Linux操作系统

何青

Unix操作系统的历史

二十世纪六十年代，贝尔实验室派出几个研究人员到MIT参加一个被称为“Multics”的操作系统的开发工作。 Multics是早期的分时操作系统，它实现了当代多任务操作系统的大多数思想。遗憾的是Multics过于复杂和笨拙， 在当时的条件下难以实现。 到了二十世纪六十年代末，贝尔实验室决定不再继续从事 Multics 的研究工作，并从MIT撤回了这批研究人员。无法再使用Multics的这批研究人员没有现代操作系统可以利用，于是他们决定自己建立一个操作系统。 最初的设计者是Ken Thompson和Dennis Ritchie，他们找到了一台废弃的DEC PDP-7开始了工作。最初的Unix系统就是这样产生的。

1973年，Ken Thompson和Dennis Ritchie开发了C语言，并用C语言重新开发了Unix系统的核心。但是，由于AT&T公司无法进入计算机市场，无法将Unix作为商品出售， 因此只能自己使用和发放到大学供研究使用。

Unix操作系统的版本

Bell 实验室继续进行 UNIX 的开发工作，发行了几个版本的 UNIX：System V 第一版（SVR1）、SVR2、SVR3、SVR4 和后来的 System IIV。

在维护和发展 UNIX 过程中，Bell 实验室将 UNIX 的源代码向教育界公开。许多学校接受了 UNIX 的拷贝，并向操作系统增加了许多新特性，California 大学的 Berkeley 分校就是其中一个。最终，该学校发行了自己的 UNIX 版本，称为 Berkeley Software Distribution（BSD）UNIX。BSD UNIX 使用最广泛的版本是 4.3 和 4.4（称为 4.4BSD）。

到 4.4BSD UNIX 分布时，这个版本中已经很少有最初 Bell 实验室的 UNIX 代码了。不久，几个小组相继编写了新的代码，替换掉 Bell 实验室代码的剩余那部分，使 BSD UNIX 适合在 Intel 386 处理器上运行。这导致了用于 Intel PC 的免费的 BSD UNIX 的 FreeBSD 和 NetBSD 版本的出现。

GNU计划

1983年，为了反对软件所有权私有化的趋势，Stallman 建立了GNU计划来推进免费软件模型。
GNU 是 GNU Is Not UNIX 的递归缩写，是自由软件基金会的一个项目，该项目的目标是开发一个自由的 UNIX 版本，这一 UNIX 版本称为 HURD。尽管 HURD 尚未完成，但 GNU 项目已经开发了许多高质量的编程工具，包括 emacs 编辑器、著名的 GNU C 和 C++ 编译器（gcc 和 g++），这些编译器可以在任何计算机系统上运行。所有的 GNU 软件和派生工作均适用 GNU 通用公共许可证，即 GPL。GPL 允许软件作者拥有软件版权，但授予其他任何人以合法复制、发行和修改软件的权利。

Minix操作系统

在AT&T发布版本7时，它开始认识到Unix的商业价值，于是发布的版本7许可证禁止在课程中研究其源代码。许多学校为了遵守该规定，就在课程中略去Unix的内容而只讲操作系统理论。

1987年，Andrew S. Tanenbaum决定编写一个在用户看来与Unix完全兼容，然而内核全新的操作系统MINIX。通过它读者可研究Unix系统的内部运作方式。

在MINUX发布后不久，便出现了一个面向它的USENET新闻组，在数周之内便有多达40000个用户订阅该新闻组。其中大多数人都想向MINIX 加入一些新特性以使之更大、更有用。然而Tanenbaum在几年内一直坚持不采纳这些建议，目的是使MINIX保持足够的短小精悍，以便于学生理解。

Linux操作系统

芬兰的一个大学生Linus Torvalds决定编写一个类似于MINIX系统的操作系统，但是他特征繁多、面向实用而非教学，这就是Linux。它是符合POSIX规范的操作系统。

- 1990, Linus Torvalds 首次接触 MINIX.

Linux操作系统

芬兰的一个大学生Linus Torvalds决定编写一个类似于MINIX系统的操作系统，但是他特征繁多、面向实用而非教学，这就是Linux。它是符合POSIX规范的操作系统。

- 1990, Linus Torvalds 首次接触 MINIX.
- 1991 中, Linus Torvalds 开始在 MINIX 上编写各种驱动程序等操作系统内核组件.

Linux操作系统

芬兰的一个大学生Linus Torvalds决定编写一个类似于MINIX系统的操作系统，但是他特征繁多、面向实用而非教学，这就是Linux。它是符合POSIX规范的操作系统。

- 1990, Linus Torvalds 首次接触 MINIX.
- 1991 中, Linus Torvalds 开始在 MINIX 上编写各种驱动程序等操作系统内核组件.
- 1991 底, Linus Torvalds 公开了 Linux 内核.

Linux操作系统

芬兰的一个大学生Linus Torvalds决定编写一个类似于MINIX系统的操作系统，但是他特征繁多、面向实用而非教学，这就是Linux。它是符合POSIX规范的操作系统。

- 1990, Linus Torvalds 首次接触 MINIX.
- 1991 中, Linus Torvalds 开始在 MINIX 上编写各种驱动程序等操作系统内核组件.
- 1991 底, Linus Torvalds 公开了 Linux 内核.
- 1993, Linux 1.0 版发行，Linux 转向 GPL 版权协议.

Linux操作系统

芬兰的一个大学生Linus Torvalds决定编写一个类似于MINIX系统的操作系统，但是他特征繁多、面向实用而非教学，这就是Linux。它是符合POSIX规范的操作系统。

- 1990, Linus Torvalds 首次接触 MINIX.
- 1991 中, Linus Torvalds 开始在 MINIX 上编写各种驱动程序等操作系统内核组件.
- 1991 底, Linus Torvalds 公开了 Linux 内核.
- 1993, Linux 1.0 版发行，Linux 转向 GPL 版权协议.
- 1994, Linux 的第一个商业发行版 Slackware 问世.

Linux操作系统

芬兰的一个大学生Linus Torvalds决定编写一个类似于MINIX系统的操作系统，但是他特征繁多、面向实用而非教学，这就是Linux。它是符合POSIX规范的操作系统。

- 1990, Linus Torvalds 首次接触 MINIX.
- 1991 中, Linus Torvalds 开始在 MINIX 上编写各种驱动程序等操作系统内核组件.
- 1991 底, Linus Torvalds 公开了 Linux 内核.
- 1993, Linux 1.0 版发行，Linux 转向 GPL 版权协议.
- 1994, Linux 的第一个商业发行版 Slackware 问世.
- 1996, 美国国家标准技术局的计算机系统实验室确认 Linux 版本 1.2.13（由 Open Linux 公司打包）符合 POSIX 标准.

登录

要想使用Linux操作系统，每个用户必须建立一个账号，设置个人的密码。通过各种方式登录 到Linux系统中。

登录方式:

- 主控台登录
- 远程登录：telnet
- Secure Shell登录：ssh

密码的设置:

用户的密码通常要求是：6-8个字符或数字，最好加一些特殊符号。尽量避免使用用户名以及某些个人信息作为密码，也不要使用字典中的常用词。

用户登录系统后，可以使用passwd命令修改密码。

目录操作:

- 列目录: `ls -l`

目录操作:

- 列目录: `ls -l`
- 改变目录: `cd`

目录操作:

- 列目录: `ls -l`
- 改变目录: `cd`
- 建立目录: `mkdir`

目录操作:

- 列目录: ls -l
- 改变目录: cd
- 建立目录: mkdir
- 目录的层次结构:
  - 根目录: /
  - 用户目录: /home
  - 常用命令目录: /bin /usr/bin /sbin /usr/sbin
  - 管理目录: /etc
  - 日志目录: /var
  - 设备目录: /dev

权限

Linux是一个多用户操作系统，它允许多个用户同时登录和工作。对文件和目录来讲，每个文件和目录都有一组权限标志和它们结合在一起。
对每个文件都有4类不同的用户。每类用户各有一组读、写和执行的访问权限。4类用户是:

- root:系统管理员
- owner:拥有文件的用户
- group:同组的用户
- other:其他用户

在目录结构中，每个文件和目录都有一组9位的权限位分别给文件拥有者，用户组和其他用户指定对文件和目录的读、写和执行权限。

改变权限

对于文件的拥有者，有权力改变文件的权限。

- 命令：chmod mode files

改变权限

对于文件的拥有者，有权力改变文件的权限。

- 命令：chmod mode files
- 其中mode指定新的权限位值，files则是要改变权限位设置的文件的清单。

改变权限

对于文件的拥有者，有权力改变文件的权限。

- 命令：chmod mode files
- 其中mode指定新的权限位值，files则是要改变权限位设置的文件的清单。
- Mode参数可以用两种不同方式指定，可以用符号表示也可以用八进制的位表示

文件系统

Linux使用树状的文件系统。每个系统都有一个根目录，从这 里开始可以遍历整个系统中的所有文件，它没有驱动器和盘符的概念。Linux系统将磁盘 划分为几个分区，每个分区都是一个文件系统，有它自己的目录结构。Linux系统通过 mount命令将他们装配起来，形成了一个无缝的整体。

在建立文件系统时，对存在磁盘分区中的文件，都给它分配一个号码，称为索引结点号(inode number)。 它实际是存在盘上的一个数组的入口的索引号。数组的每个元素是一个索引结点，它保存了一个文件的管理信息。

目录则是将文件名称和它的索引结点结合在一起的一张表，目录中每一个文件名称和索引结点号称为一个连接(link)

文件操作

- 拷贝文件：cp

文件操作

- 拷贝文件：cp
- 移动文件：mv

文件操作

- 拷贝文件：cp
- 移动文件：mv
- 删除文件：rm

文件操作

- 拷贝文件：cp
- 移动文件：mv
- 删除文件：rm
- 符号链接：ln

文件操作

- 拷贝文件：cp
- 移动文件：mv
- 删除文件：rm
- 符号链接：ln
- 察看文件内容：more

文件操作

- 拷贝文件：cp
- 移动文件：mv
- 删除文件：rm
- 符号链接：ln
- 察看文件内容：more
- 察看文件类型：file

系统帮助

man，是 UNIX 系统手册的电子版本。根据习惯，UNIX系统手册通常分为不 同的部分（或小节，即section），每个小节阐述不同的系统内容。目前的小节划分如下:

1. 命令: 普通用户命令
2. 系统调用: 内核接口
3. 函数库调用: 普通函数库中的函数
4. 特殊文件: /dev 目录中的特殊文件
5. 文件格式和约定: /etc/passwd 等文件的格式
6. 游戏。
7. 杂项和约定: 标准文件系统布局、手册页结构等杂项内容
8. 系统管理命令。
9. 内核例程: 非标准的手册小节。便于 Linux 内核的开发而包含

手册页一般保存在 /usr/share/man 目录下，其中每个子目录（如 man1, man2, ..., manl, mann）包含不同的手册小节。使用 man 命令查看手册页。

帮助系统info

Linux 中的大多数软件开发工具都是来自自由软件基金会的 GNU 项目，这些工具软件件的在线文档都以 info 文件的形式存在。info 程序是 GNU 的超文本帮助系统。 info 文档一般保存在 /usr/share/info 目录下，使用 info 命令查看 info 文档。
info 帮助系统的初始屏幕显示了一个主题目录，你可以将光标移动到带有 * 的主题菜单上面，然后按回车键 进入该主题，也可以键入 m，后跟主题菜单的名称而进入该主题。例如，你可以键入 m，然后再键入 gcc 而进 进入 gcc 主题中。

info的命令键

如果你要在主题之间跳转，则必须记住如下的几个命令键:

- n: 跳转到该节点的下一个节点;

info的命令键

如果你要在主题之间跳转，则必须记住如下的几个命令键:

- n: 跳转到该节点的下一个节点;
- p: 跳转到该节点的上一个节点;

info的命令键

如果你要在主题之间跳转，则必须记住如下的几个命令键:

- n: 跳转到该节点的下一个节点;
- p: 跳转到该节点的上一个节点;
- m: 指定菜单名而选择另外一个节点;

info的命令键

如果你要在主题之间跳转，则必须记住如下的几个命令键:

- n: 跳转到该节点的下一个节点;
- p: 跳转到该节点的上一个节点;
- m: 指定菜单名而选择另外一个节点;
- f: 进入交叉引用主题;

info的命令键

如果你要在主题之间跳转，则必须记住如下的几个命令键:

- n: 跳转到该节点的下一个节点;
- p: 跳转到该节点的上一个节点;
- m: 指定菜单名而选择另外一个节点;
- f: 进入交叉引用主题;
- l: 进入该窗口中的最后一个节点;

info的命令键

如果你要在主题之间跳转，则必须记住如下的几个命令键:

- n: 跳转到该节点的下一个节点;
- p: 跳转到该节点的上一个节点;
- m: 指定菜单名而选择另外一个节点;
- f: 进入交叉引用主题;
- l: 进入该窗口中的最后一个节点;
- TAB: 跳转到该窗口的下一个超文本链接;

info的命令键

如果你要在主题之间跳转，则必须记住如下的几个命令键:

- n: 跳转到该节点的下一个节点;
- p: 跳转到该节点的上一个节点;
- m: 指定菜单名而选择另外一个节点;
- f: 进入交叉引用主题;
- l: 进入该窗口中的最后一个节点;
- TAB: 跳转到该窗口的下一个超文本链接;
- RET: 进入光标处的超文本链接;

info的命令键

如果你要在主题之间跳转，则必须记住如下的几个命令键:

- n: 跳转到该节点的下一个节点;
- p: 跳转到该节点的上一个节点;
- m: 指定菜单名而选择另外一个节点;
- f: 进入交叉引用主题;
- l: 进入该窗口中的最后一个节点;
- TAB: 跳转到该窗口的下一个超文本链接;
- RET: 进入光标处的超文本链接;
- u: 转到上一级主题;

info的命令键

如果你要在主题之间跳转，则必须记住如下的几个命令键:

- n: 跳转到该节点的下一个节点;
- p: 跳转到该节点的上一个节点;
- m: 指定菜单名而选择另外一个节点;
- f: 进入交叉引用主题;
- l: 进入该窗口中的最后一个节点;
- TAB: 跳转到该窗口的下一个超文本链接;
- RET: 进入光标处的超文本链接;
- u: 转到上一级主题;
- d: 回到 info 的初始节点目录;

info的命令键

如果你要在主题之间跳转，则必须记住如下的几个命令键:

- n: 跳转到该节点的下一个节点;
- p: 跳转到该节点的上一个节点;
- m: 指定菜单名而选择另外一个节点;
- f: 进入交叉引用主题;
- l: 进入该窗口中的最后一个节点;
- TAB: 跳转到该窗口的下一个超文本链接;
- RET: 进入光标处的超文本链接;
- u: 转到上一级主题;
- d: 回到 info 的初始节点目录;
- h: 调出 info 教程;

info的命令键

如果你要在主题之间跳转，则必须记住如下的几个命令键:

- n: 跳转到该节点的下一个节点;
- p: 跳转到该节点的上一个节点;
- m: 指定菜单名而选择另外一个节点;
- f: 进入交叉引用主题;
- l: 进入该窗口中的最后一个节点;
- TAB: 跳转到该窗口的下一个超文本链接;
- RET: 进入光标处的超文本链接;
- u: 转到上一级主题;
- d: 回到 info 的初始节点目录;
- h: 调出 info 教程;
- q: 退出 info。

Linux操作系统的编辑器

在Linux操作系统上有许多编辑工具，最常用的编辑器是vi和emacs。其中emacs是GNU系列的第一个软件，它是用Lisp语言编写的。

emacs不仅是一个编辑器，而且还是一个完整的工作环境。在emacs内，你不仅可以创建、修改文本文件，还可以做许多其它工作。例如：玩游戏、收发邮件、参加讨论组、编译和修改LaTeX文档，还可以在emacs中开发各种程序，它的作用就相当于一个集成环境。

在标准的Unix系统中，最常用的编辑器是vi，在我们的课程中，重点介绍vi的使用方法。

vi的历史

Unix的第一个编辑器被称为ed，它是一种面向行的编辑器，或叫行编辑器。文本中的行都被编了号，用户可以以行号输入命令。后来，U. C. Berkeley的Bill Joy又开发出一种功能强大的行编辑器，称为ex，ex比ed更易于理解且功能更强。Joy为ex编写了面向屏幕 的界面，称为vi (visual editor)，vi支持ex的所有命令，且具有自己的专用命令以及充分利用全屏幕的某些约定。vi成为了Unix系统的标准 编辑器。所有的Unix操作系统都提供vi这个编辑器。

vi 的工作模式

Vi 在初始启动后首先进入编辑模式，这时用户可以利用一些预先定义的按键来移动光标、删除文字、复制或粘贴文字等。这些按键均是普通的字符，例如 l 是向右移动光标，相当于向右箭头键，k 是向下移动光标，相当于向下箭头键。在编辑模式下，用户还可以利用一些特殊按键选定文字，然后再进行删除、或复制等操作。

当用户在编辑模式下键入 i, a, o 等命令之后，可进入插入模式；键入：可进入命名模式。在插入模式下，用户随后输入的，除 Esc 之外的任何字符均将被看成是插入到编辑缓冲区中的字符。按 Esc 之后，从插入模式切换到编辑模式。

在命令模式，Vi 将把光标挪到屏幕的最下方，并在第一个字符的位置显示一个：（冒号）。这时，用户就可以键入一些命令。这些命令可用来保存文件、读取文件内容、执行 Shell 命令、设置 Vi 参数、以正则表达式的方式查找字符串或替换字符串等。

vi的模式切换及存盘

当我们进入vi时，我们就处于编辑模式。在编辑模式下，使用i, a, o 等命令之后，可进入插入模式。在插入模式下可以输入 文本。使用ESC健可以回到编辑模式。在编辑模式下键入: (冒号)，就进入了命令模式。

vi的存盘:

:wq	存盘并退出vi
:q	在未作修改的情况下退出
:q!	放弃所有修改，退出编辑程序
ZZ	存盘并退出vi

编辑模式

当我们进入vi时，我们就处于编辑模式。在编辑模式下，可以进行的操作有:

- 移动光标
- 替换和删除
- 粘贴和复制
- 搜索字符串
- 撤销和重复

要对正文内容进行修改，首先必须把光标移动到指定位置。移动光标的最简单的方式是按键盘的上、下、左、 右箭头键。除了这种最原始的方法之外，还可以利用 vi 提供的众多字符组合键，迅速到达指定的行或列，实现定位。

移动光标

k、j、h、l　功能分别等同于上、下、左、右箭头键

移动光标

k、j、h、l	功能分别等同于上、下、左、右箭头键
Ctrl+b	在文件中向上移动一页（相当于 PageUp 键）
Ctrl+f	在文件中向下移动一页（相当于 PageDown 键）

移动光标

k、j、h、l	功能分别等同于上、下、左、右箭头键
Ctrl+b	在文件中向上移动一页（相当于 PageUp 键）
Ctrl+f	在文件中向下移动一页（相当于 PageDown 键）
H	将光标移到屏幕的最上行（Highest）
nH	将光标移到屏幕的第 n 行
M	将光标移到屏幕的中间（Middle）
L	将光标移到屏幕的最下行（Lowest）
nL	将光标移到屏幕的倒数第 n 行

移动光标

k、j、h、l	功能分别等同于上、下、左、右箭头键
Ctrl+b	在文件中向上移动一页（相当于 PageUp 键）
Ctrl+f	在文件中向下移动一页（相当于 PageDown 键）
H	将光标移到屏幕的最上行（Highest）
nH	将光标移到屏幕的第 n 行
M	将光标移到屏幕的中间（Middle）
L	将光标移到屏幕的最下行（Lowest）
nL	将光标移到屏幕的倒数第 n 行
w	在指定行内右移光标，到下一个字的开头
e	在指定行内右移光标，到一个字的末尾
b	在指定行内左移光标，到前一个字的开头

移动光标

k、j、h、l	功能分别等同于上、下、左、右箭头键
Ctrl+b	在文件中向上移动一页（相当于 PageUp 键）
Ctrl+f	在文件中向下移动一页（相当于 PageDown 键）
H	将光标移到屏幕的最上行（Highest）
nH	将光标移到屏幕的第 n 行
M	将光标移到屏幕的中间（Middle）
L	将光标移到屏幕的最下行（Lowest）
nL	将光标移到屏幕的倒数第 n 行
w	在指定行内右移光标，到下一个字的开头
e	在指定行内右移光标，到一个字的末尾
b	在指定行内左移光标，到前一个字的开头
0	数字0，左移光标，到本行的开头
$	右移光标，到本行的末尾
^	移动光标，到本行的第一个非空字符

替换和删除

将光标定位于文件内指定位置后，可以用其他字符来替换光标所指向的字符，或从当前光标位置删除一个或多 个字符。例如:

rc	用 c 替换光标所指向的当前字符
nrc	用 c 替换光标所指向的前 n 个字符

替换和删除

将光标定位于文件内指定位置后，可以用其他字符来替换光标所指向的字符，或从当前光标位置删除一个或多 个字符。例如:

rc	用 c 替换光标所指向的当前字符
nrc	用 c 替换光标所指向的前 n 个字符
x	删除光标所指向的当前字符
nx	删除光标所指向的前 n 个字符

替换和删除

将光标定位于文件内指定位置后，可以用其他字符来替换光标所指向的字符，或从当前光标位置删除一个或多 个字符。例如:

rc	用 c 替换光标所指向的当前字符
nrc	用 c 替换光标所指向的前 n 个字符
x	删除光标所指向的当前字符
nx	删除光标所指向的前 n 个字符
dw	删除光标右侧的字
ndw	删除光标右侧的 n 个字
db	删除光标左侧的字
ndb	删除光标左侧的 n 个字

替换和删除

将光标定位于文件内指定位置后，可以用其他字符来替换光标所指向的字符，或从当前光标位置删除一个或多 个字符。例如:

rc	用 c 替换光标所指向的当前字符
nrc	用 c 替换光标所指向的前 n 个字符
x	删除光标所指向的当前字符
nx	删除光标所指向的前 n 个字符
dw	删除光标右侧的字
ndw	删除光标右侧的 n 个字
db	删除光标左侧的字
ndb	删除光标左侧的 n 个字
dd	删除光标所在行，并去除空隙
ndd	删除 n 行内容，并去除空隙

粘贴和复制

从正文中删除的内容（如字符、字或行）并没有真正丢失，而是被剪切并复制到了一个内存缓冲区中。用户可 将其粘贴到正文中的指定位置。完成这一操作的命令是:

p　小写字母 p，将缓冲区的内容粘贴到光标的后面

P　大写字母 P，将缓冲区的内容粘贴到光标的前面

如果缓冲区的内容是字符或字，直接粘贴在光标的前面或后面;如果缓冲区的内容为整行正文，则粘贴在当前 光标所在行的上一行或下一行。

注意上述两个命令中字母的大小写。vi 编辑器经常以一对大、小写字母（如 p 和 P）来提供一对相似的功能 。通常，小写命令在光标的后面进行操作，大写命令在光标的前面进行操作。

有时需要复制一段正文到新位置，同时保留原有位置的内容。这种情况下，首先应当把指定内容复制（而不是 剪切）到内存缓冲区。完成这一操作的命令是:

yy　复制当前行到内存缓冲区

搜索字符串

和许多先进的编辑器一样，vi 提供了强大的字符串搜索功能。要查找文件中指定字或短语出现的位置，可以 用 vi 直接进行搜索，而不必以手工方式进行。搜索方法是：键入字符 / ，后面跟以要搜索的字符串，然后 按回车键。编辑程序执行正向搜索（即朝文件末尾方向），并在找到指定字符串后，将光标停到该字符串的 开头；键入 n 命令可以继续执行搜索，找出这一字符串下次出现的位置。用字符 ? 取代 / ，可以实现反向 搜索（朝文件开头方向）。例如：

/str1	正向搜索字符串 str1
n	继续搜索，找出 str1 字符串下次出现的位置
?str2	反向搜索字符串 str2

无论搜索方向如何，当到达文件末尾或开头时，搜索工作会循环到文件的另一端并继续执行。

撤销和重复

在编辑文档的过程中，为消除某个错误的编辑命令造成的后果，可以用撤消命令。另外，如果用户希望在新 的光标位置重复前面执行过的编辑命令，可用重复命令。

u　撤消前一条命令的结果

.　重复最后一条修改正文的命令

插入模式

在编辑模式下正确定位光标之后，可用以下命令切换到插入模式:

- i　在光标左侧输入正文
- a　在光标右侧输入正文
- o　在光标所在行的下一行增添新行

插入模式

在编辑模式下正确定位光标之后，可用以下命令切换到插入模式:

i	在光标左侧输入正文
a	在光标右侧输入正文
o	在光标所在行的下一行增添新行
O	在光标所在行的上一行增添新行
I	在光标所在行的开头输入正文
A	在光标所在行的末尾输入正文

退出插入模式的方法是，按 ESC 键

正文替换

上面介绍了几种切换到插入模式的简单方法。另外还有一些命令，它们允许在进入插入模式之前首先删去一段 正文，从而实现正文的替换。这些命令包括:

s　　用输入的正文替换光标右侧 n 个字符

正文替换

上面介绍了几种切换到插入模式的简单方法。另外还有一些命令，它们允许在进入插入模式之前首先删去一段 正文，从而实现正文的替换。这些命令包括:

s	用输入的正文替换光标右侧 n 个字符
cw	用输入的正文替换光标右侧的字
ncw	用输入的正文替换光标右侧的 n 个字

正文替换

上面介绍了几种切换到插入模式的简单方法。另外还有一些命令，它们允许在进入插入模式之前首先删去一段 正文，从而实现正文的替换。这些命令包括:

s	用输入的正文替换光标右侧 n 个字符
cw	用输入的正文替换光标右侧的字
ncw	用输入的正文替换光标右侧的 n 个字
cb	用输入的正文替换光标左侧的字
ncb	用输入的正文替换光标左侧的 n 个字

正文替换

上面介绍了几种切换到插入模式的简单方法。另外还有一些命令，它们允许在进入插入模式之前首先删去一段 正文，从而实现正文的替换。这些命令包括:

s	用输入的正文替换光标右侧 n 个字符
cw	用输入的正文替换光标右侧的字
ncw	用输入的正文替换光标右侧的 n 个字
cb	用输入的正文替换光标左侧的字
ncb	用输入的正文替换光标左侧的 n 个字
cd	用输入的正文替换光标的所在行
ncd	用输入的正文替换光标下面的 n 行

正文替换

上面介绍了几种切换到插入模式的简单方法。另外还有一些命令，它们允许在进入插入模式之前首先删去一段 正文，从而实现正文的替换。这些命令包括:

s	用输入的正文替换光标右侧 n 个字符
cw	用输入的正文替换光标右侧的字
ncw	用输入的正文替换光标右侧的 n 个字
cb	用输入的正文替换光标左侧的字
ncb	用输入的正文替换光标左侧的 n 个字
cd	用输入的正文替换光标的所在行
ncd	用输入的正文替换光标下面的 n 行
c$	用输入的正文替换从光标开始到本行末尾的所有字符
c0	用输入的正文替换从本行开头到光标的所有字符

命令模式

在 vi 的命令模式下，可以使用复杂的命令。在编辑模式下键入“:”，光标就跳到屏幕最后一行，并在那里显示冒号，此时已进入命令模式。命令模式又称“末行模式”，用户输入的内容均显示在屏幕 的最后一行，按回车键，vi 执行命令。
vi和ex实际上是同一程序的两种表现形式。在命令模式输入的命令实际上就是ex的命令。

行号与文件

编辑中的每一行正文都有自己的行号，用下列命令可以移动光标到指定行:　　: n　将光标移到第 n 行

命令模式下，可以规定命令操作的行号范围。数值用来指定绝对行号；字符“.”表示光标所在行的行号；字符 符“$”表示正文最后一行的行号；简单的表达式，例如“.+5”表示当前行往下的第 5 行。例如:

:345　　将光标移到第 345 行

行号与文件

编辑中的每一行正文都有自己的行号，用下列命令可以移动光标到指定行:　　: n　将光标移到第 n 行

命令模式下，可以规定命令操作的行号范围。数值用来指定绝对行号；字符“.”表示光标所在行的行号；字符 符“$”表示正文最后一行的行号；简单的表达式，例如“.+5”表示当前行往下的第 5 行。例如:

:345	将光标移到第 345 行
:345w file	将第 345 行写入 file 文件
:3,5w file	将第 3 行至第 5 行写入 file 文件
:1,.w file	将第 1 行至当前行写入 file 文件
:.,$w file	将当前行至最后一行写入 file 文件
:.,.+5w file	从当前行开始将 6 行内容写入 file 文件
:1,$w file	将所有内容写入 file 文件，相当于 :w file 命令

在命令模式下，允许从文件中读取正文，或将正文写入文件。例如:

:w	将编辑的内容写入原始文件，用来保存编辑的中间结果
:wq	将编辑的内容写入原始文件并退出编辑程序
:w file	将编辑的内容写入 file 文件，保持原有文件的内容不变
:a,bw file	将第 a 行至第 b 行的内容写入 file 文件

在命令模式下，允许从文件中读取正文，或将正文写入文件。例如:

:w	将编辑的内容写入原始文件，用来保存编辑的中间结果
:wq	将编辑的内容写入原始文件并退出编辑程序
:w file	将编辑的内容写入 file 文件，保持原有文件的内容不变
:a,bw file	将第 a 行至第 b 行的内容写入 file 文件
:r file	读取 file 文件的内容，插入当前光标所在行的后面
:e file	编辑新文件 file 代替原有内容

在命令模式下，允许从文件中读取正文，或将正文写入文件。例如:

:w	将编辑的内容写入原始文件，用来保存编辑的中间结果
:wq	将编辑的内容写入原始文件并退出编辑程序
:w file	将编辑的内容写入 file 文件，保持原有文件的内容不变
:a,bw file	将第 a 行至第 b 行的内容写入 file 文件
:r file	读取 file 文件的内容，插入当前光标所在行的后面
:e file	编辑新文件 file 代替原有内容
:f file	将当前文件重命名为 file
:f	打印当前文件名称和状态， 如文件的行数、光标所在的行号等

字符串搜索

给出一个字符串，可以通过搜索该字符串到达指定行。如果希望进行正向搜索，将待搜索的字符串置于两个“/”之间；如果希望反向搜索，则将字符串放在两个“?”之间。例如:

:/str/	正向搜索，将光标移到下一个包含字符串 str 的行
:?str?	反向搜索，将光标移到上一个包含字符串 str 的行
:/str/w file	正向搜索，并将第一个包含字符串 str 的行写入 file 文件
:/str1/,/str2/w file	正向搜索，并将包含字符串 str1 的行至包含字符串 str2 的行写入 file 文件

正文替换

利用 :s 命令可以实现字符串的替换。具体的用法包括:

:s/str1/str2/	用字符串 str2 替换行中首次出现的字符串 str1
:s/str1/str2/g	用字符串 str2 替换行中所有出现的字符串 str1
:.,$ s/str1/str2/g	用字符串 str2 替换正文当前行到末尾所有出现的字符串 str1
:1,$ s/str1/str2/g	用字符串 str2 替换正文中所有出现的字符串 str1
:g/str1/s//str2/g	功能同上

拷贝、移动和删除正文

在命令模式下，同样可以拷贝、移动和删除正文中的内容。例如:

:1,4copy 10	将1到4行拷贝到第10行以后
:1,4m 10	将1到4行移动到第10行以后
:d	删除光标所在行
:3d	删除 3 行
:.,$d	删除当前行至正文的末尾
:/str1/,/str2/d	删除从字符串 str1 到 str2 的所有行

恢复文件

vi 在编辑某个文件时，会另外生成一个临时文件，这个文件的名称通常以 . 开头，并以 .swp 结尾。vi 在 正常退出时，该文件被删除，若意外退出，而没有保存文件的最新修改内容，则可以使用恢复命令:

:recover　恢复文件

也可以在启动 vi 时利用 -r 选项。

选项设置

为控制不同的编辑功能，vi 提供了很多内部选项。利用 :set 命令可以设置选项。基本语法为：　:set option　设置选项 option
常见的功能选项包括:

autoindent	设置该选项，则正文自动缩进
ignorecase	设置该选项，则忽略规则表达式中大小写字母的区别
number	设置该选项，则显示正文行号
ruler	设置该选项，则在屏幕底部显示光标所在行、列的位置
tabstop	设置按 Tab 键跳过的空格数。 例如 :set tabstop=n，n 默认值为 8
mk	将选项保存在当前目录的 .exrc 文件中

什么是Shell

Shell是一个命令处理器。它是一个对输入的命令进行读取并解释的程序。每次输入一个Unix命令时，都是由Shell解释其目的。
除了是一个命令解释器之外，Shell也是一种编程语言。用户可以写程序让Shell来解释，这种程序称为“脚本”。
对Shell更准确地解释是：Shell是用户与Unix系统之间的接口。
在Unix中有两种主要类型的Shell:

- Bourne Shell(包括sh, ksh和bash)

什么是Shell

Shell是一个命令处理器。它是一个对输入的命令进行读取并解释的程序。每次输入一个Unix命令时，都是由Shell解释其目的。
除了是一个命令解释器之外，Shell也是一种编程语言。用户可以写程序让Shell来解释，这种程序称为“脚本”。
对Shell更准确地解释是：Shell是用户与Unix系统之间的接口。
在Unix中有两种主要类型的Shell:

- Bourne Shell(包括sh, ksh和bash)
- C Shell(包括csh和tcsh)

在Linux系统下，最常用的shell是bash。Bash是由自由软件基金会作为Bourne shell的兼容程序开发的。它还容纳了其他shell程序的许多好的 特征，它是功能最全面的shell程序。几乎所有的Linux发布都提供Bash shell。

特殊字符

在Shell环境下特殊的字符有:

* ? [] {}\' " ‘ % ^ & $

对于这些特殊的字符，Shell有特殊的解释。下面我们就来分别解释他们的含义。

路径名通配符

特殊字符

在Shell环境下特殊的字符有:

* ? [] {}\' " ‘ % ^ & $

对于这些特殊的字符，Shell有特殊的解释。下面我们就来分别解释他们的含义。

路径名通配符

*	代表任意多个字符
?	代表任意单个字符
[str]	代表方括号内的任意单个字符
{ }	包括一系列用逗号分隔的字

输入输出重定向

在默认的情况下，Unix命令从键盘接受输入，并将命令的输出送到屏幕显示。它们的错误信息也送到屏幕 上显示。有时候，从文件接受输入或将结果送到文件中是很有用处的。

>	输出的重定向
<	输入的重定向
>>	附加的方式

在Unix系统下，我们用数字来代表标准输入、标准输出和标准出错。

0	标准输入(STDIN)
1	标准输出(STDOUT)
2	标准出错(STDERR)

管道

将一个程序的标准输出写到一个文件中去，再将这个文件的内容作为另一个命令的标准输入，等效于通过临时文件将两个命令结合在一起。这种情况在Unix系统中很普遍，使用Unix提供的一种功能，不需要用临时文件就可以将两条命令结合起来。这种功能称为管道(Pipe)
管道使用竖杠字符(—)作为重新定向操作符。最常用的操作是:

```
ls -l|more
```

管道行不限于将两条命令结合在一起。任何数目的命令都可以用管道操作符将相邻的一对命令结合在一起。

进程管理

在Unix系统下，无论何时执行任何一个命令，它都创建或启动一个新进程。操作系统使用一个叫 做pid或进程ID的5位数字来跟踪进程。 用户可以用ps命令查看当前正在运行什么进程和系统中的所有进程。在Linux系统中还可以用top命令来查看 当前进程的运行情况。

当用户启动一个进程时，有两种运行方式：前台和后台。

前台进程：缺省时，用户启动的每个进程都在前台运行，此时进程从键盘得到输入并把输出送到屏幕上。 这时我们必须等待一个命令执行完成后再执行下一个命令。

后台进程：如果一个命令会运行很长的时间，我们可以把它放到后台去运行。具体方法是在命令的后面加一个&符号。

作业控制

每次用bash执行一条命令时，他都给命令分配一个作业号(job number)，作业控制允许将进程挂起，在以后再来执行

ctrl-z挂起

jobs 可以显示当前作业的清单

如果我们要恢复进程的执行可以使用两个方法

fg　把命令回到前台执行

bg　把命令放到后台去执行

对进程进行管理: 杀死一个进程 kill -9 PID

在top命令下也可以给进程发送信号。k可以杀死某个进程。

历史表

在Bash环境下，可以用历史表(history list)来保存以前执行过的命令。历史表一般能保存1000行命令。 在我们退出登录时，bash自动将当前的历史表保存到一个文件中，文件的名字是： .bash_history。

使用history命令就可以将历史表中的记录列出来。历史表中的每一行称为一个事件(event)，行号称为事件号。 如果想重复执行历史表中的命令，可以使用历史替换操作符(!)和事件号。 如果你要重复最后一条命令，只要是用两个!!就可以了。使用键盘上的上下箭头也可以重复执行以前执行过的命令。

命令补全

bash还有另一项有用的功能，它可以为你补全命令行。在输入命令的任何时刻，可以按Tab键。 此时Shell将试图补全此时已经输入的部分命令。
如果你输入的字符串不足以使bash唯一地确定它应该使用的命令，它将发出警告声。再按一次Tab键将使 bash显示出可能补全的所有命令的清单。这样，就可以在命令中增加足够的字符，使得下次按Tab键时bash能克服多义性。
除了用这种方式补全命令名称外，当用文件名或目录名作为命令参数时，bash也能补全文件名称。

空间

在Linux系统下，free可以反映内存空间的使用情况。
观察硬盘空间的使用情况可以使用df命令。
在Linux系统下可以给df一些参数来显示不同的信息。
使用df可以显示系统总体的硬盘使用情况，要想了解某个用户的空间使用情况，需要使用du命令

du

du -m

du -sm

正则表达式

正则表达式在 shell、工具程序、Perl 语言中有非常重要的地位。正则表达式通过一些特殊符号表示特定的字符串模式。常见的特殊字符包括:

字符	功能
^	置于待搜索的字符串之前，匹配行首的字
$	置于待搜索的字符串之后，匹配行末的字
\<	匹配一个字的字头
\>	匹配一个字的字尾
.	匹配任意单个正文字符
[str]	匹配字符串 str 中的任意单个字符
[^str]	匹配不在字符串 str 中的任意单个字符
[a-c]	匹配从 a 到 c 之间的任一字符
*	匹配前一个字符的 0 次或多次出现
\	忽略特殊字符的特殊含义，将其看作普通字符

搜索

Unix系统提供一系列命令可以进行文本的搜索 最常用的命令是grep。其他命令例如： vi more man less sed awk perl也可以使用正则表达式对文本进行搜索。
grep的含义是：globbally regular expression print。
grep命令的基本语法是：

```
grep string files
```

其中string是一个正则表达式。如果string中包含特殊的字符，一般我们用单引号，将string扩起来，使Shell不理会它们。

扩充字符集

如果在grep命令中加-E选项，还可以使用一组扩充的特殊字符集:

字符	功能
+	重复匹配前一项 1 次以上
?	重复匹配前一项 0 次或 1 次
{j}	重复匹配前一项 j 次
{j, }	重复匹配前一项 j 次以上
{, k}	重复匹配前一项最多 k 次
{j, k}	重复匹配前一项 j 到 k 次
s|t	匹配 s 或 t 中的一项
(exp)	将表达式 exp 作为单项处理

grep的选项

grep还有许多其它选项，分别为：

grep -n	显示行号
grep -i	不区分大小写
grep -q	与其他命令一起使用时，抑制输出显示
grep -s	抑制文件的出错信息
grep -num	在每个匹配行的前后各显示num行

find命令

find命令的主要作用是对目录树进行彻底检查。 find命令的一般格式为： find pathname -expression

-name pattern　条件表达式，检查文件名是否和模式pattern相同

find命令

find命令的主要作用是对目录树进行彻底检查。 find命令的一般格式为： find pathname -expression

-name pattern	条件表达式，检查文件名是否和模式pattern相同
-group grp	条件表达式，检查文件是否有与grp相同的组名

find命令

find命令的主要作用是对目录树进行彻底检查。 find命令的一般格式为： find pathname -expression

-name pattern	条件表达式，检查文件名是否和模式pattern相同
-group grp	条件表达式，检查文件是否有与grp相同的组名
-user usr	条件表达式，检查文件是否有与usr相同的用户名

find命令

find命令的主要作用是对目录树进行彻底检查。 find命令的一般格式为： find pathname -expression

-name pattern	条件表达式，检查文件名是否和模式pattern相同
-group grp	条件表达式，检查文件是否有与grp相同的组名
-user usr	条件表达式，检查文件是否有与usr相同的用户名
-type t	条件表达式，检查文件的类型是否是t 。对目录，t值可以是d。对普通文件，t的值是f。对连接，t的值是l

find命令

find命令的主要作用是对目录树进行彻底检查。 find命令的一般格式为： find pathname -expression

-name pattern	条件表达式，检查文件名是否和模式pattern相同
-group grp	条件表达式，检查文件是否有与grp相同的组名
-user usr	条件表达式，检查文件是否有与usr相同的用户名
-type t	条件表达式，检查文件的类型是否是t 。对目录，t值可以是d。对普通文件，t的值是f。对连接，t的值是l
-mount	选项表达式，用来防止find命令的搜索范围超出当前文件系统（Linux支持）

find命令

find命令的主要作用是对目录树进行彻底检查。 find命令的一般格式为： find pathname -expression

-name pattern	条件表达式，检查文件名是否和模式pattern相同
-group grp	条件表达式，检查文件是否有与grp相同的组名
-user usr	条件表达式，检查文件是否有与usr相同的用户名
-type t	条件表达式，检查文件的类型是否是t 。对目录，t值可以是d。对普通文件，t的值是f。对连接，t的值是l
-mount	选项表达式，用来防止find命令的搜索范围超出当前文件系统（Linux支持）
-print	操作表达式，将查找到的文件名输出。

find命令

find命令的主要作用是对目录树进行彻底检查。 find命令的一般格式为： find pathname -expression

-name pattern	条件表达式，检查文件名是否和模式pattern相同
-group grp	条件表达式，检查文件是否有与grp相同的组名
-user usr	条件表达式，检查文件是否有与usr相同的用户名
-type t	条件表达式，检查文件的类型是否是t 。对目录，t值可以是d。对普通文件，t的值是f。对连接，t的值是l
-mount	选项表达式，用来防止find命令的搜索范围超出当前文件系统（Linux支持）
-print	操作表达式，将查找到的文件名输出。
-exec cmd	操作表达式，用来执行cmd命令。如果要将当前的文件名传送给命令，应该加标记，用分号来表示cmd的结束。

过滤器

过滤器：对于输入的文本进行某些处理后产生输出，对原来的文本文件没有影响。

1. 最简单的过滤器：cat

过滤器

过滤器：对于输入的文本进行某些处理后产生输出，对原来的文本文件没有影响。

1. 最简单的过滤器：cat
2. 只显示文本的最后几行：tail

过滤器

过滤器：对于输入的文本进行某些处理后产生输出，对原来的文本文件没有影响。

1. 最简单的过滤器：cat
2. 只显示文本的最后几行：tail
3. 只显示文本的前几行：head

过滤器

过滤器：对于输入的文本进行某些处理后产生输出，对原来的文本文件没有影响。

1. 最简单的过滤器：cat
2. 只显示文本的最后几行：tail
3. 只显示文本的前几行：head
4. 记录文件中的字符，词，行数：wc

排序命令

sort排序命令是一个过滤程序。它将输入的正文按照一定规则根据指定的字段排序。 默认的sort字段分隔符是空白字符。

sort命令的选项:

-b	不管排序键前面的空格字符
-f	不区分大小写
-n	将排序键作为数字而不是作为正文
-r	按从高到低的顺序排序
-o file	将输出送到file
-t s	用s代替空白字符作为字段分隔符
-k s1,s2	用s1字段到(s2-1)字段作为排序键

切割和粘连命令

cut命令提供了从它的输入行中提取特定列的方法，cut的选项有：

-b　按照字节分割
-c　按照字符分割
-d　指定分割符，省缺分割符是Tab
-f　字段

paste命令将字段接在一起

字符转换

tr命令从标准输入中读入正文，将转换结果送到标准输出。tr命令的基本形式是:

tr str1 str2

其中：str1和str2是字符串。在str1中出现的字符将被转换为str2中的对应的字符。str1和str2长度一般要求一样。如果str1比str2短，将str2后面的字符截去。如果str1比str2长，将str2用它的最后一个字符补充使它和str1等长。

tr的选项有:

-c　把所有不在str1的字符用str2进行转换

-s　将str2中的任何字符的多次连续出现压缩为一个字符

-d　删除str1中的字符

变量

在shell环境下可以定义任何变量,定义的方法如下

name=value

显示变量的值

echo $name

取消一个变量的值

unset name

当Shell在运行时，存在三种主要变量:

- 局部变量

变量

在shell环境下可以定义任何变量,定义的方法如下

name=value

显示变量的值

echo $name

取消一个变量的值

unset name

当Shell在运行时，存在三种主要变量:

- 局部变量
- 环境变量

变量

在shell环境下可以定义任何变量,定义的方法如下

name=value

显示变量的值

echo $name

取消一个变量的值

unset name

当Shell在运行时，存在三种主要变量:

- 局部变量
- 环境变量
- Shell变量

局部变量和环境变量

在任何时候，建立的变量只是当前Shell的局部变量。局部变量不能被shell运行的 其他命令或脚本所使用。如果想把某个变量提供给其他命令使用，可以用导出命 令export来实现。
环境变量是Shell的任何子进程都能使用的变量，许多程序要正确运行都需要使用环境变量。通常 Shell脚本只定义程序运行时所需要的环境变量。
export命令的作用是将局部变量放入环境中，以供其它脚本使用。

Shell变量

shell变量是Shell在初始化和使用过程中设置的变量

PS1	它保存shell命令行提示字符串。
PS2	它保存shell的第2提示字符串。当shell发现你的命令不全，还需要有更多输入时使用这个提示符
PWD	它保存当前工作目录的绝对路径名
UID	用户识别号
HOME	用户的起始目录
PATH	可执行命令的搜索路径

Shell脚本的参数

通常我们是在Shell脚本中定义变量，但是许多脚本需要外部的参数，Shell可以将送入的命令行自动分成单字，然后将这组子作为值赋予一组特殊的变量。 在脚本内部可以读取这些变量的值。
这些变量分别用$0, $1, $2等符号表示。
$0的值是命令行中的第一个字（即命令的名称）。$1的值是命令行的第一个参数，$2的值是命令行的第二个参数。 例如在下面的命令行中

```
test1 param1 param2
$0=test1
$1=param1
$2=param2
```

交互输入

在Shell脚本中可以用read命令从标准输入设备读入若干字符,并将它赋值给一个或多个变量. Read命令的最简单的用法是只有命令名,在这种情况下把输入的字符串赋值给REPLY变量.
如果在read命令后面指定变量名，输入将直接送给变量。
在read命令后面指定的变量名可以是多个. 如果词数比变量个数多，每个变量都得到相应的值，多余的词附加到最后一个变量中。

命令替换

如果我们需要把某个命令输出的结果赋值给变量，可以使用命令替换的方法

```
name=‘command‘
name=$(command)
```

另一条常用作命令替换的命令称为basename，这条命令用完整的文件路径名作为参数，它将路径名去掉只送回基本文件名，例如:

```
basefile=‘basename /usr/bin/man‘
echo $basefile
man
```

自动化状态变量

当执行任何Linux命令时，可能出现两种状态:命令可能无错误地顺利结束运行，也可能由于某种原因不能完成任务. 当命令运行结束时返回一个值，这个值称为命令的退出状态. 退出状态为0表示命令执行成功。 退出状态非0表示命令执行不成功。

$?	保存shell最后执行的命令的退出状态
$$	当前shell的PID
$#	保存传送给当前script的命令参数的数目
$*	保存传送给当前script的所有命令参数清单

流程控制

像其他高级程序设计语言一样，Shell提供用来控制程序执行流程的命令，包括一些选择和循环结构。用户可以用这些命令建立非常复杂的程序。
常用的流程控制语句包括:

- if命令
- test命令
- while命令
- until命令
- for循环
- case选择
- select循环

if命令

当shell执行命令或管道行，返回退出状态值，可以用if命令来测试返回值并决定下一步做什么。

```
if condition commands
then
true commands
else
false commands
fi
```

如果Shell成功的执行了条件命令(condition commands)，它的退出状态值用来决定执行条件为真是执行的命令(true commands)还是条件为假时执行的命令(false commands)。由于退出状态值为0表示成功地执行了命令，因此0作条件的真值。任何非0值作为条件的假值。

test命令

在实际使用if语句时，我们经常要检测各种条件语句，Shell提供了一个特殊的命令test来测试一系列条件然后返回相应的退出状态。test命令的格式为:

```
test expression
```

如果指定的表达式(expression)为真，返回0退出状态；如果是假，返回1退出状态。
test命令可以用来测试几种不同类型的表达式，包括:

- File tests 检查文件的特征
- String comparisons 字符串比较
- Numerical comparisons 数字比较

file test

在第一种情况下，给出一个命令行开关，后面跟单个参数，即文件名称。test命令根据开关指定的特征对文件进行检验，本根据检验结果给出退出状态。最常用的命令行开关有:

-d file	True if file exists and is a directory.
-e file	True if file exists.
-f file	True if file exists and is a regular file.
-r file	True if file exists and is readable.
-w file	True if file exists and is writable.
-x file	True if file exists and is executable.

test语句还有一个等价的写法是:

```
[ expression ]
```

字符串比较

字符串类型表达式可以用一个或两个字符串参数，它既可以是文字字符串也可以是Shell变量的内容。最常用的串表达式有:

-z string	True if string has zero length.
-n string	True if string has nonzero length.
string1 = string2	True if the strings are equal.
string1 != string2	True if the strings are not equal.

注意在进行字符串比较时，string1、string2与中间的等号之间必须有一个空格

数值的比较

数值表达式具有将字符串或变量的内容作为数值来处理的能力，并完成标准的数值比较。数值表达式的形式为：

int1 -eq int2	True if int1 equals int2.
int1 -ne int2	True if int1 is not equal to int2.
int1 -lt int2	True if int1 is less than int2.
int1 -le int2	True if int1 is less than or equal to int2.
int1 -gt int2	True if int1 is greater than int2.
int1 -ge int2	True if int1 is greater than or equal to int2.

test命令还可以使用逻辑操作符组合表达式：

expr1 -a expr2	True if both expr1 and expr2 are true.
expr1 -o expr2	True if either expr1 or expr2 is true.
! expr	True if expr is false.

算术计算

像test命令能将字符串作为数值比较一样，bash也能完成简单的算术运算，bash用下列方式建立算术表达式:

```
$[expression]
```

例如:

```
num1=2
num1=$[$num1*3+1]
echo $num1
7
```

while命令

while命令允许在Shell中插入条件循环，命令的一般格式为：

```
while condition
do
commands
done
```

其中condition可以是任何命令或管道行，它的退出状态将用来决定下一步操作。如果退出状态为0，就执行do和done之间的各条命令。接着转移到循环顶部，重新检查条件。如果退出状态为假，则跳过这些命令，执行done后面的命令。

until命令

until命令是另一种循环结构。它和while命令非常相似，它具有下面的一般结构:

```
until command
do
  commands
done
```

它和while命令的差别在于：while命令在条件为真时，继续执行循环。而until则相反，在条件为假时，继续执行循环，直到条件为真时停止循环。

for循环

for循环的一般格式是:

```
for variable in wordlist
do
    commands
done
```

基本概念是：从wordlist清单中依次取指定变量variable的每个值，然后用取得的值执行循环体内的命令。
有时for循环中的wordlist可以用特殊变量$*来代替，由于

```
for variable in $*
```

是经常用到的结构，它可以被缩写为

```
for variable
```

形式。由Shell命令自动补充in $*的部分

case选择

根据字符串或变量的值，从许多选像中选出一项，这是case命令要完成的任务。命令的一般形式为:

```
case string in
pattern1)
     commands1
     ;;
pattern2)
     commands2
     ;;
*)
      default commands
     ;;
esac
```

在执行case命令时，Shell将计算字符串string的值，然后将结果依次和模式pattern1, pattern2等进行比较，直到找到一个匹配的表达式为止。

case表达式可以用正则表达式给出，用*号作为case命令的最后表达式的含义是：它可以和任何字符串匹配。如果前面的比较全部失败，这就是默认的入口。

select循环

select循环提供了一种从用户可选项中创建编号菜单的简捷方式。当要求用户从一个选像清单中选择一项或多项时，就需要使用select循环。
select循环的基本语法为:

```
select name in wordlist
do
    list
done
```

这里name是变量名，wordlist是字符串序列，用户选择后要执行的命令集合由list指定。通常情况下list中应包含case命令

select循环

select循环的执行过程如下：

1. wordlist中的每一项都和一个数字一起显示

select循环

select循环的执行过程如下：

1. wordlist中的每一项都和一个数字一起显示
2. 显示一个提示符，通常为#？

select循环

select循环的执行过程如下：

1. wordlist中的每一项都和一个数字一起显示
2. 显示一个提示符，通常为#?
3. 当用户输入一个值时，$REPLY被赋予该值

select循环

select循环的执行过程如下：

1. wordlist中的每一项都和一个数字一起显示
2. 显示一个提示符，通常为#?
3. 当用户输入一个值时，$REPLY被赋予该值
4. 若$REPLY包含所显示的项，则name所指定的变量被赋值为wordlist中被选中的项，否则，wordlist中德项再次显示

select循环

select循环的执行过程如下：

1. wordlist中的每一项都和一个数字一起显示
2. 显示一个提示符，通常为#?
3. 当用户输入一个值时，$REPLY被赋予该值
4. 若$REPLY包含所显示的项，则name所指定的变量被赋值为wordlist中被选中的项，否则，wordlist中德项再次显示
5. 当作了有效选择后，执行相应的命令

select循环

select循环的执行过程如下：

1. wordlist中的每一项都和一个数字一起显示
2. 显示一个提示符，通常为#?
3. 当用户输入一个值时，$REPLY被赋予该值
4. 若$REPLY包含所显示的项，则name所指定的变量被赋值为wordlist中被选中的项，否则，wordlist中德项再次显示
5. 当作了有效选择后，执行相应的命令
6. 若相应的命令中没有使用循环控制机制(break)从select循环中退出，则整个过程从第一步重新开始

循环控制

有时我们需要停止循环或跳过循环的某些迭代，这就需要控制循环的命令:

- break
- continue

break命令可以从任何循环中退出。如果break位于嵌套的循环中，break命令通常是退出一层循环。
break命令也可以接收一个参数：一个大于等于1的整数，用于指出退出嵌套循环的层数。
continue命令的作用是：它从循环的当前迭代退出而不是整个循环。

在vi, ex等行编辑器中可以使用正则表达式过滤文本，查找相应的行，然后进行编辑操作。
sed(代表stream editor)是为执行脚本而创建的一个专用编辑器。sed是面向流机制的编辑器。sed命令并不改变输入的文件。它可以通过正则表达式过滤文本，同时对相应的行进行编辑操作。

在vi, ex等行编辑器中可以使用正则表达式过滤文本，查找相应的行，然后进行编辑操作。
sed(代表stream editor)是为执行脚本而创建的一个专用编辑器。sed是面向流机制的编辑器。sed命令并不改变输入的文件。它可以通过正则表达式过滤文本，同时对相应的行进行编辑操作。
awk和sed有一些相似之处:

- 激活语法相同
- 利用它们，用户可以指定为输入文件的每一行都执行的命令
- 为匹配模式使用正则表达式

awk和sed的激活语法为:

awk和sed的激活语法为:

command 'script' filename

awk和sed的激活语法为:

command 'script' filename

这里command是awk或sed, script是可以被awk或sed理解的命令清单， filename表示命令所作用的文件清单。

awk和sed的激活语法为:

command 'script' filename

这里command是awk或sed, script是可以被awk或sed理解的命令清单， filename表示命令所作用的文件清单。

为了避免Shell执行替换，要使用单引号将script引起来。script的真正内容对于awk和sed来说大大不同。

如果没有给出文件名，awk和sed都从STDIN中读入，这使得它们可用作其他命令的输出过滤器。

awk和sed的激活语法为:

command 'script' filename

这里command是awk或sed, script是可以被awk或sed理解的命令清单， filename表示命令所作用的文件清单。

为了避免Shell执行替换，要使用单引号将script引起来。script的真正内容对于awk和sed来说大大不同。

如果没有给出文件名，awk和sed都从STDIN中读入，这使得它们可用作其他命令的输出过滤器。

基本操作： 当awk和sed命令运行时，执行如下操作:

- 从输入文件中读取一行

awk和sed的激活语法为:

command 'script' filename

这里command是awk或sed, script是可以被awk或sed理解的命令清单， filename表示命令所作用的文件清单。

为了避免Shell执行替换，要使用单引号将script引起来。script的真正内容对于awk和sed来说大大不同。

如果没有给出文件名，awk和sed都从STDIN中读入，这使得它们可用作其他命令的输出过滤器。

基本操作： 当awk和sed命令运行时，执行如下操作:

- 从输入文件中读取一行
- 为该行做一个拷贝

awk和sed的激活语法为:

command 'script' filename

这里command是awk或sed, script是可以被awk或sed理解的命令清单， filename表示命令所作用的文件清单。

为了避免Shell执行替换，要使用单引号将script引起来。script的真正内容对于awk和sed来说大大不同。

如果没有给出文件名，awk和sed都从STDIN中读入，这使得它们可用作其他命令的输出过滤器。

基本操作： 当awk和sed命令运行时，执行如下操作:

- 从输入文件中读取一行
- 为该行做一个拷贝
- 在该行上执行所给的脚本script

awk和sed的激活语法为:

command 'script' filename

这里command是awk或sed, script是可以被awk或sed理解的命令清单， filename表示命令所作用的文件清单。

为了避免Shell执行替换，要使用单引号将script引起来。script的真正内容对于awk和sed来说大大不同。

如果没有给出文件名，awk和sed都从STDIN中读入，这使得它们可用作其他命令的输出过滤器。

基本操作： 当awk和sed命令运行时，执行如下操作:

- 从输入文件中读取一行
- 为该行做一个拷贝
- 在该行上执行所给的脚本script
- 为下一行重复第一步。

脚本结构及脚本的执行

awk和sed所指定的脚本script包含一行或多行记录，这些记录的格式为：/pattern/action

脚本结构及脚本的执行

awk和sed所指定的脚本script包含一行或多行记录，这些记录的格式为：/pattern/action
pattern为正则表达式，action表示一些动作

脚本结构及脚本的执行

awk和sed所指定的脚本script包含一行或多行记录，这些记录的格式为：/pattern/action
pattern为正则表达式，action表示一些动作
对于每条记录执行如下过程:

脚本结构及脚本的执行

awk和sed所指定的脚本script包含一行或多行记录，这些记录的格式为：/pattern/action
pattern为正则表达式，action表示一些动作
对于每条记录执行如下过程：

- 顺序搜索每个模式pattern直到发生一个匹配

脚本结构及脚本的执行

awk和sed所指定的脚本script包含一行或多行记录，这些记录的格式为：/pattern/action
pattern为正则表达式，action表示一些动作
对于每条记录执行如下过程:

- 顺序搜索每个模式pattern直到发生一个匹配
- 当发现匹配后执行相应的动作

脚本结构及脚本的执行

awk和sed所指定的脚本script包含一行或多行记录，这些记录的格式为：/pattern/action
pattern为正则表达式，action表示一些动作
对于每条记录执行如下过程:

- 顺序搜索每个模式pattern直到发生一个匹配
- 当发现匹配后执行相应的动作
- 当action执行完毕，到达下一个模式并重复第一步

sed的用法

sed 'script' files
script是一个或多个如下格式的命令:
/pattern/action
action:
p 打印该行
d 删除该行
y 字符转换
s/pattern1/pattern2/ 用第二种模式pattern2替换第一种模式pattern1
sed在省缺情况将每行都打印出来,可以用-n选项.

什么是awk

awk是一种编程语言，该语言可以基于模式搜索多个文件并改变带有这些文件的记录。awk这个名字来源于它的创建者们的姓的首字母。

当前，可以使用的awk有三个主要版本:

- 最初的awk
- 新版nawk
- GNU版的gawk

基本语法

awk的基本语法为:

awk 'script' files

这里files是一个或多个文件，script由一个或多个命令组成。命令的格式如下:

/pattern/{actions}

这里，pattern为一个正则表达式，actions是一个或多个命令，若去掉pattern，则awk为输入文件的每一行执行指定动作(actions)。

Linux 的编程常识

- 标准 (ANSI C, POSIX, SVID, XPG, ...)
- 函数库和系统调用
- 在线文档 (man, info, HOW-TO, ...)
- C 语言编程风格
- 库和头文件的保存位置
- 共享库及其相关配置

ANSI C

ANSI C: 这一标准是 ANSI（美国国家标准局）于 1989 年制定的 C 语言标准。 后来被ISO（国际 标准化组织）接受为标准，因此也称为 ISO C。
ANSI C的目标是为各种操作系统上的C程序提供可移植性保证，而不仅仅限于UNIX。 该标准不仅定义 了C编程语言的语发和语义，而且还定义了一个标准库。这个库可以根据头文件划分为15个部分，其中 包括: 字符类型 (ctype.h)、错误码 (errno.h)、 浮点常数 (float.h)、数学常数 (math.h)、 标准定义(stddef.h)、 标准I/O (stdio.h)、工具函数 (stdlib.h)、字符串操作 (string.h)、 时间和日期(time.h)、可变参数表 (stdarg.h)、信号 (signal.h)、 非局部跳转 (setjmp.h)、 本地信息 (local.h)、程序断言 (assert.h) 等等。

函数库和系统调用

众所周知，C 语言并没有为常见的操作，例如输入/输出、内存管理，字符串操作等提供内置的支持。 相反，这些功能一般由标准的“函数库”来提供。GNU 的C函数库，即glibc，是Linux上最重要的函数 库，它定义了ANSI C标准指定的所有的库函数，以及由POSIX或其他UNIX操作系统统变种指定的附加特色， 还包括有与GNU系统相关的扩展。目前，流行的Linux系统使用glibc 2.0以上的版本。

其他重要函数库

除 glibc 之外，流行的 Linux 发行版中还包含有一些其他的函数库，这些函数库具有重要地位，例如:

GNU Libtool: GNU Libtool 实际是一个脚本生成工具，它可以为软件包开发者提供一般性 的共享库支持。

CrackLib: CrackLib 为用户提供了一个 C 语言函数接口，利用这一函数，可避免用户选择 容易破解的密码。该函数库可在类似 passwd 的程序中使用。

LibGTop: LibGTop 是一个能够获取进程信息以及系统运行信息的函数库，这些信息包括: 系统的一般信息、SYS V IPC 限制、进程列表、进程信息、进程映射、文件系统使用信息等。

图形文件操作函数库: 包括 libungif、libtiff、libpng、Imlib 等，可分别用来操作 GIF、TIFF、PNG 以及其他一些格式图形文件。

系统调用

系统调用是操作系统提供给外部程序的接口。在C语言中，操作系统的系统调用通常通过函数调用的形式 完成，这是因为这些函数封装了系统调用的细节，将系统调用的入口、参数以及返回值用C语言的函数调 用过程实现。在Linux系统中，系统调用函数定义在glibc中。

库和头文件的保存位置

1. 函数库
   - /lib: 系统必备共享库
   - /usr/lib: 标准共享库和静态库
   - /usr/i486-linux-libc5/lib: libc5 兼容性函数库
   - /usr/X11R6/lib: X11R6 的函数库
   - /usr/local/lib: 本地函数库
2. 头文件
   - /usr/include: 系统头文件
   - /usr/local/include: 本地头文件

共享库及其相关配置

- /etc/ld.so.conf：包含共享库的搜索位置
- ldconfig：共享库管理工具，一般在更新了共享库之后要运行该命令
- ldd：可查看可执行文件所使用的共享库

GCC

Linux中最重要的软件开发工具是GCC。GCC是GNU的C和C++编译器。实际上，GCC能够编译三种语言：C、C++ 和Object C（C语言的一种面向对象扩展）。利用gcc命令可同时编译并连接C和C++源程序。

假设我们有下面一个非常简单的源程序(hello.c)

要编译这个程序，我们只要在命令行下执行:

gcc -o hello hello.c

gcc 编译器就会为我们生成一个名为hello的可执行文件。执行./hello就可以看到程序的输出结果了。

```
#include <stdio.h> #include <stdlib.h>

int main (int argc, char **argv) {
    printf ("Hello world!\n");
    printf ("Hello world again!\n");

    return 0;
}
```

如果你有两个或少数几个C源文件，也可以方便地利用GCC编译、连接并生成可执行文件。例如，假设你有两个源文件main.c和factorial.c两个源文件，现在要编译生成一个计算阶乘的程序。

利用如下的命令可编译生成可执行文件，并执行程序：

```
gcc -o factorial main.c factorial.c
```

GCC 可同时用来编译 C 程序和 C++ 程序。一般来说，C 编译器通过源文件的后缀名来判断是 C 程序还是 C+ + 程序。在 Linux 中，C 源文件的后缀名为 .c，而 C++ 源文件的后缀名为 .C 或 .cpp。但是，gcc 命令只能编译 C++ 源文件，而不能自动和 C++ 程序使用的库连接。因此，通常使用 g++ 命令来完 完成 C++ 程序的编译和连接，该程序会自动调用 gcc 实现编译。

```
#include <stdio.h>
#include <stdlib.h>
int factorial (int n);
int main (int argc, char **argv){
    int n;
    if (argc < 2) {
        printf ("Usage: %s n\n", argv [0]);
        return -1;
    } else {
        n = atoi (argv[1]);
        printf("Factorial of %d is %d.\n",n,factorial(n));
    }
    return 0;
}
```

```
#include <stdio.h>
#include <stdlib.h>

int factorial (int n)
{
    if (n <= 1)
        return 1;

    else
        return factorial (n - 1) * n;
}
```

gcc的主要选项

选项	解释
-c	只编译并生成目标文件。
-E	只运行 C 预编译器。
-g	生成调试信息。GNU 调试器可利用该信息。
-IDIRECTORY	指定额外的头文件搜索路径DIRECTORY。
-LDIRECTORY	指定额外的函数库搜索路径DIRECTORY。
-ILIBRARY	连接时搜索指定的函数库LIBRARY。
-o FILE	生成指定的输出文件。用在生成可执行文件时。

GNU make

在大型的开发项目中，通常有几十到上百个的源文件，如果每次均手工键入 gcc 命令进行编译的话，则会 非常不方便。因此，人们通常利用 make 工具来自动完成编译工作。这些工作包括: 如果仅修改了某几个 源文件，则只重新编译这几个源文件；如果某个头文件被修改了，则重新编译所有包含该头文件的源文件。 利用这种自动编译可大大简化开发工作，避免不必要的重新编译。

实际上，make 工具通过一个称为 makefile 的文件来完成并自动维护编译工作。makefile 需要按照某种 语法进行编写，其中说明了如何编译各个源文件并连接生成可执行文件，并定义了源文件之间的依赖关系。 当修改了其中某个源文件时，如果其他源文件依赖于该文件，则也要重新编译所有依赖该文件的源文件。

makefile 基本结构

makefile 中一般包含如下内容:

- 需要由 make 工具创建的项目，通常是目标文件和可执行文件。通常使用“目标（target）”一词来表示要创建的项目。
- 要创建的项目依赖于哪些文件。
- 创建每个项目时需要运行的命令。

例如，假设你现在有一个 C++ 源文件 test.C，该源文件包含有自定义的头文件 test.h，则目标文件 test.o 明确依赖于两个源文件：test.C 和 test.h。另外，你可能只希望利用 g++ 命令来生成 test.o 目标文件。这时，就可以利用如下的 makefile 来定义 test.o 的创建规则:

```
# This makefile just is a example.
# The following lines indicate how test.o depends
# test.C and test.h, and how to create test.o

test.o: test.C test.h
    g++ -c -g test.C
clean:
    rm -f *.o
```

从上面的例子注意到，第一个字符为 # 的行为注释行。第一个非注释行指定 test.o 为目标，并且依赖于 test.C 和 test.h 文件。随后的行指定了如何从目标所依赖的文件建立目标。注意：在建立目标的每一个命令行的第一个字符必须是制表符。

当 test.C 或 test.h 文件在编译之后又被修改，则 make 工具可自动重新编译 test.o，如果在前后两次 编译之间，test.C 和 test.h 均没有被修改，而且 test.o 还存在的话，就没有必要重新编译。这种依赖 关系在多源文件的程序编译中尤其重要。通过这种依赖关系的定义，make 工具可避免许多不必要的编译工 作。当然，利用 Shell 脚本也可以达到自动编译的效果，但是，Shell 脚本将全部编译任何源文件，包括 哪些不必要重新编译的源文件，而 make 工具则可根据目标上一次编译的时间和目标所依赖的源文件的更新 时间而自动判断应当编译哪个源文件。

运行 make

我们知道，直接在 make 命令的后面键入目标名可建立指定的目标，如果直接运行 make，则建立第一个 目标。
一个 makefile 文件中可定义多个目标，利用 make target 命令可指定要编译的目标，如果不指定目标， 则使用第一个目标。通常，makefile 中定义有 clean 目标，可用来清除编译过程中的中间文件。
运行 make clean 时，将执行 rm -f *.o 命令，最终删除所有编译过程中产生的所有中间文件。

makefile 变量

GNU 的 make 工具除提供有建立目标的基本功能之外，还有许多便于表达依赖性关系以及建立目标的命令的特 色。其中之一就是变量或宏的定义能力。如果你要以相同的编译选项同时编译十几个 C 源文件，而为每个目 标的编译指定冗长的编译选项的话，将是非常乏味的。但利用简单的变量定义，可避免这种乏味的工作。

在下面的例子中，CC 和 CCFLAGS 就是 make 的变量。GNU make 通常称之为变量，而其他 UNIX 的 make 工具称之为宏，实际是同一个东西。在 makefile 中引用变量的值时，只需变量名之前添加 $ 符号，如 上面的 $(CC) 和 $(CCFLAGS)。

GNU make 的主要预定义变量

GNU make 有许多预定义的变量，这些变量具有特殊的含义，可在规则中使用。

AR	归档维护程序的名称，默认值为 ar。
ARFLAGS	归档维护程序的选项。
AS	汇编程序的名称，默认值为 as。
ASFLAGS	汇编程序的选项。
CC	C 编译器的名称，默认值为 cc。
CCFLAGS	C 编译器的选项。
CPP	C 预编译器的名称，默认值为 $(CC) -E。
CPPFLAGS	C 预编译的选项。
CXX	C++ 编译器的名称，默认值为 g++。
CXXFLAGS	C++ 编译器的选项。
FC	FORTRAN 编译器的名称，默认值为 f77。
FFLAGS	FORTRAN 编译器的选项。

```
# Define macros for name of compiler
CC = gcc

# Define a macr o for the CC flags
CCFLAGS = -D_DEBUG -g -m486

# A rule for building a object file
test.o: test.c test.h
    $(CC) -c $(CCFLAGS) test.c
```