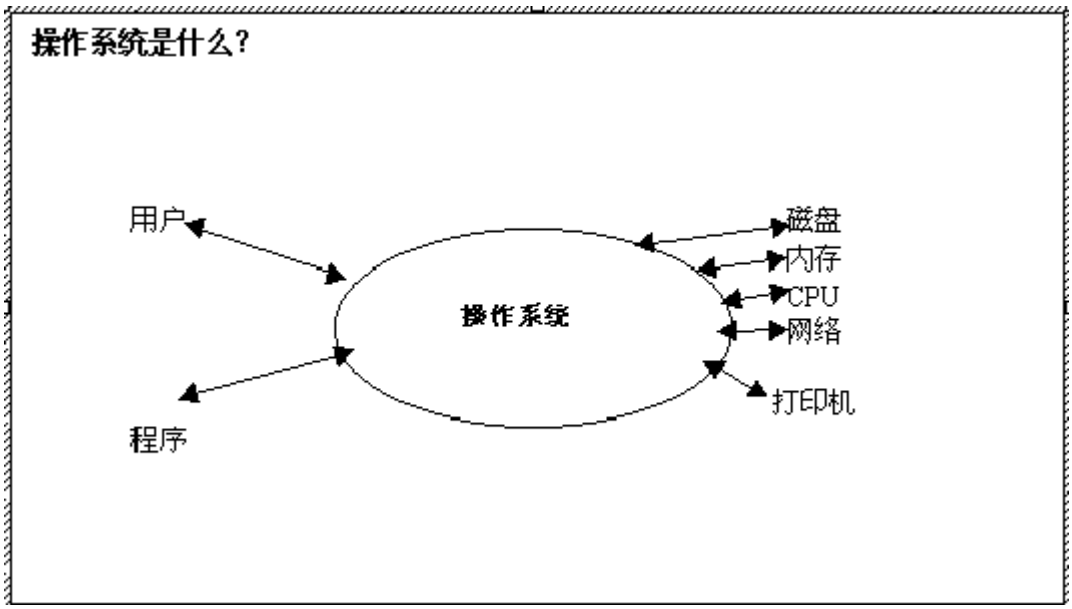
第一章：UNIX 操作系统简介

学习目标

学完这一章，你能做到以下事情：

描述 UNIX 系统的基本组成和基本功能

1.1 什么是操作系统？



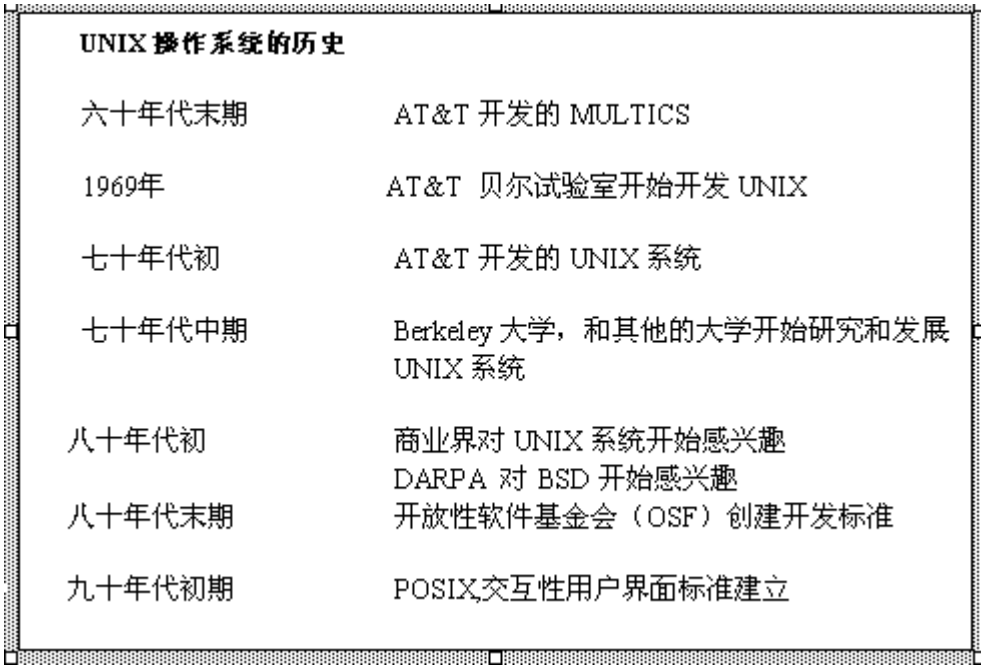
操作系统是一种特殊的用于控制计算机（硬件）的程序（软件）。

操作系统在资源使用者和资源之间充当中间人的角色。为众多的消耗者协调分配有限的系统资源。系统资源包括，CPU，内存，磁盘，和打印机。举个例子，一个用户（也可以是程序）将一个文件存盘，操作系统就会开始工作：管理磁盘空间的分配，将要保存的信息由内存写到磁盘等。

当用户要运行一个程序时，操作系统必须先将程序载入内存，当程序执行时，操作系统会让程序使用 CPU。在一个分时系统中，通常会有多个程序在同一时刻试图使用 CPU。

操作系统控制应用程序有序地使用 CPU ，就好象一个交通警察在一个复杂的十字路口指挥交通。十字路口就象是 CPU ;每一条在路口交汇的支路好比一个程序，在同一时间，只有一条路的车可以通过这个路口，而交通警察的作用就是指挥让哪一条路的车通过路口，直到让所有路口的车辆都能通过路口。

2. UNIX 操作系统的历史



UNIX 操作系统 1969 年在贝尔实验室诞生。Ken Thompson 在 Rudd Canaday, ,Doug Mcllroy,Joe Ossana,and Dennis Ritchie,的协助下，写出一个小的分时系统，开始得到关注，在许诺为实验室的管理人员提供一个文档准备工具后，UNIX 先驱们可以使用到一台更大的计算机，从而得以继续他们的开发工作。
在七十年代的中期，一些大学得到使用 UNIX 的许可，并很快在学院之间得到广泛流行，其主要的原因是：

- 小巧：最早的 UNIX 系统只占用 512K 字节的磁盘空间，其中系统内核使用 16K，用户程序使用 8K,文件使用 64K。
- 灵活：源代码是可利用的，UNIX 是用高级语言写成，提高了操作系统的可移植性。
- 便宜：大学能以一盘磁带的价格得到一个 UNIX 系统的使用许可。早期的 UNIX 系统提供了强大的性能，使其能在许多昂贵的计算机上运行。

以上优点在当时掩盖了系统的不足：

- 没有技术支持;AT&T 在当时大部分的资源和都用在 MUTICS 上，没有兴趣开发 UNIX 系统。
- Bug 的修补：由于没有技术支持，bug 的修补也得不到保证。
- 很少的，或者根本没有说明文档,用户有问题经常只能是去看源代码。

当 UNIX 传播到位于 California 的 Berkeley 大学的时候，Berkeley 大学的使用者们创建了自己的 UNIX 版本，在得到国防部得支持后，他们开发出了许多新的特性。但是，作为一个研究机构，Berkeley 大学提供的版本和 AT&T 的版本一样，也没有技术支持。
当 AT&T 意识到这种操作系统的潜力后就开始将 UNIX 商业化，为了加强产品性能，他们在 AT&T 的不同部门进行 UNIX 系统开发，并且开始在系统中结合 Berkeley 开发出的成果。

UNIX 最终的成功可以归结为：

- 一个灵活的、包含多种工具的用户界面与操作环境。
- 模块化的系统设计可以很容易地加入新的工具。
- 支持多进程，多用户并发的能力。
- Berkeley 大学 的 DARPA 支持。

- 强大的系统互连的能力。
- 能在多种硬件平台上运行。
- 标准化的界面的定义促进应用的可移植性。

1.3 UNIX 系统的特性

UNIX 为用户提供了一个分时的系统以控制计算机的活动和资源，并且提供一个交互，灵活的操作界。UNIX 被设计成为能够同时运行多进程，支持用户之间共享数据。同时，UNIX 支持模块化结构，当你安装 UNIX 操作系统时，你只需要安装你工作需要的部分，例如：UNIX 支持许多编程开发工具，但是如果你并不从事开发工作，你只需要安装最少的编译器。用户界面同样支持模块化原则，互不相关的命令能够通过管道相连接用于执行非常复杂的操作。

运行中的系统

内核是运行中的系统，它负责管理系统资源和存取硬件设备。内核中包含有它检测到的每个硬件的驱动模块，这些模块提供了支持程序用来存取ＣＰＵ、内存、磁盘、终端、网络的功能。当安装了一种新的硬件，新的模块会被加入到内核之中。

运行环境

工具和应用程序

UNIX 的模块化设计在这里表现得非常明显，UNIX 系统命令的原则就是每条命令做好一件事情，组合一系列命令就组成工具箱，选择合适的命令就可以完成你的工作，恰当地组合这些工具能够帮你完成复杂的任务。

从开始，UNIX“ 工具箱 ”就包括了一些可以同系统进行交互的基本命令，UNIX 系统也提供了以下几种工具：

电子邮件(mail,mailx)

文字编辑(ed,ex,vi)

文本处理(sort,grep,wc,awk,sed)

文本格式化(nroff)

程序开发(cc,make,lint,lex)

源程序版本管理(SCCS,RCS)

系统间通讯(uucp)

进程和用户帐号（ps ,du,acctcom）

因为 UNIX 系统的用户环境被设计为一种交互的，可编程的，模块化的结构，新的工具能很容易地开发，并且添加到用户工具箱之中，而哪些不是必须的工具能够被省略，这种省略不会影响系统的操作。

举个例子，一个程序员和一个打字员同时在使用 UNIX 系统，他们会使用许多普通命令，虽然他们的工作性质不相同。他们会用一些与他们的工作相关的工具。程序员使用的工具会包括程序开发和程序管理的工具，而打字员会使用字处理，文档管理的工具。有趣的是：程序员用来修订程序的工具同时也被打字员用来文档的修订。因此，他们的系统看上去十分相似，但是，每一个用户选择模块都与他或她的应用要求密切相关。
UNIX 系统的流行很大程度可以归结与：

- UNIX 系统的完整性与灵活性使其能适应许多的应用环境。
- 众多的集成的工具提高了用户的工作效率。
- 能够移植到不同的硬件平台。

SHELL

Shell 是一个交互的命令解释器。命令是在 SHELL 提示符下键入，shell 会遵照执行键入的命令。用户通过 shell 与计算机交互。shell 从键盘获得用户键入的命令，然后将命令翻译为内核能够理解的格式。然后系统就会执行这个命令。
你会注意到 shell 与内核是分离的两部分。如果你不喜欢当前 shell 提供的特性，你能很容易地用另一种 shell 代替当前的 shell。
一些 shell 是命令行方式，一些提供菜单界面。UNIX 系统支持的普通的 shell 都包括一个命令解释器和一个可编程的接口。

有四个最通用的 shell，分别是：

- Bourne shell --- 由 AT&T 提供的最原始的 shell，由贝尔实验室的 Stephen Bourne 开发。可提供命令的解释，支持可编程接口,提供诸如变量定义，变量替代，变量与文件测试，分支执行与循环执行等功能。

- C Shell (/usr/bin/csh)----C Shell 是在 California Berkeley 大学的 Bill Joy 开发，一般存 在于 BSD 系统中，于是被称为 California shell,简写名称为 C Shell。它被认为是 Bourne Shell 的一个改进版本。因为它提供拉交互的特征例如命令堆栈（一允许简单地调用和编辑以前输入的命令，）别名（提供对已有命令取个人的别名）

- Korn Shell(/usr/bin/ksh)---- 贝尔实验室最新的开发成果，由 David Korn 开发成功。它被认为是是一种增强型的 Bourne Shell，因为它提供对简单可编程的 Bourne Shell 界面的支持，同时提供 C Shell 的简便交互的特征。它的代码也被优化来提供一种更快，更高效的 shell。

- POSIX Shell: POSIX 是一种命令解释器和命令编程语言，这种 shell 同 Korn Shell 在许多方面都很相似，它提供历史机制，支持工作控制，还提供许多其他有用的特性

表 1—1 shell 特征的比较

特性	描述	Bourne	Korn	C	POSIX
命令历史	允许将命令存储到缓冲区，可以修改重利用	不支持	支持	支持	支持
行编辑	用一个文本编辑器修改当前或先前的命令的能力	不支持	支持	不支持	支持
文件名自动完成	自动完成命令行中正在键入的文件名	不支持	支持	支持	支持
命令别名	允许用户重命名命令自动包括命令的选项简化长的命令行	不支持	支持	支持	支持
限制的 shell	提供一个有限功能的受控制下的环境是一个安全特性。	支持	支持	不支持	支持
任务控制	用于查找和存取后台运行的进程	不支持	支持	支持	支持

1.4 UNIX 的其他特征

层次化的文件系统

存储在磁盘上的信息称为文件。每一个文件都分配有一个名字，用户通过这个名字来访问文件，文件的内容通常是数据，文本，程序等等，UNIX 系统通常有几百个文件存在，于是另外一种容器：**目录**被用来让用户在一个逻辑上的分组里管理它的文件。在 UNIX 系统中，目录被用来存储文件和其它的目录。

文件系统的结构非常复杂，如果用户的工作部门改变，用户的文件和目录能很容易移动，改名，或组织到新的或不同的目录中，这些操作只需使用一些简单的 UNIX 系统的命令即可完成。文件系统就象一个电子排列柜，它能让用户分割，组织他们的信息到适合自己环境与应用的目录中去。

多任务

在 UNIX 系统中，能有几个不同的任务在同一时刻执行。　一个用户在一个终端可以执行几个程序，看上去好象是同时在运行。这意味着一个用户可以编辑一个文本文件时格式化另一个文件，同时打印另一个文件

实际上，CPU 在同一时刻只能执行一个任务，但是 UNIX 系统能够将 CPU 的执行分成时间片，通过调度，使在同一时间内执行，对用户看来，就好象在同时执行不同的程序一样。

多用户

多用户就是允许多个用户在同一时刻登录和使用系统。多个终端和键盘能连接在同一台计算机上。这是多任务功能的一种自然延伸。如果系统能够同时运行多个程序，一些程序也能够支持多个用户线索。另外，一个单个用户能够通过多个终

端在不同的时刻登录同一个系统。　这种体系结构的一个很大的好处是：工作组的成员能同时操作相同的数据。

第二章：登录过程和普通命令

目标

完成了这一章，你能够做以下事情：

- 登录 UNIX 系统
- 退出 UNIX 系统
- 使用联机帮助查找命令用法
- 理解 shell 命令的格式
- 用一些简单的命令来鉴别系统用户
- 用一些简单的命令来与系统中的用户通信
- 用一些简单的命令来实现多种功能并输出结果

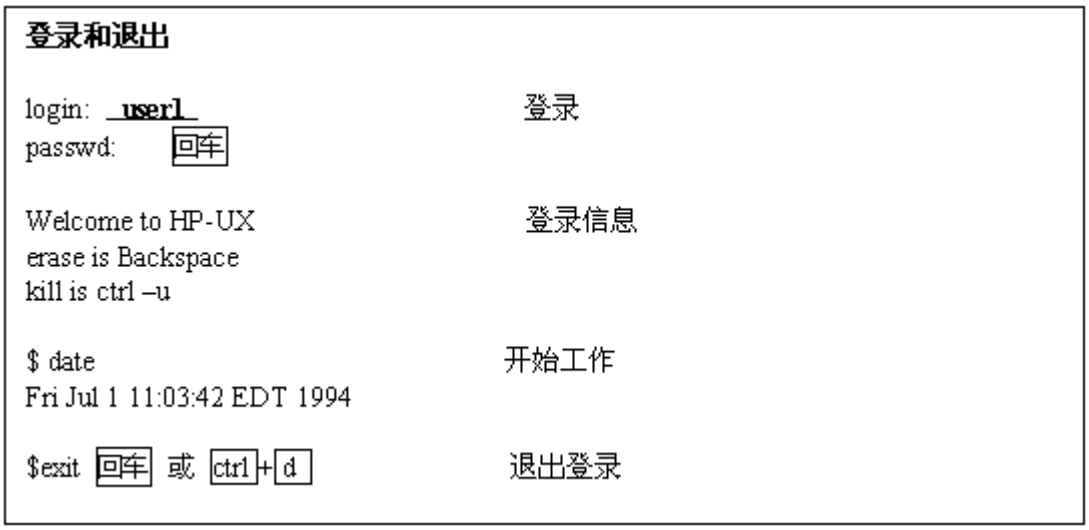
2.1 一个典型的终端会话过程

为了能和计算机通信，需要有以下条件：

- 一个具有完全 ASCII 字符集的终端
- 一条连接计算机的数据通信线路
- 一个登录 ID（用户身份确认）
- 一个密码

一个终端会话过程开始于用户登录一个系统认可的终端，结束于退出信号（logoff），计算机会在你登录期间响应你的键入的命令
UNIX 通过用户名（有时也称为登录ＩＤ）来识别是否是系统中的用户。你的用户名，就是系统管理员分配给你的名称，通常是你的名字或是名字的缩写。你帐号的密码可以是一个随意的的字符串，你的系统管理员会提供给你一个你可以更改的初始化密码，你的密码是你私人所有的，你自己决定密码是什么，没人知道或能找出你的密码是什么。如果你忘记了你的密码，你就不得不求助于你的系统管理员。因为只有系统管理员有权删除修改用户的密码。
在终端上出现登录提示符时，你就可以输入用户名和密码登录以系统。
在登录上系统后，你可以输入命令，SHELL 会解释命令，操作系统会为你执行命令，任何执行命令产生的响应都会显示在你的屏幕上。
当工作结束后，你可以退出系统以终止终端对话，这会释放终端以便让其他人可以通过这台终端登录系统。建议你在离开时退出系统，以防其他人使用你的帐号使用系统。

2.2 登录和退出



你需要执行以下步骤来登录系统：

- 打开终端，如果一些终端显示已经超时（现象为屏幕上没有任何显示）你只需要敲一个键（例如 shift 键）激活显示。

- 如果你没有看到 **login：**提示符，或者出现一些垃圾字符，敲一下回车，如果仍旧不工作，敲一下 break 键。垃圾字符通常是计算机试图以错误的速度与你的终端联结，break 键告诉计算机尝试另外一种速度，你可以通过敲 break 键来实验不同的速度，但记住在每次敲键后要等待机器响应。

- 当 login:提示符出现后，键入你的登录ＩＤ。

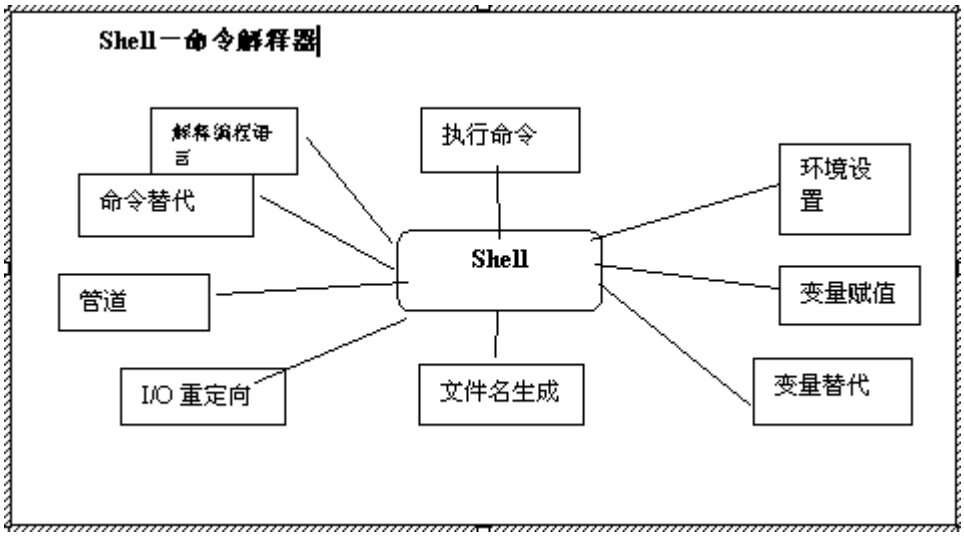
- 如果 passwd:提示符出现，键入你的密码，为了确保安全，你键入的密码不会在屏幕上显示。***注意***：键盘上的退格键在登录过程中没有删除功能。

  **$** 符是 Bourne Shell(/usr/old/bin/sh),korn Shell(/usr/bin/ksh),和 POSIX Shell(/usr/bin/sh)的标准提示符，%符通常表示这是 C Shell(/usr/bin/csh)。如果你用的是 POSIX shell，你会看到一个$提示符。而#提示符一般是为系统管理员保留的。当你以系统管理员的身份登录时,会出现这个符号。这也为你的特殊身份提供一个有用的提示功能：因为作为系统管理员，你能更改（或删除）系统中的任何数据。
  密码的说明：
  第一次登录系统，你的帐号可能被设置为要你键入密码。你提供的密码必须满足以下条件：

- 你的密码必须不小于六个字符
- 前六个字符中至少要有两个字母。
- 前六个字符中至少要有一个不是字母。

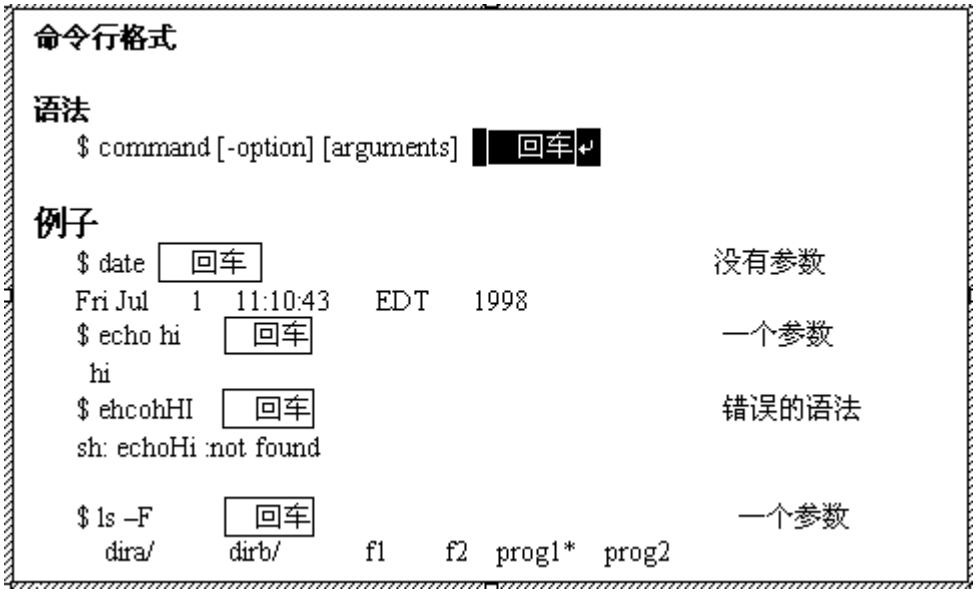
当你第一次输入你的密码后，系统会提示你重新输入一次以确认密码。然后系统会重新出现登录提示符，你可以用你的新的密码来完成登录步骤。

2.3 Shell-命令解释器



当你登录的过程中，shell 已经开始在为你工作。shell 在终端上显出提示符，并且解释你输入的命令，我们会在本章剩下的部分讨论不同的命令，包括如何让你存取在线帮助，找出谁登录了你的系统，和如何同你系统中的其他用户通信。
正如你在上图中所看到的，shell 还支持许多其他的命令解释功能

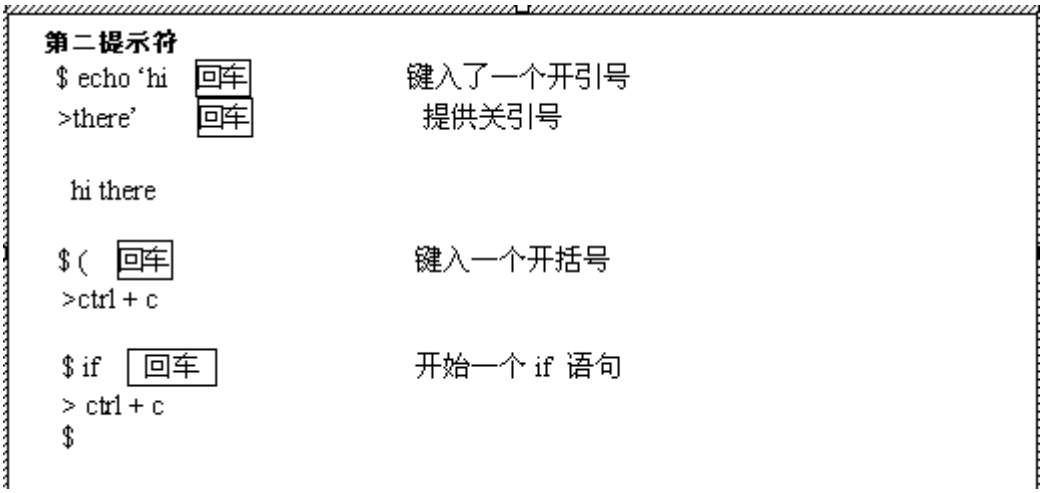
2.4 命令行的格式



在你看到 shell 提示符($)后，你就可以键入命令。一个正确的命令名通常是在命令行的第一项。许多的命令还有命令选项，选项的作用是为了扩展命令功能，而参数通常是一些文本，一个文件名，或者是目录名等命令可以操作的东西。选项通常以短横线（-)开始。

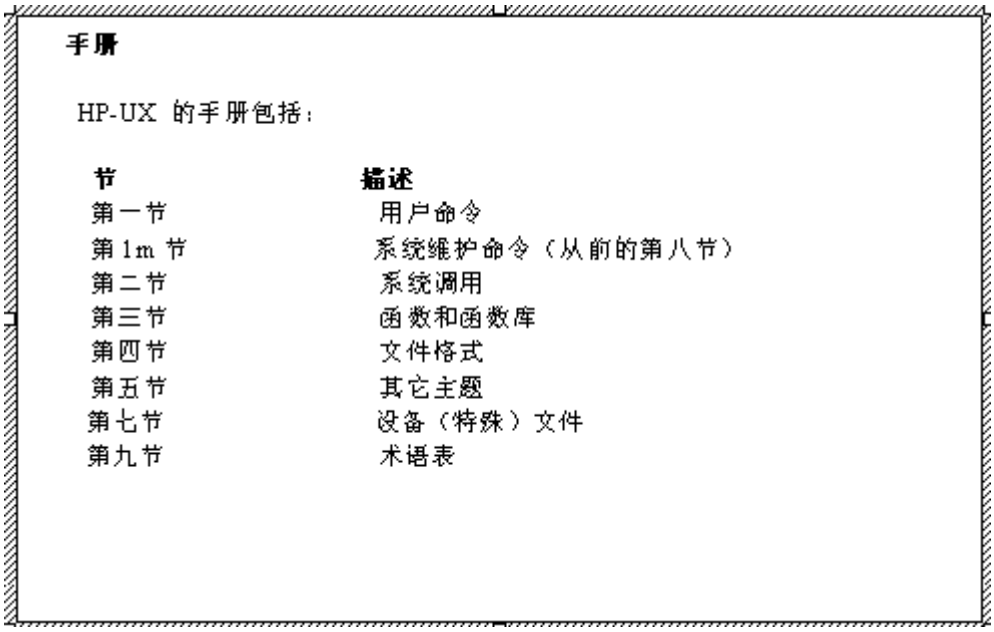
空字符的作用是分割命令，选项，参数，空字符被定义为一个或多个空格符（space)，或 TAB 键，例如：命令 echo Hi 和 echoHi 是有很大的差别的，计算机会将第一个理解为命令 echo 和一个命令的参数 （Hi），而第二个会被认为是一个名为 echoHi 的命令，而此命令可能不是一个有效的命令，
每个命令都是以回车符结束，回车符会将命令送往计算机执行。在上图中，回车符会被系统识别，而其他的字符不会产生作用。
终端的输入输出支持预先键入字符，预先键入的意思是在你键入一个命令后在提示符返回前立即键入另一个命令，你键入的命令会先被缓存，并在当前命令执行完成之后执行。
注意：UNIX 的命令是对大小写敏感的，大多数的命令和参数都是小写的字符。因而，echo hi 是一个合法的命令而 ECHO hi 不能被系统识别。
注意：你可以在一个命令行中同时键入两个命令，方法是用分号（；）分割两个命令，
如：
```
$ ls;pwd
```

2.5 第二提示符



Bourne ，Korn,和 POSIX shells 都支持多命令行。如果 shell 要求更多的输入来完成一个
命令，在回车前，就会出现第二提示符(>)。
一些命令要求有一个结束命令，一些字符要求有结束字符，例如一个开始的 if 要求 fi
来结束，开始的括号要求一个结束的括号，一个开始的引号要求一个结束的引号。
如果你键入一个错误的命令，就像上图中描述的一样，SHELL 会显示第二提示符。通常同
时按 ctrl +c 会中止当前正在运行的程序，然后返回到 SHELL 提示符下，你可以通过 stty
-a 命令来自行定义你的中止键

2.6 帮助手册


手册

HP-UX 的手册包括：

节	描述
第一节	用户命令
第 1m 节	系统维护命令（从前的第八节）
第二节	系统调用
第三节	函数和函数库
第四节	文件格式
第五节	其它主题
第七节	设备（特殊）文件
第九节	术语表

联机手册对查找命令用法非常有用，但是它不是作为教材而设计的,它对学习如何使用 UNIX 操作系统的帮助也不大。有经验的 UNIX 系统用户使用帮助来查找命令的细节和用法。就像上图中表示的那样，帮助手册被分为几个部分。
以下是每一节的简短的说明：

第一节：用户命令

这一节描述直接被用户或是 shell 脚本使用的程序。任何系统中的用户都可以使 用这个帮助。

第 1M 节：系统维护

这一节描述系统管理员用来维护系统的命令。只有 root 用户才能使用。

第二节：系统调用

这一节描述与 UNIX 系统内核接口的函数，包括 C 语言的接口。

第三节：函数与函数库

这一节阐述了系统提供的与直接系统调用不同的一些二进制格式的函数。这些函 数通常通过 C 程序来存取，例如：输入输出处理和数学计算的函数。

第四节：文件格式

这一节定义用户配置文件的组成，文档，和不同文件类型的格式（例如 a.out)

第五节：多方面的主题

这一节包括不同的信息，例如头文件的描述，字符集，宏包。和其它的主题。

第七节：特殊的设备文件

这一节讨论特殊的设备文件的特性，这些文件提供 uniz 系统与系统 I/O 设备（例 如磁盘，磁带机，打印机）之间的链接。

第九节： 字典

这一节定义了被选择的贯穿整个帮助手册中的术语。

在每一个章节里，命令以字母的顺序列出，用户可以通过检索手册的索引来找到 一个命令。

2.7 手册页中的内容

手册页中的内容

名字	**例子**
大纲	**警告**
描述	**相关**
外部影响	**作者**
网络特性	**文件**
返回值	**同样可看**
诊断信息	**臭虫**
错误	**标准一致性**

了解帮助页的格式是非常重要的。在整个 UNIX 系统的文档中，参考以 cmd(n)的格式给出，其中 cmd 是命令名，而 n 代表八个帮助章节的其中一个，因而，date(1)代表在帮助手册第一节中关于 date 命令的内容，在每一个章节里，命令是以字母的顺序列出的，这是由维护帮助手册的方式造成的，页的编号没有意义，每一个命令都是从第一页开始的。
每一个帮助页（一些命令有多个帮助页）有几个主要的标题。帮助页也不总是有标题。

以下是每个标题的列表和其内容的描述；
名字 包括命令名和简短的描述，章节中的这个文本被用来产生索引。
大纲 定义了如何引用一个命令。黑体字的条目表示必须在终端上正确输入的部分。方括 号中的是任选项，规则类型的条款会被你所选择的合适的文本所代替，省略号（ .....)被用来显示先前重复的参数。如果对摘要的意思有疑问，你可以去阅读 DESCRIPTION 项。
描述 包括每个命令和选项的功能的详细描述。
额外的影响 提供不同口语的编程信息，这对于国际化的支持非常有用。
网络特性 基于网络特征的功能。
返回值 描述程序调用完成后的返回值。
诊断信息 解释命令可能出现的错误信息

错误　　列出错误的条件合相应的错误信息或是返回值。

范例　　提供命令使用的范例。

警告　　指出潜在的陷阱。

相关性　　指出与不同硬件平台相关的 UNIX 系统操作的变化。

作者　　命令的开发人员。

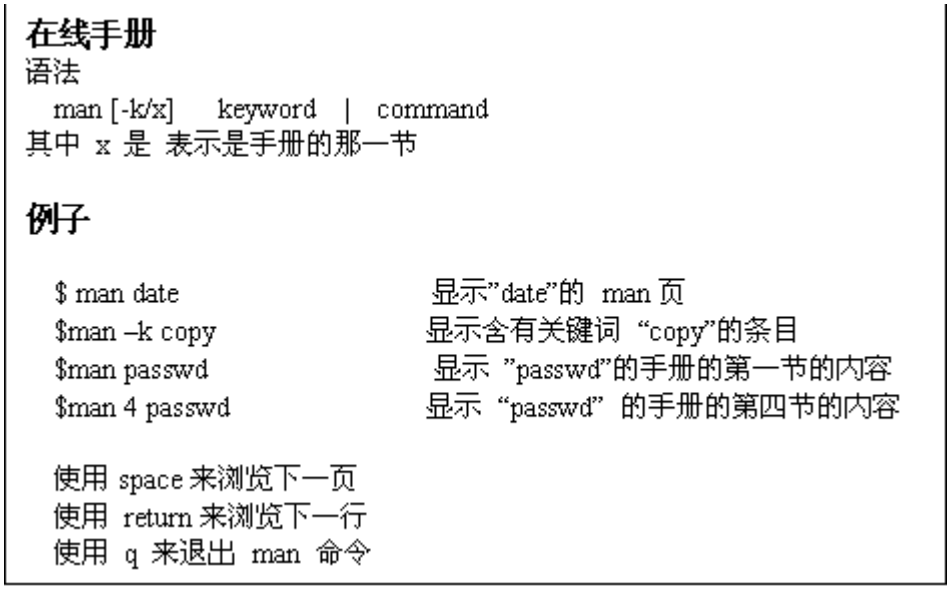
文件　　命令使用的任何特殊的文件。

SEE ALSO　　指明帮助手册中的其他页，或是其他包含附加信息的文档。

BUGS　　讨论已知的漏洞和缺陷和所支持的修正

标准一致性　　描述每一个条目支持的标准。

2.9 联机手册



有另外一种方法从帮助手册中检索信息。

在许多 UNIX 系统中，手册通常是在线的。在线帮助手册通过 man 命令来存取。

语法是：

```
man -k keyword
```

或者

```
man [12345791m] command
```

其中

`man -k keyword` 列出所有的命令，在命令描述中有字符串 keyword。

`man [1234579m] command` 显示指明帮助章节的命令的帮助页。

`man command`　显示命令的默认的帮助条目。也许一个命令的条目，在不止一个的帮助页中。

以上的所有的命令要求系统管理员已经正确地安装了在线帮助。在以上的例子中，`man passwd` 会显出改变密码的命令。`man 4 passwd` 会显示 passwd 文件的格式。

在指定命令的帮助条目的第一页已经出现在屏幕上。你可以键入以下的键：

return 显示下一行
space 显示下一页
Q 或 q 退出 man 命令并且回到提示符下：
有时，在你读取在线手册时会看到一下的信息：
reformatting retry .wait.....
这个信息的意思是指定命令的帮助手册页需要解压缩，因为对当前用户来说是第一次使用。这条信息在用户下一次检索这个手册时不会再出现。
多个帮助手册页
一些命令有多个帮助条目。你可以用 whereis 命令来显示帮助的章节。例如：

```
$ whereis passwd
passwd :/sbin/passwd /usr/bin/passwd /usr/share/man/man.1.z/passwd.1
/usr/share/man/man4.Z/passwd.4
$whereis nothere
nothere:
```

这说明在章节 1 和 4 中有一个关于 passwd 命令的帮助条目，没有关于 nothere 的帮助手册。

2.10 一些初级命令

一些初级命令

id	显示用户和组的 id 号
who	识别正在系统中的其他用户
date	显示系统的时间和日期
passwd	给你的用户帐号分配一个密码
eho	在你的屏幕上显示简单的信息
clear	清除终端屏幕
write	向其他用户的终端发送信息
mesg	允许/拒绝其他用户向你的终端发送信息
news	显示系统的新闻

我们将学习一些基本的命令，这些命令中的大多数除本书讲到选项以外的还有更多的选项，如果你想要查看其他的选项，可以使用联机手册。

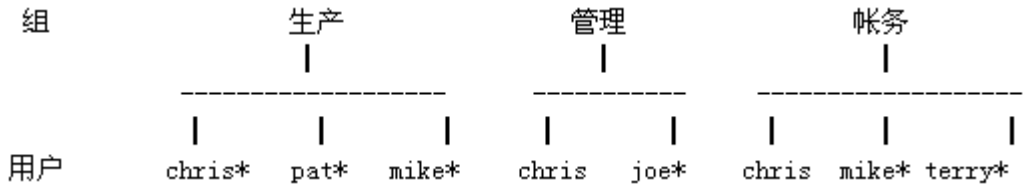
id 命令

为了让你存取文件和执行程序，UNIX 必须要知道你的用户和组的 id 号，这种由计算机维护的数字的 id 号，对应哪些为用户方便而使用文字名称，当你登录时，你的用户身份会被初始化。在你已经登录后，你也可以更改你的用户和组。id 命令的作用就是显示你当前的用户和组的定义。

计算机将所有的用户的信息存储在文件 /etc/passwd 中，组的信息存储在/etc/group 中。

组

组让一组用户可以共享地存取文件。由系统管理员定义组中的用户，每一个用户都能存在于不同的组中。组通常由机构中已经定义的工作组组成。例如，一个机构可能由生产，管理，帐务组组成。这些组的用户结构可能定义为一下的结构：



*号指明登录时的组的定义

- chris 是所有的三个组的成员。
- mike 是两个组的成员。

在这个机构内，chris 能够使用与生产，管理，帐务三个部门有关的文件，mike 能够存取与生产，帐务部门有关的文件。其他的用户只能存取他们登录组中的文件。

who 命令

语法

who [am I]　　报告当前系统中的用户的信息
woami

例子

```
$ who
  root      tty1p5    Jul  01  08:01
  user1     tty1p4    Jul  01  09:59
  user 2    tty0p3    Jul  01  10:01

$ who am i
  user2     tty0p3    Jul  01  10:01
$whoami
  user2
```

who 命令会报告有哪些用户已经登录系统，每一个用户连接的终端和登录时间的信息。who am i 报告本用户的用户名和端口信息，whoami 命令报告系统与本地终端的关联的用户名。用户是否有权执行一个命令，依靠的是用户的身份，一个用户能够更改他或她的身份标识来存取其它的命令或程序。

date 命令

date 命令被用来报告系统的当前日期和时间。date 可以加上参数来重新格式化
输出的格式。
通常 date 命令不带任何选项和参数使用。
只有系统管理员有权限修改系统时间和日期。

passwd 命令

在许多的系统中，系统管理员控制着用户的密码。然而，在 UNIX 系统中，系统管理员能允许用户直接控制他们自己的密码，passwd 命令让用户改变他们的口令，语法如下:
passwd
输入这个命令后，系统会要求输入你当前的密码，（老密码）。这是为了避免在你登录了系统后，离开你的终端时间内有人更改你的密码。然后系统会要求你输入新密码，并要求重输入一次以确认你的新密码，这样做是为了避免你打字的错误。你的新旧密码必须至少有三个字符不同。
在你输入密码时，新旧密码字符不会出现在屏幕上。

密码限制

你的密码至少有六个字符，前六个字符中至少要有两个是字母，前六个字符中至少有一个不是字母。
系统管理员可以不遵守这些条件，所以如果系统管理员给你的帐号分配一个密码，这个密码可能不符合这些规则。

13. echo 命令

语法

echo [参数...。] 输出参数到终端

例子

```
$ echo how are you
how are you

$ehco 123 abc
123 abc
```

echo 命令使你可以显示命令行参数的，这个意思是，一个如下的命令：
echo hello
导致输出：
helllo

这个命令可能看上去十分平常，但是这个命令让我们可以很方便地在 shell 程序中显示用户信息和检查 shell 变量的值。在 shell 编程中，echo 命令用的很普遍。

clear 命令

clear 命令清除终端屏幕上的字符。这个命令仅仅清除当前的屏幕，所以，用户有可能向上翻屏来查看以前的屏幕信息，想要清除所有的屏幕信息，按 HOME 键，将光标至到 home，然后键入 clear 命令。

2.17 write 命令

你可以用 write 命令向当前登录到同一个系统的用户的终端发送信息。当你使用 write 时，write 会让你输入信息，每一次你敲回车，信息就会被传送到接收者的终端上，接收者可以向你回写信息，你可以通过你的终端进行交互的对话。当你完成键入信息后，敲入 ctrl+d.就可以结束你的对话。

注意：除非你禁用这项功能，否则，在任何时候，别人都可以发送信息到你的终端。，如果这时你正在使用一个工具，如 man,mail,或是一个编辑器的时候，一个用户给你发生一个信息，这行信息会出现在你的屏幕上，这会造成你的混乱。

如果你想要发送信息给一个用户，而这个用户当前没有登录系统，你会得到如下提示：

user **is not logged on**(用户没有登录系统)，其中 user 表示你试图发送信息的人的

用户名

mesg 命令

语法

mesg [y|n] 允许或拒绝他人"写" 到你的终端上

例子：

```
$ mesg
is y
$ mesg n
$ mesg
is n

$ mesg y
$ mesg
is y
```

你可以通过 **mesg** 命令来禁止其他用户发送信息到你的终端上。如果你给一个已经禁止接收其他用户发送信息的用户发送信息，你会接到 **Permission Denied** 错误（没有许可）。

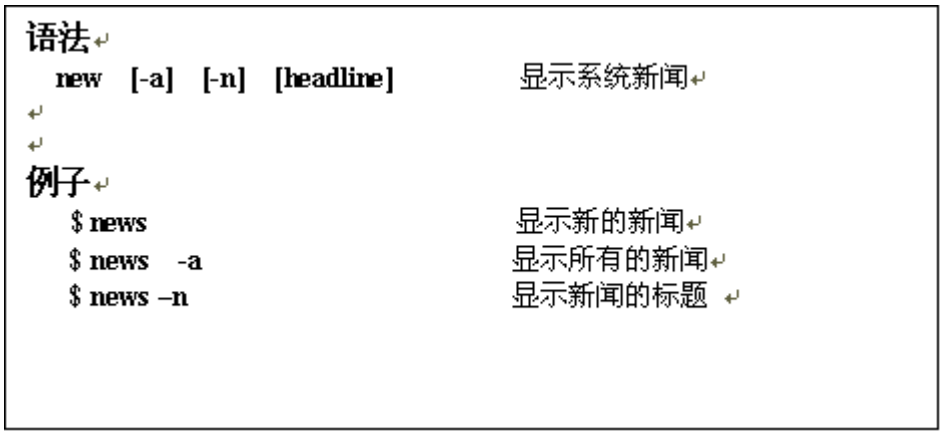
mesg n 拒绝其他人 write 到你的终端。

mesg y 允许其他人 write 到你的终端。

mesg 报告是允许或是不允许其他人写到你的终端。

即使你的终端是禁止写入的，系统管理员一样能发送信息到你的终端。

news 命令



系统中的所有用户都感兴趣的信息可以通过 news 命令广播出去。这个命令通常是系统管理员对系统中所有用户进行通告的时候，例如在系统关闭，备份时，或是在新的硬件生效时使用。

你可以键入 news 命令来阅读新闻。如果命令后没有选项，只有那些你还没有阅读过的信息会显示。

news 命令的选项有：

-a 读取所有的新闻，不管是否已经被读取过。

-n 只显示未读过的新闻的标题

每一个存取新闻的用户在他的 HOME 目录下都有一个.news_time 文件。每一个 UNIX 系统中的文件都有一个时间标志，时间标志记录有上一次文件被修改的时间。.news_time 上的时间标志会被更新，以匹配你最后读取得新闻信息的时间。如果一条新的新闻加入，news 命令知道这条新闻还没有被阅读，因为你的.news_time 文件的时间标志比新的新闻的时间标志早

第三章 文件系统导航

目标

完成这一节，你能做以下事情：

- 描述 UNIX 文件系统的布局
- 描述文件与目录的不同之处
- 顺利地操作一个 UNIX 文件系统。
- 建立和删除目录
- 描述绝对路径和相对路径的不同之处
- 在可能的情况下，使用相对路径来简化你的输入。

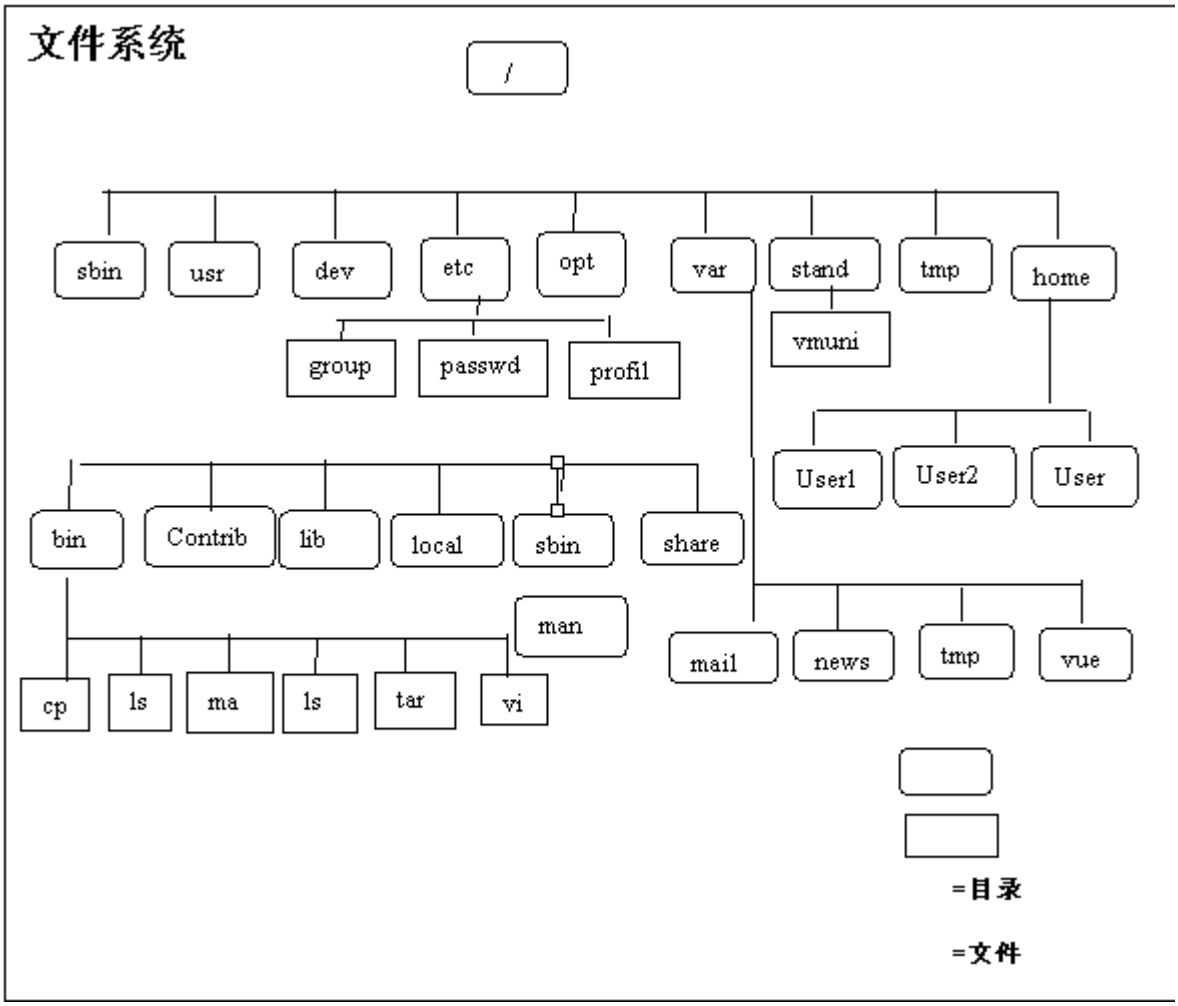
3.1 什么是文件系统

UNIX 系统提供文件系统来管理和组织你的文件和目录。文件通常是数据的一个容器，而目录是文件和（或）其他目录的容器，一个目录包含下的另一个目录通常被称为子目录，
UNIX 的文件系统与文件柜十分相似。整个文件系统就象一个文件柜，文件柜包含所有的抽屉，文件夹，和报表。抽屉同子目录一样能够包含报告和文件夹。一个文件夹当它包含报表时就像一个子目录。报表就代表一个文件，因为它存储实际的数据。

3.2 树型结构

目录结构能够用一个层次化的树形结构来表示。树上的每一个分支可以是目录或者文件。目录用椭圆来表示，文件用矩形来表示，以便图表中能够区别它们。

3.3 文件系统层次



象整齐的文件柜一样，UNIX 文件系统层次提供了一种简单有效的机制来组织你的文件。由于一个 UNIX 发布版本通常包括几百个文件和程序，于是每个 UNIX 系统都支持一种默认的目录结构。在目录的顶端是根目录（因为它在一个倒转的树的顶部），根目录由一个反斜杠（/）来表示。
UNIX 系统同时提供了一些命令，可以让你在你需要时很容易地创建新的目录，也可以从一个目录移动或是拷贝文件到另一个目录。就象加入一个新的文件夹到一个文件柜的抽屉中和将新的文件夹移动一个报表到一个老的文件夹一样容易。

在 HP-UX 10.0 中，文件系统由两个主要的部分组成：静态文件和动态文件。

静态文件：（共享的文件）有三个重要的目录：/opt, /usr , /sbin

/opt 这个目录会用来存放应用程序和产品。开发人员和系统管理员会用它来安装新的产品和本地的应用程序。

/usr/bin 这个目录包含了基础的 UNIX 系统操作和文件处理的命令，所有的用户都有权限读取这个目录（"bin" 是 binary 的缩写）。

/usr/sbin 这个目录中有所有的在帮助手册 1m 章节中的命令，这些命令都是系统管理命令。必须是超级用户才能使用其中的大多数命令。在帮助手册 1m 中有关于这些命令的文档。

/usr/lib 这个目录包括应用程序使用的文档和共享的库

/usr/share 这个目录包括独立提供的文件（其中最重要的是帮助手册）

/usr/share/man 这个目录包括所有的语在线帮助页有关的所有的文件。

/uar/local/bin 这个目录通常用来存放本地开发的程序和工具。

/usr/contrib/bin 这个目录通常用来存放公用的程序和工具

/sbin 这个目录包括基本的用于启动与关闭系统的命令。

动态文件（私有的文件）在这个节有七个重要的目录：

/home, /etc, /stand , /tmp , /dev, /mnt , 和/var;

/home 每一个 UNIX 系统的用户都有他或她自己的帐号。同登录 id 和口令一起，系统管理员会提供给你你自己的目录。**/home** 目录一般都包含每个系统用户的一个子目录，你对你自己的目录有完全的控制权。你有责任在你自己的目录建子目录和文件时对这些目录和文件进行组织和管理。当你登录进入一个系统，你会进入与你帐号相联系的目录中，这个目录，通常被称为 **HOME** 目录或是**登录**目录。从这个目录，你可以进入任何呢曾经存取的其他的目录，最少你可以存取在你 HOME 目录中的任何东西；最多你可以移动到任何地 UNIX 系统的目录（默认的 情况），直到系统管理员限制用户对系统中特殊目录的存取。

/etc 这个目录中有许多的系统配置文件，这些文件在帮助手册的

第四章节有说明文档。

/stand/vmUNIX 这个文件存储的是 UNIX 系统内核的文件。当系统启动时，这个程序 被装载入内存，控制所有的系统操作。

/tmp 这个目录通常被操作系统的一个临时空间，通常是在操作系统创建 中间文件，或是工作临时文件时使用。

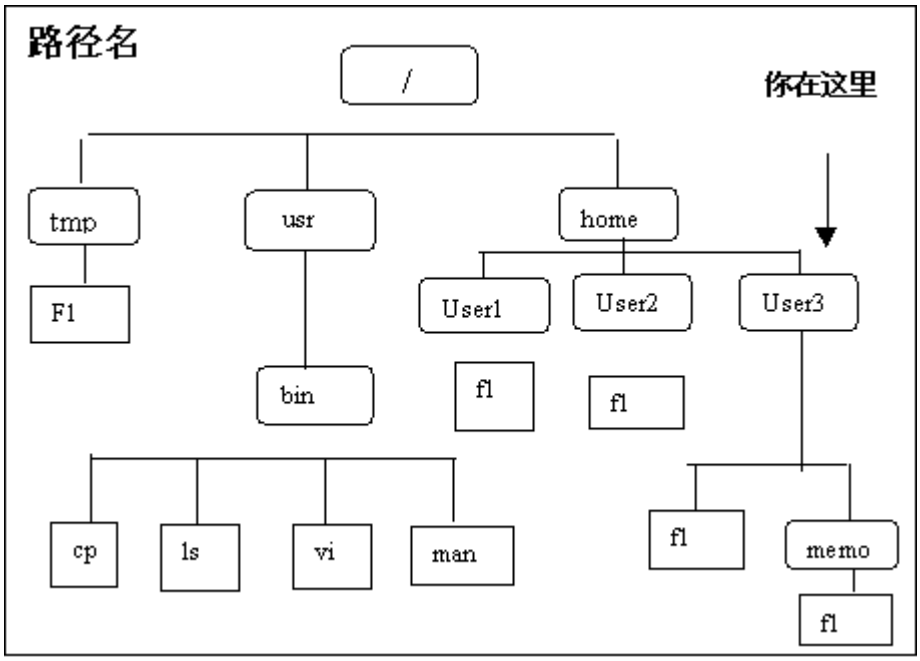
注释：UNIX 系统的惯例：任何时间，任何以 tmp 为名的目录下的任何文件都可以被删除。

/dev 这个目录下有那些可以被联接到你系统中的硬件设备的文件，由于 这些设备是作为一个到设备之间的联接，数据从来不会被直接存储

到这些文件中，这些问文件通常被叫 做特殊文件或是设备文件。

/mnt　　这个用来安装其它的设备 （例如：光驱）
/var/mail　　这个目录包括每一个有邮件的用户的信箱。
/var/news　　这个目录包括当前的新闻信息的所有的文件。他的内容可以通过键入 new -a 来显示。
/var/tmp　　这个目录通常被用于用户的临时空间。

3.4 目录名称



绝对路径：	相对于/home/user3 的路径
/home/user3/f1	f1
/home/user3/memo	memo
/home/user3/memo/f1	memo/f1

	相对/home/user1 的路径
/home/user1/f1	f1

许多的 UNIX 系统命令的操作对象是文件和（或）目录。为了告诉命令你所要求操作的文件或目录，你需要提供一个路径名作为这个命令的一个参数。**路径名**代表遍历一个层次结构来找到你所需要的文件或目录所经历的路由。

```
$ command [options] [pathname pathname.......]
```

为了阐明目录名的概念，我们使用模拟的方法，用一只铅笔从沿着 UNIX 系统的树形结构从一个位置画到另一个位置。路径名也就是铅笔的笔迹通过的层次结构所遇到的节点（即目录）的一个列表，这个列表直到你想要到达的目录或文件为止。

当指明文件或目录的路径名时，反斜杠（/）被用来分割目录或文件名。

Directory/directory/directory（目录/目录/目录）
Directory/file(目录/文件)

无论你何时登录一个 UNIX 系统，你会被定位到层次结构的一些目录下，你可以 通过 UNIX 系统命令来更改你的目录到其他的目录中去，但是你通常是在一些目录 中。举个**例子**：当你登录是，你会被初始化置于你的 HOME 目录中。

绝对路径和相对路径都能够指明文件和目录的位置

绝对路径

- 给出文件或目录的位置的完全的描述。
- 通常由层次结构的顶端开始（根目录）。
- 通常第一个字符是 /.
- 不依靠你当前在目录结构中的位置。
- 整个目录结构只有一条路线。

绝对路径名的例子

以下的路径名指明目录结构中的所有的叫做 f1 的文件的位置。请**注意**，有许多 f1 文件，但是每一个文件的绝对路径都是不相同的。

```
/tmp/f1
/home/user1/f1
/home/user2/f1
/home/user2/f1
/home/user2/f1
/home/user3/memo/f1
```

相对路径

- 通常由目录结构中的当前的位置开始
- 不由 / 开始。
- 相对当前的位置只有唯一表示方法。
- 一般都比绝对路径要短。

相对路径的例子

以下的**例子**再次指明名为 f1 的文件，但是他们的相对路径的定义是依靠用户在目录结构中的当前位置。

假设当前的位置是 **/home:**

```
user1/f1
user2/f1
user3/f1
user3/memo/f1
```

假设当前的位置是**/home/user3:**

```
f1
```

memo/f1
假设当前的位置是/home/user3/memo
f1

请*注意*相关的文件名，f1 不是唯一的，但是 UNIX 系统知道应该去找那个文件，因为系统知道你是在/home/user1 检索/home/user1/f1，还是在/home/user3/memo 的位置检索/home/user3/memo/f1/。同时，你会*注意*到，相对路径可能比绝对路径要短的多，例如，如果在目录/home/user3/mem 中，你可以用一下两个命令来打印文件 f1:
绝对路径： **lp /home/user3/memo/f1**
相对路径： **lp f1**
这表明使用相对路径名可以节约你大量的键盘操作。
***注释*：如果系统中有同名的文件存在于不同的目录中，而你又要使用相对路径来存取文件，这时，知道你当前的位置，是非常重要的。**
在 UNIX 系统内部，系统通过**绝对路径**来查找所有的目录和文件。因为绝对路径名绝对并且唯一的确定一个文件和目录，（由于只有一个根（/））,UNIX 系统允许使用相对路径仅仅是为了方便用户的键入。

3.5 一些特殊的目录

绝对路径	相对于/home/user3 的路径
/home	..
/home/user2	../user2
/home/user1/f1	../user1/f1
/	../..
/tmp/f1	../../tmp/f1
/usr/bin/vi	../../bin/vi

任何目录在创建时，两个条目会自动被创建，它们分别是 点(.),和点点（..）。在使用相对路径的时候通常会用到这两个条目。在上一个**例子**中，你也许已经**注意**到：相对路径的**例子**仅仅只能向下穿越文件结构，但如果使用.. ，你也能够向上穿越文件系统。

登录目录

当一个新的用户被加入到系统中，他（或她）会被分配一个登录 id，可能还有一个密码，和一个用户自己拥有和控制的目录。这个目录通常创建在/home 目录下，与用户的登录 id 名相同，然后这个用户就可以在此目录下任意创建自己的文件和子目录。
当你登录一个系统时，UNIX 系统会将你放在这个目录下，这个目录被称为你的登录目录或者是你的 HOME 目录。

点（.）

这个称为点的条目代表你当前所在地目录。

点（.）的例子

如果你当前正处于目录/home/user3 中：

. 代表你当前的，目录/home/user3

./f1 代表 /home/user3/f1

./memo/f1 代表/home/user3/memo/f1

点点（..）

这个称为点点的条目代表当前目录的上一级目录。通常被称为父目录。每一个目录在自己的下面都可以有多个文件和多个子目录。但是每一个目录只能有一个父目录。这样，当向上进入文件系统的结构时不会有混淆。

根目录（/）象其他的目录一样，包含有点，和点点条目，但是由于根目录没有父目录，所以它的点点代表是根目录本身。

点点（..）的例子：

如果你当前所在地目录是/home

.. 代表/ 目录

../.. 同样代表/目录

../tmp 代表 /tmp

../tmp/f1 代表/tmp/f1

如果你当前的目录是/home/user3

.. 代表 /home 目录

../.. 代表/ 目录

../user2 代表/home/user2

../user1/f1 代表/home/user1/f1

../../tmp/f1 代表/tmp/f1

在例子的最后，绝对路径比相对路径还要短，如果相对路径使你穿越 / 目录，你可以使用绝对路径来代替相对路径。

3.6 基本的文件系统命令

文件系统命令	
pwd	显示你当前在文件系统层次结构中的目录名
ls	查看当前目录下地文件和目录
cd	改变你当前在文件系统中的目录到另一个目录中去。
find	查找文件
mkdir	创建一个目录
rmdir	删除一个目录

目录，就像一个文件夹，是组织你的文件的一种方式。下部分会介绍基本的目录操作命令，这些命令可以：

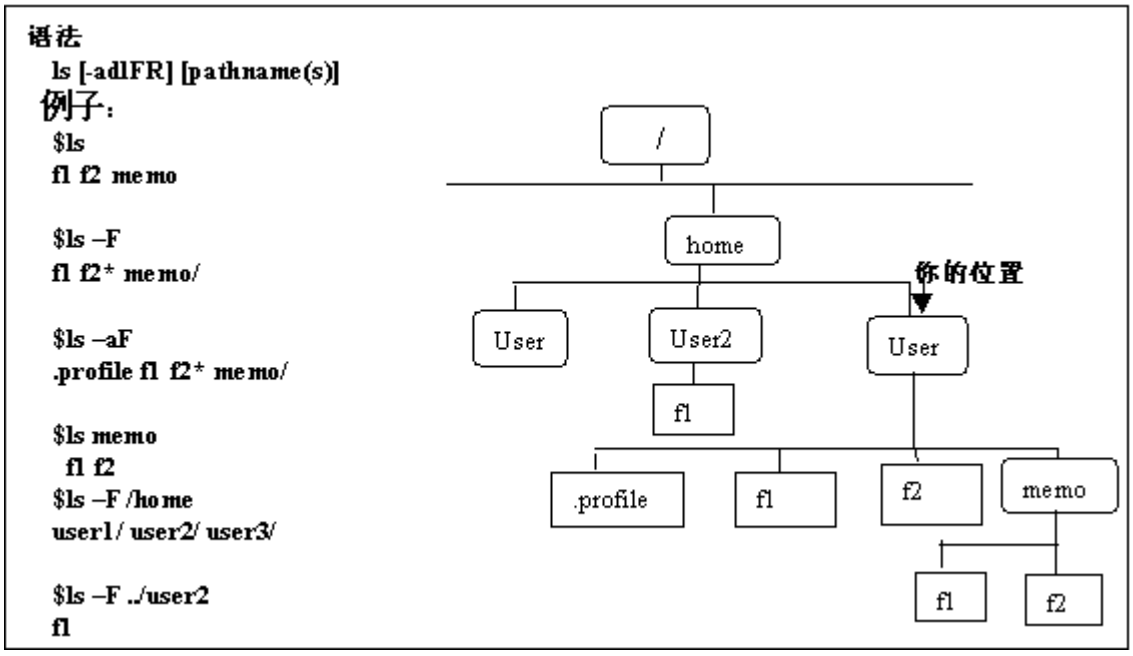
- 显示你当前所处的位置的目录名
- 了解当前目录下有什么文件和目录。
- 使你进入文件系统中的另一个目录中去
- 创建一个目录
- 删除一个目录

在这一节中，我们不会去操作一个目录中的文件，我们只操作目录。

3.7 pwd—Prenset working Directory （呈现工作中的目录）

无论何时你登录你的 UNIX 系统，你都会被置于文件系统中的某个目录下，这个目录通常被认为是你的工作目录。
pwd 命令会报告你当前在 UNIX 文件系统中位置的绝对路径名，pwd 是英文呈现工作目录的一个缩写。
由于 UNIX 系统允许用户在文件系统中任意地移动位置，所有的用户都依靠这个命令来确定他们在系统中的位置。新用户在文件系统中移动的时候，会频繁地使用这个命令来了解他们的当前的位置

3.8 ls —列出目录中的内容



ls 命令的作用是列出目录和文件的名称

如果没有参数，ls 列出当前目录下的文件和目录名。

ls 也可以接收指定文件或目录的相对和绝对路径名作为参数。当文件的路径被提供给 **ls，ls** 会报告指定的文件的信息。当一个目录的路径被提供给 ls ，ls 会显示指定目录中的内容。

ls 支持许多的选项。这些选项提供附加的信息。一个命令行可以支持多个选项，以显示更多更全的文件或目录的信息。其中常用的选项在下面列出：

-a 列出所有的文件，包括以点（.）开头的文件，通常，这些文件是隐藏的，除非 使用 – a 选项才会显现出来，这些以点开始的文件通常记录你用户线索 和应用的配置信息。

-d 列出目录的描述，而不是列出目录的内容。通常与-l 一起使用来显示目录的状态。

-l 提供一个关于每个文件德描述属性的长列表，包括类型，模式，链接数，属

主，组，大小（字节），更改日期，和名称。

-F 在每个目录后面添加反斜杠（/），在可执行文件后面添加星号（*）。

-R 递归地列出给出的目录和所有子目录中的文件。

例子：

$pwd

$/home/user3 绝对路径作为一个参数

$ls – F .. 相对路径作为一个参数

user1/user2/user3/

$ls – F ../user1 相对路径作为一个参数

f1

$ls – l memo 一个目录的相对路径作为一个参数

-rw-rw-rw 1 user3 class 27 Jan 24 06:11 f1

-rw-rw-rw 1 user3 class 37 Jan 23 19:03 f2

$ls –ld memo 显示目录 memo 的信息

drwxr-xr-x 2 user3 class 1024 Jan 20 10:23 memo

$ls –l f1 f2 多个参数，文件的相对路径

-rw-rw-rw 1 user3 class 27 Jan 24 06:11 f1

-rw-rw-rw 1 user3 class 37 Jan 37 19:03 f2

$ls –R 子目录的递归列表

./memo:

f1 f2

$ls user2

user2 not found 当前目录中不存在 user2

HP-UX 中的特例：

UNIX 系统的命令	HP-UX 等价
ls –F	lsf
ls –l	ll
ls –R	lsr

3.9 cd —Change Directory(改变目录)

将树形结构想象成为一个显示你系统中的所有的目录和文件的位置的一个道路交通图。你通常在一个目录中， **cd** 命令让你改变目录，并且移动到层次结构中的其他位置。

语法：

cd path_name

其中，路径名是你想要去的目录相对或绝对的路径名。当执行 cd 命令不带参数时，你会回到你的 HOME 目录，所以，如果你在目录中迷路，简单地键入 cd 会让你回到 HOME 目录。

注释：当使用 cd 命令在文件系统中移动时，切记要经常使用 pwd 命令来确认你所处的位置

POSIX Shell 中的 cd 命令

POSIX shell 记录有你上一次进入的目录的位置，cd 命令同样有改变目录到你期望的目录的功能。但它还有一些附加的特征可以减少你的键入。

cd 命令有一个你先前进入的目录的储存器，（存储在环境变量 OLDPWD 中），这个目录可以通过 cd - 来存取。

```
$pwd

/home/user3/tree

$cd /tmp

$ pwd

/tmp

$cd - 让你进入到先前进入的目录

$pwd

/home/user3/tree
```

3.10 find 命令

语法：

find path_list	**expression**	执行一个文件系统的有规律的查找
路径列表	表达式	路径列表示查找路径的一个列表 表达式指明查找的标准和行为。

例子：

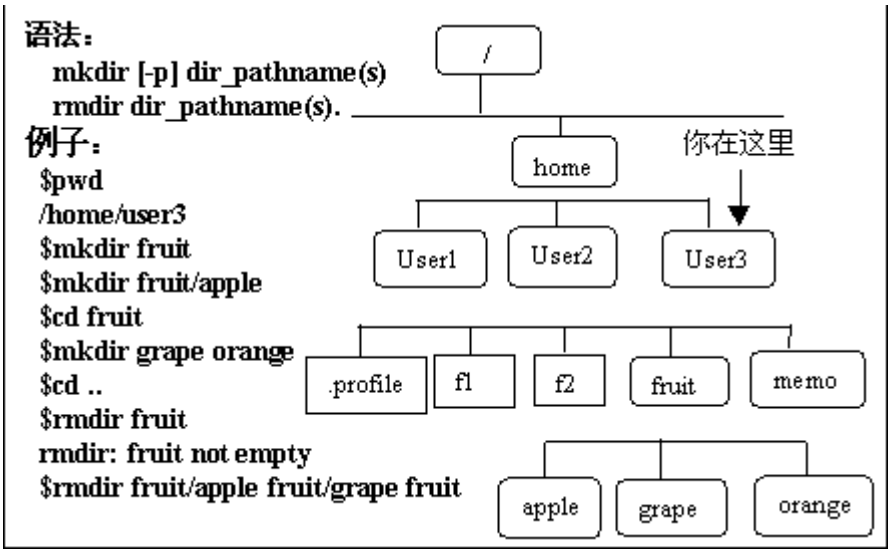
```
$ find . –name .profile
./.profile
$
```

find 命令是在文件系统中执行自动查找的唯一个命令。这个命令执行非常慢，并且会耗费许多 CPU 资源。因此建议不要经常使用。

路径列表是一个路径名的列表，典型的情况是一个目录的情况。通常是点文件。在查找一个文件时会被递归地搜寻路径名，查找符合表达式定义的文件。一个最普通的查找任务就是显示出匹配的路径名。

表达式由关键字和参数组成，参数能指明查找标准和查找一个匹配的任务，一件使查找操作更复杂的事情是表达式中使用的关键字都以－开头，这样看上去参数在选项之前似的

3.11 mkdir 和 rmddir 创建和删除目录



mkdir 命令允许你创建一个目录，这些目录能被用来帮助组织我们的文件。每当一个目录被创建，两个子目录会被自动地创建，它们是代表当前目录的点（.），和代表父母目录的点点（..），请*注意*，创建目录不会改变你当前在文件系统中的位置。

在默认的情况下，创建目录时指明的相对或绝对路径中的所有的中间目录都必须是已存在的目录，但另外一种情况，你可以使用一下的选项：

-p 如果中间目录不存在，将会创建这些目录

-m mode（模式）在创建了指定的目录后，目录被设置成默认的权限。

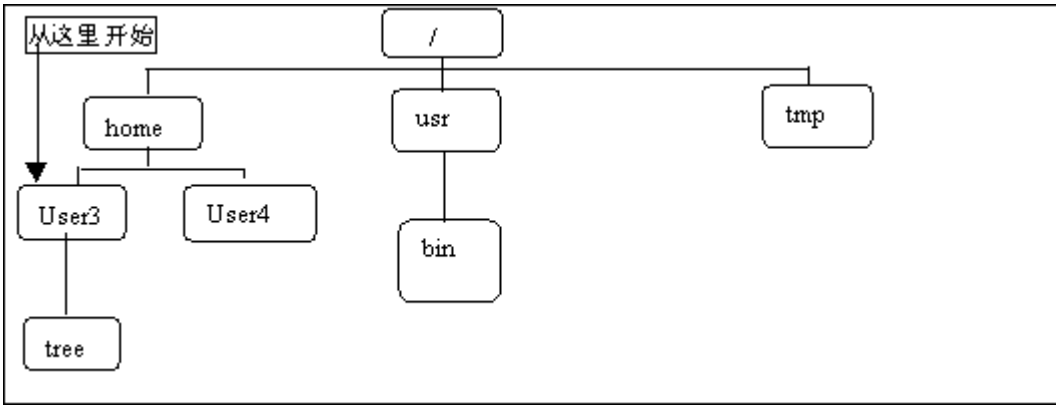
以下的命令创建 fruit 目录 ，假设这个目录并不存在。

```
$mkdir  –p fruit/apple fruit/grape fruit/orange
```

rmdir 命令让你删除一个目录，被删除的目录必须是空的（也就是除了点和点点外没有其他的东西），同时，你不能删除在你当前目录和根目录之间的目录。

两个命令都可以有多个参数，**mkdir** 的参数代表的是新的目录名，**rmdir** 的参数必须是已经存在地目录名。同时，任何用目录名和文件名作为参数的命令都可以使用绝对路径和相对路径。

3.12 复习



使用上图来复习 cd 和 pwd 命令的用法，和绝对路径和相对路径的用法

在上图中，如果你从目录 user3 开始，每一个 cd 命令的结果会使你处于上什么位置？

```
$pwd        /home/user3

$cd ..

$pwd        ......................

$cd usr

$pwd       ........................

$cd /usr

$pwd       ........................

$cd ../tmp

$pwd       ........................
```

```
$cd .

$pwd        ..............................
```

3.13 文件系统—总结

文件	一个数据的容器
目录	文件和其他目录的容器
树形结构	UNIX 系统的一种层次化的结构
路径名	文件和目录在层次结构中的位置的一种确认标识。
HOME	代表你登录目录的路径名
pwd	显示你当前在系统中的位置
cd	更改你当前的位置到其他的目录中去
ls	列出目录中的内容
find	查找指定的文件
mkdir	创建目录
rmdir	删除目录

第四章—管理文件

目标

完成这一章，你能做以下事情：

- 使用普通的 UNIX 系统文件操作命令。
- 解释使用排队打印队列系统的作用。
- 认识和使用同系统交互的排队打印命令。
- 监视排队打印系统的状态。

4.1 什么是文件

一个数据的容器或者是一个设备的链接

- **每一个文件有一个名字，文件可能保存有存储在磁盘上的数据。**
- **有几种不同类别的文件：**

－ 普通文件
－ 文本文件，数据文件，图片
－ 可执行的程序
－ 目录
－ 设备文件

UNIX 系统中的每一事物都是一个文件，包括：
普通文件 文本，邮件信息，数据，图片，程序源代码
程序 可执行的程序，例如 **ksh,who,date,man**,和 **ls**
目录 特殊的文件，记录有所包含的文件和目录的名字和文件系统中的标识。
设备 一种特殊的文件，提供到硬件设备的接口，包括有磁盘，终端，打印机，内存

简单地说，文件就是联接存储在一个存储设备（通常是磁盘）上的数据的一个名字，
在 UNIX 系统受到关注之前，文件仅仅是一个数据字节流的定义。没有预定义的记录，域，记录结束标志，文件结束标志。这为应用开发人员提供了许多的灵活性，他们可以定义自己的内部文件的描述。

普通文件通常包含有 ASCII 码的字符，典型的普通文件的创建是在终端上使用一个文本编辑器。

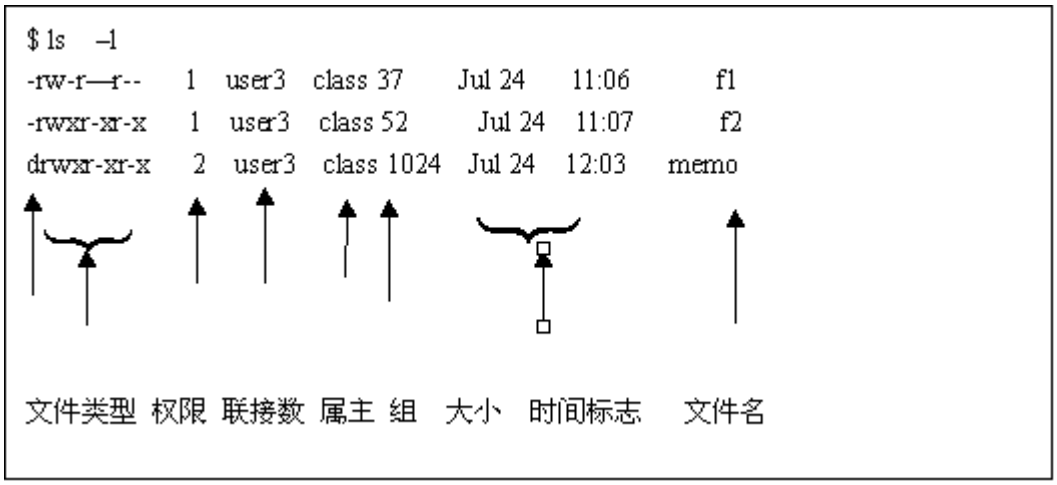
程序文件是包含可执行指令的普通文件。通常包含在终端上不能显示的编译过的代码 如（mail,who,date），或者是可以显示到终端上的 UNIX 系统 shell 命令，（通常是 shell 脚本）

目录是一个特殊的包含它下面的文件和目录的名字的文件，目录同时也存储了每个条目的 inode （i 节点号），inode 号标识有文件信息和数据在文件系统中存储的地址。（*注意*：目录不是普通的文本文件）
设备文件是一种特殊的文件，它提供了内核与实际硬件设备的接口，由于这些文件是作为接口存在的，它们不会存储有任何实际的数据。它们一般都在/dev 目录下，你的计算机的每一个需要联接的硬件设备都会有一个设备文件相对应。

2. 我们能够对文件进行什么操作？

ls 查看一个文件的描述
cat 查看一个文件的内容
more 查看一个文件的内容,一次一屏
lp 打印一个文件
cp 制造一个文件的一个拷贝。
mv 改变一个文件或者目录的名字
mv 将一个文件移动到另外一个目录中去
ln 为一个文件创建另外一个文件
rm 删除一个文件
在 UNIX 系统上的大多数活动是围绕文件和目录进行的，有许多的命令来操作文件和目录。
你已经知道一些目录操作的命令，在这一章中，我们会介绍其他的用于文件和目录的命令。
你也许会创建文件和处理文件中的内容。这通常可以使用编辑器来完，例如 vi.。

4.3 文件的描述



一个文件有几个相关的描述，可以用 ls -l 来显示这些描述
类型 普通文件或是特殊文件
权限或模式 文件的存取定义
链接数 与一个独立的数据集相关联的文件名的数目
属主 文件所有者的用户标识
组 文件存取得组的标识

大小 文件包含的字节数
时间标志 最后更改的日期
名字 最大 14 个字符（如果长文件名支持，最大 255 个字符）

文件名规范

- 最大 14 个字符
- 最大 255 个字符，(如果长文件名支持)。
- 一般由字母（a-zA-Z）,数字（0-9），点（.），短横线（-）和下划线（_），组成。

有许多其他的字符在 shell 中有“ 特殊 ”的意义，例如空格和反斜杠，所以你一般不能用这些字符组成一个文件名。其他的特殊字符有，*,<,>,\,$,|。如果你试图用这些字符组成一个文件名，你经常会得到不能预料的结果。
两个单词组成的文件名一般由下划线来联接：
$cd a dir 错误的语法
cd 看到两个参数
$cd a_dir 合法的语法
cd 只看到一个参数
在 UNIX 系统中，点（.）是一个普通的字符，所以,它能出现在文件名的任何位置，文件名 a.bcdefg，a.b.c.d，和 a…b 都是合法的文件名。点只有一点特殊，就是作为一个文件名的第一个字符的时候,在这种情况下它指明这是一个隐藏文件。你可以使用 ls –a 来显示一个隐藏文件。

文件类型

UNIX 系统支持很多的文件类型，使用 ls –l 输出的第一个字符表示文件类型。普通的文件类型包括：

- 普通文件

d 目录
l 链接文件
n 网络专用文件
c 字符设备文件（终端，打印机）
b 块设备文件（磁盘）
p 命名的管道（一种内部过程通信通道）

3. cat 显示一个文件的内容

cat 命令是用来联结（concatenate），和无缝地显示一个文件的内容。它不对文件的输出进行格式化,包括在一个文件的末尾和下一个文件的开始之间没有分割符。

语法：

cat [file…]
一个典型的 cat 命令的运用是查看单个文件的内容。例如：

cat funfile
这会将文件 **funfile** 的内容输出到屏幕上，然而，如果这个文件超过终端屏幕显示的大小，文件的文本会快速地向下滚动使你几乎看不清楚内容。所以我们需要一个更加聪明的方法来在屏幕上显示文件的内容。
当 **cat** 命令不带任何参数使用的时候，它会等待你从键盘输入，就和你使用 **mail**，**write** 命令一样，回车加，ctrl +d 用来结束输入，一旦你的输入结束，你输入的文本会在屏幕上显示出来。
注意： 如果文件包括控制字符，例如一个已编译的程序，你 **cat** 这个文件到你的终端，你的终端可能会失效。你可以用以下一种方法重新设置你的终端：
方法 1：

1. 试图退出登录—回车后使用 **exit** 命令。
2. 开关你的终端—关掉然后又打开。
3. 重新登录—你能登录继续正常工作。

方法 2：

1. 敲入 break 键。
2. 同时按下 shift+ctrl+reset。
3. 回车
4. **Tset –e –k**
5. **Tabs**

另外，你的系统管理员可以终止你的终端对话。

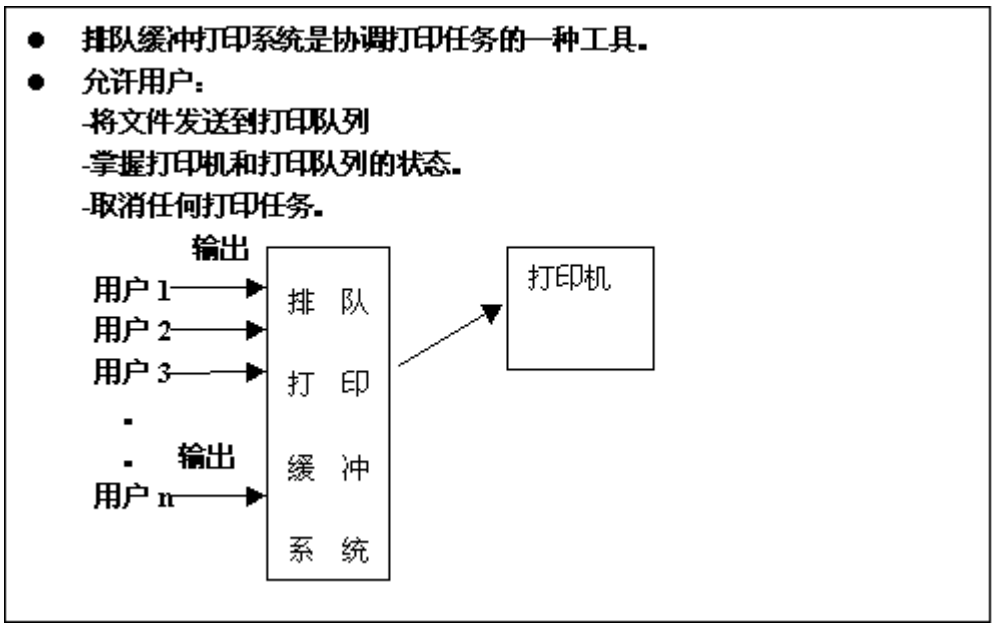
3. more – 显示一个文件的内容

more 命令显示出文件的内容，它一次仅仅只能显示一屏。要看下一屏，敲一下空格键。要看下一行，敲一下回车键。想要退出 more 命令，敲一下 q。

more 命令支持许多其他的特征，你可以使用手册来查找其他有用的用法的详细说明。

4. tail--- 显示文件的末尾

语法：
tail [-n] [文件名]
tail 命令的作用是显示一个文件的最后 n 行。（***注释***：如果没有指定，n 的默认值是 10），这个命令对那些周期性添加信息的长的日志文件特别有用。使用 tail 命令，你可以直接看到已记录的最后的信息，而不需要用 cat 和 more 来滚动整个文件到末尾。
4.7 排队打印缓冲系统



UNIX 操作系统提供一种称为排队打印缓冲系统的工具用来配置和控制你系统中的打印。lp 缓冲区的机制是：接受系统中的所有用户打印请求，然后配置打印机，让打印机一次打印一个请求。想一想如果我们没有一个打印缓冲，每一次一个用户想要打印一个文件，他或她不得不先确认当前没有其他的人在打印文件。而两个用户不能同时使用同一个打印机。

lp 缓冲系统有许多的特性，这些特性让打印平稳进行，并最大限度地减少管理员的干预。你提交打印请求到 lp 打印缓冲区，在那里打印请求会在一个队列里等待打印。你可以检查队列中的文件和系统的状态。如果你决定不打印一个文件，你也可以取消一个已经排入队列的打印请求

4.8 lp 命令

- **打印文件排队**
- **分派一个唯一的 ID**
- **许多参数在定制路由和打印的时候都是有效的。**

语法： lp [-dprinter] [-options] filename …

例子：

```
$ lp report
request id kp-112 (1 file)
$lp  –n2 memo1 memo2
request id is dp-113 (2 files)
$ lp  –dlaster  –t  “ confidential ”  memo3
request  –d is laser  –114 (1 file)
$
```

lp 命令让用户将文件送往打印队列来打印文件。每个使用 lp 提交的打印请求都会被分配一个唯一的任务标识号（称为一个请求 ID）。

lp 会将文件排队打印，或者读取标准输入。

最简单的 lp 的用法是提供一个文件名作为一个参数，lp 会将这个文件送到默认的打印机排队打印。

lp 命令有许多的选项让你可以定制路由和你任务的打印。
lp 命令的语法是：
lp [-d dest] [-n number] [-o option] –t title [-w] [file….]
一些 **lp** 命令的选项：
-nnumber 文件打印的重复的份数（默认是 1）。
-ddest 打印请求会被送到的打印机的名字。
-ttitle 在打印输出的标题页中打印标题。
-ooption 指定你的打印机的具体的选项，例如字体，间距，灰度，等等
-w 在文件打印完成以后，写一条信息到用户的终端。
在 lp（1） 中有一个选项的完整的列表。
在上图的第一个**例子**中，显示 lp 命令最简单的格式。我们将文件 report 送到系统默认的打印机。lp 返回一个打印请求的 ID 和提交给队列的文件的个数。这里，文件 report 已经被送到打印机 “ dp ”，打印请求的 ID 号是 dp-112。
在第二个例子中，我们我们送出两个文件 memo1 和 memo2 到打印机打印，同时我们想要两份拷贝（-n2）。
在第三个例子中，你可以指定你的打印请求会被送到哪个打印机。打印的输出将被标题为 ”confidentia 机密 ”

5.9 lpstat 命令

语法：

lpstat [-t]

- **lpstat 报告你已经送往打印机队列的打印请求。**
- **lpstat –t 报告调度表的状态，默认的打印机名，设备，打印机状态，和所有的排队打印的请求。**

lpstat 命令报告 lp 缓冲系统不同部分的状态。lpstat，当不加任何参数的时候，报告你当前送往打印的打印请求。
-t 选项显示系统中的所有的打印机的状态信息。
lpstat –t 命令告诉我们几个事情：

```
$ lpstat
rw-55 john 4025 Jul 6 14:26:33 1994
$
$lpstat –t
scheduler is running
system default destination: rw
device from rw: /dev/lp2235
rw accepting requests since Jul 1 10:56:20 1994
printer rw now printing rw-54. Enabled sine Jul 4 14:32:52 1994
rw55 john 4025 Jul 6 14:26:33 1994 on rw
rw-56 root 966 Jul 6 14:27:58 1994
```

sheduler is running 调度表（scheduler）是一个程序，负责将你的打印请求送往正确的打印机。如果调度表没有运行，任何东西都不

能打印。
system default destination ：rw rw 是默认的系统打印机的名字。如果你使用 lp 没有
加上 –d printer 的选项，你的打印请求会被送到这台
名叫 rw 的默认的打印机。***注意***你的默认的系统打印机可能有不同的名字（如 lp）。
device for rw：/dev/lp2235 这表明打印机被联结到计算机的缓冲器。
printer rw now printing rw-55 ID 为 rw-55 的打印请求正在被打印。
enabled 打印请求能够在 rw 上被打印。如果一个打印机是
disable 你可以提交请求，但是它们不会被打印直到打印机被 enabled。
剩下的这些行是被打印的请求。这些字段列出请求的 ID，排在哪个用户后打印，打印请求的大小，打印请求生成的时间。

4.10 cancel 命令

语法：

```
cancel id [id …]
cannel printer [ printer…]
```

例子：

取消一个由 lp 命令产生的打印任务。

```
$ cancel dp-115
```

取消当前在指定打印机上的打印任务

```
$ cancel laser
```

cancel 命令被用来从打印队列中删除打印请求。通过取消当前在打印机上的打印任务使下一个打印请求能被打印。你在打印非常长的文件或试图错误地打印一个二进制文件时（例如/usr/bin/cat），你可能会想要取消一个打印请求。请记住，lp 通常打印文本文件。、如果你没有指定合适的选项（例如=oraw（图象打印）），打印其他类型文件的会使打印机混乱，并且会浪费许多的纸张。
要取消一个打印请求，你必须通过给 cancel 命令一个参数来告诉打印缓冲器那一个打印请求是你想要取消的。**cancel** 的参数有两种类型：

- 一个请求的 ID（lp 和 lpstat 给出的）
- 一个打印机的名字

通过赋予 **cannel** 一个打印请求的 ID，指明的哪个打印请求就会被取消。如果你在 cancel 后面的参数是一个打印机名，当前在哪个打印机上正被打印的任务会被停止，打印队列中的下一个打印请求会开始打印。

```
$ lpstat
rw-113 mike 6275 Jul 6 18:46 1995
rw-114 mike 3349 Jul 6 18:47 1995
rw-115 mike 3258 Jul 6 18:49 1995
$ cannel rw-115
request “ rw-115 ” canceled
```

```
$ lpstat
rw-113 mike 6275 Jul 6 18:46 1995
rw-114 mike 3349 Jul 6 18:47 1995
$cancel rw
request “ rw-113 ” canceled
$lpstat
rw-114 mike 3349 Jul 6 18:47 1995
```

这个命令可以被任何用户执行以取消任何打印请求，你甚至可以取消其他人的打印请求；然而，被取消请求的用户会收到一个 mail，告诉他谁取消了他的打印请求。系统管理员可以限制用户只能取消他们自己的请求。

4.11 cp－拷贝文件

语法：

cp [-i] file1 new_fle 拷贝一个文件
cp [-i] file [file…] dest_dir 拷贝一个文件到一个目录中去
cp －r [-i] dir [dir …] dest_dir 拷贝目录

例子：

```
$ ls －F
f1 f2* memo/ note memind
cp f1 f1.copy
$ ls －F
f1 f1.copy f2* memo/ note remind
$cp note remind memo
$ ls －F memo
note remind
```

cp 命令被用来制造文件的一个拷贝。以下是使用 cp 命令的一些*注意*事项：

- 要求必须有两个或两个以上的参数—源与目标。
- 任何一个参数中都可以使用相对和绝对路径名。
- 当拷贝一个单个的文件是，目标可能是一个文件的路径，或是一个目录。如果目标是一个文件，而这个文件又不存在，它会被创建。如果目标文件已经存在，它的内容会被源文件的内容代替。如果目标是一个目录，文件会被拷贝到这个目录下去，同时文件名不变。

- -i（interactive）选项会在目标文件已经存在时给你警告，并要求你确认是否覆盖这个文件。

$ cp f1 f1.copy 在当前目录下创建一个叫做 f1.copy 的文件
$ cp f1 memo 在 memo 目录下创建一个叫 f1 的文件
$ cp f1 memo/f1.copy 在 memo 目录下创建一个叫做 f1.copy 的文件
当拷贝多个文件的时候，目标必须是一个目录。

```
$ cp note remind memo
```

一个文件不能被拷贝成自己。

```
$ cp f1 f1
cp : f1 and f1 are identical
```

一个目录可以使用-r （recursive 递归）选项被拷贝

***注意*： 在默认的情况下，cp 会覆盖已经存在的文件而不会给出任何提示！**

```
$ cp f1 note
$cat f1
This is a sample file to be copied
$ cat note
This is a sample file to be copied
```

4.12 mv 移动或是重命名文件

语法：

mv [-i] file new_file 重命名一个文件
mv [-i] file [file…] dest_dir 移动一个文件到一个目录下去
mv [-i] dir [dir…] dest_dir 重命名或是移动目录

例子：

```
$ ls  – f
f1 f2* memo/ note remind
$mv f1 file1
$ ls  – F
file1 f2$ memo/ note remind
$mv f2 memo/file2
$ls  – F
file1 memo/ note remind
$ls  – F memo
file2*
$ mv note remind memo
$ ls  – F
file1 memo/
$ ls  – F memo
file2* note remind
$ mv memo letters
$ ls  – F
file1 letters/
```

mv 命令被用来重命名一个文件或是移动一个或多个文件到另一个目录中去。以下是使用 mv 命令时的一些***注意***事项：

- 要求至少有两个参数： 源与目标。
- 任何一个参数都可以使用绝对或是相对路径。

- 当重命名一个单个的文件的时候，目标可以是一个文件的路径或一个目录。如果目标是当前目录下的一个文件，这个文件会被重命名，如果目标是一个目录，源文件会被移动到这个目录。如果文件不存在，文件会被创建。

- 如果目标文件已经存在，它的内容会被源文件所代替，如果目标是一个目录，文件会被移动到哪个目录。

- -i（interactive 交互）选项会在目标文件或目录存在的情况下提醒你，并且会要你确认是否覆盖文件或目录。

$mv f1 file1 在当前目录将 f1 重命名为 file1
$mv file1 memo 将文件 file1 移动到 memo 目录中去
$ mv f2 memo/file2 将文件 f2 移动到目录 memo，并且改名为 file2
当移动多个文件时，目标必须是一个目录。
$ mv note remind memo
当源是个目录的时候，它会被重命名为目标名。
$ mv note letter
注释： 在默认的情况下，mv 会移动或重命名已经存在的文件而不给出任何提示。

4.13 ln—链接文件

语法

ln file new_file 链接到一个文件
ln file [file…] dest_dir 链接文件到一个目录

例子：

```
$ ls –l f1
-rw-rw-r-- 1 user3 class 37 Jul 24 11:05 f1
$ ln f1 /home/user2/f1.link
$ls –l f1
-rw-rw-r-- 2 user3 class 37 Jul 24 11:05 f1
$ ls –l /home/user2
-rw-rw-r-- 2 user3 class 37 Jul 24 11:05 f1.link
$ ls –i f1 /home/user2/f1.link
1789 f1 1789/home/user2/f1.link
```

链接提供了一种机制：可以用多个文件名来索引磁盘上的同一数据。在多个用户想要共享同一个文件的时候可以使用链接，但是他们宁愿选择在他们自己的目录下有这个文件。如果用户 3 修改了 f1，用户 2 在下一次存取 f1.link 的时候会看到这些改变。

注释：UNIX 系统没有限制多个用户同时存取和修改同一个文件。每一个用户的修改，都会在内存中有一份私有的映象，但是最后存盘的用户会决定磁盘上的文件的版本。有的应用程序会提醒一个用户文件已经被打开，可能限制其他的用户来存取那个已经打开的文件。

当许多的文件被链接到一起的时候，用 ls –l 显示出的链接数会比 1 要大，如果任何一个链接被删除，唯一改变的是链接数会减少，文件的内容不会改变，直到链接数减少到 0，在这个时候磁盘的空间也被释放出来。

例子：

```
$ ls –l f1
-rw-rw-r-- 1 user1 classs 37 Jul 24 11:06 f1
$ ln f1 /home/user2/f1.link
$ ls –l f1
-rw-rw-r-- 2 user1 classs 37 Jul 24 11:06 f1
$ls –l /home/user2
-rw-rw-r-- 2 user1 classs 37 Jul 24 11:06 f1
```

```
ls –I f1 /home/user2/f1.link
1789 /home/user2/f1.link 1789 f1
```

4.14 rm – 删除文件

语法：

rm [-if] filename [filename…] 删除文件

rm –r[if] dirname [filename…] 删除目录

例子：

```
$ ls –F
f1 f2 fruit/ memo/
$rm f1
$ls –F
f2 fruit/ memo/
$ rm –i f2
$f2? <user|y|
$rm fruit
rm: fruit directory
$ rm –r fruit
```

rm 命令被用来删除文件。一旦文件被删除是不可挽回的，rm 命令至少要有一个参数（一个文件名），如果指定的文件名超过一个，所有的指明的文件都会被删除。

以下是最常用的选项：

-f 强制删除文件—用户不会得到任何提示，甚至在发生一个错误的时候。

-r 递归地删除指定目录中的所有的内容。

-i 询问或交互模式，它会要求用户确认来完成删除。你的回答有 y（yes）和 n（no），回车的作用和回答 no 是一样的。

***注意*：** **通常是在极端的情况下才能使用 –r 选项。使用不正确，会删除你的所有的文件，一旦一个文件被删除，只能从备份磁带上恢复这个文件。如果你必须要用 –r 选项，请和-i 选项一起应用。**

例如： rm –ir dirname

4.15 文件目录操作命令总结

ls –l	显示文件的特性
cat	在屏幕上联结和显示文件的内容
more	格式化和在屏幕上显示文件的内容
tail	显示文件的结尾部分
cp	拷贝文件或目录
mv	移动或重命名文件或目录
ln	链接文件名
rm	删除文件或目录
lp	将打印请求送到打印机排队打印
lpstat	显示打印缓冲区的状态信息
cancel	取消在打印队列中的打印请求

第五章—文件的权限和存取

目标

完成这一章，你可以做以下的事情：

- 描述和更改文件的属主和组。
- 描述和更改一个文件的许可权限。
- 描述和建立新文件的默认权限。
- 描述如何更改文件的用户和组的身份。

5.1 文件的权限和存取

存取一个文件需要用户的身份，和与文件相关的权限。这一章会介绍如何存取文件
权限 理解文件的读，写，和执行权限
ls （ll，ls －l） 确定文件被赋予的存取权限。
chmod 改变文件的存取权限。
chown 改变一个文件的属主。
umask 改变默认的文件存取权限
chgrp 改变一个文件的组。
su 转换你的用户的身份。
newgrp 转换你的组的身份

每一个文件都属于系统中的一个用户。文件的属主控制谁可以来存取文件这个文件。文件的属主有权力允许或是拒绝其它的用户存取文件。

5.2 谁有存取一个文件的权利

- UNIX 系统合并了一个三级的结构来定义谁有权存取一个文件或目录。
  - user 文件的所有者
  - group 有权存取这个文件的组
  - other 其他的所有用户
- ls -l 命令显示有权存取文件的属主和组。

```
 $ ls  -l
-rw-r--r--   1    user3    class   37   Jul  24  11:06  f1
-rwxr-xr-x   1    user3    class   37   Jul  24  11:08  f2
drwxr-xr-x   2    user3    class   1024 Jul  24  11:08  memo
```

属主　　组

UNIX 系统为一个文件提供三层存取结构：
user 代表文件的所有者
group 代表对文件有存取权限的组
other 代表系统中的其他所有的用户
每一个文件都属于系统中的一些用户。文件的属主有完全控制权决定什么人有什么权限存取文件，属主能允许或拒绝其系统中的其他用户存取文件。属主决定什么组有权限存取他的文件，属主也能将文件给系统中的其他用户，但是一旦文件的所有权被转换，文件原始的属主不再能够控制这个文件。
由于文件属于用户，并且与组相关联，你可以使用 id 命令来显示你的身份和你对系统中文件的存取权限。
上图中的文件的属主是用户 user3，class 组的成员也可以存取这些文件，另外，user3 能够允许系统中的其他用户也存取这些文件。

5.3 存取权限的类型

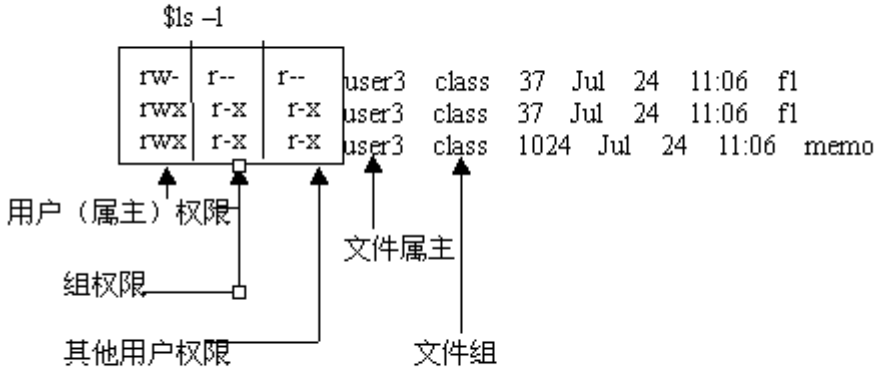
文件和目录有三种类型的存取权限：

- read（读）
- write（写）
- execute（执行）

不同的 unix 系统命令在存取程序或文件的时候会要求某种权限，例如：cat 一个文件，要求要有可读的权限，因为 cat 命令必须能够读取文件的内容，才能将其显示在屏幕上。类似，在使用 ls 命令的时候，要求一个目录要读的权限，才能够列出目录中的内容。
请注意，存取权限决定于是否你正在存取一个文件或目录。例如，一个文件的“写”的权限意味着文件的内容可以被改变。取消“写”权限会禁止用户改变一个文件的内容。但是并不保护文件不被删除。目录的“写”权限控制一个目录中的内容能否被改变。如果一个目录没有“写”的权限，它的内容就不会被改变。目录中的文件就不能被删除，添加，或是重命名。
注释：想要作为一个程序一样运行一个文件，要求文件要有读和执行的权限。

5.4 权限

权限可以用 ls –l来显示：



你对一个文件的存取权限是在以下部分定义的：你的用户身份，你的组，和与文件关联的权限设置。文件的权限在 mode （模式）中指明，文件的**模式**是一个包含九个字符的字段，其中定义了文件属主的权限，文件所属的组的权限，和其他系统中的用户的权限。

5.5 chmod 改变一个文件的权限

语法：
chmod mode_list file... 更改文件的权限

mode_list [who[operator] permission] [,...]

who 用户，组，其他所有用户
operator +(增加)，-（减），=（等于）
permission 读，写，执行

例子：
最初的权限： mode user group other
rw-r--r-- rw- r-- r--

$ chmod u+x g+x o+x file 或 $ chmod +x file
最后的权限： mode user group other
rwxr-xr-x rwx r-x r-x

chmod 命令被用来更改一个文件或目录的权限，只有文件的属主（或 root-系统管理员）才能改变权限，
为了保护一个文件不被删除和破坏，文件所在的目录和文件本身一定不能有写的权限。文件写的权限允许用户改变（或是覆盖）文件的内容，如果一个目录有写

的权限，会允许用户删除目录中的文件。**chmod** 命令支持一种字母的方式定义文件的权限。
你可以指定你想要更改的权限：
r 读的权限
w 写的权限
x 执行的权限
和你如何更改权限：
+ 增加权限
- 减少权限
= 将权限设置为
你同时可以指明你想要修改哪一组的权限：
u 用户（文件的属主）
g 组（文件关联的组）
o 其他用户
a 所有用户（系统中的每个用户）
none 分配权限给所有的域
注释：想要禁止一个文件的所有的权限，键入以下的命令：
chmod = 文件名
例子：

```
$ ls -l f1
-rw-r--r-- 1 user3 class 37 Jul 24 11:06 f1
$chmod g=rw,o= f1
$ ls  – l f1
-rw-rw-rw- 1 user3 class 37 Jul 24 11:06 f1
$ls  – l f2
-rw-rw-rw- 1 user3 class 37 Jul 24 11:08 f2
$chmod u+x,g=rx,o-rw f2
$ls  – l f2
-rwxr-x--- 1 user3 class 37 Jul 24 11:-8 f2
```

chmod 命令也支持数值的（八进制）表达式来指定文件的权限。这种表达式十分陈旧，但这是一种十分通用的格式。

1. 要改变文件的权限，你必须将权限的每一个域转变成适当地数字表达式。对属主，组，和其他用户都有存取定义。每种类型的存取赋值可用以下的数字来表示：

   read=4

   write=2

   execute=1

2. 将这些关联到存取权限的数字加起来。
3. 将三个值合起来的数字就是 **chmod** 命令的参数。

举个**例子**，如果你期望的权限：属主是 rw-，组是 r--，其他是---：
属主 组 其他 *转变成数字形式*
rw- r-- ---
4＋2＋0 4＋0＋0 0＋0＋0
6 4 0
因此：chmod 命令是：
chmod 640 filename
注释：想要禁止所有的权限，你可以使用一下命令：
chmod 000 file

5.6 umask 一权限掩码

语法：
umask [-s] [mode] 用户文件创建模式掩码
例子：
原先默认的权限： rw- rw- rw-
设置默认权限： rw- r-- ---
$umask g=r,o=
[-S]选项以符号的格式显示当前的文件创建的掩码，[-S]选项和符号格式在 Bourne 和 C shell 无效。
选项 **a-rwx** 是 **u-rwx,g-rwx,o-rwx** 的一个缩写。一个常用的新建文件的默认权限是 **rw-rw-rw-**,这意味着系统中的任何用户都能修改文件中的内容。新建目录的默认权限是 **rwxrwxrwx**,这意味着任何用户能更改这个目录，和删除这个目录中的任何东西。
为了保护你的创建的文件，你可以使用 **umask** 命令，这个命令会取消你创建的文件和目录的默认权限设置。组用户的写权限和其他用户的写权限大概是最重要的需要取消的权限。你指定的掩码在你登录系统时一直有效，**umask** 对已经存在的文件没有任何作用。

5.7 touch 一 更新文件的时间标志

语法：

touch [-amc] file...　　更新文件的存取和/或更改时间

例子：

```
$ ls –l
-rw-r—r--  1  karenk  users  25936  Aug  24  09:53  firstfile
-rw-r—r--  1  karenk  users  10246  Aug  24  09:53  secondfile
$touch newfile
$ ls -l
-rw-r—r--  1  karenk  users  25936  Aug  24  09:53  firstfile
-rw-r—r--  1  karenk  users  10246  Aug  24  09:53  secondfile
-rw-r—r--  1  karenk  users   0     Aug  25  10:02  newfile
$touch secondfile
$ls -l
-rw-r—r--  1  karenk  users  25936  Aug  24  09:53  firstfile
-rw-r—r--  1  karenk  users  10246  Aug  25  10:05  secondfile
-rw-r—r--  1  karenk  users   0     Aug  25  10:02  newfile
$
```

touch 命令可以用来创建新文件，空文件。如果指定的文件已经存在，touch 会更新文件的时间标志。它对文件的内容没有影响。

touch 命令有以下选项：

-a time 更改存取(Access)时间为指定的时间

-m time 更改修改(Modify)时间为指定的时间

-t time 使用指定时间(time)来代替当前时间。

-c 如果文件不存在，不创建新文件。

例子：

```
$ touch test_file1
$ ls  – l test_file1
-rw-rw-rw- 1 user3 class 0 Jul 24 11:08 test_fiile1
$ umask a-rwx,u=rw,g=r (或 umask 137)
$ umask  – S (或 umask)
u=rw,g=r,o= (或 137)
$ touch test_file2
$ ls  – l test_file2
$-rw-r----- 1 user3 class 0 Jul 24 11:10 test_file1
```

5.8 chown 更改文件的所有权

语法：

chown owner [:group] filename... 更改一个文件的属主，和可选地，组 ID

例子：

```
$ id
uid=303 (user3), gid=300 (class)
$cp f1 /tmp/user2/f1
```

```
$ls  – l /tmp/user2/f1
-rw-r----- 1 user3 class 3976 Jan 24 13:13 f1
$ chown user2 /tmp/user2/f1
$ ls  – l /tmp/user2/f1
-rw-r----- 1 user2 class 3976 Jan 24 13:13 f1
```

只有文件的属主和 root 才能更改一个文件的所有权。

只有文件的属主才能控制文件的属性和存取，如果你想要将一个文件的所有权给系统中的另外一个用户，你可以使用 chown 命令。例如，user3 拷贝他的文件 f1 给 user2。User2 对他个人的文件应该有完全的控制权，于是 user3 将/tmp/user2/f1 的所有权转换给 user2。可选地，chown 可以改变一个或多个文件的组 ID。要改变的组可以是一个数字的组 ID，或者是一个 passwd(group)文件中的登录名。

注释：一旦一个文件的属主已经被改变，只有新的属主或 root 可以更改文件的所有权和模式。

属主是一个系统认可的用户标识。文件/etc/passwd 中包含有系统中所有用户的 ID。

例子：

在上图中，在 user3 将文件/tmp/user2/f1 的所有权转换给 user2 后，他仍然有读的权限，因为 class 组中的任何成员对这个文件对都有读的权限。

5.9 chgrp 命令

语法：

chgrp newgroup filename.... 更改一个文件的组的存取。只有文件的属主和 root

更改文件的组。

例子：

```
$ id
uid=303(user3) ,gid=300(class)
$ ls  – l f3
-rw-r----- 1 user3 class 3967 Jan 24 13:13 f3
$chgrp class2 f3
$ls  – l f3
-rw-r----- 1 user3 class2 3967 Jan 24 13:13 f3
$chown user2 f3
$ls  – l f3
-rw-r----- 1 user2 class2 3967 Jan 24 13:13 f3
```

在长列表中的*组*域确定什么用户组可以存取这个文件。你可以用 chgrp 命令来更改。

*new_group(新组)*是一个系统认可的组的标识，在文件**/etc/group** 中有系统中所有用户的组 ID,

如果指定的新组不存在，**chgrp** 命令不会产生作用。组和组的成员是由系统管理员来控制的

注释：只有文件的属主和 root 才能更改一个文件的组。

例子：

在上图中，当 user3 将文件 f1 的组转变为 class2 组，他的存取权限没有受到影响，因为他仍然是这个文件的属主。在 user3 将文件的所有权给了 user2，他不再能够存取这个文件，因为 user3 当前的组是 class 组。

5.10 su – 转变用户 ID

语法:

su [user_name] 更改你的用户 ID 和组 ID

例子：

```
$ ls –l /usr/local/bin/class_setup
-rwxr-x--- 1 class_admin teacher 3967 Jan 24 13:13 class_setup
$ id
$uid=303(user3), gid=300(class)
$su class_admin
$passwd
$id
uid=400(class_admin), gid=300(class)
$ /usr/local/bin/class_setup
$
```

退出 su 对话线程。

```
$ ctrl + D
```

su 命令可以用来交互地更改你的用户 ID 和组 ID。 Su 是 *switch user 或 set user id* 的一个缩写。这个命令让你开启一个子进程，成为新的用户 ID 和赋予你存取与这个用户 ID 关联所有文件的存取权限。因此，出于安全的考虑，你在实际转换身份时，会被要求输入这个用户帐号的密码。

如果没有参数，**su** 命令将你转换为 *root*(系统管理员)。root 帐号有时也被称为超级用户，因为这个用户可以存取系统中的任何文件。也正是这个原因，许多人将 su 命令看成是 supper-user(超级用户)的一个缩写。当然，你必须要提供 root 密码。

注释：想要回到你原先的用户身份，不要再使用 su 命令，你只需要使用 exit 命令退出你使用 su 命令而生成的新的对话进程。

例子

在上图中，user3 没有程序/usr/local/bin/class_setup 的存取权限，因为她不是 teacher 组的成员。如果她输入命令 su class_admin，他就可以存取这个程序。作为用户 class_admin，她还可以修改程序 class_setup 的内容。当她运行完成这个程序后退出 su 的对话，就恢复自己原先的用户状态。

su – username

一些配置文件是为你的的对话线索而设立的。。当你使用命令 su username 时，你的对话特征和你原始的登录身份一样。如果你想要你的对话进程拥有转换后的用户 ID 一致的特征，你要使用短斜杠：su – username.

5.11 newgrp 命令

语法：
newgrp [group_name]　更改组 ID

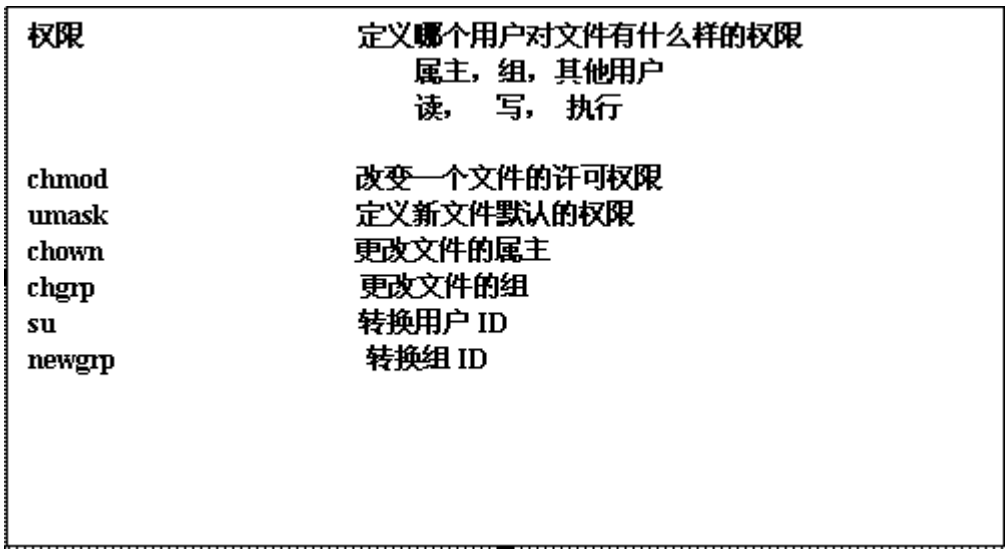
例子：
$ ls –l /usr/local/bin/class_setup
$-rwxr-x---　1　class_admin　teacher　3967　Jan 24　13:13　class_setup
$ id
uid=303 (user30)，gid=300(class)
$newgrp teacher
$id
uid=303(user3) gid=33(teacher)
$ /usr/local/bin/class_setup
$ newgrp
回到登录的组的状态
$ newgrp other
sorry
$

newgrp 命令同 su 命令十分相似。这个命令可以更改你的组的 ID 号。
系统管理员会定义你能转变的组。通过查看文件/etc/group,你能确定你有权改变到那些组。如果你没有被允许成为指定的组的成员，你会得到一条信息：Sorry.
由于 newgrp 命令没有开启一个新的对话线索，你只需要使用 newgrp 来回到你最初的组的状态。

例子：
在以上的图中，user3 仍旧无权存取程序 /usr/local/bin/class_setup，因为他被初始化定义为组 class 中，user3 能 newgrp teacher 到 teacher 组，因为系统管理员已经赋予属于他 teacher 的一员。现在他可以允运行这个程序，因 teacher 组的所有成员对为这个程序都有执行权限，但是他不能更改这个程序的内容，只有用户 class_admin 可以更改这个程序。当 user3 完成他的工作，他可以 newgrp 回到他原始的组状态。

/etc/group 文件的例子：
teacher: : 33: class_admin,user3
class: :
300 :user1,user2,user3,user4,user5,user6,class_admin

5.12 文件权限和存取－总结



关于文件权限需要注意的有：

- 文件的路径名中指出的所有的目录必须要有执行的权限，否则文件不能得到存取。
- 要保护一个文件，可以取消文件和文件所在目录的写的权限。
- 只有文件的属主（或 root）才能更改文件模式（chmod），所有权（chown），和文件的组（chgrp）

第六章－shell 基础

目标

- 完成这一章，你能够作以下事情：
- 了解 shell 的工作原理
- 描述用户登录的过程
- 描述用户环境变量和这些环境变量的功能。
- 设置和修改 shell 变量。
- 了解和修改一些特殊的环境变量例如 PATH ,TERM 等等。
- 为特定的应用定制用户环境变量。

6.1 什么是 shell?

shell 是一个交互性命令解释器。Shell 独立于操作系统，这种设计让用户可以灵活选择适合自己需要的 shell。shell 让你在命令行键入命令，经过 shell 解释后传送给操作系统（内核）执行。
这一章介绍 POSIX shell 提供的交互的特征。
以下是 shell 功能的一个汇总：

- 查找命令的位置并且执行相关联的程序
- 为 shell 变量赋新值
- 执行命令替代
- 处理 I/O 重定向和管道功能
- 提供一个解释性的编程语言界面，包括 tests,branches 和 loops 等语句
- 当你登录到一个 unix 系统，shell 会为你的终端登录线索定义一些特征，然后出现你的提示符。在 POSIX,Bourn,K shell 中$符号为默认的提示符。C shell 中默认的提示符是%号。

7.2— 一些通常使用的 shell

```
/usr/bin/sh          POSIX shell
/usr/bin/ksh          Korn shell
/usr/old/bin/sh      Bourne shell
/usr/bin/csh          C shell
/usr/bin/keysh        A contest-sensitive softkey shell
/usr/bin/rksh        Restricted Korn shell
/usr/bin/rsh          Restricted Bourne shell
```

POSIX shell 、是一个 POSIX 兼容的命令编程语言和命令解释器。它可以从终端或者是一个文件中读取并且执行命令。POSIX 在许多方面同 korn shell 相似。有历史机制。支持任务控制，和其他有用的特性。
Korn shell 同 POSIX shell 十分类似，是由贝尔试验室的 David Korn 开发的。
Bourne shell :缺乏许多在 POSIX,和 korn shell 中的功能。它是由 Stephen R. Bourne 开发的，是 AT&T unix 中最先使用的 shell

C shell。是一个普通的语言解释器，有命令历史机制，类似 C 语言的语法。和任务控制工具。它是由 university of California at Berkeley 的 william Joy 开发的。
Rsh 和 rksh 是 Bourne shell 和 Korn shell 的受限制的版本。受限制的 shell 在设置登录名和环境的时候比普遍的 shell 有更多的限制。受限制的 shell 使用起来就象是具有异常现象的标准的 shell。
使用受限制的 shell 的用户不能：

- 更改目录
- 重新设定 PATH 变量的值
- 在路径名中使用/符号。
- 重定向输出。

shell 特征的一个比较

特性	描述	POSIX	Bourne	Korn	C
命令历史	在缓冲区内存储命令，以便可以修改和重用	支持	不支持	支持	支持
行编辑	可以使用一个文本编辑器来修改当前的或是先前输入的命令	支持	不支持	支持	不支持
文件名自动完成	在命令行自动完成键入的文件名	支持	不支持	支持	支持
命令别名	用户可以重命名命令，自动包含命令选项，或缩短长命令行输入	支持	不支持	支持	支持
受限制的 shell	提供一个可控制的和一些有限的功能的安全性的特性	支持	支持	支持	不支持
任务控制	一个跟踪和存取后台运行的进程的工具	支持	不支持	支持	支持

6.3 POSIX shell 的特性

POSIX shell 是 unix 提供的 shell 中的一个。这种 shell 有许多 Korn shell 拥有，但 Bourne shell 没有的特征。即使你不会使用到所有的高级特征，你仍然会发现 POSIX shell 是一个非常方便的用户界面。以下是 POSIX shell 的一些特性：

- 命令的历史机制。
- 命令行重调用和编辑

- 任务控制。
- 文件名自动完成。
- 命令的别名。
- 增强的 cd。
- 先进的编程能力。

6.4 别名

别名就是命令的一个新的名称。使用**别名**可以缩短长命令行输入，创建新的命令，或用一种称为别名的新的命令来替代原始的命令执行。别名的组成可以是一个字母或一个短的单词。例如，许多的人非常频繁地使用 **ps –ef** 命令。如果你使用 **psf** 代替这个命令是否会更容易一些？你使用 **alias** 命令可以创建别名：

$ alias name= string

在这里，name 是你要取的别名，而 string 是 name 要取代的命令或者字符串。如果 string 包括有空格，你要需用引号将整个字符串括起来。别名对于减少键盘输入，减少打字错误，或是创建新的命令会很方便。

别名在使用的时候和其他的命令一样。 执行一个真的的 unix 系统命令和执行引用到一个 unix 命令的别名对于用户来说是透明的，没有任何区别。

别名经常用于全路径名的一个缩写。

如果不带差数，alias 命令会报告当前定义的所有的别名。

想要列出一个特定的别名，使用 alias name 命令。

禁止一个别名，可以使用 unalias 命令，语法是：

unalias name

例子：

```
$ alias go= ’ cd ’
$ alias there = /home/user3/tree/ford/sports
$ go there
$pwd
/home/user3/tree/ford/sports
```

6.5 文件名自动完成

例子：

```
$ more fra esc esc

$ more frankenstein

.

.
```

```
$ more abcdef esc =
1) abcdefx1mnop

2) abcdefy1mnop

$ more abcdef
```

然后打 x 或 y，然后连续按两下 esc 键
系统就会自动完成相关联的文件名。

当你想要存取一个长文件名的时候，你可以使用文件名自动完成功能，如果你输入的字符足够多,能唯一标识一个文件名后,你可以连续敲两下 esc 键，POSIX shell 就会自动补全文件名的剩余部分。如果输入的字符串并不唯一，POSIX shell 就不能解析到文件名，你的终端会发出蜂鸣。
直到没有文件名冲突时候，连续敲两下 esc 键，shell 会自动完成文件名。你也可以通过敲一下 esc 键来列出所有可能的文件名。在 POSIX shell 列出了有效的选项的时候，你就能够使用编辑命令来增加后来的字符来唯一地确定你所需要的文件名，然后你可以连续敲两下 esc 键来完成文件名。
文件名自动完成可以在所有的使用文件名路径的时候使用，例如：
$cd tr esc esc do esc esc r esc esc
会显示以下的命令行：
$cd tree/dog.breeds/retriever

6.5 命令的历史机制

语法：

history [-n| a z] 显示以前输入的命令。

例子：

$ history –2 列出最近输入的两个命令

15. cd
16. more .profile

$ history 3 5 列出命令号从 3 到 5 的所有的命令

3. date
4. pwd
5. ls

POSIX shell 保留一个历史文件来存储你所键入的命令，你可以重输入这些命令。这个历史文件会在你登录时建立，在退出登录时被删除。
history 命令会显示你最近输入的 16 个命令,每一个命令前面都会有一个命令号。你可以通过这个命令号来引用你以前键入的命令。
你可以键入 history –n 来显示少于或者多于 16 个命令。其中 n 代表命

令号。
你可以键入 history a z 来显示一个命令号的范围。其中 a z 代表命令号的一个范围。
HISTSIZE 变量定义了保存的的历史命令的数目。(默认值是 128) HISTFILE 变量定义了保存历史命令的文件的一个文本文件名（默认值是.sh_history）.

6.6 重新输入命令

你可以通过简单地键入
r c
来从命令历史中运行任意的命令。
其中 c 代表命令号。你也可以键入命令的第一个字母来执行你最近键入的命令，例如：

```
$ history
1 date
2 cat file1
3 ls  – l
$ r d
Mon Jul 4 10:03:13 1994
```

6.7 用户环境

语法：

env

例子：

```
$ env
HOME=/home/gerry
PWD=/home/gerry/develop/hasics
EDITOR=vi
TERM=70092
......
PATH=/usr/bin:/usr/contrib/bin:/usr/local/bin:/home/gerry/bin
```

用户环境记录了关于用户程序的线程的许多信息。 你的环境中的信息有：

- 你 home 路径的路径名
- 电子邮件的存放地址。
- 你所在地的时区
- 你登录的用户名
- shell 查找命令的路径
- 终端类型
- 你的应用可能需要的其他东西。

例如：命令 vi 和 more 需要知道你的终端类型才能以正确的格式输出数据。
与用户环境类似的是你的办公环境。在办公室里的灯光，声音，和温度对所有的工作者都是相同的。而其他的对你来说唯一的因素组成了你的特殊的环境。这些因素包括你正在执行的工作，你的写字台的布局，和你和办公室其他人的关系。你的工作环境就像你的用户环境一样都是唯一的。
许多应用需要你按照一些方式来定制你的环境。这种定制是通过修改你的.profile 文件来完成的。
你可以运行 env 命令来检查你的环境。这个命令会显示你的环境中的每一个变量和变量的值。
每一个环境变量的设置都有一个的意义，以下是一些普通的环境变量和它们的含义：

TERM,COLUMNS,LINES	你使用的终端的类型。
HOME	你的 home 目录的路径
PATH	查找命令的位置的一个目录的列表。
LOGNAME	登录时使用的用户名
ENV,HISTFILE	特殊的 POSIX shell 变量
DISPLAY	特殊的 x windows 变量

在这些变量中，一些是系统设置的变量，其他都是在/etc/profile 或.profile 中设置。

6.8 设置 SHELL 变量

- **shell 变量是一个名称，代表一个值**
- **与这个名称管关联的值可以被修改**
- **一些 shell 变量是在登录的过程中定义的。**
- **用户可以自己定义新的 shell 变量**

语法：

```
name=vales
```

例子：

```
$ PATH=/usr/bin/X11:/usr/bin
```

shell 变量同代数中的变量很相似。它是代表一个数值。给一个变量赋值就是将一个数值分配到一个变量名上。然后就可以通过这个变量名来存取

这个数值。如果这个值被修改，仍然能够通过变量名来存取新的值。给 shell 变量赋值的语法是：

name=value

你可以在终端的 shell 提示符下输入,或者在 shell 脚本中加入这条语句。注意，在等号（＝）的前后都没有空格。区别 shell 变量的名称和 shell 变量的值十分重要。当一个赋值语句执行的时候，变量的值才会被设定

例如

TERM＝70092

这会告诉 shell 记住 TERM 这个名称,并且在 TERM 变量的值被要求的时候，返回 70092。

变量名限制

变量名必须由字母开始可以包含字母，数字。下划线，变量名所包含的字符的数量没有限制。

6.9 两个重要的变量

PATH 变量

－shell 搜寻你键入的命令所在位置的一个目录的列表

TERMb 变量

－描述你的终端类型和屏幕尺寸（为你运行的程序需要）

```
$ env
. . .
PATH=/usr/bin:/usr/contrib/bin:/usr/local/bin
$TERM=70092
$
$ tset
Erase is Backspace
kill is Ctrl＋U
$
```

PATH 变量 是 shell 在寻找命令所要查找的路径的一个列表。它使我们可以简单地键入一个命令名而不需要在前面加上一个完整的一个路径列表。（例如，vi 代表/usr/bin/vi）。以下是一个默认的 PATH 变量的例子：

PATH=/usr/bin:/usr/contrib/bin:/usr/local/bin

这意味着当你键入一个命令的时候，shell 会在路径/usr/bin 中查找，然

后是/usr/contrib/bin,等等，直到找到这个命令，或者找完这些目录都没有发现这个命令。如果没有找到这个命令，会有一个 **command: not found** 的错误信息出现在屏幕上。

TERM 是一个描述你的终端类型的环境变量。为了许多的命令能正确运行，它们需要知道你使用的终端类型。例如，ls 命令需要屏幕有多少列，更多的需要知道屏幕有多少行，vi 为了正确地工作，同时需要知道你的终端的行与列的数目和关于你的终端更多的信息。终端的类型被设置成为终端的型号的数字。（例如 2392，70092，等等）

设置终端变量的默认方法是使用以下的方法：

TERM=(hp)

你的系统管理员也许已经设置好你的系统，所以系统不会询问你的终端类型。在这种情况下你可以使用 env 命令来检查 TERM 环境变量的值。如果你使用的工作站只有一个显示器，TERM 变量的值就很可能是正确的，不需要更改。

如果你的终端在你使用 more 或者 vi 的时候出现异常,请检查 TERM 变量，如果设置正确，执行 tset 命令。这个命令会使用 TERM 变量中的值来重新设定你的终端类型。

6.10 常用的变量赋值

黑体字为你要定义的变量。

EDITOR=/usr/bin/vi 使用 vi 作为行编辑器

ENV=$HOME/.shrc 在 shell 启动的时候执行$HOME/.shrc 文件

FCEDIT=/usr/bin/vi 在先前的命令行开始 vi 编辑线索。

HOME=/home/user3 指明你的登录的目录

~(波浪符号) 在 POSIX shell 中等同于你的 HOME 目录

HISTFILE=$HOME/.sh_history 存储所有的键入的命令的文件

LOGNAME=user3 指明你的登录标识或是用户名。

MAIL=/var/mail/user3 指明你的系统邮箱

OLDPWD=/TMP 指明你所进入的前一个目录

PATH=/usr/bin:$HOME/bin 指明查找命令的路径。

PS1= 指明你的主提示符。

PS1=’[!] $’ 在提示符中显示命令的行号

PS1=’$PWD $’ 在提示符中显示当前的工作路径（注意：必须使用单引号。

PS1=’[!] $PWD $’ 在提示符中显示行号和当前工作目录

PWD=/home/user3/tree 指明你的当前所在的目录。

SHELL=/usr/bin/sh 指明你的命令解释程序

TERM=2392a 指明你使用的终端的类型，启动的时候使用命令 eval ‘tset -s -Q -h‘ 系统会自动读取/etc/ttytype 文件,并使用恰当地终端类型来映射你的终端的端口。这在你的系统联接有不同

型号的　　终端的时候十分有用。
TMOUT = 300　　如果在指定的时间内没有命令或回车键入，shell 会中止　　或者超时
TZ=EST5EDT　　定义系统时区以显示正确的时间

6.11 在登录的时候发生了什么？

当你坐下来开始在系统中工作，你会在屏幕上看见 login: 提示符。当你键入你的用户名，系统会读取你的用户名，并且提示你输入密码。当你键入你的密码，系统会检查你的用户名和密码（使用/etc/passwd 文件）。如果你键入的用户名和密码是正确的，系统会让你进入到你的 home 目录，并且系统会为你启动 shell。我们已经看到了每一次我们登录的时候发生的事情。我们的问题是－当 shell 启动的时候，系统究竟做了什么？

1 getty

a. 显示/etc/issue 中的内容

b.出现登录提示符

1. c.运行登录过程

2. Login

ａ．确认用户名和密码的正确性。

ｂ．将用户放在 home 目录下

ｃ．运行用户的 shell

3. shell

a. 执行/etc/profile 文件(POSIX,Bourne,和 Korn shell) 或者是/etc/csh.login(C shell)

b. 执行用户 home 目录下的.profile 或者.login 文件

c. 执行用户 home 目录下的.kshrc 文件(POSIX 和 Korn shell)。如果用户创建了这个文 或是　　在 ENV 变量中设置.kshrc 在.profile 文件中。

d.出现 shell 提示符

一旦 shell 开始运行,他会从系统命令文件/etc/profile 中读取命令执行.用户无论在什么时候登录和启动一个 shell,这个文件都会被读取，然后 shell 会读取你自己的.profile 文件。这两个 shell 脚本被用来定制一个用户的环境。

/etc/profile 中设置了系统中每个用户的基本的环境。而.profile 文件更多的是定制你特殊的需要。因为每一个人都会使用 /etc/profile 文件，所以系统管理员有责任维护这个文件。

当这两个程序都完成后就会出现 shell 提示符。

6.12 shell 启动文件

Korn(/usr/bin/ksh)	.profile .kshrc
Bourne(/usr/old/bin/sh)	.profile
POSIX(/usr/bin/sh)	.profile

.kshrc

Restricted(/usr/bin/rsh, .profile
/usr/bin/rksh)

C(/usr/bin/csh) .profile
.cshrc

系统在配置你的线程的时候通常需要一些环境变量（例如：PATH,EIDTOR.）。正如你已经看到的，它们必须在你每次登录的时候定义。.profile 文件和.kshrc 文件十分有效。它们是简单的 shell 脚本文件，可以用来定制你的环境变量，定义别名，在登录时执行程序。由于 POSIX shell 是起源于 Korn shell,而 Korn shell 又是起源于 Bourne shell，所以，POSIX shell 支持同样的配置文件。

.profile

任何想要定制由系统管理员提供的默认环境的用户都可以创建或修改.profile 文件。这个文件通常定义或定制了环境变量，设置用户的终端，在登录过程时执行如 date 这样的程序。用户的应用同样也可以通过在.profile 中的 exec applicationname 来启动。在这种方式下，用户不会存取到 shell 提示符，并且，当应用程序退出，用户同时会退出登录。

/etc/profile

这个文件是一个对系统中的所有用户都有效的初始化文件，前提是这些用户使用的是 Bourne,Korn,或者 POSIX shell.。系统管理员可以定制这个文件来提供所有的用户的一个一致的用户环境。普通的用户一般没有权限更改这个文件，用户定制他们自己的环境只能通过自己的.profile 和.kshrc 文件

.kshrc

POSIX 和 K shell 都有一个可选的叫做.kshrc 的配置文件。它的作用同.profile 一样是配置你的用户环境，同.profile 不同的是，.kshrc 是在每次你启动一个新的 shell 时被读取。而不仅仅是在登录的时候。这种方式允许在每次你启动一个新的 shell 地时候设置你的别名或甚至提示符。在象 X11 windows 这种环境中,你可能同时运行几个 shell，你可以使用.kshrc 文件，这样，每一个 shell 看上去都是一样的。

.kshrc 这个文件名不是一个必须的文件名。当你激活一个 shell，它会通过 ENV 这个变量来索引这个文件。这个文件通常都是.kshrc，但它也可以使用其他的名字。

```
ENV=~/.kshrc
export ENV
```

这会告诉 K shell 你使用你的 home 目录下的.kshrc 文件，现在就可以在..kshrc 中加入你的别名和命令。

如果你同时使用 Bourne 和 POSIX shell，你可以存储 POSIX shell 的环境变量到这个文件中，因为 Bourne shell 不会读这个文件。

.cshrc 和 .login

当你使用 C shell 作为你的登录的 shell 的时候，C shell 会在你的 home 目录下找一个.login 地文件。如果找到，在 shell 提示符出现之前，这个文件中的命令就会被执行。这和 POSIX,Korn,和 Bourne shell 中的.profile 的用法完全一样。如果文件存在，文件.cshrc 中的命令也会被执行。

系统在配置你的线程的时候通常需要一些环境变量（例如：PATH,EIDTOR.）。正如你已经看到的，它们必须在你每次登录的时候定义。.profile 文件和.kshrc 文件十分有效。它们是简单的 shell 脚本文件，可以用来定制你的环境变量，定义别名，在登录时执行程序。由于 POSIX shell 是起源于 Korn shell,而 Korn shell 又是起源于 Bourne shell，所以，POSIX shell 支持同样的配置文件。

.profile

任何想要定制由系统管理员提供的默认环境的用户都可以创建或修改.profile 文件。这个文件通常定义或定制了环境变量，设置用户的终端，在登录过程时执行如 date 这样的程序。用户的应用同样也可以通过在.profile 中的 exec applicationname 来启动。在这种方式下，用户不会存取到 shell 提示符，并且，当应用程序退出，用户同时会退出登录。

/etc/profile

这个文件是一个对系统中的所有用户都有效的初始化文件，前提是这些用户使用的是 Bourne,Korn,或者 POSIX shell.。系统管理员可以定制这个文件来提供所有的用户的一个一致的用户环境。普通的用户一般没有权限更改这个文件，用户定制他们自己的环境只能通过自己的.profile 和.kshrc 文件

.kshrc

POSIX 和 K shell 都有一个可选的叫做.kshrc 的配置文件。它的作用同.profile 一样是配置你的用户环境，同.profile 不同的是，.kshrc 是在每次你启动一个新的 shell 时被读取。而不仅仅是在登录的时候。这种方式允许在每次你启动一个新的 shell 地时候设置你的别名或甚至提示符。在象 X11 windows 这种环境中,你可能同时运行几个 shell，你可以使用.kshrc 文件，这样，每一个 shell 看上去都是一样的。

.kshrc 这个文件名不是一个必须的文件名。当你激活一个 shell，它会通过 ENV 这个变量来索引这个文件。这个文件通常都是.kshrc，但它也可以使用其他的名字。

```
ENV=~/.kshrc
export ENV
```

这会告诉 K shell 你使用你的 home 目录下的.kshrc 文件，现在就可以在..kshrc 中加入你的别名和命令。

如果你同时使用 Bourne 和 POSIX shell，你可以存储 POSIX shell

的环境变量到这个文件中，因为 Bourne shell 不会读这个文件。

.cshrc 和 .login

当你使用 C shell 作为你的登录的 shell 的时候，C shell 会在你的 home 目录下找一个.login 地文件。如果找到，在 shell 提示符出现之前，这个文件中的命令就会被执行。这和 POSIX,Korn,和 Bourne shell 中的.profile 的用法完全一样。如果文件存在，文件.cshrc 中的命令也会被执行。

系统在配置你的线程的时候通常需要一些环境变量（例如：PATH,EIDTOR.）。正如你已经看到的，它们必须在你每次登录的时候定义。.profile 文件和.kshrc 文件十分有效。它们是简单的 shell 脚本文件，可以用来定制你的环境变量，定义别名，在登录时执行程序。由于 POSIX shell 是起源于 Korn shell,而 Korn shell 又是起源于 Bourne shell，所以，POSIX shell 支持同样的配置文件。

.profile

任何想要定制由系统管理员提供的默认环境的用户都可以创建或修改.profile 文件。这个文件通常定义或定制了环境变量，设置用户的终端，在登录过程时执行如 date 这样的程序。用户的应用同样也可以通过在.profile 中的 exec applicationname 来启动。在这种方式下，用户不会存取到 shell 提示符，并且，当应用程序退出，用户同时会退出登录。

/etc/profile

这个文件是一个对系统中的所有用户都有效的初始化文件，前提是这些用户使用的是 Bourne,Korn,或者 POSIX shell.。系统管理员可以定制这个文件来提供所有的用户的一个一致的用户环境。普通的用户一般没有权限更改这个文件，用户定制他们自己的环境只能通过自己的.profile 和.kshrc 文件

.kshrc

POSIX 和 K shell 都有一个可选的叫做.kshrc 的配置文件。它的作用同.profile 一样是配置你的用户环境，同.profile 不同的是，.kshrc 是在每次你启动一个新的 shell 时被读取。而不仅仅是在登录的时候。这种方式允许在每次你启动一个新的 shell 地时候设置你的别名或甚至提示符。在象 X11 windows 这种环境中,你可能同时运行几个 shell，你可以使用.kshrc 文件，这样，每一个 shell 看上去都是一样的。

.kshrc 这个文件名不是一个必须的文件名。当你激活一个 shell，它会通过 ENV 这个变量来索引这个文件。这个文件通常都是.kshrc，但它也可以使用其他的名字。

```
ENV=~/.kshrc
export ENV
```

这会告诉 K shell 你使用你的 home 目录下的.kshrc 文件，现在就

可以在..kshrc 中加入你的别名和命令。
如果你同时使用 Bourne 和 POSIX shell，你可以存储 POSIX shell 的环境变量到这个文件中，因为 Bourne shell 不会读这个文件。

.cshrc 和 .login

当你使用 C shell 作为你的登录的 shell 的时候，C shell 会在你的 home 目录下找一个.login 地文件。如果找到，在 shell 提示符出现之前，这个文件中的命令就会被执行。这和 POSIX,Korn,和 Bourne shell 中的.profile 的用法完全一样。如果文件存在，文件.cshrc 中的命令也会被执行。

6.13 shell 内部命令和 unix 命令

shell 内部命令是内建在 shell 中的命令
例如：

cd

ls

pwd

echo

unix 命令存在在/usr/bin 下。
例如：

more

file

一些“ 内部 ”命令同样可以作为“ 单独 ”的命令使用。

系统使用 PATH 变量来确定 UNIX 命令的位置

一些你通过键盘键入的命令是系统中的一个文件,例如在/usr/bin 下的文件，这些命令都是 unix 命，但由许多的命令，例如 cd ,pwd ,echo 实际上是 shell 内部自带的命令。这些命令在 unix 文件系统中并不存在，这些命令就叫做 shell 内部命令。
由于 unix 命令可能存在在几个目录下,shell 必须知道到什么地方去找这些命令。PATH 变量就是 shell 查找命令的目录的位置。
Unix 命令可以与 shell 内部命令同名，为了使用这些命令，用户必须在这些命令前加上其绝对路径名，以免同同名的内部命令冲突。

6.14 查找命令－whereis

UNIX 命令主要存在在四个目录：/sbin,./usr/bin,/usr/local/bin,和 /usr/contrib/bin。Whereis 命令主要在这些目录下查找命令所在的位置。许多的用户在他们的登录目录下有自己的 bin 目录。whereis 命令不

会去查找这些路径。有时你忘记命令的路径和命令的手册的位置。UNIX 系统通过 whereis 命令来定位命令和手册在系统中的位置。
Whereis 命令接受单一的参数就是命令的名字。它返回的结果是可执行代码的位置和命令的手册页。

Shell 高级特征

目标

完成这一章，你可以做到以下事情：

- 使用 shell 的替代功能，包括变量替代，命令替代，和波浪号替代。
- 设置和修改 shell 变量。
- 将局部变量传给环境。
- 使变量对子进程生效。
- 解释进程是如何被创建的。

7.0 shell 的替代功能

在 shell 中有三种类型的替代：

- 变量替代
- 命令替代
- 波浪号替代

替代的作用是加速命令行的键入和执行

7.1 Shell 变量存储

Shell 内有两块内存区域用于存储 shell 变量，它们是：**局部数据区域**和**环境。**当定义了一个新的变量时，内存会被分配给局部数据区域，在这个区域中的变量是当前 shell 私有的，通常称为*局部变量，*任何以后的子进程都不会存取到这些局部变量。但是，子进程能够存取那些传送到环境中去的变量。

在你的登录进程过程中，有几个特殊的 shell 变量会被定义。其中大多数的变量存储在环境中：一些变量，例如 *ps1* 和 *ps2*，存储在局部数据区域。这些变量的值能够被改变，用于定制你的终端特性。

env 命令能够显示当前保存在环境中所有的变量，例如：

```
$ env

MANPATH=/usr/share/man: /usr/contrib/man: /usr/local/man

PATH=/usr/bin: /usr/ccs/bin:/usr/contrib/bin: /usr/local/bin

LOGNAME=user3
```

ERASE=^H

SHELL=/usr/bin/sh

HOME=/home/user3

TERM=hpterm

PWD=/home/user3

TZ=PST8PDT

EDTOR=/usr/bin/vi

7.2 设置 Shell 变量

语法： name=value

例子：

$color=lavender　　给一个局部变量赋值

$count=3　　给一个局部变量赋值

$dir_name=tree/car.models/ford　　给一个局部变量赋值

$PS1=hi_there　　更改环境变量的值

$set　　显示所有的变量和值

当一个用户创建了一个新的变量，例如 color ，这个变量会存储在局部数据区域中。当给一个已经存在的环境变量赋予一个新值，例如 *path* ，这个新值会代替环境中的旧的值。

7.3 变量替代

1. **语法：**

$name　　执行一个变量替代

例子：

$ echo $PATH

/usr/bin:/usr/contrib/bin:/usr/local/bin:/home/user3:.

$ echo $HOME

/home/user3

$file_name=$HOME/file1

$more $file_name

<contents of /home/user3/file1>

每一个变量都有一个关联值。当使用“ $变量名 ”来对这个变量进行引用时 ，shell 会用变量的值来代替这个参数。这个过程被称为**变量替代**，这是 shell 在执行输入的命令前执行的任务之一。在 shell 完成了所有的命令行的变量替代后，就会开始执行这个命令。因此，变量可以代替命令，命令参数，或者一条完整的命令行。这提供了一种机制来方便用户重命名哪些经常使用的长的路径名，或长的命令字符串。

例子：

上例示范了一些 shell 变量的用法。请注意，变量替代能够出现在命令行的任何位置，在一个命令行中可以有多个变量。如上例所示，一个已经存在的变量的值能够用来更新当前变量的值。

```
$ echo $PATH

/usr/bin:/usr/contrib/bin:/usr/local/bin

$ PATH=$PATH:$HOME:.

$echo $ PATH

/usr/bin:/usr/contrib/bin:/usr/local/bin:/home/user3:.

$ echo $HOME

/home/user3

$ file_name=$HOME/file1 file_name=/home/user3/file1

$ more $file_name more /home/user3/file1

<contents of /home/user3/files1>
```

echo $name 命令是最常用的方式用于显示变量当前的值。

{} 的用法

确认你有一个叫 file 和一个叫 file1 的变量。能够使用以下的语句给它们赋值：

```
$ file =this
```

```
$ file1= that
$echo $fileand$file1 寻找变量 fileand,file1
sh: fileand: parameter not set
$ echo ${file} and $file1 寻找变量 file,file1
thisandthat
```

2. 花括号被用来区分变量名和周围的文本。

7.4 变量替代（2）

3.
```
$dir_name=tree/car.models/ford
$echo $dir_name
tree/car.models/ford
$ls  – F $dir_name
sedan/ sports/
$ my_ls = ” ls  – aFC ”
$ $my_ls
./ file.1 tree/
../ file.2
$ $my_ls $dir_name
./ ../ sedan/ sports/
$ cd /tmp
$ dir_name=/home/user2/tree/dog.breeds/retriever
$ $my_ls $dir_name
./ ../ golden labrador mixed
```

在指定一个文件或目录时，使用一个绝对路径作为一个变量的值。会让你在文件系统的任何位置都可以存取你想要的文件或目录。

上例的解释：

$dir_name= tree/car.models/ford

$ echo $dir_name *echo tree/car.models/ford*

tree/car.models/ford

$ ls －F $dir_name *ls －F tree/car.models/ford*

swdan/ sports/

$ my_ls =“ ls －aFC” *使用引号让 shell 忽略空格*

$ $my_ls *ls -aFC*

./ file.1 tree/

../ file.2

$my_ls $dir_name *ls －aFC tree/car.models/ford*

./ ../ sedan/ sports/

$ cd /tmp

$ dir_name=/home/user2/tree/dog.breeds/retriever

$ $my_ls $dir_name *ls －aFC /home/user2/tree/dog,breeds/retriever*

./ ../ golden labrador mixed

7.5 命令替代

4. **语法：**

$(command)

例子：

$pwd

/home/user2

```
$ curdir=$(pwd)

$ echo $curdir

/home/user2

$ cd /tmp

$ pwd

$ cd $curdir

$ pwd

/home/user2
```

命令替代用来替代一个命令和命令行输出。命令替代的标准语法，也是 POSIX 鼓励的一种语法是：$(command).

命令替代让你捕获一个命令的输出，用它作为另一个命令的参数，或是赋值给一个变量。象在变量替代中的一样，命令替代的执行是在命令行开始之前完成的。当命令行输出包含回车换行，它们会被空格代替。

同变量替代相似，命令替代使用一个美元符号之后的用括号包围的一个命令。

所有有效的 shell 脚本都可以加入命令替代。Shell 扫描每行脚本，执行它发现的开始于一个开括号，结束与于一个闭括号的命令。

命令替代的另外一种格式是用单引号来环绕一个命令象：

‘ command ’

它和$(command) 是等价的，并且这是 Bourne Shell 认证的唯一的形式。’ command ’ 形式可以用在 POSIX 的脚本中和 Bourne Shell 的脚本中。

命令替代通常是在将一个命令的输出赋给一个变量或以后的处理时使用。通常 pwd 命令将它的输出送到你的屏幕。当你执行以下的赋值语句：

$ curdir=$(pwd) 或 $ curdir= ’ pwd ’

pwd 的输出被赋给变量 curdir。

7.6 波浪号替代

```
$ echo $HOME
```

```
/home/user3
$ echo ~
/home/user3
$ cd tree
$ echo $PWD
/home/user3/tree
$ ls ~+ /dog.breeds
collie poodle
$ehco $OLDPWD
/home/user3/mail
$ ls ~-
/home/user3/mail/from.mike /home/user3/mail/form.jim
$echo ~tricia/file1
/home/tricia/file1
```

如果一个单词以一个波浪符（～）开头，这个单词被执行一个波浪符扩充，注意波浪符扩充只在一个单词的开始才会起作用，这个意思是：/~home/user3 没有波浪号扩充的功能，波浪扩充有以下的规则：

单个地波浪号或是在/ 之前代表 HOME 变量中设置的路径名。

一个波浪号跟一个＋号代表 PWD 变量的值。PWD 的值是在 cd 到一个新的，当前的，工作目录时被设定的。

一个波浪号跟一个－号会代表 OLDPWD 变量的值。OLDPWD 变量是在 cd 前一个工作目录时被设定的。

如果一个波浪号跟一个字符串，然后是一个/ 符号 ，shell 会检查字符串是否与用户在系统中的名字一致。如果一致，～字符串会被用户登录的路径名所代替。

波浪符号也能在别名中引用：

```
$pwd
/home/user3
$alias cdn= ’ cd ~/bin ’
$ cdn
$pwd
/home/user3/bin
```

7.7 显示变量的值

```
$ echo $HOME
/home/user3
$env
HOME=/home/user3
PATH=/usr/bin:/usr/contrib/bin:/usr/local/bin
SHELL=/usr/bin/sh
$ set
HOME=/home/user3
PATH=/usr/bin:/usr/contrib/bin:/usr/local/bin
SHELL=/usr/bin/sh
dolor=lavender
```

```
dir_name=/home/user3/tree
$ unset dir_name
```
变量替代，($变量)，可以被用来显示一个独立变量的值，无论这个变量是在本地数据区域或是在环境中。
env 命令用来显示所有的当前环境中的定义的变量，和它们的值。
set 命令会显示所有的当前定义的变量，本地和环境中的变量，和它们的值。
unset 命令用来删除指定变量的当前的值。这个指被赋为空值 NULL。
set 和 **unset** 都是 shell 内建的命令，而 **env** 是 UNIX 命令 **/usr/bin/env.**

7.8 传送局部变量到环境

语法：
export 变量 传递变量到环境
传送变量 color 和 count 到环境的过程，执行了以下的命令：

```
$ color=lavender
$ export color
$ export count=3
$ export
export PATH=/usr/bin:/usr/ccs/bin:/usr/contrib/bin:/usr/local/bin
export color=lavender
export count=3
```

为了使其它的进程也能使用一个变量，这个变量必须在环境中存在。当一个变量被定义，这个变量存储在局部数据空间，必须被 **export** 到环境中去。
exprot 命令将指定的变量从局部数据空间传递到环境数据空间，**export** 变量＝值 会对变量进行赋值（也可能是更新），同时将这个变量放到环境中去。如果不带参数，**export** 命令同 env 命令一样会显示所有 exported（输出的），变量的名字和值，注意 **export** 是一个 shell 内部的命令。

7.9 传递变量给一个应用

系统中的每一个应用或命令都会有一个相关联的存储在磁盘上的程序文件。大多数的 UNIX 系统命令在目录**/usr/bin** 下面。当执行一个命令的时候，命令相关的程序文件必须被定位，再将代码装载入内存然后执行。在 UNIX 系统中一个运行中的程序被称为**进程**。
当你登录进入一个 UNIX 系统时，shell 程序会被装载，一个 shell 进程会被执行。当你在 shell 提示符下输入一个应用的（或命令）的名字后，一个子进程会被创建和执行：其过程如下：
一个 **fork**(分叉)会复制你的 shell 进程，包括程序代码，环境数据空间，和局部数据空间。
一个 **exec** 会使用子进程的代码和本地数据空间来替代原进程的本地数据空间。
exec 会在执行要求的应用进程后结束。

当子进程在执行的时候，shell (父进程)会进入睡眠状态，等待子进程结束。一旦子进程结束执行，它会中止，释放自己使用的内存，并且唤醒父

进程，父进程又可以准备接受另外的命令请求，当 shell 提示符返回到屏幕上，你就知道子进程已经结束。

局部变量 和 环境变量

无论何时定义一个新的变量，它都会存储在于与本进程相联系的局部数据空间中。如果一个子进程想要存取这个变量，这个变量必须被传送到环境中（使用 export），一旦一个变量在环境中，它对每个后来的子进程都会有效，，因为环境变量对每个子进程都有效。

在上图中，在 vi 命令之前，color 变量在 shell 的局部数据空间里，TERM 变量在环境里。当 vi 命令执行时，shell 执行一次 fork 和 exec；子进程的局部数据空间比子进程的程序代码所覆盖，但环境被完整地传递给子进程。因此，子进程 vi 不能存取变量 color，但是它可以存取变量 TERM。Vi 编辑器需要知道用户的终端类型，用来适当地格式化它的输出。它通过读取环境变量 TERM 的值来获得这些信息。

唯一的传递数据给（子）进程的方法是通过环境。

7.10 监视进程状态

```
$ ps  – f
UID PID PPID C STIME TTY TIME COMMNAD
user3 4702 1 1 08:46:40 ttyp4 0:00 -sh
user3 4895 4702 18 09:55:10 ttyp4 0:00 ps -f
$ ksh
$ ps  – f
UID PID PPID C STIME TTY TIME COMMAND
user3 4702 1 0 08:46:40 ttyp4 0:00 -sh
user3 4896 4702 1 09:57:20 ttyp4 0:00 ksh
user3 4898 4896 18 09:57:26 ttyp4 0:00 ps  – f
$exec ps  – f
UID PID PPID C STIME TTY TIME COMMAND
user3 4702 1 0 08:46:40 ttyp4 0:00 -sh
user3 4896 4702 18 09:57:26 ttyp4 0:00 ps  – f
$
```

系统中的每一个进程在启动的时候都被赋予一个唯一的号码，这个号码被称为进程 ID(**PID**)。Ps 命令显示当前在你系统中运行（或睡眠）的进程的信息，包括每个进程的 PID，每个进程父进程的 PID（PPID）。通过 PID 和 PPID，你能够追踪你系统中任何进程的体系。Ps 命令同时也会报告每一个进程的属主，终端号，和其他的有用的信息。

ps 命令通常不带参数使用，结果是给出一个与你当前终端对话相关的进程的一个简单的报表，例如：

```
$ ps
PID TTY TIME COMMAND
```

```
ttyp4 0:00 sh
ttyp4 0:00 ps
```

正如你看到的那样，这个命令显示只有 shell，**sh**,和 **ps** 命令正在运行。注意两个进程的 PID 号码。当使用**-f** 选项时候，**ps** 命令给出一个全列表，包括 PPID 号，我们可以看到 ps －f 命令是作为 shell 的一个子进程，因为它的 PPID 号同 shell 的 PID 号是一致的。

请记住 shell 同其他的 UNIX 命令一样都是一个程序。如果你在当前的 POSIX shell 的提示符下使用 **ksh** 命令，一个 **fork** 和 **exec** 会执行，一个 Korn shell 的子进程回北创建，并开始执行。当我们执行另一个 **ps －f** 的时候，我们可以发现，**ksh** 作为原 shell，sh 的一个子进程运行，新的 **ps** 命令作为 Korn shell 的一个子进程。

exec 命令是一个 shell 内建的命令。如果用我们用 **exec ps －f** 来代替 **ps －f** ，ps 的程序代码会覆盖当前的进程（**ksh**）的程序代码。很明显，这是因为 **ps －f** 的 PID 号与 ksh 使用的 PID 是一致的。当 **ps －f** 结束后，我们会发现回到了最初的 POSIX shell 提示符。

7.11 子进程和环境

```
$ export color=lavender
$ ksh (创建一个子 shell 进程)
$ ps -f
UID PID PPID C STIME TTY TIME COMMAND
user3 4702 1 0 08:46:40 ttyp4 0:00 -sh
user3 4896 4702 1 09:57:20 ttyp4 0:00 ksh
user3 4898 4896 18 09:57:26 ttyp4 0:00 ps －f
$ echo $color
lavender
$ color=red
$ echo $color
red
$ exit （退出子 shell）
$ ps －f (回到父 shell)
UID PID PPID C STIME TTY TIME COMMAND
user3 4702 1 0 08:46:40 ttyp4 0:00 -sh
user3 4895 4702 1 09:58:20 ttyp4 0:00 ps －f
$echo $color
lavender
```

上例说明了子进程不能够更改它们的父进程的环境。

```
$ ps －f
UID FSID PID PPID C STIME TTY TIME COMMAND
user3 default_system 4702 1 0 08:46:40 ttyp4 0;00 -sh
user3 default_system 4895 4702 1 09:58:20 ttyp4 0:00 ps －f
```

如果最初执行一个 ps -f 命令，它会显示只有你的登录 shell（当然还有 ps ）在运行。如上表中，我们给一个变量 color 赋值 lavender，然后将这个变量传递到环境。下一步，我们执行一个子进程。执行一个 ksh 命令，创建一个子 Korn shell 进程。当然父进程的环境已经传递给这个子 Korn shell 进程，我们可以观察到变量 color 的值是 lavender。我们然后更改变量 color 的值为 red。echo 命令确认在子 shell 的环境中变量 color 的值已经被改变。当我们退出子 shell 回到父 shell，我们发现富进程的环境并没有被子进程改变，变量 color 仍然保持原来的值 lavender.

第八章　引用

目标

完成这一章，你能够做以下事情：

- 在命令行中使用引用机制来忽略特殊字符的特殊含义

8.1 介绍引用

- **许多的字符在 shell 中有“ 特别 ”的含义：**
  - **空格**
  - **回车**
  - **$**
  - **#**
  - *****
  - **< >**
- **引用消除(忽略）这些字符的特殊含义。**

在 unix 系统中，许多特殊字符对 shell 来说都有特殊含义，例如，空格是命令和参数的分割符。回车会发送给 shell 执行命令的信号，$符号被用来显示与变量名相关联的值。
在一些特殊的情况下，你不想要 shell 去解释这些特殊字符的特殊含义。你只要求字面上的意义。因此，unix 必须提供一种机制来忽略或消除一个指定的字符的含义。这种机制就叫做引用。

8.2 引用符号

**反斜杠 **
单引号 ‘
双引号 “

反斜杠消除紧跟在它后面的特殊字符的特殊意义。

单引号（’）会消除特殊字符的特殊含义。在单引号包围之中的所有的特殊字符的特殊含义都会被忽略。单引号不能被忽略，它要被用来关闭被引用的字符串。双引号（”）的包容性要差一点，大多数的字符都可以使用双引号来去除特殊含义。只有$符号（当其被用来作为变量和命令替代的时候），和反斜杠例外。你可以在双引号中使用反斜杠来除掉　$号的特别含义

8.3 引用-- \

```
语法：

\      除去下一个字符的特别含义

例子：

$ echo the \ \ escapes the next character

the \ escapes the next character

$ color=red\ white\ and\ blue

$ ehco the value of \ $ colore is $color

the value of $colore is red white and blue

$ echo one two \

>three four

one two three four
```

反斜杠会忽略下一个特殊字符的特殊含义。没有任何的例外。

8.3 引用-‘

```
语法：

‘        除去包括在单引号中的所有字符的特殊含义。

例子：

$ color =’red white and blue’

$ echo ‘the value of \$colore is $colore’

the value of $color is $color
```

```
$ echo ‘ the value of $color is ‘  $color

the value of $color is red whie and blue

$ echo ‘ this doesn ’ t work ’

>ctrl + c

$ echo ‘ ************** ”

*************
```

单引号会去除所有的包含在其中的特殊字符的特别含义。

8.5 引用 - “

```
“        去除包含在其中的特殊字符的特别含义，但\，$，和 “ 除外。
```

例子：

```
$ color = ‘ red white and blue ”

$ echo “ the value of \$colore is $color ”

the value of $volor is red white and blue

$ cur_dir= ”$LOGNAME  –  your current directory is $(pwd) ”

$ echo $cur_dir

user3  –  your current directory is /home/user3/tree

$ echo “ they ’ re all here ,\\ ,  ‘ , \ ”  “

they ’ re all here,/,  ‘ , “
```

双引号引用不如单引号引用全面，大多数的特别字符都会被忽略，例外是你可以进行变量替代，$variable,和命令替代， $(cmd).
当你使用双引号引用的时候，你也许会想要忽略这些特殊字符的含义。因而，反斜杠（\）同时也保持了它的特殊含义，可以用它来除去特殊字符$,或 ’ 的特殊含义。前提是他们出现在双引号当中。
注意：引用机制只能在单个命令行中使用。

8.6 总结

机制	目标

反斜杠	忽略下一个字符
单引号	忽略所有的在‘ ‘ 中的字符
双引号	忽略所有的在“ ”中的字符，除了 \, $, {变量名}， 和$(comand)。

第九章 输入输出重定向

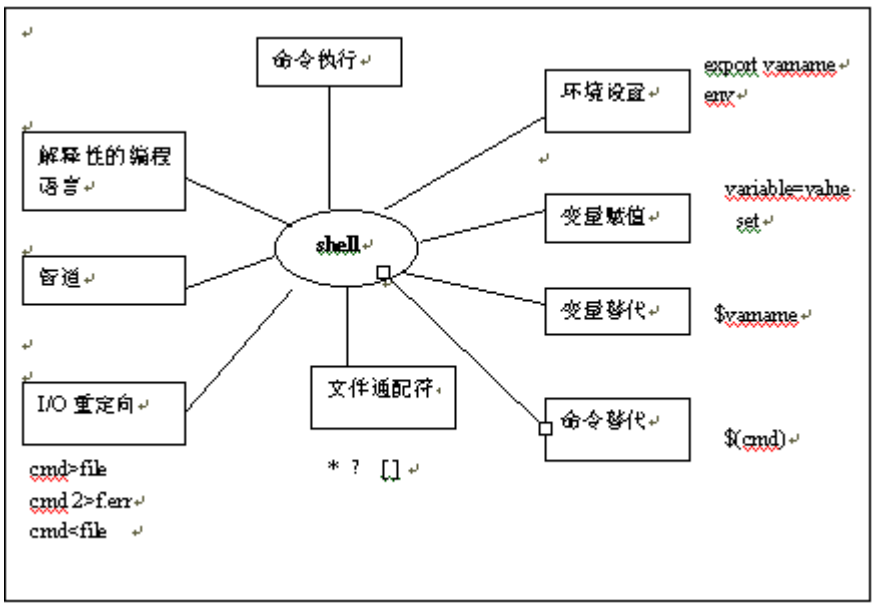
目标

完成这一章，你能够做以下事情：

- 改变 UNIX 命令的输出，使其输出到其它文件。
- 改变 UNIX 命令产生的错误信息的输出到其他文件。
- 改变 UNIX 命令的输入源。
- 定义一个过滤器

使用一些基本的过滤器例如 sort,grep 和 wc。

9.1 输入输出重定向简介



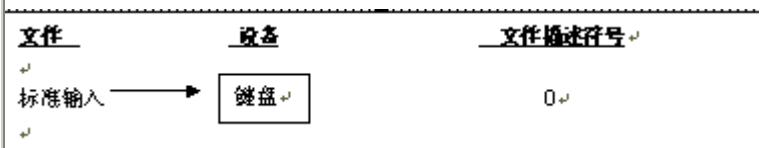
shell 提供重定向一个命令的输入和输出的功能。大多数的命令的输出是输出到终端；比如 date,ls,who 等等。一些命令从你的键盘得到输入，例子包括 mail,write,cat。

在 UNIX 系统中任何事物都是一个文件，包括你的终端和键盘。**输出重定向** 让你将一个命令的输出送到除终端以外的其他的文件中。而**输入重定向**让你从键盘以外的文件中得到输入。

输出重定向可以用来捕获一个命令的输出，作为记录日志的需要或对其进行更进一步的处理。输入重定向让你可以使用一个编辑器创建一个文件，然后将这个文件送到一个命令，而来代替没有编辑的能力的交互式的输入（例如 mail 命令）。

这一章介绍输入输出重定向，然后介绍一些 UNIX 的过滤器。过滤器是一种特殊的工具，它能进一步地处理一个文件的内容。

9.2 标准输入，标准输出，和标准错误



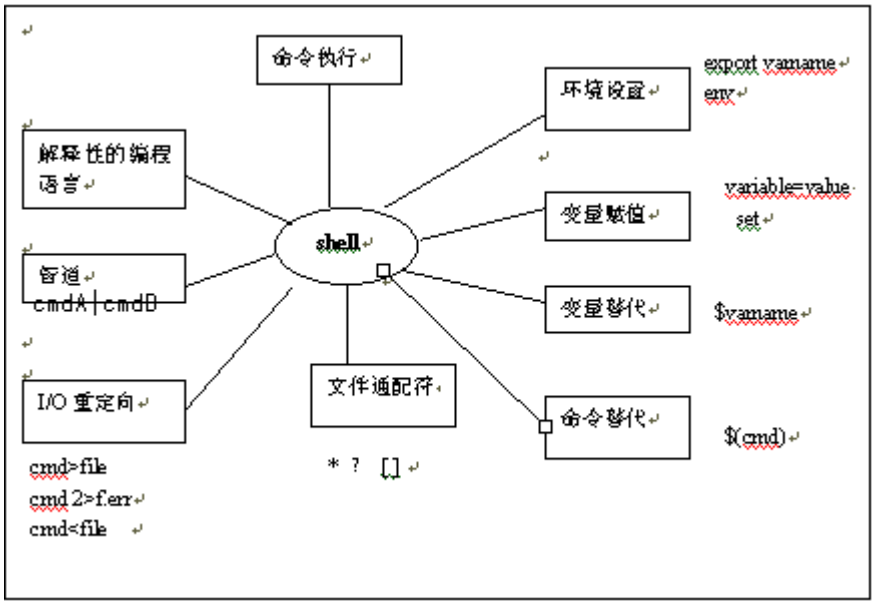
第十章 管道

目标

完成这一章，你能够做以下事情：

- 了解管道的用途。
- 建立一个管道，使其从一个命令获得输出，然后为另一个命令制造输入。
- 使用 tee,cut,tr,more,和 pr 过滤器

10.1 管道简介



shell 提供的一个有用的特性是通过管道将命令连接起来的能力。UNIX 系统操作环境的灵活性通过过滤文件的内容来体现。使用管道，你能够过滤一个命令的输出。
这一章会介绍管道，然后介绍一些过滤器（cut,tr,tee 和 pr），这样你就能对文件或命令的输出作进一步的处理。

10.2 为什么使用管道

$ who > temp_file
 	
$ wc -l < temp_file
 	
$ rm temp_file
	who | wc -l

如果你使用 I/O 重定向来完成过滤一个命令的输出的操作，你会首先重定向一个命令的输出到临时文件然后过滤这个临时文件的内容。当你完成这些操作，你又要删除这个临时文件。虽然这种方式扩展了命令的能力，但是删除临时文件的操作显得不是很方便。
管道让你直接传送一个命令的输出到另一个命令的输入。你不需要创建一个临时文件；因此，当你完成操作的时候不需要进行清除操作。
这就是 UNIX 系统的灵活和强大的具体体现。命令和命令可以被链接在一起，提高了单个命令行的处理能力。

10.3 | 符号

| 符号（读成管道符）的作用是联接两个命令。管道符左边命令的标准输出会被用作管道符右边的命令的标准输入。出现在管道中间的命令，必须能够接收标准输入和输出到标准输出。
过滤器例如 wc,sort,和 grep 接收标准输入并产生标准输出，所以，它们可以出现在一个管道的中间。通过将命令和过滤器链接到一起，使你能够执行非常复杂的过程。
以下是管道的每一个位置的命令的要求：

- 任何在一个|符号左边的命令必须能够输出到标准输出。
- 任何在 | 符号右边的命令必须能够从标准输入读取数据。
- 任何在两个 | 符号之间的命令必须能够接收标准输入并且能够制造输出到标准输出。（是一个过滤器）

more 命令
more 命令被用来在屏幕上显示文件的内容，一次显示一屏。more 命令同样也能读取标准输入。因此，它能出现在一个管道的右边,被用来控制任何命令的输出,并且产生输出到标准输出。这在一个命令在你的屏幕上产生很长的输出，你想要一次浏览一屏的时候非常有用。

10.4 管道与输入输出重定向

输入输出重定向	**管道**
语法：	
`cmd_out > file`	`cmd_out | cmd.in`
or	
`cmd_in < file`	
例子：	

```
who > who.out                who | sort

sort < who.out
```

输入输出重定向总是在一个命令和一个文件之间进行。输出重定向会捕获一个命令的标准输出，并且把它输出到一个文件。输出重定向一般用来记录日志或者长时间地存贮一个命令的输出数据。输入重定向将输入从键盘变成从文件。输入重定向执行起来不是很明确，因为大多数的命令接收标准输入，同样也接收文件名作为命令行参数（mail 和 write 除外）。但是具备输入重定向的能力是一个命令能出现在一个管道符号的右边的必须要求。
管道通常被用来联接两个命令。如果你想要使用一个接收标准输入的命令来进一步处理一个命令的输出，应该建立一个管道。输入输出重定向被用来在命令和文件之间。管道被用来在命令之间

10.5 管道中的重定向

每一个命令都有三个有效的数据流：标准输入，标准输出，标准错误。管道中的每一个命令都保留有某一个流。而没有被管道使用的流可以被重定向。
以下是一个管道的不同部分的重定向的有效性的一个汇总：

- 任何在一个管道符号的左边的命令能够被重定向输入，和错误，因为它的输出被传送到管道右边的命令。
- 任何在一个管道右边的命令能够被重定向输出，和错误。因为它的输入来自管道中前一个命令。
- 任何在管道中间的命令能重定向错误，因为它的输入来自前一个命令而它的输出要送到管道中的下一个命令。

例子：

```
$ grep user /etc/passwd | sort > sorted.users
$ grep usesr < /etc/passwd 2> grep.err | sort > sorted.users 2> sort.err
$ grep user < /etc/passwd | sort 2 > sort.err | wc  – l > wc.out 2> wc.err
```

上例中的输出会送到一个文件，在屏幕上不能看到标准命令的输出。

10.6 一些过滤器

```
cut          抽取指定的列或者字段。并且显示在屏幕上。

tr            转换字符

tee          输出到一个文件同时输出到标准输出。

pr          打印并且格式化输出到标准输出
```

过滤器如 sort,或者 grep，提供了一种灵活的机制来对命令的输出进行处理。这一章剩下的部分介绍在管道中执行三种新的过滤器。像其他的过滤器一样，这些

命令接收标准输入，所以它们可以出现在一个管道的右边，同时它们产生标准输出，所以它们同样能出现在管道的左边（或者在一个管道的中间）
cut 命令允许你从标准输入或者一个文件中抽取出文本的列或者是字段，然后将结果送到标准输出。
tee 命令允许你将命令的输出送到一个文件和标准输出。
pr 命令被用来格式化输出，通常被用来准备输出一个文件到打印机。
正如所有的过滤器一样，这些命令不会修改原始文件的内容。处理的结果会被送到标准输出。

10.7 cut 命令

语法：

```
cut -clist [file...] 从文件或标准输入抽取列或者字段

cut -flist [-dchar][-s][file...]
```

例子：

```
$ date | cut –c1-3

$ tail –1 /etc/passwd

user3:mdhbmkdj:303:30:student user3:/home/user3:/usr/bin/sh

1       2         3   4       5              6           7

$ cut –f1,6 –d: /etc/passwd |

$ cut –f1,6 –d: /etc/passwd | sort –r

$ ps –ef | cut –c49- | sort –d
```

cut 命令被用来从标准输入或者一个文件中抽取特定的列或者字段。被抽取的列或字段会被送到标准输出。-c 选项剪切列，-f 选项剪切字段。cut 命令能从标准输入或者一个文件接收输入，由于它接收标准输入，所以它可以出现在一个管道的右边。
一个数字列表被用来告诉 cut 要抽取得哪些列或字段。列的定义同 sort 命令类似。定义列或者字段列表有以下的格式：
A-B 从 A 到 B 的列或者字段

A. 从 A 到列或字段的最后一行

-B 从列或字段的开头到 B
A,B 字段 A 或者 B
以上的格式也可以组合在一起使用，例如：

cut -f1,3 5-7 /etc/passwd
会剪切出/etc/passw 文件的第一和第三列，和第五行到第七行。
默认的字段分隔符为 tab 符号。如果你要用其他的分隔符,你可以使用-d char 选项，其中 char 是分割你的输入的字段的字符。（这类似于 sort 命令的-t X 选项）。冒号是一个通用的分隔符，因为它在 shell 中没有什么特殊含义。
同样，-s 选项，在剪切字段的时候，会丢弃所有的没有分隔符的行。
例子：

```
$ cut –c1-3 回车
12345
123
abcdefgh
abc
ctrl + d
$ date | cut –c1-3
```

10.8 tr 命令

语法：

tr [-s] [string1] [string2]　　转换字符

例子：

```
$ who | tr –s “ “

$

$ date | cut –c1-3 | tr “ [:lower:] “ “ [:upper:] “
```

tr 命令的作用是转换字符。它接收标准输入也接收文件名作为参数；因此它能被用在管道中。
tr 命令可以用来转换许多连续的空白字符为一个空白字符，如上例所示。你也许注意到许多的 UNIX 命令会在它们的字段之间插入一定数量的空白。因此，当你想要在字段之间使用一个单个的分割符号的时候，在一个管道中的 tr 能够方便地进行一次前期处理，。
tr 命令同样能够被用来替代文字的字符或者转换文本的大小写，就象上例中的第二个例子。

10.9 tee 命令

语法：

tee [-a] file [file...] 分流管道

例子：

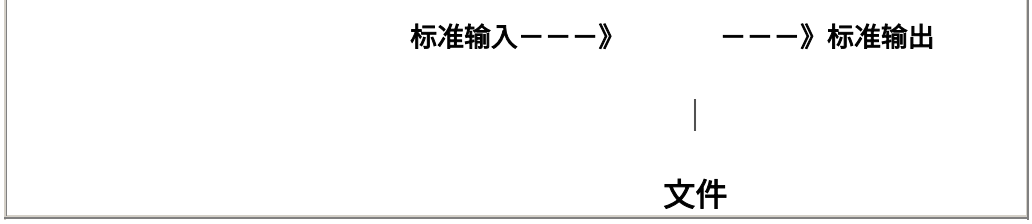
```
$ who | sort

$ who | tee unsorted | sort

$ who | tee unsorted | sort | tee sorted

$ who | wc  – l

$ who | tee whoson | wc  – l
```



通常你在执行一个复杂的管道操作的时候，管道中间的命令的输出被提交到管道中的下一个命令处理，这时你不能够看到中间命令的输出。tee 命令被用来分接一个管道。Tee 从标准输入读取数据，然后输出到标准输出，同时输出到一个指定的文件。如果使用-a 选项，tee 会在文件末尾添加输出而不是覆盖文件。
tee 命令适合在两种环境中使用。

- 从管道收集中间命令的输出：

当你在一个管道中加入 tee，你就可以捕捉中间处理，然后传送到管道的下一个命令。
发送一个命令的最后的输出到屏幕同时到一个文件。

- 这是一个有用的日志机制。你也许想要交互运行一个命令并且观察它的输出，同时要保存这个输出到一个文件。请记住，当你仅仅重定向一个命令的输出到文件的时候，在屏幕上是没有显示。所以，这个工具能被用在管道的最后，或者在任何产生输出的命令的最后。

10.10 pr 命令

语法：

pr [-option] [file....] 对标准输入格式化并且产生标准输出

例子：

```
$ pr  – n3 funfile
```

```
$ pr –n3 funfile | more

$ ls | pr –3

$ grep home /etc/passwd | pr –h “user Accounts”
```

pr 命令可以*打印到标准输出*：它被用来格式化标准输入数据流或者格式化指定文件的内容。然后输出到屏幕，而不是到打印机，pr 命令典型的执行情况是，在文件被送到打印机打印前格式化文件的内容。
pr 命令在打印长文件的时候非常有用，因为它会在每一个新页的顶部插入一个标题，其中包括文件名（或者是使用-h 选项定义的标题），和页号。
pr 命令支持许多的选项。你可以使用 man 来查找每个选项的具体用途。

10.11 管道－总结

```
管道 cmd_out | cmd_in

cmd_out | cmd_in_out | cmd_in

cut 抽取列或者字段输出到标准输出

tee 发送输出到标准输出同时到一个指定的文件

pr 在屏幕上格式化打印信息，通常和 lp 一起使用。

tr 转换字符
```

第十一章 使用网络服务

目标

完成这一章，你能够做以下事情：

- 描述 UNIX 中不同的网络服务
- 理解局域网（LAN）的功能
- 找出本地主机的主机名和局域网中其它系统的主机名。
- 使用 ARPA/Berkeley 服务来执行远程登录，远程文件传输，和远程命令执行。

11.1 局域网是什么？

局域网（LAN）是在一个小的区域中连接两台或者多台计算机。在 LAN 中通常安装有超过一台的计算机，以便让用户可以在不同的计算机上工作，而不用在不同的他们想要使用的计算机之间跑来跑去
我们这章讨论的 LAN 服务是一些程序，这些程序可以让你使用 LAN 在不同的计算机之间执行许多的操作。这些操作有：

- 从一个计算机拷贝文件到另一个。如果没有 LAN，你只能用磁带拷贝你的文件。然后走到另外一台计算机，然后重新读取磁带。
- 在本地计算机上通过一个终端登录另外的计算机。如果没有 LAN， 通常你只能走到哪个计算机的终端来进行登录。
- 在另外的计算机上执行命令，在本地观察结果。同样，如果你没有 LAN，你只能到另外的计算机去执行命令。
- 在远程计算机上存取文件，这意味着我们可以使用另外计算机上的磁盘，而不用将文件拷贝到本地磁盘上。

11.2 LAN 服务

在这一节,我们会了解有两组不同的服务可以让我们使用基本的 LAN 功能。这些服务是：

- ARPA 服务
- Berkeley 服务

ARPA 服务最早是 Defense Advaced Research（DARPA）在 60 年代初期定义的。这些服务成为通过单个 LAN 连接许多不同种类计算机的一个标准。我们要讨论的 ARPA 服务是：telnet，和 ftp。
DARPA 雇用 Berkeley 学院的 Bolt,Baranek 和 Newman 来开发这些服务。在 70 年代中期，Berkekey 大学开始使用新的 UNIX 操作系统上。他们最终开发出一套更健壮的服务，用在运行 UNIX 操作系统的计算机之间。这些服务现在被叫做 Berkeley 服务。我们在这章会介绍 rcp,rlogin 和 remsh 等

Berkeley 服务。

11.3 hostname 命令

语法：

hostname 报告你的计算机的网络名称

例子：

```
$ hostname

fred

$

$ more /etc/hosts

       192.1.2.1       fred

       192.1.2.2        barney
```

你的计算机有一个主机名。这个主机名被用于在 LAN 上鉴别你的系统。要想知道你的主机名，可以使用 hostname 命令。

```
$ hostname
fred
```

如果你想要同 LAN 上的其他计算机通信，你必须知道它的主机名。你可以向你的系统管理员询问其他的计算机的名字。如果你在你想要工作的主机上有一个用户帐号，你也可以自己检查主机名。

注意：想要使用任何 LAN 服务，你必须是远程主机的一个有效的用户。

你同样可以在文件/etc/hosts 中找到主机名。然而，如果你的 LAN 的规模很大，这个文件会包括几百个主机名。

11.4 telnet 命令

语法：

telnet hostname ARPA 服务，来远程登录另外的计算机。

语法：

telnet hostname ARPA 服务，来远程登录另外的计算机。

例子：

```
$ telnet fred

Trying.....

Conneted to fred

Escap character is ‘^]’.

HP-Ux fred 10.0 9000/715

Login:
```

telent 是 ARPA 服务中的远程登录的软件。
如果你键入命令：
$ telnet hostname
你会在你的屏幕上看到名称为 hostname 的计算机的登录提示符。然后你可以输入你在那台计算机上的用户名和口令，然后你就能登录那台计算机。
一旦你登录进了那台计算机，你的终端看上去就像是远程计算机上的终端一样。你能运行 shell 命令或者程序，甚至使用远程计算机上的打印机。你做的所有的工作都是在远程计算机完成的。你本地的计算机只起到一个信息传递的作用。
想要关闭 telnet 联接，可以使用 ctrl + d 或者 exit 从远程计算机退出即可。

11.5 ftp 命令

语法：

ftp hostname　　一种 ARPA 服务，用来从本地计算机拷贝文件到远程计算

？ 列出所有的 ftp 命令

quit 退出 ftp

想要使用 ARPA 服务，在本地和远程计算机之间拷贝文件。使用 ftp 命令。ftp 代表文件传送协议。像 telnet 一样，你必须指定远程计算机的主机名。

```
$ ftp hostname
```

ftp 会提示你输入你在远程计算机上的用户名和口令。这要求你在远程计算机上设置有一个口令。一旦你输入的登录信息正确，你就可以连接到 hostname 上。

在你看到 ftp>提示符的时候，你可以使用 ftp 命令来进行你的工作。这里是一些普通的 ftp 命令：

get rfile lfile	拷贝远程计算机上的 rfile 到本地计算机上的 lfile，你也可以使用全路径名
put lfile rfile	拷贝本地计算机上的 lfile 到远程计算机的 rfile。
Ls	列出远程计算机上的文件，这个 ls 同我们用过的 ls 命令相似。
Help command	显示一个关于命令的简短的帮助信息。
Quit	断开同远程计算机的联接，退出 ftp。

11.6 rlogin 命令

语法：

rlogin hostname　　一种 Berkekey 服务，用来远程登录计算机；rlogin

```
$ hostname

barney
```

rlogin 命令的功能与 telnet 命令类似。如果你键入：
$ rlogin hostname
你会自动登录到名字为 hostname 的系统上。rlogin 假定你登录远程主机的用户名同你登录本地主机的用户名一样。所以，它不会提示你输入你的用户名。
如果你的系统管理员已经配置好一个/etc/host.equiv 的文件，rlogin 不会提示你输入密码。这会使使用变得非常快捷和方便。同时，一个叫做.rhost 的文件会在你的 HOME 目录下被创建。这个文件会让你远程登录不需要输入密码。
同 telnet 相似，简单的退出指令就可以中断同远程主机的连接。

11.7 rcp 命令

```
语法：

rcp source_pathname target_pathname      一种 Berkeley 服务，让你可以从远程主机拷贝或者                                                拷贝文件到远程主机；使用起来同 cp 命令类似。

例子

$ rcp funfile fred: /tmp/funfile

$
```

rcp 代表远程拷贝。这是因为它的运行同 cp 命令类似。它可以运行在两个运行 Berkeley 服务的计算机之间。这个命令的一般的格式为：
$ rcp host1: souce host2:dest
这些命令的参数意味着拷贝 host1 上的 source 文件到 host2 上的 dest 文件。source 和 dest 同样能是全路径名。
如果你在本地主机拷贝文件到远程主机或者从远程主机拷贝文件到本地主机，你可以不使用本地主机名和冒号，
例如：
拷贝本地主机上的 funfile 文件到远程主机 fred 上的/tmp/funfile
$ rcp funfile fred: /tmp/funfile
拷贝主机 fred 上的/tmp/funfile 到主机 barney 上的/tmp 目录下：
$ rcp fred:/tmp/funfile barney: /tmp
所有的 cp 命令的规则同样适用于 rcp 命令。
注意： 在 rcp 命令工作的时候，、/etc/hosts.equiv 或者.rhost 文件必须被正确设置。否则 rcp 命令不会工作。

11.8 remsh 命令

```
语法：

remsh hostname command          一种 Berkeley 服务，可以在远程主机上运行命令
```

存储介质

在 UNIX 系统中，文件的存储借助于存储介质，如磁盘，软磁盘，光盘等而得以存储的。因此在 UNIX 系统中。掌握如何有效地对存储介质加以使用，是一项非常重要的技术。

学完这一章，你将能够做以下事情：

- 如何查看系统中存储空间的使用情况。
- 如何在软盘，磁带这类设备上生成文件系统。

存储空间的使用情况

在建立好一个文件系统并使之可用之后。随着用户在其上新建文件、修改文件或者删除文件，该文件系统的空闲 inode 数、空闲数据块等会相应地减少或者增多。如果文件系统的空闲空间减少到一定的程度，系统响应速度会大幅度下降甚至无法工作。这时，就需要对各文件系统的使用情况进行调查，然后采取相应的对策。

1.文件系统空闲情况的统计

在文件系统中，索引节点数和数据块数是两个至关重要的参数。当某个文件系统索引节点或数据块全部被占满是，将无法在在此文件系统上建立任何文件。这是一件比较可怕的事。为了使用户能方便地了解系统中每个文件系统中还剩下多少可用的索引节点和数据块，系统为我们提供了 df 命令。
df 命令为 disk free 的缩写。df 可以单独使用，也可以带参数。具体使用你可以参考联机帮助。

2.磁盘空间占用情况

在文件系统满了或者我们希望得知某个目录或文件对文件系统存储空间的使用情况时，可以使用 du 命令。du 命令为 disk usage 的缩写。在使用这个命令时可以指定一个或多个目录，如：

```
$ du /home/yxz
2       /home/yxz/Unix.dir
10      /home/yxz
$
```

使用上述命令，就能查出系统中某个用户占用了多少磁盘空间（假定用户只能在他的 HOME 目录下建立文件）。
如果没有指定文件或目录名的情况下，du 将显示当前目录占用的磁盘空间的情况。显示出的信息中包含有指定目录下所有的各个子孙目录，磁盘空间的使用情况。若要得知每个子目录及文件的信息，可以加上-a 选项。
若只想知道指定的目录占用磁盘空间的总数，可以使用-S 选项。
还有一些其它的选项可以参考帮助手册。

用户占用文件系统空间的统计

系统中的每个文件都有相应的拥有者。在某些情况下我们需要知道在某个文件系统上，每个用户各占用多少存储空间。为此可以使用 quot 命令。

quot 是 quotient 的缩写。它可接受一个文件系统名（块设备文件名）作为参数，然后统计出在该文件系统中建立有文件的各个用户所占用的磁盘块的数目。如：

```
# quot /dev/dsk/0s4
/dev/dsk/0s4:
82883   sybase
  340   vmsys
158     yxz
28      oasys
14      root
2       xyz
3       yxz
1       install
```

注意，此命令只能供超级用户使用。

存储介质上的文件系统

文件系统的物理基础就是系统所使用的各种存储介质，如磁盘、软盘，CD-ROM、磁带，等等。

UNIX 对其各种存储设备用一种统一的方法对待，那就是每个物理设备都是文件系统中的文件。当然这种文件比较特殊，被称为设备特殊文件（Device Special File），这些文件被组织在文件系统树的/dev 目录下。本节将介绍 UNIX 如何组织各种存储设备；如何对这些设备进行初始化（格式化）；以及如何将特定设备上的文件系统“ 挂接 ”到系统的文件系统树上，以及如何将其从文件系统树上“ 摘下来 ”。

存储介质的组织方式

每一种特性的物理设备在文件系统中都占据有一个相应的节点，即具有一个文件名称。这里介绍 UNIX 是如何组织硬盘、软盘、磁带、CD-ROM 这些常用的存储设备的。

1.硬盘的组织

我们知道，硬盘时一种块设备。也就是说，它每次同其它设备的数据交换均是以一个数据块（通常为 512 字节）为单位的。这样硬盘的存储空间也就是从 0 号块到最大块号。例如一个 1GB 的硬盘上，将会有 2M 个数据块。其编号将是从 0 至 2 的 21 次方－1。

对一个硬盘，可以对之进行“ 分区 ”操作。在硬盘的各个分区中，只有一个分区是“ 活动的 ”。也就是说，接通电源后机器将从活动分区中被引导从而进入相应的操作系统状态。

进行硬盘分区的实用程序是名为 fdisk。在那里还可以指定活动分区。由于此命令只能供系统管理员使用，所以在此不做介绍。

每个硬盘分区实际上就相当于一个独立的硬盘。所以在下面的叙述中我们就假定没有对硬盘进行分区。

硬盘在使用前要进行格式化操作。这个操作主要是在硬盘上写上有关存取数据和管理用的信息。不同种类的 UNIX，格式化操作的具体内部动作是不一样的。用户不用理它，格式化好之后，呈现在用户面前的硬盘才是若干编号连续的存储块。下面的问题是如何对这些存储块进行管理了。

我们可以把整个存储空间分成若干个段。每一段内存储块的编号都是连续的。各段首尾相邻，但任何段都不运允许有交叉。（这一操作类似于硬盘的分区）然后可以为每段存储块指定一个名称，系统将用指定的名称在/dev 目录下建立一个文件。此时这个文件所表示的就是该段连续的存储空间了。

在对每一个分段命名后，我们就得到相应的设备文件，此时就可以用此设备文件来在相应的磁盘段上建立文件系统了。

2．软盘、磁带、CD-ROM 的组织方式

上述三种存储设备因为其携带方便，易于保存，并且磁带和 CD-ROM 还具有存储量大（海量存储）的特点，因而在计算机系统中被广泛使用。在 UNIX 系统中这些设备也被按文件的组织方式加以组织。

但由于此类设备的多样性以及存取格式、存取方式的不同，使得对此类设备的使用也复杂起来。举例来说，对于软盘就有好多种不同的类型，如 3.5 英寸，5.25 英寸,有高密的，也有低密的，有的软盘上有引导块，有的没有，等等。并且在对它们进行访问的时候，有时不必格式化就可以访问，有些命令则只能访问格式化后的设备。

系统为上述特点进行区分，采取得方法是：对不同存取特征的组合设定不同的设备文件名。在需要按某种特征组合去访问相关介质时，指定相应的设备文件名即可。

对于磁带和 CD-ROM 等设备，系统对它们的组织方式与软盘类似，也是用不同的名称代表不同的格式的设备。例如在 Sun 0S 上，用/dev/rmt/?代表 1/2 英寸磁带设备，而用/dev/rst/?代表 scsi 磁带。对于 CD-ROM，其名称则是/dev/rdsk/c0t6d0s0。在使用磁带和 CD-ROM 时，如果发现用指定的名称不能正确访问数据，那么可换其它格式试试。如果所有的格式都不能完成任务，那么表明系统不支持所用的设备。

存储介质上文件系统的使用

对于软盘，磁带，和 CD-ROM 地的使用，一种方法就是先在其上生成某种类型的文件系统，然后将该文件系统安装到文件系统树的某个节点三。这样以后我们就能用常规的文件操作命令，如 mkdir,rmdir,cp,mv,rm 等在相应的存储介质上建立并且存储文件。

1.介质的格式化

软盘、磁带等存储设备在使用之前都要进行格式化操作（CD-ROM 由于其只读的特殊性，无需格式化，也无法对之格式化）。格式化的操作的具体内容随 UNIX 操作系统的不同而有较大的差异。但总的来说，它们都要在存储介质上的某些存储区域中写入一些操作系统访问该介质所必不可少的管理信息，这之外的那些存储区域才被用来存放用户数据。并且在格式化的过程中，一般还要对存储介质上的存储区域进行校验，看看其中有没有被损坏的地方。

不同系统提供的格式化工具也不尽相同。例如在 AT&T UNIX System VR 4.0 上的格式化命令为 format。

2.介质上文件系统的生成

这里我们所讲述的内容主要也是针对软盘和磁带这些存储介质的。CD-ROM 因为在作好之后，它上面的文件系统实际上也就算是生成了，因此也就不用我们再生成。另外并不是所有的磁带上都可以生成文件系统。大多数 UNIX 系统中提供的生成文件系统的命令均为 mkfs.mkfs 命令要求提供一个裸设备文件名作为参数。并且在命令行中我们还可以指定文件系统的索引节点数和数据块数目。另外债命令行中我们还可以指定所建文件系统的类型。每种 UNIX 系统支持的文件系统类型的数量也不同，但 S5,UFS 等这类比较通用的文件系统都还是支持的，读者可以自己看看所用的机器上都支持哪些种类的文件系统。在我们不指定文件系统类型的时候，系统会在指定设备上建立一种缺省的文件系统。

3.文件系统的安装

在一个文件系统建立好之后，还并不能马上就使用它。在使用之前必须先将它“ 挂接 ”到系统的文件系统树上某个节点处。这种操作被称为文件系统的安装。具体的细节请看 UNIX 系统管理的文件系统部分。

4.文件系统的拆卸

具体的细节请看 UNIX 系统管理的文件系统部分

离线文件存储

目标

完成这一章，你能够做以下事情：

- 使用 tar 命令存储文件到磁带上
- 使用 find 和 cpio 存储文件到磁带上。
- 使用 tar,和 cpio 检索存储过的文件

1 存储文件到磁带

- 要想存储文件到磁带，首先要知道你的磁带机对应的*设备文件*
- 典型的设备文件名是：
  /dev/rmt/0m　　9 磁道磁带或者 DDS 磁带（旧名）
  /dev/rmt/c0t3d0BEST　9 磁道磁带或者 DDS 磁带

- 向你的系统管理员询问磁带的设备文件名。
- 执行备份的命令有：
  tar
  cpio

许多时候，UNIX 系统的用户都需要保存文件的备份到一些可移动的介质上。通常备份使用的介质有 9 磁道的磁带或者 DDS 格式的数字磁带。这一章会介绍基本的备份数据到磁盘上或者从磁盘检索数据的方法。记住你的系统管理员通常只负责备份整个系统;你应该同你的系统管理员协同进行磁带备份的工作。

注释：要恢复一个被删除的文件的唯一的方法就是从一个备份磁带上恢复数据。

2 tar 命令

语法：tar – key [f device_file] [file...]

例子：

创建一个备份档案：

```
$ tar – cvf /dev/rmt/0m   myfile
```

语法：tar –key [f device_file] [file...]

例子：

创建一个备份档案：

```
$ tar –cvf /dev/rmt/0m   myfile
```

从一个备份档案得到一个档案内容列表：

```
$ tar –tvf /dev/rmt/0m
```

从备份档案中提取一个文件

```
$ tar –xvf /dev/rmt/0m myfile
```

tar 命令将文件存档到磁带。这个命令能存储和恢复磁带上的文件。这些功能是由命令的第一个参数称为 key argument 来控制的。
有效的 key 参数有：
c 创建（create）一个新的档案。
x 从档案中提取一个文件
t 打印档案中的内容的列表。
r 在档案的末尾增加文件。
u 如果文件是新的或者修改过的，就添加到档案的末尾。
v 在存档和恢复的时候在屏幕上显示文件名（冗长模式 verbose）
f file 指明要创建档案的文件名。注意，这个文件不一定必须是一个磁带的设备文件，你可以在你的磁盘上的目录下创建一个档案文件。如果不指定，其默认值为/dev/rmt/0m

3 cpio 命令

两种模式：

cpio –o [cvx]　　创建一个档案，从标准输入读取文件列表。档案被写到标准输出。

cpio –i [cdmtuvx]　从一个档案中恢复数据。档案从标准输入读取。

创建一个你当前目录下的所有文件的存档文件。

```
$ find . | cpio –ocv > /dev/rmt/0m
```

从一个档案文件中恢复所有的文件。

```
$ cpio –icdmv < /dev/rmt/0m
```

这个命令制造文件或是目录的拷贝档案。cpio 代表 copy input and output。Cpio 有两种模式：

-o　　制造一个备份。读取标准输入并且拷贝每一个文件到标准输出。

-i　　恢复一个备份。读取备份文件并且在磁盘上重新创建文件。

当创建备份的时候，cpio –o 命令使用标准输入作为它的文件名的来源。标准输出作为档案的输出。默认得情况是一个文件列表作为标准输入，一个档案文件作为标准输出，你不得不指定一个磁带作为一个设备，同时必须提供一个要存储的文件列表。通常是通过管道输送 find 的输出到 cpio 执行。

要恢复一个档案，使用 cpio –i 从标准输入读取这个档案，同时恢复文件内容到你的磁盘上。被创建的文件名依靠这个档案文件被创建时使用的是相对还是绝对的路径名。

同主选项-o,-i 一起，我们还可以使用几个其它的选项：

-o	-i	选项功能
-c	-c	写文件头为 ASCII 码格式（如果同-o 一起使用，在-i 同样要使用这个选项）
—	-d	在需要时，重新建立目录结构
—	-m	保留当前修改的数据。（在版本控制时，十分有用）
—	-t	显示备份档案的内容列表
—	-u	无条件恢复（如果文件已经存在，这个选项会覆盖这个文件
-v	-v	显示被拷贝的文件的一个列表
-x	-x	处理特殊的设备文件

其他例子：

获得档案内容的列表：

```
$ cpio –ict < /dev/rmt/0m
```
恢复单个文件：
```
$ cpio –icudm “filename” < /dev/rmt/0m
```
恢复所有同模型匹配的文件：
```
$ cpio –icudm ‘ filename* ’ < /dev/rmt/0m
```

关于 find 命令

find 命令可以和备份命令一起使用，其作用是产生需要备份的文件的文件名列表。注意，find 可以产生一个相对路径的列表(find .)和一个绝对路径的列表（find /home/user3）。产生文件名列表的方式会决定文件名存储在磁带上的方式。

语法：

```
find path-list [expression]
```

expression 支持许多关于搜索条件的关键字。想要知道细节，请使用 man 来查找联机帮助。

第十九章：进程控制

目标

完成这一章，你能够做以下事情：

- 使用 ps 命令
- 在后台运行进程 ，并且使用 ps 命令监视正在运行的进程的状态
- 运行一个后台进程并使其在你退出系统后不被挂起。
- 切换后台进程到前台运行。
- 挂起一个进程。
- 停止进程的运行。

19.1 ps 命令

```
语法：

ps [-efl] 报告进程的状态

例子：

$ ps

PID TTY TIME COMMAND

1324 ttyp2 0:00 sh

1387 ttyp2 0:00 ps



$ ps –ef

UID PID PPID C STIME TTY TIME COMMAND

Root 0 0 0 Jan 1 ? 0:20 swapper

Root 1 0 0 Jan 23 ? 0:00 init

Root 2 0 0 Jan 23 ? 0:16 vhand

User3 1324 1 3 18:03:21 ttyp2 0:00 -sh

User3 1390 1324 22 18:30:23 ttyp2 0:00 ps –ef
```

系统中每个进程在启动的时候都会被分配一个唯一的确认号码，这个号码就叫进程 ID（PID）。ps 命令会显示当前运行中（或睡眠中）的进程的信息，包括每

个进程的 PID 和每个进程父进程的 PID（PPID）。通过 PID 和 PPID 号，你可以追踪你系统中任何进程的起源。ps 命令也会报告每个进程的属主和是进程在哪个终端上执行的。
ps 命令可以不带参数执行，结果会报告你当前的终端线索的进程信息。 -e 选项会报告系统中每一个进程的信息，而不仅仅是你自己运行的进程信息。 -f 和 -l 选项会报告长的（long），或完全的（full）包含其他的细节的列表。
注意：ps 命令执行时会集中使用 CPU，你也许会注意到当这个命令执行的时候要过一会才有响应

19.2 后台进程

语法：

命令>cmd.out &

例子：

$ grep user * > grep.out &

1. **194**

   $ ps
   PID TTY TIME COMMAND
   164 ttyp2 0:00 sh
   194 ttyp2 0:00 grep
   195 ttyp2 0:00 ps

命令> cmd.out &

- 使命令在后台执行。
- 当后台任务开始后立刻返回提示符。
- 重定向命令的输出，这样命令的输出就不会影响当前屏幕的显示。
- 退出系统会中止后台运行的进程。用户会在第一次试图退出时收到一个警告“There are running jobs”。用户必须再次键入 Exit 或 ctrl+d 来中止这个线索。

一些命令会执行很长的时间，例如在全部的磁盘上查找单个文件，或使用一个文本处理工具来格式化和打印一个手册。UNIX 操作系统允许你启动一个很消耗时间的程序，并且在让它在后台运行，在后台，UNIX 会让这个程序继续执行。和其他的立即执行的命令不同的是，shell 不会等待后台程序执行完毕，而会立即返回提示符，这样你就可以继续其他的工作。在一个命令的末尾使用一个“ & ”符号可以使一个命令在后台运行。通常需要重定向后台运行的程序的输出，以免后台命令的输出影响你当前的终端线程。如果没有重定向输出，后台命令会使用标准输出，也就是会输出到你的终端上。

由于 shell 控制标准输入，而后台运行的命令不能够从标准输入接收数据，因此，任何后台运行的命令要求输入都必须使用输入重定向从一个文件得到输入数据。
当一个命令被放在后台执行,如果设置了 monitor 选项(set -o moniotr)，shell 会报告命令的任务号和进程 ID,任务号定义了与你终端线索相联系的任务。进程 ID 为 UNIX 分配给每一个被执行的进程的唯一的号码。
Monitor 选项会导致在后台进程完成之后会在终端上显示一条信息：
[1] + Done grep user * > grep.out & 提示进程执行完毕
由于在后台运行的命令不和键盘通信，所以你不能够通过中止键，ctrl + c 来中断一个后台进程，后台进程可以通过 kill 命令来中断或通过退出系统来中断。
注意： 一个后台进程都要明确地重定向输入和输出。
注意： 一个后台任务可能包含多个命令，只要将这些命令用括号括起来（命令 1，命令 2，命令 3），然后操作系统就会作为一个任务执行这些命令。

19.3 将任务放在后台/前台运行

jobs	显示当前运行的任务
ctrl + z	将当前前台运行的程序挂起。
fg [%number]	将指定任务号的任务放在前台运行
fg [%string]	将以指定字符串开头的命令放在前台运行
bg [%number]	将指定任务号的任务放在后台运行
bg [%number]	将指定任务号的任务放在后台运行

在 POSIX shell 中，进程可以被放在前台或后台运行。如果你在前台正在运行一个冗长的进程，你可以向这个进程发送一个 susp（挂起）信号，这个信号通常被设置为 ctrl + z.。这个挂起字符通常是在登录时通过.prifile 指定的，语法为：stty susp ^Z 。这个动作会暂时停止你的前台进程，同时出现一个 shell 提示符。你然后可以使用 bg 命令来将这个命令放到后台运行。
同样，如果你有一个在后台运行的进程，你想要把它放在前台运行，你可以使用 fg 命令，然后前台进程会控制你的终端，直到运行完毕或者被挂起。

15.4 nohup 命令

语法：

nohup 命令 （使命令不被挂起）。

例子：

```
$ nohup cat * > bigfile &

$ ctril + d

login: user3

passwd :



$ ps -ef |grep cat

UID PID PPID COMMAND

User3 972 1 cat * >bigfile &
```

UNIX 系统提供一个 nohup 命令来使命令不被挂起。nohup 命令是一组特殊 unix 命令的一个，这组命令被称为 prefix command(前缀命令)，这个命令在其他命令之前使用。记住，退出系统通常都会中止后台的进程。但是如果一个后台命令以 nohup 开头，你就可以放心地退出系统，系统会继续完成你的进程。即使这个程序的父 shell 不再运行。注意，当 nohup 命令的父进程中止后，这个命令会被一号进程（init）所收养。你可以再次登录观察 nohup 命令执行后的状态和结果。当使用 nohup 的时候，用户通常都要重定向输出到文件中。如果用户没有指定一个输出文件，nohup 会自动地重定向输出到 nohup.out 文件中。注意，nohup.out 文件中会同时有标准输出和标准错误信息。

19.5 nice 命令

```
语法：

nice [-N] 命令 使一个命令以更低的优先级运行

N 为一个 1 到 19 的号码

例子：

$ nice -10 cc myprog.c -o myprog

$ nice -5 sort * > sort.out &
```

UNIX 操作系统是一个分时的系统，进程的优先级决定程序使用系统资源的频率。优先级低的任务比优先级高的用户存储系统的频率更少。例如：你的终端线索有相对更高的优先级得到提示和系统响应。
nice 命令是另一个前缀命令，能让你降低一个程序的优先级。这个命令对运行不要求立即完成的命令的时候有用处，例如格式化整个帮助手册。
语法是：
nice [-increment] 命令

increment 是一个 1 到 19 的整数。默认值为 10。一个进程的 nice 值越高，它的优先权就越低。nice 值不是一个绝对的优先级修改量。
你可以通过 ps -l 来浏览进程的优先级。优先级在字段 PRI 那一栏显示。更高的优先级任务的优先值更低。nice 值在字段 NI 下显示。
在大多数的 UNIX 系统中，前台进程默认 nice 值为 20，后台进程的值为 24。最大值为 39，所以最大的增量为 19 和 15。更多的增量都不会增加超过 39。而负的增长只能 root 使用。

19.6 kill 命令

语法：

kill [-s signal_name] PID [PID....] 发送一个信号给指定的进程。

例子：

```
$ cat /usr/share/man/cat1/ * > bigfile1 &
```

1. 995

```
$ cat /usr/share/man/cat2/ * > bigfile2 &
```

2. 996

```
$ kill 995
[1] -Terminated cat /usr/share/man/cat1/* > bigfile1 &
$ kill -s INT %2
[2] + interrupt cat /usr/share/man/cat2/ * >bigfile2 &
$ kill -s KILL 0
```

kill 命令能被用来中断任何命令，其中包括 nohup，和后台命令。kill 发送一个信号给一个进程。更具体的说，kill 发送一个信号给一个进程。进程收到其中的大多数的信号后，默认的行动都是死亡。发送信号者必须是进程的属主；kill 不能够被用来杀掉其他用户的进程除非使用 kill 的是超级用户。
在 UNIX 系统中，实际上是不可能真正地杀死一个进程。大多数的 UNIX 系统所做的只是要求进程自己结束。默认的情况是，kill 发送 TERM 信号（软中断信号）给指定的进程。但进程通常不会捕捉或者会忽略这个信号。而有其他的信号，如上表列出的信号，能够通过-s 选项指定。UNIX 系统提供的，确保 kill 执行成功的，最方便的办法就是 KILL 信号（kill signal）
要想杀死一个进程，你可以指明进程 ID 或任务号。当指明任务号的时候，必须使用%作为前缀。如果指明的进程号为 0，会终止所有的与当前 shell 相关的所有的进程，包括当前的 shell。
注意： 命令 kill －l 会列出所有支持的 signal-name（信号名）的值。当 －l 选项被使用的时候，每个信号的符号名称都会输出到标准输出上：

```
$ kill -l
HUP INT QUIT ILL TRAP ARRT EMT FPE KILL BUS SEGV SYS PIPE ALRM TERM
USI USR2 PWR VTALRM PROP IO WINCH STOP TSTP CONT TTIN TTOU URG LOST
```

vi 全屏幕编辑器

vi(Visual)是以视觉为导向的全屏幕编辑器、共分为三种方式（mode):

- **command 方式：**

任何输入都会作为编辑命令，而不会出现在屏幕上，若输入错误则有“ 岬 ”的声音；任何输入都引起立即反映

- **insert 方式：**

任何输入的数据都置于编辑寄存器。在 command 方式下输入（I,a,A 等）,可进入 insert 方式， insert 方式下按 ESC，可跳回 command 方式。

- **escape 方式：**

以“ :”或者“ / ”为前导的指令，出现在屏幕的最下一行，任何输入都被当成特别指令。

进入 vi（在系统提示符下面输入以下指令）：

vi	进入 vi 而不读入任何文件
vi filename	进入 vi 并读入指定名称的文件（新、旧文件均可）。
vi +n filename	进入 vi 并且由文件的第几行开始。
vi +filename	进入 vi 并且由文件的最后一行开始。
vi + /word filename	进入 vi 并且由文件的 word 这个字开始。
vi filename(s)	进入 vi 并且将各指定文件列入名单内，第一个文件先读入。
vedit	进入 vi 并且在输入方式时会在状态行显示“ INSERT MODE"。

编辑数个文件（利用 vi filename(s))进入 vi 后）

:args	显示编辑名单中的各个文件名
:n	读入编辑名单中的下一个文件
:rew	读入编辑名单中的第一个文件
:e#	读入编辑名单内的前一个文件
:e file	读入另一个文件进 vi(此文件可不在编辑名单内）,若原文件经修改还没有存档，则 应先以: w 存档。
:e! file	强迫读入另一个文件进入 vi，原文件不作存档动作。

存储及退出 vi

:w filename	存入指定文件，但未退出 vi（若未指定文件名则为当前工作的文件名）。
:wq 或者 :x 或者 zz	存文件，并且退出 vi.
:q	不作任何修改并退出 vi。

:q!	放弃任何修改并退出 vi。
:!command	暂时退出 vi 并执行 shell 指令，执行完毕后再回到 vi。
:sh	暂时退出 vi 到系统下，结束时按 Ctrl + d 则回到 vi。

加数据指令

i	在关标位置开始插入字符，结束时候按 ESC 键。
I	在光标所在行的最前面开始加字，结束时按 ESC 键。
a	在光标位置后开始加字，结束时按 ESC 键。
A	在光标所在行的最后面开始加字，结束时按 ESC 键。
o	在光标下加一空白行并开始加字，结束时按 ESC 键。
O	在光标上加一空白行并开始加字，结束时按 ESC 键。
!command	执行 shell 指令，并把结果加在光标所在行的下一行。

删除指令

nx	删除由光标位置起始的 n 个字符（含光标位置，按一个 x 表示删除光标所在的字符）
nX	删除由光标位置起始的 n 个字符（不含光标位置）。
ndw	删除光标位置其实的 n 个字符组（word)。
d0	将行的开始到光标位置的字符全部删除。
d$ 或 D	将光标位置起始到行尾的字符全部删除。
ndd	将光标位置起始的 n 行（整行）删除（dd 表示删除光标所在行）。
:start,endd	删除文件的第 start 到 end 行。

关标移动

0	移到一行的开始
$	移到一行的最后
[	移到文件开始位置
]	移到文件结束位置
nh	往左移 n 位
nl 或者 spacebar	往右移 n 位
nk	向上移 n 行
n+	向上移 n 行，光标在该行的起始
ni	向下移 n 行
n-	向下移 n 行，光标在该行的起始
H	移到屏幕的左上角
M	移到屏幕的中间行开头
L	移到屏幕的最后一行
G	移到文件的最后一行
nG 或者:n	移到文件的第 n 行
nw	右移 n 个字组，标点符号属于字组
nW	右移 n 个字组，标点符号不属于字组
nb	左移 n 个字组，标点符号属于字组
nB	左移 n 个字组，标点符号不属于字组

Ctrl + u	屏幕上卷半个菜单
Ctrl + d	屏幕下卷半个菜单
Ctrl + b	屏幕上卷一个菜单
Ctrl + F	屏幕下卷一个菜单

修改指令

r	修改光标文件的字符
R	从光标位置开始修改，结束时按 ESC 键
new	更改 n 组字符，结束时按 ESC 键
ncc	从光标所在位置开始更改 n 行，结束时按 ESC 键

重排各行长度

i	并按 Enter 将该行由光标所在处断开，并进入 insert 方式
J	把下一行的数据连接到本行之后

寻找指令

/text	从光标位置往下找字串 text
?text	从光标位置往上找字串 text
n	继续找下一个字串（在输入上面的寻找指令之后使用）

寻找并且取代指令

:getxt1/s/ /text2/options	将各行的 text1 替换为 text2
	option=g 表示文件中所有的 text1 均被取代，若未输入任何 option,则只有各行中的第一个出现的 text1 被取代
	option=go 在屏幕显示各取代的行
	option=gc 在每个字串取代之前要求确认
Start,endgtext1/s/ / text2/options	同上，只寻找并取代第 start~end 行。

或:Start,ends/text1/text2/options

复制及移动文件

:first,last co dest	将 first 到 last 行的数据复制到目标行(dest）下面
:Start,end m dest	将 start 到 end 行的数据移动到目标行（dest)下。
:r filename	将指定文件的内容读入光标所在行下。
nY	将光标所在位置开始的 n 行数据暂存
p	复制暂存数据在光标的下一行
P	复制暂存数据在光标的上一行

其他命令

.	重复前一指令
u	取消前一指令
Ctrl + l	刷新屏幕显示
:set number	显示文件的行号，但不会存文件
:set nonumber	解除行号显示
:set ai	设置每行起始位置（以光标当前位置为起始）

:set noai	取消行起始位置设定
:f 或<Ctrl> + g	告诉用户有关现行编辑文件的数据。