2968528 1791011
New York       NY      7322564 7071639 3314000
# cut -f 1,4 censusFixed2
Chicago 2783786
Houston 1630553
Los Angeless    2968528
New York       7071639
```

Cut 命令选项

选项	代码	结果
字符	-c	抽取列编号指定的固定列
字段	-f	抽取被分隔的列
分隔符	-d	指定分隔符（如不是默认的制表符）
抑制	-s	如果行内没有分隔符，抑制输出

b. Paste 命令

Paste 命令将行结合在一起。他从两个或多个文件中输入数据。要指定输入来自标准输入流，可使用连字符（-）来代替文件名。paste 把每个文件的每一行看作一列，如果多个文件，则将所有行通过制表符加以分割的各个文件的相应行写入输出流。Cat 和 paste 命令类似：cat 命令垂直结合文件，paste 命令水平结合文件。Paste 只有一个-d 选项，作用是指定文件间的分隔符。可以指定用于分隔数据的一个或多个分隔符。如果是三个文件可以指定两个分割符。第一个分隔符用在第一个和第 2 个文件之间，而第 2 个分割符用在第 2 个和第 3 个文件之间。

5. 排序

处理大量数据时，需要将其组织成具有可分析性和有效的形式进行处理。一个最简单，也是最强大的组织技术就是排序。

a. sort 命令

sort 命令使用选项、字段说明符和输入文件。字段说明符告诉 sort 使用那些个字段进行排序。Sort 命令格式如下：

```
sort option field specifiers input files
```

选项	代码	结果

检查顺序	-c	检查数据排序的正确性
替换分隔符	-t	指定替代分隔符，默认为制表符和空格
数字式排序	-n	数据是数字，而不是字符串
合并	-m	输入文件被排序。合并结果
唯一性排序	-u	删除重复性的排序字段，只保留最后一个
逆向排序	-r	按降序排序数据
忽略前导空格	-b	把前导空格看作分隔字段的一个空格
字典排序	-d	首先排序特殊字符
重叠大小写	-f	不区分大小写

b. 按行排序

比较两个字符时，实际上就是比较它们相对应的 ASCII 值。

c. 按字段排序

i. 字段

Sort 将字段定义为由单个空格或制表符分隔的字符集。Sort 默认的分隔符是制表符和空格。

ii. 字段说明符

当需要字段排序时，需要定义用来排序的某个或某些字段。字段说明符是一个两个数字的集合，他们共同表示排序关键字中的第 1 个和最后一个字段。格式如下;

+number1 -number2

Number1 指定相对于排序字段开始位置的字段编号，而 number2 指定相对于行首位置到达关键字结尾位置的字段编号。

d. 选项

i. 检查排序

检查排序选项（-c）检查文件是否被排序。如果未排序，显示未排序文件的第一行。

ii. 分隔符

分隔符选项（-t）指定一个可替代的分隔符。像 cut 中的分隔符规范一样，他也应该被放置在引号中。

iii. 数字式排序字段

数字式排序选项（-n），指示 sort 命令按照正确的数字比较排序。

iv. 合并文件

合并文件选项（-m）把多个已排序文件结合到一个排序文件中。

v. 唯一排序字段

唯一选项（-u）当排序字段相同时，则只保留一行，并删除其他重复行。

vi. 逆向（降）排序

要从最大到最小对数据进行排序，可以指定逆向排序（-r）。

vii. 忽略前向导空格

空格选型（-b）对固定格式字段具有强制性。

viii. 字典排序

由于特殊字符混杂在 ASCII 中，所以进行排序后的结果不会像预期一样。所以使用字典（-d）选项。

ix. 重叠小写

要使排序忽视大小写之间的差别，应该使用重叠选项（-f）。在重叠选项中，大写字符被合并到小写字符里。

e. 多途径排序

6. 转换字符

a. tr 命令

tr 命令把用户指定的字符集中的各个字符替代为第 2 个指定字符集中的相应字符。第一个集合中的第一个字符用第二个集合中的第一个字符替换，第一个集合中的第 2 个字符用第二个集合中的第 2 个字符替换，以此类推。直到所有的匹配字符度被替代为止。字符串使用引号指定。只能用键盘指定，转换不接受来自文件的输入。

b. 简单转换

转换从键盘接受输入，然后把输入写入到标准输出。

```
# tr "aeiou" "AEIOU"
It is very easy to use TRANSLATE.
It Is vEry EAsy tO UsE TRANSLATE.
```

c. 不匹配转换字符串

当转换的字符串长度不同时，结果取决于更短的字符串。

```
#  tr "aeiou" "AE?"
It is very easy to use TRANSLAN
It ?s vEry EAsy to usE TRANSLAN
```

d. 删除字符

要删除转换中的匹配字符，应使用删除选项（-d）。

```
# tr -d "aeiouAEIOU"
It is very easy to use TRANSLATE
t s vry sy t s TRNSLT
```

e. 压缩输出

压缩选项（-s）删除输出同一字符的连续出现部分。

```
# tr -s "ie" "dd"
the fiend did dastardly deeds
thd fdnd d dastardly ds
```

f. 求余

求余选项（-c）倒转第 1 个字符的含义。他不指定要改动的字符，而是指定不改动的字符。

7. 带有重复行的文件

a. uniq 命令

uniq 命令删除重复黄，只保留第一行，其他的则删除。这些行只有在相邻的情况下才能被删除，不相邻的重复行不会被删除。要删除不相邻的重复行，必须排序文件。

除非特别指定，整行都将用来比较。Uniq 命令还提供用于比较特定字段或字符的选项。无论比较行，字段或者是字符，都只能比较到行尾的内容。比较在行中间的一个域是不可能的。

Uniq 命令共有三个选项：输出格式、忽略前导字段和忽略前导字符。

b. 输出格式

有四种输出格式：非重复行和每个重复行的第一行（默认）、只有为一行（-u）、只有重复行（-d）以及显示重复行的重复数（-c）。

i. 默认输出格式

使用不带任何选项的 unique 命令，写出所有非重复行和重复行集的第 1 行。

```
# more uniqFile
```

```
5 completely duplicate lines
5 completely duplicate lines
5 completely duplicate lines
5 completely duplicate lines
5 completely duplicate lines
Not a duplicate--next duplicate first 5
5 completely duplicate lines
Last 3 fields duplicate:one two three
Last 3 fields duplicate:one two three
Last 3 fields duplicate:one two three
The next 3 lines are duplicate after char 5
abcde Duplicate to end
fghij Duplicate to end
klmno Duplicate to end
# uniq uniqFile
5 completely duplicate lines
Not a duplicate--next duplicate first 5
5 completely duplicate lines
Last 3 fields duplicate:one two three
The next 3 lines are duplicate after char 5
abcde Duplicate to end
fghij Duplicate to end
klmno Duplicate to end
```

ii. 非重复行（-u）

非重复行选项-U。它抑制重复行的输出，只列出文件中的为一行（不重复行）

```
# uniq -u uniqFile
Not a duplicate--next duplicate first 5
5 completely duplicate lines
The next 3 lines are duplicate after char 5
abcde Duplicate to end
fghij Duplicate to end
klmno Duplicate to end
```

iii. 只有重复行（-d）

与非重复行相反的是只写出重复行，该选项是-d。

```
# uniq -d uniqFile
5 completely duplicate lines
Last 3 fields duplicate:one two three
```

iv. 计数重复行（-c）

计数重复行选项（-c）写出所有行，抑制重复部分，并在行开始处给出重复行数目的计数。

```
# uniq -c uniqFile
   5 5 completely duplicate lines
   1 Not a duplicate--next duplicate first 5
   1 5 completely duplicate lines
```

```
3 Last 3 fields duplicate:one two three
1 The next 3 lines are duplicate after char 5
1 abcde Duplicate to end
1 fghij Duplicate to end
1 klmno Duplicate to end
```

c. 忽略前导字段

默认情况是比较整行来判断两行是否相同，还可以指定比较的开始位置。忽略重复字段选项（-f）跳过指定字段数目（从行首位置开始计算）以及他们之间的空格。记住：字段被定义为一个有空格或制表符分隔的 ASCII 字符集。两个连续的空格是两个字段。

```
# uniq  -d -f4 uniqFile
5 completely duplicate lines
Last 3 fields duplicate:one two three
abcde Duplicate to end
```

d. 忽略前导字符

也可以指定在开始比较前跳过的字符数，使用选项-s。

```
# uniq -d -s5 uniqFile
5 completely duplicate lines
Last 3 fields duplicate:one two three
abcde Duplicate to end
```

8. 计数字符、单词或行

有时需要知道一个文件中包含单词数或行数或字符数。

wc 命令

wc 命令计数一个或多个文件中的字符、单词和行数。字符计数中包含换行符（\n）。wc 的选项可用来限制输出项。

单词计数选项

选项	代码	结果
字符计数	-c	计数每个文件中的字符数
行数	-l	计数每个文件中的行数
单词计数	-w	计数每个文件中的单词数

9. 比较文件

有 3 个 UNIX 命令可用于比较两个文件的内容：比较字节（cmp）、比较行（diff）和查找相同行（comm）。

a. 比较字节（cmp）命令

Cmp 命令按字节检验两个文件。其所采取的行为依赖于所使用的选项码。

i. 不带选项的 cmp

当 cmp 执行不带任何选项时，它将停止在第 1 个不同的字节上，报告出这个字节所处的位置

```
# cat >cmpFile1
123456
7890
# cat >cmpFile2
123456
sfeewfg
```

```
# cmp cmpFile1 cmpFile2
cmpFile1 cmpFile2 differ: char 8, line 2
```

注意：当文件一样时，不显示信息。但当文件有差别时，第一个有差别的位置就被显示出来。

ii. 带列表选项（-l）的 cmp

列表选项按照字节显示文件中发现的所有差别。

```
# cmp -l cmpFile1 cmpFile2
   8  67 163
   9  70 146
  10  71 145
  11  60 145
  12  12 167
```

第一列是指不同字符的位置，如第一行 8 代表第 8 个字符不相同，注意，换行符也是一个字符。第二列是第一个文件在第 8 个字符位置的字符值，第三列是第二个文件在第 8 个字符位置的字符的八进制值。

iii. 带抑制列表选项(-s)的 cmp

抑制列表选项（-s）类似于默认情况，不过他不显示输出，在编写脚本是通常会用到当不显示输出时，结果可以通过测试推出状态来判断。如果退出状态为 0，则两个文件相同；如果是 1，则至少有一个字节有差别。

```
# cmp -s cmpFile1 cmpFile2
# echo $?
1
```

b. 比较行（diff）命令

diff 命令显示两个文件之间行对行的差别。对第 1 个文件和第 2 个文件做比较，差别被标出，以便修改第一个文件可以使之匹配第 2 个文件。

Diff 命令用于处理文件。其参数为两个文件，一个文件和一个目录或两个目录。当指定一个文件和一个目录时，他在指定的目录下查找具有相同名字的文件。如果给出两个目录，则使用每个目录下名字相匹配的所有文件。

c. 查找相同行（comm）命令

comm 命令找出两个文件中的相同行，按行比较两个文件，并显示一个 3 列的结果。左边一列包含文件 1 中的独有行；中间列包含文件 2 的独有行；右边一列包含两个文件的相同行。

第七节

通信

一、 用户通信

a. talk 命令

talk 命令允许两个 UNIX 用户彼此聊天。格式如下：

talk options user id terminal

被呼叫用户有两种响应方式：第一种同意聊天，第二种是拒绝聊天，使用 mesg n 命令。

b. write 命令

write 命令用来给另一个终端发送消息。像 talk 一样，它需要接收方登录。Write 和 talk 之间的区别在于 write 是单向传输。

write options user id terminal

二、 电子邮件

1. 邮件地址
2. 邮件模式

   当进入邮件系统时，或者处于发送模式，或者处于读模式。处于发送模式时，可以切换到读模式处理收到的邮件；相反，处于读模式也可以切换到发送模式答复收到的邮件。
3. mail 命令

   mail 命令用于阅读和发送邮件。包含一个用于编写邮件内容的文本编辑器。
4. 发送邮件
5. 读模式

三、 远程访问

telnet 概念

a. 分时环境

b. 登录

连接到远程主机

在命令行下输入 telnet 命令，一旦给命令被输入，我们就位于 telnet 提示符所指示的 telnet 系统下。

telnet 接口子命令

命令	含义
open	连接到一个远程计算机
close	关闭连接
display	显示操作参数
mode	改变到行模式或字符模式
set	设置操作参数
status	显示状态信息
send	发送特定字符
quit	退出 telnet
?	帮助命令

四、 文件传输

1. ftp 命令

文件传输协议（ftp）是从一个计算机向另一个计算机复制文件的 TCP/IP 标准。ftp 协议与其他客户端-服务器应用不同，因为它在主机间建立两个连接。一个连接用于数据传输，另一个用于控制信息。命令和数据传输分离使 ftp 效率更高。

客户端有 3 个元素：用户接口、客户端控制过程和客户端数据传输过程。服务器有两个元素：服务器控制过程和服务器数据传输过程。

2. 建立 ftp 连接
3. 关闭 ftp 连接
4. 传输文件

文件可以从本地系统传输到远程系统，也可以从远程系统传输到本地系统。

传输文件有两个命令：get 和 put。他们都在本地系统上进行。

完整的 ftp 命令列表

!	debug	mdir	pwd	size
$	dir	mget	quit	status
accout	direct	mkdir	quote	struct
append	disconnect	mls	recv	sunique
ascii	form	mode	reget	system
bell	get	modtime	rename	tenex
binary	glob	mput	reset	trace
bye	hash	newer	restart	type
case	help	nlist	rhelp	umask
cd	idle	nmap	rmdir	user
cdup	image	ntrans	rststus	verbose
chmod	lcd	open	runique	win
close	ls	prompt	send	?
cr	macdef	proxy	sendport	
delete	mdelete	put	site	

第八节

vi 和 ex

一、 Vi 编辑器

1. 命令

命令是 vi 中基本编辑实用程序。

作为一个通用规则，命令是区分大小写的，不需要 Return 的热键。

2. 命令种类

Vi 中命令被划分为三个单独类别：局部命令、范围命令和全局命令。

二、 Vi 中的局部命令

局部命令是应用于相对于光标当前位置的文本的命令。

插入文本命令（i、I）

i 插入字符在当前位置

I 插入字符在当前行行首

附加文本命令（a、A）

a 在当前字符后

A 在当前行尾

换行命令（o、O）

o 在当前行下一个新行增加文本

O 在当前行上一行中增加文本

替换文本命令（r、R）

r 用另一个字符替换当前单个字符

R 可以一次替换许多字符

替代文本命令（s、S）

s 用一个或多个字符替换一个字符

S 用心文本替换整个当前行

删除字符命令（x、X）

x 删除当前字符

X 删除光标前字符

标记文本命令（m）

主要用来标记后面的范围命令所用的位置。标记命令（m）需要一个成为被标记位置的名字的字母。

更改大小写命令（~）

更改大小写命令（~）更改当前字符的大小写。一次更改一个字符。

放置命令（p、P）

在文字处理程序中，放置命令被称为“粘贴”。小写 p 复制光标位置后暂时缓冲区中的内容。大写 P 复制光标前的缓冲区

结合命令（J）

可使两行结合到一起。

三、 Vi 中的范围命令

1. 文本对象

一个文本对象就是两个位置之间的一段文本：光标和目标。该对象可从光标位置处扩展到文档开始处或文档的结尾处。目标是标识被定义的对象结尾的标记。七个常用的对象类：字符、单词、句子、行、段落、块和文件。

a. 字符对象

字符对象只由一个字符组成。这就意味着对象的开始和结尾是一样的。要立即定位光标前一个字符，使用指示符 h。要定位当前字符，使用指示符 1。

b. 单词对象

单词对象定义为终止空格字符的一系列非空格字符。一个单词对象可为整个单词或一个单词的一部分，这依赖于当前字符的位置和对象指示符。有 3 种单词指示符：指示符 b 意味着返回到当前单词的开头；指示符 w 意味着前移到下一个单词的开头；指示符 e 意味着前移到当前单词的结尾。

c. 行对象

行是开始于换行符后第一个字，一直到下一换行符的所有文本，一个行对象可由一行或多行组成。有 6 种行指示符：要定位当前行的开始，使用 0（零）；要定位行的结尾，使用$；要定位当前行上一行的开始，使用-（减号）；要定位下一行的开始，使用+（加号）；要定位当前字符上的字符，使用 k

d. 句子对象

一个句子是以句点或问号结束后跟两个空格或换行符的文本范围。定位句子使用括号，左边的括号可在句子的开始处，右边的括号在句子的结尾处。

e. 段落对象

使用大括弧标识（{}）。

f. 块对象

块是由标记符标识的文本范围。

g. 屏幕对象

我们可以把部分文本或整个屏幕定义为一个屏幕对象。

H：移动光标到屏幕最上面文本行的开始

L：移动光标到屏幕最下面文本行开始处

M：移动光标到屏幕中间文本行开始处

h. 文件对象

要移动到文件对象中最后一行的开始，或使用它创建一个范围，使用指示符 G

2. 文本对象命令

范围命令总结

命令	描述
空格	移动光标
d	删除一个文本对象
c	用新文本替换一个文本对象
y	剪切一个文本对象

四、 Vi 中的全局命令

1. 滚动命令

命令	描述
ctrl+y	上滚一行
ctrl+e	下滚一行

ctrl+u	上滚半屏
ctrl+d	下滚半屏
ctrl+b	上滚整屏
ctrl+f	下滚整屏

2. 撤销命令

命令	描述
u	只撤销租后一次编辑
U	撤销当前行的所有修改

3. 重复命令(.)

4. 屏幕再生命令

命令	描述
z return	刷新屏幕，并在屏幕顶端放置当前行
z .	刷新屏幕，并在屏幕中间放置当前行
z -	刷新屏幕，并在屏幕底部放置当前行
ctrl+L	刷新屏幕，不移动光标

5. 显示文档状态行

6. 保存和退出命令

五、 在 vi 中重新组织文本

1. 移动文本

a. 使用适当的删除命令删除文本。

b. 将鼠标移动到文本被放置的位置

c. 使用适当的放置命令（p 或 P）将缓冲区内的文本复制到文件缓冲区

2. 复制文本

a. 使用粘贴块，y，粘贴文本

b. 移动光标到复制位置

c. 使用 p 或 P 放置文本

3. 命名的缓冲区

Vi 编辑器使用一个暂时缓冲区和 35 个命名的缓冲区。所有缓冲区内的文本都可以使用放置命令检索。但在暂时缓冲区内的数据在使用一个非位置命令时会丢失。这意味着删除后跟一个插入操作会清空暂时缓冲区。

命名的缓冲区在其内容被替换前均可用。前 9 个命名缓冲区成为数字缓冲区，因为他们由数字 1 到 9 标识。其余 26 个命名缓冲区成为字母缓冲区；他们由小写字母 a 到 z 标识。

要检索暂时缓冲区内的数据，使用基本的放置命令。要检索命名缓冲区内的文本，在放置命令前需要加双引号以及如下列命令语法所示的缓冲区名。注意，在双引号、缓冲区名和放置命令之间没有空格。

“buffer-namep

使用该语法，下面三个例子检索暂时缓冲区、缓冲区 5 和缓冲区 k 内的文本。

p “5p “kp

a.数字命名缓冲区

当句子、行、段落、屏幕或文件被删除时自动使用数字缓冲区。被删除的文本自动复制到第一个缓冲区（1），并在以后可以加以引用。前面一次删除中被删除的文本可使用放置命令检索或通过检索数字缓冲区 1 加以检索。最后的 9 次删除都是在数字缓冲区内检索到的。

b.字母命名缓冲区

字母命名缓冲区可存储 26 个文本实体。数字缓冲区只能保存至少一个句子长的文本对象，而在一个字母缓冲区内可以保存任意大小的文本对象。任何删除和剪切命令都可以被用来复制数据到字母缓冲区。

要使用字母缓冲区，必须在删除或剪切命令中指定缓冲区名称。如下所示：

“ad　　#删除并且保存在 a 缓存区内

六、 ex 编译器

本部分略

第九节

正则表达式

一、 原子

原子指定要匹配的文本的内容以及发现它的位置。正则表达式中的原子包含如下 5 种类型：单个字符、点、类、锚和向后引用。

1. 单个字符

最简单的原子是单个字符。单个字符出现在正则表达式中时，它与自己进行匹配。

2. 点

一个点匹配除了换行符（\n）外的任意单个字符。这种全体匹配的功能使其成为正则表达式操作中一种强大的元素。

3. 类

类原子定义了一个 ASCII 字符集。其中任意一个都要匹配文本中的任意字符。匹配过程中使用的给字符集要用方括号（[]）括起来。类集合是一种强大的表达方式元素。它的强大使其可扩展额外的表示：范围、排除和转义字符。文本字符范围由破折号（-）指出。如指出字符范围 a 到 d 的表达式[a-d]。有时很容易的指出哪个字符被排除在集合外——也就是指定其余集。可使用 UNIX 的非运算符（^）。例如，要指定任意非元音字符，可使用[^aeiou]。在类原子括号集内的^符号表示余集，被解释为除了和集中标出字符外的任意 ASCII 字符。第 3 个扩展出的标记是转义字符（\）。在匹配字符为其他另两种标记之一时使用它。如使用转义字符指出破折号是一个字符，不是一个范围标记时，编码为[aeiou\-]。

4. 锚

锚是在将模式与字符串的特定部分进行行对齐时使用的原子。换句话说，锚不是匹配文本，而是定义模式中下一个字符在文本中必须出现的位置。锚有四种类型：行首（^）、行尾（$）、单词开头（\<）和单词结尾（\>）。

锚原子经常在界河中使用，例如，要定位以字母 Q 开头的一个字符串，表达式编码为^Q,类似，要找出以 g 结尾的单词，表达式编码为 g\>。

5. 向后引用

将文本暂时保存在 9 个存储缓冲区内的其中一个，使用一个向后引用引用保存在缓存区内的文本。使用转义符和范围 1 到 9 的数字对向后引用进行编码，如下所示：

\1\2……\9

向后引用用于将当前或目标缓冲区内的文本和保存在系统的 9 个缓冲区其中之一中的为本进行匹配。

二、 运算符

为增强正则表达式的功能，可将原子和运算符结合起来。正则表达式运算符扮演的角色和输血运算符是一样的。可以讲正则表达式分为 5 种不同的类型：序列运算符（空）、替换运算符（|）、重复运算符（{\m,n\}）、组运算符（（…））和保存运算符（\(…)\）。

1. 序列

序列运算符为空。他的含义是：如果原子系列，如字符序列，出现在正则表达式中，则暗示在原子间有一个不可见的序列运算符。

2. 替换

替换运算符（|）用于定义一个或多个替代物。例如，如果要选择 A 或 B，则编码正则表达式为 A|B。

3. 重复

重复运算符是转义括号（\{m,n}\）的集合，它包含由逗号分割的两个数字的集合。他指定重复运算符前的原子或表达式的重复次数。由第 1 个数字（m）指出前面的原子在文本中出现的最少次数；第 2 个数字(n)指出它出现的最大次数。

a. 基本重复形式

m 和 n 的取值是可选的，但必须至少指定一个。

b. 缩写形式运算符

UNIX 为三种常用的重复形式提供了特殊的缩写运算符。星号（*）可用来重复一个原子 0 或多次（等同于\{0,\}）；加号（+）用于指定原子必须出现 1 次或多次（等同于\{1,\}）；问号（？）用于只重复一个模式 0 或 1 次（等同于（\{0,1\}

c. 贪婪模式匹配

贪婪算法设计目的为最大化其操作。当使用一个包含重复运算符的正则表达式时，重复部分会试图使结果尽可能多的匹配文本。

4. 组运算符

组运算符是一对圆括号。当一组字符被括在圆括号内，则下一次运算符应用于整个组，而不止是前面的字符。

5. 保存

保存运算符是转义圆括弧集合\(…\)，他把匹配的文本字符串复制到 9 个缓冲区的其中一个，以便后面引用。

第十节

grep

grep 表示全局正则表达式打印（global regular expression print）。用来对输入文件中匹配指定正则表达式的所有行进行搜索，并将之写入标准输出文件。

一、 操作

要编写操作正确的脚本，必须要理解 grep 实用程序的工作方式。对标准输入的每一行，grep 执行如下操作：

a． 把下一输入行复制到模式空间中。模式空间是指可保存一个文本行的缓冲区。

b． 对模式空间应用正则表达式

c． 如果有匹配存在，该行从模式空间被复制到标准输出。

Grep 实用程序对输入的每一行重复这三个步骤

1. grep 流程图
2. grep 操作示例

- grep 是一个搜索实用程序：它只能搜索匹配一个正则表达式的一行的存在性。
- grep 可以对一行采取的唯一动作是把它发送到标准输出。如该行不匹配，则不打印。
- 行的选择只基于正则表达式。行编号或其他准则不能用于选择行。
- grep 是一个过滤器，它可被用在一个管道的左边或右边。

除了上面列出的几点，下面的 grep 限制也必须时刻牢记：

- grep 不能用于增加、删除和修改行
- grep 不能用于只打印行的一部分
- grep 不能只读文件的一部分
- grep 不能基于前面的内容或下一行来选择一行。只有一个缓冲区，只保留当前行。

二、 grep 家族

在 grep 家族中有三个实用程序：grep、egrep 和 fgrep。

grep：只支持数量有限的正则表达式。

fgrep：支持字符串模式，不支持正则表达式

egrep：支持大多数正则表达式，但不是全部

1. grep 家族选项

选项	解释
-b	在每行前加上所在文件块编号
-c	只打印匹配模式的行编号计数
-i	在匹配文本时忽略大小写
-l	打印至少有一行匹配模式的文件列表
-n	在每行前显示其行号
-s	哑模式。执行其功能，但抑制所有的输出
-v	逆向输出。打印不匹配模式的行
-x	只打印完全匹配模式的行
-f file	要匹配的字符串列表在文件 file 中

2. grep 家族表达式

原子	grep	fgrep	egrep	操作符	grep	fgrep	egrep
字符	√	√	√	序列	√	√	√
点	√		√	重复	除了？都可以		*？+
类	√		√	替换			√
锚	√		^$	组			√
先后引用	√			保存	√		

grep 实用程序中的表达式往往很复杂，经常把几个原子和/或运算符结合到一个大的表达式中。当运算符和原子被结合时，它们通常被括在单引号或双引号内。从技术角度讲，引号只有在使用空格或其他对 grep 有特殊意义的字符时才是必须的。

3. grep

文件匹配的最初实用程序 grep 可处理大部分正则表达式。再不需要对表达式分组或使用重复来匹配的情况下影使用它。它是 grep 家族唯一允许保存匹配结果以待后用的成员。

4. 快速 grep

如果搜索规则只需要序列表达式，则快速 grep 是最好的选择。

5. 扩展 grep

扩展 grep 是三种 grep 实用程序中功能最强大的。虽然没有保存选项，但它允许更复杂的模式。

第十一节

sed

sed 是流编辑器（stream editor）头字母缩写词。Sed 按行对输入文件进行浏览，对输入文件的每行执行指令列表（称 sed 脚本）。

Sed 实用程序有三个可用选项。选项-n 禁止自动输出，它允许编写可以控制打印的脚本。选项-f，指出有一个脚本文件，她紧跟在命令行中该选项后。选项-e 是默认的，表明脚本在命令行上，而不是在一个文件中。

一、 脚本

1. 脚本格式

```
$sed –e ‘address command’ input_file
```

或

```
$sed –f scriot.sed input_file
```

2. 指令格式

每个指令都由一个地址和一个命令组成的。

地址选择命令要处理（或不处理）的行。惊叹号（！）是可选地址取余符号。当其未出现时，地址必须精确匹配一行来选择该行；当给出取余运算符时，将选择任何与地址不匹配的行，匹配地址的行被忽略。命令指出 sed 对匹配地址的每个输入行应用的动作。

3. 注释

注释是一个脚本行，用来归档或解释脚本中的一个或多个指令。

二、 操作

Sed 为输入文件的每行提供一个行编号，此编号可用来定位文本中的行。对于每行，sed 执行下列操作：

- 将一个输入行复制到模式空间中。模式空间是一个特定的缓冲区，可以保存用于处理的一个或多个文本行。
- 从上到下，将脚本中所有指令，依次应用到匹配指令中指定地址的所有模式空间行。
- 如果未使用-n 选项标志禁止自动输出，则将模式空间内容复制到输出文件中。

注意：sed 不改变输入文件。所有被改动的输出都被写入标准输出中，如果要保存的话，则必须重定向到一个文件。

三、 地址

指令中的地址用来判断输入文件中的哪一行要被指令 zhognde 命令所处理。Sed 中的地址可为下面四种类型：单行、行集合、行范围、嵌套地址。

1. 单行地址

一个单行地址制定输入文件中的一行（且只指定一行）。有两种单行地址格式：行编号或美元符号，后者制定输入文件的最后一行。

2. 行集合地址

行集合地址是一个正则表达式，他输入文件中一行或不一定连续的多行。该正则表达式下在两个斜线间。输入文件中匹配正则表达式的任意行被指令命令处理。需要注意的两点：首先，正则表达式匹配的行可连续也可以不连续。其次，即使某行匹配，指令也不一定对该行起作用。

3. 范围地址

范围地址定义一个连续行集合。其格式是：开始地址+中间无空格的逗号+结尾地址：

```
Start-address,end-address
```

开始地址和结尾地址可为 sed 行编号或正则表达式。

当模式空间中的一行匹配开始范围时，他被选择进行处理。Sed 会注明指令是在一个范围内，每个输入行都要被指令的命令所处理，直到结尾地址匹配的那一行。

4. 嵌套地址

嵌套地址是包含在另一地址中的地址。根据定义，外部地址范围必须是行集合地址或范围地址；嵌套地址可为单行地址、行集合地址或另一范围地址。

四、 命令

指令中可用的命令共有 25 个。

1. 行编号命令

行编行命令（=）把该行写入输出时，在不影响模式空间的情况分离下，于行首位置写入当前行编号。它类似于 grep –n 选项，唯一的差别是行编号被写在一个分离行上。

```
# sed '=' TheRavenV1
1
Once upon a midnight dreary,while I pondered,weak and weary,
2
Over many a quaint and curious volume of forgotten lore
3
While I nodded,nearly napping,suddenly there came a tapping,
4
As of someone gently rapping,rapping at my chamber door.
5
"This some visitor,"I muttered."tapping at my chamber door"
6
Only this and nothing more."
```

为达到只打印以大写字母”O”开头的行的编号。使用-n 选项，关闭自动打印功能。使用正则表达式/^O/来选择只以“O”开始的行。

```
# sed -n '/^O/=' TheRavenV1
1
2
6
```

2. 修改命令

修改命令用于插入、附加、修改或删除一行或多行。修改命令要求将与其相关的任意文本放在脚本的下一行上。因此，脚本必须是个文件；不能直接在 shell 命令行中编码。

另外，修改命令对整行进行操作，换句话说，他们是行替换命令。所有命令都应用于整行，不能只修改一行的一部分。

a. 插入命令（i）

插入命令在给定地址前把一行或多行直接添加到输出。该命令只能与单行和行集合地址一起使用，不能与范围地址一起使用。

```
# cat insertTitle.sed
#Script Name:insertTitle.sed
#Add a title to file
1i\            #数字1，和插入命令i
    The Raven\
    by\
    Edgar Allan Poe\
```

```
# sed -f insertTitle.sed TheRavenV1
    The Raven
    by
    Edgar Allan Poe

Once upon a midnight dreary,while I pondered,weak and weary,
Over many a quaint and curious volume of forgotten lore
While I nodded,nearly napping,suddenly there came a tapping,
As of someone gently rapping,rapping at my chamber door.
"This some visitor,"I muttered."tapping at my chamber door"
Only this and nothing more."
#
```

b. 附加命令（a）

类似于插入命令，不同之处是它直接把文本在指定行之后写入到输出。像插入一样，附加命令不能和范围地址共同使用。

注意:被插入和附加的文本不能出现在 sed 的模式空间中,他们既不能匹配一个正则表达式,也不能影响 sed 的内部行计数。

```
# cat appendLineSep.sed
a\
------------------------------
$a\
\
      The End
# sed -f appendLineSep.sed TheRavenV1
Once upon a midnight dreary,while I pondered,weak and weary,
------------------------------
Over many a quaint and curious volume of forgotten lore
------------------------------
While I nodded,nearly napping,suddenly there came a tapping,
------------------------------
As of someone gently rapping,rapping at my chamber door.
------------------------------
"This some visitor,"I muttered."tapping at my chamber door"
------------------------------
Only this and nothing more."
------------------------------

      The End
```

c. 更改命令（c）

更改命令用新文本替换匹配行。与插入和附加不同，他接受所有四种地址类型。

```
# more TheRavenV1
Once upon a midnight dreary,while I pondered,weak and weary,
Over many a quaint and curious volume of forgotten lore
```

```
While I nodded,nearly napping,suddenly there came a tapping,
As of someone gently rapping,rapping at my chamber door.
"This some visitor,"I muttered."tapping at my chamber door"
Only this and nothing more."
# more change.sed
#Script Name:change.sed
#Replace second line of The Raven
2c\
Over many an obscure and meaningless problem of calculus bore
# sed -f change.sed TheRavenV1
Once upon a midnight dreary,while I pondered,weak and weary,
Over many an obscure and meaningless problem of calculus bore
While I nodded,nearly napping,suddenly there came a tapping,
As of someone gently rapping,rapping at my chamber door.
"This some visitor,"I muttered."tapping at my chamber door"
Only this and nothing more."
```

d. 删除模式空间命令（d）

使用小写删除命令（d），删除整个模式空间。位于删除命令后的任何针对已被删除文本的脚本命令将被忽略，因为这些文本在模式空间中已经不存在。

```
# more TheRavenV1
Once upon a midnight dreary,while I pondered,weak and weary,
Over many a quaint and curious volume of forgotten lore
While I nodded,nearly napping,suddenly there came a tapping,
As of someone gently rapping,rapping at my chamber door.
"This some visitor,"I muttered."tapping at my chamber door"
Only this and nothing more."
# sed '/^O/d' TheRavenV1
While I nodded,nearly napping,suddenly there came a tapping,
As of someone gently rapping,rapping at my chamber door.
"This some visitor,"I muttered."tapping at my chamber door"
```

e. 只删除首行的命令（D）

当使用大写删除命令（D）时，只有模式空间的第一行被删除。

3. 替代命令（s）

模式替代是 sed 中功能最强大的命令之一。通常，替代命令会把正则表达式选择的文本置换成一个替换字符串。因此，它类似于文本编辑器中的搜索和替换操作。替代命令的格式：

地址 s / 模式 / 替换字符串 / 标记

a. 搜索模式

Sed 搜索模式只使用正则表达式原子核模式的一个子集。

原子	允许	操作符	允许
字符	√	序列	√
点	√	重复	* ? \{…\}
类	√	替换	√
锚	^ $	组	

向后引用	√	保存	√

当一个文本行被选中时，其文本与模式进行匹配，如果找到匹配文本，则其被替换字符串所替换。

b. 模式匹配地址

在进一步了解替代操作前，先看一个特殊情况：地址包含的正则表达式于我们要匹配的模式一样。这种情况下，不需要再替代命令里重复该正则表达式，而是需要通过在模式前加两个斜线来忽略它。

```
# more browning.txt
How do I love thee?Let me count the ways.
I love thee to the depth and breadth and height
My soul can reach,when feeling out of sight
For the ends of being and ideal grace.
I love thee to the level of everyday's
Most quiet need,by sun and candle-light.
# sed '/love/s//adore/' browning.txt
How do I adore thee?Let me count the ways.
I adore thee to the depth and breadth and height
My soul can reach,when feeling out of sight
For the ends of being and ideal grace.
I adore thee to the level of everyday's
Most quiet need,by sun and candle-light.
```

在这种情况下的搜索模式和地址模式是一样的，因此不需要再重复它。注意：替换命令后的两个斜线模式被忽略了。

c. 替换字符串

替换文本是一个字符串。在替换字符串中，只可使用一个原子和两个元字符。被允许的替换原子是向后引用，两个元字符标志是（&）和反斜线转义（\）。&用于防治替换字符串中的模式；当需要在替换文本中包含&符号时，反斜线（\）用于对其进行转义（如果不对其进行引用，她会被模式所替换）。下面的例子显示了元字符的使用方式。第 1 个例子中，替换字符串变成***UNIX***。第 2 个例子中，替换字符串变成&forever。

```
$sed ‘s/UNIX/***&***/’ file1
$sed ‘/now/s/now\& forver/’file1
```

d. 替代操作

当模式匹配文本时，sed 首先删除文本，然后插入替换文本。这意味着我们可以使用替代命令添加、删除和替换一行中的某一部分。

- 删除一行的一部分：要删除一行的一部分，使替换文本为空即可。换句话说，删除行的某个部分是替换物空的特殊替换情况。例如删除标准输入中所有的数字。

```
# sed 's/[0-9]//g'
123abc456
321cba654
abc
cba
```

由于缓冲空间只能存一段文本，所以当输入完成第 2 行后，第 1 行的输出结果就自动出现在第 2 行的下面。

Sed 命令只对一行中一个模式的第一个实例进行操作。上例中要删除所有的数字，因此，可在模式结尾使用全局标记（g），如果不使用它，则只删除每行的第一个数字。

- 修改一行的一部分：要修改一行的某一部分，需要创建一个匹配被修改部分的模式，然后把新文本放在替换表达式中。下例中将文件中所有的空格修改为制表符。

- 向一行的一部分添加内容：要向一行加入文本，需要定位文本的模式以及被加入的文本。因为该模式删除文本，所以必须将其包含在新文本中。下例在每行开头加入两个空格，在结尾加入两个破折号。

```
# more addPart.sed
#!/bin/ksh
# Script Name:addPart.sed
# This script adds two spaces to the beginning and
# --to the end of each line.
s/^/  /
s/$/--/
# sed -f addPart.sed
Now is the time
For all good students
  Now is the time--
  For all good students--
```

e. 向后引用

有时需要还原搜索中删除的数据。

Sed 实用程序在替代替换字符串中使用两种不同的向后引用。完全模式（&）和编号缓冲区（\d）。完全模式把被删除的文本置换到替换字符串中。在编号缓冲区置换中，无论何时一个正则表达式匹配了文本，该文本都被按次序放到 9 个缓冲区的其中一个。编号缓冲区置换（\d）中的 d 是 1 到 9 之间的一个数字，它把编号缓冲区内容置换到替换字符串中。

- 完全模式替代：如果一个模式替换命令匹配模式空间中的文本，则匹配的文本自动被保存在一个缓冲区内（&）。然后便可以检索其内容，并随时随地将其插入到替换字符串中。因此使用&缓冲区允许匹配自动被删除的文本，并还原他。

  下例代码将使用替换命令实现在每行的开头加入两个空格，在每行的结尾加入两个破折号。

```
# more file1
Now is the time
For all good students
To come to the aid
Of their college
# sed 's/^.*$/  &--/' file1
  Now is the time--
  For all good students--
  To come to the aid--
  Of their college--
```

下一个例子，是将饭馆的现有菜单价格列表使用完全模式替代在价格前加上美元符号。

```
# more priceFile
**Bargain Meals**
All you can eat
Breakfast       3.99
Lunch          6.49
Dinner         14.29
# sed 's/[0-9]/$&/' priceFile
**Bargain Meals**
All you can eat
Breakfast       $3.99
Lunch          $6.49
Dinner         $14.29
```

此情况下是用替换字符串中的文本替换掉实用搜索模式进行了匹配并删除的文本。在替换模式中，指定新文本为美元符号加上在搜索中删除的数字。只替换一个数字所用的方式与替换整行的方式是相同的。

- 编号缓冲区替代：编号缓冲区替换使用一个或多个正则表达式编号缓冲区。如果模式只匹配输入文本的一部分，而不是全部。

  一个保险号码有三个部分：三个数字——两个数字——四个数字。要求查找并重新格式化。例子中使用一个搜索模式，该模式使用编号缓冲区来保存三个连续数字，后跟两个数字，在后面是四个数字。一旦发现完全匹配，就用编号缓冲区重新格式化该号码。

```
# more empFile
George Washington      001010001
John Adams            002020002
Thomas Jefferson       003030003
James Madison          123456789
# sed 's/\([0-9]\{3\}\)\([0-9]\{2\}\)\([0-9]\{4\}\)/\1-\2-\3/'
empFile
George Washington      001-01-0001
John Adams            002-02-0002
Thomas Jefferson       003-03-0003
James Madison          123-45-6789
```

4. 替代标志

在替代命令结尾。可加入四种标志，来修改其行为：全局替代（g）、指定次数替代（数字）、打印（p）和写文件（w file-name）。

a. 全局标志

替代命令只替换一个模式的第一次出现。如果模式出现多次，则第 1 个之后的实例不会发生变化。

b. 指定次数标志

指定次数的替代（数字）则修改匹配模式的文本的任何次出现。该数字指出那次出现被修改；要修改一个模式的第 2 次出现，使用 2；

一个命令中只有一个特定的出现可以被修改。

c. 打印标志

有时不想打印所有的输出，可以通过-n 选项关闭自动打印，然后就可以在替代命令中添加打印标志。

如下事例中，要创建一个只包含常规文件名及其权限的列表。因为要限制文件的输出，所以使用 sed 的-n 选项关闭自动输出。替代命令的第 1 个正则表达式保存权限。权限以破折号开头，后跟 0 或多个非空格字符，在第一个空格位置停止，由此可以标识权限。为了跳过该行其余部分直达文件名，创建一个包含 0 或多个字符，并以冒号后跟两个字符（时间域中的第 2 个数字）结尾的模式。然后再保存文件名，以最后时间数字后的空格字符开头。

```
# ls
Desktop          dev              lost+found       system
Documents        devices          mnt              tmp
TT_DB            etc              net              usr
bin              export           opt              var
boot             home             platform         vol
cdrom            kernel           proc
yang-solaris-color
core             lib              sbin             yang_test
# ls -l | sed -n '/^-/s/\(-[^ ]*\).*:..\(.*\)/\1\2/p'
-rw------- core
-rwxrwxrwx yang-solaris-color
```

d. 写文件标志

写文件命令类似打印命令。注意，在命令和文件名之间只能有一个空格。

上例中的命令可以通过更改将输出文件输出到一个文件中。

```
# ls -l | sed -n '/^-/s/\(-[^ ]*\).*:..\(.*\)/\1\2/w'
```

5. 转换命令（y）

当需要将一个字符集转换为另一个字符集时，使用转换命令。转换命令（y）同时需要两个字符集，第一个字符串中的每个字符都要被修改为第二个字符串中相应字符的取值。格式如下：

[地址] [y] [/] [源文件] [/] [替换字符]

例如：

```
# sed 'y/aeiou/AEIOU/'
A good time was had by all
Under the Harvest Moon last September
A gOOd tImE wAs hAd by All
UndEr thE HArvEst MOOn lAst SEptEmbEr
```

6. 输入和输出命令

Sed 实用程序自动从输入文件中读取文本，并把数据写入输出文件，通常为标准输出。有 5 种类型的输入、输出命令：下一行（n）、附加下一行（N）、打印（p）、打印首行（P）和列表（l）。

a. 下一行命令（n）

下一行命令强制 sed 读取输入文件的下一文本行。但在读下一行前，它把当前模式空间的内容复制到输出中，并删除模式空间中的当前文本，然后用下一输入行的文本重新填充他。

b. 附加下一行命令（N）

此命令把下一输入行加入到模式空间的当前内容中。在需要同时对两行或多行应用模式时，很有用。

```
# more appendLines.sed


N
s/\n//
# more appendLines.dat
11122fsdgd
sdfasgd
dsfsdg
w523534
dsaf53asdf
# sed -f appendLines.sed appendLines.dat
11122fsdgdsdfasgd
dsfsdgw523534
```

c. 打印命令（p）

打印命令（p）把模式空间中当前内容复制到标准输出文件中。如果在模式空间中有多行，他们全部被复制。模式空间的内容不会被打印命令删除。

```
# sed 'p' appendLines.dat
11122fsdgd
11122fsdgd
sdfasgd
sdfasgd
dsfsdg
dsfsdg
w523534
w523534
dsaf53asdf
dsaf53asdf
```

d. 打印首行命令（P）

打印命令打印模式空间的整个内容，而打印首行命令（P）只打印第一行。

e. 列表命令（l）

列表命令将非打印字符转换为八进制编码。

7. 文件命令

a. 读取文件命令（r）

读取文件命令（r）读取一个文件，在移向下一个命令之前将其内容放入输出。用于在一个文件一段文本后插入一或多行。文件的内容出现在输出中当前行（模式空间）的后面。

b. 写文件命令（w）

写文件命令（w）把当前模式空加的内容写入一个文件。用于把选择的数据写入文件。

8. 分支命令

分支命令修改脚本文件中命令的正规流向。

a. 分支标签

每个分支命令必须有一个目标，它或者是一个标签或者是脚本中的最后一个指令（一个空格标签）。标签由那些以冒号（：）开始的行组成，后跟标签名（最多由 7 个字符组成）。在脚本标签行，除了冒号和标签名外，可能再没有其他的命令和文本。标签名必须紧跟在冒号后；在冒号和名字之间可以没有空格，名字中不能嵌入空格。标签如下：

```
:comHere
```

b. 分支命令

分支命令（b）的格式类似于一般指令格式，由地址、命令（b）和属性（目标）组成。

目标必须是空格或匹配脚本内一个脚本标签。如果未给出标签，则分支一直到脚本的结尾结束，在该位置，模式空间的当前内容被复制到输出文件中，脚本对下一输入行进行重复处理。

```
# more branch.sed
# Script Name: branch.sed
# This script prints a line

/(1)/b                    #定义分支1
/(2)/b print2             #定义分支2
/(3)/b print3             #定义分支3

#Branch to end of script
b                         #分支1

# print three             #分支3
:print3
p
p
b

#print two
:print2                   #分支2
p
# more branch.dat
(2)print me twice
# sed -f branch.sed branch.dat
(2)print me twice
(2)print me twice
```

c. 替代命令的分支

与无条件分支不同，只有在已经替代后才使用分支，这种情况下使用替代分支，或称测试命令（t）。其格式与基本分支命令相同。

```
# more branchSub.sed
# Script Name:branchSub.sed
# This scripts a line multiple times(up to 3)
#depending on the first characters of the line

s/(1)//
t
```

```
s/(2)//
t print2
s/(3)//
t print3

#Branch to end of script
b

#print three
:print3
p
p
b

#print two
:print2
p
# sed -f branchSub.sed branch.dat
print me twice
print me twice
```

9. 保留空间命令

保留缓冲区用于保存模式空间，有 5 个命令用于在模式空间和保留空间之间来回移动文本：保留并销毁（h）、保留并附加（H）、取得并销毁（g）、取得并附加（G）和交换（x）。

a. 保留并销毁命令

保留并销毁命令（h）把模式空间的当前内容复制到保留空间中，并销毁保留空间里当前的所有文本。

b. 保留并附加

保留并附加命令（H）把模式空间的当前内容附加到保留空间

c. 取得并销毁命令

取得并销毁命令（g）把保留空间的文版复制到模式空间中，并销毁模式空间中当前所有文本。

d. 取得并附加

取得并附加命令（G）把保留空间的当前内容附加到模式空间。

e. 交换命令

交换命令（x）交换模式和保留空间的文本。

演示保留命令的使用，下面编写一个脚本交换一个文件的每两行。换句话说，行 1 和行 2 交换，行 3 和行 4 交换。

```
# more exchange.sed
# Script Name:exchange.sed
# This script exchanges pairs of lines in a file.

#copy pattern to hold
h
```

```
#read next line
n

#retrieve hold area
G
p
# sed -nf exchange.sed exchange.dat
line2
line1
lin34
line3
line6
line5
```

10. 退出

退出命令（q）终止 sed 实用程序。

五、 应用

六、 grep 和 sed

七、 练习（练习为本人自己做的，可能不是正确方案或最优方案）

1. 编写一个 sed 命令，删除一个文件每行中的第 1 个字符

```
# more test1
efgs:asgfe
sssssssss
rsts
abc
# sed "s/^.//g" test1
fgs:asgfe
ssssssss
sts
bc
```

2. 编写一个 sed 命令，删除一个文件每行中的第 2 个字符

```
# sed 's/\(.\).\(.*\)/\1\2/' test1
```

3. 编写一个 sed 命令，删除一个文件每行中的最后一个字符

```
# sed "s/.$//g" test1
```

4. 编写一个 sed 命令，删除一个文件每行中的倒数第 2 个字符

```
sed 's/\(.*\).\(.$\)/\1\2/' test1
```

5. 编写一个 sed 命令，删除一个文件每行中的第 2 个单词

```
# sed 's/\([^a-z]*\) [a-z]*\( [a-z].*\)/\1\2/' test1
$ sed 's/[^[:space:]]*[[:space:]]*//2' test1  (cu的Edengundam提供)
```

6. 编写一个 sed 命令，删除一个文件每行中的倒数第 2 个单词

```
# sed 's/\(.*\) .*\( .*\)/\1\2/' test1
```

注：如果文件中只有两个单词，将没有单词被删除。

7. 编写一个 sed 命令，删除一个文件每行中的最后一个单词。

```
# sed 's/\(.*\) .*/\1/' test1
```

8. 编写一个 sed 命令，交换一个文件每行中的第 1 个字符和第 2 个字符

```
# sed 's/\(.\)\(.\)/\2\1/' test1
```

9. 编写一个 sed 命令，交换一个文件每行中的第 1 个字符和最后一个字符

```
# sed 's/\(.\)\(.*\)\(.\)/\3\2\1/' test1
```

10. 编写一个 sed 命令，交换一个文件每行中的第 1 个和最后一个单词

11. 编写一个 sed 命令，删除一个文件中说有的整数

12. 编写一个 sed 命令，删除俄米行开始处所有的前导空格

13. 编写一个 sed 命令，用制表符替换每行开始处所有的单一空格

14. 编写一个 sed 命令，把所有大写字母用括号括起来

15. 编写一个 sed 命令，打印每行 3 次

16. 编写一个 sed 命令，隔行删除

17. 编写一个 sed 命令，把行 22 到行 33 复制到行 56 后面。

18. 编写一个 sed 命令，把行 22 到行 33 移动到行 56 后面

19. 编写一个 sed 命令，把行 22 到行 33 移动到行 9 后面

20. 编写一个 sed 命令，抽取每行的第一个单词

21. 编写一个 sed 命令，打印一行的第一个和第 3 个单词

22. 编写一个 sed 命令，从形式为 mm/dd/yy 的日期中抽取月份；日期；年份

第十二节

awk

awk 实用程序有两个 UNIX 选项。-F 选项规定输入字段分隔符。-f 选项对脚本文件命名。当脚本包含在命令行时，它应该用引号引起来，在 shell 中形成保护。

一、 执行

awk 实用程序的调用与其他实用程序相同。除了输入数据外，awk 还需要一个或多个提供编辑指令的指令。当只有少数指令时，可以在命令行从键盘输入。但大多说情况下，将他们放在称为 awk 脚本的文件中。Awk 脚本中每个指令包含一个模式（patterm）和一个动作（action）。

如果脚本简短，适于一行，可以直接在命令行编码。在命令行编码时，脚本放在引号内，命令行脚本格式如下：

```
#awk ‘patterm{action}’ input-file
```

对于较长的脚本，或对于要在规定时间内重复执行的脚本，选择一个单独的脚本更合适。脚本创建后，可以使用文件选项（-f）执行它，此选项告诉 awk 脚本在一个文件中。

```
#awk –f scriptFile.awk input-file
```

二、 字段和记录

awk 实用程序将文件看成字段和记录的集合。字段时有信息内容的数据单位。awk 中每个信息字段与其他字段用一个或多个空格字符或用户定义的其他字符分隔。

awk 中每行是一个记录。记录时作为一个基本单位的字段集合。一般而言，记录中所有的数据都应该有关系。

当文件由有组织成记录的数据组成时，就成为数据文件，与由单词、行和段落组成的文本文件相对照。

1. 缓冲区和变量

awk 实用程序提供两种缓冲区：记录和字段。缓冲区是在数据处理期间保存数据的存储器区域。

a. 字段缓冲区

输入文件的当前记录有多少字段，就有多少字段缓冲区。每个字段缓冲区有一个名称，是美元符号（$）后跟当前记录的字段号。字段号从 1 开始，为 $1（第一字段缓冲区）。

b. 记录缓冲区

只有一个记录缓冲区。其名称为 $0。它拥有整个记录。即，内容是所有字段的合并，其中每个字段之间有一个字段分隔符。

只要任何字段的内容没有改变， $0 就拥有与输入文件相同的数据。但是，任一字段改变，$0 的内容，包括字段分隔符，都发生变化。

2. 变量

awk 中有两种不同类型的变量：系统变量和用户定义变量

有多于 12 个系统变量供 awk 使用：其名称和作用都是由 awk 定义的。有 4 个是由 awk 完全控制的，其他有标准的默认值。

变量	功能	默认
FS	输入字段分隔符	空格或 tab
RS	输入记录分隔符	换行
OFS	输出字段分割符	空格或 tab
ORS	输出记录分割符	换行
NF	当前记录非空字段的编号	
NR	从所有万分建读入的记录号	
FNR	记录读取的文件编号—当前文件的记录编号	

FILENAME	当前文件名	
ARGC	命令行参数的数量	
ARGV	命令行参数数组	
RLENGTH	由内置字符串函数匹配的字符串长度	
RSTART	由内置字符串函数匹配的字符串开始位置	

用户定义变量：

在 awk 脚本内可以定义任意多的用户定义变量。可以使数字、字符串或数组。变量名以字母开头后跟任何序列的字母、数字和下划线。不需要声明：仅当它们被首次引用时开始存在。所有变量最初创建为字符串，并初始化为空字符串（“”）。

三、 脚本

所有 awk 脚本分为 3 部分：开始、主体和结束。

1. 初始化处理（BEGIN）

初始化处理仅在 awk 开始读取文件之前一次完成。它由关键字 BEGIN 来标识，指令封入一组大括号内。

开始指令用来初始化变量、创建报告标题并完成文件处理开始之前必须完成的其他处理。

2. 主体处理

主体是一个处理文件数据的循环。awk 从文件读取首记录或行时，主体开始。然后可以通过主体指令处理数据，并将他们合理应用。当到达主体指令结束时，awk 通过读取下一个记录或行来重复处理过程，并依靠主体指令处理它。awk 实用程序通过主体内的指令逐个处理文件的每个记录或行。

注：awk 命令不像其他实用程序，awk 不写入或打印记录，除非文件中有明确的指令指定这么做。

3. 结束处理（END）

所有输入数据被读取后，执行结束处理。此刻，处理期间积累的信息可以被分析或打印，或进行其他的结束动作。

四、 操作

五、 模式

awk 实用程序可以使用几种不同类型的模式。当它执行脚本时，对文件找到的记录模式求模式值。如果模式匹配记录，就采取动作，否则，就跳过动作。没有模式的语句总为 ture，即始终要采取动作。

awk 模式分为两类：简单和范围

1. 简单模式

简单模式匹配 1 个记录。当匹配与 1 个记录匹配时，结果为 true，并执行动作语句。有 4 种简单模式：BEGIN、END、表达式和无（无表达式）。

a. BEGIN 和 END

在文件开头读取第 1 条记录之前 BEGIN 为真。他用于处理任何数据之前初始化脚本；例如，设置字段分隔符和其他系统变量。

END 用在脚本的结束。典型的用法是打印处理期间积累的用户定义变量。

b. 表达式

awk 实用程序支持 4 个表达式：正则、算术、关系和逻辑。

- 正则表达式

  awk 正则表达式（regexp）是那些在 egrep 中定义的表达式。除了表达式外，awk 需要两个运算符之一:匹配（~）与不匹配（!~）。使用正则表达式时，记住它必须放在两个斜线“/”和斜线“/”之间。

```
$0~/^A.*B$/       #记录必须是以 A 开头且 B 结尾
$3!~/^ /          #3field 不能以空格开头
$4!~/bird/        #4field 不能包含 bird
```

- 算术表达式

  算术表达式是算术运算符的结果。如果表达式是算术的，当值为非 0 时，即为正或负时，它匹配记录；值为 0 时（false）不匹配记录。

运算符	举例	说明
*/%^	a^2	变量 a 的 2 次方
++	++a，a++	a 加 1
--	--a，a--	a 减 1
+-	a+b，a-b	两值相加或相减
+	+a	一元加号；值不变
-	-a	一元减号；值求补
=	a=0	a 赋值为 0
*=	x*=y	x=x*y；将 x*y 结果赋值给 x
/=	x/=y	x=x/y；将 x/y 结果赋值给 x
%=	x%=y	x=x%y;其中%是模运算;即将 x/y 的模赋值给 x
+=	x+=5	x=x+5
-=	x-=5	x=x-5

- 关系表达式

  关系表达式比较两值，并取定第 1 个值是小于、等于还是大于第 2 个值。若两值是数字使用代数方法比较；若为字符串就使用字符串比较。如果字符串与数字比较，将数字转换成字符串再用字符串比较。但是，注意可以通过线数字强加一个空字符串，使其成为字符串，也可通过给字符串强加一个 0，使它成为数字。

运算符	说明
<	小于
<=	小于等于
==	等于
!>	不等于
>	大于
>=	大于等于

- 逻辑表达式

  逻辑表达式使用逻辑运算符联结两个或多个表达式

运算符	说明
!expr	非表达式
expr1&&expr2	与表达式
expr1||expr2	或表达式

逻辑“与”表达式的结果只有当两个表达式都为 true 时才为 true，如果其中一个为 false 则为 false。或表达式只要有一个为 true，则结果为 true。

2. 无（无模式）

当无地址模式输入时，awk 在输入文件的所有行应用动作。这是指定处理所有行的最简单的方式。

3. 范围模式

范围模式与记录或行的范围相联系。它是由两个逗号分隔的简单模式组成。如下所示：

```
start-pattern,end-pattern
```

范围从与开始模式相匹配的记录开始，到与结束模式相匹配的记录结束。如果开始和结束模式相同，那么范围只有一个记录。

每个简单模式只能匹配一个表达式：表达式不能为 BEGIN 或 END。

如果范围模式在文件匹配多组记录，那么每组就采取一个动作。但是，组不能重叠。因此，如果结束范围之前的开始范围出现了两次，那么只有一个匹配组，即从第 1 个匹配记录到最后一个匹配记录。如果没有匹配结束范围，匹配组就以文件中开始匹配记录开始，以最后记录结束。

六、 动作

编程语言总动作称为指令或语句。在 awk 中成为动作是因为他们当模式为 ture 时行动。

awk 中动作是与一个模式相联系的一个或多个语句。在动作和模式之间只是一对一的关系：一个动作只与一个模式相联系。动作语句必须放在一组大括号中；及时只有一个语句也必须使用大括号。一组大括号包含模式/动作对或称为块的语句。当动作由几个语句组成时，他必须用语句分隔符分隔。awk 中语句分隔符有分号、换行符或一组大括号。

在 awk 中，结束语句由换行符、分号或闭大括号来指定。

awk 脚本可以在脚本开头有一个可选的模式/动作对，在脚本结束有一个可选的模式/动作对，在脚本主体有 0 个或多个模式/动作对。但这种限制不严格，因为 awk 将块当作一个语句。

1. 表达式语句

表达式即可以用在模式中，又可以用在指令的动作部分。

当表达式用在模式部分时，它有一个值，有时候有效果。值不是 true（非 0）就是 false（0），用来选择或跳过要处理的行。如果产生效果，将改变缓冲区或或变量的内容。

当表达式用在动作部分时，它也有值和效果。但是，值为数学值或逻辑值（true 或 false），其效果可以改变变量或缓冲区（$1 或 $9）的值。当使用表达式只是为了其效果而放弃其值时，表达式就是表达式语句。

2. 输出语句

awk 中有 3 个输出语句。第一个是 print，第二个是 c 格式化的打印语句 printf，第三个是 sprintf。

a. 打印（print）

打印将指定的数据写到标准输出文件。每个打印动作写一个单独的行。当要写多个字段或变量时，它们必须用逗号分隔。如果没有指定数据，就打印整个记录。每行被处理时，都被拷贝到标准输出。由于打印整行，所以使用行字段分隔符（制表符），将每个字段对齐到列中。

b. 格式化打印（printf）

awk 实用程序还包括 C 格式化打印语句 printf。每个 printf 动作由格式化字符串组成，封入双引号内，打印一列 0 个或多个值。数据格式由字段规范来描述，用一个百分号开头，用定义数据类型的格式码来结束。任何不是字段规范的内容都是要打印的文本。

%	标志	最小宽度	精度	转义码

字段规范包括 3 个修饰语。宽度用于打印队列对齐数据。但是，注意当使用时它指定最小的宽度。如果数据需要的空格多于列宽度规范中指定的数目是，数据将上溢到下面的打印区域。

标志用列控制数据的格式。通常，数据打印时右对齐；左对齐使用减号，将文本放置在打印区域的最左端。符号标志为数字数据加上标记，正值用加号，负值用减号。空格标志不是用加号，而是用开头的空格打印正数。最后，零标志用开头 0 替代空格来格式化数字值。在将值打印为数字标识符时，这很重要。

精确度修饰语指定了浮点数字打印时的小数位数。其格式为.n，n 表示要打印的精确度位数。与宽度规格结合在一起时就成了 m，n。例如要将美元值打印在一个 10 位数的列中，就要使用 10.2 的宽度精确度规格。

字段规范的结尾是变换码。它描述了被打印的数据类型。大部分变换码是自解释，浮点格式需要另外解释。当浮点数字打印成标准格式如美元和美分时，使用 f 变换码。如使用科学计数法时，使用 e 变换码。

如：

```
$awk ‘{printf ( “%2d %-12s $%9.2f\n”,$1,$2,$3)}’ sales2.dat | head -5
```

c. 字符串打印（sprintf）

此命令使用格式化打印概念将两个或多个字段合并到一个字符串，并在后面的脚本中用作变量。

3. 判定语句

判定语句是做选择的语句。除了使用模式匹配隐式的进行选择的方法。awk 也提供了动作语句显示的选择。

a. if-else 语句

if-else 动作语句可以求表达式的值并采用相应的动作。真动作是始终需要的，尽管可能为空语句。但假动作是可选的。可以使用任何归纳为 true 或 false 的表达式，最常见的是逻辑表达式。

当 if-else 中动作语句是另一个 if-else 时，就称为嵌套 if 动作语句。使用嵌套 if 语句必须十分小心，确保 else 动作与恰当的 if 正确联合。每个 else 语句与最近的不成双的 if 语句匹配。

4. 控制动作

有 3 个语句控制整个脚本的执行：next、get-line 和 exit。

a. next

next 语句结束当前记录的处理，并开始下一个记录的处理。如果没有更多的记录可处理，当指定了结束动作是，它将控制转换到结束动作。这就等于立即移到脚本结尾，并读下一个记录。

b. getline

与 next 一样，getline 读一个记录（行）；但又与 next 不同，它会继续执行脚本，而不会返到主体的开始处。getline 之后的动作和语句将应用于刚刚读取得新记录，其格式：

```
getline
```

虽然 getline 的用法好似一个语句，但实际上是一个函数。作为函数，它有 3 个强大性能：

- 输入可以指向＄0（默认）或单独的变量
- getline 返回一个可以被估计的值，值为：
  - a：1，读记录成功
  - b：0,文件结尾
  - c： -1，读错误
- 可以从另一个文件输入

awk 脚本必须有一个输入，要么为标准输入，要么是传递为参数的文件。此文件可以自动被脚本处理或使用 getline 读取。不管它如何读取，当读取到文件结尾时，awk 将控制产地给结束部分，或者如果没有结束部分，就中止执行。

getline 函数可以使用重订向运算符（<）读取其他文件。

```
getline Variable < file
```

变量是可选的。如果没有提供变量，字段将读入 $0。由于当读文件是 getline 并不自动跳到文件结尾的结束处理，所以当语句的其它部分没有处理时，应该测试 0 返回之和结束处理分支，

c. exit

当 next 跳到脚本结尾并读下一个记录时，exit 终止脚本。如果已经指定了结束模式，它执行结束语句然后退出。如果没有指定结束，它立即退出脚本。当结束模式中使用 exit 时，脚本会立即终止，而不执行语句的其他部分。

exit 应该保留给错误条件。

5. 循环

a. while 循环

while 循环当表达式为 true 时进行迭代。如果无法提前知道循环中迭代的次数就应该用 while 循环。

此循环为预先测试循环。在每次迭代之前，都进行界限测试，包括第一次。一次如果开始时，界限条件为假，将没有循环动作可做。

```
# more students.dat
1234    87      83      91      89
2345    71      78      83      81
3456    81      82      79      89
5678    78      86      81      79
# more stuWhile.awk
# stuWhile.qwk script
 {
   total=0
   count=0
      i=2

   while (i<=NF)
       {
         total+=$i
         count++
         i++
       } #while

#test for zero divide
  if (count>0)
    {
      avrg=total/count
       print($1,avrg)
    }#zero divide test
```

```
}#body
# awk -f stuWhile.awk students.dat
1234 87.5
2345 78.25
3456 82.75
5678 81
```

b. for 循环

与 while 循环一样，for 循环也是预先此时循环。for 循环是独立循环：循环初始化、界限测试和更新都包含在循环自身内部。其格式为：

```
for (初始化： 界限测试；更新)
语句
```

```
# more stuFor.awk
# for loop example
{
 total=0
 count=0
 for (i=2;i<=NF;i++)
       {
       total+=$i
       count++
       }#for
} #end of student scores
#test for zero divide
count>0 {
       avrg=total/count
       print($1,avrg)
       }#zero divide test
#end
# awk -f stuFor.awk students.dat
1234 87.5
2345 78.25
3456 82.75
5678 81
```

注意在以上代码中将 for 循环的所有 3 个部分写在了一行上。如果需要，也可以通过转移结尾的换行符实现将么个写入单独的行。例如：

```
for (i=2;\
     i<=NF;\
     i++)
```

c. do-while 循环

用途最小的循环是 do-while。实际上，它并不包括在所有的 awk 实现中。为了保证循环至少执行一次，将界限测试移至循环结尾。因此，它也称为后测试循环。

```
# more stuDowhile.awk
#do-while example
{
total=0
count=0
i=2
}#initialization
NF>i{
        do
        {
         total+=$i
         count++
         i++
        }#do body
        while (i<=NF)

        avrg=total/count
        print($1,avrg)
        } #NF>1
#end script
# awk -f stuDowhile.awk students.dat
1234 87.5
2345 78.25
3456 82.75
5678 81
```

注意整个 do 动作语句必须在一组大括号内。这是因为 do-while 本来只能有一个动作语句。当需要多个动作语句时，它们必须放在大括号内。

七、 关联数组

数组可以作为单个或整个引用的变量集合。要引用数组中单个元素，应使用索引：要使用整个数组（集合），应使用数组名称。

awk 使用字符串作为索引。每个索引项是标识存储在数组元素中的数据的关键字或词组；既索引项与数组元素相关联。因此，数组称为关联数组。但是，尽管索引和元素相联并标识元素，但元素值与索引不同。

当使用关联数组时，必须记住它们的几个设计约束条件。

i. 索引必须是唯一的
ii. 数据值可以是重复的
iii. 保证索引与其值得关联
iv. 索引没有强加顺序
v. 数组索引不能被排序，数组中的数据值可以被排序

1. 处理数组

a. 数组循环：for…in 循环

要处理关联数组，awk 提供了一个新的循环格式：for…in 循环。循环的一般格式为：

```
for (index_variable in array_name)
```

索引变量可以为数据或者仅仅为序列号。

数组元素在任何时候首次使用数组运算符（方括号）时创建。

- 例 1：读取文件中的学生数据并按相反的顺序打印：

```
# more listStuBackward.awk
#listStuBackward.awk
{lines[NR]=$0}

END {
       for (i=NR;i>0;i--)
              print lines[i]
       } #end END
# awk -f listStuBackward.awk students.dat
5678    78      86      81      79
3456    81      82      79      89
2345    71      78      83      81
1234    87      83      91      89
```

主程序中只有一个语句。它将当前记录（$0）赋给名称为 lines 的数组。NR 是当前记录号（更确切的说是此脚本读取得记录号）的 awk 系统变量。当读取了全部数据后，代码清单转到 END。for 语句从数组结尾（NR）开始并朝第 1 项移动，每次打印 lines[i]表示的行。注意，i 不是数字，是字符串。

- 例 2，创建一个大学书店销售额文件确定每个部门的销售额总和。

```
# more sales1.dat
1       clothing        3141
1       computers       9161
1       textbooks       21312
2       clothing        3252
2       computers       12321
2       supplies        2242
2       textbooks       15462
# more salesDeptLoop.awk
#salesDeptLoop.awk script
BEGIN  {OFS="\t"}
{deptSales[$2]+=$3}
END    {for (item in deptSales)
              {
               print item, ":",  deptSales[item]
               totalSales+=   deptSales[item]
              }#for
       print "Total Sales", ":", totalSales
}#END
# awk -f salesDeptLoop.awk sales1.dat
computers       :       21482
supplies        :       2242
```

```
textbooks       :       36774
clothing        :       6393
Total Sales     :       66891
```

对任何部门的数组，所需要做的全部工作就是创建新变量并使用方括号将它标识为数组元素，使用部门字段编号建立索引。

要打印数组，创建一个索引变量并将它命名为 item。此脚本主处理过程创建的数组名称是 deptSales。要打印数组，在循环中使用索引变量和数组。如下语句所示：

```
for (item in deptSales)
```

index_in 格式也可以用来确定索引值是否已经存在于数组中。在部门销售的实例中，就可以使用下列代码测试任何字段：

```
if (“magazines” in deptSales)
```

如果杂志是数组中的索引，此语句将返回 true，否则为 false。此格式还可以与变量一起使用。例如，要测试当前记录的部门是否在数组中，可以使用以下语句：

```
if ($2 in deptSales)
```

b. 删除数组项

delete 函数从数组中删除元素。其格式为：

```
delete array_name[index]
```

索引必须与索引（关联数组）项匹配。如果索引一数组中的项不匹配，不做动作，脚本继续执行。

```
# more salesDltEntry.awk
#salesDeptLoop.awk script
BEGIN  {OFS="\t"}
{deptSales[$2]+=$3}
END    {print "Deleting \"supplies\" index entry"
              delete deptSales["supplies"]
       for (item in deptSales)
              {
               print item, ":",  deptSales[item]
               totalSales+=   deptSales[item]
              }#for
       print "Total Sales", ":", totalSales
}#END
# awk -f salesDltEntry.awk sales1.dat
Deleting "supplies" index entry
computers       :       21482
textbooks       :       36774
clothing        :       6393
Total Sales     :       64649
```

八、 字符串函数

awk 实用程序包含丰富的字符串函数数组。

1. length

length 函数格式：

length （字符串）

长度返回字符串参数中的字符数。

例如：统计文件中的字符总数，包括空格字符，但不包括换行符。

```
# more countChar.awk
{
 print len=length($0),"\t",$0
 cntChar+=len
}
END {print cntChar "Total characters in", FILENAME}
```

2. 索引（index）

index 函数返回一个字符串内第 1 个子字符串的位置，其格式：

index （字符串，子字符串）

字符串可以是字符串常量或变量。如果没有找到字符串，返回 0。

3. 子字符串（substr）

索引返回子字符串的开始位置，而 substr 函数从字符串中抽取子字符串。它有两个格式：

substr （字符串，位置）

substr （字符串，位置，长度）

两个函数唯一的区别是返回的数据长度。两者都是从字符串开始位置返回子字符串。如果指定了长度，将返回指定长度的字符数。如果没有指定长度，将返回直到字符尾的所有字符。

4. 分割（split）

split 有两种格式：

split （字符串，数组）

split （字符串，数组，字段分隔符）

第 1 个格式中，字符串中的字段被复制到数组中。每个字段的结尾有字段分隔符字符（即系统变量 FS 的当前设置）标识。第 2 种格式中，字段分隔符被指定为第 3 个参数。

5. 替代（sub）

字符串替代函数 sub，其格式是：

sub (regexp,replacement_string,in_string)

如果替代成功 sub 函数返回 1（true），如果目标字符串没有找到就返回 0（false），并且不进行替代。可以在替代字符串中使用保留模式(&)，但不能使用向后引用，如\1 或\2。

```
# more stringSub.awk
 #stringSub.awk
{success=sub(/bird/,"RAVEN",$0)}
{if (success>0
       #if  a substitution was made
       print NR, $0
}#if
#end script
```

6. 全局替代（gsub）

可以使用全局替代（gsub）改变所有出现的值。全局替代函数的格式与 sub 相同。唯一的区别就是它替代所有出现的匹配文本。

7. match

match 字符串函数返回行中匹配表达式的起始位置。如果没有匹配的字符串，它返回 0。另外，它设置两个系统变量：RSTATR 对应起始位置，RLENGTH 对应匹配文本字符串长度。其格式如下：

```
startPos=match(string,regrxp)
```

例如：打印所有包含从一行开始到逗号结束的字符串的行。

```
# more stringMatch.awk
#stringMatch.awk
{if (match ($0,/^.*,/)>0)
      print NR,substr($0,RSTART,RLENGTH)
}#end script
```

8. toupper 和 tolower

这两个函数是将大写字符转换为小写字符，或反之亦然。toupper 函数将字符串中的小写字符转换为大写，任何非小写字符不变。tolower 将大写字符转换为小写字符，不改变大写字符。

九、 数学函数

函数	注释
int	将浮点值舍位为整数
rand（）	返回系列中下一个随机数，范围 0…1
srand（seed）	种子随机数系列，种子应该是素数
cos(x)	返回余弦
exp(x)	返回 e^x
log(x)	返回 x 的自然对数（基数是 e）
sin(x)	返回 x 的正弦
sqrt(x)	返回 x 的平方根
atang2(y,x)	返回 y/x 的反正切，范围在-π到+π

十、 用户定义函数

在 awk 中，可以编写自己的函数。其格式需要函数头，主体和可选的返回语句。基本格式如下：

```
function name (parameter list)
{
    code
}
```

尽管概念与 C 相似，但有些语法差异需要注意：

1. 函数名称结尾和函数调用的参数列表的开圆括号之间不能有空格。
2. 函数名称结尾和函数定义的开圆括号之间不能有空格。
3. 函数语句最后没有分号
4. 变量在使用之前不需要声明

十一、 在 awk 中使用系统命令

系统命令在 awk 脚本中使用有 3 种使用方法：管道、循环和系统函数。

1. 使用管道

当程序的结果只有一行时，就可以将程序的编码为双引号内的字符串，并将结果管道传递回脚本。管道输出用 getline 命令读取。

例 1：获取日期

```
# more date.awk
#date.awk
BEGIN {
```

```
        "date"|getline
        print($1,$4)    #Day of Week&time of day
} #end of BEGIN
# awk -f date.awk
Wed 00:30:36
```

2. 使用系统函数

awk 实用程序包含一个系统函数，它执行传递给他作为参数的命令。命令必须返给成功（0）或失败（1），然后在传递给脚本。

例如：编写一个脚本修改已有文件时，将它复制一个备份文件作为脚本的一部分。

```
# more sysCopy.awk
BEGIN{
        if (system("cp sysCopy.dat sysCopy.bak") !=0)
                {
                 print ("Errot coping sysCopy.dat")
                 exit
                }
                print "sysCopy.dat copied"
    }
```

十二、 应用程序

十三、 awk 和 grep

十四、 awk 和 sed

十五、 练习

1. 编写一个脚本打印所有输入行

2. 编写一个脚本打印第 8 行

3. 编写一个 awk 命令，打印所有输入行的第 1 个字段的值

4. 编写一个 awk 命令，打印输入行总数

5. 编写一个 awk 命令，打印每行的字段数

6. 编写一个 awk 命令，打印最后一行的最后一个字段的值

7. 编写一个 awk 命令，打印多余 4 个字段的所有输入行

8. 编写一个 awk 命令，打印文件中字段的总数

9. 编写一个 awk 命令，打印多于 5 个字段的行的总数

10. 编写一个 awk 命令，如果其第一个字段大于 9 就打印第 2 个字段

11. 编写一个 awk 命令，打印所有的行，字段顺序必须是字段 4、字段 3、字段 2 和字段 1。

12．编写一个 awk 命令，打印 5 到 56 行

13．编写一个 awk 命令，在文件顶部加上标题“Document”

14．编写一个 awk 命令，隔行删除

15．编写一个 awk 命令，每行抽取第一个单词

16．编写一个 awk 命令，打印每行的第 1 和第 3 个单词

第十三节

交互式 Korn shell

一、 Korn shell 特性

1. Korn shell 会话
2. 标准流

   定义 3 种标准流——标准输入（0）、标准输出（1）和标准错误（2）。
3. 重定向
4. 管道
5. tee 命令
6. 组合命令

   有 4 种组合命令方式：顺序命令、分组命令、链式命令和条件命令。
7. 命令行编辑
8. 引号
9. 命令替代
10. 作业控制
11. 别名

    别名用 alias 命令创建。其格式如下：

    alias name=command-difinition

    列出别名

    alias

    删除别名

    unalias

二、 两个特殊文件

1. 垃圾文件(/dev/null)

   垃圾文件是用来删除数据的特殊文件。在设备（dev）目录下可以找到，它有一个很特殊的特征：在收到数据后，其内容总是立即清空。物理上系统只有一个垃圾文件：属于超级用户。此文件是一个可以被任何用户读写的字符特殊文件。

   因为是一个文件，所以既可以用作源文件也可以用作目标文件。不过，作为源文件时，结果总是文件尾。因为它总是空的。
2. 终端文件（/dev/tty）

   在 UNIX 中每个终端都有一个命名文件，但实际上只有一个逻辑文件/dev/tty。此文件在设备目录下：每个用户的终端。普通终端文件/dev/tty 属于 root 用户；所有人都可以对它读写，它也可以作为源文件或目标文件。终端文件代表终端，不能存储数据。

三、 变量

Korn shell 允许在变量中存储值。shell 变量是在一个内存中存储值的位置。在 Korn shell 中，所有数据均被存储为字符串。变量可分为两类：用户定义和预定义。

1. 用户定义变量

   用户定义变量是由用户创建。变量名称不应该选择与预定义变量相同的名称。每个变量必须有一个名称，变量名称必须以字母或下划线字符开头，后面可以跟多个字母或下划线字符。
2. 预定义变量

   预定义变量是 shell 变量或环境变量。shell 变量用来配置 shell。

3. 在变量中存储值

有几种在变量中存储值的方式。最简单的方法是用赋值运算符对变量赋值。

4. 访问变量值

要访问变量的值，变量名称前面必须加上一个美元符号。变量名称的美元符号不应该与系统提示符 $ 混淆。存储在变量中的值可以有很多用途。值可用在字符串的任何位置。也可以存储在另一个变量中。

5. 空变量

如果访问的变量没有设置（没有存储值），将接收一个空值（无）。还可以通过赋给它一个控字符串“”或不赋值在变量中明确地存储控制。即：任何变量的默认值是空。

6. 清除变量

使用命令 unset 清除变量

7. 存储文件名称

还可以在变量中存储文件名，甚至可以使用通配符。shell 将包括通配符的文件名存储到变量中，而不展开他。当使用值时，就会展开。

8. 存储文件内容

还可以把文件内容存储到变量来处理。注意变量不包括文件中所有额外的空格，将文件作为一个字符串，单词间用单个空格字符分割，所有非空空白字符转化为一个空格，所有多余空格被删除。

9. 在变量中存储命令

在变量中存储命令只对单一命令有用，如果命令是复合的（例如使用管道），命令变量将不起作用。对于符合命令，需要使用 eval 命令

10. 只读变量

名称常量定义了一个不可改变的值。尽管 Korn shell 没有名称常量，但可以通过创建一个变量，给它赋值，然后用 readonly 命令来固定它，产生同样的效果。命令格式如下：

```
readonly variable-list
```

注意命令希望将变量列表作为其参数。这表明可以一次创建多个只读值。

11. 输出变量

一个 shell 可以创建另一个 shell。新的 shell 称为 subshell，或称为子 shell。创建另一个 shell 的过程称为派生（forking）一个 shell。shell 变量不会自动输出到子 shell 中的，如果让变量对子 shell 有用，必须告诉当前 shell 将它输出。export 格式如下：

```
export variable-list
```

一个命令输出可以导出多个变量。一旦变量标记为可输出的，无论何时派生新 shell，变量都会在子 shell 中创建，并用变量的当前值初始化。如果在子 shell 中再创建一个子 shell，变量将会再次被初始化。每个 shell 的变量是单独实体；如果改变子 shell 的变量值，父 shell 的值不会改变。

12. 变量属性：typeset 命令

在 Korn shell 中，通过 typeset 命令可以更改一个或更多的属性。一旦属性和变量联系起来，就可以控制变量的内容和格式。要联系变量和属性，把属性传递给 typeset 命令作为一个属性，格式如下：

```
$typeset –attribute variable_name
```

要从变量中删除一个属性，使用如下格式：

```
$typeset +attribute variable_name
```

注意，属性有改变变量值物理格式的作用。

属性	特征
l(小写 L)	小写
U	大写
i	整数
Rn	右对齐，宽度为 n 个字符（额外截去）
Ln	左对齐，宽度为 n 个字符（额外截去）
x	自动输出
r	只读
RZn	右对齐，宽度为 n，（额外截去），并用 0 填充
LZn	左对齐，宽度为 n，（额外截去），并用 0 填充

四、 输出

Korn shell 的输出语句是 print 命令。print 命令从参数中创建一个文件（在标准输出流），其参数可以是字符串或变量。注意：使用变量时必须遵守引用规则：变量不能直接嵌入到单引号中。print 命令自动在最后一个参数后加上一个终止换行符。如果由于某种原因不需要换行符，使用-n 选项。为格式化输出，有 9 种像 c 一样的转义码可用。使用这些转义码时，必须放在引号内。

代码	用法
\b	退格
\c	没有换行符（与-n 相同）
\f	换页
\n	换行符
\r	回车符
\t	制表符
\v	纵向制表符
\\	打印反斜杠
\0ddd	打印 ASCII 字符的八进制代码

五、 输入

从终端或文件读取数据使用 read 命令。read 命令读一行并将文字存储到变量中。它必须用回车来结束，输入行必须紧随命令之后。

1. 逐词读

当执行 read 命令时，shell 从标准输入读一行并将它逐词存储到变量中。词是以空格或制表符分隔的字符。第 1 个词存储到第 1 个变量中，以此类推。即 read 命令把输入的字符串（行）分解成词。如果此的数量多于变量数，所有多余的词放在最后一个变量中。如果词少于变量，未分配的变量值为空。在读之前其中的任何值都将丢失。

```
# read word1 word2 word3
Now is $date
# print $word1
Now
# print $word2
is
# print $word3
$date
```

2. 逐行读

处理多余词的设计提供了一种将整行存储到一个变量中的简单技术。仅使用 read 命令，给它一个变量。执行后，整行都存储到变量中。

read 命令从键盘或一个重定向文件只读首行；要读取多行或整个文件，必须使用循环。

3. 从文件中读

Korn shell 允许脚本从一个用户文件中读取，通过流描述符选项（-u）来实现。流描述符是文件的数字指示。

```
read -u4 variable_name
```

六、 命令的退出状态

在 Korn shell 中，当执行命令时，返回一个称为命令的退出状态的值。退出状态用一个（？）名称存储在 shell 变量中。像其他命名变量一样，退出状态可以用其名称访问（$？）。如果命令执行成功，返回一个 0 值，解释为 true。如果不成功，返回一个非 0 值（false）。

七、 eval 命令

eval 命令用在 Korn shell 中执行一个命令之前需要对它两次求值得情况下。例如：

```
# x=34
# y=x
# eval print \$$y
34
```

当执行 eval 命令时，其第 1 次求值 $y，产生字符串值$x。第 2 次求值算出变量$x 的值，并将产生的正确结果，打印保存在 y 中的变量。

例：存储一个命令打印文件列表并将其结果管道传送给 head。

```
# list="ls -l|head -4"
# eval $list
total 40
-rw-r--r--   1 root     sys            57 Dec  4 10:27 a.txt
-rw-r--r--   1 root     sys            93 Dec  5 16:41 listStuBackward.awk
-rw-r--r--   1 root     sys            47 Dec  4 11:19 mingling
```

八、 环境变量

环境变量控制用户环境。

变量	说明
CDPATH	当目录参数是相对路径名称时，包含 cd 命令的搜索路径
COLUMNS	按字符定义的终端宽度。默认为 80
EDITOR	命令行编辑器的路径名称
ENV	环境文件的路径名称
HISTFILE	历史文件的路径名称
HISTSIZE	历史文件保存命令的最大数量
HOME	用户根目录的路径名称
LINES	按行定义终端的显示高度，默认为 24
LONGAME	在/etc/passwd 文件中包含用户的登录名称
MAIL	用户邮箱的绝对路径名称
MAILCHECK	检查新邮件的时间间隔，默认为 600 秒

OLDPWD	在最后的 cd 命令之前的工作目录的绝对路径名称
PATH	命令的搜索路径
PS1	主提示符，如 $ 或%
PS2	次提示符，用在首行还没有输入完整命令时，默认>
PS3	选择命令提示符，只为 select 命令所用
PS4	调试提示符，默认为加号（+）
PWD	当前目录的绝对路径名称
RANDOM	每次调用时返回一个随机数
REPLY	read 命令的临时缓冲区
SECONDS	shell 激活后的秒数
SHELL	登录 shell 的路径名称
TERM	终端类型
TMRM	决定终端自动退出系统之前空转时间的变量。默认为 0，表示会话永不超时。
VISUAL	与 EDITOR 相同。但 VISUAL 与 EDITOR 优先级更高。如果设置了 VISUAL 系统就不检查 EDITOR 值。

九、 选项

在 Korn shell 中使用选项控制其命令的执行方式。

选项	说明
allexport	设置输出的所有变量
emacs	对命令行编辑器和历史文件使用 emacs
ignoreeof	不允许使用 Crtl+d 退出 shell
noclobber	不允许重定向损坏现有文件
noexec	禁止通配符展开
noglob	对脚本的命令进行读取和句法检查
verbose	在执行命令之前将其打印
vi	对命令行编辑和历史文件使用 vi
xtrace	在执行之前打印命令和参数

用来设置、清除和显示选项的命令

操作	命令
设置	set –o option
清除	set +o option
全部显示	set -o

十、 启动脚本

1. 系统配置文件

在目录 etc 中存在一个系统配置文件。其中包含应用于每个系统用户的普通命令和变量设置。

2. 个人配置文件
3. 环境文件
4. 启动过程

十一、 命令历史记录

5. 历史文件

所键入的每一个命令都保存在用户根目录的历史文件中。默认文件名为~/.sh_history。可以重新命名，其路径名称存储在 HISTFILE 环境变量中。其默认大小为 128,可以用 HISTSIZE 变量改变它的大小。

6. 历史命令

从历史文件中列表、编辑和执行命令的命令是 fc 命令，在 Korn shell 中包含一个预置别名 history。执行时不带任何参数，将列出最后 16 命令。

7. 重做命令（r）

8. 重做命令中的替换

十二、 命令执行过程

第十四节

Korn shell 编程

一、 脚本基本概念

shell 脚本是一个包含可执行命令的文本文件。

1. 脚本组件

每个脚本有 3 个部分：揭示其标志行、注释和 shell 命令

a. 解释器标志符行

脚本的首行必须是标志符行，甚至其前面不能有空行；告诉 UNIX 到适当的 shell 解释器的路径。标志符行用“#！”开头，如果删除标志符行，UNIX 将使用当前 shell 作为解释器。

```
#!/bin/ksh
```

b. 注释

注释是加到脚本上帮助理解的文档。注释用“#”标记。Korn shell 只支持行注释。

c. 命令

脚本可以使用 UNIX 的任何命令。

- 命令分隔符

  脚本中的命令与交互会话中一样，应该彼此分隔。shell 使用两个记号：分号和换行符。shell 会自动跳过首尾的空格。

- 空行

  命令分隔符可以重复。当脚本检测到多个分隔符时，把他们当成一个分隔符。表明可以在脚本中插入多个空行便于理解。唯一不能放空行的位置是 shell 解释器标志符前。

- 组合命令

  可以使用管道、分组命令或条件命令来链接命令。

2. 使脚本可执行

创建脚本后，通过 chmod 命令修改脚本权限使脚本可执行。

3. 执行脚本

脚本可执行后，他就是一条命令，可以向其他命令一样执行。有两个特定方法执行脚本：作为一个独立命令或作为子 shell 命令的参数。

a. 独立命令

作为独立命令执行脚本，仅仅使用其名称，如下所示：

```
$script_name
```

b. 子 shell 执行

确保脚本的正确执行，可以创建一个子 shell 并在 shell 中执行它。可以通过在脚本名称前指定 shell 来实现。如下所示：

```
$ksh script_name
```

4. 脚本终止（exit 命令）

当脚本必须停止时，使用 exit 命令。exit 命令结束并设置退出状态。它可以使用数字参数，也可以不带参数。当使用一个数字时，脚本的退出状态为指定数字，不带参数时，脚本退出状态为 0。

5. 参数和位置参数

Korn shell 使用参数和位置参数通用化脚本。参数是用户提供的数据，跟在命令行脚本名后面，输入脚本。位置参数是在 shell 脚本中预定义的内存变量（缓冲区）。有 9 个位置参数，标记为 $1 到 $9，用来保存用户输入的参数。

执行脚本时，shell 将第一个参数放在第 1 个位置参数（$1）中，依此类推，直到全部保存为止。

二、表达式

表达式用一系列运算符和运算对象求一个单值。运算符可以是数学运算符、关系运算符、文件测试运算符或者是逻辑运算符。

1. 数学表达式

a. 数学运算符

+、-、*、/（得到整数值）、%（得到余数值）

b. let 命令

Korn shell 使用 expr 命令或 let 命令对表达式求值，并将结果保存在另一个变量中。使用 let 命令时，变量不需要使用$。例如：

```
$let y=x+34
```

还有一种可供选择的运算符，一组双括弧，可以替代 let 命令。例如

```
（（y=x+34））
```

2. 关系表达式

关系表达式比较两值并返回逻辑值 true 或 false。

a. 关系运算符

逻辑运算符

数学比较	意义	字符串比较
>	大于	
>=	大于等于	
<	小于	
<=	小于等于	
==	等于	=
!=	不等于	!=
	字符串长度不为 0	-n
	字符串长度为 0	-z

字符串等于和不等于逻辑运算符支持第 2 个（右边）运算对象模式。如下表：

string	必须完全匹配第一个运算对象
?	匹配 0 个或一个单字符
[…]	在集合内匹配一个单字符
*	重复模式 0 次或多次
?(pat1|pat2|…)	匹配模式中的任何一个或 0 个
@(pat1|pat2|…)	恰巧匹配模式中的一个
*(pat1|pat2|…)	匹配模式中的 0 个或更多个
+(pat1|pat2|…)	匹配模式中的一个或更多个
-(pat1|pat2|…)	匹配非模式中的任何模式

b. 关系测试命令

Korn shell 使用测试命令((…))或[[…]]之一。整数使用((…));字符串使用[[…]]，并且需要在变量前加$符号。

3. 文件表达式

a. 文件运算符

运算符	解释
-r file	文件存在且可读时为 true
-l file	文件存在且是符号链接时为 true
-w file	文件存在且可写时为 true
-x file	文件存在且可执行时为 true
-f file	文件存在且是常规文件时为投 true
-d file	文件存在且是目录时为 true
-s file	文件存在且其大小大于 0 时为 true
file1 –n file2	file1 比 file2 新时为 true
file1 –ot file2	file1 比 file2 旧时为 true

b. 文件测试命令

尽管可以使用 test 命令，但在 Korn shell 中推荐使用双方括号运算符测试文件状态。

4. 逻辑表达式

a. 逻辑运算符

Korn shell 有 3 个逻辑运算符:非（!）、与（&&）和或（||）

b. 逻辑测试命令

使用双方括号

5. 表达式类型小结

三、 判定：作出选择

1. if-then-else

shell 从 if 后面的命令中求退出状态值。当退出状态为 0，就执行 then 命令组，当退出状态为 1，就执行 else 命令组。每个 if-then-else 语句用关键字 fi 结束，与语句开始用 if 关键字相反。

a. 退出状态

if-then-else 语句估算的条件必须是命令的退出状态。if 求值中的语句可以被链接（管道）：即，可以不止一个命令，当多个命令链接时，条件求值是链中最后命令的退出状态。

b. 测试命令求值

if 语句中表达式必须是一个命令，if 语句测试此命令的退出状态决定它执行成功与否。因此，要使用关系或逻辑运算符比较数据，必须使用测试命令或相等运算符——((…))和[[…]]。测试命令的状态为 true 或 fales。如果关系或逻辑运算符的结果为 true。测试命令的退出状态就是成功的（true）；相反，如果表达式结果为 false，测试命令的退出状态就是失败的（fales）。

c. 不带 else 的 if

有时会需要只测试 true 时的动作，false 时不需要动作。所以，只需要一个 then 语句，没有 false 的动作时，只需删除命令的 else（false）部分。

d. else 没有 if：空命令

尽管可以在没有 else 动作的情况下使用 if-then-else 语句，但却不能没有 then 动作。当 if-then-else 中没有为 true 时的操作，使用所谓的空命令进行 true 操作。空命令是一个冒号（：），除了满足 then 操作的需求外不做任何事情。

e. 嵌套 if 语句

当在 if-then-else 命令的 true 或 false 分支中发现另一个 if-then-else 语句，被称为嵌套 if。其使用一种简洁格式 elif。尽管 true 或 false 动作都可以使用嵌套，但更常用于 false 情况下的操作。

2. 多路选择

case 语句实现多路选择。假定有一个字符串和一个模式列表选择对象，case 语句依次将字符串与各种模式匹配。

a. case 语法

case 语句包含被求值的字符串。它以 case 结束标志 esac 结尾。在 case 语句的首位之间是模式列表。对要测试的所有模式，在模式列表中定义一个单独的模式。此模式以闭圆括号结束。与每种模式相联系的是一个或多个命令。命令必须遵守附加的正式规则，即最后的命令必须以两个分号来结束。模式列表中最后的操作通常是通配符*，如果没有任何一种其它情况相匹配，使之为默认。

```
# more caseDigit.scr
#!/bin/ksh
#       Script:caseDigit.scr
print "Enter a digit and I'll spell it for you: \c"
read digit
print "\n You entered $digit.It is spelled: \c"

case $digit in
        0) print Zero. ;;
        1) print One.  ;;
        2) print two.  ;;
        3) print three. ;;
        *) print Not a digit. ;;
esac
# ./caseDigit.scr
Enter a digit and I'll spell it for you: 3

 You entered 3.It is spelled: three.
# ./caseDigit.scr
Enter a digit and I'll spell it for you: e

 You entered e.It is spelled: Not a digit.
```

b. 通配符

美中匹配模式都结束 case，所以只有当没有任何其他模式匹配的情况下才选择默认匹配符。

四、 重复

1. 命令控制和列表控制循环

在 Korn shell 中循环分成两大类：命令控制和列表控制。

a. 命令控制循环

在命令控制循环中，命令执行决定是否执行循环体。Korn shell 中有两种命令控制循环：while 和 until 循环。

- while 循环

  while 循环是基本命令控制循环。用 while 开始，包含循环命令和命令退出状态为 true（0）时的多次循环。当退出状态为 false（1）时，循环终止。

```
# more loopadd.scr
#! /bin/ksh
#  Scipt:loopadd.scr

print "This utility adds numbers entered form the"
print "keyboard.When all numbers have been entered"
print "key ad(eof) to see the total. \n"

((sum=0))
print "Enter a number :  \c"
while read data
       do
              let sum=sum+data
              print "Enter next number: \c"
       done
print "\n      sum is: " $sum
# ./loopadd.scr
This utility adds numbers entered form the
keyboard.When all numbers have been entered
key ad(eof) to see the total.

Enter a number :  3
Enter next number: 5
Enter next number: 2
Enter next number: ^D
       sum is:  10
```

- until 循环

  until 循环用法与 while 循环相同，只是在退出状态为 false 时进行循环。在这种意义上它是 while 循环的补充。

```
# more until.scr
#! /bin/ksh
#  Script: until.scr

if [[-r $1]]
then
       :
else
```

```
        print "File $1 is not available.Waiting \c"
        until [[-r $1]]
              do
                sleep 5
                print ". \c"
              done
    fi
    print $1 "is available for processing"
```

b. 列表控制循环

列表控制循环中有一个控制列表。列表中元素个数控制循环次数。

- for-in 循环

  for-in 循环列表可以是任何类型的字符串。

```
# more loopfor.scr
#! /bin/ksh
# Script:loopfor.scr

for i in 1 2 3 4 5
do
      print $i Hello
done
# ./loopfor.scr
1 Hello
2 Hello
3 Hello
4 Hello
5 Hello
```

- select 循环

  select 循环是专为创建菜单设计的特殊循环。菜单是在显示器中的选项列表。用户选择菜单选项之一，然后用此脚本处理。select 循环个是与 for-in 循环相似。以关键字 select 开头，后跟变量和字符串列表。

```
# more selectOne.scr
#! /bin/ksh
#  Script:selectOne.scr

clear
select choice in month year quit
do
      case $choice in
      month) cal;;
      year)  yr=$(date "+%Y")
             cal $yr;;
      quit)  print "Hope you found your date"
             exit;;
```

```
        *)    print "Sorry,I don't understand your answer"
        esac
done
# ./select.scr
1) month
2) year
3) quit
#? 1
   2006 年 12 月
日 一 二 三 四 五 六
             1  2
 3  4  5  6  7  8  9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
31
#?
```

用户实际的响应保存在 shell 变量 REPLY，对应于用户响应列出的文件保存在 select 命令指定的变量中。

2. 后台循环

当循环有多次迭代时，它应该在后台执行。要在后台执行循环，再循环结尾处的单词 done 后面颊上一个&符号。

3. 循环重定向

来自文件的输入可以重定向到循环，循环的输出可以重定向到文件。

a. 输入重定向

要从文件把输入重定向到循环，只需要把重定向放在循环语句结尾即可。

b. 输出重定向

通过在循环结尾加上重定向，可以把循环的输出重定向到文件。只有 3 个循环可以使用输出重定向，分别是 while、until、for-in。

c. 循环管道

循环输入和输出可以被管道传递。要把输入管道传送给循环，把管道放在循环语句之前。要把输出管道传递给另一个命令，把管道放在循环之后。

d. 输入管道

用管道把输入传送到循环，如果文件名存储在一个文件中，就可以 cat 此文件，并用管道把它传送给循环。

e. 输出管道

要把一个循环输出管道传送给另一个命令，需要把 pipe 命令放在循环的结尾

f. 其他循环控制语句

其他两个与循环相关的语句：break 和 continue。

break: break 语句立即从循环中退出（但不从脚本中退出）。继续循环之后的第 1 个命令。break 语句通常是选择语句（如：if-then-else 或 case）中的动作

continue:continue 语句使循环忽略主体命令的其余部分，立即把控制转换到测试命令。如果循环还没有结束，开始下一次迭代。

五、 特殊参数和变量

1. 特殊参数

a. 脚本名称（$0）

脚本名称参数（$0）保存脚本的名称。通常一个脚本调用其他脚本时，此脚本名称参数可以传递给被调用的脚本，使它知道谁在调用它。另一个作用，当脚本需要发布一个错误消息时，可以将脚本名称包括在消息之中。

b. 参数个数（$#）

此参数保存传递给脚本的参数个数。脚本可以按几种可编程方式使用此参数，用用为$#。

c. 全部参数（$*和$@）

有两个特殊参数把 9 个位置参数合并为一个字符串。他们可以带引号，也可以不带引号。

- 所有参数不带引号

  当使用$*和$@不带引号时，即创建一个参数列表。注意：在输入参数时，用引号括起来的两个单词，在输出时也被当作两个单独的词。不管使用哪个全部参数，都会得到相同的结果。

- 带引号的全部参数

  当全部参数标记放在引号内时，其结果将不同。带引号的字符串标记（$*）表示把所有参数合并成一个单字符串。当用作列表时，列表只有一个成员——包含所有参数的字符串。引用列表标记（$@）创建一个字符串列表，其中每个参数是一个单独字符串。

d. 退出状态变量（#? ）

2. 特殊变量

内部字段分隔符（IFS）

内部字段分隔符变量保存 shell 命令使用的标记，把一个字符串分解成子字符串，如单词。默认标记是 3 个空白标记：空格、制表符和换行符。内部字段分隔符普遍用法之一是把 read 字符串分解成单独得单词。

```
# more IFS.scr
#! /bin/ksh
#  Script:IFS.scr

IFS=:    #Set IFS for passwd

#Read and process one line at a time.sk1 etc skip fields
#Find and print login ID and user name
while read id sk1 sk2 sk3 user sk4
do
  if [[ $id=$1]]
    then
      print $id "belongs to" $user
      exit 0
  fi
done </etc/passwd

print $1 not found
exit 1
```

以上代码，运行时报错，未查出原因。

六、 改变位置参数

只有使用 set 命令才能在脚本内部改变位置参数：不能对它们直接复制。set 命令分解输入字符串，并把字符串的每个独立部分放在不同的位置中（最多 9 个）。如果内部字段分隔符设置为默认，set 分析单词。可以把 IFS 设置成任意的标记并据此使用 set 分析数据。

举例如下：需要把日期保存在打印报告中，就可以使用 set 命令把 date 命令输出赋值给位置参数，这样就可以用日期打印了。

```
# more getDate.scr
#! /bin/ksh
# Script:getDate.scr

set $(date)    #assigns date fields to paramenters 1..6
               #$1:alpha day (ddd)    $2:alpha month(mmm)
               #$3:dat of month       $4:time(hh:mm:ss)
               #$5:time zone          $6:year(yyyy)
print "Complete date is:" $*
print
today="$2,$3,$6"
print "Today's date is:" $today
# ./getDate.scr
Complete date is: Thu Dec 14 00:45:05 BEIST 2006

Today's date is: Dec,14,2006
```

脚本分析：为了把 date 命令的输出转化为字符串以便保存，使用命令替代（第 4 行），要显示 date 命令的输出，使用＄*特殊处理参数。如果有其他不应该显示的参数，可以对日期使用单个参数（＄1…＄6）。最后，注意使用注释在每个参数中注解日期格式。

shift 命令

shift 命令是 shell 脚本中一个十分有用的命令。shift 命令把参数中的值移至参数列表的开头。要理解 shift 命令，把参数当作列表而不是单个的变量。当移位时，把每个参数向列表左边移。因此，如果移动 3 个位置，第 4 个参数将在第 1 个位置，第 5 个参数将在第 2 个位置。注意特殊参数不参与移位，只有当移位发生时改变其内容。如果 shift 没有参数默认移动 1 位。

示例如下：只有前 9 个参数可以按名称引用（$1…$9）。要处理其余参数，必须使用 shift 命令。示例通过循环使用 shift 只打印第 1 个参数（$1），示范 shift 操作。注意使用参数的个数（$#）作为循环控制。每个参数移出组后，shift 将参数总数减 1。

```
# more shiftAll.scr
#! /bin/ksh
# Script:shiftAll.scr
# shift and print all parameters

count=0
while (($#>0))
do
```

```
        ((count=count+1))
        print "$1 \c"
        shift
done
print "\n"
print "There are now" $# "parameters"
print "End of script"

# ./shiftAll.scr s f g r w y u d e h r s w t fd dingding
s f g r w y u d e h r s w t fd dingding

There are now 0 parameters
End of script
```

七、 参数有效性验证

1. 参数个数有效性验证

包含参数的脚本第一段代码应该对参数的个数进行有效性验证，有些脚本使用固定的参数个数；其他脚本石油可变的参数个数，即使参数个数可变，通常也需要一个最小数，固定和可变数量的参数都用参数个数参数（$#）来确认

2. 参数类型有效性验证

脚本需要验证每个参数类型的正确性。虽然所有参数都是作为字符串来传递的，但字符串内容可以是数字、文件名或任何其他可验证的类型。

a. 数字的有效性验证

使用 let 命令对参数进行数学计算，通过参数$?判断处理是否成功，如成功则参数为数字类型

b. 文件类型验证

如果输入参数是文件，可以检验文件的存在性和可读性。输出文件不用验证。

c. 其他验证

八、 调试脚本

在 Korn shell 中有两个选项用于帮助脚本调试的：冗长（verbose）选项和执行跟踪(xtrace)选项。

verbose 选项回显每个语法正确的语句，如果句法错误显示错误消息。如果脚本有输出，就产生输出。

xtrace 选项在每个命令执行前打印命令，并在前面加上加号（+），还替代语句中每个变量访问的值。

有两种方法使用这两个选项：

- 在脚本中使用 set 命令。如：set –o verbose
- 在命令执行的命令行中包含调试选项。如：ksh –o xtrace debugOptions.scr