

目 录

第 1 章 绪论	1
1.1 引言	1
1.2 计算机简介	1
1.3 计算机硬件	2
1.3.1 输入设备	3
1.3.2 处理器单元	3
1.3.3 内存	4
1.3.4 外部存储器	6
1.3.5 输出设备	6
1.4 指令执行过程	6
1.4.1 性能指标	7
1.5 什么是软件	8
1.5.1 系统软件	8
1.5.2 应用软件	13
习题	13
第 2 章 UNIX 操作系统	14
2.1 UNIX 操作系统:历史简介	14
2.1.1 UNIX 系统 V	15
2.1.2 Berkeley UNIX	15
2.1.3 UNIX 标准	15
2.2 其他 UNIX 系统	15
2.3 UNIX 操作系统概要	16
2.4 UNIX 系统特征	17
习题	19
第 3 章 UNIX 入门	20
3.1 UNIX 系统的登录和退出	20
3.1.1 登录	20
3.1.2 修改口令:passwd 命令	21
3.1.3 退出系统	22
3.2 一些简单的 UNIX 命令	23
3.2.1 命令行	23
3.2.2 基本的命令行格式	23
3.2.3 显示日期和时间:date 命令	24
3.2.4 用户信息:who 命令	25

3.2.5 显示日历:cal 命令	26
3.3 UNIX 帮助信息	27
3.3.1 使用 learn 命令	27
3.3.2 使用 help 命令	27
3.3.3 获取更多的帮助信息:UNIX 用户手册	28
3.3.4 使用电子手册:man 命令	28
3.4 更正键盘输入错误	29
3.5 使用 shell 和系统工具	30
3.5.1 shell 的种类	30
3.6 登录过程	31
命令小结	33
习题	34
上机练习	34
第 4 章 vi 编辑器入门	36
4.1 什么是编辑器	36
4.1.1 UNIX 支持的编辑器	36
4.2 vi 编辑器	37
4.2.1 vi 的工作模式	37
4.3 基本 vi 编辑器命令	38
4.3.1 进入 vi 编辑器	38
4.3.2 文本输入模式	40
4.3.3 命令模式	44
4.4 存储缓冲区	50
命令小结	50
习题	52
上机练习	53
第 5 章 UNIX 文件系统介绍	55
5.1 磁盘组织	55
5.2 UNIX 里的文件类型	55
5.3 目录详述	55
5.3.1 用户主目录	56
5.3.2 工作目录	57
5.3.3 理解路径和路径名	57
5.3.4 使用文件和目录名	58
5.4 目录命令	60
5.4.1 显示目录路径名:pwd 命令	60
5.4.2 改变工作目录:cd 命令	61
5.4.3 创建目录	62

5.4.4 删除目录:mdir 命令	64
5.4.5 目录列表:ls 命令	65
5.5 显示文件内容	73
5.6 打印文件内容	73
5.6.1 打印:lp 命令	73
5.6.2 取消打印请求:cancel 命令	75
5.6.3 获取打印机状态:lpstat 命令	76
5.7 删除文件	76
5.7.1 删除文件之前	78
命令小结	78
习题	80
上机练习	81
第 6 章 vi 编辑器:高级使用	82
6.1 更多有关 vi 编辑器的知识	82
6.1.1 调用 vi 编辑器	82
6.1.2 使用 vi 调用选项	83
6.1.3 编辑多文档	83
6.2 重排文本	85
6.2.1 移动行:dd,p 或 P	85
6.2.2 复制行:yy,p 或 P	86
6.3 vi 操作符的域	86
6.3.1 使用带域控制键的删除操作符	87
6.3.2 使用带域控制键的拾取操作符	88
6.3.3 使用带域控制键的修改操作符	89
6.4 在 vi 中使用缓冲区	90
6.4.1 数字编号缓冲区	90
6.4.2 字母序缓冲区	92
6.5 光标定位键	93
6.6 定制 vi 编辑器	94
6.6.1 选项格式	94
6.6.2 设置 vi 环境	95
6.6.3 行长度和行回绕	97
6.6.4 缩写与宏	97
6.6.5 .exrc 文件	99
6.7 最后的 vi 命令	100
6.7.1 运行 shell 命令	100
6.7.2 行连接	101
6.7.3 搜索和替换	101
命令小结	101

习题	103
上机练习	103
第7章 UNIX 文件系统(续)	105
7.1 读取文件	105
7.1.1 vi 编辑器的只读版:view 命令	105
7.1.2 读取文件:pg 命令	105
7.1.3 指定页和行数	106
7.2 shell 重定向	107
7.2.1 输出重定向	107
7.2.2 输入重定向	108
7.2.3 回顾 cat 命令	109
7.3 增强的文件打印功能	111
7.4 文件处理命令	114
7.4.1 复制文件:cp 命令	114
7.4.2 移动文件:mv 命令	116
7.4.3 链接文件:ln 命令	117
7.4.4 计算字数:wc 命令	119
7.5 文件名置换	120
7.5.1 “?”元字符	121
7.5.2 “*”元字符	121
7.5.3 “[]”元字符	122
7.5.4 元字符和隐藏文件	123
7.6 其他文件操作命令	124
7.6.1 寻找文件:find 命令	124
7.6.2 显示文件头部:head 命令	127
7.6.3 显示文件尾部:tail 命令	128
7.6.4 选择文件的一部分:cut 命令	129
7.6.5 连接文件:paste 命令	130
7.6.6 另一个页查看工具:more 命令	131
7.7 UNIX 内部:文件系统	131
7.7.1 UNIX 磁盘结构	132
7.7.2 整体过程	133
命令小结	134
习题	137
上机练习	138
第8章 探索 shell	140
8.1 UNIX shell	140
8.1.1 理解 shell 的主要功能	141

8.1.2	显示信息:echo 命令	142
8.1.3	消除元字符的特殊含义	143
8.2	shell 变量	145
8.2.1	显示和清除变量:set 和 unset 命令	145
8.2.2	给变量赋值	146
8.2.3	显示 shell 变量的值	146
8.2.4	理解标准 shell 变量	147
8.3	更多的元字符	150
8.3.1	执行命令:使用后单引号	150
8.3.2	命令排序:使用分号	151
8.3.3	命令编组:使用括号	151
8.3.4	后台计算:使用 & 符号	152
8.3.5	链接命令:使用管道操作符	153
8.4	更多 UNIX 系统工具	153
8.4.1	延时计时:sleep 命令	153
8.4.2	显示 PID:ps 命令	154
8.4.3	继续执行:nohup 命令	155
8.4.4	终止一个进程:kill 命令	155
8.4.5	分离输出:tee 命令	157
8.4.6	文件搜索:grep 命令	157
8.4.7	文本文件排序:sort 命令	159
8.4.8	按指定字段排序	161
8.5	启动文件	164
8.5.1	系统策略	164
8.5.2	用户策略	164
8.6	KORN shell(ksh)	165
8.6.1	Korn shell(ksh)变量	165
8.6.2	Korn shell(ksh)选项	166
8.6.3	命令历史(ksh):history 命令	167
8.6.4	重复执行命令(ksh):r(重复执行)命令	168
8.6.5	别名(ksh):alias 命令	168
8.6.6	命令行编辑(ksh)	169
8.6.7	登录和启动:Korn shell 风格	171
8.6.8	将事件号码添加到提示符中(ksh)	171
8.7	UNIX 进程管理	172
	命令小结	175
	习题	177
	上机练习	177

第 9 章 UNIX 通信	179
9.1 通信方式	179
9.1.1 全双工通信:write 命令	179
9.1.2 禁止消息:mesg 命令	180
9.1.3 显示新闻:news 命令	181
9.1.4 广播消息:wall 命令	182
9.1.5 全双工通信:talk 命令	183
9.2 电子邮件	184
9.2.1 使用邮箱	184
9.2.2 发送邮件	185
9.2.3 读邮件	186
9.2.4 退出 mailx:q 和 x 命令	187
9.3 mailx 输入模式	188
9.3.1 发送已有的文件	192
9.3.2 给一群用户发送邮件	192
9.4 mailx 命令模式	193
9.4.1 阅读/显示邮件	193
9.4.2 删除邮件	195
9.4.3 保存邮件	196
9.4.4 回复邮件	197
9.5 定制 mailx 环境	198
9.5.1 mailx 使用的 shell 变量	198
9.5.2 设置 .mailrc 文件	200
9.6 与系统外的用户通信	201
命令小结	201
习题	203
上机练习	203
第 10 章 程序开发	205
10.1 程序开发	205
10.2 编程语言	205
10.2.1 低级语言	205
10.2.2 高级语言	206
10.3 编程机制	207
10.3.1 建立可执行程序步骤	207
10.3.2 编译器/解释器	208
10.4 简单的 C 程序	209
10.4.1 改正错误	210
10.4.2 重定向标准错误	211
10.5 UNIX 编程工具	212

10.5.1	make 工具	212
10.5.2	SCCS 工具	212
习题		213
上机练习		213
第 11 章	shell 编程	214
11.1	理解 UNIX shell 编程语言:介绍	214
11.1.1	写一个简单脚本	214
11.1.2	执行脚本	215
11.2	写更多的 shell 脚本	217
11.2.1	使用特殊字符	219
11.2.2	退出系统的风格	220
11.2.3	执行命令:dot 命令	221
11.2.4	读取输入:read 命令	222
11.3	探索 shell 编程基础	224
11.3.1	注释	224
11.3.2	变量	224
11.3.3	命令行参数	225
11.3.4	条件和试验	229
11.3.5	不同类别的判断	234
11.3.6	参数替换	238
11.4	算术运算符	241
11.4.1	算术运算(sh):expr 命令	241
11.4.2	算术操作(ksh):let 命令	243
11.5	循环结构	244
11.5.1	for 循环:for-in-done 结构	244
11.5.2	While 循环:while-do-done 结构	245
11.5.3	Until 循环:until-do-done 结构	248
11.6	调试 shell 程序	249
11.6.1	sh 命令	249
命令小结		252
习题		253
上机练习		253
第 12 章	shell 脚本:编写应用程序	255
12.1	编写应用程序	255
12.1.1	程序 lock1	255
12.2	UNIX 内部:信号(SIGNAL)	256
12.2.1	信号数俘获信号:trap 命令	257
12.2.2	陷阱复位	258

12.2.3	设置终端参数:stty 命令	258
12.3	终端的进一步讨论	260
12.3.1	终端数据库:terminfo 文件	260
12.3.2	设置终端功能:tput 命令	260
12.3.3	解决 lock1 程序的问题	262
12.4	更多的命令	264
12.4.1	多路分支:case 结构	264
12.4.2	回顾 greetings 程序	266
12.5	菜单驱动的应用程序	267
12.5.1	层次图	267
12.5.2	ERROR 程序	271
12.5.3	EDIT 程序	271
12.5.4	ADD 程序	273
12.5.5	获取记录	276
12.5.6	DISPLAY 程序	276
12.5.7	UPDATE 程序	279
12.5.8	DELETE 程序	283
12.5.9	REPORTS 程序	285
12.5.10	REPORT_NO 程序	286
	命令小结	289
	习题	290
	上机练习	290
第 13 章	告别 UNIX	291
13.1	磁盘空间	291
13.1.1	df 命令:显示未用磁盘空间	291
13.1.2	统计磁盘空间使用情况:du 命令	292
13.2	高级 UNIX 命令	292
13.2.1	banner 命令:显示标题	292
13.2.2	at 命令:在指定时间执行程序	293
13.2.3	type 命令:显示命令类型	295
13.2.4	time 命令:显示命令执行时间	295
13.2.5	calendar 命令:提醒服务	296
13.2.6	显示详细用户信息:finger 命令	296
13.2.7	tar 命令:存档和分发文件	298
13.3	拼写错误更正	301
13.3.1	创建用户词汇文件	302
13.4	UNIX 系统安全	303
13.4.1	口令保护	304
13.4.2	文件保护	304

13.4.3	目录访问权限	305
13.4.4	超级用户	306
13.4.5	文件加密: <i>crypt</i> 命令	306
13.5	使用 FTP	307
13.5.1	FTP 的工作原理	307
13.5.2	匿名 FTP	313
13.6	使用压缩文件	315
13.6.1	<i>compress</i> 和 <i>uncompressed</i> 命令	315
	命令小结	315
	习题	318
	上机练习	318
附录 A	命令索引	320
附录 B	分类命令索引	321
附录 C	命令小结	323
附录 D	<i>vi</i> 命令小结	334
附录 E	ASCII 表	338
附录 F	参考文献	342

第 1 章 绪 论

本章简要介绍计算机硬件和软件的基本知识,解释基本的计算机词汇和概念。本章将讨论软件分类,解释操作系统的重要性,并探讨操作系统的主要功能。

1.1 引言

大多数人是通过参加介绍性的计算机课程,或在工作或家庭环境中使用计算机来获得计算机基本知识的。如果没有学过任何计算机课程或已忘记所学的课程,本章首先简要介绍计算机,解释一些常用的硬件和软件词汇,再讨论操作系统软件。

1.2 计算机简介

什么是计算机?韦伯斯特学生词典(Merriam Webster's Collegiate Dictionary)把计算机定义为可存储、检索和处理数据的可编程电子设备。本章将详述计算机的定义,探究计算机系统的各个组成部分。

依据大小、能力和速度的不同,计算机可分为以下四类:

- 巨型计算机
- 大型计算机
- 小型计算机
- 微型计算机

【说明】

这是一个很不明确的分类,一类的低端系统可能会与另一类的高端系统重合。

巨型计算机:巨型计算机是最快和最贵的计算机,比大型计算机要快 1000 倍。巨型计算机是为天气预报、三维模拟、计算机动画等复杂的计算型应用而设计的。所有这些任务都需要异常巨大的复杂计算和巨型计算机的性能。巨型计算机通常包括几百个处理器,使用最新和最昂贵的硬件设备。

【说明】

巨型计算机也用于其他数值应用。甚至好莱坞也利用巨型计算机的强大图像处理能力来制作电影特技。

大型计算机:大型计算机是为大型机构的信息处理而设计的大型、快速的系统。大型计算机可支持几百个用户,同时运行数百个程序。它拥有强大的输入/输出(I/O, Input/Output)能力,包括大容量的内存和外存。大型计算机主要用于银行、医院等大型商务环境和大学等其他大型机构。大型机是昂贵的,通常要求由受过专业培训的人员进行操作和维护。

【说明】

I/O 设备是计算机与外界(人或其他计算机)进行通信的工具。各种 I/O 设备在通信介质和速度上差异很大。

小型计算机:直到 20 世纪 60 年代后期,所有的计算机都是大型计算机,只有大型机构能使用计算机。随后出现了小型计算机,其最初的用途是执行专门任务,最初是在大学和科研环境中使用的。小型计算机很快就在中小型机构的数据处理业务中广泛使用了。今天一些“小型计算机”的性能甚至胜过大型计算机。大多数小型机都是通用计算机,能同时向多个用户提供信息处理服务,可并行执行许多应用程序,并比大型计算机便宜且易于安装维护。

微型计算机:也称为微机、个人计算机或 PC(Personal Computer),微型计算机是市场上最便宜和使用最广泛的计算机。微机的体积很小,可放在办公桌上或公文包中。不同微机的价格和性能有很大差异,一些微机的性能超过小型计算机和老的大型计算机。微机可用于多种商务应用。它们既可单独使用,也可与其他计算机连在一起,以扩展功能。

表 1.1 列出了各类计算机的典型配置(存储容量、用户数目等)和大致速度。

表 1.1 计算机的分类

类 型	典型配置	大致速度
微型计算机	64MB 内存、4GB 外存、单用户	每秒处理 1000 万条指令
小型计算机	128MB 内存、10GB 外存、1 个磁带机、128 个用户	每秒处理 3000 万条指令
大型计算机	1GB 内存、100GB 外存、多个磁带机、几百个用户、4 个以上处理器	每秒处理 5000 万条指令
巨型计算机	1GB 内存、100GB 外存、64 个以上处理器	每秒处理 20 亿个浮点运算操作

1.3 计算机硬件

无论计算机是简单还是复杂,都包括 4 个基本组成:输入、处理、输出和存储。计算机也可分成 2 个完全不同的部分:硬件和软件。这 2 个部分互为补充,它们的结合使计算机能实现其基本功能。

图 1.1 给出了计算机系统的 4 个基本组成部分和各部分的典型设备。

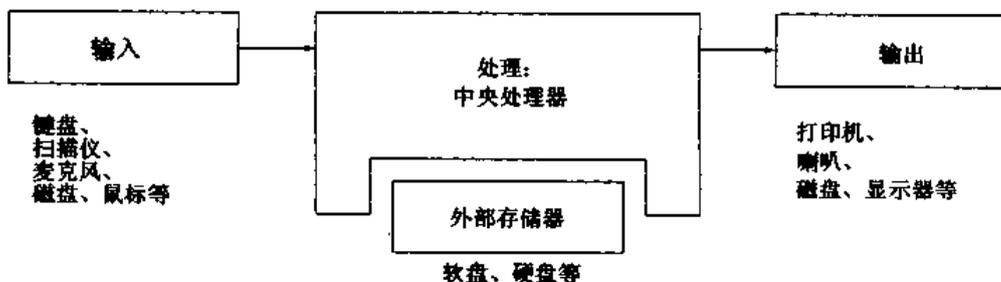


图 1.1 计算机系统的 4 个基本组成部分

大多数计算机系统包括 5 个基本的硬件模块,它们共同实现计算机的基本功能。不同计

计算机系统的硬件模块在数量、实现方式、复杂性和性能上差异很大。但每个模块所执行的功能通常十分相似。这些模块包括：

- 输入设备
- 处理器单元
- 内存
- 外部存储器
- 输出设备

这些模块的组成方式称为系统硬件配置。例如：在一个系统中，处理器单元和外部存储器单元可能放在一个机箱中，而在另一个系统中，它们可能是分离的。

1.3.1 输入设备

输入设备用于向计算机输入指令或数据。有很多种类的输入设备，并且种类还在不断增加。键盘、光笔、扫描仪和鼠标是最常用的输入设备，几乎所有的计算机都有键盘。

【说明】

某些设备既可作为输入设备，也可作为输出设备。例如：磁盘和触摸屏。

1.3.2 处理器单元

处理器单元是计算机系统的智能部分。它也称为中央处理器单元或 CPU (Central Processing Unit)，控制着计算机的行为。CPU 控制任务的执行，如把键盘的输入送到内存、处理数据、把操作结果送到打印机。CPU 包括 3 个基本组成部分：

- 算术逻辑单元 (ALU, Arithmetic and Logic Unit)
- 寄存器
- 控制单元 (CU, Control Unit)

【说明】

CPU 也称为计算机的大脑、心脏或思维模块。

算术逻辑单元：算术逻辑单元或 ALU 是 CPU 中控制所有算术运算和逻辑运算的电路。算术运算包括加法、减法、乘法和除法。乘方、对数等更复杂的运算也能实现。逻辑运算包括字母、数字或特殊字符的比较，以判断它们的关系是大于、等于或小于。

归纳一下，ALU 负责如下功能的完成：

- 执行算术运算
- 执行逻辑运算

寄存器：CPU 内通常包括少量临时存储单元，这些临时存储单元称为寄存器。一个寄存器通常只保存一条指令或数据。寄存器用于存储需要频繁、快速访问的数据和指令。例如：可把 2 个即将相加的数据保存在寄存器中，ALU 读入这 2 个数据进行计算，再把相加的结果保存在另一个寄存器中。因为寄存器位于 CPU 的内部，其内部可被 CPU 内的其他部分快速访问。

归纳一下，寄存器负责如下功能的完成：

- 在 CPU 内部存储指令和数据

控制单元:控制单元是 CPU 内控制和协调其他部分动作以执行程序指令的电路。它并不执行指令本身,而是向相应的电路发送可激活其他部分的电信号。它负责指令执行过程中从内存读取数据和指令,还负责控制 ALU。为了得到 CPU 所需的程序指令和数据,控制单元把它们从内存送到寄存器。

归纳一下,控制单元负责如下功能的完成:

- 激活 CPU 内的其他部分
- 把内存中的指令和数据传送到寄存器

1.3.3 内存

计算机是一个两状态机器。这两种状态可解释为 1 或 0、真或假、上或下等。几乎任何可保存两种状态的设备都可用作存储器。绝大多数计算机都使用可保存二进制数字或位的集成电路存储器。每位可为 0 或 1,表示两种状态中的一种。一台小计算机有可保存数百万位的存储器,而一台大计算机拥有可保存数十亿位的存储器。它们的区别在于存储容量的大小,而存储功能是相同的。

内存可用于存储如下内容:

- 当前程序指令
- 程序要处理的数据
- 执行程序指令时产生的中间结果

【说明】

内存是短期存储元件,只保存程序在运行期间使用的数据。

内存的内容可分为正在执行的程序和相应的数据两部分。程序在执行时要在内存和 CPU 间进行大量的数据传送。

CPU 是一个快速设备,需要可快速访问的设备作为内存。在现在的计算机硬件中,内存是一种称为存储芯片的硅半导体器件。计算机通常有两种内存:

- 随机存储器
- 只读存储器

随机存储器:随机存储器(RAM, Random Access Memory)是计算机的工作存储器。它能提供 CPU 要求的访问速度,允许 CPU 依据地址读写特定位置的存储器。当计算机工作时,程序和数据临时保存在 RAM 中。存储在 RAM 中的数据可被修改和删除。

【说明】

RAM 不是长期稳定的存储设备,不能提供永久存储。当计算机关闭时(或者是其他 RAM 电源中断的情况下),RAM 的内容将丢失。

只读存储器:只读存储器(ROM, Read Only Memory)是第二种内存,用于永久保存计算机生产厂家放置在系统中的程序和数据。CPU 只能从 ROM 中读出指令,而不能修改、删除或重写 ROM 的内容。当计算机关闭时,存储在 ROM 中的数据不会丢失。有时把 ROM 中的程序称为

固件,它界于硬件与软件之间。

数据表示

人们习惯于使用十进制计数系统,以 10 为基础的十进制系统包括从 0 到 9 这 10 个数字。另一方面,计算机使用以 2 为基础的包括 0 和 1 这两个数字的二进制计数系统。

位(二进制数字):每位可保存一个 0 或 1。位(bit)是计算机能理解的最小信息单位。

字节:计算机的存储器必须能保存字母、数字和符号,一个单独的位并不很有用。许多个位组合在一起可表示一些有意义的数。一个 8 位的组合称为字节(byte),可代表一个字符。一个字符可为一个大写或小写字母、一个数字、一个标点符号或一个特殊符号。

ASCII 码:在向计算机输入数据时,系统必须把用户认识的数据(字母、数字和符号)转变成计算机能理解的形式。美国信息交换标准代码(ASCII, American Standard Code for Information Interchange)就是一种用 8 位字节代表字符的编码方案。

【说明】

ASCII 码包括所有大写和小写字母、数字、标点符号和特殊符号,它最多可表示 256 个字符。

字:字节适合存储字符,但不能存储较大的整数。事实上,256 是在一个字节中可保存的最大整数。当然,计算机应能表示较大整数才有用。大多数计算机能处理一种称为字(word)的字节组合。在不同系统中字的大小是不同的,可为 16 位(2 字节)、32 位(4 字节)或 64 位(8 字节)。

存储器的层次结构

如上所述,最基本的存储单位是位,位可组合成字节,字节可进一步组合成字。图 1.2 描述了存储器的层次结构和各种存储单位可表示的整数范围。

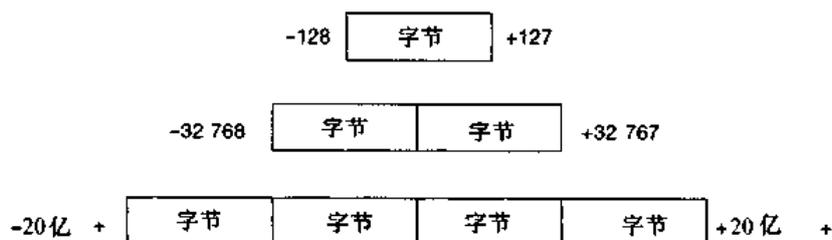


图 1.2 存储器的层次结构

存储容量:字母 K 用来表示内存或磁盘空间的容量单位。K 代表千,即米制长度单位中的千。但在计算机存储容量测量时,K 代表千字节,即 1024 字节存储器(2^{10})。例如:32K 存储器表示 32 768 字节(32×1024)。在描述计算机存储器的容量时,还用到其他一些容量单位:

- 兆字节(MB, megabyte)表示大约 100 万字节
- 千兆字节(GB, gigabyte)表示大约 10 亿字节

存储器地址

内存可被视为连续或相近的存储单元序列。每个存储单元有一个唯一的代表它在存储单元序列中位置的地址。在按字节寻址的计算机中,内存中每个字节都有自己的地址,即一个标识它在内存中准确位置的编号。

在大多数计算机中,内存字节或字的地址是从 0 开始顺序指派的,依次为 0,1 等等。CPU 使用地址表示要读入 CPU 的指令或数据,还用地址表示要写到内存的数据。例如:当从键盘输入字母 H 时,该字符被读入内存并存储在一个特定的地址(如地址 1000)。地址 1000 代表内存中的一个字节,系统可在任何时候引用或处理存储在该地址的数据。

CPU 需要访问内存中不同部分的指令或数据,于是要求内存是直接访问或随机访问设备,即能指定内存读写的位置。

1.3.4 外部存储器

外部存储器是内存的非易失性扩展。内存是昂贵的,在大多数计算机中,内存是不充足的资源。内存是易失的,当电源关闭时,其内容丢失。因此,把程序和数据保存在其他介质上是有意义的。外部存储器可为软盘、硬盘或磁带等。

【说明】

1. 外存是内存的扩展,它并不取代内存。计算机必须首先把磁盘上的程序或数据复制到内存,才能进行程序执行或数据处理。

2. 内存保存当前执行的程序和正在处理的数据,同时外存长期保存程序和数据。

表 1.2 总结了计算机的各种存储设备类型及其通常的存储内容。

表 1.2 存储器的特征

存储器类型	在系统中的位置	用途
寄存器	在 CPU 内部访问速度最快	当前执行指令;指令;部分相关数据
内存	在 CPU 外部 RAM 的访问速度很快	当前执行程序的全部或部分;部分相关数据
外存	低速设备电磁介质或光介质	当前未执行的程序;大量数据

1.3.5 输出设备

基本的输出设备是显示终端。阴极射线管(CRT, Cathode Ray Tube)、视频显示终端(VDT, Video Display Terminal)和显示器都是指显示字符图像的电视类设备。屏幕上的显示图像是暂时的,称为软拷贝。把输出传送到打印机,可得到一个称为硬拷贝的永久性副本。当然,还有其他类型的输出设备,如输出声音的喇叭、绘制图形硬拷贝的绘图仪等。

1.4 指令执行过程

计算机执行一个程序时会出现一个复杂的事件序列。开始时,程序和相关数据被装入内

存。控制单元把第一条指令和它的数据从内存读入到 CPU。如果这条指令是一条计算或比较指令,控制单元通知 ALU 要执行运算、输入数据的位置、输出数据的存储位置。一些指令(如与 I/O 设备、外部存储器间的输入和输出指令)是由控制单元自己执行的。这个过程会一直进行下去,直到从内存读入该程序的最后一条指令到 CPU 中的寄存器并执行完毕。

【说明】

ALU 利用称为指令码或操作码(op code)的整数来区别不同的指令。

指令的执行过程可分成两个阶段:取指令期和执行期。图 1.3 描绘了取指令期和执行期的操作步骤。下面将解释这些操作步骤。

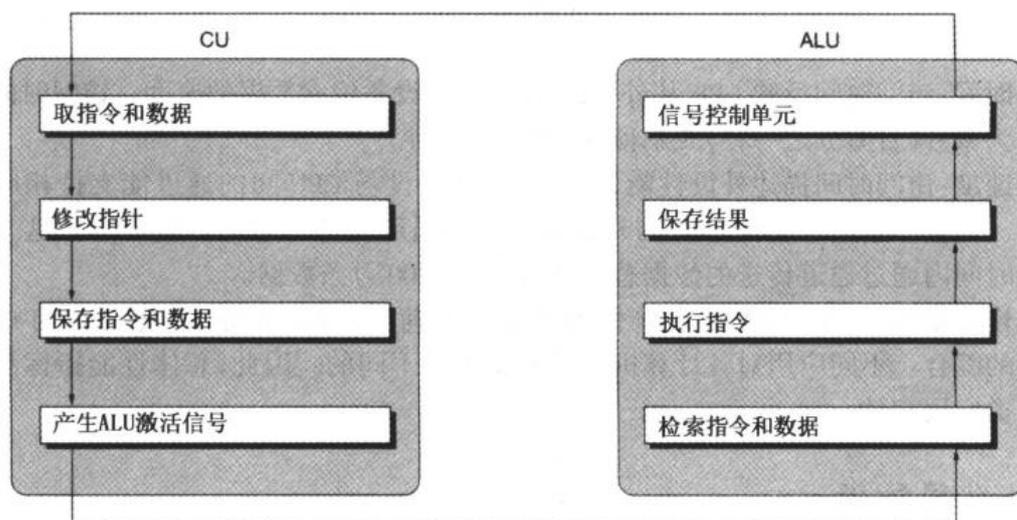


图 1.3 处理器的指令执行操作步骤

取指令期:在取指令期进行的操作可分为以下 3 步:

第一步:控制单元从内存读取一条指令到 CPU 内的一个寄存器,称为指令寄存器。

第二步:控制单元控制指令指针寄存器指向内存中下一条指令的位置。

第三步:控制单元产生信号,通知 ALU 执行该指令。

执行期:在执行期进行的操作可分为以下 3 步:

第一步:ALU 从指令寄存器中得到指令中的操作码,以确定要执行的功能;获取指令的输入数据。

第二步:ALU 执行指令功能。

第三步:把指令的执行结果保存在寄存器中,也可能交由控制单元写入内存单元。

【说明】

取指令期的操作是由控制单元控制执行的,而执行期的操作是由算术逻辑单元控制执行的。

1.4.1 性能指标

大多数计算机是通用计算机,可支持多种类型的应用程序。计算机生产厂商不可能了解每台计算机在实际应用中的工作量并向用户提供相应的性能分析。但计算机生产厂商分析计

算机的各种操作性能,如指令执行、内存访问、磁盘读写等方面的性能特征。性能指标通常是针对每台计算机的组成部件、各部件间的通信能力和所有性能指标的综合测量。

CPU 速度:针对 CPU 的最初性能指标是指令的执行速度,该指标可描述成每秒执行多少百万条指令(MIPS, Millions of Instructions Per Second)。但遗憾的是,不同指令的执行时间各不相同。加法指令等一些指令的执行很快,而分数(浮点数)除法等另外一些指令的执行要慢得多。专门描述分数计算性能的指标是每秒执行多少百万个浮点计算(MFLOPS, Millions of Floating-Point Operations Per Second)。

【说明】

指针 MFLOPS 通常用于工作站和巨型计算机。在这些计算机上应用程序通常包含大量浮点计算。

访问时间:访问时间反映 CPU 从外存或输入/输出设备检索数据的速度。访问时间的测定单位通常为微秒(百万分之一秒)或纳秒(10 亿分之一秒)。

通道速度:访问时间描述外设性能,但不能保证外设与 CPU 间的通道能支持相应速度的数据传送。数据传输速率反映 CPU 与外设间的通信通道支持的数据传输能力。这个速率是指在一定时间内通过通道传送的数据总量,例如每秒 100 万条数据。

总体性能指标:计算机系统的总体性能是指 CPU 速度、外设访问时间和外设与 CPU 间的通道速度的综合。不同应用对各计算机部件的要求是不同的。因此,总体性能指标只对特定实例或一类应用有效。

1.5 什么是软件

没有软件的计算机仅仅是一台机器(硬件),并没有功能。软件给计算机带来了多种多样的功能。计算机只有利用软件才呈现出多种个性。利用相应软件,计算机可成为一个文字处理机、计算器、数据库管理系统、智能通信设备等,甚至同时具有上述多种功能。

【说明】

通常把计算机程序称为软件。在同一台机器上,通过运行不同程序可完成不同的功能。

程序:程序是控制计算机系统行为的指令集合。它由一组执行特定操作的指令序列构成。程序是用计算机编程语言写成的。计算机编程语言对方便应用程序开发十分必要。有大量程序开发工具可用于辅助程序员开发应用程序和系统程序。

软件分类:计算机软件可分成应用软件和系统软件两类。操作系统是最重要的系统软件,也是计算机完成其功能所必需的软件。然而,使计算机在不同环境下完成多种有用工作的软件是应用软件。图 1.4 给出软件的通常分类和相应的软件实例。

用户通常与应用软件和部分系统软件进行交流。图 1.5 是软件的一种分层划分。这种划分的好处在于用户和应用开发人员不需要了解和处理物理过程的技术细节。基于这种划分,可对用户屏蔽机器的硬件细节和系统软件内的许多基本操作。

1.5.1 系统软件

系统软件是控制计算机内部主要功能的程序集合。最重要的系统软件是操作系统,它控

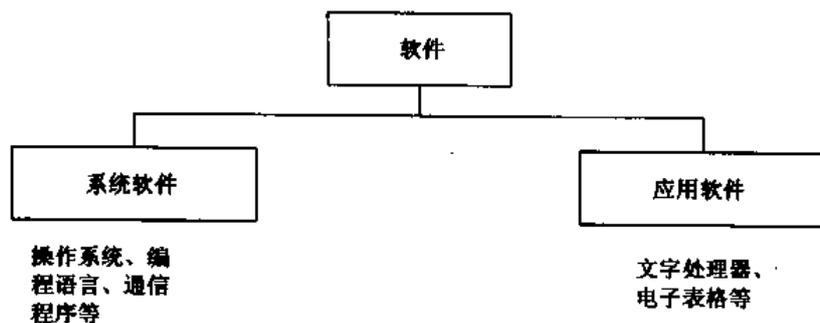


图 1.4 软件分类

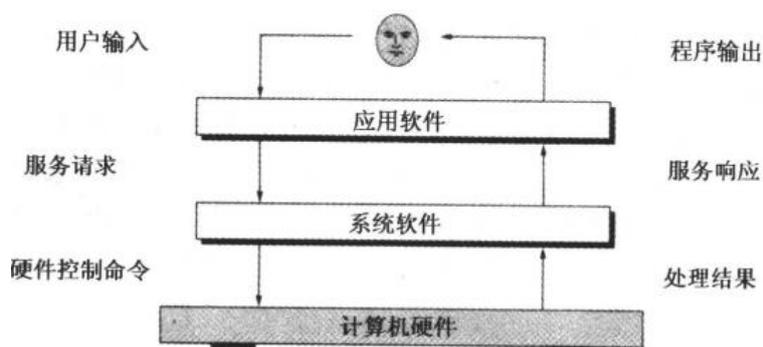


图 1.5 用户与软件层次划分的相互作用

制计算机的基本功能,向应用程序提供一个基本环境。数据库管理系统(DBMS, Database Management System)和通信软件等也是系统软件。

谁是系统控制者

操作系统是系统控制者,也是计算机系统软件中最重要的组成部分。它是计算机中所有软件和硬件的控制程序的集合。操作系统的主要部分在计算机加电时装入内存,并一直保留到关闭计算机电源。在计算机系统中,操作系统扮演着提供服务、管理硬件和维护用户接口等多种角色。我们不打算给出一个广泛接受的操作系统的定义,而是探讨它的角色和任务。操作系统的主要用途和功能包括:

- 向用户和应用程序提供一个控制底层硬件功能的接口
- 向各用户的应用程序分配硬件资源
- 按用户要求加载和执行应用程序

【说明】

操作系统的基本部分是一直驻留在内存中的。

作为资源管理者的操作系统

操作系统负责控制内存、CPU 时间、外设等计算机资源。在典型的计算机系统中通常有多个并发执行的任务,相互竞争使用计算机资源。依据资源的使用状态和运行程序的优先级,操作系统向各程序分配资源。操作系统负责 CPU 时间的分配,同时存在多个运行程序,但

CPU 只有一个,因而任何时刻只有一个运行程序占用 CPU。

各程序的大小和内存需求都不同,操作系统负责监视和分配程序占用的内存,避免同时在内存的不同程序间出现内存使用混乱。操作系统协调外设的使用。例如:选择先使用打印机的作业和磁盘读写的顺序。

【说明】

操作系统连续不停地响应程序的资源请求,解决资源冲突,并优化资源分配。

作为用户接口的操作系统

操作系统提供用户与计算机进行交流的手段。每个操作系统都提供一组操作计算机的命令,称为命令式用户接口。用户很难学习、记忆和使用复杂的命令语法。命令式用户接口有两种变形。一种变形是菜单式用户接口,它提供一组让用户选择操作功能的菜单。另一种变形是图形用户接口(GUI, Graphic User Interface),它提供基于图形的图标式用户接口,通过控制屏幕上各种图形来执行大多数操作系统命令。例如,可用文件夹中的图标表示文件,用打字机图标表示某个文字处理程序。这种通过可视图标表示命令和操作功能的方法提供了一个简单易学的用户接口。在 GUI 方式下,用户可利用鼠标移动和点击图标来激活程序。

操作系统模型

从软件的层次划分角度来讲,可把操作系统看作一个分层的软件集。图 1.6 是一种操作系统的分层模型。用户或应用程序的输入通过这个分层结构到达硬件机构,从硬件机构返回的结果再通过相同的分层结构反馈给用户。下面讨论操作系统中各层的功能。

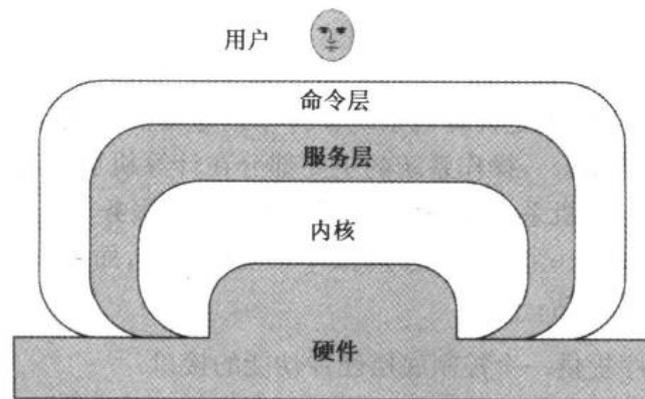


图 1.6 操作系统的分层结构

内核层:内核是操作系统软件的最内层。它是惟一与硬件直接交流的部分。这提供了一种使操作系统与机器硬件独立的方法。至少从理论上讲,只修改内核部分就可使同一操作系统在不同硬件环境下运行。内核提供最基本的操作系统功能。例如,加载和执行程序,向各程序分配 CPU 时间及磁盘访问等硬件资源。把软件与硬件的信息交流限制在内核层,可实现应用层用户与硬件的隔离,用户不需要直接了解硬件规范。

服务层:服务层接受来自命令层或应用程序的服务请求,并转换成送给内核的详细的操作。如果有处理结果返回,服务层还要把相应结果送到提出请求的程序。服务层包括一进程

序,提供4类服务:

- I/O 设备访问:如应用程序到打印机或终端的数据传送
- 存储设备访问:如磁带机或磁盘到应用程序的数据传送
- 文件操作:如打开文件、关闭文件、读文件和写文件
- 其他服务:如窗口管理、通信网络访问、基本数据库服务

命令层:命令层也称为命令解释程序(shell),它是操作系统的最外层,提供用户接口,是操作系统中与用户直接进行信息交流的部分。命令层提供对操作系统命令集的支持。命令集及其语法规则称为命令语言,有多种命令语言变种。

操作系统环境

有多种途径可实现操作系统的功能。在单用户环境(如多数微机)下,所有可用资源都分配给一个程序,这个程序是机器上的惟一运行程序。在较大的计算机和联网的微机上,计算机的资源是多个用户共享的,操作系统必须解决多个程序同时申请使用同一资源时产生的冲突。下面讨论几个描述操作系统特征和差异的基本概念和术语。

【说明】

虽然在微机内存中可有多个程序,但在一个时刻只有一个程序在执行。

单任务:单任务操作系统是为运行单个进程而设计的。它通常是在微机上运行的针对某种特定应用的单用户环境。

多任务:多任务操作系统能够在同一时间段内执行一个用户的多个程序。当在操作一个前台任务的同时,操作系统可同时在后台执行多个程序。例如,当用户控制操作系统在后台对一个大文件进行排序时,可在前台使用一个文字处理程序输入备忘录;操作系统在后台任务完成时发一个通知。多任务是指操作系统允许一个终端用户并发执行多个程序。

多用户:在多用户环境下,多个用户(终端)使用同一台主机。多用户操作系统是同时向多个用户提供服务的复杂软件。虽然机器中只有一个CPU,它在一个时刻只能执行一个程序,但利用计算机与外设(磁盘、打印机等)间的速度差异,多用户操作系统可使多个用户程序在内存中同时运行。与处理器的速度相比,输入/输出设备是非常慢的。于是,在一个程序等待I/O操作时,处理器有大量时间来执行其他在内存中的程序。用户感觉不到程序的切换过程,就好像只有用户自己在使用整个计算机系统。

图1.7是一个多用户系统的实例,它有4个用户、1台打印机和一些其他I/O设备。所有用户共享同一主机和它的资源。

【说明】

多用户操作系统能够利用同一主机向许多用户(终端)提供服务。

另一种构建多用户系统的方法是把多台计算机连在一起,形成一个计算机网络。图1.8所示的计算机网络系统包括4台计算机,其中3台计算机为独立的单用户主机,另一台作为文件服务器。

分时:分时操作系统是为包含快速处理的在线处理环境而设计的,涉及多用户共享同一主机的处理时间。分时操作系统在多个任务间快速切换,轮流为每个用户任务分配时间片,每个

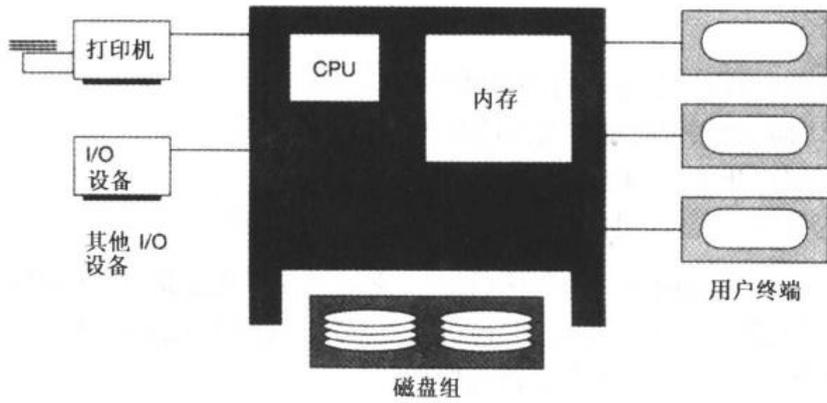


图 1.7 一个多用户计算机系统

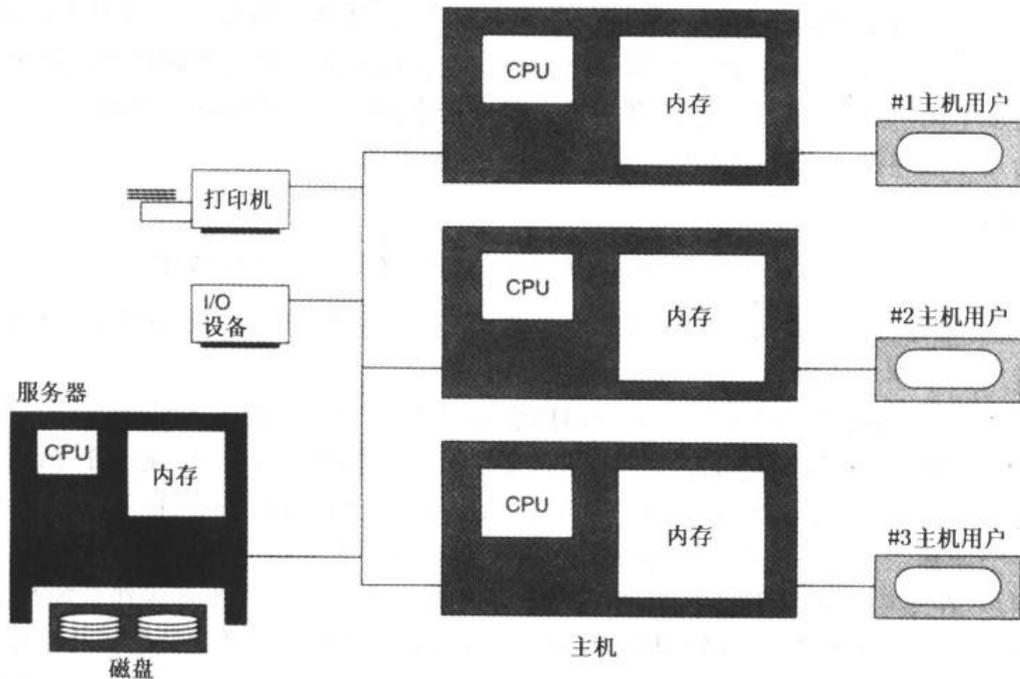


图 1.8 网络环境下的多用户系统

时间片内执行用户任务的一小部分。

批处理:批处理操作系统是为那些不需要用户交互的程序执行(批处理)而设计的。批处理通常使用非交互式 I/O 设备。例如,可从磁盘或纸带读入数据,并把结果返回到相同外设。批处理仅用于大型事务处理环境,如银行中每天晚上的票据处理。

内存容量限制

一个计算机程序有多大? 如果在执行的全过程中,整个程序和相关数据必须在内存中,则计算机的内存容量是应用程序大小的上限。但实际情况并不是这样。计算机是一个顺序处理机构,在一个指令执行周期只能执行一条指令,没有必要把整个程序一直放在内存中。

虚拟存储:使用虚拟存储技术时,程序被分成许多称为页的小块。为了程序执行,操作系统只把少量必需的程序页交换到内存。这种方法允许用户在较小的内存空间上运行相对较大

的程序。

【说明】

虚拟存储系统把外部存储设备当作内存的扩展。

1.5.2 应用软件

应用软件的目的是解决实际问题以及提高工作效率和自动化程度,可用于日常生活、商务和科研等方面。应用程序可满足各种数据处理的需求。用户可购买到各种现成的应用软件,如财务软件、仓库管理软件、文字处理软件、电子表格等等。用户的惟一工作是选择适合需要的应用软件。如果不喜欢现成的应用程序,用户还可以用各种计算机编程语言自己写程序。

习题

1. 什么是 CPU? 它由哪几个部分组成?
2. 什么是寄存器? 它有什么作用?
3. 什么是内存?
4. 分别解释取指令期和执行期的步骤。
5. 什么是计算机系统的指令集?
6. 解释单任务和多任务的概念,说明其区别。
7. 解释单用户和多用户的概念,说明其区别。
8. 什么是系统软件?
9. 什么是应用软件?
10. 操作系统由哪几个功能模块组成?
11. 解释操作系统的软件层结构。
12. 什么是内核? 它的功能是什么?
13. 什么是服务层? 它的功能是什么?
14. 内存与外存的区别是什么?
15. 解释虚拟存储的含义。为什么要引入虚拟存储技术?

第 2 章 UNIX 操作系统

本章将简要介绍 UNIX 操作系统的历史和发展过程,讨论主流的 UNIX 版本和重要的系统特征。

2.1 UNIX 操作系统:历史简介

在 20 世纪 60 年代早期,许多计算机都采用批处理方式,只能执行单个作业。程序员只能使用穿孔纸带输入程序,然后等待行式打印机输出结果。UNIX 操作系统诞生于 1969 年,其目标是解决程序员面临的困境,并寻求可以帮助程序员完成工作的新的计算工具。

UNIX 操作系统是由贝尔实验室的两位研究人员 Ken Thompson 和 Dennis Ritchie 首先开发出来的。当时, Ken Thompson 正在开发一个称为太空旅行的程序,模拟太阳系的行星运动。这个程序运行在 Multics 操作系统上,该操作系统是第一代在 General Electric 6000 系列计算机上提供多用户环境的操作系统。由于 Multics 操作系统大而慢,还要占用大量计算机资源,于是 Thompson 找到一台较小的计算机,把太空旅行程序传到这台机器上运行。这台计算机就是由数字设备公司(DEC, Digital Equipment Corporation)生产的 PDP-7。在这台计算机上, Thompson 采用了 Multics 中的一些先进概念,开发了一种称为 UNIX 的新操作系统。其他操作系统也或多或少地具有类似的特征,但通过组合这些操作系统中最有价值的部分,UNIX 很好地利用了这些操作系统的工作成果。

1970 年,UNIX 被移植到 PDP-11/20 计算机上,随后又被移植到 PDP-11/40、PDP-11/45 和 PDP-11/70 上。在这个过程中,随着机器硬件的逐渐复杂,UNIX 所支持的特征不断丰富。Dennis Ritchie 和其他贝尔实验室的研究人员继续开发 UNIX,增加文字处理程序等应用程序。

与大多数操作系统类似,UNIX 最初是用汇编语言开发的。汇编语言是一种依赖机器体系结构的低级编程语言。用汇编语言编写的程序是与机器相关的,只能在一种或一类计算机上运行。因此,把 UNIX 从一种计算机移植到另一种计算机需要重写大量的代码。

Multics 的代码是用一种称为 PL/1 的高级编程语言编写的, Thompson 和 Ritchie 是 Multics 的专家,他们了解用高级编程语言开发操作系统的好处(如:高级语言比汇编语言更容易使用)。他们决定用高级语言重写 UNIX 操作系统。他们选择的高级语言是 C 语言。C 语言是一种具有高级命令和结构的通用编程语言。1973 年, Ken 和 Dennis 成功地用 C 语言重写了 UNIX 操作系统。

【说明】

UNIX 操作系统中 95% 的代码是 C 语言代码;只有很小的一部分是汇编语言代码,这部分是内核中直接与硬件打交道的代码,十分精练和高效。

大学在 UNIX 操作系统的推广过程中起了重要作用。1975 年,贝尔实验室以很低的价格向教育机构提供了 UNIX 操作系统。UNIX 课程成为计算机专业的大学课程,学生们逐渐熟悉

了 UNIX 和它成熟的编程环境。当学生毕业后进入工作岗位时,把他们所受的 UNIX 训练带入商业领域,进而把 UNIX 引入工业领域。

UNIX 操作系统有 2 个主要版本:

- AT&T UNIX 系统 V
- Berkeley UNIX

其他的 UNIX 变种都是基于这两个版本的。

2.1.1 UNIX 系统 V

1983 年 AT&T 发布标准的 UNIX 系统 V,它是基于 AT&T 内部使用的 UNIX 系统开发的。随着 UNIX 开发的推进,一些已有特性得到改进,也不断有新特性被引入。经过多年发展,UNIX 系统 V 变得越来越大,同时出现了大量 UNIX 系统 V 上的系统工具和应用程序。许多改进的和新的特性被加入到 UNIX 操作系统的后续版本中。1983 年发布了 UNIX 系统 V 的第 3 版,1988 年发布了 UNIX 系统 V 的第 4 版。

UNIX 系统 V 的第 4 版把许多 Berkeley UNIX 和其他 UNIX 系统的流行特性加入到系统 V 中。这次合并简化了 UNIX 产品,减少了生产厂商开发新的 UNIX 变种的必要。

【说明】

本书讨论的命令适用于 UNIX 系统 V 第 4 版(SVR4,UNIX System V Release 4)。

2.1.2 Berkeley UNIX

美国加州大学伯克利分校的计算机系统研究中心对 UNIX 操作系统进行了重大改进,引入了许多新特性。他们的 UNIX 版本称为 BSD(Berkeley Software Distribution)版本,在大学中得到广泛使用。

2.1.3 UNIX 标准

UNIX 操作系统可用于从微机、小型机、大型机到巨型机的所有种类的计算机,是一种重要的计算机操作系统。随着市场上多种基于 UNIX 的系统和应用程序的出现,人们开始对 UNIX 进行标准化。AT&T 的 UNIX 系统 V 第 4 版是 UNIX 系统标准化的一个里程碑,它推动了可在所有 UNIX 版本上运行的应用程序的开发。AT&T 的 UNIX 标准称为系统 V 用户接口定义(SVID, System V Interface Definition)。其他一些 UNIX 操作系统和 UNIX 相关产品厂商联合开发了一个称为计算环境中的可移植操作系统接口(POSIX, Portable Operating System Interface for Computer Environments)。POSIX 在很大程度上是基于 SVID 的。

2.2 其他 UNIX 系统

几乎所有主要的 UNIX 厂商都提供基于 UNIX 系统 V 的 UNIX 版本。大多数 UNIX 变种的大量命令和特征都与 SVR4 相似。下面是这些 UNIX 变种的简要介绍。

2.2.1 AIX

AIX 是 IBM 公司的 UNIX 操作系统版本。它与 SVR4 相似,并针对 IBM 的机器进行了优化和增强。

2.2.2 HP-UX

HP-UX 是惠普公司的 UNIX 操作系统版本。它是为在惠普计算机和工作站上使用而开发和销售的。它是基于 UNIX 系统 V 第 2 版开发的。

2.2.3 LINUX

UNIX 操作系统的 Linux 变种是芬兰赫尔辛基大学计算机科学专业的一个学生 Linux Torvalds 的天才想法,它是为基于 Intel 处理器的个人计算机而设计的。Linux 发布以来,许多人一直在改进和增强它。不像其他 UNIX 版本, Linux 是 UNIX 的一个免费使用版本。它具有许多 UNIX 系统 V 的特性并进行了许多增强,是在个人计算机上十分流行的 UNIX 版本。

2.2.4 Solaris

SunOS,后称为 Solaris,是 Sun 公司基于 UNIX 系统 V 第 2 版和 BSD 4.3 开发的操作系统。Solaris 2.0 是基于 SVR4 的,现在的 Solaris 版本为 Solaris 2.4,该版本的装机量十分大。Solaris 2.4 上有许多图形用户界面的系统工具和应用程序。

2.2.5 UnixWare

Novell 公司的 UNIX 版本是基于 UNIX 系统 V 的,称为 UnixWare。Novell 把 UnixWare 卖给了 Santa Cruz Operation(SCO)公司,现在是 SCO 公司提供 UnixWare 及其相关产品。UnixWare 有 2 种版本: UnixWare 个人版和 UnixWare 应用服务器。前者是为台式机设计的,后者用于服务器。UnixWare 只用于使用 Intel 处理器的计算机,UnixWare 上有许多应用软件。

2.3 UNIX 操作系统概要

如前所述,一个典型的计算机系统包括硬件、系统软件和应用软件这三部分。操作系统是控制和协调计算机行为的系统软件。与其他操作系统相同,UNIX 操作系统也是一个程序的集合,其中包括文本编辑器、编译器和其他系统程序。

图 2.1 给出了 UNIX 操作系统的结构。UNIX 操作系统采用的是分层结构。

内核:UNIX 内核,也称为基本操作系统,负责管理所有与硬件相关的功能。这些功能由 UNIX 内核中的各个模块实现。内核包括直接控制硬件的各模块,是系统中最重要的一部分。用户不能直接访问内核。

【说明】

1. 系统工具和 UNIX 命令都不是内存的组成部分。
2. 操作系统保护各用户的应用程序不会被其他用户的无意写操作破坏。

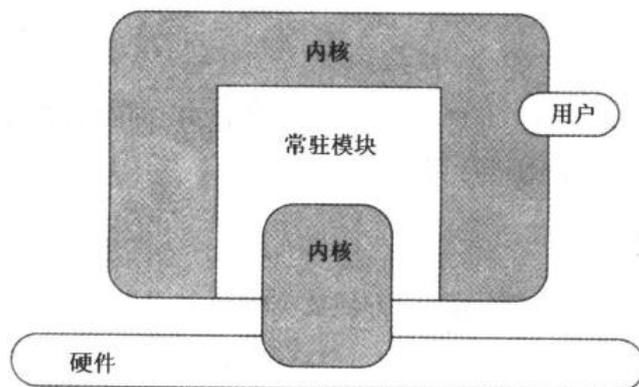


图 2.1 UNIX 系统结构

常驻模块层:常驻模块层提供执行用户请示的服务例程。它提供的服务包括输入/输出控制服务、文件/磁盘访问服务以及进程创建和中止服务。用户程序通过系统调用访问常驻模块层。

工具层:工具层是 UNIX 的用户接口,通常称为 shell。shell 和其他 UNIX 命令和工具都是单独的程序,它们是 UNIX 系统软件的组成部分,但不是内核的组成部分。UNIX 中有 100 多个命令和工具,向用户提供各种类型的服务和应用程序。

虚拟计算机:UNIX 操作系统向系统中的每个用户指定一个执行环境。这个称作虚拟计算机的环境包括一个与用户进行交流的终端和共享的其他计算机资源,如内存、磁盘和最重要的 CPU。作为多用户操作系统,UNIX 可视为一个虚拟计算机的集合。对用户而言,每个用户都有一个自己的专用虚拟计算机。由于 CPU 和其他硬件资源是多个虚拟计算机共享的,虚拟计算机比真实的计算机要慢一些。

进程:UNIX 操作系统通过进程向用户和程序分配资源。每个进程有一个作为进程标识的整数和一组相关的资源。进程是在虚拟计算机环境中执行的,即进程在虚拟计算机上执行就好像它在一个专用的 CPU 上执行一样。

2.4 UNIX 系统特征

本节简要讨论 UNIX 操作系统的一些特征。UNIX 的一些特征是大多数操作系统所共有的,而另一些特征是它自己所特有的。

2.4.1 可移植性

C 语言的使用使 UNIX 成为一种可移植操作系统。今天,UNIX 操作系统可运行在从微机到巨型机的各种计算机上。可移植特征减少了用户更换系统的学习时间,也提供了在各种硬件厂商间进行选择的机会。

2.4.2 多用户

在 UNIX 中,大量用户可同时共享计算机资源。依据所使用的机器,UNIX 可支持 100 个以上的用户同时使用,各用户可执行不同的程序。UNIX 提供安全机制,以确保各用户仅能访问各自有权限访问的数据和程序。

2.4.3 多任务

UNIX 允许用户在启动一个任务后,继续执行其他任务,同时原任务还在后台执行。UNIX 允许用户在前后台的多个任务间进行切换。

2.4.4 多级文件系统

UNIX 支持对数据和程序进行分组,以方便数据管理。用户可方便地查找数据和确定程序的位置。

2.4.5 与设备独立的输入和输出操作

由于把打印机、终端、磁盘等所有设备都视为文件,UNIX 的输入和输出操作是与设备独立的。在 UNIX 中,用户可把命令输出重定向到任何设备或文件。重定向也可用于数据输入,用户可把终端输入重定向成从磁盘读入。

2.4.6 用户界面:shell

UNIX 用户界面最初是为有开发背景的用户设计的。有经验的程序员会发现 UNIX 的简单、精练和优雅。另一方面,初学者会觉得 UNIX 很简练、不友好并且很难学,它没有反馈、警告或确认。例如,命令“`rm *`”会直接删除所有文件而不给出任何提示。

用户与 UNIX 系统的交互是由一个功能强大的命令解释程序 shell 控制的。shell 是 UNIX 系统对外的接口界面和与用户交互最多的部分。但 shell 只是操作系统的一部分,是用户与 UNIX 交互的一种方式,不是操作系统的核心组成部分,可对它进行改动。用户可选用一个命令式用户界面,也可从菜单中选择命令(菜单式用户界面),或用鼠标点击图标(图形用户界面)。用户甚至可开发自己的用户界面。

UNIX shell 是一个复杂而成熟的用户界面,它支持许多已有的或创新的特性。用户可通过已有命令的组合而形成新的命令。例如,由 2 个命令“`data`”和“`lp`”组合而成的命令“`data | lp`”,可把当前日期打印到打印机上。

shell 脚本:许多数据处理程序是经常使用的,如每天、每周或其他时间间隔周期性地使用。在另一种情况下,一些命令要重复键入许多次。一次又一次地重复键入相同的命令会使人烦躁,且容易出错。解决这个问题的一种方法是使用 shell 脚本。shell 脚本是由一系列命令组成的文件。

UNIX shell 是一种很成熟的编程语言,第 11 章和第 12 章将讨论 UNIX shell 脚本的描述能力和编程方法。

2.4.7 系统工具

UNIX 系统包括 100 多个系统工具,也称为命令。系统工具是标准 UNIX 系统的组成部分,可完成用户所需的各种功能。这些系统工具包括:

- 文本编辑和格式化程序(第 4 章和第 6 章)
- 文件管理程序(第 5 章和第 7 章)
- 电子邮件程序(第 9 章)

- 开发工具程序(第10章)

2.4.8 系统服务

UNIX 系统提供许多用于系统管理和维护的系统服务。对这些系统服务的讨论超出了本书的范围。下面是一些 UNIX 系统服务：

- 系统管理服务
- 系统配置服务
- 文件系统维护服务
- 文件传输服务(UUCP, UNIX to UNIX Copy)

习题

1. UNIX 系统的主要版本是哪两种？
2. 什么是内核？
3. 什么是 shell？
4. 简要解释虚拟计算机的概念。
5. 什么是进程？
6. 为什么要重写 UNIX？重写时使用的是什么编程语言？
7. UNIX 是多用户、多任务操作系统吗？
8. UNIX 可移植吗？
9. 什么是 shell 脚本？
10. 列出一些 UNIX 变种。
11. 你的 UNIX 系统是什么版本？

第3章 UNIX 入门

本章将介绍如何登录和退出 UNIX 系统,解释口令的功能,介绍如何修改口令。在说明命令行格式时,会给出一些命令实例,并解释其用途。本章也会介绍如何更正键盘输入错误。最后,详细介绍 UNIX 系统的登录过程和一些 UNIX 系统的内部操作。

3.1 UNIX 系统的登录和退出

UNIX 操作系统上的登录和退出过程包括一系列的提示信息显示和用户输入。一个会话是指一次用户使用计算机的全过程。

3.1.1 登录

UNIX 是一个多用户操作系统,你不太可能是系统的惟一用户。系统识别用户身份并允许用户使用的过程称为登录过程。用户在使用 UNIX 系统前必须进行登录。

为了使用 UNIX 操作系统,用户需要一种与系统进行通信的方法。在大多数情况下,这种通信是通过键盘输入和屏幕显示输出来实现的。用户打开计算机的显示屏幕后,从按回车键开始登录过程。当 UNIX 系统完成登录准备时会显示一些信息(不同系统的显示信息会有所不同),并给出 **login:** 提示(图 3.1)。

```
UNIX System V release 4.0
login:
```

图 3.1 登录提示

登录名:大多数 UNIX 系统要求用户在使用计算机前建立用户账号。用户账号包括登录名(也称为用户标识或用户名)和口令等信息。用户标识通常是由系统管理员指定的(在大学的计算机系统上也可能由导师指定)。用户标识是惟一的,也是系统对用户的表示。

作为对 **login:** 提示的响应,用户输入自己的用户标识并按回车键。输入用户标识(例子中的用户标识为 **david**)后,UNIX 系统会显示 **password:** 提示(见图 3.2)。

```
UNIX System V release 4.0
login: david
password:
```

图 3.2 口令提示

如果用户账号没有设置口令,UNIX 不会显示 **password:** 提示。至此,登录过程完成。

口令:与用户标识类似,口令是由系统管理员提供的。口令是一个由字母和数字组成的序列,UNIX 系统用它来验证是否允许使用用户标识。

用户输入口令并按回车键。为了保护口令不被别人窃取,UNIX不会回显用户输入的口令,即从屏幕上看不到口令的。UNIX系统对用户标识和口令进行验证,通过验证后,系统会为用户初始化系统环境。

用户标识和口令验证后,UNIX系统将显示一些信息,通常是日期和系统信息(如系统管理员给用户的信息)。UNIX系统通过显示命令提示符来表示可接受用户的命令输入。标准提示通常是一个美元符号\$或百分号%。我们假设命令提示符为美元符号\$,它会显示在每一行的第一个字符处(见图3.3)。

```
UNIX System V release 4.0
login: david
password:

Welcome to super duper UNIX system
Sat Nov 29 15:40:30 EDT 2001
* This system will be down from 11:00 to 13:00
* This message is from your friendly system administrator!

$_
```

图 3.3 登录过程

3.1.2 修改口令:passwd 命令

passwd 命令的功能是修改用户的当前口令。如果没有口令,系统会创建一个口令。输入 passwd 并按回车键,系统会显示 Old password:(旧口令)提示(见图3.4)。

```
$ passwd
Changing password for david
Old password:
```

图 3.4 旧口令提示

【注意】

出于系统安全原因,UNIX系统永远不会在屏幕上显示用户口令。

UNIX系统通过验证用户的旧口令来确保没有未经授权的用户修改口令。如果没有口令,系统不会显示 Old password:提示。输入当前口令并按回车键,系统会显示 New password:(新口令)提示(见图3.5)。

```
$ passwd
Changing password for david
Old password:
New password:
```

图 3.5 新口令提示

用户输入新口令后,系统会显示 Re-enter new password:(再输入一次新口令)提示。这

样,系统可验证用户在第一次输入新口令时没有错误(见图 3.6)。

```
$ passwd
Changing password for david
Old password:
New password:
Re-enter new password:
```

图 3.6 再输入一次新口令提示

如果用户两次输入的新口令相同,UNIX 系统将修改用户口令。

【注意】

请牢记新口令。在下一次登录时系统会要求用户输入新口令。

口令格式:passwd 命令不会把任何字符序列当成口令。用户口令必须满足下列条件:

- 新口令与旧口令至少有 3 个字符不同
- 口令长度至少为 6 个字符,其中至少有 2 个字母和 1 个数字
- 口令不能与用户标识相同

如果检测到任何一个条件不满足,系统都会显示一条错误信息,并再一次显示 **New password:**提示(见图 3.7)。

```
Password is too short - must be at least 6 digits (口令太短,口令必须至少有 6 个字符)
New password:
Password must differ by at least 3 positions (口令至少要有 3 个字符不同)
New password:
```

图 3.7 口令错误信息

3.1.3 退出系统

用户完成在 UNIX 系统上的工作时的退出过程,称为退出系统或退出。用户只能在显示命令提示符时退出系统,而不能在进程执行过程中退出系统。退出系统的操作为,在命令提示符下按组合键[Ctrl-d](即同时按下 Ctrl 键和字符键 d)。按下该组合键时屏幕并不显示任何信息,但系统能识别该退出命令。系统对退出命令的响应动作是显示由系统管理员设置的退出信息,随后系统显示标准的欢迎信息和登录提示。这使用户确信已正确退出系统并可开始下一次用户登录。

【实例】

图 3.8 所示的实例给出了登录和退出系统的过程。

【注意】

在没有退出系统前直接关掉终端电源并没有结束用户与 UNIX 系统间的交互。

```
UNIX System V release 4.0
login: david
password:
Welcome to super duper UNIX system
Sat Nov 29 15:40:30 EDT 2001
* This system will be down from 11:00 to 13:00
* This message is from your friendly system administrator!

$[Ctrl-d]

UNIX system V release 4.0
login:
```

图 3.8 登录和退出过程

3.2 一些简单的 UNIX 命令

在 UNIX 系统上有几百个命令和系统工具。一些基本命令是频繁使用的,另一些是偶尔用到的,还有一些命令可能有的用户永远也用不到。对于 UNIX 这一类有大量命令的操作系统,要掌握每一个命令细节是十分困难的;但大多数 UNIX 命令的基本结构相同,而且大多数 UNIX 系统都有可帮助用户记忆的在线帮助信息。

3.2.1 命令行

每个操作系统都有一些方便用户使用系统的命令。通过输入命令,用户可控制 UNIX 系统完成一定功能。例如, **date** 命令的功能是让系统显示日期和时间,为此要输入 **date** 并按回车键。这一行命令称为命令行。UNIX 把回车键解释为命令行的结束符,随后在屏幕上显示如下的当前日期和时间:

```
Sat Nov 29 14:00:52 EDT 2001
$ _
```

用户每输入一个 UNIX 命令,系统就执行相应的功能,然后显示一个新的命令提示符,表示可输入下一个命令。

3.2.2 基本的命令行格式

每个命令行分成 3 个字段:

- 命令名
- 命令选项
- 命令参数

图 3.9 为 UNIX 命令的通用格式。

【说明】

1. 字段间用一个或多个空格分开;

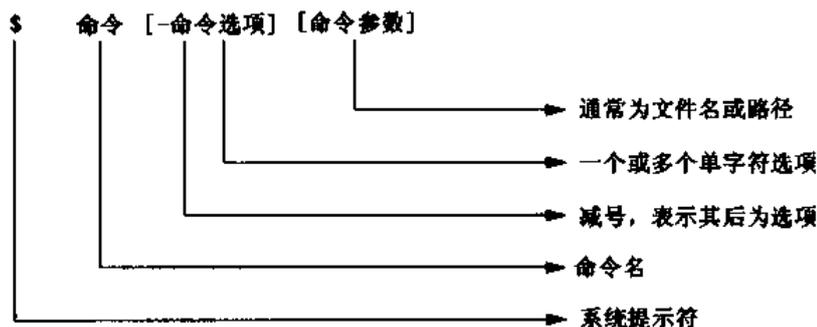


图 3.9 命令行格式

2. 在大多数命令格式中,放在方括号内的字段表示命令选项。

【注意】

在每个命令的结尾,必须按回车键,表示命令输入过程的结束。

命令名:任何 UNIX 命令或系统工具都可作为命令名。在 UNIX 系统中,UNIX 命令和系统工具是不同的,但在本书中,它们都被视为命令。

【注意】

UNIX 系统对大小写敏感,只接受小写命令名。

命令选项:命令选项表示命令变种。如果命令行中有选项,它的前面通常有一个减号 (-)。大多数命令选项是单个小写字母,一个命令行中指定多个命令选项。本书并不详细讨论每个命令的所有可能选项。

命令参数:命令参数表示命令的操作对象或类型,如打印文件名或显示信息等。时常需要在命令中指定操作对象,即用户需要说明的附加信息。表示命令操作对象的附加信息称为命令参数(argument)。例如,在打印命令 `print` 中,用户要说明打印文件的名称和表示打印文件在磁盘上位置的路径;这里的文件名和路径就是命令参数。

3.2.3 显示日期和时间: date 命令

如图 3.10 所示, `date` 命令在屏幕上显示当前日期和时间。日期和时间是由系统管理员设置的,普通用户不能修改。

```
$ date
Sat Nov 29 14:00:52 EDT 2001
$_
```

图 3.10 date 命令

【实例】

显示当前日期和时间。

【说明】

`date` 命令显示星期、月、日、时间(美国东部时间)和年。UNIX 使用 24 进制时间。

3.2.4 用户信息:who命令

who命令可列出当前登录到系统的所有用户的登录名、终端号和登录时间。可用**who**命令检查系统状态,或某个用户是否正在使用系统(见图3.11)。

```
$ who
david      tty04      Nov 28  08:27
daniel     tty10      Nov 28  08:30
$_
```

图3.11 who命令

【实例】

显示系统的当前用户列表。

【说明】

1. 第一列显示用户的登录名。
2. 第二列标识用户使用的终端。
3. 终端号可给出终端位置的某种指示。
4. 第三、四列给出每个用户的登录日期和时间。

【实例】

如果用户输入**who am I**或**who am i**,UNIX系统会显示本终端用户的信息(见图3.12)。

```
$ who am i
david      tty04      Nov 28  08:48
$_
```

图3.12 带参数am i的who命令

我是谁?

who命令的选项

表3.1列出**who**的一些命令选项。还有一些其他选项,但我们仅试一下这几个选项,不考虑细节。

表3.1 who命令的选项

选项	功能
-q	简要信息功能,仅显示各用户名和用户总数
-H	显示各列信息的标题
-b	显示系统的启动日期和时间
-s	只显示用户名、终端号和登录时间

【注意】

1. 在命令名和命令选项间必须用空格分开；
2. 在命令选项前有一个作为前缀的减号；
3. 在减号与命令选项字符间没有空格；
4. 命令选项字符的大小写必须正确。

命令选项使用实例

下面的例子说明如何利用命令选项来修改显示输出格式和详细程度。

【实例】

使用 **-H** 选项显示各列信息的标题(见图 3.13)。

```
$ who -H
NAME      LINE      TIME
david     tty04     Nov 28  08:27
daniel    tty10     Nov 28  08:30
$_
```

图 3.13 带选项 -H 的 who 命令

【实例】

使用 **-q** 选项显示简要用户列表和用户总数(见图 3.14)。

```
$ who -q
david daniel
# users = 2
$_
```

图 3.14 带选项 -q 的 who 命令

【实例】

使用 **-b** 选项显示系统的启动日期和时间(见图 3.15)。

```
$ who -b
system boot Nov 27 08:37
$_
```

图 3.15 带选项 -b 的 who 命令

3.2.5 显示日历: cal 命令

cal 命令显示指定年份的日历表。如果同时指定年和月,只显示一个月的日历表。年和月的指定都是命令参数。**cal** 命令的默认参数为当前月。

【实例】

显示2001年11月的日历表(见图3.16)。

```
$ cal 11 2001
November 2001
S   M   Tu  W   Th  F   S
    1   2   3
4   5   6   7   8   9   10
11  12  13  14  15  16  17
18  19  20  21  22  23  24
25  26  27  28  29  30
$_
```

图3.16 cal命令的输出

【说明】

1. 年份参数必须写全。如:可输入 **cal 1998**,而不能输入 **cal 98**。
2. 可使用数字表示月份(01~12),而不能使用月份的名字。
3. 不带参数的 **cal** 命令显示当前月份的日历表。
4. 只有年份参数而没有月份参数的 **cal** 命令显示指定年份的日历表。

3.3 UNIX 帮助信息

UNIX 系统并没有遗忘对初学者和健忘的用户的帮助。**learn** 和 **help** 命令是为使用 UNIX 提供辅助的程序。它们是为初学者设计的,易于使用。但这些命令在不同 UNIX 系统中有一些区别,并有可能没有安装在某些系统中。

3.3.1 使用 learn 命令

learn 命令是一个包括若干课程的计算机辅助教学程序。它显示一个课程列表,引导用户选择课程,显示课程内容等等。

要使用 **learn** 命令,可在命令行输入 **learn** 并按回车键。如果系统安装了 **learn** 程序,会显示图3.17所示的主菜单;否则,会显示下面的错误信息。

```
learn: not found (learn 命令没有找到)
```

3.3.2 使用 help 命令

help 命令比 **learn** 命令更流行,可在大多数 UNIX 系统上使用。**help** 命令提供一个多级菜单,通过一系列菜单选择和提问来引导用户学习最常用的 UNIX 命令。要使用 **help** 命令,可在命令行输入 **help** 并按回车键。系统会显示 **help** 命令的主菜单(见图3.18);如果系统中没有安装 **help** 程序,系统会显示如下错误信息:

```
help: not found (help 命令没有找到)
```

```

$ learn
These are the available courses
  files
  editor
  vi
  more files
  macros
  eqn
  c
If you want more information about the courses, or if you have never used 'learn'
before, press RETURN; otherwise type the name of the course you want followed by
RETURN

```

图 3.17 learn 命令的主菜单

```

$ help
help:UNIX System On-line Help
      Choices      description
s      starter:general information
l      locate:find a command with keyword
u      usage:information about command
g      glossary:definition of terms
r      redirect to a file or a command
q      Quit
Enter choice

```

图 3.18 help 命令的主菜单

3.3.3 获取更多的帮助信息:UNIX 用户手册

用户可从一个称为用户手册的文档中找到 UNIX 系统的详细说明。在采购时,用户可得到一本印刷的用户手册。磁盘上有该手册的电子版,称为在线手册。如果系统中安装了用户手册,可方便地在终端上显示。但 UNIX 用户手册十分简练而难读,它更像一个参考手册,而不像真正的用户手册。对于那些了解命令基本用途但忘记了准确使用方法的专业用户而言,UNIX 用户手册十分有用。

3.3.4 使用电子手册:man 命令

man 命令可显示在线系统文档的内容。通过输入 man 和命令名,可得到相应命令的帮助信息。当用户要学习一个新命令时,可使用 man 得到命令的详细使用说明。例如,为了得到 cal 命令的详细说明,可输入 man cal 并按回车键。随后系统显示与图 3.19 类似的内容。系统在查找命令帮助信息时,可能有时需要等待几秒钟。

```
cal(1)          User Environment Utilities cal(1)

NAME
  cal - print calendar
SYNOPSIS
  cal [ [ month ] year ]
DESCRIPTION

  cal prints a calendar for the specified year. If a month
  is also specified, a calendar just for that month is
  printed. If neither is specified, a calendar for the present
  month is printed. Year can be between 1 and 9999. The year is
  always considered to start in January even though this is
  historically naive. Beware that "cal 83" refers to the
  early Christian era, not the 20th century.
  The month is a number between 1 and 12.
  The calendar produced is that for England and the United States.
EXAMPLES

  An unusual calendar is printed for September 1752. That is
  the month 11 days were skipped to make up for lack of leap
  year adjustments. To see this calendar, type: cal 9 1752
```

图 3.19 用 man 命令得到的 cal 命令帮助

UNIX 系统的用户手册是按章节组织的,命令名后面圆括号内的数字是指相应内容在手册中的章节号。例如,CAL(1)是指第 1 章节,即用户命令部分。其他章节还有系统管理命令、游戏等。

3.4 更正键盘输入错误

即使最具天分的打字员也会出错,或在输入命令时改变主意。shell 程序只在用户按回车键后才开始解释命令行。在结束命令行(即按回车键)之前,用户都有机会更正输入错误或删除整个命令行。

如果输入一个敲错的命令名并按了回车键,系统会显示一个通用的错误信息(见图 3.20)。如果输入一个系统中没有安装的命令,也会显示相同的错误信息。

```
$ daye
daye: not found
$_
```

图 3.20 UNIX 错误信息

【实例】

这个 `date` 命令敲错了。

删除字符:用回退键([Back Space])删除字符。当按回退键时,光标左移并删除经过处的字符。另一种删除字符的方法是按[Ctrl-h]键,每次删除一个字符。例如,在输入 `calendar` 后按回退键 5 次,结果是光标左移 5 格,屏幕上只剩下 `cal`。这里按回车键,系统会执行 `cal` 命令。每次删除命令行上一个字符的按键称为删除键。

删除一行:在按回车键前,任何时候都可删除整个命令行。按[Ctrl-u]键可删除整个命令行,并使光标移动到一个新行。例如,假设用户输入 `passwd` 后不想修改口令,可按[Ctrl-u]键删除这个 `passwd` 命令。每次删除一行字符的按键称为行删除键。

【注意】

虽然[Ctrl-u]键包括 2 个按键,但它被当作单个字符对待。

中断程序运行:如果正在运行的程序还需要一段时间才能完成,而用户没有时间等待下去,他可能会选择中断运行。中断正在运行程序的按键称为中断键。在大多数系统中,[Del]或[Ctrl-c]就是中断键。中断键的功能是中断正在运行的程序,并显示 shell 命令提示符 `$`。

【说明】

可在你的系统上使用删除键和行删除键。如果它们不能正常工作,可向系统管理员询问。

3.5 使用 shell 和系统工具

UNIX shell 具有很强的功能和灵活性。shell 程序负责用户与 UNIX 系统间的交互。UNIX 命令的处理是由位于用户与操作系统其他部分之间的 shell 完成的。shell 是一个命令解释程序。每次用户输入一个命令并按回车键后,命令行被传到 shell 进行分析,然后开始执行用户请求。例如,输入 `date` 命令时,shell 首先查找 `date` 程序,然后执行该程序。从技术上讲,与用户进行交流的是 shell 程序,而不是 UNIX 系统。

shell 命令:一些 UNIX 命令是 shell 程序的一部分,称为内部命令或 shell 命令。内部命令由 shell 程序识别,并在 shell 内部执行。

系统工具:大多数 UNIX 命令是由 shell 查找和加载执行的可执行程序,称为系统工具或外部命令。

【注意】

本书中把 shell 命令和系统工具都称为命令。

3.5.1 shell 的种类

与其他程序一样,shell 也仅仅是一个程序,在 UNIX 系统中没有特权,这导致多种风格 shell 的存在。一个专业程序员甚至可编写自己的 shell 程序。SVR4 为用户提供了 3 种不同风格的 shell 程序: Bourne shell(`sh`)、Korn shell(`ksh`)和 C shell(`csh`)。

Bourne shell: Bourne shell 是大多数 UNIX 操作系统的标准 shell,也是系统中的默认命令解释程序。Bourne shell 的命令提示符是“`$`”符号。

Korn shell: Korn shell 是 Bourne shell 的一个超集,除具有 Bourne shell 的基本语法和特征外,还有一些其他特征。为 sh 开发的脚本通常不需要任何改动就可在 ksh 下执行。Korn shell 的命令提示符也是“\$”符号。

C shell: C shell 是由美国加州大学伯克利分校开发的,是 BSD UNIX 的一部分。C shell 的脚本语法不同于 sh 和 ksh,它使用一种 C 语言风格的语法。C shell 的命令提示符为“%”。

本书的 shell: Bourne shell 是许多 UNIX 系统的默认命令解释程序。就用户所关心的基本命令和特征而言,这 3 种 shell 十分相似。虽然也涉及 ksh 中方便用户的一些增强功能,但本书默认使用 Bourne shell。

【说明】

仅适用于 Korn shell 的命令会用“ksh”明确标注。

3.6 登录过程

当 UNIX 系统启动时,操作系统的常驻部分(内核)被装入内存;而操作系统程序的其他部分(系统工具)通常保存在系统磁盘上,在用户请求使用时装入并执行。当用户登录时,shell 程序被装入内存。了解登录过程中的系统响应顺序,可帮助更好地理解 UNIX 操作系统的内部操作。

当 UNIX 系统完成启动过程后,init 程序为系统中的每一个终端端口激活一个 getty 程序, getty 程序在相应终端上显示 login: 登录提示,并等待用户输入用户名(见图 3.21)。

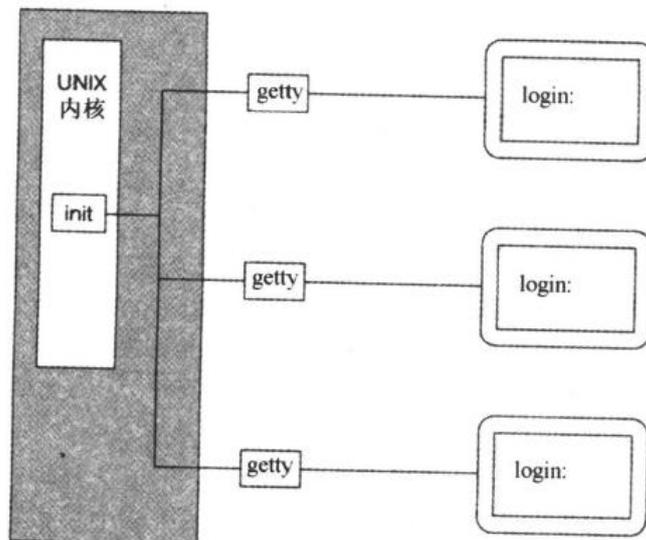


图 3.21 getty 程序显示的登录提示信息

用户输入用户名时,由 getty 程序读取用户输入并启动 login 程序,由 login 程序完成登录过程。getty 程序把用户在终端上输入的字符串传递给 login 程序,该字符串被视为用户名(也称为登录名)。随后,login 程序开始执行并在终端上显示 password:(输入口令)提示,等待用户输入口令(见图 3.22)。

用户输入口令后,login 程序会核实用户名和口令。随后 login 程序还要检查下一步要执行

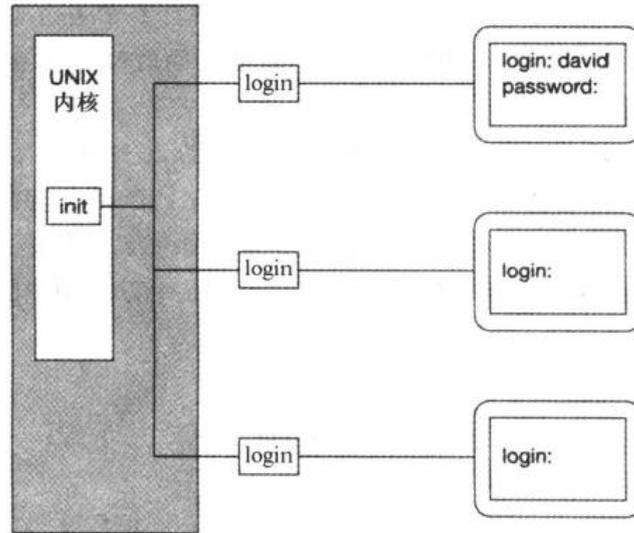


图 3.22 login 程序显示的输入口令提示

的程序名,通常这个程序就是 shell 程序。如果使用系统的默认 shell 程序(Bourne shell),它会显示 \$ 命令提示符。现在,shell 程序已完成准备工作,用户可输入命令(见图 3.23)。

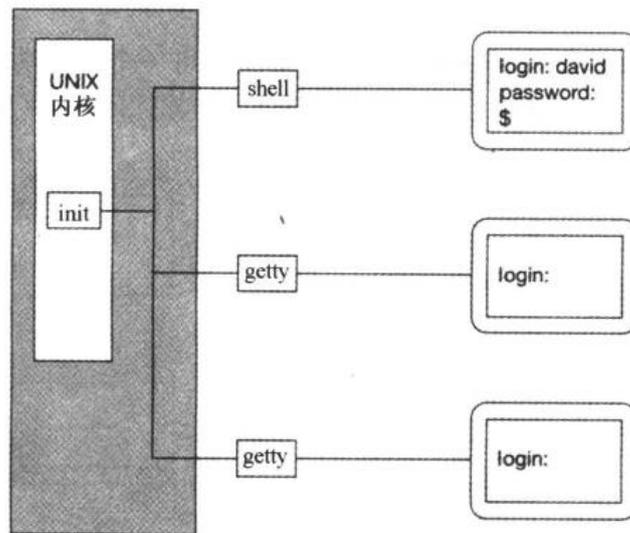


图 3.23 shell 程序显示的命令行提示

当用户退出系统时,shell 程序终止执行,UNIX 系统在终端上启动一个新的 getty 程序并等待新的用户登录。只要系统启动运行,这个登录、退出循环过程就会一直进行下去(见图 3.24)。

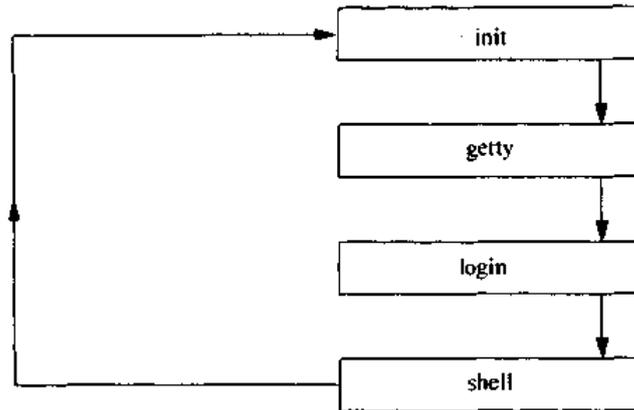


图 3.24 登录、退出循环过程

命令小结

本章讨论了下面的 UNIX 命令。为了便于读者掌握这些命令,图 3.25 再一次列出了 UNIX 命令的命令行格式。

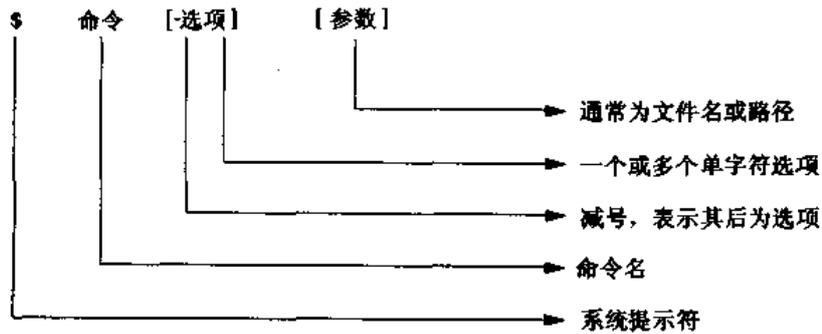


图 3.25 命令行格式

date
显示当前时间和日期。

cal
显示指定年份 12 个月的日历表或指定月份的日历表。

who
显示当前系统所有用户的登录名、终端号和登录时间。

选 项	功 能
-q	简要信息功能, 仅显示各用户名和用户总数
-H	显示各列信息的标题
-b	显示系统的启动日期和时间
-s	只显示用户名、终端号和登录时间

learn

一个包括若干课程的计算机辅助教学程序。它显示一个课程列表,引导用户选择和学习相应课程。

help

通过显示一系列菜单选择和提问,引导用户学习大多数常用的 UNIX 命令。

man

该命令可显示在线系统文档的内容。

passwd

该命令可修改用户登录口令。

习题

1. 试说明用户登录过程。
2. 试说明用户退出过程。
3. 为什么要给用户分配登录名?
4. 试说明在 UNIX 系统启动时的内部操作序列。
5. 什么是 shell 程序? 它在 UNIX 系统中起什么作用?

上机练习

在第一次登录前,请用几分钟时间确保您已了解下列系统信息:

找到您的用户名(登录名);

如果系统要求登录口令,请向管理员询问自己的登录口令;

在键盘上找到下列功能键:

- 删除键
- 行删除键
- 中断键

现在您已完成准备工作,可打开终端并等待 **login:** 登录提示。

1. 使用自己的用户名和口令来登录系统。请注意屏幕上出现的信息。
2. 查看系统的命令提示符,确定系统使用的 shell 程序类型。
3. 用 **who** 命令来查看当前登录在系统中的用户列表。
4. 利用 **who** 命令的选项来查看当前系统中的用户总数和系统启动时间。
5. 查找系统中的帮助系统工具。
6. 用 **date** 命令查看当前日期和时间。
7. 用 **cal** 命令查看自己的生日是星期几。
8. 查看 2001 年的日历表。

9. 用 `passwd` 命令修改口令。
10. 试图用一个不满足口令组成要求的字符串作为新口令,以便了解 UNIX 系统显示的错误信息的类型。
11. 在成功修改口令后,退出系统并用新口令重新登录。
12. 试用删除键更正输入错误。
13. 试用行删除键终止命令行输入。
14. 退出系统,整个练习完成。

第 4 章 vi 编辑器入门

对于 vi(发音为 vee-eye)编辑器,总共用了两章来介绍,本章为入门部分。第 6 章是高级部分。开始先大概解释一下什么是编辑器,然后讨论编辑器的类型及各自的应用。在介绍完 UNIX 支持的编辑器后,开始介绍 vi 编辑器。其余部分列出了用 vi 进行简单编辑所需要的基本命令。vi 编辑器的基本概念和操作包括以下几点:

- vi 编辑器的工作模式
- 存储缓冲区
- 打开一个文件进行编辑
- 保存文件
- 退出 vi

4.1 什么是编辑器

编辑一个文本文件是经常使用到的计算机操作。事实上,用户想做的许多事情迟早都需要某种文件编辑。

文本编辑器会方便新文件的创建和旧文件的修改。这些文件也许会包含笔记、备忘录、程序源代码等等。文本编辑器是简化的字处理器,没有字处理器所具有的黑体、居中、下划线等等这些印刷上的特征。每一种操作系统软件至少支持一种文本编辑器。通常有两种方式的编辑器:

- 行编辑器
- 全屏编辑器

行编辑器:在行编辑器中,每次几乎所有的修改只能在一行之中或组行之间进行。为了进行修改,需要先给出行号,然后再修改。行编辑器一般难以使用,因为不能看见所编辑任务的范围和上下文。行编辑器比较有利于全局的操作,如搜索、替换或者复制文件中的大块文本。

全屏编辑器:全屏编辑器能显示用户正在编辑的那一屏文本,而且用户能够在屏中任意移动光标修改,用户做出的修改马上就能在屏幕上显示出来。在任何时刻可以很容易地浏览屏上其余的文本。全屏编辑器的界面要比行编辑器的界面友好,比较适用于日常的编辑工作。

4.1.1 UNIX 支持的编辑器

UNIX 操作系统支持许多种类的行编辑器和全屏编辑器,用户可以很方便而有效地创建或者修改文件。列举几个,如:行编辑器 emacs 和 ex,全屏编辑器 vi。

ed 是早期 UNIX 上支持的老版本的行编辑器。目前 ex 是许多 UNIX 操作系统都支持的行编辑器。最初 ex 提供了一种显示特性,可以显示全屏的文本,而且用户可以在全屏上工作而不仅仅限于一行。可以使用 vi 命令在 ex 使用全屏特性。后来,因为这种全屏模式得到了广泛

使用,ex 的开发者就单独开发了一个全屏编辑器,这就是 vi。这样用户就可以直接使用 vi 而无需先打开 ex 编辑器了。

文本格式器:UNIX 下的编辑器都不支持文本格式。不支持行居中、留出空白等这些字处理器具有的功能。为了给出文本格式,UNIX 中提供了一些工具如 nroff 和 troff。

文本格式器一般用来准备档案文献。输入是由 vi 等编辑器产生的文本文件,格式器(formatter)输出的文件都已加了页码标注,可以直接在屏幕上显示或者直接输出到打印机。表 4.1 列出了 UNIX 下支持的编辑器及种类。

表 4.1 UNIX 支持的编辑器

编 辑 器	种 类
ed	最初的行编辑器
ex	在 ed 上扩展更为复杂的编辑器
vi	可视化的全屏编辑器
emacs	公共域的全屏编辑器

4.2 vi 编辑器

vi 是大多数 UNIX 系统都支持的全屏文本编辑器。它具有字处理器的灵活性和简单易用的特性。如前所述,vi 从行编辑器 ex 发展而来,因此就有可能在 vi 中使用 ex 的命令。在使用 vi 时,用户对文件的修改立即会在屏幕上显示出来,屏幕上光标的位置也对应了这时用户在文件中的位置。vi 提供了 100 多种命令,有强大的功能。学会这么多的命令确实具有一定的难度。不过不要惊慌,简单的编辑工作一般只用到少数几个命令。

vi 有两个版本,view 编辑器和 vedit 编辑器,为特殊的任务或者用户提供剪裁功能。这些版本与 vi 的功能类似,区别在于有一些标志预先做了设置。不是所有的系统都提供这两种版本。在第 6 章的末尾,介绍如何根据用户的需求定制 vi 环境变量。

view 编辑器:view 编辑器对 vi 设了只读标志。view 编辑器对那些只想看看文件的内容,而对文件不做任何修改的用户来说就很有用,这时就可以使用 view 编辑器。view 编辑器阻止用户不经意间对文件所做的修改,这样用户的文件会完好地保存着。

vedit 编辑器:vedit 编辑器对 vi 做了几个标志设置。vedit 编辑器是面向初学者的。这些标志的设置简化了 vi 的使用。

4.2.1 vi 的工作模式

vi 有两种基本工作模式:命令模式和文本输入模式。一些命令适用于命令工作模式,而另一些命令只适用于文本输入模式。在 vi 使用中,用户可以在两种工作模式间切换。

命令模式:vi 初始启动时先进入命令模式。在命令模式下,键的输入不会在屏幕上显示,只会被解释执行。在命令模式下,用户可以利用一些预先定义的按键来移动光标、查找某个字、删除几行、粘贴文字等。

一些命令以冒号(:)、斜杠(/)或者问号(?)开头。用户输入的命令均显示在编辑器屏幕的最后一行。在输入命令的行尾,按回车键,vi 执行命令。

文本输入模式:在文本输入模式下,键盘就成了用户的打字机。vi 会按序显示用户的输入。按键不是被解释为命令执行,而只是作为文本写入到用户的文件中。

状态行:屏幕底部一行,通常是 24 行,被 vi 编辑器用来反馈编辑操作结果。错误消息或者提供信息的信息会在状态行中显示出来。vi 还会在 24 行显示那些以冒号(:)或问号(?)开头的命令。

4.3 基本 vi 编辑器命令

简单的编辑任务一般会涉及到以下操作:

- 创建一个新文件或者修改一个已存在的文件(打开文件)
- 进入文本输入状态
- 删除文本
- 搜索文本
- 修改文本
- 保存文件,退出编辑(关闭文件)

下面通过编辑几个任务,展示 vi 编辑器的工作模式,举例说明在文本输入模式和命令模式下如何使用一些必需的命令。

【说明】

仅仅依靠这本书来学习 vi 不是一个好方法。极力推荐读者在读完本章后,在自己的系统上做一些习题和上机练习。

假设:为了避免重复,下面的假设适用于本章所有的例子。

- 当前行指光标所在行
- 双下划线(=)指示光标所在行的位置
- myfirst 文件用来展示各种按键和命令的操作。如果必要,例子会在屏幕上显示。如果有多屏时,最上面一屏显示文本的当前状态,下面一屏显示特定的按键或者命令执行完后文本的变化
- 为了节约版面,只显示与屏幕相关的部分
- 例子之间不具有连续性。也就是说,编辑的变化不是由一个例子到另一个例子的延续

4.3.1 进入 vi 编辑器

和使用其他软件一样,使用 vi 的第一步是学会如何开始和结束程序。本部分介绍如何启动 vi 编辑器创建一个小的文本文件,然后如何保存这个文件。

启动 vi

为了启动 vi,输入 vi,按空格,再输入新文件名(本例中用 myfirst 作为文件名),然后再按回车键。

如果 myfirst 文件已存在,vi 就会在屏幕上显示这个文件的第一页(前 23 行)。如果 myfirst 是

一个新文件,vi 会清除屏幕,显示 vi 的灰屏,光标会处在屏幕的左上角。屏幕每行的第一列显示 ~。vi 编辑器这时处于命令工作模式,准备接受用户的命令。图 4.1 显示了 vi 的灰屏。

```

~
~
~
~
"myfirst". [new file] 0 lines, 0 characters

```

图 4.1 vi 编辑器的灰屏

【说明】

1. 状态行说明文件名,并且这是一个新文件。
2. 为了节约版面,本书中的屏幕没有全屏显示(24 行)。

为了输入文本,必须把 vi 从命令模式切换到文本输入模式。确保大写字母键[Caps Lock]处于关闭状态,然后按下 i(Insert)键,vi 不会显示字母 i,而是进入到文本输入模式。现在开始输入图 4.2 中显示的文本。vi 编辑器会在屏幕上显示输入的文本。用回退键[Back Space],或者[Ctrl-h]键可以删除文字。在行尾按回车键就会另起一行。不要过分关心输入文本的词法和语法,因为这里的主要目的是如何创建文件,如何保存文件。

```

The vi history
vi is an interactive text editor that is supported by most of the
UNIX operating systems.
~
~
~
~
~
~
"myfirst" [new file]

```

图 4.2 vi 编辑器的显示屏

【注意】

确保大写字母键[Caps Lock]处于关闭状态,是因为在命令模式下,大小写字母具有不同的含义。

【说明】

1. vi 编辑器处于命令模式下时,许多命令是在用户按下命令键时就执行了。这样在命令行的末尾就没有必要再按回车键。
2. vi 编辑器没有任何反馈信息提示用户当前所处的工作模式。这个问题在第 6 章再进行深入讨论。
3. 如果用户的文件没有填满全屏,vi 编辑器就会在其余行的第一列填上(~)。
4. 如果用户的屏幕和上图显示的不一样,可能是用户的终端类型没有设置或设置错误。这个问题在第 8 章再深入讨论。

退出 vi

为了保存用 vi 创建或编辑的文件,用户必须使 vi 编辑器工作于命令模式。可以按[Esc]键。如果用户终端的声音是激活的,就会听到蜂鸣声,表示 vi 编辑器这时处在命令工作模式。保存文件和退出 vi 命令都是以冒号(:)开头,输入冒号(:),光标就会处在屏幕的最后一行。然后输入 wq(Write and Quit),再按回车键。vi 编辑器就会保存 myfirst 文件,并把控制权返回给 shell。shell 显示 \$ 提示符,\$ 符表示现在用户已退出 vi 编辑器回到了 shell 中,系统正在等待用户的下一个命令。图 4.3 显示输入 wq 后 vi 编辑器显示的内容。

```
The vi history
The vi editor is an interactive text editor that is supported by
most of the UNIX operating systems.
~
~
~
~
:wq
"myfirst" [new file]
$ _
```

图 4.3 输入 wq 后 vi 编辑器屏幕

【说明】

vi 编辑器的反馈信息显示在屏幕的最后一行。反馈信息显示文件名、共有多少行、多少字符。

4.3.2 文本输入模式

用户只有处于文本输入模式才能输入文字到文件中。然而,不同的按键进入文本输入模式的方式都有些不同。在哪儿插入用户的文本取决于用户光标所在的位置和进入文本输入模式时的按键。

表 4.2 总结了从 vi 的命令模式切换到文本输入模式状态下的命令键。还是以上节创建的文件 myfirst 为例。假定用户已打开 myfirst 文件,现在想输入文字 999 到文件中(999 没有什么特别的含义,这儿仅仅为了演示用户插入文本的位置与用户当前光标的位置是相关的。如果愿意,可以插入一些其他的文字。)而且假定当前光标在单词 most 的字母 m 上,由双划线标注。由于选择进入文本输入模式的命令键不同,文字 999 将被插入到文件的不同位置。

表 4.2 切换到文本输入模式的命令键

命令键	功能
i	在光标左侧输入正文
I	在光标所在行的开头输入正文
a	在光标右侧输入正文
A	在光标所在行的末尾输入正文
o	在光标所在行的下一行增添新行,并且光标位于新行的开头
O	在光标所在行的上一行增添新行,并且光标位于新行的开头

【说明】

1. 光标的位置用双下划线标注。
2. 当前行是指光标所在行。

插入正文:用 i 或者 I

键 **i** 或者 **I** 都可以切换到文本输入模式。区别只在于用户想在哪儿插入正文。输入 **i** 意味着在光标的左侧输入正文,而 **I** 表示在光标所在行的开头输入正文。下面来试验一下 **i** 的用法:

- 先按 **[Esc]** 键。确保 vi 处于命令模式。
- 按 **i** 键。将 vi 切换到文本输入模式。
- 按字符 **9** 三次。这时会发现 999 出现在 **m** 的前面。

这时光标仍停留在 **m** 上,vi 还处于文本输入模式,直到用户按了 **[Esc]** 键回到命令状态。

```
The vi history
The vi editor is an interactive text editor that is supported
by most of the UNIX operating systems.
```

```
The vi history
The vi editor is an interactive text editor that is supported
by 999most of the UNIX operating systems.
```

【实例】

下面来试验 **I** 的用法:

- 先按 **[Esc]** 键。确保 vi 处于命令模式。
- 按 **I** 键。将 vi 切换到文本输入模式,并且光标处于当前行的开头。
- 按字符 **9** 三次。999 出现在行首,并且当前光标移到了字符 **T** 上。

```
The vi history
The vi editor is an interactive text editor that is supported
by most of the UNIX operating systems.
```

```
The vi history
999The vi editor is an interactive text editor that is supported
by most of the UNIX operating systems.
```

这时光标定位在 **T** 上,vi 还处于文本输入模式,直到用户按了 **[Esc]** 键回到命令状态。

添加文本:用 a 或者 A**【实例】**

键 **a** 或者 **A** 都可以切换到文本输入模式,区别只在于用户想在哪儿插入正文。输入 **a** 意味着在光标的右侧输入正文,而 **A** 表示在光标所在行的末尾输入正文。下面来试验一下 **a** 的用法:

- 先按 [Esc] 键。确保 vi 处于命令模式。
- 按 **a** 键。将 vi 切换到文本输入模式, 这时光标移到了当前位置的右边, 也就是字符 o 上。
- 按字符 9 三次。这时会发现 999 出现在 m 的后面。

```
The vi history
The vi editor is an interactive text editor that is supported
by most of the UNIX operating systems.
```

```
The vi history
The vi editor is an interactive text editor that is supported
by m999ost of the UNIX operating systems.
```

这时光标仍停留在 o 上, vi 还处于文本输入模式, 直到用户按了 [Esc] 键回到命令状态。

【实例】

下面来试验 A 的用法:

- 先按 [Esc] 键。确保 vi 处于命令模式。
- 按 **A** 键。将 vi 切换到文本输入模式, 并且光标处于当前行的末尾。
- 按字符 9 三次。999 出现在当前行的最后一个字符. 的后面。

```
The vi history
The vi editor is an interactive text editor that is supported
by most of the UNIX operating systems.
```

```
The vi history
The vi editor is an interactive text editor that is supported
by most of the UNIX operating systems.999.
```

这时光标定位在行尾, vi 还处于文本输入模式, 直到用户按了 [Esc] 键回到命令状态。

增添新行: 用 o 或者 O

【实例】

键 **o** 或者 **O** 都可以切换到文本输入模式。输入 **o** 意味着在光标所在行的下一行增添新行, 而 **O** 表示在光标所在行的上一行增添新行。下面来试验一下 **o** 的用法:

- 按 **o** 键。将 vi 切换到文本输入模式。在光标所在行的下一行增添新行, 而这时光标移到新行的开头。
- 按字符 9 三次。这时会发现 999 出现在新行上。

```
The vi history
The vi editor is an interactive text editor that is supported
by most of the UNIX operating systems.
```

```
The vi history
The vi editor is an interactive text editor that is supported
999o
by most of the UNIX operating system.
```

这时光标仍停留在新行的末尾,vi 还处于文本输入模式,直到用户按了[Esc]键回到命令状态。

【实例】

下面来试验 O 的用法:

- 按 O 键。将 vi 切换到文本输入模式,在光标所在行的上一行增添新行,而这时光标移到了新行的开头。
- 按字符 9 三次。这时会发现 999 出现在新行上。

```
The vi history
The vi editor is an interactive text editor that is supported
by most of the UNIX operating systems.
```

```
The vi history
999 _
The vi editor is an interactive text editor that is supported
by most of the UNIX operating systems.
```

这时光标仍停留在新行的末尾,vi 还处于文本输入模式,直到用户按了[Esc]键回到命令状态。

【注意】

在文本输入模式下,尽量避免光标键的使用。因为在一些系统里,光标键会被解释为标准的 ASCII 字符。如果在文本输入模式下使用了光标键,这些光标键的 ASCII 码就会作为正文输入到文件中。

使用空格键、[Tab]键、回退键和回车键

vi 编辑器在文本输入模式下,在屏幕上显示的是用户输入的文字。但是并不是键盘上的所有键都能在屏幕上显示。举例来说,用户为了将光标移到下一行,按了回车键。对于用户来说,并不希望在屏幕上看到一个[Return]的符号。就我们现在所知,根据 vi 的工作模式不同,不同键的含义也不同。下面将讨论在文本输入模式下,空格键[Spacebar]、[Tab]键、回退键[Back Space]和回车键[Return]的使用。

空格键和[Tab]:空格键表示在当前光标的位置前插入 1 个空格。[Tab]键通常用来插入 8 个空格。[Tab]键能插入的空格数是可变的,可以设置。(详细设置见第 6 章)

回退键:回退键把光标回退一个字符的位置。

回车键:回车键会增添一个新行。根据光标在当前行的位置的不同,会在当前行的上一行或下一行增添新行。

- 如果光标的右面已经没有文字,处于当前行的末尾,这时回车键会在当前行的下一行增添一新行。
- 如果光标的左面没有文字,处在当前行的头部,这时回车键会在当前行的上一行增添一新行。
- 如果光标的左面和右面都有文字,处在当前行的中间,这时回车键会把当前行分成两

行,处于光标右面的文字就移到新行上。

【注意】

上面的解释只适用于 vi 处于文本输入模式下。

4.3.3 命令模式

启动 vi 编辑器,默认进入命令工作模式。可以按[Esc]键从文本输入模式切换到命令模式。为了确认 vi 确实处于命令工作模式,可以按[Esc]键两次。如果 vi 已处在命令模式,用户还按了[Esc]键,那什么也不会发生,vi 还保持在命令模式。

光标移动键

要对正文内容进行删除、修改或者插入,首先必须把光标移动到指定位置。在命令模式下,用户可以用的最简单的方法是用箭头键来移动光标。在有些终端上,箭头键不是按它字面的含义工作的,或者有时候箭头键就不工作。在这些情况下,用户可以用 h、j、k 和 l 键左、下、上、右地移动光标。按光标移动键可以一个空格、一个字或者一行地移动光标。

表 4.3 总结了光标移动键及其应用。每个键的应用会在下面的例子以及后面的练习中说明。图 4.4 显示这些光标键的移动效果。

表 4.3 vi 的光标移动键

键	功能
h 或左箭头	把光标左移一个空格
j 或下箭头	把光标下移一行
k 或上箭头	把光标上移一行
l 或右箭头	把光标右移一个空格
\$	把光标移到当前行的末尾
w	右移光标,到下一个字的开头
b	左移光标,到前一个字的开头
e	右移光标,到一个字的末尾
0(数字)	左移光标,到本行的开头
回车键	移动光标到下一行的开头
空格键	把光标右移一个空格
回退键	把光标左移一个空格

j 和 k: j 或下箭头把光标的位置下移一行。k 或上箭头把光标的位置上移一行。如果当前行已处于文件的最后一行,这时用户还按了 j 或下箭头键,用户就会听到蜂鸣声,光标的位置保持不动。同样的情形会发生在光标处于文件的第一行,按了 k 或上箭头键的情况。这些键不会使正文回绕。

h 和 l: 每次用户只要按下 h 或者左箭头键,光标的位置就会左移一个字符,直到光标到达行首为止。然后就会发出蜂鸣声,表示光标已不能再左移了。l(小写的 L)或右箭头以同样的方式右移光标。这些键也不会使正文回绕。

\$ 和 0: \$ 键会使光标的位置移到当前行的末尾。\$ 键在一行中只能按一次,也就是说,当

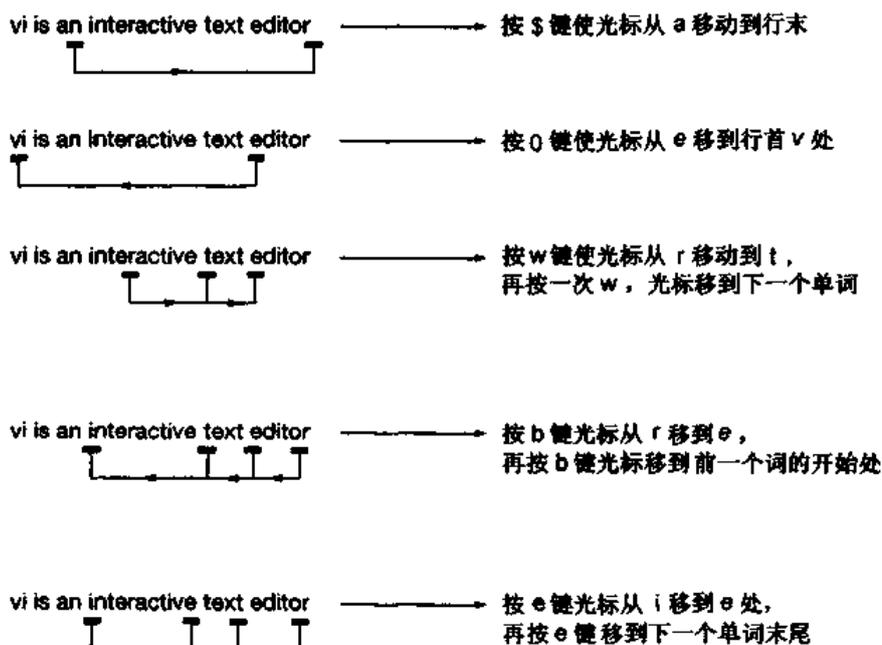


图 4.4 光标的移动

光标已处在行的末尾时,再按 **\$** 键,什么也不会改变。同样,**0** 会使光标的位置移到当前行的开头。

w、**b** 和 **e**:每次用户按下 **w** 后,光标右移到下一个字的开头。**b** 会使光标左移到前一个字的开头。**e** 使光标右移到一个字的末尾。这些键会在文本中回绕,如果必要,光标会移到下一行。

回车键:用户每按一次回车键,光标就会下移一行,直到到达文件末尾。

文本修改

vi 处于命令模式时,可以替换(覆盖)字符,或者删除字符、一行或者数行。也可以利用 **undo** 命令修改错误。正如 **undo** 的含义,undo 命令放弃用户最近的一次操作。这些文本修改命令仅适用于 vi 的命令模式,而且大部分命令不会改变 vi 的工作模式。表 4.4 总结了删除文本的键及其应用。

表 4.4 修改文本的 vi 键

键	功 能
x	从指定位置开始删除字符
dd	从指定位置删除行
u	放弃最近的修改
U	放弃对当前行做的所有修改
r	替换当前光标所在的字符
R	从当前光标的位置开始替换字符,并且使 vi 进入文本输入模式
.	重复上次的修改

删除字符: x 键的使用

【实例】

假定用户已将 `myfirst` 文件打开在屏幕上,准备对该文件做些修改,当前光标定位在单词 `most` 的字母 `m` 上。通过 `x` 键,就可以从当前位置开始删除字符。

```
The vi history
The vi editor is an interactive text editor that is supported
by most of the UNIX operating systems.
```

```
The vi history
The vi editor is an interactive text editor that is supported
by ost of the UNIX operating systems.
```

- 按 `x` 键。vi 删除字母 `m`,光标移到右面一个字符 `o` 上。vi 编辑器仍处于命令工作模式。
- 按 `x` 键三次。vi 分别删除字母 `o`、`s` 和 `t`。

```
The vi history
The vi editor is an interactive text editor that is supported
by __of the UNIX operating systems.
```

vi 编辑器仍处于命令模式,这时光标就移到字符 `o` 之前的空格上。如果想一个命令删除多个字符,可以用 `nx` 命令。`n` 是一个整数,表示用户想删除的字符数。例如,命令 `5x` 就从当前光标处开始删除 5 个字符。

【说明】

vi 的其他命令也可以重复使用。例如,`dd` 删除一行,那么 `3dd` 就删除 3 行。

删除和恢复:dd 和 u 的使用

【实例】

按 `d` 两次,vi 就从当前行开始删除一行。

```
The vi history
The vi editor is an interactive text editor that is supported
by most of the UNIX operating systems.
```

- 按 `d` 两次。vi 编辑器删除当前行,无论当前光标处在行的哪个位置。

```
The vi history
The vi editor is an interactive text editor that is supported
```

- vi 编辑器仍处于命令模式,这时光标移到了上一行的开头。按 `u` 以后,vi 编辑器会放弃上一次的删除。

```
The vi history
The vi editor is an interactive text editor that is supported
by most of the UNIX operating systems.
```

vi 编辑器仍处于命令模式,这时光标移到了行的开头。如果用户想一次删除多行,可以使

用 `ndd` 命令。`n` 是一个整数,表示一次想删除的行数。例如,命令 `5dd` 就从当前行开始删除 5 行。

文本替换:r、R 和 U 的使用

通过 `r` 和 `R`,用户可以从当前位置开始替换一个或一组字符。然而,`R` 会使 vi 编辑器处于文本输入模式,需要 `[Esc]` 返回到命令模式。

【实例】

下面来学习掌握 `r` 的使用:

- 按 `r` 替换光标所在的字符。
- 按 `9`。vi 编辑器会用 `9` 替换原来的 `m`。这时 vi 编辑器仍处于命令模式,当前光标的位置不变。

```
The vi history
The vi editor is an interactive text editor that is supported
by most of the UNIX operating systems.
```

```
The vi history
The vi editor is an interactive text editor that is supported
by 9ost of the UNIX operating systems.
```

【实例】

以下学习掌握 `R` 和 `U` 的使用:

- 按 `R` 从当前光标开始替换字符。vi 编辑器进入文本输入模式。
- 按 `9` 三次。vi 编辑器会从当前光标开始用 `999` 替换 `ost`。这时 vi 仍处于文本输入模式。

```
The vi history
The vi editor is an interactive text editor that is supported
by 9999 of the UNIX operating systems.
```

- 按 `[Esc]` 可以返回到命令模式。
- 按 `U` 可以恢复对当前行所做的修改。

vi 编辑器这时又把当前行恢复到原来的状态。

```
The vi history
The vi editor is an interactive text editor that is supported
by most of the UNIX operating systems.
```

搜索字符串:/和? 的使用

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。字符 `/` 和 `?` 分别提供了在文件中前向和后向的搜索功能。如果用户正在编辑一个很大的文件,可以通过这些操作把光标定位在指定的位置。举例来说,如果用户想查找 UNIX 在文件中的位置,可以先按 `[Esc]` (确保 vi 处于命令工作模式)然后输入 `/UNIX`,再按回车键。

当用户键入 `/` 后,vi 在屏幕的底部显示 `/`,等待键入要查找的字符串。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。继续键入 n,可以继续查找指定字符串的位置,直到文件的末尾。然后光标回到文件的开头,继续查找。用字符? 取代/,可以实现反向搜索(朝文件开头方向)。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。

重复以前的修改操作:.的使用

(句点)用来在命令模式下重复用户最近所做的文本修改操作。如果用户想在一个文件中做很多重复性的改动时,这个功能就非常有用。

【实例】

下面来实验.的使用。

- 键入 dd 删除当前行。

```
The vi history
The vi editor is an interactive text editor that is supported
by most of the UNIX operating systems.
```

```
The vi history
The vi editor is an interactive text editor that is supported
```

- 用光标键将光标移到另外一行。

```
The vi history
~
~
~
```

- 键入.(句点)。vi 编辑器重复上次的文本修改操作,就删除了当前光标所在行。光标移动到前一行,vi 仍处于命令模式。

退出 vi

进入 vi 的方法只有一种,但是却有多种退出 vi 的方法。用户可以根据自己的操作意图选择不同的退出方式。表 4.5 总结了 vi 编辑器的退出方法。

表 4.5 vi 编辑器的退出命令

键	功能
wq	保存文件,退出 vi
w	保存文件,但不退出 vi
q	退出编辑器
q!	退出编辑器,同时放弃所做的修改
ZZ	保存文件,退出 vi

:wq 命令:多数情况下,用户在编辑结束时,用:wq 命令保存文件内容,然后退出 vi。UNIX 出现 \$ 提示,表明 shell 已返回。**ZZ 命令**和:wq 命令功能一样。

:q 命令:用户如果只是读文件的内容,而不对文件的内容作任何修改,可以用:q 命令退出 vi。如果用户对文件的内容作了修改,但用:q 退出 vi,那么 vi 就会在屏幕的底行提示下面一句简洁典型的 UNIX 信息,vi 编辑器还保留在屏幕上:

```
No write since last change (:q! overrides).
```

:q! 命令:如果用户对文件的内容做了修改,然后决定放弃所做的修改,就可以用:q! 命令退出 vi 编辑器。在这种情况下,源文件保持不变。

:w 命令:用户在做一个很长的编辑任务时,使用:w 命令,可以定期保存文件内容,防止意外丢失所做的工作。如果不想覆盖源文件,可以用:w 命令输入一个新的文件名,然后将文件的内容写入新文件中。

ZZ 命令:ZZ 命令快速保存文件内容,然后退出 vi。

【说明】

ZZ 命令的前面不用:,而且也不需要键入[Return]完成命令。只需键入 ZZ,整个操作就完成了。

图 4.5 表示了 vi 编辑器操作模式和键的次序,或 vi 编辑在不同模式间转换的命令。

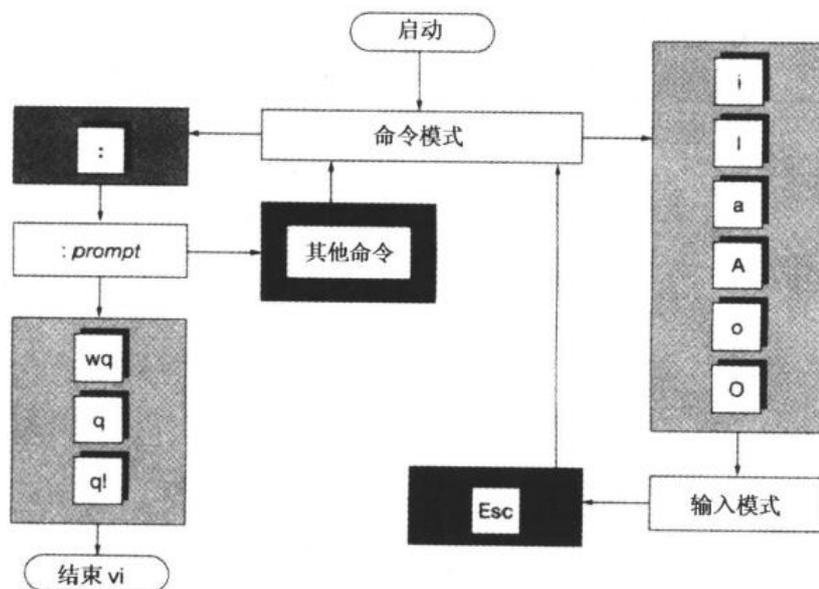


图 4.5 vi 编辑器的工作模式

【注意】

1. 大部分命令都以: 开头;
2. 键入:, 光标就会定位在屏幕上的最后一行。vi 编辑器在最后一行上显示用户输入的命令;
3. 记住要键入[Return]表示命令输入完毕。

4.4 存储缓冲区

vi 编辑器为用户生成或者修改一个文件创建一个临时工作区。如果想生成一个新文件, vi 为新文件开辟一个临时工作区。如果指定的文件已存在, vi 把源文件复制到临时工作区。用户对文件所做的修改只作用于工作区中的副本, 而不是源文件。这种临时的工作区就称为一个缓冲区或者工作缓冲区。vi 编辑器在一个编辑任务中, 可以使用几个不同的缓冲区来管理用户的文件。如果用户想保存所做的修改, 就需要将缓冲区中的副本替换源文件。对源文件的修改不能自动完成, 需要用户执行一个写入的操作。

如果用户打开一个文件编辑, vi 把文件复制到临时的工作区, 在屏幕上显示前 23 行。缓冲区中 23 行组成的那个窗口, 就是用户在屏幕上见到的(见图 4.6(A))。上下移动窗口, vi 就可以显示缓冲区中的其他内容。如果用户用光标或者命令将窗口下移 10 行, 那么屏幕上的前 9 行就滚动过去(从屏幕上消失), 然后用户在屏幕上看到的就是 10~32 行的文本(见图 4.6(B))。使用光标或者命令键可以上下移动窗口到文件的任何位置。

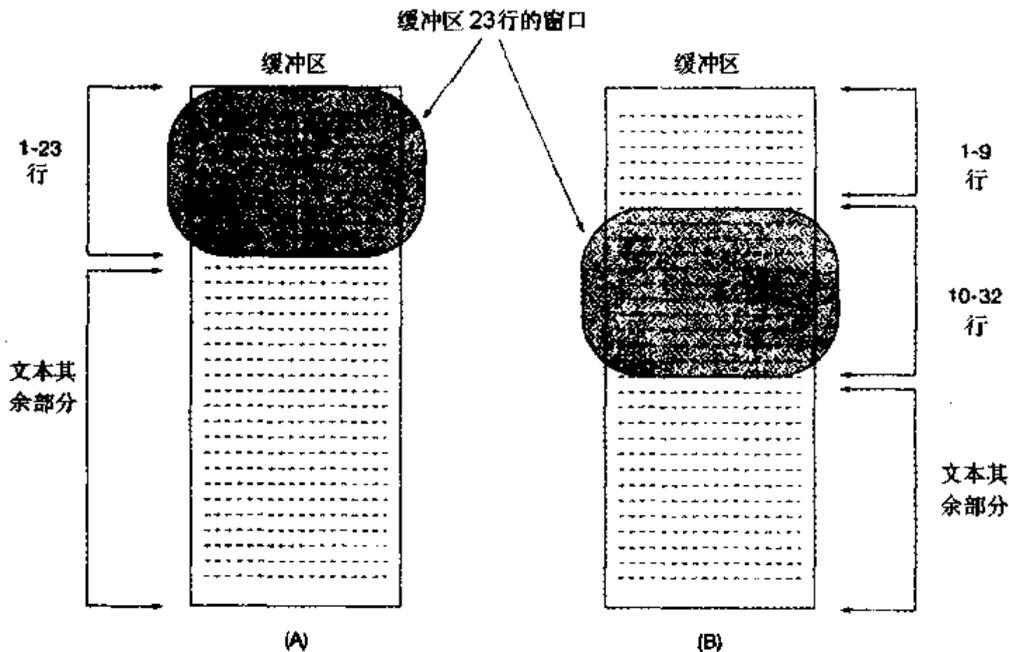


图 4.6 vi 编辑器临时缓冲区

【注意】

请务必记住在退出 vi 编辑器时, 要保存修改的内容, 否则用户所做的修改就被丢弃了。

命令小结

以下关于 vi 编辑器的命令和操作在本章已经讨论过。为了加深读者印象, 图 4.7 对 vi 的工作模式做了总结。

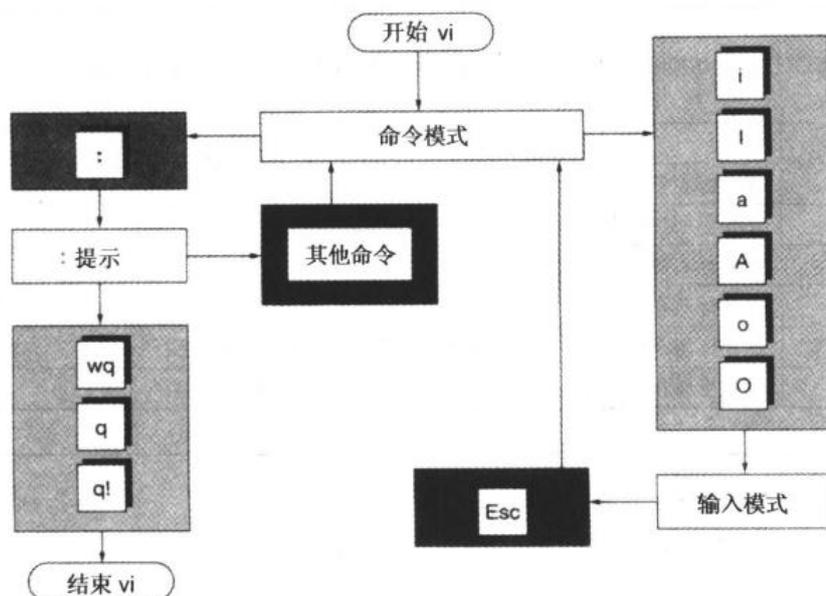


图 4.7 vi 编辑器的工作模式

退出 vi 的命令

除了 ZZ 命令外,其他的命令都以:开头,以[Return]结尾。

键	功 能
wq	保存文件,退出 vi
w	保存文件,但不退出 vi
q	退出编辑器
q!	退出编辑器,同时放弃所作的修改
ZZ	保存文件,退出 vi

键	功 能
h 或左箭头	把光标左移一个空格
j 或下箭头	把光标下移一行
k 或上箭头	把光标上移一行
l 或右箭头	把光标右移一个空格
\$	把光标移到当前行的末尾
w	右移光标,到下一个字的开头
b	左移光标,到前一个字的开头
e	右移光标,到一个字的末尾
0	数字 0,左移光标,到本行的开头
回车键	移动光标到下一行的开头
空格键	把光标右移一个空格
回退键	把光标左移一个空格

模式切换键

这些键把 vi 从命令工作模式切换到文本输入模式。不同的键使 vi 进入不同的文本输入模式, [Esc] 键使 vi 返回到命令工作模式。

命令键	功 能
I	在光标左侧输入正文
l	在光标所在行的开头输入正文
a	在光标右侧输入正文
A	在光标所在行的末尾输入正文
o	在光标所在行的下一行增添新行, 并且光标位于新行的开头
O	在光标所在行的上一行增添新行, 并且光标位于新行的开头

修改文本的键

这些键仅仅适用于命令模式。

键	功 能
x	从指定位置开始删除字符
dd	从指定位置删除行
u	放弃最近的修改
U	放弃对当前行做的所有修改
r	替换当前光标所在的字符
R	从当前光标的位置开始替换字符, 并且使 vi 进入文本输入模式
.	重复上次的修改

搜索命令

这些键允许用户在文件中向前或者向后搜索指定的字符串。

键	功 能
/	从指定位置向后开始搜索指定的字符
?	从指定位置向前开始搜索指定的字符

习题

1. 编辑器是什么?
2. 文本格式器是什么?
3. 列出 UNIX 操作系统支持的编辑器。
4. 列出 vi 编辑器工作模式的种类。
5. 列出把 vi 切换到文本输入模式的键。
6. 解释为什么 vi 编辑器要使用缓冲区。
7. 列出保存文本, 退出 vi 编辑器的键。
8. 列出仅保存文件, 但是仍处于 vi 的键。
9. 列出使得 vi 处于命令工作模式的键。

10. 给出删除 1 行的操作和删除 5 行的操作。
11. 给出删除 1 个字符和删除 10 个字符的操作。
12. 列出能重复用户最新文本修改的键。
13. 把左列的命令与右列的解释相匹配。
- | | |
|----------|----------------------|
| 1. x | a. 把光标上移一行 |
| 2. r | b. 删除当前光标所在的字符 |
| 3. / | c. 把光标移到当前行的开头 |
| 4. ? | d. 把输入的文本插入到当前行的末尾 |
| 5. h | e. 把光标移到当前行的末尾 |
| 6. A | f. 从当前光标开始向后搜索指定的字符串 |
| 7. q! | g. 退出 vi 编辑器, 而不保存文件 |
| 8. wq! | h. 保存文件, 退出 vi |
| 9. a | i. 把输入的文本插入到当前光标的后面 |
| 10. \$ | k. 把光标左移一格 |
| 11. 0(零) | l. 从当前光标开始向前搜索指定的字符串 |
| 12. k | m. 替换当前光标所在的字符 |

上机练习

【实例】

这次上机练习, 要求生成一个小文件, 练习使用 vi 提供的编辑键。用户可以充分展开想像, 而不必拘泥于这个小文件。

尽量使用本章介绍的所有命令。

1. 用 vi 生成一个名为 `test` 的文件, 输入屏 1 上显示的内容。
2. 保存文件。

屏 1

```
The vi history
The vi editor was developed at the University of california,berkeley
as part of the berkeley unix system.
~
~
"test" [new file]
```

3. 重新打开文件, 插入一些文本后, 如屏 2 所示。
4. 再次保存文件。
5. 再次打开文件, 对其进行编辑, 使其看起来如屏 3。

屏 2

```
The vi history
The vi editor was developed at the University of california
berkeley as part of the berkeley unix system.
```

At the beginning the vi editor was part of another editor
The vi part of the ex editor was often used and became very.
This popularity forced the developers to come up with a separate
vi editor.
now the vi editor is independent of the ex editor and is available on
most of the UNIX operating system.
The vi editor is a good editor for everyday editing jobs.

屏 3

The vi history
The vi editor was developed at the University of California
Berkeley as part of the Berkeley UNIX system.
At the beginning the vi (visual) editor was part of the ex editor
and you had to be in the ex editor to use the vi editor.
The vi part of the ex editor was often used and became
very popular. This popularity forced the developers to come up
with a separate vi editor.
Now the vi editor is independent of the ex editor and is available
on most of the UNIX operating systems.
The vi editor is a good ,efficient editor for everyday editing jobs
although it could be more user friendly.

~
~
~
~

第 5 章 UNIX 文件系统介绍

本章是该书讨论 UNIX 文件系统两章中的前一章,第 7 章继续讨论。本章描述文件目录的基本概念和树状层次结构的组织,定义 UNIX 文件系统中使用的术语,讨论用于管理文件系统的命令,并解释文件和目录的命名规则。同时,用一些实际的例子表明如何使用相关命令。

5.1 磁盘组织

计算机的信息保存在文件中。用户写的便笺、建立的程序、编辑的文本都被保存成文件。然而,文件保存在何处?如何找到它们?用户可以命名每个文件,文件就保存在硬盘的某个部分。但是,磁盘的容量很大,怎么才能找得到文件保存在何处呢?用户可以把磁盘分成更小的单元和子单元,分别命名,然后把相关的信息保存在同一个单元或子单元中。

UNIX 处理磁盘遵循同样的过程。使用一台计算机时,用户操作的文件保存在计算机的随机访问存储器(RAM,或称为内存)中。UNIX 用 RAM 进行短期存储。但是,要长期保存,文件就通常保存在硬盘上(硬盘是最常见的存储媒质,然而也可以把文件存储在其他媒质上,如软盘、zip 盘或者磁带)。

UNIX 允许用户把硬盘分成许多单元(称作目录)和子单元(称作子目录),因此目录与目录形成网状。UNIX 提供了命令创建、组织和寻找目录与文件的功能。

5.2 UNIX 里的文件类型

在 UNIX 系统中,文件是字节序列。UNIX 不支持另外一些操作系统支持的其他结构(例如记录或者域)。UNIX 有 3 类文件。

规则文件:规则文件包含字节序列,可以是程序代码、文本等。用 vi 编辑器创建的文件是规则文件,大多数用户管理使用的都是这类文件。

目录文件:在很多方面,目录文件和其他文件一样,用户像命名其他文件一样命名目录文件。但是它不是标准的 ASCII 文本文件。目录文件包含关于其他文件的信息(例如,文件名)。它是由一组按照操作系统定义的特殊格式的记录所组成的。

特殊文件:特殊文件(设备文件)包含对应于外围设备(如打印机、磁盘等)的特殊信息。UNIX 把 I/O 设备看作文件,系统中的每一个设备——打印机、软盘、终端等——都有一个特殊文件。

5.3 目录详述

目录是 UNIX 文件系统的基本特征。目录系统提供了磁盘组织文件的结构。为了形象描述磁盘和它的目录结构,可以把磁盘想像成文件柜。文件柜可能有几个抽屉,可以用来比作目

录。每个抽屉可能分成几部分,与子目录类似。

UNIX 中,目录结构是分层组织。这种结构允许用户方便地组织和查找文件。最高层目录称作根目录(root),所有其他目录是它的直接或者间接分支。目录不包含所含文件的信息,而是提供一个索引路径,允许用户组织和查找文件。图 5.1 显示根和一些其他目录。

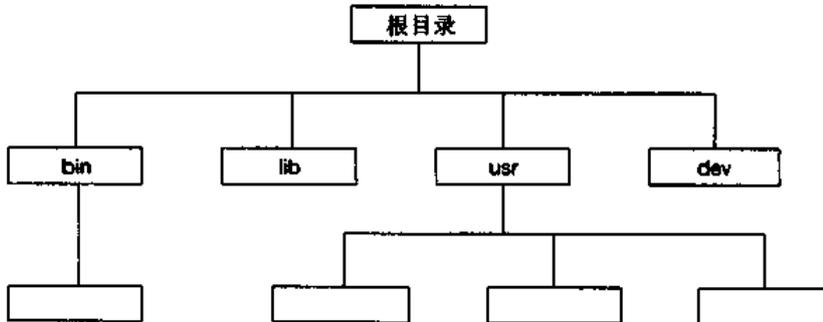


图 5.1 目录结构

层次结构的一个例子是家谱。一对夫妇可能有一个孩子,孩子又可能有几个孩子,这些孩子还可能有孩子。术语父和子用来描述层次之间的关系。图 5.2 显示了这种关系。只有根目录没有父亲。它是所有其他目录的祖先。

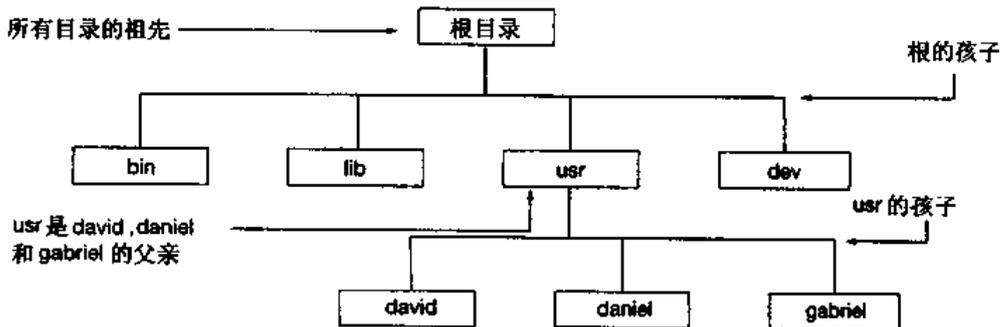


图 5.2 父子关系

层次目录结构经常用树来说明。代表文件结构的树通常颠倒过来画,根在顶部。用这样的树作类比,树根代表根目录,树枝是其他目录,叶子是文件。

5.3.1 用户主目录

系统管理员创建系统中所有的用户,为每个用户账号分配一个特定的目录。这个目录称为用户主目录或主目录。用户登录到系统时,自动处在用户主目录中。用户可以根据需要从这个目录扩展目录,能够添加任意多个子目录,子目录还可以再划分多个子目录。图 5.3 显示了 usr 目录有三个子目录,分别是 david、daniel 和 gabriel。david 目录包含三个文件,但是其他目录为空。

【说明】

1. 图 5.3 并不是标准的 UNIX 文件结构,文件结构随着安装程序的不同而不同。
2. 用户登录名和用户主目录名通常相同,由系统管理员指定。

3. 所有的 UNIX 文件结构中都有根目录。
4. 根目录的名字总是斜杠(/)。

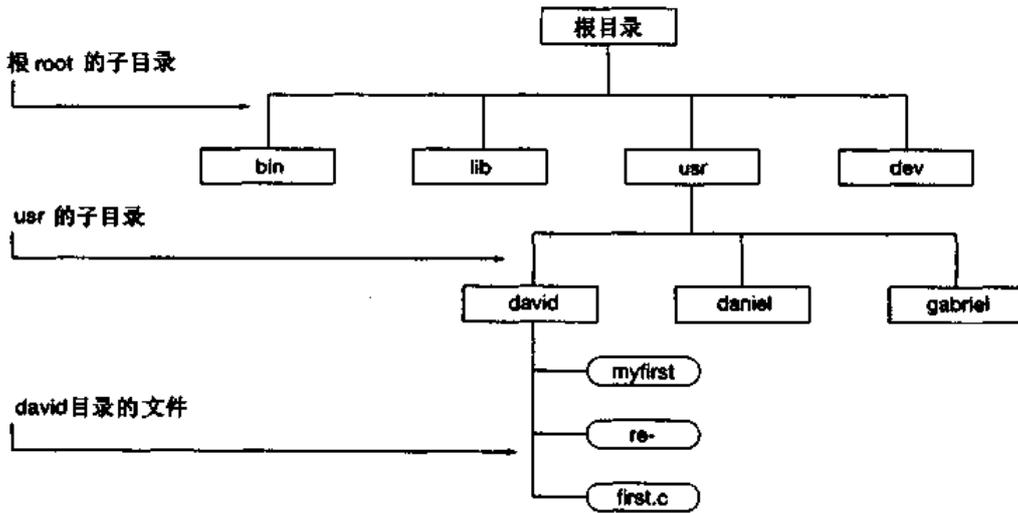


图 5.3 目录、子目录和文件

5.3.2 工作目录

在 UNIX 上工作时,用户总处于某个目录中。相关或者说正在工作的这个目录称为工作目录或当前目录。有几个命令允许用户查看和更改工作目录。

5.3.3 理解路径和路径名

每个文件有一个路径名。路径名在文件系统中定位文件。图 5.4 显示了目录与文件的层次和路径名。从根目录开始,经过所有中间目录直到某个文件,用户就确定了文件的路径名。例如,在图 5.4 中,如果当前目录是 root,那么访问 david 目录下的文件(例如,myfirst)的路径是/
usr/david/myfirst。

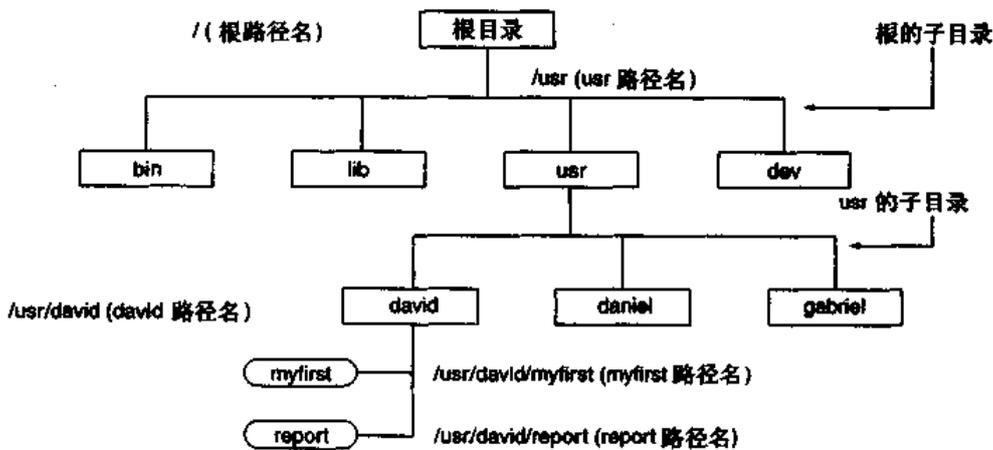


图 5.4 目录结构的路径名

每个路径的最后或者是一个普通文件(称作文件),或者是一个目录文件(称为目录)。普

通文件在路径的末尾,不能再有更下级目录,打个比方,叶子不能再长出树枝。目录文件是文件结构中能够支持其他路径的位置点,就像树枝还能有分支一样。

【说明】

1. 路径名开始的斜杠(/)代表根目录。
2. 其他斜杠用来分隔目录和文件名。
3. 工作目录的文件可以立即访问。访问其他目录的文件需要用路径名指定该文件。

绝对路径名:绝对路径名(完全路径名)是从根开始定位一个文件。绝对路径名总是以根目录名(/)开始。例如,如果工作目录是 `usr,david` 目录下 `myfirst` 文件的绝对路径名是 `/usr/david/myfirst`。

【说明】

1. 绝对路径名确切指定从哪里找到文件。因此,可以用来指定工作目录和其他任何目录的文件定位。

2. 绝对路径名总是从根目录开始的,因而在路径名开始处总有斜杠(/)。

相对路径名:相对路径名是路径名的一种更短的形式。它从工作目录开始定位一个文件。与绝对路径名类似,相对路径名也可以描述经过多个目录的路径。例如,如果工作目录在 `usr,david` 目录下的文件 `REPORT` 的相对路径名是 `david/REPORT`。

【说明】

相对路径名开始没有斜杠(/)。它总是从当前目录开始。

5.3.4 使用文件和目录名

每个普通的目录文件都有文件名。UNIX 给用户许多自由来命名文件和目录。文件名的最大长度取决于 UNIX 的版本和系统厂商。所有 UNIX 系统允许至少 14 个字符长的文件名,大多数可支持长达 255 个字符的长度。

文件通过字母和数字的组合来命名。惟一的例外是根目录,它用斜杠(/)命名和索引。其它文件都不能用这个名字。

对于使用的 shell(命令解释程序)来说,某些字符有特殊含义。如果那些字符被用在文件名中,shell 把它们作为命令的一部分进行解释和执行。尽管有办法重载特殊字符的解释,但是应该避免这样做。特别注意,不要在文件名中用下面这些字符:

< >	小于大于号
()	小括号
[]	中括号
{ }	大括号
*	星号
?	问号
"	双引号
'	单引号
-	减号

\$ 美元符号
^ 脱字符

从下面的列表中选择字符作为文件名会减少混乱:

(A ~ Z) 大写字母
(a ~ z) 小写字母
(0 ~ 9) 数字
(_) 下划线
(.) 点(句点)

UNIX 用空格表示命令或者文件名的开始和结束。所以,文件名中想要用空格的地方可以用句点或者下划线。例如,如果想把一个文件命名为 MYNEWLIST,可以叫作 MY_NEW_LIST 或者 MY.NEW.LIST。

选择的文件名应该有意义。像 junk、whoopee 或者 dxqx 这些名字是正确的文件名,但它们并不好,因为这些名字不能帮助用户回忆起文件中保存的是什么。选择的文件名应该尽可能描述文件的内容。下面的文件名有正确的语法,同时也表示了文件内容的信息。

```
REPORTS      Jan_list      my_memos
shopping.lis Phones      edit.c
```

UNIX 操作系统大小写敏感:大写字母和小写字母有区别。在文件名中,可以大小写字母混合使用。但是要记住,文件名 MY_FILE、My_File 和 my_file 被看作不同的文件。

UNIX 不考虑普通文件和目录文件名字之间的区别。因此,目录和文件可能有同样的名字。例如,目录 lost + found 中可以有文件名为 lost + found。

用父子做类比,同一个目录不能有相同名字的两个文件,就像同一个父亲的孩子的名字都不同一样。给孩子取不同名字的父亲是明智的,而在 UNIX 中,这是强制的。但是,和不同父亲的孩子一样,不同目录的文件名可以相同。

【注意】

文件名的任何部分都不要使用空格。

文件名扩展:文件名扩展用来进一步分类,描述文件内容。扩展名是文件名的一部分,跟在句点后,大多数情况是可选的。某些语言的编译器,例如 C,要依赖特定的扩展名(编译器在第 10 章解释)。图 5.5 中,first.c 有一个典型的扩展名,.c 表示的是 C 语言程序。

下面的例子显示了一些扩展名。

```
report.c      report.o
memo.04.10
```

注意 UNIX 中允许在文件扩展中使用多个句点。

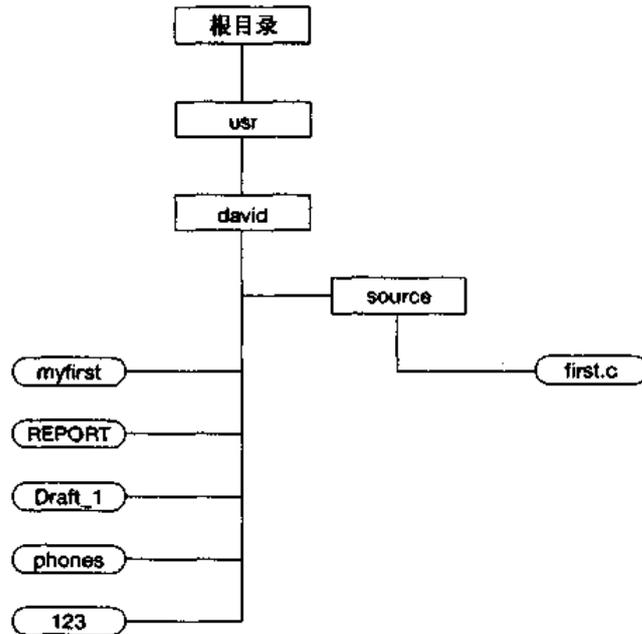


图 5.5 目录结构示例

5.4 目录命令

我们已经对基本的文件概念和定义有了一定的了解,现在来学习如何操作文件和目录。下面的例子和命令序列展示了如何使用这些命令来管理文件。

在以下的例子中,假设用户登录名为 david,图 5.5 为目录结构,用户主目录是 david。

5.4.1 显示目录路径名:pwd 命令

pwd(print working directory,打印工作目录)命令显示用户当前所在目录的绝对路径名。例如,当用户初次登录到 UNIX 系统上时,处在用户主目录中。为了显示用户主目录的路径名,这里也就是工作目录,当登录以后就可以用 **pwd** 命令查看。

【实例】

登录并显示用户主目录的路径名。

```

login:david[Return].....输入登录名(例如 david)
password: .....输入口令
Welcome to UNIX!
$ pwd[Return].....显示用户主目录的路径
/usr/david
$_. .....命令提示符
  
```

【说明】

1. /usr/david 是用户主目录的路径名。
2. /usr/david 也是用户当前工作目录的路径名。
3. /usr/david 以斜杠“/”开始,是一个绝对路径名,表示从根目录开始直到用户主目录的路径。

4. david 是登录名和用户主目录名。

定位工作目录中的文件:工作目录是 david,图 5.5 显示有 5 个文件和 1 个名为 source 的目录,其中有 1 个文件。david 目录中的文件 myfirst 的路径名是 /usr/david/myfirst,这是文件的绝对路径名。如果文件在工作目录中,不需要用完整的路径名来指示,文件名(这里是 myfirst)就足够了。

定位其他目录中的文件:当文件不在工作目录时,需要指定文件所在的目录。例如,工作目录是 usr。用户 source 目录中的文件 first.c 的路径名为 david/source/first.c。

【说明】

david/source/first.c 称为相对路径。它并不是从根目录开始的。

5.4.2 改变工作目录:cd 命令

用户不能总是工作在用户主目录中,工作目录需要从一个目录改变到其他目录。cd (change directory,改变目录)命令用来指定工作目录。

【实例】

把工作目录改变到 source 目录,输入如下命令。

```
$ pwd [Return].....检查当前目录
/usr/david
$ cd source [Return].....改到 source 目录
$ pwd [Return].....显示工作目录
/usr/david/source
```

假设用户有权限把工作目录改为 daniel,如下操作:

```
$ cd /usr/daniel [Return].....改为 daniel 目录
$ pwd [Return].....查看工作目录
/usr/daniel
$_.....命令提示符
```

返回用户主目录:如果目录有许多层,而当前目录处在目录结构中比较深的位置时,不用输入太多字符就可以回到用户主目录会非常方便。用户可以用 \$HOME(保存用户主目录路径名的变量)作为 cd 命令的目录名。或者仅输入 cd 然后按回车键,默认回到用户主目录。

【实例】

练习 cd 命令的使用。

```
$ cd $HOME [Return].....回到用户主目录
$ cd source [Return].....改为 source 目录
$ cd [Return].....没有指定目录名;默认是用户主目录
$ pwd [Return].....检查是否在用户主目录
/usr/david
$ cd xyz [Return].....改为 xyz 目录,该目录不存在,得到如下信息:
xyz: not a directory
$_.....等待下面的命令
```

5.4.3 创建目录

第一次登录 UNIX 系统时,用户从用户主目录开始工作。此时,用户主目录中很可能没有文件或者子目录,用户需要自己建立子目录。

创建目录的优点

UNIX 的目录结构没有限制。把所有文件都放在用户主目录中是允许的。尽管创建有效的目录结构需要花一点时间,但会带来许多好处。尤其当用户有大量文件时,好处就很明显了。下面列出使用目录的优点:

- 把相关的文件成组放在同一个目录中,记忆和访问都很容易
- 屏幕上显示更短的文件列表,方便用户迅速找到文件
- 不同的目录中可以保存具有相同文件名的文件
- 目录可以使多个用户共享一个大容量磁盘,每个用户拥有合适的空间
- 可以利用管理目录的 UNIX 命令

使用目录应该仔细计划。如果把文件合理分组到各个目录,能节省很多时间;相反如果只是杂乱地创建并填充目录,那么寻找文件和识别它们可能很费力。

创建目录:mkdir 命令

`mkdir` 命令能在工作目录或者其他任何命令指定的目录下创建一个新的子目录。例如,下面的命令显示了如何在工作目录和其他目录中创建子目录。图 5.6 表明增加新目录后的目录结构。

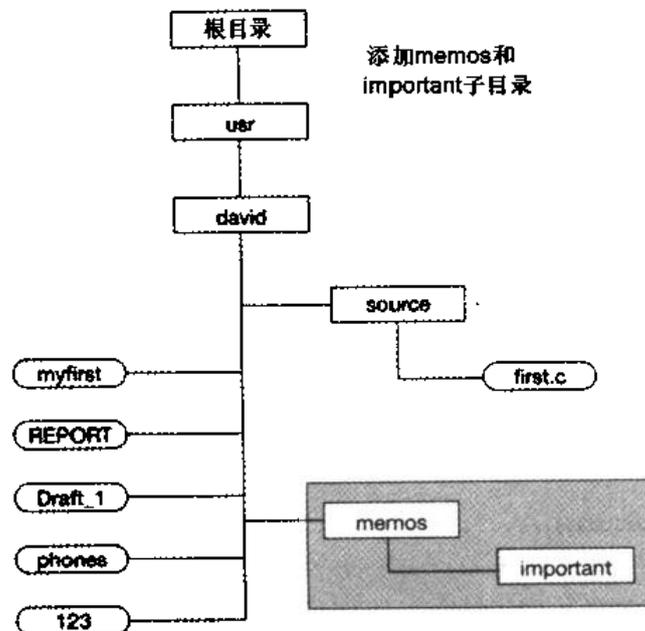


图 5.6 图 5.5 增加新目录后的目录结构

【实例】

在用户主目录下创建名为 `memos` 的目录。

```

$ cd [Return].....确保在用户主目录
$ mkdir memos [Return].....创建名为 memos 的目录
$ pwd [Return].....查看工作目录

/usr/david

$ cd memos [Return].....改为新目录
$ pwd [Return].....查看工作目录/usr/david/memos

$ _.....当前目录是 memos

```

【实例】

回到用户主目录,在 memos 目录下创建名为 important 的子目录。

```

$ cd [Return].....确保在用户主目录
$ mkdir memos/important [Return].....指定新目录的路径名
$ cd memos/important [Return].....改到新目录
$ pwd [Return].....查看工作目录

/usr/david/memos/important

$ _.....现在工作目录是 important

```

【说明】

目录结构可以根据指定的要求来创建。

-p 选项:一行命令就可以创建一个完整的目录结构。使用 **-p** 选项在当前目录下逐级创建目录。例如,假设在用户主目录内创建一个三层的目录,下面的命令显示了如何去做。图 5.7 描绘了运行命令后的目录结构。

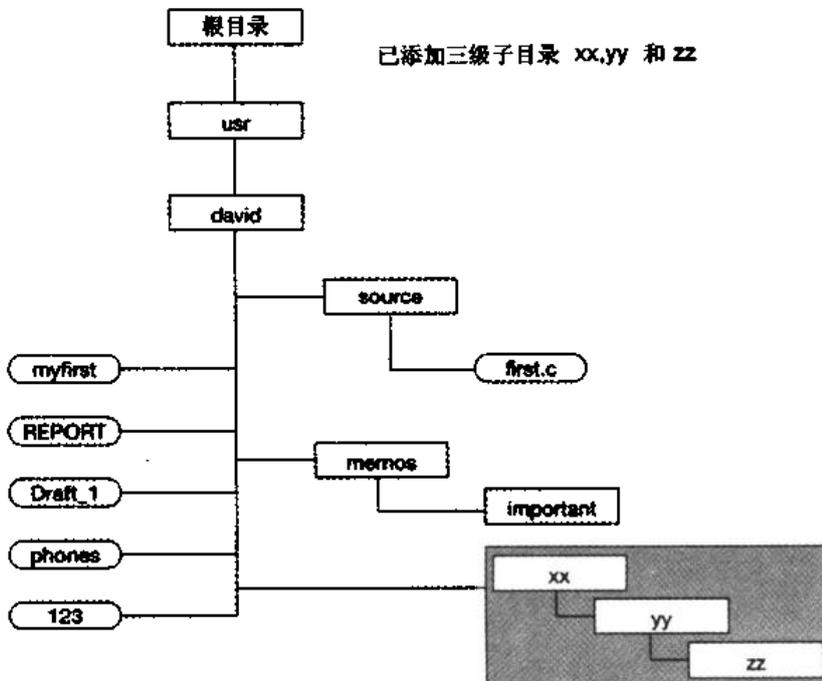


图 5.7 图 5.6 创建三层深的目录后的目录结构

【实例】

处于用户主目录,创建三级目录结构。

```
$ cd [Return].....确保在用户主目录
$ mkdir -p xx/yy/zz [Return].....创建目录 xx;xx 下创建目录 yy;yy 下创建目录 zz
$_.....等待用户命令
```

【注意】

1. 创建的目录应该不存在。
2. 用不着非得创建当前目录的子目录。只要给出新目录的路径名,可以在任何一级目录运行该命令。

5.4.4 删除目录:rmdir 命令

有时候不再需要某个目录,或者误建了一个目录。在这两种情况下,用户要删除不需要的目录,UNIX 提供了该命令。

rmdir(remove directory,删除目录)命令删除指定的目录。但是,它只能删除空目录——除了本目录(.)和父目录(..)外,该目录中不包含任何其他子目录或者文件。

【实例】

删除 memos 目录中的 important 目录。

```
$ cd memos [Return].....工作目录改为 memos
$ pwd [Return] ..... 确保在 memos 中
/usr/david/memos
$ rmdir important [Return]..... 删除 important 目录
$_.....等待命令
```

【说明】

1. important 子目录是一个空目录,所以能够删除。
2. 必须在父目录中删除子目录。

【实例】

从 david 目录中试着删除 source 子目录。

```
$ cd [Return].....改成 david 目录
$ rmdir source [Return] .....删除 source 目录
rmdir: source: Directory not empty
$ rmdir xyz [Return] .....删除 xyz 目录
rmdir: xyz: Directory does not exist
$_.....等待命令
```

【说明】

1. 由于 source 目录不空,其中还有文件,因此无法删除。
2. 如果用户给出错误的目录名,或者在指定的路径名中无法定位目录,**rmdir** 就返回错误信息。
3. 必须处在父目录或者更高层的目录中删除子目录。

5.4.5 目录列表:ls 命令

`ls`(list,列表)命令用来显示指定目录的内容。命令根据文件名的字母顺序,列出信息,列表包括文件名和目录名。如果没有指定目录,就只列出当前目录的信息。如果指定文件名而不是目录名,那么 `ls` 显示该文件名以及其他相应的信息。

在下面的例子和命令中要用到图 5.8 所示的目录结构,后续的插图说明了在该目录结构上使用各命令实例的效果,这些插图有利于更好地理解命令实例。

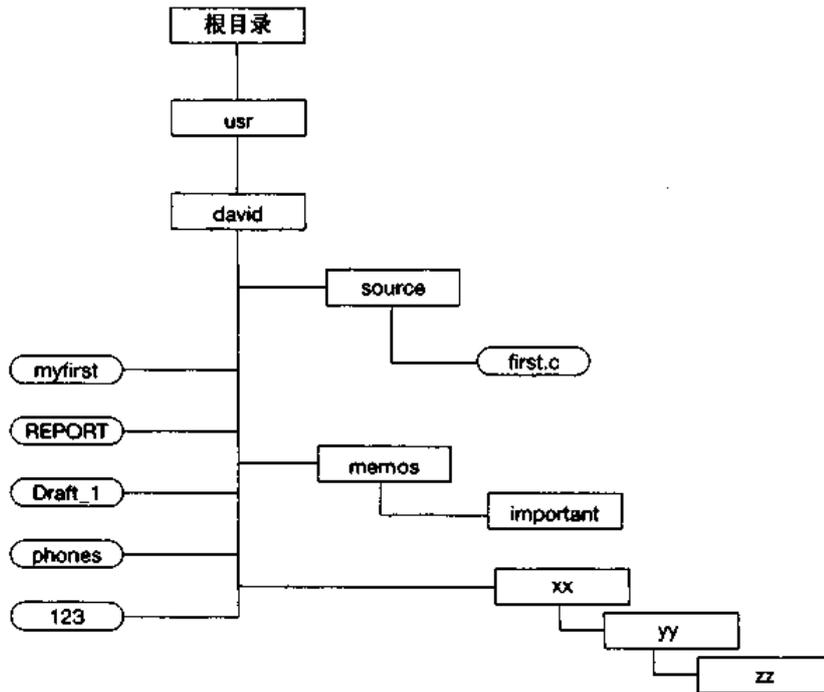


图 5.8 命令实例所使用的目录结构

【说明】

1. 目录列表只包含文件和子目录名。其他命令允许用户读取文件内容。
2. 如果没有指定目录名,默认为当前目录。
3. 文件名并不表明它是文件还是目录。
4. 默认情况下按照字母顺序排序:数字在字母前,大写字母在小写字母前。

【实例】

假设当前目录是 `david`,输入 `ls` 显示用户目录的内容。用户也可以列出其他目录的内容,或者列出一个单独的文件查看它是否存在于指定的目录。

```

$ ls
123
Draft_1
REPORT
memos
myfirst
phones

```

```
source
xx
$
```

【实例】

在用户主目录 david 下,列出目录 source 的文件:

```
$ cd [Return].....确保在 david 的主目录
$ ls source [Return].....显示目录 source 的文件列表
first.c
$_ .....等待用户命令
```

【实例】

在用户主目录查看 first.c 是否存在于 source 目录。

```
$ ls source/first.c [Return].....显示目录 source 中的 first.c,看它是否存在。
first.c .....确实存在,所以显示文件名
$ ls xyz [Return].....显示是否存在文件 xyz。如果不存在,得到错误信息
xyz: Not such a file or directory
$_ .....恢复命令提示符
```

ls 选项:当需要文件的更多信息或者想以不同格式列表时,可以用带选项的 **ls** 命令。这些选项提供了许多方式显示文件,如按照列格式、显示文件大小或者区分文件名和目录名来显示。

表 5.1 列出了大部分 **ls** 命令选项。使用这些选项,观察其输出。

表 5.1 ls 命令选项

选 项	功 能
-a	列出所有文件,包括隐藏文件
-C	以多列的格式列表,按列排序
-F	如果是目录,文件名后加斜杠(/);如果是可执行文件,加星号(*)表示
-l	按照长格式列表,显示文件的详细信息
-m	按页宽列文件,以逗号分隔
-p	如果是目录,文件名后加斜杠(/)
-r	以字母反序列列表
-R	循环列出子目录的内容
-s	以文件块为单位显示文件大小
-x	以多列的格式列表,按行排序

【说明】

1. 每个选项字母前需要加连字符。
2. 命令名和选项之间必须要有空格。
3. 加目录的路径名列出其他目录的文件。
4. 一行命令中可以多于一个选项。

信息最多的选项是 `-l`。带 `-l` 选项的 `ls` 命令产生的列表每行显示一个文件或子目录的多列信息。

【实例】

按照长格式列表显示当前目录。

□ 输入 `cd`, 按回车键。然后输入 `ls -l`, 按回车键。

图 5.9 是显示输出图。“total 11”表明显示出的文件总数。文件大小以块(通常是 512 字节)为单位显示块数。

```
$ cd
$ ls -l
total 11
-rw-r--r-- 1 david student 1026 Jun 25 12:28 123
-rw-r--r-- 1 david student 684 Jun 25 12:20 Draft_1
-rw-r--r-- 1 david student 342 Jun 25 12:28 REPORT
drw-r--r-- 1 david student 48 Jun 25 12:28 memos
-rw-r--r-- 1 david student 342 Jun 25 12:28 myfirst
-rw-r--r-- 1 david student 342 Jun 25 12:28 phones
drw-r--r-- 1 david student 48 Jun 25 12:28 source
drw-r--r-- 1 david student 48 Jun 25 12:28 xx
$_
```

图 5.9 `ls` 命令和 `-l` 选项

图 5.10 表明每列所表示的信息。

1	2	3	4	5	6	7
-rwx rw--	1	david	student	342	Jun 25 12:28	myfirst
Column 1.....	Consists of 10 characters. The first character indicates the file type and the rest indicate the file access mode.					
Column 2.....	Consists of a number indicating the number of the links.					
Column 3.....	Indicates the owner's name.					
Column 4.....	Indicates the group name.					
Column 5.....	Indicates the size of the file in bytes.					
Column 6.....	Shows the date and time of last modification.					
Column 7.....	Shows the name of the file.					

图 5.10 长格式的 `ls` 命令

文件类型:第 1 列由 10 个字符组成, 每行的第一个字符表示文件类型。下面总结了文件类型。

- 普通文件
- d 目录文件
- b 块设备文件, 例如磁盘
- c 字符设备文件, 例如打印机

文件访问模式:第 1 列接着的 9 个字符由三组 r、w、x 和连字符(-)组成,描述了文件的访问模式。它们表明了系统中的每个用户能够如何访问指定的文件,称为文件访问或许可模式。表 5.2 解释了每组字母 r、w、x 和连字符(-)的含义。

表 5.2 文件许可权字符

关键字	许可权设置
r	允许读
w	允许写
x	允许执行(允许作为程序来运行)
-(连字符)	权限不允许

【说明】

1. 如果是可执行文件,标记执行许可权(x)。
2. 如果显示连字符而不是字母,表示该权限被禁止。
3. 如果是目录文件,x 解释成查找目录中指定文件的许可权。

每一组字符允许或者拒绝不同用户组的许可权。第一组 rwx 字母允许所有者的读、写和执行权限,第二组 rwx 允许用户组,第三组允许其他用户。通过对不同用户组设置不同的访问字符,用户可以控制谁访问该文件以及具有何种类型的访问权。例如,假设在 david 目录,运行如下命令:

```
$ ls -l myfirst [Return].....长格式列表 myfirst
-rwx rw- --- 1 david student 342 Jun 25 12:28 myfirst
```

由于第 1 个字符是连字符,所以输出显示 myfirst 是一个普通文件。第一组字符(rwx)表明所有者可以读、写和执行。第二组字符(rw-)表明该用户组可以读写,但是不能执行。第三组字符(- - -)是三个连字符,说明其他用户所有权限都被拒绝。

链接数:第 2 列显示链接数。链接(ln 命令)在第 7 章讨论。这里的例子中,myfirst 的链接数是 1。

文件所有者:第 3 列显示文件的拥有者。该名字通常是创建文件的用户 ID,例如 david。

文件组:第 4 列显示用户组。每一个 UNIX 用户有用户 ID 和组 ID,这些是由系统管理员指定的。例如,同一个项目的人设成相同的组 ID。这个例子中,文件组是 student。

文件大小:第 5 列显示文件大小,这是文件的字节数。这里文件大小为 342 字节。

日期和时间:第 6 列显示最后更改的日期和时间。myfirst 最后在 6 月 25 日 12 点 28 分更改。

文件名:第 7 列显示文件名,这里是 myfirst。

【实例】

在 david 目录中,以长格式列出 source 目录的文件。

```
$ cd [Return] .....确保在用户主目录
$ ls -l source [Return].....列出 source 目录的文件
-rwx rw- -- 1 david student 342 Jun 25 12:28 first.c
$_.....命令提示符
```

【说明】

1. 在用户主目录,列出 source 目录的内容。
2. source 目录中只有一个文件。
3. 连字符(-)表明文件 first.c 是普通文件。
4. 用户具有读、写和执行权限(rwx)。用户组有读写权限(rw)。其他用户的各种权限都被拒绝(---)。
5. first.c 有 1 个链接。
6. 所有者是 david,组名是 student。
7. first.c 大小为 342 字节。
8. first.c 最后更改时间是 Jun 25 12:28。

【实例】

为了以字母反序列出用户主目录中的文件名,可键入 `ls -r`,然后按回车键(见图 5.11)。

```
$ ls -r
xx
source
memos
myfirst
phones
REPORT
Draft_1
123
$_
```

图 5.11 ls 命令和 -r 选项

【注意】

选项是小写字母 r。

【实例】

为了以列格式显示当前目录的内容,可键入 `ls -C`,然后按回车键。

【说明】

每列按照字母排序(见图 5.12)。

```
$ ls -C
123          Draft_1      REPORT      memos
myfirst      phones      source      xx
$_
```

图 5.12 ls 命令和 -C 选项

【实例】

为了显示当前目录的内容,以逗号分隔,可键入 `ls -m`,然后按回车键(见图 5.13)。

```
$ ls -m
123 Draft_1,REPORT,memos,myfirst,phones,source,zz
$_
```

图 5.13 `ls` 命令和 `-m` 选项

隐藏文件

以句点开始的文件名称为不可见文件或者隐藏文件,目录列表命令通常不显示它们。启动文件通常不可见(文件名以句点开始),以避免搞乱目录(启动文件在第 7 章讨论)。

用户可以在用户主目录或者其他任何子目录中创建自己的不可见文件。只要文件名以句点(.)开始即可。除了根目录,每个目录中都有两个特殊的不可见文件项——单点和双点(“.”和“..”)。

.和..文件项:`mkdir` 命令自动为创建的目录设置两项——单句点和双句点,分别代表当前目录和高一级目录。用父子类比,双点(..)代表父目录,单点(.)代表子目录。

这些目录缩写可以用在 UNIX 命令中表示父和当前目录。

【实例】

列出 `source` 目录的所有文件,包括隐藏文件。

- 输入 `cd source`,按回车键,改为 `source` 目录
- 输入 `ls -a`,按回车键,列出所有文件(包括隐藏文件)

```
.
..
first.c
```

【说明】

1. 点(.)表示当前目录(/usr/david/source)。
2. 双点(..)表示父目录(/usr/david)。

【实例】

工作目录改成父目录,如下操作。

```
$ cd .. [Return].....改为父目录(/usr/david)
$ pwd [Return].....查看现在的位置
/usr/david
$_.....命令提示符
```

【说明】

1. 现在,点(.)表示当前目录(/usr/david)。
2. 双点(..)表示父目录(/usr)。

【实例】

列出 david 父目录的文件,用逗号分隔。

```
$ cd [Return] .....回到 david
$ ls -m .. [Return].....列出 david 父目录(是 usr)的文件;用逗号分隔显示
david, daniel, gabriel
$_.....等待输入
```

使用多个选项

一行命令中可以带多个选项。例如,如果想要列出所有文件,包括不可见文件(选项 - a)、长格式(选项 - l)和按照字母反序排列(选项 - r),输入 `ls -mar` 或者 `ls -m -a -r`,然后按回车键。

【说明】

1. 可以用一个连字符开始选项,但是选项字母之间不能有空格。
2. 命令行中选项字母的顺序并不重要。
3. 可以给每个选项一个连字符,但是在每个选项之间需要有空格。

【实例】

按行满屏列出用户目录的内容,目录名用斜杠(/)表明。

```
$ cd
$ ls -m -p
123, Draft_1, REPORT, memos/, myfirst, phones, source/, xx/
$_
```

【说明】

这里有两个选项; - m 逐行满屏显示文件名, - p 在目录文件名后加斜杠(/)。

【实例】

以逗号分隔,显示所有的文件名,斜杠表明目录,星号表明可执行文件。

```
$ cd
$ ls -am*
./, ../, 123, Draft_1, REPORT, memos/, myfirst, phones, source/, xx/
$_
```

【说明】

1. 用了三个选项, - a 显示隐藏文件, - m 满屏显示文件名, - F 在文件名后分别用斜杠和星号指示目录和可执行文件。
2. 两个不可见文件是目录文件,在文件名后有斜杠表明。

【实例】

列出用户主目录的所有文件,按照列格式,反序。

```
$ ls -arC
xxx          phones          memos          REPORT         123
```

```
source      myfirst    Draft_1
$_
```

【实例】

列出 david 的文件,用逗号分隔,显示文件大小。

```
$ ls -s -m
total 11, 3 123, 2 Draft_1, 1 REPORT, 1 memos, 1 myfirst, 1 phones,
1 source, 1 xx
$_
```

【说明】

1. 第 1 个域(total 11)表示文件总大小,通常以 512 字节大小的文件块为单位。
2. 选项 -s 显示文件大小;无论文件可能多小,每个文件至少 1 块(512 字节)。

【实例】

按照列格式列出 david 目录的所有文件,并显示大小。

```
$ ls -a -x
total 13          3 123          1 memos      1 source
1 .              3 Draft_1    1 myfirst    1 xx
1 ..            1 REPORT                    1 phones
$_
```

【说明】

1. 总大小是 13 块,因为两个隐藏文件加进来了。
2. -x 选项与 -C 的列格式有所不同。每行按照字母排序。

【实例】

以列格式显示 david 的目录结构。

```
$ cd
$ ls -R -C
123          Draft_1  REPORT  memos  myfirst  phones
source      xx
./memos:
./source:
first.c
./xx:
yy
./xx/yy
zz
./xx/yy/zz:
$_
```

【注意】

命令选项是大写 R 和 C。

【说明】

1. 选项 -R 列出当前目录 david 中的文件名,有三个子目录:memos、source 和 xx。
2. 路径名后跟随冒号(例如 ./memos:)显示每个子目录,然后列出该目录中的文件。

3. 路径名是相对路径,从当前目录开始(当前目录以路径名开始处的点号表示)。

5.5 显示文件内容

本书到目前为止,我们学习了用文件管理命令来查看目录定位文件和查看文件名列表。要是查看文件内容该怎么办呢?用户可以把文件打印出来,得到内容的复印件;或者用 vi 编辑器打开文件,在屏幕上看。另外,也可以用 `cat` 命令达到这个目的。

`cat` 命令用来显示、创建或者合并文件。本章只介绍 `cat` 命令的显示功能。例如,输入 `cat myfirst`,然后按回车键,显示名为 `myfirst` 的文件内容。

如果当前目录中有 `myfirst`(没有指定路径,默认为当前目录),`cat` 命令在屏幕(标准输出设备)上显示 `myfirst` 的内容。如果指定了两个文件名,那么可以看到两个文件的内容(以命令行中指定的顺序一个接一个显示)。例如,输入 `cat myfirst yourfirst`,然后按回车键,显示两个文件 `myfirst` 和 `yourfirst` 的内容。

如果文件很长,用户看到的只有最后的 23 行,其他行已经从眼前翻过。除非你阅读速度极快,否则什么也看不到。解决办法是通过按[Ctrl-s]停止卷屏,然后用[Ctrl-q]恢复。

用这样的方式查看文件内容相当不方便。但是请耐心,更好的办法将在后边介绍。UNIX 有其他的命令可一次一页地显示长文件。

【注意】

每一个[Ctrl-s]要用[Ctrl-q]取消,否则屏幕保持锁定,不响应键盘输入。

5.6 打印文件内容

通过 vi 编辑器或 `cat` 命令可以在屏幕上查看文件的内容。但是,有时用户也需要文件的复印件。

UNIX 提供了命令。它能把用户文件发送到打印机,显示打印工作的状态,并在用户改变主意时允许撤消打印任务。下面说明这些命令。

5.6.1 打印:lp 命令

`lp` 命令把文件发送到打印机,得到文件的硬(纸张)拷贝。例如,想要打印文件 `myfirst` 的内容,输入 `lp myfirst`,然后按回车键。

UNIX 通过显示类似下面的请求 ID 来确认用户请求:

```
request id is lp1 - 8054 ( 1 file )
```

【注意】

与所有命令相同,命令(`lp`)和参数(文件名)之间有空格。

如果指定的文件名不存在,或者 UNIX 找不到该文件,那么 `lp` 返回如下信息:

```
lp: can't access file "xyz"  
lp: request not accepted
```

【实例】

一行命令中可以指定多个文件。

```
$ lp myfirst REPORT phone [Return].....打印文件 myfirst,REPORT 和 phone
request id is lp1 - 6877 ( 3 files )
$_.....等待其他命令
```

【说明】

这个请求只生成一个标题页(第一页)。但是,每开始打印一个文件时都换新页。

每个打印请求与一个 id 号相联系。id 号表明这个打印任务,例如当取消打印任务时需要 id 号。表 5.3 显示用户可以指定的选项。

表 5.3 lp 命令选项

选 项	功 能
-d	在指定的打印机上打印
-m	完成打印请求后,向用户邮箱发邮件
-n	打印指定份数
-s	取消反馈消息
-t	输出的标题页打印指定标题
-w	完成打印请求后,在用户终端显示消息

-d 选项:系统可能接通了多个打印机。用 **-d** 选项指定打印机。如果没有指定,使用默认打印机。

【实例】

显示 **-d** 选项的例子。

```
$ lp -d lp2 myfirst [Return].....在打印机 lp2 打印 myfirst
request id is lp2 - 6879 ( 1 file )
$_.....等待下面的命令
```

【说明】

打印机的名称没有标准化,随着安装时的不同设置而不同。

-m 选项:当正常完成打印请求时,**-m** 选项向用户发送邮件(邮件和邮箱在第 9 章讨论)。

【实例】

显示 **-m** 选项的例子。

```
$ lp -m myfirst [Return].....打印 myfirst 文件,完成时发邮件
request id is lp1 - 6869 ( 1 file )
$_.....等待下面命令
```

完成打印任务时,邮箱里收到类似下面的邮件:

```
From LOGIN:
printer request lp1 - 6869 has been printed on the printer lp1
```

-n 选项:当需要打印多份副本时,使用 **-n** 选项。默认为 1 份。

【实例】

显示 **-n** 选项的例子。

```
$ lp -n3 myfirst [Return].....默认打印机上打印 3 份 myfirst
request id lp1 - 6889 ( 1 file )
$_.....等待下面命令
```

-w 选项:打印完成时,在用户终端显示消息。如果用户没有登录,消息以邮件形式发送到用户邮箱。

【实例】

显示 **-w** 选项的例子。

```
$ lp -w myfirst [Return].....打印 myfirst 文件,完成时显示消息
request id lp1 - 6872 ( 1 file )
$_.....等待下面命令
```

完成打印任务时,系统响铃并显示如下消息:

```
lp: printer request lp1 - 6872 has been printed on the printer lp1.
```

-t 选项:在输出的标题页(第 1 页)上打印指定字符串。

【实例】

显示 **-t** 选项的例子。

```
$ lp -t hello myfirst [Return].....打印 myfirst 文件,标题页打印"hello"
request id lp1 - 6889 ( 1 file )
$_.....等待下面命令
```

5.6.2 取消打印请求:cancel 命令

cancel 命令用来取消不想要的打印请求。如果用户把一个错误的文件送到打印机,或者不愿再等一个长时间的打印任务,这时就可以用 **cancel** 命令。使用该命令时,需要指定打印任务的 ID(**lp** 提供)或者打印机名。

【实例】

下面的命令说明如何使用 **cancel** 命令。

```
$ lp myfirst [Return].....默认打印机上打印 myfirst
request id lp1 - 6889 ( 1 file )
$_.....等待其他命令
$ cancel lp1 - 6889 [Return].....取消指定的打印请求
Request "lp1 - 6889" canceled
$_.....等待其他命令
$ cancel lp1 - 6889 [Return].....取消打印机 lp1 上当前的请求
request "lp1 - 6889" canceled
$ cancel lp1 - 85588 [Return].....取消打印请求,错误的请求 id 系统显示错误信息
cancel request "lp1 - 85588" nonexistent
$ cancel lp1 [Return].....取消打印机 lp1 的当前任务;如果没有打印工作,显示
```

消息

```
cancel: printer "lp1" was not busy
$_..... 等待其他命令
```

【说明】

1. 指定打印请求 ID, 取消打印任务(即使现在正在打印该任务)。
2. 指定打印机名称, 仅仅取消正在该打印机上打印的任务。队列中的其他打印任务仍然被打印。
3. 这两种情况都不影响打印机打印下一个请求。

5.6.3 获取打印机状态: lpstat 命令

lpstat 命令用来获得打印请求和打印机状态的信息。使用选项 **-d** 找出系统的默认打印机名。

【实例】

使用如下命令来练习使用 **lpstat**。

```
$ lp source/first.c [Return].....打印 source 目录的 first.c
request id lp1 - 6877 ( 1 file )
$ lp REPORT [Return]..... 打印 REPORT
request id lp1 - 6878 ( 1 file )
$ lpstat [Return]..... 显示打印机状态
lp1 - 6877 student 4777 jun 11 10:50 on lp1
lp1 - 6877 student 4777 jun 11 10:50 on lp1
$ cancel lp1 - 6877 [Return]..... 取消指定的打印请求
request "lp1 - 6877" canceled
$ lpstat-d [Return]..... 显示默认打印机名
system default destination: lp1
$ cancel lp1 [Return]..... 取消当前打印任务
```

【说明】

如果队列中没有打印请求和正在打印的任务, **lpstat** 什么也不显示就返回 **\$** 命令提示符状态。

5.7 删除文件

用户已经知道了如何创建文件和目录, 也知道如何删除空目录。但是, 如果目录不空的话, 应该怎么做呢? 首先必须删除所有的文件和子目录。如何删除文件呢?

使用 **rm**(remove) 命令可以删除不再需要的文件。指定文件名, 删除工作目录里的文件; 或者指定了路径名, 删除位于其他目录的文件。图 5.14 显示了删除一些文件之后的目录结构。

【实例】

下面的命令显示如何使用 **rm** 命令。

```
$ cd [Return].....改变到用户主目录
```

```

$ rm myfirst [Return]..... 从用户主目录删除 myfirst
$ rm REPORT phones [Return]..... 删除两个文件 REPORT 和 phones
$ rm xyz [Return]..... 删除 xyz;如果文件不存在,系统显示错误信息
rm: file not found
$_..... 等待用户命令
    
```

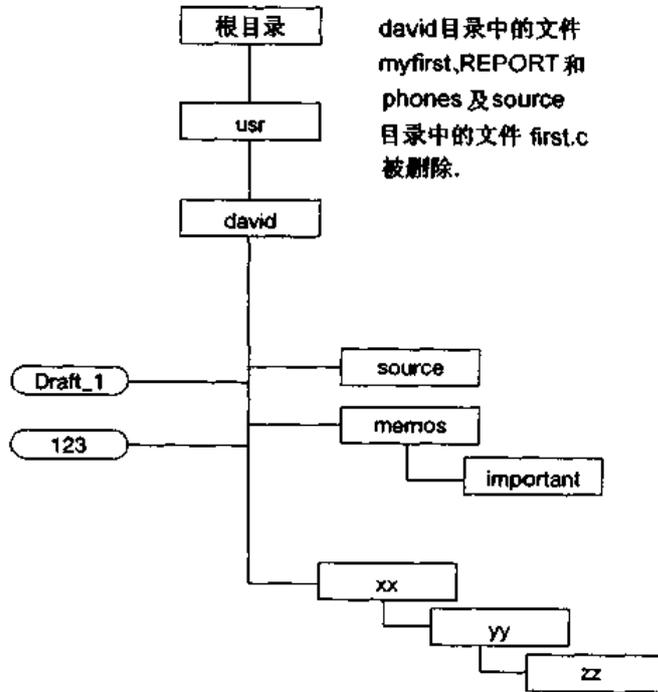


图 5.14 删除文件后的目录结构

【注意】

rm 命令不给用户任何警告信息,当删除文件时,不出问题的话,文件就被删掉了。

rm 选项

和多数 UNIX 命令一样,rm 选项以相反的方式更改 rm 命令的功能。表 5.4 总结了 rm 的选项。

表 5.4 rm 命令选项

选 项	功 能
-i	删除文件前,请求确认
-r	删除指定的目录及目录中的所有文件和子目录

-i 选项:进行删除操作时,该选项给用户更多控制权。如果用该选项,在删除每个文件之前,rm 提示用户确认。如果确实想要删除指定文件,按[y]确定;如果不想删除,按[n]。这是最安全的删除方法。

【实例】

下面的命令显示了使用 -i 选项的例子。

```

$ pwd [Return]..... 检查当前位置
    
```

```

/usr/david
$ ls source [Return] .....列出 source 目录的文件
first.c
$ rm -i first.c [Return].....删除 first.c;系统显示删除确认提示,按[y]确认
rm: remove first.c? y
$ ls source [Return].....检查文件是否被删
$_.....source 目录里没有文件了

```

-r 选项:删除目录中的所有文件和子目录。用 rm 带 -r 选项可以删除整个目录结构。这样的命令正是使得 UNIX 操作系统不断成长的原因所在。

【实例】

我们来看带 -r 选项的命令。图 5.15 显示了操作后的目录结构。这个例子里,星号(*)是一个通配符,表示所有文件。通配符将在第 7 章解释。

```

$ cd [Return].....回到用户主目录
$ rm -r * [Return].....删除 david 目录下的所有文件
$ ls [Return].....列出 david 目录的文件
$_.....david 下什么都没有了

```

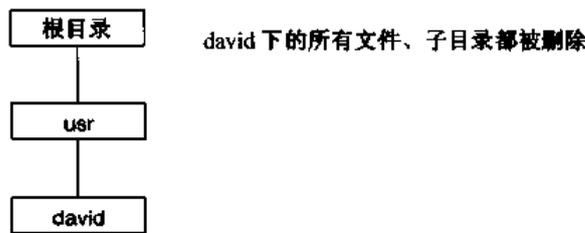


图 5.15 删除整个结构后的目录结构

【说明】

1. 用 -i 选项得到确认命令提示符。
2. 小心使用 -r 选项,只有认为绝对必要时才用。
3. rmdir 命令删除目录。

5.7.1 删除文件之前

UNIX 系统中,删除文件和目录确实很容易。但是,不像其他的操作系统,UNIX 不给用户反馈或者警告信息。用户知道之前,文件已经被删除了,并且删除无法恢复。因此,在输入 rm 之前,注意以下几点。

【注意】

1. 别在凌晨两点做重要的删除操作。
2. 确保知道删除的是哪个文件,文件的内容是什么。
3. 按回车键之前,请三思。

命令小结

本章介绍了如下 UNIX 命令。

pwd(print working directory, 打印工作目录)

这条命令显示用户当前所在目录的绝对路径名, 或者任何指定的目录。

cd(change directory, 改变目录)

这条命令把当前目录改到其他目录。

ls(list, 列表)

这条命令用来显示指定目录的内容。

选 项	功 能
-a	列出所有文件, 包括隐藏文件
-C	以多列的格式列表, 按列排序
-F	如果是目录, 文件名后加斜杠(/); 如果是可执行文件, 加星号(*)表示
-l	按照长格式列表, 显示文件的详细信息
-m	按页宽列文件, 以逗号分隔
-p	如果是目录, 文件名后加斜杠(/)
-r	按字母反序列表
-R	循环列出子目录的内容
-s	以文件块为单位显示文件大小
-x	以多列的格式列表, 按行排序

mkdir(make directory, 创建目录)

这条命令在工作目录或者其他任何命令中指定的目录下创建一个新的子目录。

选 项	功 能
-p	在一个命令行中创建多层目录

lp(line printer, 行打印)

这条命令打印指定的文件。

选 项	功 能
-d	在指定的打印机上打印
-m	完成打印请求后, 向用户邮箱发邮件
-n	打印指定份数
-s	取消反馈消息
-t	输出的标题页打印指定标题
-w	完成打印请求后, 在用户终端显示消息

rm(remove, 删除)

这条命令删除当前目录或者其他指定目录不再需要的文件。

选项	功能
-l	删除文件前, 请求确认
-r	删除指定的目录及目录中的所有文件和子目录

lpstat(line printer status, 行打印机状态)

这条命令可用来获得打印请求和打印机状态的信息。

cancel(cancel print requests, 取消打印请求)

这条命令用来取消不想要的打印请求。

习题

- 目录文件和普通文件的区别是什么?
- 文件名中, 可以使用/(斜杠)字符吗?
- 用目录组织文件的优点是什么?
- 相对和绝对路径名有什么区别?
- 把左列的命令和右列的解释匹配起来。

1. ls	a. 在屏幕上显示文件 xyz 的内容
2. pwd	b. 删除文件 xyz
3. cd	c. 删除文件前, 要求确认
4. mkdir xyz	d. 在默认打印机上打印文件 xyz
5. ls -l	e. 删除目录 xyz
6. cd ..	f. 取消打印机 lp1 的打印任务
7. ls -a	g. 显示默认打印机的状态
8. cat xyz	h. 列出当前目录的内容
9. lp xyz	i. 在当前目录创建 xyz 目录
10. rm xyz	j. 显示当前目录的路径名
11. rmdir xyz	k. 以长格式列出当前目录的文件
12. cancel lp1	l. 改变工作目录到当前目录的父目录
13. lpstat	m. 列出所有文件, 包括不可见的文件
14. rm -i	n. 改变当前目录到用户主目录
- 判断下面每一项是绝对路径名、相对路径名还是文件名?

a. REPORT	
b. /usr/david/temp	
c. david/temp	
d. ..(双点)	
e. my_first.c	
f. lists.01.07	

7. 进行如下操作的命令分别是什么?

- a. 删除文件
- b. 删除目录
- c. 删除前,得到确认信息
- d. 打印文件
- e. 取消打印请求
- f. 检查打印机状态
- g. 把打印工作重定向到另一台打印机
- h. 打印文档的多份副本
- i. 进行文件列表
- j. 列出包括隐藏文件在内的所有文件
- k. 以长格式列出文件
- l. 改变到用户自己的用户主目录
- m. 改变到其他目录
- n. 创建目录
- o. 创建一个二级目录结构
- p. 改变到根目录

上机练习

登录到系统中,练习下面的命令。观察输出和命令的反馈(错误信息等)。

在用户主目录创建一个目录结构,练习各种命令,直到用户熟练掌握目录和文件管理命令。

1. 显示当前目录。
2. 改变到用户主目录。
3. 确认用户主目录。
4. 列出当前目录的内容。
5. 当前目录下,创建名为 xyz 的新目录。
6. 在 xyz 目录创建名为 xyz 的文件。
7. 确认工作目录。
8. 显示当前目录内容。
 - a. 按照字母逆序
 - b. 按照长格式
 - c. 按照水平格式
 - d. 显示不可见的文件
9. 打印 xyz 目录中的 xyz 文件。
10. 检查打印机状态。
11. 取消打印请求。
12. 删除 xyz 目录的 xyz 文件。
13. 删除当前目录中的 xyz 目录。

第 6 章 vi 编辑器:高级使用

在第 4 章中讨论了 vi 编辑器,本章将继续讨论。本章描述了更多 vi 编辑器的功能和灵活性,介绍更多高级命令,并结合其他命令解释这些高级命令的范围和用途。此外,本章还将讨论 vi 编辑器对临时缓冲区的操作,以及几种根据用户需要定制 vi 编辑器的方法。学习完本章后,读者将能熟练使用 vi 完成自己的编辑工作。

6.1 更多有关 vi 编辑器的知识

vi 编辑器是编辑器中 ex 家族的一部分。vi 是 ex 中的屏幕编辑器,并可以随时在 vi 与 ex 间切换。实际上,以:开始的 vi 命令就是 ex 编辑器命令。在 vi 命令模式下,按:在屏幕底部显示冒号提示符,并使 vi 进入等待命令状态。当键入命令完成时,按回车键,ex 执行命令并在完成后返回 vi。

要想切换到 ex 编辑器,只需键入:Q 并按回车键。如果用户有意或无意地切换到了 ex 编辑器,键入 vi 即可返回 vi 编辑器,或键入 q 来退出 ex 编辑器,回到 shell 提示符状态。

6.1.1 调用 vi 编辑器

在第 4 章,用户学习了如何启动 vi,如何保存文件以及如何退出 vi。现在扩展那些命令,我们来探索启动和结束 vi 编辑器的其他方法。

可以不提供文件名而启动 vi 编辑器。这种情况下,使用写(:w)或者写退出(:wq)命令来命名文件。

【实例】

下面的命令序列显示了怎么去做:

- 键入 vi,然后按回车键,激活不带文件名的 vi 编辑器。
- 键入:w myfirst 然后按回车键,把临时缓冲区的内容保存到 myfirst 文件,仍然停留在 vi 编辑器中。
- 键入:wq myfirst 然后按回车键,把临时缓冲区的内容保存到 myfirst 文件,并退出 vi 编辑器。

如果当前编辑的文件尚未起名,键入:w 或:wq 后又不输入文件名,则 vi 显示如下信息:

```
No current filename
```

vi 编辑器通常防止覆盖一个已存在的文件。因此,如果用户键入:w myfirst 并按回车键,而 myfirst 文件已存在时,vi 会显示如下信息提出警告:

```
"myfirst" File exists - use ":w!" to overwrite"
```

如果想覆盖已存在的文件,使用:w! 命令。

写命令(:w)也可以用于保持当前文件内容完整不变,而将当前文件的全部或部分存入另一个文件名中。如下命令显示给一个文件命名或给当前编辑的文件改名的方法:

- 键入 **vi myfirst** 然后按回车键,调用 vi 编辑器并将 myfirst 的内容调入到临时缓冲区内。
- 键入:**w yourfirst** 然后按回车键,将当前临时缓冲区的内容(myfirst)写入文件 yourfirst 中。用户当前编辑的文件仍为 myfirst。屏幕将显示如下信息:

```
"yourfirst" [New file] 3 lines, 106 characters
```

- 键入:**wq yourfirst** 然后按回车键,保存临时缓冲区的内容到文件 yourfirst 中,然后退出 vi 编辑器。原始文件 myfirst 的内容不变。

6.1.2 使用 vi 调用选项

vi 编辑器从一开始就提供了极大的灵活性。用户在调用 vi 编辑器时,可以在命令行键入参数来调用特定的选项。

只读选项: -R 选项(只读)使一个文件只读并允许用户逐行查看文件内容时不会意外修改文件。对 myfirst 文件使用该选项只需键入 **vi -R myfirst** 然后按回车键。vi 编辑器将在屏幕底端显示如下信息:

```
"myfirst" [Read Only] 3 lines, 106 characters
```

如果用户试图用:w 或:wq 命令来保存只读文件,vi 将显示如下信息:

```
"myfirst" File is read only
```

命令选项: -c(命令)选项允许用户在命令行指定 vi 命令作为参数。该选项对于用户开始编辑文件时先进行光标定位或查找特定的样式非常有用。对文件 myfirst 使用该选项,键入 **vi -c/most myfirst** 然后按回车键。可以给出一条搜索命令(/most)作为命令行的一部分。vi 将文件 myfirst 复制到临时缓冲区中,然后将光标定位在第一次找到单词 most 的位置。

6.1.3 编辑多文档

用户可以打开 vi 编辑器并给它一系列文件,而不仅仅是一个文件。这时,当完成一个文件编辑时,可以继续编辑下一个文件而不需要重新调用 vi。可以键入:**n(下一个)**来调出下一个要编辑的文件。当用户发出:n 命令时,vi 用下一个文件的内容代替当前工作缓冲区的内容。但是,如果已经修改了当前工作缓冲区中的内容,vi 将显示如下信息:

```
No write since last change (:next ! overrides)
```

可以用 **n!** 命令来忽略该提示信息。这时,对当前文件的修改将会丢失。

如果想查看打开的文件名序列,使用:**ar** 命令。vi 将显示打开的文件名列表,并指出目前正在编辑的文件。

【实例】

下面命令显示在一个命令行中指定多个文件的例子。

- 键入 **vi file1 file2** 然后按回车键,调用 vi 打开两个文件:file1 和 file2;vi 响应:

```
2 files to edit
"file1" 10 lines, 410 characters
```

- 键入:w 然后按回车键,保存 file1。
- 键入:ar 然后按回车键,显示打开的文件名。vi 列出打开的文件,并将正在编辑的文件用括号括起来:

```
[File1] file2
```

- 键入:n 并按回车键,开始编辑 file2;vi 响应:

```
"file2" 100 lines, 700 characters
```

编辑另一个文件:编辑多文件的另一种方法是使用:e(编辑)命令来打开一个新文件。在 vi 编辑器中,键入:e 后面紧跟要编辑的文件名,然后按回车键。通常,用户在导入一个新文件前应保存正在编辑的文件,除非没做任何修改,否则 vi 将提示用户在切换到下一个文件前保存当前文件。

键入如下命令来体验修改文件。

- 键入 vi 然后按回车键,调用 vi 编辑器而不指定任何文件名。
- 键入:e myfirst 然后按回车键,调用文件 myfirst。用户当前编辑的文件是 myfirst;vi 将显示 myfirst 的文件名和长度:

```
"myfirst" 3 lines, 106 characters
```

读取另一个文件:vi 允许用户将另一个文件导入到用户当前编辑的文件中。在 vi 编辑器命令状态下,键入:r 后跟文件名,然后按回车键。:r 命令将指定文件的副本放到缓冲区中用户光标后面的位置。指定文件成为用户正在编辑文件的一部分。

【实例】

使用如下命令导入(读)另一个文件到用户当前工作缓冲区。

- 键入 vi myfirst 然后按回车键,调用 vi 编辑器编辑文件 myfirst。
- 键入:r yourfirst 然后按回车键,将 yourfirst 的内容加入到当前编辑的文件中。vi 显示导入文件的名称和长度:

```
"yourfirst" 10 lines, 212 characters
```

【说明】

:r 命令只将指定文件的副本加入到当前编辑文件中,指定文件的内容保持不变。

写入另一个文件:vi 编辑器允许用户将当前编辑文件的一部分写(保存)到另一个文件中。用户只需指定要写的行的范围,使用:w 命令完成写操作。例如,如果想把第 5 行到第 100 行之间的文本写到文件 temp 中,只需键入:5,100 w temp,然后按回车键即可。vi 将把第 5 到第 100 行存入文件 temp 中,并显示与如下相似的信息:

```
"temp" [new file] 96 lines, 670 characters
```

如果要写的文件已经存在,则 vi 显示错误信息。用户可以另行指定一个新文件或使

用:w!命令覆盖已有文件。

6.2 重排文本

删除、复制、移动和修改文件全部被认为是剪切和粘贴操作。表 6.1 总结了在文件中组合使用剪切和粘贴进行操作中用到的操作符。所有这些命令都用于 vi 的命令状态中。除了修改命令外,对所有其他命令,vi 都是在完成命令后返回命令状态的。修改命令使 vi 进入文本编辑状态,这时,用户可以使用[Esc]键回到命令状态下。

表 6.1 vi 编辑器剪切和粘贴键

键	功 能
d	删除指定位置的文本,并将这些文本存放在临时缓冲区中。该临时缓冲区可用 put 操作符访问
y	将指定位置的文本复制到一个临时缓冲区中。该临时缓冲区可用 put 操作符访问
P	将指定缓冲区中的内容放到当前光标位置之上
p	将指定缓冲区中的内容放到当前光标位置之下
c	删除文本并将 vi 置为文本编辑状态。该命令是删除和插入操作的结合

假设用户当前屏幕显示为文件 myfirst,光标位于 m 上,如下命令显示剪切和粘贴应用。

6.2.1 移动行:dd、p 或 P

【实例】

使用 delete 和 put 操作符,用户可以将文本从文件的一个位置移动到另一个位置。

- 键入 dd,vi 删除当前行,并将删除的行保存在临时缓冲区中,然后光标移到 U 上。
- 键入 p,vi 将前面删除的行放置在当前行下面。

```
The vi history
The vi editor is a text editor which is supported by most of the
UNIX operating systems.
```

```
The vi history
UUNIX operating systems.
```

```
The vi history
UNIX operating
UThe editor is a text editor which is supported by most of the
```

- 使用光标移动键,将光标移到第一行中的任何字母上。
- 键入 P,vi 将删除行放置在当前行上。

```
The vi editor is a text editor which is supported by most of the
The vi history
UNIX operating systems.
The vi editor is a text editor which is supported by most of the
```

【说明】

被删除的文本仍留在临时缓冲区中,用户可以将它移动到本文件中的任何地方。

6.2.2 复制行:yy、p 或 P

使用 `yank` 和 `put` 操作符,可以将文件的一部分文本从一个位置复制到另一个位置。

- 键入 `yy`,vi 将当前行复制到临时缓冲区中。
- 使用光标移动键,将光标移至第一行。
- 键入 `p`,vi 将临时缓冲区的内容复制到当前行的下面。

```
The vi history
The vi editor is a text editor which is supported by most of the
Unix operating systems.
```

```
The vi history
The vi editor is a text editor which is supported by most of the
The vi editor is a text editor which is supported by most of the
Unix operating systems.
```

- 使用光标移动键将光标移至最后一行。
- 键入 `P`,vi 将临时缓冲区的内容复制到当前行的上面。

```
The vi history
The vi editor is a text editor which is supported by most of the
The vi editor is a text editor which is supported by most of the
The vi editor is a text editor which is supported by most of the
UNIX operating systems.
```

【说明】

被复制的文本仍然保留在临时缓冲区中,直到下一个删除或复制操作。用户可以将临时缓冲区中的内容复制任意次到当前文件中的任何位置。

6.3 vi 操作符的域

在第 4 章中,读者学习了基本 vi 命令;但是,许多 vi 命令对文本块进行操作。一块文本可以是一个字符、一个单词、一行、一个句子或其他一些特定的字符集。使用 vi 命令结合域控制键使用户在进行编辑工作时具有更多的控制能力。这种类型命令的格式为:

命令 = 操作符 + 域

对完整的一行没有特定的域控制键。要想指定一行作为命令的域,只需要将操作符键入 2 次。例如, `dd` 将删除一行,而 `yy` 将拾取一行。表 6.2 总结了一些常用的域控制键。

表 6.2 部分 vi 域控制键

域	功 能
\$	标识域为从光标所在位置到该行的行尾
0(零)	标识域为从光标前一个位置到该行所在行首
e	标识域为从光标所在位置到光标所在单词的末尾
b	标识域为从光标前一个字母到光标所在单词的开始

下面例子示范了带域控制命令的使用。在练习这些例子的开始,打开 `myfirst` 在屏幕上。使用删除、拾取和修改命令操作该文件,使用户对这些命令有实际的、直观的认识。

6.3.1 使用带域控制键的删除操作符

【实例】

删除从当前光标位置到光标所在行的行尾的文本。

□ 键入 `d$`;vi 删除从光标位置开始到当前行末尾的文本,并将光标移至单词 `by` 后。

```
The vi history
vi is a text editor which is supported by most of the
UNIX operating systems.
```

【说明】

可以使用 `u` 或 `U` 来撤消对文本最后的修改。

```
The vi history
vi is a text editor which is supported by _
UNIX operating systems.
```

【实例】

删除从当前光标之前的位置到当前行行首之间的文字,键入 `d0`;vi 删除相应文字并且光标仍在 `m` 上。

```
The vi history
vi is a text editor which is supported by most of the
UNIX operating systems.
```

```
The vi history
most of the
UNIX operating systems.
```

【实例】

删除当前光标后的一个字,键入 `dw`;vi 删除字 `most` 及其后面的空格,将光标移至字母 `o` 上。

```
The vi history
vi is a text editor which is supported by most of the
UNIX operating systems.
```

```
The vi history
```

```
vi is a text editor which is supported by of the
UNIX operating systems.
```

【实例】

删除光标后面多个字(如3个字),键入 **3dw**;vi 删除三个字:most、of 和 the 以及后面的空格,并将光标移至字 by 之后。

```
The vi history
vi is a text editor which is supported by most of the
UNIX operating systems.
```

```
The vi history
vi is a text editor which is supported by _
UNIX operating systems.
```

【实例】

删除一个字的尾部,键入 **de**;vi 删除单词 most,并将光标移到字母 o 前面的空格。

```
The vi history
vi is a text editor which is supported by most of the
UNIX operating systems.
```

```
The vi history
vi is a text editor which is supported by _of the
UNIX operating systems.
```

【实例】

删除前一个单词的前部,键入 **db**;vi 删除单词 by,光标仍保持在字母 o 前的空格上。

```
The vi history
vi is a text editor which is supported _of the
UNIX operating systems.
```

6.3.2 使用带域控制键的拾取操作符

拾取操作符可使用与删除操作符相同的域。**p** 操作符用于将要拾取的文本放到文件中的其他位置,并通过域控制键来显示希望拾取文本的部分。

【实例】

要复制从光标位置到当前行末之间的文本,进行如下操作。

- 键入 **y\$**;vi 将从光标开始到该行末尾之间的文本复制到临时缓冲区中。
- 使用光标移动键,将光标移至最后一行的末尾。
- 键入 **p**;vi 将要拾取的文本复制到光标后面。

```
The vi history
vi is a text editor which is supported by most of the
UNIX operating systems.
```

```
The vi history
vi is a text editor which is supported by most of the
```

UNIX operating systems most of the

【实例】

要复制从光标位置到该起始之间的文本,进行如下操作。

- 键入 **y0**;vi 将从光标到当前行首之间的文本复制到临时缓冲区中,而光标仍保持在字母 m 处。
- 使用光标移动键,将光标移至第一行的字母 v 上。
- 键入 **p**;vi 将拾取的文本从临时缓冲区中复制到光标位置后面。

```
The vi history
vi is a text editor which is supported by most of the
UNIX operating systems.
```

```
The vi is a text editor which is supported by _vi history
vi is a text editor which is supported by most of the
UNIX operating systems.
```

6.3.3 使用带域控制键的修改操作符

修改操作符 **c**,可以使用与删除和拾取操作符同样的域。**c**操作符功能与其他操作符不同之处在于:它将 vi 由命令状态改变为文本编辑状态。在键入 **c**后,用户可以在光标所在位置处键入文本,文本向右移动。如果需要为键入的文本创造空间,文本会自动折行。通过按 [Esc]键返回 vi 命令状态。修改操作符删除由命令的域控制键指定的文本。

有的版本的 vi 编辑器用一个标记来标记最后要被删除的字母。该标记通常是一个美元符号(**\$**),它将覆盖要被删除的最后一个字母。

【实例】

下面的例子显示如何使用带域控制键的修改操作符来修改一个单词。

- 键入 **cw**;vi 在当前单词的尾部放置一个标记,并覆盖字母 t,然后切换到文本编辑状态。光标停留在将要修改的第一个字母 m 上。
- 键入 **all** 来将单词 most 改为 all。
- 按 [Esc]键返回到 vi 命令状态。

```
The vi history
vi is a text editor which is supported by most of the
UNIX operating systems.
```

```
The vi history
vi is a text editor which is supported by mos $ of the
UNIX operating systems.
```

```
The vi history
vi is a text editor which is supported by all $ of the
UNIX operating systems.
```

6.4 在 vi 中使用缓冲区

vi 编辑器有多个用于临时存储的缓冲区。保存用户文件副本的临时缓冲区(工作缓冲区)已在第 3 章中进行了讨论。当使用写命令时,该缓冲区的内容被复制到一个临时文件中。有两类临时缓冲区:数字编号缓冲区和命名缓冲区(字母编号缓冲区),可供用户用于保存更改以及以后再找回变更。

6.4.1 数字编号缓冲区

vi 编辑器使用编号从 1 到 9 的 9 个临时缓冲区。每次删除或拾取文本时,这些文本被放进这些临时缓冲区中。用户可以通过指定缓冲区号在任何时间访问这些缓冲区。每个新删除或拾取的文本将替换临时缓冲区中的前一个内容。例如,当用户键入 **dd** 命令时,vi 将删除的行放入缓冲区 1。当用户再次使用 **dd** 命令删除另一行时,vi 将缓冲区中的内容向后移动一个缓冲区,在这个例子中,原来缓冲区 1 中的内容被移到缓冲区 2,而新删除的内容被送到缓冲区 1 中。因此,缓冲区 1 中始终保存最近改变的内容。数字编号缓冲区中的内容随着用户每产生一个删除或拾取动作而改变一次。当然,在几个改变之后,用户会忘记每个缓冲区中保存的内容是什么,但是,继续读下去,最好的部分即将到来。

数字编号缓冲区的内容可以使用前面带缓冲区编号的放置操作符来找回。例如,要找回第 9 号缓冲区中的内容,只需键入 **"9p**。命令 **"9p** 将第 9 号缓冲区的内容复制到光标所在位置。指定缓冲区的格式可以表示为:

1 个双引号 + 1n(n 是缓冲区号,在 1 到 9 之间) + (**p** 或 **P**)

有关数字编号的临时缓冲区见图 6.1 到 6.5 的描述。该例子解释 vi 编辑器在用户修改文件中的文本时的事件序列。

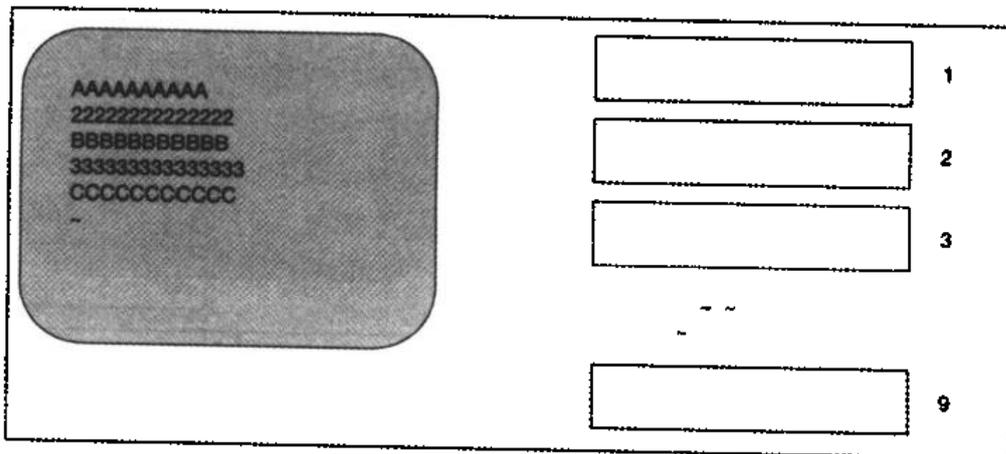


图 6.1 vi 编辑器的 9 个编号缓冲区

【实例】

假设用户当前的屏幕显示如图 6.1 所示,有 5 行文本,并且数字编号缓冲区为空,因为用户尚未进行任何编辑操作。

- 将光标移至第 1 行,使用删除命令删除当前行。vi 编辑器将删除的行保存在缓冲区 1 中,这时,屏幕显示和缓冲区如图 6.2 所示。

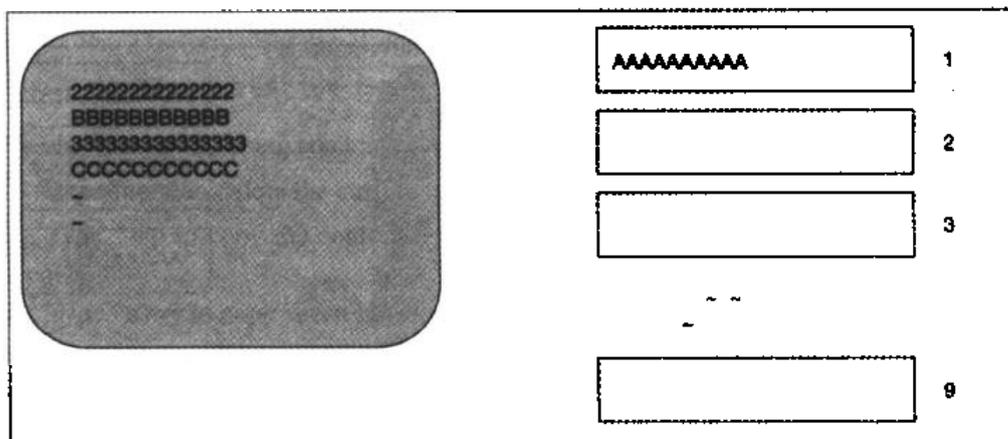


图 6.2 第一次删除后屏幕显示和缓冲区情况

- 用删除命令一次删除 2 行。vi 编辑器响应为:从现有文本中删除 2 行并将临时缓冲区中的内容向后移动一个缓冲区以将缓冲区 1 置空。接着,它将删除的行保存在缓冲区 1 中。这时,屏幕显示和缓冲区情况如图 6.3 所示。

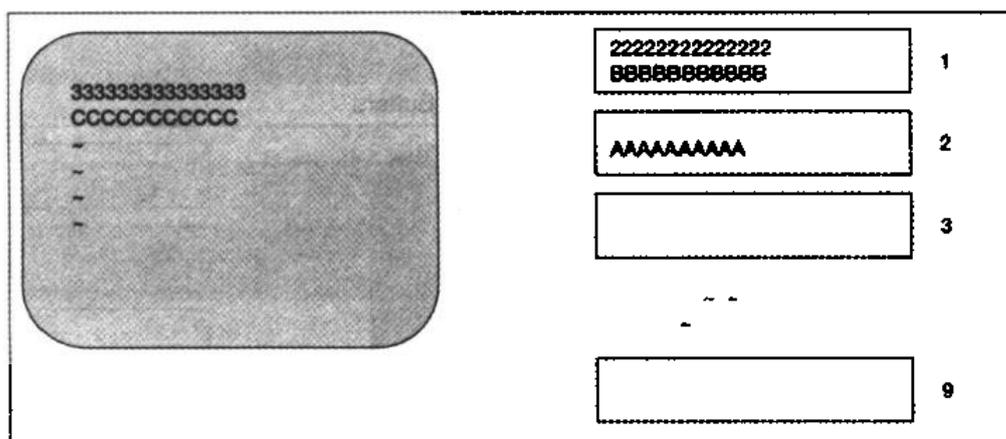


图 6.3 第二次删除后屏幕显示和缓冲区情况

【说明】

1. 被删除的 2 行保存在同一个缓冲区中。数字编号缓冲区可以保存用户改变的任意大小的文本,而不仅仅是一行。任意数量的文本,无论是 1 行还是 100 行都可以保存在 1 个缓冲区中。

2. 当所有 9 个缓冲区都满了时,如果这时 vi 需要向缓冲区 1 写入新内容,缓冲区 9 的内容将丢失。

- 用户拾取的文本也保存在这些临时缓冲区中。现在,将光标放到第 1 行,然后键入 yy 并按回车键。vi 将用户拾取的行复制到缓冲区 1 中。记住,vi 必须将所有缓冲区中的内容依次后移一个缓冲区来空出缓冲区 1,接着,新的内容被放到缓冲区 1 中。见图 6.4。

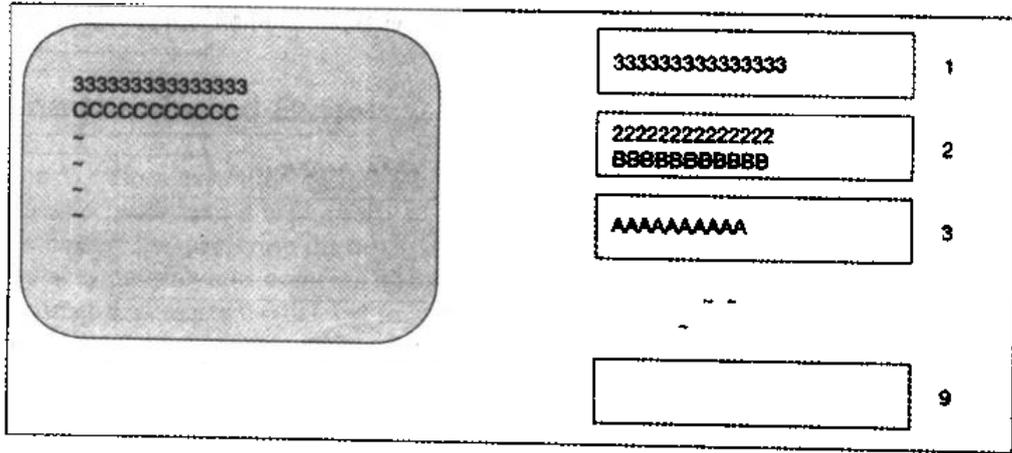


图 6.4 拾取后屏幕显示和缓冲区情况

□ 现在,如果用户还记得在各个缓冲区中的内容,就可以通过在命令中指定缓冲区号来访问任何缓冲区。例如,要将缓冲区 2 中的内容复制到文件末尾,键入"2p,vi 将缓冲区 2 中的内容复制到光标所在位置。图 6.5 显示放置命令后的屏幕显示及缓冲区情况。

【说明】

访问缓冲区不会改变它们的内容。

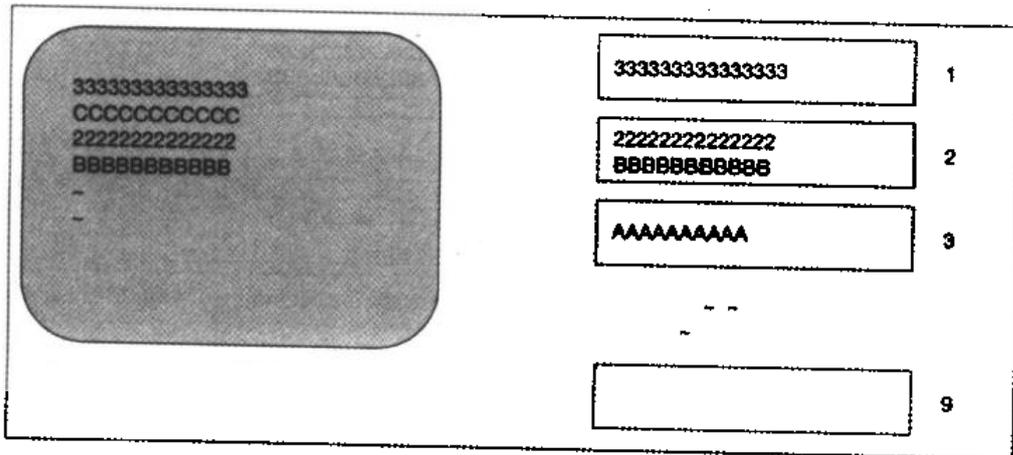


图 6.5 放置命令后的屏幕显示和缓冲区情况

6.4.2 字母序缓冲区

vi 编辑器还使用 26 个命名缓冲区。这些缓冲区以从 a 到 z 的小写字母命名,用户可以通过显式地指定它们的用户名来调用。这些缓冲区与数字编号缓冲区相似,不同之处在于,vi 并不在用户每次删除或拾取时自动改变它们的内容。这使得用户在操作时拥有更多控制权。用户可以将删除或复制的文本存入一个指定的缓冲区,并在以后使用放置命令从命名缓冲区中复制文本到文件的其他位置。而在字母编号缓冲区中的内容将保持不变,直到用户在删除或拾取操作指定相应的缓冲区。在命令中指定缓冲区的格式如下:

1 个双引号 + 缓冲区名(从 a 到 z 的小写字母) + 命令

【实例】

进行如下操作,体验使用命令操作指定缓冲区。

- 键入"wd",删除当前行,并在 w 缓冲区中保存一个副本。
- 键入"wp",将 w 缓冲区中的内容复制到光标所在位置。
- 键入"z7yy",复制 7 行到 z 缓冲区。
- 键入"zp",复制 z 缓冲区(7 行)的内容到光标所在位置。

【注意】

1. 这些命令不会显示在屏幕上。
2. 字母编号缓冲区以小写字母从 a 到 z 命名。
3. 这些命令不需要按回车键。

6.5 光标定位键

屏幕一次显示 24 行,如果用户文本多于 24 行,用户可以使用光标移动键向上或向下滚动。如果文件有 1000 行,则需要大约 999 次击键才看到第 999 行。这是很麻烦而且不实用的。要克服这样的问题,用户可以使用 vi 编辑器的翻页操作。表 6.3 总结了翻页操作符和它们的功能。

表 6.3 vi 的翻页键

键	功 能
[Ctrl - d]	使光标向文件尾移动,通常一次 12 行
[Ctrl - u]	使光标向文件头移动,通常一次 12 行
[Ctrl - f]	使光标向文件尾移动,通常一次 24 行
[Ctrl - b]	使光标向文件头移动,通常一次 24 行

【注意】

[Ctrl - d] 的意思是同时按下 Ctrl 键和 d 键。这种组合也适用于其他控制键。

如果用户文件确实很大,即使用翻页键也不能很好地移动光标,还有另一种定位方法是使用 G 前紧跟要去的行号。

【实例】

要使第 1000 行成为当前行,进行如下操作。

- 键入 1000G,使光标移至第 1000 行。
- 键入 1G,将光标移到第 1 行。
- 键入 G,将光标移动到文件末尾。

另一种有用的命令是[Ctrl - g],该命令报告当前行的行号。例如,如果用户在命令状态下按[Ctrl - g],vi 编辑器响应显示类似如下信息:

```
"myfirst" line 30 of 90 - 30%
```

6.6 定制 vi 编辑器

vi 编辑器有很多参数(也叫选项或标记)可供用户设置,以控制自己的工作环境。这些参数拥有默认值,并且是可调整的,包括制表符设定、右边缘设定等。

要想查看参数的完整列表和它们的当前设定,进入命令状态并键入:`set all` 然后按回车键。

读者的终端显示应当与图 6.6 类似,但可能会有自己的选项设置。

```

~
~
~
:set all          noshowmode
noautoindent     number          terms5wyse50
nobeautify       readonly       noterse
hardtabs = 8     report = 10       window = 23
noignorecase     shiftwidth = 8     wrapmargin = 0
magic
[Hit return to continue]

```

图 6.6 屏幕显示的 vi 选项

6.6.1 选项格式

`set` 命令用来设置选项。通常选项的格式分为三类,设置方式各不相同:

- 布尔(触发器)
- 数字式
- 串

假设有一个选项名为 X,下面例子显示如何设置上述三种选项。

布尔选项

布尔选项如同一个触发开关:用户可以将它们打开或关上。这种选项通过键入选项名进行设定,通过在选项名前加 `no` 来取消设定。键入 `set X` 设定选项 X,键入 `set noX` 禁止该选项。

【注意】

在 `no` 与选项名之间没有空格。

数字式选项

数字式选项接收一个数字值,根据选项的不同,该数字值的范围也不同。键入 `set X = 12`,将值 12 赋给选项 X。

串选项

串选项与数字选项相似,只接受串值。键入 `set X = PP`,将串 PP 赋给选项 X。

【注意】

在等号两侧都没有空格。

set 命令

`set` 命令用来设置不同的 vi 环境选项、显示这些选项或获得某指定选项的值。`set` 命令的基本格式如下所示,每个命令以按回车键结束:

```
:set all
```

在屏幕上显示所有选项。

```
:set
```

只显示修改过的选项。

```
:set X?
```

显示选项 X 的值。

6.6.2 设置 vi 环境

vi 编辑器的行为可以通过设置编辑参数来自定义,并且有多种方法进行这种设置。最直接的方法是使用 vi 的 `set` 命令进行设置。这种情况下,vi 在进行设置前必须处于命令状态。使用这种方法用户可以设置任何选项,但是,选项的改变是临时的,并且只在用户当前编辑会话下有效。当用户退出 vi 编辑器时,设置会被丢弃。

本小节介绍一些有用的 vi 参数,表 6.4 汇总了这些(按字母顺序列出)。大多数选项名有缩写形式,用户进行设置时既可使用选项名的全称,也可使用缩写。

表 6.4 部分 vi 环境选项

选 项	缩写	功 能
<code>autoindent</code>	<code>ai</code>	将新行与前一行的开始对准
<code>ignorecase</code>	<code>ic</code>	在搜索选项下,忽略大小写
<code>magic</code>	-	在搜索时,允许使用特殊字符
<code>number</code>	<code>nu</code>	显示行号
<code>report</code>	-	告知用户,用户最后一个命令作用行的行号
<code>scroll</code>	-	设定使用[Ctrl-d]命令翻滚的行数
<code>shiftwidth</code>	<code>sw</code>	设定缩进空格数。与 <code>autoindent</code> 一同使用
<code>showmode</code>	<code>smd</code>	在屏幕右角显示 vi 编辑器模式
<code>terse</code>	-	缩短错误信息
<code>wrapmargin</code>	<code>wm</code>	将右边界设为一定的字符个数

autoindent 选项: autoindent 选项(ai)将用户键入的每个新行与前一行的开始对齐。该选项对于使用 C、Ada 或其他结构化程序设计语言编写程序时十分有用。使用[Ctrl-d]减少一级缩进。每个[Ctrl-d]增加一个由 shiftwidth 选项指定的数值。本选项的默认值为 noai。

ignorecase 选项: vi 编辑器提供大小写敏感搜索,也就是说它区分大写字母与小写字母。要使 vi 忽略大小写,键入: set ignorecase 并按回车键。要返回大小写敏感状态,键入: set noignorecase 并按回车键。

magic 选项: 某些符号(如方括号[])在用于搜索中有特殊含义。当用户将该开头置为 nomagic 时,这些符号不再有特殊含义。该选项默认值为 magic。

number 选项: vi 编辑器一般情况下不显示每行的行号。但是,有时用户希望显示行号,如想通过行号指定某行,或者,显示行号可以使用户对自己文件的大小及自己正在编辑文件的哪一部分等心里有数。要显示行号,键入: set number, 然后按回车键。

如果不希望显示行号,键入: set nonumber 并按回车键。

【说明】

行号并不是文件的一部分;它们只有在 vi 编辑器中才会显示。

report 选项: vi 编辑器对用户的编辑工作并不给予任何反馈。例如,如果用户键入 = dd, vi 删除从当前行开始的 5 行文本,但不会在屏幕上显示任何确认信息。如果希望在屏幕上看到自己编辑的反馈信息,用户可以使用 report 选项来实现。该参数被设为使 vi 编辑器报告发生变化的行的最小行数。

要将 report 选项设为改变 2 行时有效,键入: set report = 2 并按回车键。于是,当用户的编辑工作作用 2 行时,vi 显示相应报告。例如,删除 2 行(2dd)并复制 2 行(2yy)将在屏幕底部产生类似下面的报告信息:

```
2 lines deleted
2 lines yanked
```

scroll 选项: scroll 选项设定用户在使用[Ctrl-d]时希望滚动的行数。例如,要想使屏幕滚动 5 行,键入: set scroll = 5 并按回车键。

shiftwidth 选项: shiftwidth(sw)选项设定在设置了自动缩进时,使用[Ctrl-d]键时的空格数。该选项的默认设置为 sw = 8。要把该设置改为 10,键入: set sw = 10 并按回车键。

showmode 选项: vi 编辑器并不显示任何可见的反馈信息来告知当前是处于文本输入模式还是命令模式。这可能会导致混淆,尤其是对新手。用户可以设置 showmode 选项来提供可见的反馈到屏幕。

要打开 showmode 选项,键入: set showmode 并按回车键。接着,根据用户需要在文本输入模式和命令模式之间切换,而 vi 在屏幕的右下角显示不同的信息。如果用户键入 A 或 a 切换到修改模式,vi 显示 APPEND MODE;如果用户按 I 或 i,vi 将显示 INSERT MODE;如果用户按 O 或 o,vi 显示 OPEN MODE 等等。

这些信息将一直显示在屏幕上,直到用户按[Esc]键切换到命令模式。当屏幕上没有信息时,vi 处于命令模式。

要关闭 showmode 选项,键入: set noshowmode 并按回车键。

terse 选项: terse 选项使 vi 编辑器显示短错误信息。该选项默认值为 noterse。

6.6.3 行长度和行回绕

用户的终端屏幕通常为 80 行。当键入到行的末尾时(超过第 80 列),屏幕即开始一个新行,这就是所说的行回绕。在用户按回车键时,屏幕同样开始一个新行。因此,屏幕上一行的长度可以为 1 到 80 个字符之间的任何长度。但是,vi 编辑器只在用户按回车键时,才在用户文件中生成一个新行。如果用户在按回车键前键入了 120 个字符,这时键入的文本看起来是在 2 行,但实际在文件中,这 120 个字符只在 1 行中。

过长的行在文件打印时可能会出问题,并且屏幕显示的行号与实际文件中的行号相对时容易产生混淆。最简单的限制行长度的方法是在到达屏幕行末尾前按回车键。另一种方法是设定 `wrapmargin` 参数以使 vi 编辑器自动插入回车。

wrapmargin 选项: `wrapmargin` 选项是当用户输入的文本到达距右边界指定的字符时,vi 编辑器会断开用户正在输入的文本。要将 `wrapmargin` 设为 10(10 是从屏幕右边界计数的字符个数),键入:`set wrapmargin = 10` 并按回车键。于是,当用户键入到第 70 列时(80 减 10),vi 编辑器开始一个新行,如同用户自己按回车键一样。如果用户正在键入一个字时超过第 70 列,vi 将把该字整个移至新行。这也意味着右边界可能会不齐。但是要记住,vi 编辑器并不是一个文本格式化工具或字处理器。

`wrapmargin` 选项的默认值是 0(零)。要关闭该选项,键入:`set wrapmargin = 0` 并按回车键。

6.6.4 缩写与宏

vi 编辑器为用户提供一些捷径,以使用户的输入更快速、更简单。:`ab` 和 :`map` 是 2 个用于该目的的命令。

缩写操作符: `ab`(缩写)命令使得用户给任何字符串指定缩写。该功能可以帮助用户提高输入速度。用户可以为经常键入的文本选择一个易记的缩写,在 vi 编辑器中设置缩写后,就可使用该缩写代替原来的文本。例如,要缩写本书中常用的文本 UNIX Operating System,键入:`ab uno UNIX Operating System` 并按回车键。

在这个例子中,uno 是赋给 UNIX Operating System 的缩写;因此,当 vi 处于文本输入模式时,任何时间用户键入 uno 接着键入一个空格时,vi 将 uno 变为 UNIX Operating System。如果 uno 是另一个字的一部分,如 unofficial,并不会发生改变。vi 通过 uno 前后的空格来识别出 uno 是一个缩写,并把它扩展。

要取消一个缩写,用户可以使用 `unab`(未缩写)操作符。例如,要取消 uno 缩写,键入:`unab uno` 并按回车键即可。

要想列出已经设置了哪些缩写,键入:`ab` 并按回车键。

【说明】

1. 缩写在 vi 编辑器命令模式下进行指定,并用于 vi 的文本输入模式下。
2. 缩写的指定是临时的,它们只在当前编辑会话中有效。

【实例】

指定缩写。

- 键入:`ab ex extraordinary adventure` 并按回车键来将 ex 指定为串 extraordinary adventure 的

缩写。

- 键入：**ab 123 one, two, three, etc.** 并按回车键，将 123 指定为串 one, two, three, etc. 缩写。
- 键入：**ab** 并按回车键，显示所有指定的缩写：

```
ex   extraordinary adventure
123  one, two, three, etc.
```

- 键入：**unab 123** 并按回车键，取消 123 缩写。

宏操作符：宏操作符(**map**)使用户能将一系列键指定给某一个键。如同缩写操作符给用户一个文本输入模式下的捷径一样，**map** 给用户一个在命令模式下的捷径。例如，将命令 **5dd** (删除 5 行)指定为 **q**，键入：**map q 5dd** 并按回车键。此后，当 **vi** 处于命令模式时，每当用户键入 **q** 时，**vi** 应删除 5 行。

要取消一个 **map** 指定，用户可以使用 **unmap** 操作符。键入：**unmap q** 并按回车键。

要查看已经 **map** 键的列表和它们指定的内容，键入：**map** 并按回车键。

vi 编辑器将绝大多数键盘按键用作命令使用。供用户可用的键只有少数几个，包括 **K**、**q**、**V**、**[Ctrl-e]** 和 **[Ctrl-x]**。

用户也可使用 **map** 命令为自己的终端指定功能键。在这种情况下，用户键入 **#n** 作为键名，**n** 代表功能键号。例如，要指定 **5dd** 到 **[F2]**，键入：**map #2 5dd** 并按回车键。

此后，如果用户在 **vi** 的命令模式下按 **[F2]** 键，**vi** 将删除 5 行文本。

【实例】

下面例子显示部分键指定。

- 键入：**map V /unix** 并按回车键，将 **V** 键指定为搜索 **unix** 的搜索命令。
- 键入：**map #3 yy** 并按回车键，将 **[F3]** 指定为拾取一行。
- 键入：**map** 并按回车键，显示已经指定的键。

```
V   /unix
#3  yy
```

【实例】

假设用户希望在文件中查找 **unix**，并将它替换为 **UNIX**。进行下面操作。

- 键入：**/unix** 并按回车键，查找单词 **unix**。
- 键入 **cwUNIX**，然后按 **[Esc]** 键，将 **unix** 改为 **UNIX** 并返回 **vi** 命令模式。

在映射键指定中，命令行中，用户按 **[Ctrl-v][Return]** 来代表回车，用 **[Ctrl-v][Esc]** 代表 **[Esc]** 键。这样，要映射前面所说的命令到一个键中，比如说 **v** 键，键入：**map v /unix**，接着按 **[Ctrl-v][Return]**，然后键入 **cwUNIX**，再按 **[Ctrl-v][Esc]**。该命令行中使用了不可打印字符 (**[Ctrl-v]** 和 **[Esc]**)，所以用户看到的屏幕如下所示：

```
:map v /unix ^McwUNIX ^[
```

【说明】

1. 用户在 **vi** 中创建的映射键是临时的，只对当前编辑会话有效。

2. 映射键在 vi 处于命令模式下进行指定和使用。

【注意】

如果[Return]或[Esc]是映射键的一部分,用户必须在按回车键和[Esc]前加[Ctrl-v]。

6.6.5 .exrc 文件

用户在 vi 编辑器中所设置的所有选项都是临时的;当用户退出 vi 时,它们都会失效。要想使这些设置成为永久的,而不需要每次使用 vi 时重新设置,可以将选项的设置保存到文件 .exrc 中。

【说明】

以.(点)开头的文件被称为隐藏文件,第5章中介绍过。

当用户打开 vi 编辑器时,它自动查看用户当前工作目录中的 .exrc 文件,并根据在文件中找到的内容设置编辑环境。如果 vi 没有在当前目录中发现 .exrc 文件,它将查找用户主目录,并根据在那里发现的 .exrc 文件设置编辑环境。如果 vi 一个 .exrc 文件也没找到,它对选项使用默认值。

vi 检查 .exrc 文件存在的方式给用户提供了强大的工具,用户可根据自己不同的编辑需要定义 .exrc 文件。例如,可以创建一个通用的 .exrc 文件在主目录,并在自己存入 C 程序的目录中创建一个用于 C 程序设计的 .exrc 文件。用户可以用 vi 创建一个 .exrc 文件,或修改现有的 .exrc 文件。

【实例】

创建一个 .exrc 文件,键入 vi .exrc 并按回车键。然后输入用户想用的 set 和其他命令(与下图类似)。

```
set report = 0
set showmode
set nu
set wm = 10
ab uop UNIX Operating System
map q 5dd
```

【注意】

文件名头部不要忘记.(点)。

如果用户在自己的主目录下创建该文件,则每次使用 vi 时都将建立编辑环境。在前面显示的设置下,vi 将显示其所处的模式、行号,设定右边界为 10 个字符(行长度为 70 字符),每当用户键入 uop 后跟空格时替换为 UNIX Operating System,每当用户按 q 键时删除 5 行。

【说明】

1. .exrc 属于启动文件。
2. 还有其他类似于 .exrc 用途的启动文件。

6.7 最后的 vi 命令

在总结对 vi 编辑器的讨论之前,必须考虑另外一个 vi 操作符。用户已经了解了足够的 vi 命令和有关细节,并可以用它来简单、有效地创建或修改自己的文件。但是,vi 能做到的不只这些。vi 编辑器有 100 多个命令和许多变量,给这些命令再加上域控制,可以使用户在编辑工作中有详细的控制。参考附录 F 可以帮助读者找到其他书来提高 vi 技巧。

6.7.1 运行 shell 命令

用户可以在 vi 命令行运行 UNIX shell 命令。这一方便的特性使用户可以临时抛开 vi 来运行 shell 命令。!(叹号)通知 vi 后面是一个 shell 命令。例如,要在 vi 编辑器中运行 **date** 命令,键入:**! date** 然后按回车键。vi 编辑器将清除屏幕,执行 **date** 命令,读者可以看到类似如下的屏幕显示:

```
Sat Nov 29 14:00:52 EDT 2001
[Hit any key to continue]
```

按任意一个键即可返回 vi 编辑器,并可在前面离开的地方继续编辑。如果用户希望,也可以将 shell 命令的结果读进来并加到用户文本中。使用:r(读取)命令,后面紧跟! 和相应的 shell 命令来将命令的结果结合进用户正在编辑的文本中。

【实例】

要读取系统的时间和日期,键入:**r ! date** 然后按回车键;vi 响应将当前系统日期和时间放在当前行下面。

vi 编辑器保持在文本输入模式。

```
The vi history
The vi editor is a text editor which is supported by most of the
UNIX operating systems.

The vi history
The vi editor is a text editor which is supported by most of the
Sat Nov 29 14:00:52 EDT 1993
UNIX operating systems.
```

【实例】

如下命令显示! 的使用。

- 键入:**! ls** 然后按回车键,列出当前目录中的文件。
- 键入:**! who** 然后按回车键,显示谁当前登录到系统上。
- 键入:**! date** 然后按回车键,显示当前日期和系统时间。
- 键入:**! pwd** 然后按回车键,显示当前工作目录路径。
- 键入:**r ! date** 然后按回车键,读取 date 命令的结果,并将其放置在正在编辑文本的光标后面。
- 键入:**r ! cal 1 2001** 然后按回车键,读取 2001 年 1 月的日历,并将它放在当前光标后面。

- 键入:! `vi mylast` 然后按回车键,调用另一个 vi 来编辑 mylast 文件。

6.7.2 行连接

使用 J 命令把 2 行连接在一起。J 命令将当前行下面一行接到当前行的后面。如果 2 行连接导致结果行过长,vi 将其按屏幕进行折行。

【实例】

要将 2 行连接在一起,执行下面操作。

- 用光标移动键将光标移至第一行最后一个字母。
- 按 J 键;vi 将当前行下面一行连接到当前行之后。

```
The vi history _
vi is a text editor which is supported by most of the
UNIX operating systems.
```

```
The vi history vi is a text editor which is supported by most of
the UNIX operating systems.
```

6.7.3 搜索和替换

有时,用户希望更改文件中的一个字。如果该文件非常长,遍历整个文件并找到要改的字并修改它是非常笨拙的做法。此外,用户在修改中漏掉一个或几个字的机会是很大的。一个更好的办法是使用 vi 的搜索命令(/和?),并结合其他命令完成此工作。

【实例】

如下命令显示 vi 搜索和替换功能。

- 键入:/UNIX,然后按回车键向后搜索,以发现第一个 UNIX。
- 键入 cwunix 然后按回车键,将 UNIX 替换为 unix。
- 键入 n 来寻找下一个 UNIX。
- 键入.(点)重复上次替换(UNIX 替换为 unix)。
- 键入:? unix 然后按回车键,向前搜索当前行,查找第一个 unix。
- 键入 dw 删除字 unix。
- 键入 n 查找下一个 unix。
- 按.(点)来重复最近一个命令(dw),删除 unix。

命令小结

下面是本章中讨论的 vi 编辑器命令的操作符。这些命令是读者在第 4 章学到命令的补充。

剪切和粘贴键

这些键用来重新安排用户文件中的文本。这些键在 vi 命令模式下可用。

键	功 能
d	删除一段指定的文本,并将它们保存在临时缓冲区中,这些缓冲区可利用放置操作符访问
y	复制一段指定的文本到临时缓冲区中,这些缓冲区可利用放置操作符访问
P	将临时缓冲区中的内容放置到光标的前面位置
p	将临时缓冲区中的内容放置到光标的后面位置

域控制键

使用 vi 命令结合域控制键使用户在进行编辑任务中具有更多控制能力。

键	功 能
\$	域为从光标位置到所在行尾
0(零)	域为从光标前一个位置到所在行首
e	域为从光标位置到所在字的尾部
b	域为从光标前一个位置到所在字的头部

翻页键

翻页键用来一次滚动多行文本。

键	功 能
[Ctrl - d]	使光标向文件尾移动,通常一次 12 行
[Ctrl - u]	使光标向文件头移动,通常一次 12 行
[Ctrl - f]	使光标向文件尾移动,通常一次 24 行
[Ctrl - b]	使光标向文件头移动,通常一次 24 行

设置 vi 环境

用户可以通过设置 vi 环境选项来定义 vi 编辑器的行为。用户使用 set 命令改变选项的值。

选 项	缩 写	功 能
autoindent	ai	将新行与前一行的开始对准
ignorecase	ic	在搜索选项下,忽略大小写
magic	-	在搜索时,允许使用特殊字符
number	nu	显示行号
report	-	告知用户,最后一个命令作用行的行号
scroll	-	设定使用[Ctrl - d]命令翻滚的行数
shiftwidth	sw	设定缩进空格数。与 autoindent 一同使用
show mode	smd	在屏幕右角显示 vi 编辑器模式
terse	-	缩短错误信息
wrapmargin	wm	将右边界设为一定的字符个数

习题

将下面左侧命令与右侧的解释配对。所有命令只能用于 vi 命令模式。

- | | |
|-------------|-----------------------|
| 1. G | a. 用字母 x 替换光标所在位置的字母。 |
| 2. /most | b. 将光标移到文件中最后一行。 |
| 3. [Ctrl-g] | c. 复制 4 行到缓冲区 x 中。 |
| 4. 2dw | d. 将光标下移一行。 |
| 5. j | e. 显示当前行的行号。 |
| 6. "x4yy | f. 将光标定位到第 66 行。 |
| 7. \$ | g. 删除当前光标下的字符。 |
| 8. O(零) | h. 恢复缓冲区 1 的内容。 |
| 9. 66G | i. 删除 2 个字。 |
| 10. x | j. 将光标定位在当前行的行尾。 |
| 11. rx | k. 查找字 most。 |
| 12. "lp | l. 将光标定位在当前行的行首。 |

在使用 vi 编辑器时,用什么命令完成如下功能:

- 设置行号选项?
- 保存 5 行在缓冲区 x?
- 读取(导入)日期串到用户文件中?
- 列出用户当前目标?
- 创建一个缩写?
- 取消一个缩写?
- 读取另一个文件?
- 写(保存)一个文件而不退出 vi 编辑器?
- 删除一个字?

上机练习

【实例】

在上机实习中,创建一个名为 `garden` 的文件,练习使用本章中讨论的编辑键。创建的文件如 Screen 1 所示。然后使用剪切和粘贴、光标定位和其他命令使之如 Screen 2 所示。最后,在用户的 `garden` 文件上应用如下命令。

- 创建一个用户名的缩写,并把它加到用户文件的头部。
- 创建一个映射键,完成查找包含指定字的行并删除该行。
- 撤消前面修改文本的命令。
- 在 vi 中,列出用户当前目录下的文件。
- 读入日期和时间,并将它们放到文件中用户名之后。
- 读入另一个文件(导入文件),并把它追加到 `garden` 文件的尾部。

7. 用另一个名字保存文件。
8. 设置 showmode 选项。
9. 设置 line number 选项。
10. 移动到文件末尾。
11. 移动光标到文件头部。
12. 移动光标到第 10 行。
13. 搜索字 weeds。
14. 将 report 选项设置为 1。
15. 取消 line number 选项。

Screen 1

Everywhere the trend is toward a simpler, and easy to care garden. Few advises might help you to have less trouble with your gardening. I am sure you have heard them before, but listen once more. Gardening: The easy approach visit the plant nurseries, it is good for your soul. Let me tell you that There is no easy to care garden. Use plants which are suitable for your climate . Native plants are good CHOICE. Before planting, choose the right site. Use your imagination, plants grow faster than what you think. Gardening can be made easier and more enjoyable if you hire a gardener to do the job. Use mulches to reduce weeds and save time in watering the plants. Do not use too much chemicals to kill every weed insight. You are the only one who sees the weeds, let them grow. They keep the moisture and prevent soil erosion.

Screen 2

Gardening: The easy approach

Everywhere the trend is toward making simpler and easier-to-care for gardens.

However, let me tell you that there are no easy-to-care for gardens.

Gardening can be made easier and more enjoyable if you hire a gardener to do the job.

Some advice might help you to have less trouble with your gardening.

I am sure you have heard it before, but listen once more:

1. Before planting, choose the right site.

Use your imagination. Plants grow faster than you think.

2. Visit the plant nurseries; it is good for your soul.

3. Use mulches to reduce weeds and save time in watering

4. Use plants which are suitable for your climate.

Native plants are a good choice.

5. Do not use too many chemicals to kill every weed in sight.

You are probably the only one who sees the weeds.

Let them grow.

They keep moisture and prevent soil erosion.

第 7 章 UNIX 文件系统(续)

本章补充第 5 章中的内容,是讨论 UNIX 文件系统及其相关命令的第 2 部分。这一章描述管理文件的更多命令,包括复制、移动和查看文件内容等命令,同时还解释了 shell 输入/输出重定向操作符和文件替换元符号。

7.1 读取文件

第 5 章解释了如何使用 vi 编辑器和 cat 命令来读文件。只更新内存的内容,可以用 vi 带只读选项来读取文件,或者使用 cat 查看小文件。当用 cat 命令查看大文件时,[Ctrl-s]停止屏幕输出和[Ctrl-q]继续屏幕输出,使用这两个组合键将会很方便。试一试下面的例子,能对此有更多的感性认识。

【实例】

假设当前的工作目录是 david,目录里有一个名为 large_file 的文件(20 页长)。

- 用 vi 编辑器读 large_file,输入 vi -R large_file,然后按回车键。这样用只读方式打开了 large_file。large_file 的内容显示在屏幕上,可以通过 vi 的命令查看其他页。
- 用 cat 命令读 large_file,输入 cat large_file,然后按回车键。large_file 的内容显示在屏幕上并不断滚动,用[Ctrl-s]和[Ctrl-q]停止和恢复滚屏。

7.1.1 vi 编辑器的只读版:view 命令

有些 UNIX 系统提供的 vi 编辑器是用 view 命令的版本。读取文件时,由于该命令兼容 vi 编辑器的所有命令,所以它是读大文件的一个不错的工具。使用 view 命令查看文件,避免了用户把正在编辑修改的内容保存到文件里。该命令用来只读文件。

7.1.2 读取文件:pg 命令

pg 命令一次查看一屏文件内容。提示符(:)显示在屏幕底部,按回车键继续查看文件的其余内容。当到达文件结尾时,pg 命令会在屏幕的最后一行显示 EOF(文件结尾)。此时按回车键回到 \$命令提示符。

通过 pg 命令的选项,可以对文件的显示格式和查看方式进行更多的控制。表 7.1 总结了这些选项。

表 7.1 pg 命令选项

选 项	功 能
-n	不需要用回车键来完成单字母命令
-s	用反白显示消息和提示符
-num	设置每一屏的行数为 num,默认值是 23 行

(续表)

选 项	功 能
-p str	把提示符:(冒号)改成设定的字符串 str
+ line - num	从设定的 line - num 行开始显示文件
+ /pattern	从包含设定模式的第一处开始显示

【注意】

与其他命令的选项不同,一些 pg 选项以加号(+)开始。

【实例】

假设工作目录有一个名为 large_file 的文件。通过如下操作,用 pg 命令读取该文件:

- 输入 `pg large_file`,然后按回车键。这种方法一次一屏简单地看文件。
 - 输入 `pg -p Next +45 large_file`,然后按回车键,现在从第 45 行开始看 large_file,同时通常的提示符:(冒号)换成了 Next。
- 这条命令用到两个选项;-p 把默认的提示符从:改成 Next;起始行从 1 变成 45 行。
- 输入 `pg -s 1/hello large_file`,然后按回车键,用反白显示模式显示提示符和其他消息,并从包含单词 hello 的第一行开始显示。

上述命令用到两个选项:-s 用反白显示方式来显示提示符和其他消息;1/hello 查找单词 hello 第一次出现的位置。

当 pg 命令显示出提示符:(或者用户通过 -p 选项设置的提示符)时,用户可以用命令前进或者后退设定的页数或行数,来查看文件的其他部分。表 7.2 对此总结了部分命令。

表 7.2 pg 命令关键操作符

关 键 字	功 能
+ n	前进 n 屏,n 是一个整数
- n	后退 n 屏,n 是一个整数
+ n l	前进 n 行,n 是一个整数
- n l	后退 n 行,n 是一个整数
n	前进到 n 屏,n 是一个整数

7.1.3 指定页和行数

用户可以从文件开始或者相对于当前页指定页数与行数。以文件开始位置为参考,用无符号整数来指定。例如,输入 10 前进到第 10 页,输入 60l(分别是 6、0 和小写字母 l)到文件的第 60 行。

以相对于当前页为参考,用带符号整数来指定。例如,输入 +10 向前 10 页,输入 -30l(3、0 和小写字母 l)后退 30 行。如果只输入 + 或者 - 而不带任何数字,命令分别被解释成 +1 或者 -1。

【注意】

只有在查看文件中 pg 显示提示符的时候,这些操作符才有效。

【说明】

如果 `pg` 带 `-n` 选项,输入单字母操作符不需要按回车键。

7.2 shell 重定向

shell 提供的最有用的功能之一是“shell 重定向操作符”。许多 UNIX 命令从标准输入设备得到输入并把输出发送到标准输出设备。这是通常的默认设置。利用 shell 重定向操作符,用户可以更改从何处得到输入以及把输出发送到哪里的命令。命令的标准(默认)输入/输出设备是用户终端。

shell 重定向操作符允许做下面的事情:

- 把进程的输出保存到文件中
- 用文件作为进程的输入

【说明】

1. 进程是任何可执行的程序,可以是 shell 命令、应用程序或者用户编写的程序。
2. 重定向操作符是针对 shell 的指令,并不是命令语法的一部分。因此,它们可以出现在命令行的任何地方。
3. 重定向是暂时的,只对使用它的命令有效。

7.2.1 输出重定向

输出重定向允许用户把进程的输出保存在文件中。然后,用户就可以对此进行编辑、打印或者用它作为其他进程的输入。

格式如下:

```
command > filename
```

或者

```
command >> filename
```

例如,为了列出当前工作目录的文件名,使用 `ls` 命令。输入 `ls` 然后按回车键。shell 默认输出设备是用户终端屏幕(标准输出设备),因此,用户能看到屏幕上的文件列表。假设用户想要把 `ls` 命令的输出(该目录的文件列表)保存到文件中,可把 `ls` 命令的输出从屏幕重定向到文件,第一种方法如下:

```
ls > mydir.list
```

这里,`ls` 的输出并没有被发送到终端屏幕,而是保存到名为 `mydir.list` 的文件中了。如果打开 `mydir.list` 文件,可以看到文件列表。

【说明】

1. 如果指定的文件名已经存在,那么就会进行写覆盖,原有文件的内容丢失。
2. 如果指定的文件名不存在,shell 创建一个文件。

除了前者是把输出附加到指定文件中外,双大于号(>>)重定向操作符与大于号(>)工作方式基本相同。如果输入 `ls >> mydir.list`, shell 把工作目录的文件名列表添加到名为 `mydir.list` 文件的结尾。

【说明】

1. 如果指定文件不存在,那么 shell 创建一个文件来保存输出。
2. 如果指定文件存在,shell 把输出添加到文件末尾,文件原来的内容保持不变。

下面的命令序列显示更多输出重定向的例子。

【实例】

得到用户主目录里文件名的硬拷贝,如下操作:

```
$ cd [Return].....改变到用户主目录
$ ls -C [Return].....用列格式列出 david 目录中的文件名。假设有
                        两个文件:
myfirst          yourlast
$ ls -C > mydir.list [Return].....把输出保存在 mydir.list 中
$ .....完成。出现命令提示符
$ cat mydir.list [Return].....检查 mydir.list 文件中的内容
myfirst          yourlast
$ lp mydir.list [Return].....打印列表
request id is lp1 - 8056 ( 1 file )
$ .....等待其他命令
```

【实例】

把系统中的用户列表添加到 `mydir.list` 文件中,如下操作:

```
$ who >> mydir.list [Return].....把系统中用户的列表附加到 mydir.list 中。
                        现在 mydir.list 包含文件名列表和当前登录
                        的用户列表
$ date > mydir.list [Return].....把命令 date 的输出保存到 mydir.list 中。现
                        在 mydir.list 原来的内容丢失,文件中记录的
                        只有最后一条命令的结果
$ .....等待其他命令
```

【实例】

把当前的年历存入 `this_year`, 打印, 然后删除文件, 如下操作:

```
$ cal > this_year [Return].....把 cal 的输出保存到 this_year 中
$ lp this_year [Return].....打印
request id lp1 - 6889 (1 file)
$ rm this_year [Return].....删除文件
$ .....等待其他命令
```

7.2.2 输入重定向

输入重定向操作符允许用户从设定的文件得到输入去运行命令或者程序。shell 把小于号(<)作为输入重定向操作符。格式如下:

```
command < filename
```

或者

```
command << word
```

例如,向其他用户发送邮件,使用 **mailx** 命令(**mailx** 将在第9章讨论),输入 **mailx daniel < memo**。这个命令告诉计算机给用户 **daniel**(该用户的 **id**)发送邮件。**mailx** 的输入并非来自标准输入设备用户终端,而是来自名为 **memo** 的文件。因此,输入重定向操作符(**<**)被用来指示输入来自何处。

【实例】

cat 命令使用输入重定向。

□ 输入 **cat < mydir.list**, 然后按回车键,显示 **mydir.list** 的内容。UNIX 的响应为:

```
myfirst yourlast
```

执行带有输入重定向符的 **cat** 命令,可以得到与带文件名作为参数的 **cat** 命令同样的结果。其他一些命令也用这种方式工作。如果在命令行中设定文件名,命令就把指定的文件名作为输入;如果不指定任何参数,命令从默认的输入设备(用户终端)取得输入;如果利用输入重定向符指定输入的来源,命令就会把指定的文件作为输入。

重定向符(**<<**)经常被用在脚本文件(shell 程序)中,为其他命令提供标准输入(第10章详细讨论该主题)。

7.2.3 回顾 cat 命令

现在,我们知道了 shell 重定向的功能,于是可以更深入地利用 **cat** 命令了。**cat** 命令在第5章中进行过介绍,用来在屏幕上显示小文件的内容。然而,**cat** 命令可以用来做比显示文件更多的事。

创建文件

执行带输出重定向符(**>**)的 **cat** 命令可以创建文件。例如,如果想要创建名为 **myfirst** 的文件,输入 **cat > myfirst**。这条命令意味着 **cat** 的输出从标准输出设备(用户终端)重定向到名为 **myfirst** 的文件。输入来自标准输入设备——用户终端(键盘)。换句话说,用户输入文本,**cat** 把它们保存在 **myfirst** 文件里。按 **[Ctrl-d]** 发送信号表示文件结束。

cat 命令的这个特点对于迅速创建小文件很有用。当然,一样可以用它创建长文件。但是必须注意,用户应是一个非常准确的打字员,因为当按回车键之后不能编辑已经输入的文本。下面的命令序列显示如何用 **cat** 命令创建文件。

【实例】

执行带输出重定向符的 **cat** 命令创建文件。

```
$ cat > myfirst [Return].....创建名为 myfirst 的文件
$_.....光标准备接收用户输入。输入如下:
I wish there were a better way to learn
UNIX. Something like having a daily UNIX pill.
[Ctrl-d].....结束输入
$_.....准备接收下一条命令
```

```

$ cat myfirst [Return].....查看 myfirst 是否已经被创建,并显示在屏
幕上
I wish there were a better way to learn
UNIX. Something like having a daily UNIX pill.
$ _.....等待下一条命令

```

【说明】

1. 如果工作目录并不存在 myfirst,那么 cat 创建该文件。
2. 如果工作目录已经存在 myfirst,那么 cat 覆盖它。旧的 myfirst 文件内容丢失。
3. 如果不想覆盖文件,用(>>)操作符。

【实例】

在当前目录中 myfirst 文件的末尾附加文本。

```

$ cat >> myfirst [Return].....创建或打开名为 myfirst 的文件
_.....输入文字
However, for now, we have to suffer and read all these boring UNIX books.
[Ctrl-d].....通知输入文本结束
$ _.....回到命令提示符下
$ cat myfirst [Return].....显示 myfirst 的内容
I wish there were a better way to learn
UNIX. Something like having a daily UNIX pill.
However, for now, we have to suffer and read all these boring UNIX books.

```

【说明】

1. 如果工作目录不存在 myfirst,cat 将创建文件。
2. 如果 myfirst 存在,cat 把输入文字附加到已存在文件的尾部。这个例子中,一行输入被加到 myfirst 中,因此显示了三行文本。

复制文件

利用带输出重定向符的 cat 命令能够把文件复制到别处。下面的命令序列显示了 cat 如何实现这种功能。

【实例】

把 david 目录中的 myfirst 复制成另一个名为 myfirst.copy 的文件。

```

$ cd [Return].....确保在用户主目录
$ cat myfirst > myfirst.copy [Return]... 把 myfirst 复制成 myfirst.copy
$ _.....等待下一条命令

```

【说明】

cat 的输入是 myfirst 文件,cat 的输出(myfirst 的内容)被保存在 myfirst.copy 中。

【实例】

把 david 目录中的 myfirst 复制到 source 目录,并改名成 myfirst.copy。

```

$ cat myfirst > source/myfirst.copy [Return].....从 myfirst 复制到 myfirst.copy
_.....放到 source 目录下
$ ls source/myfirst.copy [Return].....检查是否已经被复制

```

```
myfirst.copy
$_..... 没错,myfirst.copy 在 source 目
          录中
```

【说明】

由于用户处于用户主目录,运行命令 `cat` 和 `ls` 时,需要指定 `source` 目录中 `myfirst.copy` 的路径。

【实例】

接着,用 `cat` 命令把两个文件复制到第三个文件。

```
$ cat myfirst myfirst.copy > xyz [Return]..... 把 myfirst 和 myfirst.copy 复制
                                                    到 xyz
$_..... 等待下一条命令
```

【注意】

1. `xyz` 文件如果原来有内容的话,会丢失。
2. 命令行的每个文件名之间需要空格。

附加文件

利用 `cat` 命令,带有输出重定向附加符(`>>`)可以把许多文件一起加到一个新文件里。

【实例】

把两个文件附加到第三个文件尾部。

```
$ cat myfirst myfirst.copy >> xyz [Return].....myfirst 和 myfirst.copy 附加到名
                                                    为 xyz 的文件尾部
$_..... 显示命令提示符
```

【说明】

1. 用(`>>`)重定向符保留 `xyz` 原来的内容(如果有的话),两个文件附加到文件 `xyz` 的尾部。
2. 命令行中可以有多于两个文件名,但是必须用空格分开。
3. 被附加到输出文件的文件内容与输入文件指定的顺序一致。

7.3 增强的文件打印功能

命令 `lp` 把文件按原样送到打印机,并不改变文件的外观或者格式。但是,用户可以进行格式化,以改善输出外观——例如,在送到打印机或者显示在屏幕上之前,可以给文档增加页号、页眉、双倍行距等。

打印或者显示之前,执行 `pr` 命令格式化文件。不带参数的 `pr` 命令把指定文件格式化成每页 66 行的格式。每页的顶部有 5 行的页眉,由两行空白、一行文件信息以及另外两个空行组成。信息行包括当前日期和时间、指定文件名和页号。每页的底部也产生 5 行空白。

【实例】

用 `pr` 命令格式化 `myfirst`。

□ 输入 `pr myfirst` 然后按回车键, 格式化名为 `myfirst` 的文件(见图 7.1)。

```

                                [2 blank lines]
Nov 28 16:30 2001 myfirst Page 1
                                [2 blank lines]

The vi history

The vi editor is a text editor which is supported by most of the
UNIX operating systems. However. . . .

rest of the page . . .

                                [5 blank lines at bottom of page]
```

图 7.1 格式化的打印文件(页格式:5 行页眉和 5 行空页脚)

`pr` 的输出显示在终端(标准输出设备)上。但是,多数情况下用户希望得到长久保留的打印输出。实现该目的的一种方法是利用输出重定向符,另外还可以把格式化的文件送到打印机,例如用管道操作符(`|`)(第 8 章解释)。

【实例】

把 `myfirst` 经过格式化的版本保存到另一个文件,然后打印。

1. 选项 `-m` 或者 `-columns` 被用来产生多列输出。
2. `-a` 选项只能与 `-columns` 一起用,不可以与 `-m` 一起用。

```

$ pr myfirst > pout [Return].....把格式化过的 myfirst 保存到名为 pout 的文件
$ lp pout [Return].....打印 pout
requested id is lpl-8045 (1 file)
$ rm pout [Return].....如果不需要了,删除 pout
$_.....等待下面的命令
```

`pr` 选项:`pr` 选项允许用户用更复杂的格式控制文件外观。表 7.3 总结了 `pr` 命令选项。下面的命令序列显示带不同选项 `pr` 命令的输出。

表 7.3 `pr` 命令选项

选 项	功 能
<code>+ page</code>	从指定页开始显示。默认是第 1 页
<code>- columns</code>	用指定的列数显示输出。默认是 1 列
<code>- a</code>	以横跨页面的方式显示输出,每列一行
<code>- d</code>	双倍行距显示输出
<code>- h str</code>	用指定字符串 <code>str</code> 代替页眉的文件名
<code>- l number</code>	用指定的行数设置页长。默认值是 66 行
<code>- m</code>	用多列显示所有指定文件
<code>- p</code>	在每页末暂停,并响铃

(续表)

选 项	功 能
- character	用单个指定的字符分割列。如果没有指定字符,那么用[tab]
-t	取消5行页眉和5行页脚
-w number	用指定字符数设置行宽。默认值为72

【实例】

该例假设用户的工作目录有两个文件,用 `cat` 命令创建它们。

```
$ cat > names [Return].....创建名为 names 的文件
David [Return]
Daniel [Return]
Gabriel [Return]
Emma [Return]
[Ctrl-d]

$ cat > scores [Return].....创建名为 scores 的文件
90 [Return]
100 [Return]
70 [Return]
85 [Return]
[Ctrl-d]

$_.....等待命令
```

用列格式显示 `names`,修改页眉成 `STUDENT LIST`,输入 `pr-2-h "STUDENT LIST" names` 然后按回车键。

```
Nov 28 2001 14:30 STUDENT LIST [2 blank lines]
[Page 1]
[2 blank lines]
David Gabriel
Daniel Emma

[5 blank lines at bottom of page]
```

`-h` 选项修改了页眉,但是如果指定的字符串中有空格键,必须用引号括起来。

这样,屏幕将会滚动,用户无法看到整个文件。一种查看的办法是把 `pr` 的输出重定向到文件中,然后用 `vi` 编辑器或者 `view` 命令查看格式化的输出。

□ 输入 `pr myfirst > outfile`,然后按回车键。于是把 `pr` 的输出保存到 `outfile` 中。

□ 输入 `view outfile`,然后按回车键,查看 `pr` 命令生成的输出。

【说明】

如果系统中没有 `view` 命令,输入 `vi -R`(只读)代替。

【实例】

用横跨页的两列方式显示 `names`,取消页眉,输入 `pr-2-a-t names`。

```
David Daniel
```

```
Gabriel      Emma
```

选项 `-2` 和 `-2 -a` 的差别在于列的排列顺序。

【实例】

并排显示 `names` 和 `scores`, 输入 `pr -m -t names scores`, 然后按回车键。

```
David      90
Daniel    100
Gabriel    70
Emma      85
```

选项 `-m` 并排显示指定的文件, 顺序与命令行中设定的文件名一致。

7.4 文件处理命令

第 5 章已经讨论了一些文件处理命令, 用户知道如何创建目录(用 `mkdir` 命令)、创建文件(用 `vi` 和 `cat` 命令)以及删除文件和目录(用 `rm` 和 `rmdir` 命令)。这一节介绍更多的命令, 扩展用户管理 UNIX 文件的知识。这些命令用来复制(`cp`)、链接(`ln`)和移动(`mv`)文件。这些命令的通用格式是:

```
command source target
```

`command` 是上述任何命令, `source` 是源文件名, `target` 是目的文件名。

7.4.1 复制文件: `cp` 命令

`cp`(copy)命令用来创建一个复制的文件。用户可以把文件从某个目录复制到另一个, 对文件进行备份或者仅仅出于兴趣进行复制。

【实例】

假设当前目录有一个名为 `REPORT` 的文件, 现在要创建它的副本。

输入 `cp REPORT REPORT.COPY`, 然后按回车键。

`REPORT` 是源文件, `REPORT.COPY` 是目的文件。如果用户没有给源/目的文件提供正确的路径/文件名, `cp` 显示类似于下面的消息:

```
File cannot be copied onto itself
0 file(s) copied
```

图 7.2 显示执行 `cp` 命令前后的用户目录结构。

【注意】

如果目标文件已经存在, 那么它的内容会被毁坏。

`cp` 选项

表 7.4 总结了 `cp` 命令的选项。

`-i`选项: 保护避免覆盖存在的文件。如果目标文件已经存在, 会请求确认。如果回答

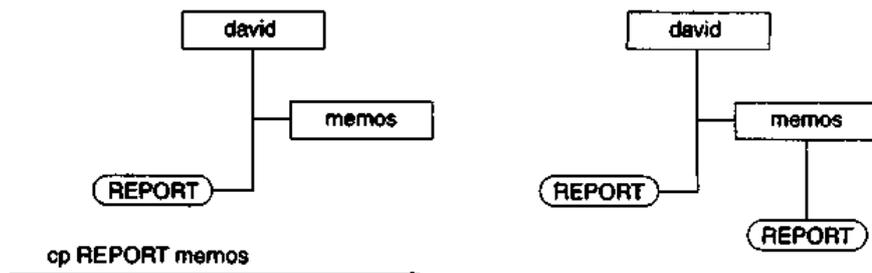


图 7.3 cp 命令的应用实例

用 `-r` 选项,把 `david` 目录下的文件和子目录复制到名为 `david.bak` 的目录。

```
$ cp -r ./memos ./davis.bak [Return].....复制 memos 目录和其中所有文件到目录
                                     david.bak
$_.....等待其他命令
```

【说明】

1. 如果当前目录存在 `david.bak`,那么 `memos` 中的文件和目录被复制到 `david.bak` 中。
2. 如果 `david.bak` 不存在,先被创建,然后 `memos` 中的文件和目录被复制进来。现在,`david.bak` 目录下的 `memos` 中文件的路径为: `./david.bak/memos`。

7.4.2 移动文件:mv 命令

`mv` 命令用来把文件从一个地方移动到别处,或者用来更改文件或目录的名字。例如,当前目录下有一个名为 `REPORT` 的文件,要把文件名改成 `REPORT.OLD`,可以输入:`mv REPORT REPORT.OLD`,然后按回车键。

图 7.4 显示了用 `mv` 命令为 `REPORT` 改名这一操作前后的目录结构。

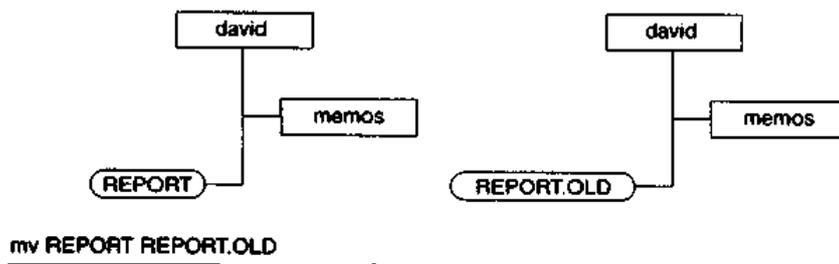


图 7.4 用 mv 给文件改名

【实例】

把 `REPORT` 移动到 `memos` 目录。

```
$ mv REPORT memos [Return].....移动 REPORT 到 memos
$_.....等待用户命令
```

图 7.5 显示用 `mv` 命令移动 `REPORT` 前后的目录结构。

`cp` 和 `mv` 命令都接受多于两个的参数,但是最后一个参数必须是一个目录。例如:

```
cp xfile yfile zfile backup
```

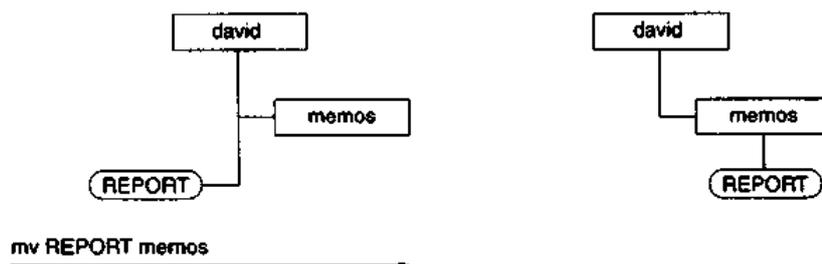


图 7.5 用 mv 移动文件

假设 backup 目录存在,以上命令复制 xfile、yfile 和 zfile 到目录 backup 中。

7.4.3 链接文件:ln 命令

ln 命令用来在现有文件和一个新文件名之间建立新链接。这样就可以为现有文件创建另外的文件名,可以用不同的名字指向相同的文件。例如,假设用户当前目录有一个名为 REPORT 的文件,输入 ln REPORT RP,然后按回车键。现在,当前目录下创建了一个文件 RP,链接到 REPORT。REPORT 和 RP 是同一个文件的两个文件名。图 7.6 显示了执行 ln 命令前后的目录结构。

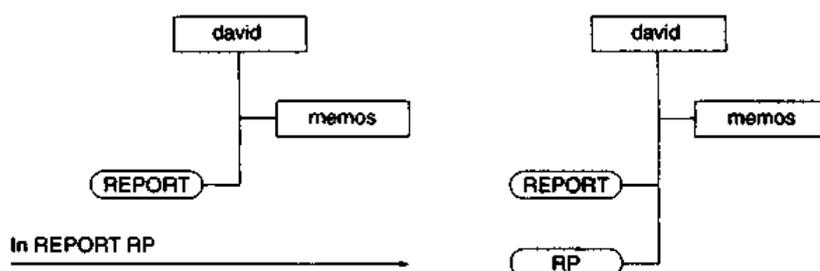


图 7.6 用 ln 命令链接文件

乍一看,ln 很像 cp 命令,但其实不同。cp 命令把文件物理复制到其他位置,用户有两个分别的文件。对其中一个文件所做的修改并不影响到另外一个文件。然而,用 ln 命令仅仅是为文件建立了另外一个文件名,并没有新文件被创建。如果改变了链接文件中任何一个的内容,无论用哪个文件名,都会看到文件发生了改变。

【实例】

按照下面的命令序列,学习使用 ln 命令。

```
$ cat > xxx [Return].....创建 xxx 文件,输入如下行:
Line 1: aaaaaa
$ [Ctrl-d].....表示输入结束
$ ln xxx yyy [Return].....链接 yyy 到 xxx
$ cat yyy [Return].....显示 xxx 的内容,但是使用 yyy 文件名;输出如我们所期待的那样,是 xxx 的内容:
Line 1: aaaaaa
$ cat >> yyy [Return].....在 yyy 尾部附加一行,输入如下行:
Line 2: bbbbbbb
[Ctrl-d].....输入结束
$ cat yyy [Return].....显示 yyy 的内容。正如所料,yyy 有两行:
```

```
Line 1: aaaaaa
Line 2: bbbbbbb
$ cat xxx [Return]..... 显示 xxx 的内容。xxx 和 yyy 实际是同一个文件,因此内容也如下:
```

```
Line 1: aaaaaa
Line 2: bbbbbbb
$_..... 等待用户输入其他命令
```

如果指定一个已经存在的目录名作为新文件名,用户用不着输入路径就可以在指定目录中访问该文件。例如,假设工作目录中有子目录 `memos` 和文件 `REPORT`,输入 `ln REPORT memos` 然后按回车键。现在可以从 `memos` 目录直接访问 `REPORT`,不用输入路径(上一个例子中原来的访问路径应该是 `../REPORT`)。

如果想指定不同的文件名,输入 `ln REPORT memos/RP`,然后按回车键。现在, `memos` 目录中的 `RP` 链接到 `REPORT`,从 `memos` 目录中可以用文件名 `RP` 来访问 `REPORT`。

图 7.7 显示了执行 `ln` 命令前后的目录结构。

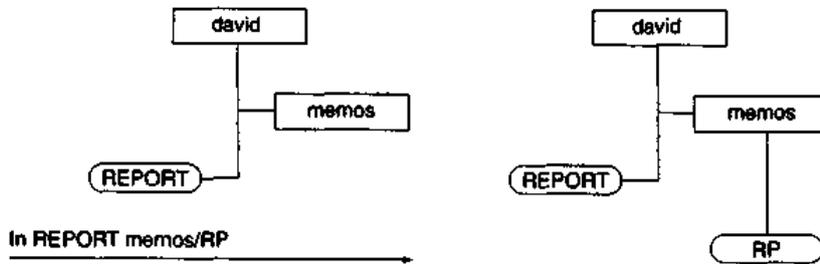


图 7.7 用 `ln` 命令把文件名链接到目录

第 5 章解释过用 `ls -l` 命令以长格式列出当前目录的文件,长格式输出的第 2 列显示链接数。

【实例】

用长格式列出 `david` 目录中的文件,输入 `ls -l`,然后按回车键。

```
$ ls -l
total 2
drwx rw- -- 1 david student 32 Nov 28 12:30 memo
-rwx rw- -- 1 david student 155 Nov 18 11:30 REPORT
```

链接 `REPORT` 到 `RP`,用带 `-l` 选项的 `ls` 命令查看链接数。输入 `ln REPORT RP` 然后按回车键,接着输入 `ls -l`,按回车键。

```
$ ln REPORT RP
$ ls -l
total 3
drwx rw- --- 1 david student 32 Nov 28 12:30 memo
-rwx rw- --- 2 david student 155 Nov 18 11:30 REPORT
-rwx rw- --- 2 david student 155 Nov 18:11:30 RP
```

【说明】

当用户创建文件时,系统同时就建立了所在目录和文件之间的一个链接。因此,每个文件

的链接数至少为 1,使用 `ln` 导致链接数增加。

小结

`cp`、`mv` 和 `ln` 这三个命令都会对文件名产生影响,命令格式虽相同,但是它们是不同的命令,用于不同的目的:

- `cp` 创建新文件
- `mv` 更改文件名或者把文件从一个位置移到别处
- `ln` 为已经存在的文件建立别名(链接)

7.4.4 计算字数: `wc` 命令

`wc` 命令用来计算一个文件或者指定的多个文件中的行数、单词数和字符数。假设当前目录下有文件 `myfirst`,下面的命令序列显示 `wc` 的输出。

【实例】

首先显示 `myfirst` 的内容,然后显示文件中的行数、单词和字符数。

```
$ cat myfirst[Return] .....显示 myfirst 的内容
I wish there were a better way to learn
UNIX. Something like having a daily UNIX pill.
However, for now, we have to suffer and read all these boring UNIX books.
$ wc myfirst[Return].....计算 myfirst 的行、单词和字符数
4 30 155 myfirst
$_.....等待下一条命令
```

第一列显示行数,第二列显示单词数,第三列显示字符数。

【说明】

单词被认为是没有空白键(空格或者制表符)的字符序列。因此, `what?` 是一个单词,而 `what ?` 是两个单词。

如果没有指定文件名, `wc` 从标准输入设备(键盘)得到输入。 [`Ctrl - d`] 表示输入结束, `wc` 在屏幕上显示结果。

【实例】

用 `wc` 命令得到键盘输入数。

```
$ wc [Return].....不带文件名执行 wc 命令
_.....命令提示符表明 shell 等待输入。输入如下:
The wc command is useful to find out how large your file is.
[Ctrl - d].....输入结束;wc 显示输出
2 13 48
$_.....等待其他命令
```

用户可以设定多于一个的参数。这种情况下,输出为每个文件显示一行,最后一行显示总数。

假设当前目录有两个文件,下面的命令说明了设定两个文件作为参数的 `wc` 命令的输出。

【实例】

计算指定文件的行数、单词数和字符数。

```
$ wc myfirst yourfirst [Return]...显示 myfirst 和 yourfirst 的计数
24          10          40 myfirst
 3          100         400 yourfirst
27          110         800 total
$_.....等待其他命令
```

wc 选项:带选项的 **wc** 命令只能够得到行数、单词数或者字符数或者其任意组合。表 7.5 总结了 **wc** 命令选项。

表 7.5 **wc** 命令选项

选 项	功 能
-l	报告行数
-w	报告单词数
-c	报告字符数

【说明】

1. 没有选项设定的话,默认为所有选项(-lwc)。
2. 用户可以使用选项的任意组合。

下面的命令说明如何使用 **wc** 选项。

【实例】

计算 **myfirst** 的行数。

```
$ wc -l myfirst [Return].....只报告行数
4 myfirst
$_.....等待输入
$ wc -lc myfirst [Return].....报告行数和字符数
155 4 myfirst
$_.....回到命令提示符状态
```

把 **wc** 的输出保存到文件中并打印。

```
$ wc myfirst > myfirst.count [Return]..用输出重定向符将 wc 的结果保存到 myfirst.
count 中
$ lp -m myfirst.count [Return].....打印 myfirst.count,完成打印任务后,发送消息
$_.....等待其他命令
```

7.5 文件名置换

许多文件操作命令需要文件名作为参数。当用户需要操作大量文件时,比如,把所有文件名以字符 **a** 开头的文件转移到另一个目录,一个接一个地输入所有的文件名费力而且烦人。**shell** 支持“文件替换”,允许用户选择那些文件名与指定模式相匹配的文件。生成这些模式是在文件名中指定一些具有特殊意义的字符。具有特殊意义的字符被称作元字符(或通配符)。

表 7.6 总结了代表文件名一个或多个字符的通配符。

表 7.6 shell 文件代替元字符

关 键 字	功 能
?	匹配任何单个字符
*	匹配任意字符串,包括空串
[list]	匹配 list 中设定的任一字符
[! list]	匹配不在 list 中指定的任一字符

【说明】

文件代替元字符(通配符)可以使用在文件名的任何部分,包括开头、中间或者末尾来建立模式。

7.5.1 “?”元字符

问号(?)是一个特殊的字符,shell 解释成单个代替字符,据此来扩展文件名。

【实例】

如下命令显示“?”元字符如何工作。

```
$ ls -C [Return].....查看工作目录的文件名。现有如下文件:
report  report1  report2  areport  breport  report32
$ ls -C report? [Return].....文件名中用一个问号
report1  report2
$_.....等待用户命令
```

shell 把文件名 report? 扩展成 report 后仅跟一个任意字符的文件名,因此,只有 report1 和 report2 两个文件与模式匹配。

```
$ ls report?? [Return].....用两个问号作为特殊字符
report32
$_.....等待其他命令
```

shell 把文件名 report?? 扩展成 report 后仅跟两个任意字符的文件名,因此,只有 report32 这个文件与模式匹配。

```
$ ls -C ?report [Return].....把“?”放在文件名开始
areport  breport
$_.....等待命令
```

shell 把文件名 ?report 扩展成 report 前仅有一个任意字符的文件名,因此,只有 areport 和 breport 两个文件与模式匹配。

7.5.2 “*”元字符

星号(*)是一个特殊的字符,shell 解释成任意个(包括 0 个)代替字符,据此来扩展文件名。

【实例】

以下命令显示“*”元字符如何工作。

```
$ ls -C [Return].....查看工作目录的文件名。现有如下文件：
report  report1  report2  areport  breport
report32
$ ls -C report * [Return].....文件名中用一个问号
report  report1  report2  report32
```

shell 把文件名 report * 扩展成 report 后跟任意个数的字符。因此,只有 areport 和 breport 与模式不匹配。“*”通配符包括 0 个字符。所以,文件名 report 虽然没有其他字符在最后,但也匹配模式,因而显示出来了。

```
$ ls -C *report [Return].....列出以 report 结尾的文件名
report  areport  breport
$_.....等待命令
```

shell 把文件名 *report 扩展成 report 前有任意个数的字符。因此,report、areport 和 breport 是与模式匹配的文件。“*”通配符包括 0 个字符。所以,文件名 report 虽然没有其他字符在开始,但也匹配模式,因而显示出来了。

```
$ ls -C r*2 [Return].....列出“r”开始和“2”结束的文件名
report2  report32
$_.....等待其他命令
```

shell 把文件名 r*2 扩展成以字符“r”作为开始,其后跟随任意字符,但是必须以字符“2”结尾的文件名。

7.5.3 []元字符

围绕着字符串的开闭方括号是特殊字符。shell 解释成包含指定字符串的文件名,据此来扩展文件名。在指定字符串前用“!”使 shell 的解释变成:指定的位置不包含字符串中的字符。

【实例】

通过如下操作练习使用方括号元字符。

□ 列出以“a”或者“b”开始的文件名,输入 `ls -C [ab]*`, 然后按回车键,系统显示:

```
areport  breport
$_.....等待其他命令
```

shell 把文件名 [ab]* 扩展成 a 或者 b 后跟任意字符的文件名,因此,只有 areport 和 breport 两个文件与模式匹配。

□ 列出所有并非以“a”或者“b”开始的文件名,输入 `ls -C [!ab]*`, 然后按回车键,UNIX 系统显示:

```
report  report1  report2  report32
```

【说明】

用户可以使用 [] 元字符设定字符或者数字的范围。例如,[5-9]意味着数字 5、6、7、8 和

9:[a-z]意味着所有小写字母。

【实例】

下面的命令显示方括号设定字符或数字范围的用法。

□ 输入 `ls *[1-32]`, 然后按回车键列出所有以 1 到 32 结尾的文件名, 系统显示:

```
report1  report2  report32
```

7.5.4 元字符和隐藏文件

用元字符显示隐藏文件——以“.”(点)开始的文件名——必须显式地以“.”(点)作为指定模式的一部分。

【实例】

列出隐藏文件, 输入 `ls -C .*`, 然后按回车键, 系统显示:

```
.exrc  .profile
```

shell 把文件名 `*` 扩展成 `.(点)` 后跟随任意字符的文件名, 因此, 只有隐藏文件被显示。

【注意】

模式为“`.*`”在点与星号中间没有空格。

【说明】

通配符并不只限于在 `ls` 命令中使用。需要文件名作为参数的其他命令一样可以使用通配符。

【实例】

下面显示文件代替符的更多用法:

```
$ rm *.* [Return].....删除所有文件名至少包含一个点号的文件
$ rm report? [Return].....删除所有文件名以 report 开始,其后仅有一个任意字符的文件
$ cp * backup [Return].....把当前目录的所有文件复制到 backup 目录
$ mv file[1-4] memos [Return]..把 file1、file2、file3 和 file4 移动到 memos 目录
$ rm report * [Return].....删除所有文件名以 report 开头的文件
$_.....等待其他命令
```

【说明】

在 `report` 和星号通配符中间没有空格。`rm report *` 命令删除所有文件名以 `report` 开头的文件。

```
$ rm report * [Return].....删除所有文件
$_.....等待其他命令
```

【注意】

上边的例子, `report` 和星号通配符之间有一个空格。这个空格可能导致灾难性的后果。命令 `rm report *` 被解释成“删除名为 `report` 的文件, 然后删除所有文件”。换句话说, 当前目录下的所有文件都被删除了。

```
$ ls -C [A-Z] [Return].....显示所有文件名为单个大写字母的文件,假设有一些单字母
                             文件名。
A          B          D          W
$_.....出现命令提示符
```

7.6 其他文件操作命令

下面的命令可方便地从一大堆目录中查找指定文件,以及迅速查看文件的特定部分。

7.6.1 寻找文件:find 命令

find 命令用来在目录层次中定位与一组给定的标准相匹配的文件。标准可以是文件名或者文件的指定属性(例如更改日期、大小或者类型)。用户也可以将命令定向到删除、打印或者其他文件操作。**find** 命令是一个有用而且重要的命令,但通常难以用到其全部功能。也许是因为它不常见的命令格式令人气馁。

find 命令的格式与其他 UNIX 命令不同。下面来看看它的语法:

```
find pathname search options action option
```

pathname 表明 **find** 开始搜索的目录名,然后会向下搜索它的子目录、子目录的子目录等等。分支搜索的过程被称作“递归搜索”。“查找选项”确定用户感兴趣的文件,“动作选项”指示一旦发现了文件该怎么做。看一个简单的例子:

```
$ find . -print [Return]
```

这条命令显示指定目录及其子目录中的所有文件。

【说明】

1. 指定目录是一个句点,表明是当前目录。
2. 在路径名之后的选项总是以连字符(-)开始。动作选项指示对文件做什么动作。这里, **-print** 表明显示它们。

【注意】

别忘了 **-print** 关键字,如果没有, **find** 不会显示任何文件名。

查找选项

表 7.7 显示了部分查找选项列表以及选项的解释。

表 7.7 find 命令的查找选项

操作符	描述
-name 文件名	寻找给定 filename 的文件
-size ± n	寻找大小 n 的文件
-type 文件类型	寻找具有给定访问模式的文件
-atime ± n	寻找 n 天以前访问的文件
-mtime ± n	寻找 n 天以前更改的文件
-newer 文件名	寻找比 filename 更近更新的文件

- **name** 选项:利用文件名发现文件。输入 - **name** 后面跟着想要找的文件名。文件名可以是简单文件名或者用 shell 通配符([],? 和 *)。如果用这些特殊的字符,用引号括起文件名。来看几个例子。表 7.7 中的符号 $\pm n$ 是十进制数,可以被设定为 + n (表明大于 n)、- n (表明小于 n)或者 n (恰好大小为 n)。

【实例】

通过名称寻找文件。

```
$ find . -name first.c -print [Return]....寻找名为 first.c 的文件
$ find . -name "*.c" -print [Return].....寻找所有以 .c 结尾的文件
$ find . -name "*.*" -print [Return].....寻找所有文件名以 "." 跟一个字符结束的文件
```

【说明】

1. 前边所有的命令,目录名都是当前目录。
2. - **name** 选项确定文件名,通配符用来生成文件名。
3. 动作选项 - **print** 用来显示发现的文件。

- **size** $\pm n$ 选项:利用以块为单位的文件大小来寻找文件。用户输入 - **size** 后跟想要查找的文件的块数。块数前的加号和减号分别表明大于和小于。看下面的例子。

【实例】

通过大小寻找文件。

```
$ find . -name "*.c" -size 20 -print [Return]....寻找确切为 20 个文件块大小的文件
这条命令查找文件名以 .c 结尾,大小为 20 个文件块的文件。
```

```
$ find . -name "*.c" -size +20 -print [Return].....寻找大于 20 块的文件
$ find . -name "*.c" -size -20 -print [Return].....寻找小于 20 块的文件
```

- **type** 选项:通过类型查找文件。输入 - **type** 并跟指定文件类型的字符。文件类型如下:

- b:块特殊文件(例如磁盘)
- c:字符特殊文件(例如终端)
- d:目录文件(例如目录)
- f:普通文件(例如用户文件)

【实例】

用文件类型选项查找文件。

```
$ find $HOME -type f -print [Return].....利用 -type 选项
```

这条命令查找所有普通文件,显示路径名。

- **atime** 选项:利用最后访问日期查找文件。输入 - **atime**,后跟从文件最后被访问开始所过的天数。天数前的加号和减号分别表示大于和小于。看下面的例子。

【实例】

通过最后访问时间发现文件。

```
$ find . -atime 10 -print [Return].....查找和显示 10 天以前访问过的文件
```

这条命令显示那些 10 天没有被读过的文件。

```
$ find . - atime - 10 - print [Return].....查找和显示少于 10 天以前访问过的文件
$ find . - atime + 10 - print [Return].....查找和显示多于 10 天以前访问过的文件
```

-mtime 选项:利用最后修改日期查找文件。输入 -atime,后跟从文件最后被修改开始所过的天数。天数前的加号和减号分别表示大于和小于。看下面的例子。

【实例】

通过最后访问时间发现文件。

```
$ find . - mtime 10 - print [Return].....查找和显示 10 天前修改的文件
```

这条命令显示那些 10 天前被更改的文件。

```
$ find . - mtime - 10 - print [Return].....查找和显示少于 10 天以前修改过的文件
$ find . - mtime + 10 - print [Return].....查找和显示多于 10 天以前修改过的文件
```

-newer 选项:寻找比指定文件名更近更新的文件。

【实例】

看如下的例子。

```
$ find . - newer first.c - print [Return].....寻找比 first.c 更新的文件
```

动作选项

动作选项指示 find 命令发现文件之后如何操作。表 7.8 总结了三个动作选项。

表 7.8 find 命令的动作选项

操作符	描述
- print	显示每个发现的文件的路径名
- exec command \;	对找到的文件执行 commands 命令
- ok command \;	执行命令前请求确认

【实例】

从用户主目录开始寻找名为 first.c 的文件。

```
$ find $HOME - name first.c - print [Return]...寻找 first.c 并显示路径名
/usr/david/first.c
/usr/david/source/first.c
/usr/david/source/c/first.c
$_.....回到命令提示符
```

【说明】

在 david 层次目录中,输出显示有三个文件 first.c 的实例。

-exec 选项:允许给出一个命令,作用于发现的文件。输入 -exec 后跟指定命令、空格、反斜杠和分号。用户能用一对大括号({})代表发现的文件名。下面的例子可以说明用法。

【实例】

寻找并删除所有大于90天名为 first.c 的文件实例。

```
$ find . -name first.c -mtime +90 -exec rm {} \; [Return]
$_.....回到命令提示符
```

搜索从当前目录开始(.表示),继续遍历下级目录。**find** 命令定位并删除所有大于90天名为 first.c 的文件。

用到两个查找选项: **-name** 和 **-mtime**, 意味着搜寻同时满足这两类条件的文件。

【注意】

这条命令由许多部分组成,语法比较特殊:

1. **-exec** 选项后跟命令(这里是 **rm**)
2. 一对大括号({})后跟空格
3. 反斜杠(\)后跟分号

所有 first.c 的实例都被删除了。没有警告或者反馈信息显示。看到 \$ 命令提示符时,上述工作已经完成。

-ok 选项:与 **-exec** 选项基本相同,但是在执行命令前请求用户确认。

【实例】

寻找并删除所有 first.c 的文件实例,但是在删除之前请求确认。

```
$ find . -name first.c 190 -ok rm {} \; [Return]
$_.....回到命令提示符
```

如果文件满足条件,会显示如下的提示:

```
<rm ... ./source/first.c> ?
```

如果回答[y]或者[Y],命令被执行(这里,first.c 被删除);否则,文件保持不变。

【说明】

可以用逻辑操作符 **or**、**and** 和 **not** 来结合查找选项。查找从当前目录开始,遍历下级目录。

7.6.2 显示文件头部:head 命令

用 **head** 命令显示指定文件的头部,这样,用户能很快捷地查看文件的开始几行。例如,显示当前目录中名为 MEMO 的文件的开始部分,输入:

```
$ head MEMO [Return]
```

默认情况,显示指定文件的前 10 行。用户能够指定行数。例如,显示 MEMO 文件的头 5 行,输入:

```
$ head - 5 MEMO [Return]
```

【说明】

指定的行数必须是正整数。

用户可以在命令行中指定多于一个的文件名。例如,显示文件 MyFile、YourFile 和 OurFile 的前 5 行,输入:

```
$ head MyFile YourFile OurFile [Return]
```

如果多于一个文件被指定,每个文件的开始如下:

```
==> filename < + ==
```

【实例】

下面的命令说明 head 的用法。

```
$ head -5 *File [Return].....显示当前目录以 File 结尾的文件名的前 5 行
$ head * [Return].....显示当前目录所有文件的前 10 行
$ head -15 MEMO [Return].....显示 MEMO 的前 15 行
```

7.6.3 显示文件尾部:tail 命令

tail 命令用来显示指定文件的尾部。用户能快捷地查看文件内容。例如,显示当前目录 MEMO 文件的最后几行,输入:

```
$ tail MEMO[Return]
```

默认情况,显示指定文件的最后 10 行。用户能够通过下面几个参数,改变默认值。

tail 选项:表 7.9 总结了 tail 的选项,使得 tail 命令以文件块、字符或者行为单位来计算。

表 7.9 tail 命令选项

选 项	功 能
b	以文件块为单位来计算
c	以字符为单位计算
l	以行为单位计算

【说明】

1. 如果选项前有加号,tail 从文件开始计算。
2. 如果选项前有连字符,tail 从文件末尾计算。
3. 如果选项前有数字,tail 以该数字代替默认的 10 行计数。

【实例】

下面的例子说明如何使用 tail 命令。

```
$ tail MEMO [Return].....显示最后 10 行(无参数)
$ tail -4 MEMO [Return] .....显示最后 4 行
$ tail -10c MEMO [Return].....显示最后 10 个字符
```

【说明】

只能指定一个文件作为参数。

7.6.4 选择文件的一部分:cut 命令

cut 命令能从文件中“切掉”指定的列(或域)。许多文件是记录的集合,每个记录由几个域组成。用户可能只对文件中的某些列或域感兴趣。图 7.8 显示一个名为 phones 文件的例子。每个记录由 5 个域组成,每个域用空格或者制表符分开。

```
$ cat phones
David Back      (909) 999999      dave@xyz.edu
Daniel Knee     (808) 888888      dan@xyz.edu
Gabe Smart      (707) 777777      gabe@xyz.edu
$
```

图 7.8 phones 文件

cut 选项

假设 phones 文件包含姓名、电话号码和电子邮件,与图 7.8 相同。表 7.10 总结了 cut 命令的选项。下面的命令说明 cut 及其选项的用法。

表 7.10 cut 命令选项

选 项	功 能
-f	指定域位置
-c	指定字符位置
-d	指定域分隔符

-f 选项: -f 后指定域列表。文件中的域被认为用分隔符(默认为制表符是 tab)分开。例如, -f 1 表明第 1 个域, -f 1,7 表明第 1 和第 7 个域。

【实例】

利用带 -f 选项的 cut 命令显示 phones 文件的第 1 个域。

```
$ cut -f 1 phones [Return].....显示 phones 的第 1 个域
David Back
Daniel Knee
Gabe Smart
$_.....显示命令提示符
```

【说明】

需要注意,默认的域分隔符是 tab 键。

显示 phones 文件的第 1 和第 3 个域。

```
$ cut -f 1,3 phones [Return]...显示 phones 的第 1 和第 3 个域
David Back dave@xyz.edu
Daniel Knee dan@xyz.edu
Gabe Smart gabe@xyz.edu
$_.....命令提示符
```

-c 选项: -c 后指定字符位置。例如, -c 1-10 表示每行的前 10 个字符。

【实例】

显示 phones 文件每行的前 4 个字符。

```
$ cut -c 1-4 phones [Return]....显示 phones 的前 4 列
Davi
Dani
Gabe
$_.....显示命令提示符
```

cut 命令可以不带文件名。

```
$ date | cut -c 12-13 [Return]....没有指定文件名
18
$_.....命令提示符
```

【说明】

date 命令的输出(日期字符串)传递给 cut 命令。cut 命令显示日期字符串的第 12 和第 13 个位置的字符,恰好是“小时”域。

-d 选项: -d 后跟域分隔符。默认字符是 tab。空格或者其他特殊意义的字符必须括在双引号中。分隔符分开文件的不同域。

【实例】

用空格键作为分隔符。

```
$ cut -d " " -f 1 phones [Return]....显示 phones 的第 1 个域
David
Daniel
Gabe
$_.....显示命令提示符
```

【注意】

如果域分隔符是空格字符,确保被括在双引号中。

7.6.5 连接文件:paste 命令

paste 命令把文件一行接一行地连接在一起,或者把两个或多个文件的域连到一个新文件里。图 7.9 显示了 paste 命令的输出,假设当前目录有两个文件 first 和 last。

paste 选项

-d 选项: 指定分隔符。默认是制表符 tab 键。

【实例】

指定:(冒号)作为分隔符。

```
$ paste -d: first last [Return].....用:作为分隔符
David:Back
Daniel:Knee
```

```

Gabriel:Smart
$_..... 回到命令提示符

$ cat first
David
Daniel
Gabriel
$ cat last
Back
Knee
Smart
$ paste first last
David      Back
Daniel     Knee
Gabriel    Smart
$

```

图 7.9 paste 命令

用空格键作为分隔符。

```

$ paste -d" " first last [Return]...用:作为分隔符
David Back
Daniel Knee
Gabriel Smart
$_..... 回到命令提示符

```

【注意】

如果分隔符是空格,要用双引号括起来。

7.6.6 另一个页查看工具:more 命令

为了用户方便,有另一个按页面查看的命令 **more**。像 **pg** 一样,用户能使用 **more** 对文本文件浏览翻页。在每页(屏)之后暂停,显示“More”,以及到屏幕底部为止所显示字符的百分比。

```
More - ( 11%)
```

【说明】

为了保持屏幕的连续,**more** 在屏与屏之间有两行重叠。

more 选项

空格键:按空格键光标翻一屏。

回车键:按回车键向前滚一行。

7.7 UNIX 内部:文件系统

UNIX 文件系统如何找到用户文件呢?它怎么知道磁盘上文件的位置呢?从用户观点来

看,创建目录来组织磁盘空间,目录和文件由文件名来确定。目录和文件的层次结果是文件系统的逻辑观点。然而,在 UNIX 内部是用不同的方式来组织磁盘和查找文件的。

UNIX 文件系统把每个文件名与一个 i 节点号相联系,用 i 节点号确定每个文件。UNIX 把所有这些 i 节点号维护在一个列表中,称作 i 节点列表。这个列表保存在 UNIX 磁盘上。

7.7.1 UNIX 磁盘结构

UNIX 里,磁盘是一个标准块设备,一个 UNIX 磁盘被分成四个块(区域):

- 主引导块(boot block)
- 超级块(super block)
- i 节点列表块(i-node list block)
- 文件和目录块

主引导块:主引导块保存着引导程序,系统启动时激活这段程序。

超级块:超级块包含磁盘自身的信息。这些信息包括:

- 磁盘的总块数
- 空闲块数
- 块大小(按字节计算)
- 使用的块数

i 节点列表块:i 节点列表块维护着 i 节点的列表。列表的每个条目是一个 64 字节存储区的 i 节点。规则文件或目录文件的 i 节点包含着所在磁盘块的位置。特殊文件的 i 节点包含确定外围设备的信息。i 节点还包含着其他如下信息:

- 文件访问权限(读、写和执行)
- 所有者与组 id
- 文件链接数
- 文件最后更改时间
- 最后访问时间
- 每个规则文件或目录文件的块位置
- 特殊文件的设备 id 号

【说明】

i 节点是顺序计算的。

i 节点和目录

2 号 i 节点记录了包含着根目录(/)的块的位置。UNIX 目录包含着文件名列表和与之相联系的 i 节点号。当创建一个目录时,自动生成两项,一个是..(双点,表示父目录),另一个是.(点,表示该目录)。

【说明】文件名被存储在目录中,而不是在 i 节点里。

7.7.2 整体过程

登录时,UNIX 读取根目录(i-node 2),找出用户主目录(home directory),存储用户主目录的 i 节点号。当用 `cd` 改变目录时,UNIX 用新目录的 i 节点号进行替换。

用系统工具或命令(例如 `vi` 或 `cat`)访问文件或者某个程序打开文件时,UNIX 查找指定文件名的目录。每个文件名与 i 节点列表中的一个 i 节点相联系。UNIX 通过用户工作目录的 i 节点开始搜索,但是如果用户给定了全路径名,则从根目录(i-node 2)开始查找。

假设当前目录是 `david`,有一个子目录 `memos`,`memos` 中有个文件 `report`,现在要访问 `report`。UNIX 开始从当前目录 `david`(有已知的 i 节点)查找,得到文件名 `memos` 和它的 i 节点。接着,系统从 i 节点列表中读取 `memos` 的 i 节点记录。`memos` 的 i 节点表明了包含 `memos` 目录的文件块。

通过查找包含 `memos` 下文件名的文件块,UNIX 找到 `report` 文件名和它的 i 节点。UNIX 重复上述过程,从 i 节点列表中读取 i 节点记录。这条记录的信息包含了磁盘上 `report` 所在文件块的位置(见图 7.10)。

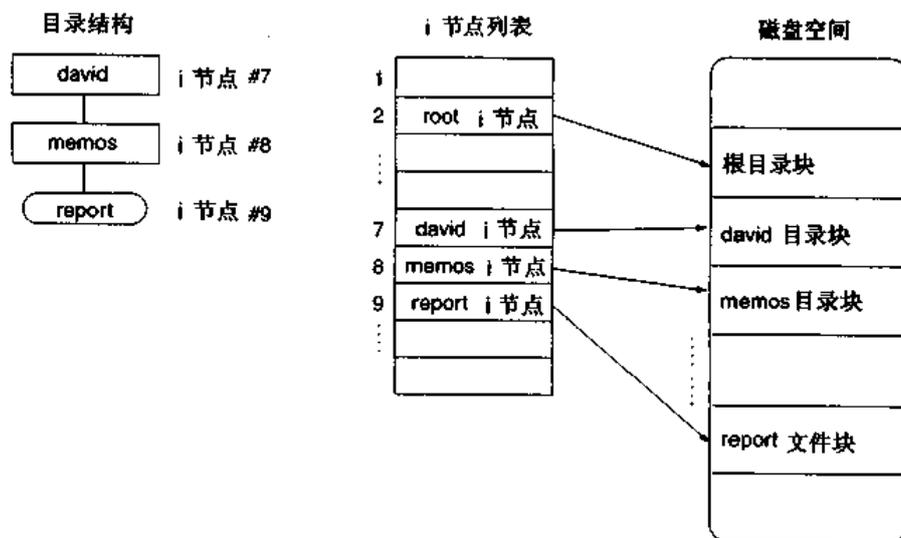


图 7.10 目录结构和 i 节点列表

用户如何知道一个文件的 i 节点号呢? 可以用带 `-i` 选项的 `ls` 命令。例如,假设工作目录是 `david`,有一个子目录 `memos` 和文件 `report`。

【实例】

列出当前目录中的文件名及其 i 节点号。

```
$ ls -i
4311      memos
7446      report
$_
```

复制 `report`, 名为 `report.old`, 然后显示 i 节点号。

```
$ cp report report.old
$ ls -i
```

```

4311      memos
7446      report
7431      report.old
$

```

report.old 的新的 i 节点号说明建立了新文件, 一个新的 i 节点与之相连。
把 report.old 移动到 memos 目录, 显示 i 节点号。

```

$ mv report.old memos
$ ls -i
4311      memos
7446      report
$ ls -i memos
7431      report.old
$

```

report.old 移动到 memos 目录中; 它的 i 节点保持不变, 但是现在与 memos 目录相连。
把 memos 目录中的 report.old 更名为 report.sav。

```

$ mv memos/report.old memos/report.sav
$ ls -i
4311      memos
7446      report
$ ls -i memos
7431      report.sav
$

```

report.sav 的 i 节点保持不变; 仅仅是与之相连的名称发生了改变。

链接 report 到一个新文件名 rpt(创建 report 的另一个文件名), 查看 i 节点的情况:

```

$ ln report memos/rpt
$ ls -i
4311      memos
7446      report
$ ls -i memos
7431      report.sav
7446      rpt
$

```

新文件名 rpt 的 i 节点号与 report 相同。report 和 rpt 的 i 节点都指向同一个文件块。

命令小结

下面是本章讨论的命令及其选项。

cut	
从文件中“切掉”指定的列(或域)。	
选 项	功 能
-f	指定域位置
-c	指定字符位置
-d	指定域分隔符

head

显示指定文件的头部,这样,用户能很快捷地查看文件的开始几行。可以在命令行中指定行数和多于一个的文件名。

paste

一行接一行地把文件连接在一起,或者把两个或多个文件的域连到一个新文件。

pg

一次查看一屏文件内容。提示符显示在屏幕底部时,可以输入参数或其他命令。

选 项	功 能
-n	不需要用回车键来完成单字母命令
-s	用反白显示消息和提示符
-num	设置每一屏的行数为 num,默认值是 23 行
-p str	把提示符:(冒号)改成设定的字符串 str
+ line - num	从设定的 line - num 行开始显示文件
+/pattern	从包含设定模式的第一处开始显示

pg 命令操作符

显示提示符时,可以输入以下键。

关 键 字	功 能
+n	前进 n 屏,n 是一个整数
-n	后退 n 屏,n 是一个整数
+nl	前进 n 行,n 是一个整数
-nl	后退 n 行,n 是一个整数
n	前进到 n 屏,n 是一个整数

cp

把文件从某个目录复制到其他地方。

选 项	功 能
-i	如果目标文件存在,请求确认
-r	复制目录到新的目录

mv

把文件从一个地方移动到别处。

ln

在存在的文件和一个新文件名之间建立新链接,使文件有多于一个的文件名。

pr

在显示或者打印之前,格式化文件。

选 项	功 能
+ page	从指定页开始显示。默认是第 1 页
- columns	用指定的列数显示输出。默认是 1 列
- a	以横跨页面的方式显示输出,每列一行
- d	双倍行距显示输出
- h str	用指定字符串 str 代替页眉的文件名
- l number	用指定的行数设置页长。默认值是 66 行
- m	用多列显示所有指定文件
- p	在每页末暂停,并响铃
- s character	用单个指定的字符分割列。如果没有指定字符,那么用 [tab]
- t	取消 5 行页眉和 5 行页脚
- w number	用指定字符数设置行宽。默认值为 72

wc

计算一个文件或者指定的多个文件中的行数、单词数和字符数。

选 项	功 能
- l	报告行数
- w	报告单词数
- c	报告字符数

more

一次一屏显示文件。对于看大文件很有用。

find

在目录层次中定位与一组给定的标准相匹配的文件。动作选项指示 find 命令发现文件之后如何操作。

查找选项	描 述
- name 文件名	寻找给定 filename 的文件
size ± n	寻找大小 n 的文件
- type 文件类型	寻找具有给定访问模式的文件
- atime ± n	寻找 n 天以前访问的文件
- mtime ± n	寻找 n 天以前更改的文件
- newer 文件名	寻找比 filename 更近更新的文件

动作选项	描 述
- print	显示每个发现的文件的路径名
- exec command \;	对找到的文件执行 command 命令
- ok command \;	执行命令前请求确认

tail

显示指定文件的尾部。用户能快捷地查看文件内容。选项灵活地设定查看方式。

选 项	功 能
b	以文件块为单位来计算
c	以字符为单位计算
l	以行为单位计算

习题

1. 重定向操作符的符号是什么?
2. 解释输入和输出重定向。
3. 读文件用什么命令?
4. 移动(mv)和复制(cp)文件之间的差别是什么?
5. 重命名文件的命令是什么?
6. 一个文件可以有多于一个的文件名吗?
7. UNIX 系统的 4 个区域(块)是什么? 请分别解释。
8. i 节点号是什么? 怎样用来定位文件?
9. i 节点列表是什么? 每个节点中存储着什么主要信息?
10. 下面哪个命令改变或创建一个 i 节点号?

- a. mv file1 file2
- b. cp file1 file2
- c. ln file1 file2

11. 定位文件,一旦发现即删除该文件的命令是什么?
12. 查看一个文件最后 10 行的命令是什么?
13. 从文件选择指定域的命令是什么?
14. 一行接一行把文件内容放在一起的命令是什么?
15. 一次一屏读文件的命令是什么?

把左列的命令与右列的解释相匹配。

- | | |
|---------------|-------------------------------|
| 1. wc xxx yyy | a. 复制 xxx 到 yyy |
| 2. cp xxx yyy | b. 重命名 xxx 为 yyy |
| 3. ln xxx yyy | c. 复制所有以 file 开始其后跟 2 个字符的文件名 |
| 4. mv xxx yyy | d. 删除当前目录的所有文件 |

- | | |
|-------------------------------------------------|---------------------------------------|
| 5. <code>rm *</code> | e. 创建 xxx 的另一个文件名 yyy |
| 6. <code>ls * [1-6]</code> | f. 显示 myfile 的内容 |
| 7. <code>cp file?? source</code> | g. 复制 myfile 到 yyy |
| 8. <code>pr -2 myfile</code> | h. 把所有文件名为 file 前加一个字符的文件内容加入 yyy 文件中 |
| 9. <code>ls -i</code> | i. 以 2 列格式化 myfile |
| 10. <code>pg myfile</code> | j. 列出所有以数字 1 到 6 结尾的文件名 |
| 11. <code>cat myfile</code> | k. 列出当前目录文件名及其 i 节点 |
| 12. <code>cat myfile > yyy</code> | l. 创建名为 yyy 的文件包含 xxx 的字符数 |
| 13. <code>cat ?file >> yyy</code> | m. 一次一屏查看 myfile |
| 14. <code>find . -name "file *" - print</code> | n. 把 xyz 的第 2 列保存到 xxx 中 |
| 15. <code>find . -name xyz -size - print</code> | o. 一次一屏读 zzz |
| 16. <code>cut -f2 xyz > xxx</code> | p. 寻找所有名为 xyz 大小为 20 个文件块的文件 |
| 17. <code>more zzz</code> | q. 显示所有文件名以 file 开头的文件 |

上机练习

本章的上机实验,用户练习本章讨论的命令,创建目录并管理目录中的文件。

1. 在用户主目录下创建名为 memos 的目录。
2. 用 vi 编辑器,在用户主目录中创建名为 myfile 的文件。
3. 用 cat 命令,把 myfile 多次附加到新创建的名为 large 的文件中。
4. 用 pg 命令及其选项,查看 large 文件。
5. 用 pr 命令及其选项,格式化 large 文件并打印。
6. 用 cp 命令把用户主目录下的所有文件复制到目录 memos 下。
7. 用 ln 命令创建 large 的另一个文件名。
8. 用 mv 命令,把 large 更名为 large.old。
9. 用 mv 命令,把 large.old 移到 memos 中。
10. 执行下面的命令时,用 ls 命令及其 -i 和 -l 选项观察 i 节点数与链接数的改变。
 - a. 改变到 memos 目录。
 - b. 创建 myfirst 的别名,成为 MF。
 - c. 复制 myfirst 到 myfirst.old。
 - d. 列出所有文件名以 my 开头的文件。
 - e. 列出所有扩展名为 old 的文件。
 - f. 更改 myfile,观察 MF 文件;myfile 的更改也使得 MF 发生改变。
 - g. 改变到用户主目录。

- h. 删除 memos 目录中所有文件名中有 file 字符串的文件。
 - i. 删除 memos 目录中的文件。
 - j. 列表用户主目录。
 - k. 删除以上操作生成的所有文件。
11. 显示一个文件的后 5 行。
 12. 显示一个文件的前 5 行。
 13. 把一个文件的最后 30 个字符存到另一个文件中。
 14. 保存用户主目录中所有 7 天前创建的文件列表。
 15. 查找名为 passwd 的文件。
 16. 查找名为 profile 的文件。
 17. 从用户主目录开始,查找所有 7 天前创建的文件。
 18. 从用户主目录开始,查找所有少于 7 天的文件。
 19. 查找所有多于 10 天的文件,保存到另一个目录中。
 20. 创建与图 7.8 和图 7.9 相同的文件。
 - a. 用 cut 命令,切出指定的域和列。
 - b. 用 paste 命令,把两个文件连在一起。
 21. 用 more 命令读大文件。

第 8 章 探索 shell

本章描述 shell 及其在 UNIX 系统中的角色,并解释它的特征和功能;讨论 shell 变量并解释变量的使用和定义方法;还介绍一些 shell 元字符以及使 shell 忽略这些元字符特殊含义的方法;此外,还解释了 UNIX 启动文件、进程和进程管理。本章继续介绍一些新命令(系统工具),以使用户可以建立起自己的 UNIX 命令词汇表。

8.1 UNIX shell

UNIX 操作系统由两部分组成:内核和应用。内核是 UNIX 系统的核心并且驻留内存(即它在系统启动时调入内存,直至系统关闭)。所有日常事务,诸如直接与硬件通信等,都由内核完成。这些事务与操作系统其他部分相比是较小的。

除内核外,其他一些基本模块也是驻留内存的。这些模块完成重要的功能,如输入/输出管理、文件管理、内存管理和处理器时间管理。此外,UNIX 为内务处理保留部分内存驻留表,以记录系统状态。

UNIX 系统的其他部分保存在磁盘中,在需要时调入内存。绝大多数 UNIX 命令是保存在磁盘上的程序(称为系统工具)。在用户键入一个命令时(请求程序执行),相应的程序即被调入内存。

用户与操作系统通过 shell 进行通信,而基于硬件的操作由内核完成。图 8.1 为 UNIX 操作系统的构成。

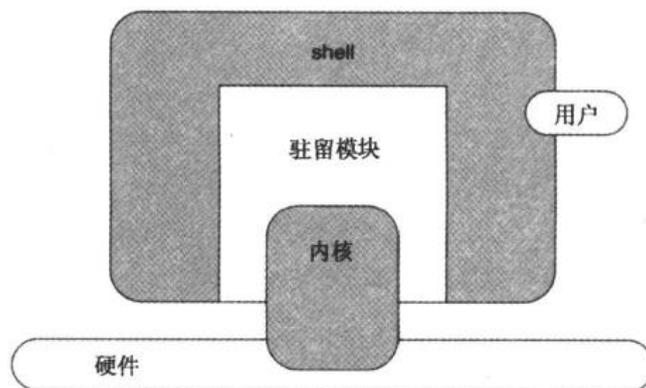


图 8.1 UNIX 系统组成部分

shell 本身是一个程序(一个系统工具),它在用户登录到系统上时即调入内存。当 shell 准备好接收命令时,它显示一个命令提示符。对大多数用户键入的命令,shell 并不执行,而是检查每个命令并启动相应的程序(系统工具)来完成请求的动作。shell 决定启动哪个程序(程序的名字与用户所键入的命令相同)。例如,用户键入 `ls` 并按回车键来列出当前目录的文件,shell 找到并启动名为 `ls` 的程序。shell 以相同的方式处理用户的系统工具:用户键入程序的名

字作为命令, shell 运行该程序。图 8.2 为用户与 shell 的交互。

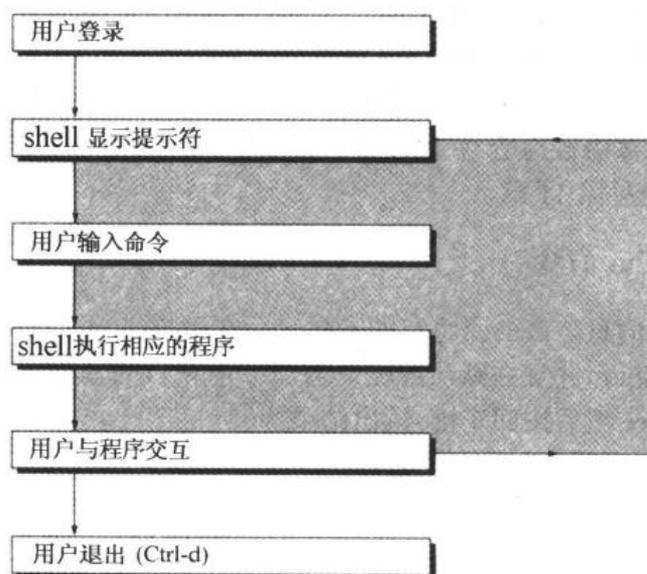


图 8.2 用户与 shell 交互

shell 也包含一些内部命令。这些命令是 shell 自身的一部分,并被 shell 在内部解释并执行。读者已经了解了几个内部命令(如 `cd`、`pwd` 等)。

标准 UNIX 系统包含多达 200 个系统工具。其中有一个是 `sh`,就是 shell 自身。

8.1.1 理解 shell 的主要功能

shell 是 UNIX 中最常用到的系统工具。它是一个复杂的程序,用来管理用户与 UNIX 系统间的对话。在工作会话中,用户重复地与 shell 交互。shell 是一个规则的可执行 C 程序,通常存放在 `/bin` 目录下。最常用的一个 shell 程序是 Bourne shell,以其开发者命名,放在 `/bin` 目录下,程序名为 `sh`。当用户登录后,一个交互式的 Bourne shell 被自动调用。用户可以通过在 `$` 命令提示符下键入 `sh` 来调用 Bourne shell 的副本。

shell 包含如下主要特征。读者对其中的部分应该已经熟悉,其他的特征将在本章研究。

命令执行:命令(程序)执行是 shell 的一个主要功能。几乎用户在命令提示符下键入的所有字符都被 shell 解释。当用户在命令行未按回车键时,shell 开始分析用户的命令;如果有文件名替换符或输入/输出重定向符,shell 将注意到并运行相应的程序。

文件名替换:如果命令行中出现文件名替换(也叫文件名生成),shell 首先完成替换,然后执行程序。shell 程序自身不包括在替换进程中(关于文件名通配符 `*` 与 `?` 的讨论见第 7 章)。

I/O 重定向:输入/输出重定向由 shell 进行处理。同样,shell 程序自身不包括在内,并且重定向在程序运行前建立。如果输入或输出重定向定义在命令行,shell 打开该文件并将它连接到程序相关的标准输入或标准输出。有关这方面的讨论见第 7 章。

管道:管道(也写作管线)使用户把简单程序连接到一起来完成一个较为复杂的任务。键盘上的竖线“`|`”就是管道操作符。

环境控制:shell 可使用户定义适合自己需要的环境。通过设定相应的变量,用户可以变更自己的主目录、命令提示符或工作环境的其他方面。

后台计算 :shell 的后台计算能力使用户能够在前台进行工作的同时,在后台运行程序。这一功能有利于费时较多的、非交互的程序。

shell 脚本:按通常顺序使用的 shell 命令可存放于被称作 shell 脚本程序的文件中。这种文件的名字可在其后被用来运行其中所保存的程序,以使用户能够使用一个命令来运行所保存的所有命令。shell 还包含语言构造器,以使用户建立能够完成复杂功能的 shell 脚本文件。有关 shell 脚本文件的讨论见第 11 章。

8.1.2 显示信息:echo 命令

echo 命令用来显示信息。该命令将它的参数显示在用户终端——标准输出设备上。如果没有参数,它产生一个空行,并且在默认情况下增加一个新行到输出。例如,如果在命令提示符下键入 **echo hello there** 然后按回车键,输出结果为:

```
hello there
```

【说明】

参数字符串长度可以是任意长。但是,如果字符串包含元字符,则该字符串必须用引号引起来(这一问题将在本章进一步讨论)。

表 8.1 为可以用作字符串的一部分、用来控制消息格式的字符。这些字符之前都有一个反斜杠 \, 并且被 shell 截取以产生所希望的输出。它们也称作扩展符。

表 8.1 扩展符

扩展符	含 义
\n	回车并换新行
\t	制表符
\b	退格
\r	回车,但不换新行
\c	禁止回车

【说明】

反斜杠本身是一个 shell 元字符。因此,如果要把它用在字符串中,必须用引号引起来。

【实例】

以下命令依次显示了如何使用 echo 命令以及在参数串中使用扩展符后的结果。

```
$ echo Hi, this is a test. [Return].....在屏幕上显示一个简单的信息
Hi, this is a test.
$ echo Hi, "\n" this is a test.[Return].....用 2 行显示同样的信息
Hi,
this is a test.
$_.....命令提示符
```

【注意】

\n 必须用引号引起来,以被解释为下一行命令。

```
$ echo Hi, "n" this is a test. > test [Return].....输出到一个文件.
```

```

$ cat test [Return].....确认 test 文件的内容
Hi,
this is a test.
$_..... 命令提示符
$ echo Hi," \n" this is a test." \c" [Return]..... 这次在信息的末尾不产生新行
Hi,
this is a test.$_

```

命令提示符 **\$** 紧跟在 `test` 之后。这是命令参数串中 `\c` 的功能。

```

$ echo This is a test. [Return].....看一下如果加空格会怎样
This is a test.

```

shell 将这一命令行解释成有 4 个参数,并且在输出时每个参数之间以一个空格分开。

```

$ echo "This is a test." [Return].....看一下引号的奇妙效果;这一次字间
空格被保留下来
This is a test.
$_..... 命令提示符

```

8.1.3 消除元字符的特殊含义

shell 元字符对 shell 来说有特殊含义,但有些时候,用户希望忽略这些特殊含义。shell 提供了一系列字符来消除元字符的含义。这种消除元字符特殊含义的过程称为引用或转义。引用字符有:

- 反斜杠 \
- 双引号 "
- 单引号 '

【说明】

在后面的例子中,多数使用了 `echo` 命令来示范这一过程是如何工作的。但是,当用户需要使用这些特殊字符作命令参数时,引用的使用也是同样适用的。

反斜杠:反斜杠 \ 用来将紧跟其后的单字符解释为一个普通的字母数字符号。例如? 是一个文件替换符(通配符),并且对 shell 有特殊含义,但是 \? 被解释成问号本身。

【实例】

从当前目录删除一个名称为 `temp?` 的文件。

```

$ rm temp? [Return].....删除 temp?

```

shell 将此命令解释为删除所有名称由 `temp` 和其后紧跟一个任意字符的所有文件。因此,shell 删除所有符合这种方式的文件,如 `temp`、`temp1`、`temp2`、`tempa`、`temp0` 等。

但是,如果用户只想删除一个名为 `temp?` 的文件,那么可以尝试如下命令:

```

$ rm temp \? [Return] ..... 再试一次,用 \? 代替?

```

这次 shell 扫描命令行,发现 \,忽略问号的特殊含义并将文件名 `temp?` 送给 `rm` 程序。

【实例】

显示元字符。

```
$ echo \< > \ \ " \ ' \ $ \ ? \ & \ | \ \ [Return].....显示所有元字符
< > " ' $ ? & \
$_.....准备接收下一个命令
```

【说明】

要消除反斜杠的特殊含义,通过在它前面加一个反斜杠来实现。

双引号:使用双引号可以忽略大多数字符的特殊意义。除 \$、单引号和双引号外,任何字符被括在一对双引号内都会失去特殊含义。要想消除上述三种符号(\$、"和')的特殊含义,可以使用反斜杠。

双引号还保留空白字符(如空格、制表符 Tab 和换行符)。双引号的这种用途适用于 echo 命令。

【实例】

以下命令显示双引号的使用。

```
$ echo > [Return].....显示">"符号
syntax error: 'newline or ;' unexpected
$_.....命令提示符
```

shell 将命令解释为将 echo 命令的输出重定向到一个文件。它寻找文件名,并且由于文件名未定义而返回一个隐含错误信息。

```
$ echo ">" [Return] .....将参数用双引号括起,则 > 被显示
>
$ ls -c [Return].....检查当前目录文件
memos myfirst REPORTS
$ echo * [Return].....使用一个元字符作为参数
memos myfirst REPORTS
```

shell 用当前目录的所有文件替换 *。

```
$ echo "*" [Return] .....现在使用双引号
*
$_.....命令提示符
```

在双引号中不发生替换,因此 * 的特殊含义被消除了。

【实例】

显示信息"The Unix System"。

```
$ echo \"The UNIX System\" [Return]
"The UNIX System"
```

【说明】

用于隐藏双引号的特殊含义时,双引号前的反斜杠是必需的。

单引号:单引号'的作用与双引号非常相似。除了单引号外,其他在一对单引号中的任何

符号的特殊含义都会被消除(要消除单引号的特殊含义使用反斜杠 \)。

单引号同样保留空白符。包含在单引号中的字符串变成 1 个单个的参数,并且空格也失去了它作为参数分隔符的含义。

【注意】

在这种用途下,开引号与闭引号都使用相同的前引号',而不是后引号。这一点至关重要。shell 把后引号解释为一个可执行的命令。

【实例】

用单引号显示特殊字符。

```
$ echo '< >' '$?&!' [Return] .....使用 echo 命令和单引号
< >' '$? & !
$_ .....命令提示符
```

字符间的空格也被保留。

8.2 shell 变量

shell 程序处理用户接口,并担任命令解释程序的角色。为了完成用户请求(运行命令、操纵文件等),shell 需要某些信息并且理解这些信息(如用户主目录、终端类型和命令提示符等)。

这些信息存放于 shell 变量中。变量是用户用来控制或自定义个人环境的特定值。UNIX 支持两种变量:环境变量和局部变量。

环境变量:环境变量也叫标准变量,拥有为系统所知道的变量名。环境变量被用于定义系统基本的特征,并且通常由系统管理员定义。例如,标准变量 TERM 被指定为用户终端类型:

```
TERM = vt100
```

局部变量:局部变量是用户定义的,并完全由用户控制。用户可以根据自己的意愿定义、更改或删除局部变量。

8.2.1 显示和清除变量:set 和 unset 命令

使用 set 命令可以查看当前使用的 shell 变量。

【实例】

在命令提示符下键入 set 后按回车键,shell 显示变量列表。读者所见到的变量列表应当与图 8.3 相似,但不完全一样。

图 8.3 中,等号左侧的标准变量名用大写字母表示,但这并不是要求的,用户可以使用大写、小写或混合方式来表示。

在等号右侧是赋给变量的值。用户在变量名中可以使用字母、数字和下划线,但第一个字符必须为字母,而不能是数字。

```
$ set
HOME = /usr/students/david
IFS =
LOGNAME = david
LOGTTY = /dev/tty06
MAIL = /usr/mail/students/david
MAILCHECK = 600
PATH = :/bin:/usr/bin
PS1 = "$"
PS2 = ">"
TERM = wyse50
TZ = EST = EDT
$_
```

图 8.3 set 命令的输出

【注意】

在指定一个变量时,必须使用准确的变量名(区分大小写字母)。

用户可以用 `unset` 命令来消除不想要的变量。如果有一个变量: `XYZ = 10`, 要想清除该变量只需键入: `unset XYZ`, 然后按回车键。

8.2.2 给变量赋值

用户可以创建自己的变量,也可以修改标准变量。用户可以通过在变量名后跟等号,之后跟要赋的值,如下所示:

```
age = 32
```

或

```
SYSTEM = UNIX
```

shell 对所有分配给变量的值按字符对待。在前面的例子中,变量 `age` 的值是串 32, 而不是数字 32。如果变量值字符串中包含空白字符(空格、制表符等),必须用双引号把整个串引起来,如下所示:

```
message = "Save your files, and log off" [Return]
```

【注意】

1. 变量名必须以字母(大写、小写均可)开始,而不能是数字。
2. 在等号两边都不能出现空格。

8.2.3 显示 shell 变量的值

需要访问 shell 变量时,必须在变量前加 `$` 符号。对前边的例子, `age` 是变量名, `$age` 是 32——保存在变量中的值。

也可使用 `echo` 命令来显示已分配值的 shell 变量的值。

【说明】

`set` 命令显示变量列表;`echo` 命令显示一个指定变量。

【实例】

使用 `echo` 命令显示 shell 变量的内容或值。

```
$ age = 32 [Return].....给变量 age 赋值 32
$ echo Hi, nice day. [Return] ..... 显示参数串
Hi, nice day.
$ echo age [Return] ..... 显示参数,单词 age
age
$ echo $age[Return].....现在参数是 $age,值存储在 age 中
32
$ echo You are $age years old. [Return] .... 加一些文本以获得一些有意义的输出
You are 32 years old.
$_..... 为下一个命令做好准备
```

【实例】

shell 变量频繁用作命令行的变量参数,如下所示:

```
$ all = -lFa [Return] .....创建一个变量 all,给其赋值为-lFa
                        (连字符,小写字母 l,大写字母
                        F 和小写字母 a)
$ ls $all myfirst [Return] .....将该变量用作命令行的一部分
command's output
$_.....显示命令提示符
```

变量名被 `$` 掩盖。于是,shell 用存在 `all` 中的值 `-lFa` 替换变量 `all`。替换后,命令变成了 `ls -lFa myfirst`。

【实例】

观察下列命令的输出。可以发现双引号中的变量在被不同方式解释时的微妙差别。

```
$ age = 32 [Return].....把 32 赋给变量 age
$ echo $age "$age" '$age' [Return].....显示变量 age
32 32 $age
$_..... 命令提示符
```

8.2.4 理解标准 shell 变量

赋给标准 shell 变量的值通常由系统管理完成。因此,当用户登录时,shell 通过这些变量来得知用户的环境信息。用户可以改变这些变量的值,但是这种改变是临时的,并且只适用于当前会话。当用户下次登录时,还要重新进行设置。如果希望长久改变,则把这些变量放到一个名为 `.profile` 的文件中。有关 `.profile` 文件将在本章后面解释。

HOME 变量

用户成功登录后,shell 把用户主目录的完整路径赋给变量 `HOME`。`HOME` 变量被多个 UNIX 命令用来定位主目录。例如,`cd` 命令在没有参数时检查 `HOME` 变量来决定主目录的路径,并设置系统当前目录为用户的主目录。

【实例】

体验 HOME 变量,试试下面命令。

```

$ echo $HOME [Return].....显示用户主目录的路径
/usr/david
$ PWD [Return] .....显示当前目录的路径
/usr/david/source source      david 中的 source 子目录
$ cd [Return].....没有参数,默认为用户的主目录
$ pwd [Return].....检查用户当前目录。当前位于 david
                               目录,即用户的主目录
/usr/david
$ HOME = /usr/david/memos/important [Return]...改变用户主目录路径。现在用户的主目
                                               录是 important
$ cd [Return].....改变到用户主目录
$ pwd [Return].....显示用户当前目录,即 important
/usr/david/memos/important
$ _ .....命令提示符

```

IFS 变量

内部字段分隔符变量是一串被 shell 解释为命令行元素分隔的字符。例如,为了得到当前目录的长列表,用户键入 `ls -l`,然后按回车键。处于 `ls` 与 `-l` 之间的空格将命令 `ls` 与它的选项 `-l` 分隔开。

其他赋值给 IFS 变量的分隔符还有制表符(tab)和换行符(回车)。

【注意】

IFS 字符是不可见的(非打印字符),因此用户无法在等号右侧看到它们。但是,它们确实存在。

【实例】

要改变 IFS 字符,运行如下命令:

```

$ ls -C [Return] .....在本命令行中,空格是分隔符
memos myfirst REPORT
$ save = $ IFS [Return].....为了安全起见,保存旧的 IFS 变量值
$ IFS = "!" [Return].....将 IFS 值改为叹号
$ ls! -l [Return] .....在本命令行中,叹号用作分隔符。它很笨拙,但工作得很好
$ IFS = $sav [Return] .....恢复原始分隔符
$ _ .....命令提示符

```

MAIL 变量

MAIL 变量被设为接收用户邮件文件的文件名。该文件存放发给用户的邮件。shell 定期检查该文件的内容,并在有邮件发给用户时通知用户。例如,将邮箱设为 `/usr/david/mbox`,只需键入 `MAIL = /usr/david/mbox`,然后按回车键。

MAILCHECK 变量

MAILCHECK 变量定义 shell 多长时间检查一次邮件(定义在 MAIL 变量中)。MAILCHECK

的默认值是 600 秒。

PATH 变量

PATH 变量设置 shell 在定位命令(程序)时所查找的目录名。例如, `PATH = :/bin:/usr/bin`。

路径字符串中的目录用冒号分隔。如果在路径字符串中第一个字符是一个冒号,则 shell 把它解释为.:(点,冒号),意思是用户当前目录是目录列表中的第一个,并且在搜索时第一个进行搜索。

UNIX 通常把可执行文件存在一个叫作 bin 的目录中。用户可以创建自己的 bin 目录,并把自己的可执行程序放在其中。如果用户把自己的 bin 目录(或用户使用的其他目录名)加入到变量 PATH 中,则 shell 在查找命令时如果在标准目录中找不到,就会到用户定义的目录中寻找。

假定用户的所有可执行程序位于用户主目录下名为 mybin 的子目录中,要把该子目录加入到 PATH 变量中,只需键入 `PATH = :/bin:/usr/bin: $HOME/mybin`,然后按回车键即可。

PS1 变量

命令提示字符串 1 变量(PS1)设置用户命令提示符字符串。Bourne shell 的基本命令提示符是美元符 \$。

【实例】

如果用户对 \$ 符号感到厌烦,则可以简单地通过给 PS1 变量赋一个新的值来改变命令提示符。

```
$ PS1 = Here: [Return].....把命令提示符变为 Here:
Here: _ ..... 就在这里
Here: PS1 = "Here: "[Return] ....在尾部加一个空格
Here: _ ..... 现在好看多了
```

【注意】

如果命令提示符中包含空格,则该命令提示符必须用引号引起来。

```
Here: PS1 = "Next Command: "[Return].....再次改变命令提示符
Next Command: _.....命令提示符改变
Next Command: PS1 = "$" [Return].....变回原来的 $ 命令提示符
$_.....恢复原来的 $ 命令提示符
```

PS2 变量

命令提示字符串 2 变量(PS2)设置 shell 在用户尚未键入完整命令前按回车键后显示的命令提示符,表明 shell 在等待命令的剩余部分。用户可以用与 PS1 相同的方式改变 PS2 变量。Bourne shell 的第 2 命令提示符默认为大于号 >。

【实例】

以下命令显示第 2 命令提示符的操作。

```

$ echo "Good news, UNIX [Return].....命令行尚未完成,因此 PS2 提示符出现
> is on videotape." [Return] ..... 现在完成了命令行
Good news, UNIX is on videotape.
$ ls \ [Return] .....反斜杠表示命令行未结束
> ..... 因此 shell 显示第 2 命令提示符并等待命令的
                      剩余部分
> -l [Return] ..... 现在命令行完成,shell 把命令行的内容放在一
                      起,形成命令 ls -l 然后执行
$_ .....命令提示符

```

CDPATH 变量

与 PATH 变量类似,CDPATH 设置一系列相互独立的路径名。CDPATH 变量影响 `cd`(改变目录)命令的操作。如果该变量未定义,则 `cd` 搜索用户的工作目录,寻找符合它的参数的文件名。如果在用户的工作目录不存在所查找的子目录,则 UNIX 显示一个错误信息。如果该变量被定义,`cd` 在该变量定义的路径内搜索要找的目录。如果找到,则该目录即成为用户的工作目录。

例如,键入 `CDPATH = : $HOME: $HOME/memos` 然后按回车键,则用户再次运行 `cd` 命令时,shell 将首先搜索用户当前目录,然后是用户主目录,最后是用户主目录下的 `memos` 目录,从中寻找与其命令参数相匹配的文件名。

SHELL 变量

SHELL 变量设置用户登录 shell 的完整路径:

```
SHELL = /bin/sh
```

TERM 变量

TERM 变量设置用户终端类型:

```
TERM = vt100
```

TZ 变量

TZ 变量设置用户所在时区。

```
TZ = EST = EDT
```

通常由系统管理员进行此项设置。

8.3 更多的元字符

如同读者所了解的,元字符或特殊字符被 shell 以特殊的方式解释执行。到目前为止,已经讨论了文件替换符和重定向元字符。本节将继续探索更多的元字符。

8.3.1 执行命令:使用后单引号

后单引号告诉 shell 执行被引号所引住的命令,并将命令的输出送到该命令所在命令行的

位置上。它也叫命令替换符。格式如下：

```
\command
```

`command` 是要执行的命令。

【实例】

以下命令序列显示命令替换符的操作。

```
$ echo The date and time is:`date` [Return].....命令 date 被执行
The date and time is: Mon 16 30:14:14 EDT 2001
$_ .....命令提示符
```

shell 扫描命令行,发现后单引号,执行 `date` 命令。然后,用 `date` 命令的输出替换命令行中的 `date`,最后执行 `echo` 命令。

```
$ echo "List of filenames in your current directory: \n" `ls -C` > LIST [Return]
$ cat LIST [Return] .....检查一下在 LIST 文件中存了什么信息
List of filenames in your current directory:
memos myfirst REPORT
$_ .....为下一条命令做好了准备
```

8.3.2 命令排序:使用分号

用户可以在一行命令中键入多个命令,其间为分号隔开。shell 将从左到右依次执行这些命令。

【实例】

体验分号元字符,尝试下列命令。

```
$ date;pwd;ls -C [Return].....按顺序执行 3 个命令
Mon Nov 28 14:14:14 EST 2001
/usr/david
memos myfirst REPORT
$ ls -C > list;date > today;pwd [Return].....按顺序执行 3 个命令,其中 2 个命令输出重定向到文件中
/usr/david
$ cat list [Return] .....查看 list 文件的内容
Memos myfirst REPORT
$ cat today [Return] .....查看 today 文件的内容
Mon Nov 28 14:14:14 EST 2001
$_ .....命令提示符
```

8.3.3 命令编组:使用括号

用户可以通过将命令放入括号中的方法来将命令编组。被编为一组的可以像一个命令那样重定向。

【实例】

体验括号作元字符,尝试如下操作。

```

$ (ls -C;date;pw) > outfile [Return].....3 个连续的命令编为 1 组,输出
重定向到 1 个文件
$ cat outfile [Return] ..... 查看 outfile 的内容
memos myfirst REPORT
Mon Nov 28 14:14:14 EST 2001
/usr/david
$ _ .....命令提示符

```

8.3.4 后台计算:使用 & 符号

UNIX 是一个多任务系统,允许用户同时运行多个程序。通常,键入一个命令后几秒钟内,命令执行的输出就在显示类终端上。但是,如果执行一个要花几分钟的程序会怎样? 在这种情况下,用户不得不在继续下一个工作前等这个命令完成。不过用户不需要等待这些非生产性的时间。shell 的元字符 & 提供了在后台运行不需要键盘输入的程序的方法。用户键入一个命令,其后紧跟 & 字符,则该命令被送往后台执行,而终端可继续键入下一个命令。

【实例】

下面例子显示 & 元字符的用法。

```

$ sort data > sorted&[Return] .....将 data 文件排序并把输出结果存在 sorted 文件中
1348..... 显示进程 ID
$ date [Return] .....命令提示符立即出现,为下一命令做好准备

```

sort 命令的输出被重定向到另一个文件。这样防止了在用户进行下一个任务时 sort 命令将它的输出发送到终端。

【说明】

1. 后台进程 ID 用来惟一地标识一个后台进程,并且可被用于终止进程或获取进程的状态。

2. 用户可在一个命令行中定义多个后台命令。

```

$ date $ pwd & ls -C &[Return].....创建 3 个后台进程,显示 3 个进程 ID
2215
2217
2216
$ echo "the foreground process" [Return] ..前台运行 echo 命令
Mon Nov 28 14:14:14 EST 2001..... 后台运行的 date 命令的输出已经显示在终端
the foreground process..... 前台运行的 echo 命令输出到终端
/usr/david..... 后台运行的 pwd 命令输出到终端
$ _..... 显示命令提示符,随后在终端显示 ls -C 命令
的输出
memos myfirst REPORT
$ _ .....为下一个命令做好准备

```

【说明】

默认情况下,后台命令的输出显示在终端。因此,前台命令的输出会被后台命令的输出所交叉,并造成一定的显示混乱。用户可以通过将后台输出重定向到文件中来避免这一问题。

8.3.5 链接命令:使用管道操作符

shell 允许将一个命令的标准输出用作另一个命令的标准输入。用户可能在两个命令间使用管道操作符|来实现这一功能。通常格式如下:

```
command A | command B
```

于是命令 A 的输出被作为命令 B 的输入。用户可将一系列命令串在一起形成一个管线。下面看一个这个有用而且灵活的 shell 功能的应用。

【实例】

键入 `ls -l | lp` 然后按回车键,则将 `ls -l` 命令的输出送到打印机。

【实例】

下例计算当前目录中的文件数目。

```
$ ls -C [Return].....看一下当前目录的文件
memos myfirst REPORT
$ ls -C > count [Return].....现在将当前目录下的文件列表存入文件 count 中
$ wc -w count [Return].....计算 count 文件的字数,也就是当前目录所包含的文件数,是 3
                                     个文件
3
$ ls -C | wc -w [Return].....使用管道操作符来获得相同的效果
3
```

命令 `ls -C`(列出用户当前目录的文件)的输出被作为 `wc -w` 命令输入。

【实例】

下例将当前登录到系统的用户数存入到一个文件中。

```
$ echo "Number of the logged in users:" `who | wc -l` > outfile [Return]
$ cat outfile [Return].....检查在 outfile 中存了什么
Number of the logged in users:20
```

在前一个命令中,shell 扫描命令行,发现后单引号,运行命令 `who | wc -l`,即把 `who` 命令的输出送往 `wc -l` 作为输入进行计算。如果有 20 个用户登录到系统,则该命令的输出为 20。shell 用 20 来替换 `who | wc -l`。接着,shell 执行 `echo` 命令,其结果是把 `Number of the logged in users:20` 存入文件 `outfile` 中。

8.4 更多的 UNIX 系统工具

这些工具为用户每天使用系统带来更多的灵活性和控制能力。并且,部分工具还用于脚本文件中(详见第 11 章、第 12 章)。

8.4.1 延时计时:sleep 命令

`sleep` 命令使运行它的程序转入一定时间的休眠状态。用户可以使用 `sleep` 来推迟一个命令一段时间后执行。例如,键入 `sleep 120; echo "I am awake!"`,然后按回车键,则执行 `sleep` 命

令导致2分钟的延迟,然后 `echo` 命令被执行,于是 `I am awake!` 显示在屏幕上。

8.4.2 显示 PID: `ps` 命令

用户可以通过 `ps` 命令来获得系统当前活动进程的状态。如果不带任何参数,该命令显示用户当前活动进程的信息。这些信息显示为 4 列(如图 8.4 所示),内容如下:

- PID: 进程 ID 数
- TTY: 控制该进程的用户终端号
- TIME: 用户进程已经运行的时间(以秒计)
- COMMAND: 命令名

```
$ ps
PID    TTY    TIME   COMMAND
24059  tty11  0:05   sh
24259  tty11  0:02   ps
$_
```

图 8.4 `ps` 命令的输出格式

`ps` 选项

本书只讨论 `ps` 命令的 2 个选项 `-a` 选项和 `-f` 选项。见表 8.2 的总结。

表 8.2 `ps` 命令的选项

选 项	功 能
<code>-a</code>	显示所有活动进程的状态信息
<code>-f</code>	显示包括全部信息,包含命令列中的全部命令行

-a 选项: `-a` 选项显示所有活动进程的状态信息。此选项默认时只显示用户的活动进程。

-f 选项: `-f` 选项显示包括全部信息,包含命令列中的全部命令行。

图 8.5 显示了同时包含 `-a` 和 `-f` 选项的 `ps` 命令。

```
PID    TTY    TIME   COMMAND
24059  tty11  0:05   sh
24259  tty11  0:02   ps
24059  tty11  0:05   sh
24259  tty11  0:02   ps
```

图 8.5 带 `-a` 和 `-f` 选项的 `ps` 命令的输出

【实例】

如下操作可用来显示一个正在后台运行的进程的数目。

```
$ (sleep 120;echo "Had a nice long sleep")&[Return]
24259 .....后台进程 ID 数
```

```

$ ps [Return]..... 显示用户进程状态
PID TTY TIME COMMAND
24059 tty11 0:05 sh .....登录的 shell
24070 tty11 0:00 sleep 1200.....sleep 命令
24150 tty11 0:00 echo ..... echo 命令
24259 tty11 0:02 ps .....ps 命令
Had a nice long sleep ..... 后台进程的输出
$_ ..... 出现命令提示符

```

sleep 命令延迟了 **echo** 命令的执行 2 分钟。命令行末的 **&** 命名该命令在后台执行。

【说明】

要用分号将命令分开,并且用括号把命令编组。

8.4.3 继续执行: **nohup** 命令

用户退出系统时,其后台进程即被终止。而 **nohup** 命令可使用户的进程免于被终止信号所终止。这在用户希望退出系统后,他的后台进程继续执行非常有用。

如果键入 **nohup(sleep 120;echo "job done")&**,然后按回车键并退出系统,该命令仍将在系统后台执行。但是这时 **echo** 命令的输出将送往哪里呢?由于用户已经退出系统,该进程不再与任何终端有关,因此输出被自动存入文件 **nohup.out** 中。

当用户再次登录时,可能查看这个文件的内容来判断后台程序的运行情况。当然,用户可以使用重定向来将输出送往自己指定的文件中。

【实例】

尝试如下命令,体验 **nohup**。

```

$ nohup (sleep 120;echo "job done")&[Return].... 创建一个后台进程
12235 ..... 该后台进程的 ID
$ [Ctrl-d]..... 退出并等一会儿
Login: david [Return]..... 再次登录
Password: ..... 输入用户口令,屏幕终端不显示口令
$ cat nohup.out [Return]..... 查看 nohup.out 文件内容
job done
$_ ..... 为下一命令做好了准备

```

8.4.4 终止一个进程: **kill** 命令

并不是所有的进程都自始至终运行正常,程序可能会进入死循环或等待不会得到的资源。有时,一个不正常的程序锁住用户的键盘,这时用户就面临真正的麻烦了。UNIX 提供 **kill** 命令来终止不想要的进程(一个进程是指一个正在运行的程序)。**kill** 命令向指定的进程发送一个信号。该信号是一个内部数,用来指示 **kill** 的类型(UNIX 是一种病态语言)。进程的指定是通过进程数 **PID** 来完成的。因此,要使用 **kill** 命令,用户必须知道想要终止进程的 **PID**。

信号:信号值为 1 到 15,并且大多数是根据系统实现的不同而不同。但是,通常值 15 都用作默认值,并导致接收进程终止。

有些进程不受 **kill** 信号影响。用户可能使用信号值 9 来终止这样的进程。

下面命令显示 **kill** 命令和其信号的使用。

【实例】

体验 kill 命令使用,尝试如下操作:

```

$ (sleep 120;echo Hi)&[Return]..... 创建一个后台进程
22515..... 进程 ID 数
$ ps [Return]..... 查看进程状态,可以看到新创建的进程
PID TTY TIME COMMAND
24059 tty11 0:05 sh..... 登录 shell
22515 tty11 0:00 sleep 1200..... sleep 命令
24259 tty11 0:02 ps..... ps 命令
$ kill 22515[Return]..... 终止后台进程
job terminated
Hi..... echo 命令的输出
$ ps [Return]..... 再次查看,后台进程已经中止了
PID TTY TIME COMMAND
24059 tty11 0:05 sh..... 登录 shell
24259 tty11 0:02 ps..... ps 命令
$_..... 命令提示符

```

【说明】

上面 kill 命令中未定义信号数。默认值为 15,该值导致接收进程终止。

【实例】

确保免疫进程被终止,尝试如下操作:

```

$ (sleep 120;echo Hi)&[Return]..... 创建一个后台进程
22515..... 进程 ID 数
$ kill 22515[Return]..... 简单终止
$ ps [Return]..... 查看进程状态,可以看到该进程还在
PID TTY TIME COMMAND
24059 tty11 0:05 sh..... 登录 shell
22515 tty11 0:00 sleep 1200..... sleep 命令
24259 tty11 0:02 ps..... ps 命令
$ kill -9 22515[Return]..... 确实终止,使用信号值 9
$ ps [Return]..... 再次查看,可以确定后台过程已被终止
PID TTY TIME COMMAND
24059 tty11 0:05 sh..... 登录 shell
24259 tty11 0:02 ps..... ps 命令

```

【注意】

用户只能终止自己的进程。系统管理员有权终止任何用户的进程。

【实例】

要终止用户的所有进程,尝试如下操作:

```

$ (sleep 120;echo "sleep tight";sleep 120)&[Return]
11234..... sleep 的 PID
$ kill -9 0 [Return]..... 用户会退出系统

```

【说明】

PID 0 导致与用户 shell 有关的所有进程被终止。这也包括用户的 shell 本身。因此,使用

了 0 信号,用户即退出系统。

8.4.5 分离输出:tee 命令

有时,用户可能既想从屏幕上看到一个程序的输出,又想将该输出存到硬盘上以备日后参考,或者希望得到一个打印件。用户可能通过如下操作来实现这样的目的:首先运行命令并从屏幕上观看输出,然后使用重定向符将输出结果存到一个文件或送往打印机。

同样,用户可以使用 `tee` 命令来在更短时间内,通过更少的键入来获得相同的结果。`tee` 命令通常与管道操作符一同使用。例如,当用户键入 `sort phone.list | tee phone.sort` 并按回车键时,管道操作符将 `sort` 命令的输出(被排序的 `phone.list` 文件)送给 `tee`。接着,`tee` 将其显示在终端,同时存往指定的文件 `phone.sort` 中。

当运行交互程序时,如果希望抓取用户/程序的对话,`tee` 命令是一个必不可少的系统工具。

【实例】

试查看当前目录,并将输出存入一个文件中。

```
$ ls -C | tee dir.list [Return].....显示当前目录文件,同时把输出结果存到
                               文件 dir.list 中
memos myfirst REPORT
$ cat dir.list [Return].....查看 dir.list 文件的内容
memos myfirst REPORT
$_ .....命令提示符
```

`ls -C` 命令的输出被通过管道送往 `tee`。`tee` 命令将它的输入显示在屏幕上(将输入送往标准输出),同时存到文件 `dir.list` 中。

tee 选项

表 8.3 总结了 `tee` 命令的 2 个选项。

表 8.3 tee 命令选项

选 项	功 能
-a	将输出追加到一个文件中,不覆盖原有内容
-i	忽略中断,不响应中断信号

【实例】

如果希望查看当前登录到系统的用户,并将结果存到文件 `dir.list` 中,键入 `who | tee -a dir.list` 然后按回车键。如果 `dir.list` 已存在,则 `who` 命令的输出被追加到 `dir.list` 文件的末尾。如果该文件不存在则创建它。

8.4.6 文件搜索:grep 命令

用户可以使用 `grep` 命令来从一个或一系列文件中寻找特定的样式。`grep` 命令所使用的样式被称为正则表达式。`grep` 命令来自该命令的名称 Global Regular Expression Print(全部正则

表达式显示)。

grep 是一个文件搜索和选择命令。用户定义文件名和要在文件中查找的样式,当 **grep** 发现给定样式匹配时,在终端显示包含匹配样式的行。如果未定义要查找的文件,则系统从标准输入设备查找输入。

【实例】

在文件 `myfile` 中查找字 UNIX。

```
# cat myfile [Return].....查看文件 myfile 的内容
I wish there were a better way to learn
UNIX. Something like having a daily UNIX pill.
# grep UNIX myfile [Return].....查找包含 UNIX 的行
UNIX. Something like having a daily UNIX pill.
```

用户可以定义多个文件,或使用文件替换符(通配符)。

【实例】

在所有 C 源文件中查找串“# include < private.h >”。

键入 `grep "# include < private.h >" *.c` 然后按回车键,在当前目录中所有扩展名为 `c` 的文件中查找该样式。

由于要查找的样式包含空格和元字符,因此用引号引住。

如果定义了多个文件进行查找,**grep** 将在输出的每行前显示文件名。

grep 选项

如果未定义选项,**grep** 显示在指定文件中包含匹配样式的行。选项使用户能更好地控制输出和样式的查找。表 8.4 总结了 **grep** 的选项。

表 8.4 grep 命令选项

选 项	功 能
<code>-c</code>	只显示每个文件中包含匹配样式的行数
<code>-i</code>	匹配时忽略大小写
<code>-l</code>	仅显示包含匹配样式的文件名,而不显示行
<code>-n</code>	在每个输出行前显示行号
<code>-v</code>	只显示不匹配的行

假设当前目录中有如下文件及文件内容。下列命令显示使用 **grep** 选项的示例。

FILE1	FILE2	FILE3
UNIX	unix	Unix system
11122	11122	11122
BBAA	CCAA	AADD
unix system		

【实例】

搜索字 UNIX。

```
$ grep UNIX FILE1 [Return].....在文件 FILE1 中查找字 UNIX
UNIX
$_ .....命令提示符
```

grep 匹配精确样式(区分大小写),因此它找到了 UNIX 而不是 unix。

【实例】

定义多个文件作参数,并使用 -i 选项。

```
$ grep -i UNIX FILE? [Return].....使用-i 选项
FILE1: UNIX
FILE1: unix system
FILE2: unix
FILE3: Unix system
$_ .....命令提示符
```

-i 选项使 **grep** 不区分大小写。因此,定义的样式 UNIX 匹配 unix、Unix 等。当定义多个文件时,文件名显示在匹配样式的行前。

【实例】

显示不含 UNIX 的行。

```
$ grep -vi UNIX FILE1 [Return].....使用-i 选项和-v 选项
11122
BBAA
$_ .....命令提示符
```

【实例】

显示在每个文件中有多少个不包含 11 的行。

```
$ grep -vc 11 FILE? [Return]....显示在文件 FILE1、FILE2 和 FILE3 中不包含 11 的行数
FILE1: 3
FILE2: 2
FILE3: 2
$_ .....命令提示符
```

【实例】

查找用户 david 是否登录到系统。

```
$ who | grep -i david [Return].....使用 grep 和管道操作符
$_ .....命令提示符
```

管道操作符把 **who** 命令的输入送往 **grep** 命令的标准输入。因此,**grep** 扫描 **who** 的输出查找是否有样式 david。在本例中,**grep** 没有输出,因此 david 并非登录到系统。

8.4.7 文本文件排序:sort 命令

用户可用 **sort** 命令对文件的内容按字符或数字顺序进行排序。默认情况下,输出被送往用户终端,但用户可以定义一个文件名作为 **sort** 命令的参数,以使该命令的输出被重定向到一

个文件。

`sort` 命令按行将指定文件内容排序。如果有 2 行的第一个字符相同,则比较第二个字符来确定排序次序。如果第二个字符也相同,则比较第三个字符。依此类推,直到比较出 2 个不同字符为止。如果有 2 行内容完全相同,则 `sort` 任意确定一行在前。

【说明】

该命令按字符序列给文件排序,但所排序顺序对不同计算机可能有不同结果,这是因为不同的计算机可能使用不同的代码集。UNIX 系统中最为常用的是 ASCII 字符集。

有许多选项可用于控制排序顺序,这里由浅入深进行介绍。

假设在当前目录下有一文件名为 `junk`。图 8.6 为文件 `junk` 的内容。图 8.7 为 `sort` 命令的输出,将 `junk` 的内容排序。

```
This is line one
this is line two

  this is a line starting with a space character
4:this is a line starting with a number
11:this is another line starting with a number
End of junk
```

图 8.6 文件 `junk`

```
  this is a line starting with a space character
11:this is another line starting with a number
4:this is a line starting with a number
End of junk
This is line one
this is line two
```

图 8.7 排序后的文件 `junk`

【说明】

1. 非文字数字字符(空格、破折号、反斜杠等)的 ASCII 值比文字数字的值小。因此,在图 8.7 中,以空格开头的一行被放在文件的顶部。

2. 大写字母排序在小写字母之前。因此,图 8.7 中 `This` 出现在 `this` 之前。

3. 数字按第一个数位排序。因此,11 排在 4 之前。

`sort` 选项

通过前面 `sort` 命令的例子读者可能会发现,`sort` 排序的结果可能并不是用户所希望的。`sort` 命令的选项可以改变这一现象,并让用户能够以真实情况自由地排序文件。表 8.5 总结了 `sort` 命令的选项。

表 8.5 sort 命令选项

选 项	功 能
-b	忽略前导空格
-d	使用字典序。忽略标点符号和控制符
-f	忽略大小写
-n	数字按数值排序
-o	将输出存入一个文件
-r	反序,从升序变为降序

-b 选项: **-b** 选项使 `sort` 命令忽略前导空格(包括制表符和空格)。这种字符在文件中通常用作定界符(字段分隔符)。在使用该选项时,`sort` 在进行排序比较时不考虑它们。

-d 选项: 按字典序进行排序,只对字母、数字和空格(包括制表符和空格)进行排序比较。忽略标点符号和控制符。

-f 选项: 使 `sort` 将所有小写字母看作大写字母,在排序比较时忽略二者的区别。

-n 选项: **-n** 选项导致数字按数值大小进行考虑而不是按数位排序,包括对负数和小数的大小比较。

-o 选项: **-o** 选项将输出存入一个文件,而不是标准输出。

-r 选项: **-r** 反转排序的次序,如用 `z` 到 `a` 顺序代替 `a` 到 `z` 顺序。

【实例】

再次使用文件 `junk`,看一下选项对排序产生的影响。

```
$ sort-fn junk [Return].....使用-f 和-n 选项排序文件 junk
  this is a line starting with a space character
End of junk
This is line one
this is line two
4:this is a line starting with a number
11:this is another line starting with a number
$_..... 命令提示符
$ sort- f - r - o sorted junk [Return].....使用-f,-r 和-o 选项排序文件 junk
$ cat sorted [Return].....显示文件 sorted 的内容
this is line two
This is line one
End of junk
4:this is a line starting with a number
11:this is another line starting with a number
  this is a line starting with a space character
$_..... 命令提示符
```

【说明】

由于用 **-o** 选项指定了输出的文件名(`sorted`),因此输出被存到文件 `sorted` 中,使用 `cat` 命令显示文件 `sorted` 的内容。

8.4.8 按指定字段排序

实际的文件很少包含像例子文件 `junk` 那样的内容。通常用户希望排序的文件包括名、事

件、地址、电话簿、通信地址等列表。默认情况下, `sort` 命令按行进行排序。但是, 用户很可能希望 `sort` 按某一特定的字段, 如按名或地址代码进行排序。

用户可能控制 `sort` 命令对某一特定字段进行排序比较, 前提是文件是按照相应的格式建立的。通过指定 `sort` 命令要略过多少个字段来到达指定字段的方式, 指定希望排序的字段。用户需要在每行用空格分隔字段, 建立用户的字段形式文件。

【实例】

创建一个文件 `phone.list`, 包含名、电话, 如图 8.8 所示, 下面用此文件为例研究 `sort` 命令的其他功能。每行由 4 个字段组成, 这些字段用空格或制表符分隔。因此在第一行中, David 是第一个字段, Brown 是第二个字段等等。

```
$ cat phone.list
David Brown      (703) 555-0014
Emma Redd       (202) 555-9000
Marie Lambert   (202) 555-6666
Susan Bahcall  (202) 555-7800
Steve Fraser    (301) 555-5566
Azi Jones       (202) 555-6500
Glenda Hardison (301) 555-8822

$ sort phone.list
Azi Jones        (202) 555-6500
David Brown      (703) 555-0014
Emma Redd       (202) 555-9000
Glenda Hardison (301) 555-8822
Marie Lambert   (202) 555-6666
Steve Fraser    (301) 555-5566
Susan Bahcall  (202) 555-7800
$
```

图 8.8 原始的 `phone.list` 文件和排序后的 `phone.list` 文件

上图中, 使用 `sort` 命令对 `phone.list` 排序时未使用任何参数, 因此, 该文件被按行排序。

用户可能并不想按名进行排序。如想按姓对该文件排序, 必须告诉 `sort` 在开始排序进程前略过 1 个字段(名)。将 `sort` 应当跳过的字段数作为命令参数。

【实例】

按姓(字段 2)对 `phone.list` 进行排序, 键入 `sort +1 phone.list`, 然后按回车键。图 8.9 显示按姓对 `phone.list` 排序的结果(注意, 外国人名在前, 姓在后)。

参数 `+1` 告诉 `sort` 在开始排序之前先略过第一个字段(名)。

【注意】

如果定义参数为 `+2`, 则 `sort` 略过第一、第二字段, 根据第三字段(在本例中是按区码)对该文件排序。

```
# sort +1 phone.list
Susan Bahcall      (202) 555-7800
David Brown        (202) 555-0014
Steve Fraser       (202) 555-5566
Glenda Hardison   (202) 555-8822
Azi Jones          (202) 555-6500
Marie Lambert      (202) 555-6666
Emma Redd          (202) 555-9000
$ _
```

图 8.9 按姓排序的 phone.list 文件

【实例】

按第三个字段(区码)对该文件排序,键入 `sort +2 phone.list`,然后按回车键,图 8.10 为这样键入导致的问题。

```
# sort +2 phone.list
Azi Jones          (202) 555-6500
Emma Redd          (202) 555-9000
David Brown        (703) 555-0014
Steve Fraser       (301) 555-5566
Susan Bahcall      (202) 555-7800
Marie Lambert      (202) 555-6666
Glenda Hardison   (301) 555-8822
$ _
```

图 8.10 不带 -b 选项的按区码排序的 phone.list 文件

`sort` 命令忽略 2 个字段,但是将第二个字段后的空格看作是第三个字段的一部分。因此,输出看起来像是按照姓的字母个数由少到多进行排序的。为了解决这个问题,必须告知 `sort` 命令忽略空格(用 `-b` 选项)。

【实例】

为了按第三字段对 `phone.list` 排序,忽略空格,键入 `sort -b +2 phone.list` 并按回车键,图 8.11 为正确排序的结果。

```
# sort -b +2 phone.list
Azi Jones          (202) 555-6500
Marie Lambert      (202) 555-6666
Emma Redd          (202) 555 -9000
Susan Bahcall      (202) 555-7800
Steve Fraser       (301) 555-5566
Glenda Hardison   (301) 555-8822
David Brown        (703) 555-0014
$ _
```

图 8.11 按区码排序的 phone.list 文件

8.5 启动文件

在登录时,登录程序根据口令文件中的内容对用户的 ID 和口令进行验证。如果用户登录成功,登录程序将用户的主目录提交给系统,设置用户 ID 和组 ID,最后启动用户的 shell。在显示命令提示符前,shell 查看 2 个特殊文件。这 2 个文件被称为策略文件(profile file),并且它们包含 shell 可执行的脚本程序。

8.5.1 系统策略

系统策略文件存放于/etc/profile 中。用户 shell 做的第一件事就是执行这个文件。该文件典型地包括显示当日邮件、建立系统环境变量等命令。该文件通常由系统管理员创建并维护,并且只有超级用户可以修改它。

图 8.12 为一个系统策略文件的例子。shell 执行文件中的命令,因此显示当前日期和时间,接着是当日邮件(保存于文件/etc/motd 中),最后是最近新闻。

```
$ cat/etc/profile
date
cat/etc/motd
news
$ _
```

图 8.12 一个简单的策略文件的例子

【说明】

1. 通常系统策略文件比较复杂,并结合了一些系统管理员命令,且需要一些编程。
2. 通过键入 `cat /etc/profile` 并按回车键,可以查看用户自己的策略文件。通常,该文件是只读的,不可写。

8.5.2 用户策略

每当用户登录时,shell 检查位于用户主目录的名为 .profile 的启动文件。如果找到该文件,则执行文件中的命令。无论是否有该 .profile 文件,shell 都将继续启动进程并显示命令提示符。

图 8.13 为一个 .profile 文件的例子。在系统管理员的允许下,用户通常有自己的 .profile 文件。用户可以修改已有的 .profile 文件或使用 `cat` 或 `vi` 命令来创建它。

【说明】

1. 文件的名称是 .profile。文件名的第一个字符是一个点。这是一个隐藏文件。
2. .profile 必须定位于用户主目录。这是 shell 惟一搜索的地方。
3. .profile 是用户可以用来自定义 UNIX 环境的启动文件之一。UNIX 还有其他启动文件,如用来定义 vi 编辑器(见第 6 章)的 .exrc 文件和用来定义邮件环境的 .mailrc 文件(见第 9 章)。

```

$ cat .profile
echo "welcome to my super Duper UNIX"
TERM= vt100
PS1="David Brown:"
export TERM PS1
calendar
du
$ _

```

图 8.13 一个 .profile 文件的例子

1. `echo` 命令显示它的参数,即 `welcome to my super Duper UNIX`。
2. 标准变量 `TERM`(终端类型)被设为 `vt100`。
3. 标准变量 `PS1`(基本命令提示符)设为 `David Brown`。
4. `export` 命令使变量 `TERM` 和 `PS1` 可用(将这 2 个变量送给所有程序)。
5. `calendar` 和 `du` 命令将在第 13 章解释。

关于 `export` 命令的更多知识

`export` 命令使 shell 变量可用于子 shell。当用户登录时,登录的 shell 知道所有标准变量(也可能还有用户定义的变量)。但是,如果用户运行一个新的 shell 时,新 shell 并不知道这些变量。

例如,如果想使用变量 `VAR1` 和 `VAR2` 在新 shell 中可用,必须将这些变量名定义为 `export` 命令的参数。如果想看已经有哪些变量传送给环境,只需键入 `export` 不加任何参数。

【实例】

使变量对其他 shell 可用。

```

$ export VAR1 VAR2 [Return].....将 VAR1 和 VAR2 送给环境
$ export [Return].....查看已有哪些变量送给了环境
VAR1
VAR2
$ _.....命令提示符

```

8.6 KORN shell(ksh)

第 3 章介绍了 Korn shell。这里解释一些它与标准 shell 不同的特征。

8.6.1 Korn shell(ksh)变量

Korn shell(ksh)使用许多与标准 shell(sh)所使用的相同变量。用户可以定义变量、重定义变量、取得变量的值和操作变量改变自己的环境,如同 sh shell 一样。以下是一些只用于 Korn shell 的重要变量:

ENV: ENV 变量存放启动时被 ksh 读取的环境文件的全路径。下面例子中,ENV 被设为告

知 ksh 从哪里找到环境文件的路径(\$HOME/mine/my_env)。

```
ENV = $HOME/mine/my_env
```

HISTSIZE: HISTSIZE 变量设置用户所希望的命令历史文件的大小,其值为 history 文件中所包含的命令数。默认为 128,但用户可以将它设为任何值。例如,下面命令将用户 history 文件中的命令条数设为 100:

```
HISTSIZE = 100
```

TMOUT: TMOUT 变量设置用户不键入命令的系统超时时间,以秒计算。如果用户在一定时间内不键入任何命令,shell 在超时后将强制用户退出系统。例如,下面命令行设置超时时间为 60 秒:

```
TMOUT = 60
```

VISUAL: VISUAL 变量用于定义命令编辑方式。如果该变量设为 vi,则 ksh 在命令行向用户提供 vi 风格的编辑方式。例如,下面命令行将命令行编辑方式设为 vi 编辑器:

```
VISUAL = vi
```

【说明】

如同在 sh 中一样,使用 set 命令可以查看当前 shell 变量及其值。

8.6.2 Korn shell(ksh)选项

Korn shell 提供了一系列选项来打开或关闭它独有的特征。要打开一个选项,使用 set 命令,后面紧跟 -o 选项,然后跟着要打开选项的名字。列出选项只需键入 set -o。

noclobber: noclobber 选项防止用户无意覆盖已有文件;也就是说,它防止用户在将一个命令重定向时覆盖一个已存在的文件。这样可以防止由于无意中覆盖一个文件而丢失重要数据。

【实例】

假设一个名为 xyz 的文件位于当前目录。下列命令示例 noclobber 选项的使用。

```
$ set -o noclobber [Return].....设置 noclobber 选项
$ who > xyz .....重定向输出到文件 xyz
xyz: file exists ..... 显示警告信息
$_ ..... 显示命令提示符
```

如果用户确实希望覆盖一个已存在的文件,ksh 可以通过在重定向操作符后加一个管道符 | 来强制执行。

```
$ who > |xyz [Return].....xyz 文件被覆盖
$_ ..... 命令提示符
```

用 set 命令和 +o 选项可以关闭 noclobber 选项。如:

```
$ set +o noclobber [Return].....关闭 noclobber 选项
$_ .....显示命令提示符
```

【说明】

ignoreeof: ignoreeof 选项可以防止用户因为偶然碰到 [Ctrl-d] 键而退出系统(读者应当已经知道,在命令行一开始键入 [Ctrl-d] 将会终止用户的 shell 并退出当前系统)。

设置该选项后,用户需要使用 **exit** 命令来退出系统。

```
$ set -o ignoreeof [Return].....打开 ignoreeof 选项
$ [Ctrl-d].....[Ctrl-d]被忽略,用户不会退出系统
$ set to ignoreeof [Return] .....关闭 ignoreeof 选项
$ _ .....命令提示符
```

8.6.3 命令历史(ksh): history 命令

Korn shell 使用一个历史文件保存用户会话中的所有命令序列。使用这一流行功能可以使用户列出已经键入过的命令,找到已执行过的命令,再次执行或简单修改后执行。

【实例】

如下操作显示最近键入的命令:

```
$ history [Return].....生成命令
101 Who am I
102 ls -l
103 pwd
104 vi myfirst
105 vm myfirst
106 history
$ _ .....提示符
```

【说明】

1. 这个例子显示了 6 个命令。ksh 保存的命令数由变量 HISTSIZE 控制。
2. 显示的最后一行是用户最近执行的命令,越早执行的命令越靠前显示。

默认的历史文件是 `.sh_history`, 由系统创建在用户主目录里。用户可以通过设定变量 HISTFILE 来指定自己的历史文件。如:

```
HISTFILE = $HOME/history/my_hist
```

重新开始历史文件

用户的命令历史文件在不同的时间会被保存,并不停地增加,因此会变得越来越大,而且难于有效查找历史命令记录。如果用户想重新开始历史文件,只需将主目录下的 `.sh_history` 文件删除。下次用户登录时,系统会为用户新建一个 `.sh_history` 文件,其中的内容重新开始记录。

【实例】

要想执行历史文件中的某命令,只需键入 **history** 并将希望执行的命令作为该命令的参数即可。

```
$ history vi [Return].....从历史文件中寻找 vi 命令
```

vi myfirst用户最近一次执行的 vi 命令

8.6.4 重复执行命令(ksh):r(重复执行)命令

使用 r 命令可以重复最近一次执行的命令。例如,假设用户最近执行的命令是 rm my-first,然后用户键入:

```
$ r [Return].....重复最近执行的命令
rm myfirst.....重复用户最后执行的命令
```

在本例中,用户最后一次执行的命令被再次执行。下面的命令显示 r(重复)命令的选项和特征。

【说明】

绝大多数情况下,用户执行命令的情况与这个例子中的不同。用户的历史文件不会保持不变,用户键入的每个命令被加入到已存在的历史文件中。下面的例子中,并没有假设特定的历史文件。

【实例】

用户可以通过将特定命令名用作 r 命令的参数值,重复历史文件中的其他命令。

```
$ r vi [Return].....重复历史文件中的 vi 命令
vi myfirst .....被重复执行的命令回显在命令行
```

【实例】

通过指定命令在历史文件中的条目序号,可以重复执行任何命令。

```
$ r 102 [Return].....重复历史文件中第 102 号命令
ls -l .....所指定的命令被执行
```

用户也可以通过指定希望执行倒数的命令条目来确定要执行的命令。

```
$ r -3 [Return].....执行倒数第 3 条命令
ls -l .....所指定的命令被执行
```

8.6.5 别名(ksh):alias 命令

用户可以使用 alias 命令来缩短频繁使用的命令名,或改为自己易记的命令名。例如:

```
$ alias del = rm [Return].....现在 del 是 rm 命令的别名
$ del xyz [Return].....删除 xyz 文件
```

【说明】

1. 现在起可以用 del 代替 rm 命令。
2. rm 命令并没有改变,并且还可以正常使用。
3. 定义别名采用定义变量同样的方法使用等号。

【实例】

将 ll 设为命令 ls -al 的别名。这样就可以键入 ll 来以长格式显示当前目录文件。

```
$ alias ll = "ls -al" [Return].....现在 ll 是 ls -al 命令的别名了
```

\$_..... 提示符

【注意】

1. 与变量赋值一样,等号两侧不能有空格。
2. 而且,如果被赋的值中有空格(如上面例子所示),则该值必须用引号引起来。

【实例】

键入不带参数的 `alias` 命令,可以查看当前系统中已经存在的别名。

```
$ alias [Return].....显示别名
alias ll ls -al .....显示系统所有的别名
$_ .....提示符
```

【实例】

用 `unalias` 命令移去指定的别名。

```
$ unalias ll [Return].....移去别名 ll
$ alias [Return].....查看一下目前系统中的所有别名
$_ .....别名已被移走,只显示提示符
```

8.6.6 命令行编辑(ksh)

Korn shell 使用户能够利用 `history` 文件中 `vi` 编辑器的行版本,编辑命令行或任何命令。这一特征增强了历史文件的功能。它使用户能够纠正和修改历史文件中以前的命令,并让用户更为简单地查找以前执行的命令。

打开命令行编辑选项

可以有 3 种方法打开命令行编辑选项:使用 `set` 命令、设置 `EDITOR` 或设置 `VISUAL` 变量。这 3 种方法的结果是一样的。

```
$ set -o vi [Return].....打开命令行编辑选项
$ EDITOR = /usr/bin/vi [Return].....打开命令行编辑选项
$ VISUAL = /usr/bin/vi [Return].....打开命令行编辑选项
```

使用 `vi` 风格的命令行编辑器

假设用户已经打开命令行编辑器选项,并且设定编辑器是 `vi`,下面讨论如何使用这一非常有用的特性。

`ksh` 命令行编辑器工作于当前命令行和用户的历史文件中。键入命令时,命令处于 `vi` 的文本输入状态。这与用户在使用 `vi` 编辑文件时 `vi` 编辑器初始化是相反的。用户按回车键来执行命令。而在 `vi` 编辑器中,用户可在任何时间键入 `[Esc]` 键切换到命令模式。当处于命令模式时,`vi` 编辑器的命令可以修改、删除和纠正用户命令行。

【说明】

1. 命令行 `vi` 编辑器是一个特殊的内部 `vi` 编辑器。
2. 用户可以使用 `j`(向下)和 `k`(向上)键来访问历史文件中的命令行。
3. 用户可以使用 `l`(向左)和 `h`(向右)键来在命令行移动光标。

【注意】

记住,在键入命令时,vi 处于输入状态。如果要使用 vi 命令键如 k 或 j 等,必须按 [Esc] 键来切换到命令模式。

【实例】

下面命令显示如何使用这一选项。

```

$ history [Return].....显示命令
10 Who am I
11 ls -l
12 pwd
13 vi myfirst
14 rm myfirst
15 history
$_ .....显示提示符
$ history 104 [Return].....显示从第 104 条命令起的命令
[Esc].....按 [Esc] 键
j .....按 j 键,在命令列表从当前命令移到上一条命令
vi myfirst .....指定的命令回显在终端
G .....显示历史文件中的最后一条命令
history .....回显该指定命令
10G .....回显历史文件中的第 10 条命令
who am I .....显示历史文件中的第 10 条命令

```

表 8.6 列出了部分可用于内部 vi 编辑器的命令。

表 8.6 内部 vi 编辑器命令

键	功 能
h 和 l	在命令行中将光标左移或右移一个字符
k 和 j	在历史文件中上移或下移一个条目
b 和 w	在命令行中将光标左移或右移一个字
\$	将光标移至行末
x	删除当前字符
dw	删除当前字
I 和 i	插入文本
A 和 a	追加文本
R 和 r	替换文本

【实例】

用户也可以调用真实的 vi 编辑器,使用它的全部功能与特性来编辑命令行。

```

$ cp xyz xyz.bak .....假设用户将编辑此命令
[Esc].....按 [Esc] 键进入命令模式
v.....按 v 键调用真实的 vi 编辑器

```

现在,用户可以使用 vi 编辑器编辑一个只有一行命令的文件,即用户当前命令行。用户可以使用 vi 来编辑用户命令或添加新命令。当退出 vi 时,Korn shell 将执行用户的命令。

用户可以使用 vi 来创建一个多命令的文件。

8.6.7 登录和启动: Korn shell 风格

如同标准 shell(sh)一样, Korn shell(ksh) 在用户登录时也读取用户主目录下的 .profile 文件。它执行用户希望在登录时执行的命令, 初始化用户在会话时用到的变量。 .profile 包含的典型的命令有 **date**、**who** 和 **calendar** 等提供登录、终端设置和变量定义等用户希望输送到环境的信息。

除用户主目录下的 .profile 文件外, ksh 还读取用户环境文件。环境文件并没有预先定义供 ksh 查找的文件名或位置。用户在 .profile 文件中用 ENV 变量定义环境文件的文件名和位置。例如, 如果用户的 .profile 文件包含如下行:

```
ENV = $HOME/mine/my_env
```

ksh 将在用户主目录的 mine 子目录中, 名为 my_env 的文件中查找用户环境文件。尽管不是必需的, 但是将环境文件定义为主目录下的 .kshrc 文件是一个很好的习惯。

```
ENV = $HOME/.kshrc
```

【说明】

1. 如果登录 shell 是 Korn shell, 用户可以在主目录下的 .profile 文件中定义所有的 ksh 变量和选项。

2. 如果用户的登录 shell 不是 ksh, 则应在 ENV 变量定义的文件中定义所有的 ksh 变量和选项。为了让系统读取用户自己的环境文件, 必须在 .profile 文件中定义 ENV 变量。

图 8.14 显示(使用 cat 命令)一个名为 .kshrc 的环境文件的例子。该 .kshrc 文件包含设置 vi 风格的命令行编辑器、打开 noclobber 和 ignoreeof 选项、设置历史文件长度为 10 个条目和设置 TMOU 选项为 600。用户可以用 vi 创建一个类似的文件。

```
$ cat .kshrc
set -o      vi
set -o      noclobber
set -o      ignoreeof
HISTSIZE = 10
TMOU = 600
$ _
```

图 8.14 一个名为 .kshrc 的环境文件的例子

8.6.8 将事件号码添加到提示符中(ksh)

有时, 了解 shell 给每个命令所赋的事件号码是很有用的。用户可以改变提示符, 以包含这一信息。例如, 如果键入:

```
PS1 = "! $"
```

叹号(!)将告诉系统从历史文件中读取最后一个事件的号码, 将它加 1 然后显示出来。随着用

户键入命令,提示符不断显示事件号码。

【实例】

改变用户提示符,使之显示事件号码。

```
$ PS1 = "! $" [Return].....改变提示符
6 $_.....新提示符
```

【说明】

该新提示符显示,用户将要输入的下一个命令事件号码是6。

```
$ PS1 = "[!] $" [Return].....按这种方式改变提示符
[6] $_.....新提示符
```

【说明】

通过将自己定义的提示符加入到.kshrc文件中,可以使它在每次登录时都显示。

8.7 UNIX 进程管理

第3章介绍了引导系统的进程。现在让我们进一步深入UNIX的内部进程,并且看一下它对进程运行的管理。

本章读者已经遇到进程这个概念。一个程序的执行被称为进程:读者可以称之为程序,但一旦它被调入到内存执行,UNIX称之为进程。

为了时刻掌握这些进程,UNIX为系统中的每个进程创建并维护一个进程表。这些进程表包含如下信息:

- 进程号
- 进行状态(准备好/等待)
- 该进程等待的事件号
- 系统数据区地址

进程由一个被称为fork的例程(系统调用控制)创建。由正在运行的进程调用fork,并且作为响应,UNIX复制该进程,创建2个同样的进程。调用fork的进程称为父进程,被fork创建的进程称为子进程。UNIX通过给父进程和子进程不同的进程ID(PID)来区分它们。

进程管理包括如下步骤:

- 父进程调用fork,因而启动这个进程(fork)
- 调用fork是一个系统调用。UNIX取得控制,并且调用进程的地址被复制到进程表的系统数据区。该地址也被称为返回地址,它可使父进程在调用完成后,在返回时知道从哪里再次取得控制
- fork复制调用进程,然后将控制还给父进程
- 父进程收到子进程的PID(一个正整数),而子进程收到返回代码0(代表负数,指示一个错误)
- 父进程收到一个正的PID,调用另一个名为wait的系统例程,然后转入休眠(sleep)状态。现在,父进程在等待子进程的完成(用UNIX术语讲就是在等待子进程死亡)

- 子进程取得控制并开始执行。它首先检查返回码;由于返回码是 0,子进程于是调用另一个系统例程,称为 **exec**。**exec** 进程的响应就是新程序覆盖子进程的数据区
- 新程序的第一条指令开始执行。当新程序执行完所有的指令后,它调用被称为 **exit** 的进程,然后子进程即死亡。子进程的死亡唤醒父进程,父进程再次接手控制

这一过程见图 8.15 至 8.18 中的描述。现举一个例子来澄清一下这一显得有些混淆的过程。假定 shell(sh 程序)正在运行中,用户键入一个命令,比如是 **ls**。现在研究一下 UNIX 运行这一命令所采取的步骤。

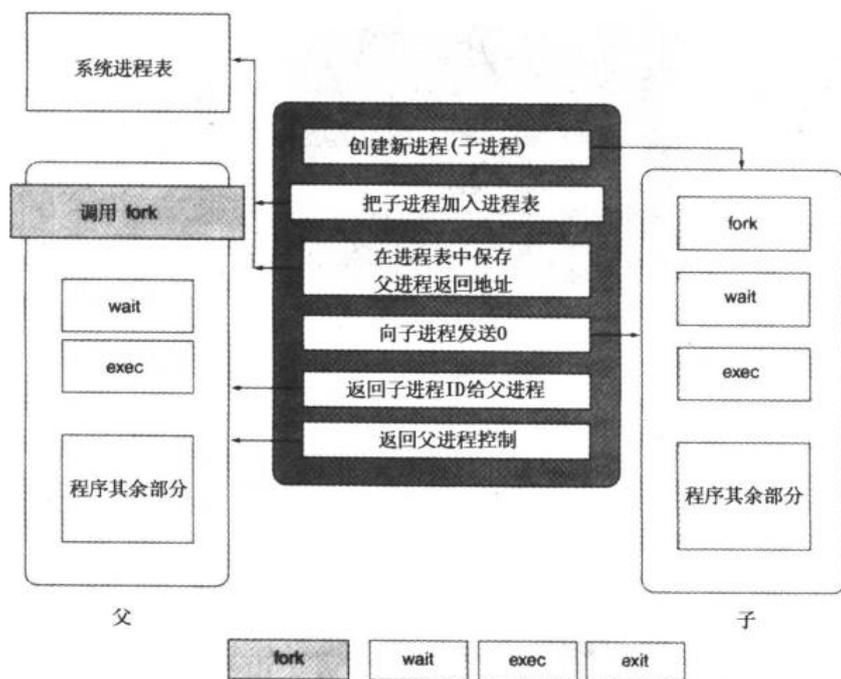


图 8.15 调用 fork 时发生的事件

shell 是父进程,在进行创建时,ls 变成子进程。父进程(shell)调用 **fork**。**fork** 例程复制父(shell)进程,如果成功创建子进程,则给予进程分配 PID,并将该 PID 加入到系统进程表中。下一步,父进程接到子进程 PID,子进程收到 0,接着系统控制权交回到父进程。shell 调用 **wait** 并转入等待状态(进入休眠)。同时,子进程获得控制并调用 **exec** 来用新程序覆盖子进程的数据区。在本例中,新程序即是用户键入的 **ls** 命令。现在 **ls** 完成命令,它列出用户当前目录中的所有文件名。在 **ls** 完成时,它调用 **exit**,于是子进程死亡。子进程的死亡产生一个事件信号。父进程(shell)一直在等待这一信号,因此父进程被唤醒并取得控制。shell 程序继续运行,从它进入休眠前的位置重新开始(调用存于进程表系统数据区的返回地址)。最后,显示提示符。

如果子进程是一个后台程序会怎样呢?在这种情况下,父进程(shell)将不会调用 **wait** 例程,而是继续在前台运行,因此,用户立即看到提示符。

【说明】

什么创建了第一个父进程和子进程?当 UNIX 被引导时,init 进程被激活。下一步,init 为每个终端创建一个系统进程。因此,init 是系统中所有进程的最原始祖先。例如,如果用户系统同时支持 64 个终端,则 init 创建 64 个进程。当用户登录到其中一个进程时,登录进程运行

shell。之后,当用户退出系统时(当 shell 死亡时),init 创建一个新的登录进程。

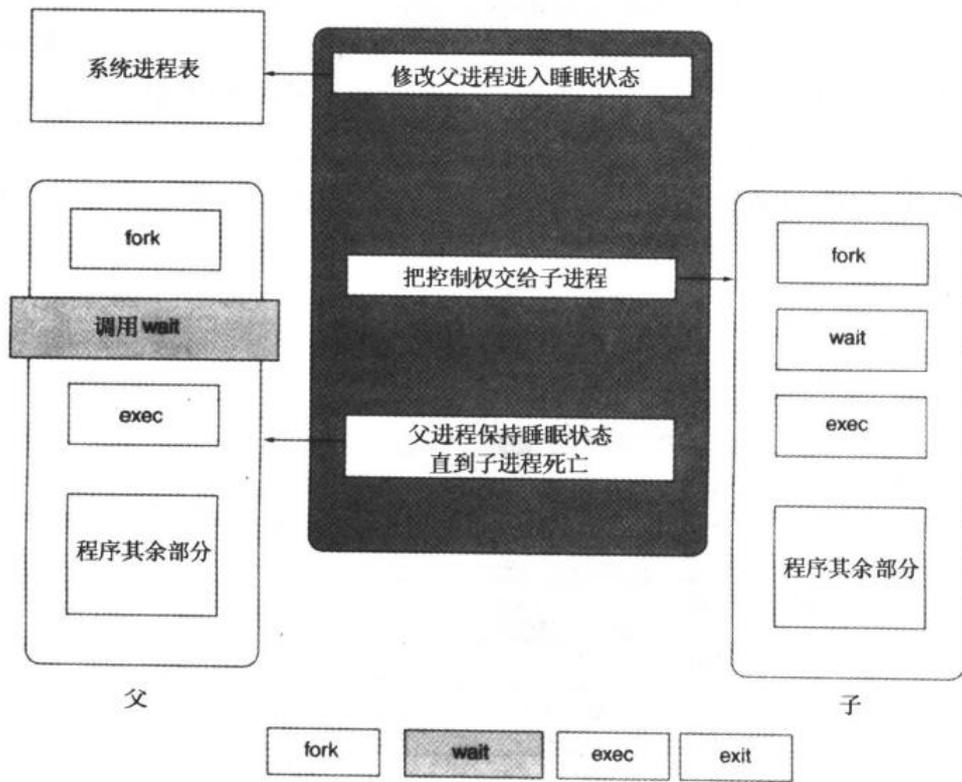


图 8.16 调用 wait 后发生的事件

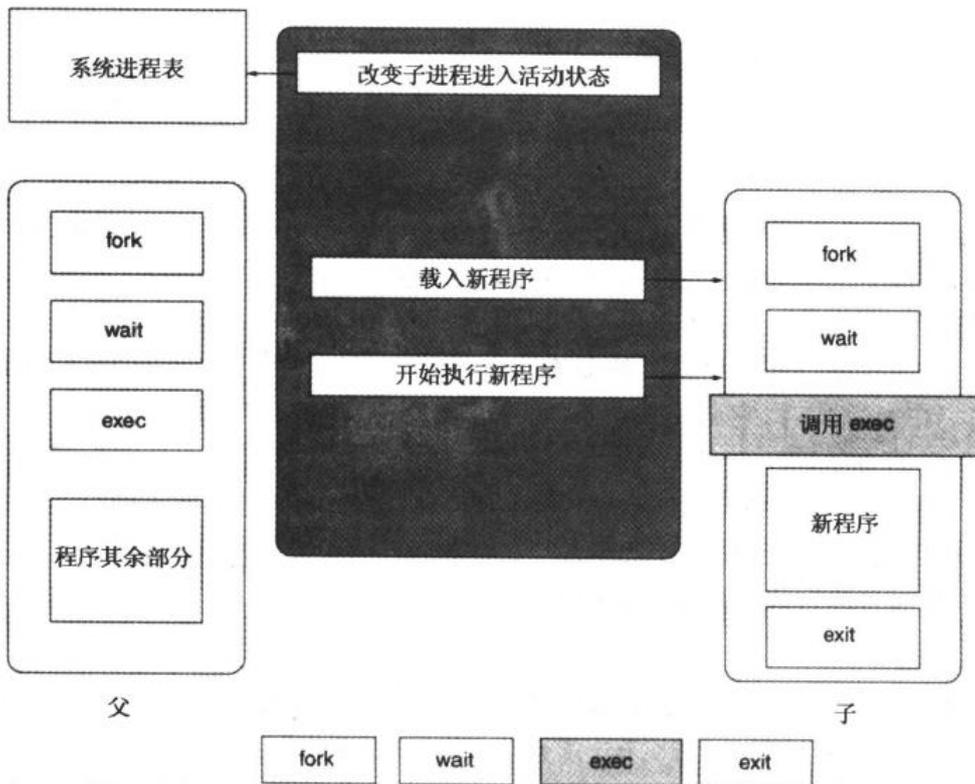


图 8.17 调用 exec 后发生的事件

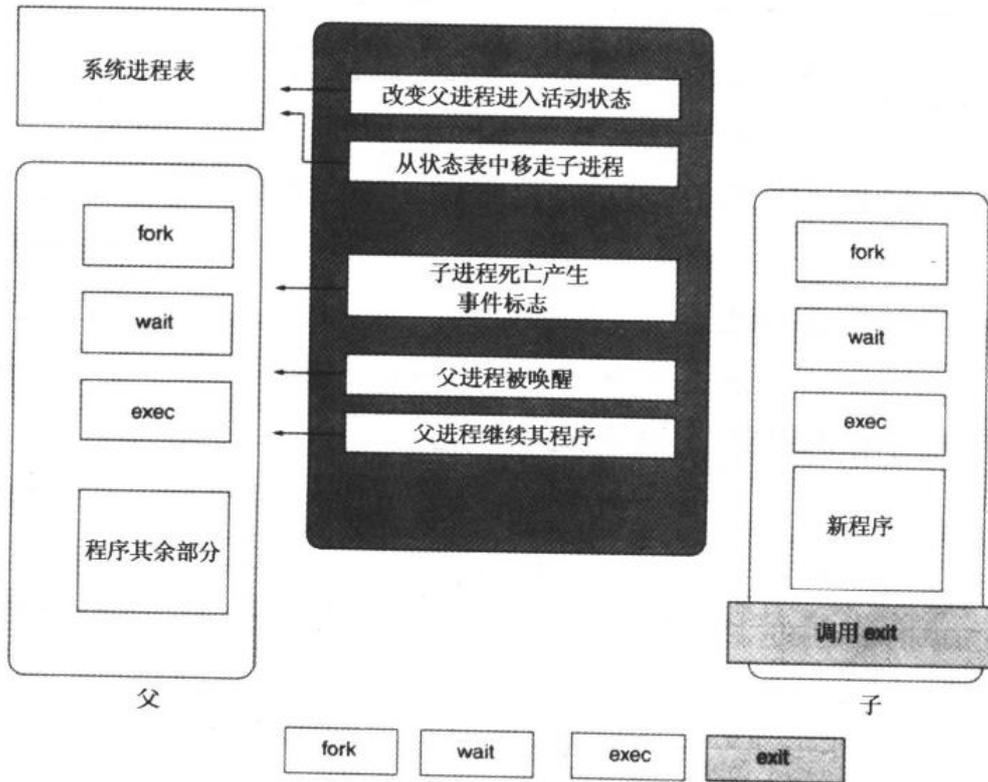


图 8.18 子进程调用 `exit` 时发生的事件

命令小结

本章中讨论了下面的 UNIX 命令。

echo

该命令将其参数显示在输出设备上。

扩展符	含义
<code>\n</code>	回车和换行符(新行)
<code>\t</code>	制表符
<code>\b</code>	退格
<code>\r</code>	回车但不换行

set

该命令在输出设备上显示环境/shell 变量。unset 命令删除不想要的变量。

ps(进程状态)

该命令显示与用户终端相关的进程 ID。

选项	操作
<code>-a</code>	显示所有激活的进程(不仅仅是用户的)
<code>-f</code>	显示全部信息,包括完整命令行

nohup

该命令使用户退出时不终止后台进程。

kill

终止不想要的或失去控制的进程。必须指定进程 ID。进程 ID0 终止与用户终端相关的所有进程。

export

将一些指定的变量输出给其他的 shell。

history(ksh)

该命令是 Korn shell 的特性,保留用户会话中键入的所有命令。

r(重新执行)

这是一个 Korn shell 命令。它重新执行最后一次执行的命令或历史文件中的命令。

alias(ksh)

为命令创建别名。

sleep

该进程进入休眠(等待)指定的时间(以秒计算)。

grep(Global Regular Expression Print)

在文件中搜索指定的样式。如果找到,则在用户终端显示包含指定样式的行。

选项	功能
-c	只显示每个文件中包含匹配样式的行数
-i	匹配时忽略大小写
-l	只显示包含匹配样式的文件名,不显示具体的行
-n	在输出的每行前显示行号
-v	只显示与样式不匹配的行

sort

将文本文件按不同顺序排序。

选项	功能
-b	忽略前导空格
-d	Z 按字典序排序。忽略标点符号和控制字符
-f	忽略大小写
-n	数字按数值大小排序
-o	将输出送往指定文件
-r	倒序。由升序变为降序

tee

分离输出。将输出送往用户终端(输出设备),同时存入一个文件中。

选项	功能
-a	将输出追加到一个文件中,对已有文件不覆盖原有内容 Z
-i	忽略中断,不响应中断信号

习题

1. shell 的主要功能是什么?
2. 你的系统的 shell 名称是什么? 它存放在哪里?
3. 什么是元字符? shell 如何解释它们?
4. 什么是引号?
5. 什么是 shell 变量?
6. 什么命令显示环境/shell 变量?
7. 什么命令删除一个变量?
8. 说出部分环境/shell 变量。
9. 什么是变量? 它们担任什么角色?
10. 如何在后台运行程序?
11. 如何中止一个后台进程?
12. 什么是进程 ID 数? 如何知道某特定进程的 ID 数?
13. 什么是管道符? 它起什么作用?
14. 如何防止在退出后,后台进程被终止?
15. 什么命令在文件中搜索指定的样式?
16. 如何延迟一个进程的执行?
17. 什么操作符将命令进行分组?
18. 什么是启动文件?
19. 什么是 .profile 文件? 什么是策略文件?
20. 在 UNIX 进程管理中什么是父进程? 什么是子进程?
21. 什么是进程?
22. 什么命令激活命令行编辑器?
23. 什么变量被用来改变历史文件的大小?
24. 如何使历史文件从 1 开始?
25. Korn shell 启动时读取哪个文件?
26. 什么命令重新执行最后一次执行的命令?
27. 什么命令重新执行历史文件中第 105 号事件?
28. 什么命令为命令创建别名?
29. 什么命令将一系列变量输送给其他 shell?

上机练习

在这个上机练习中,读者将实际练习本章中介绍的命令。下面的练习只是建议如何练习

这些命令。使用自己的例子并设计一些情况来掌握本章所学命令。

1. 使用 **echo** 命令来实现下面输出:

a. Hello There

b. Hello

There

c. "Hello There"

d. These are some of the metacharacters:

? * [] & () ; > <

e. File Name: file? Option: all

2. 使用 **echo** 和其他命令产生下面输出:

a. Display the contents of your current directory. Have a header that shows a short prompt and the current date and time before listing your directory.

b. Show the message "I woke up" with a two-minute time delay.

3. 改变用户基本提示符。

4. 创建一个名为 `name` 的变量, 将读者的名字存入该变量。

5. 显示变量名。

6. 查看主目录中是否有 `.profile` 文件。

7. 创建或修改已有的 `.profile` 文件, 以便在每次登录时显示如下内容:

```
Hello there
```

```
I am at your service David Brown
```

```
Current Date and Time:[the current date and time]
```

```
Next Command:
```

8. 创建一个后台进程, 查看它的 ID, 然后终止它。

9. 创建一个后台进程, 使用 **nohup** 在退出时不终止该进程。

10. 创建一个电话簿文件。假定收集你班上 10 位同学的姓名和电话号码存在这个文件中。使用 **sort** 命令对这个文件进行排序, 按名、按姓、按电话号码及反序排序等。

11. 使用 **grep** 命令及其选项在你的电话簿中查找一个名字。

12. 使用 **kill** 命令退出系统。

13. 如果你的 shell 是 Korn shell, 创建下面变量并练习 **ksh** 命令。

a. 把历史文件大小设置为 50。

b. 激活命令行编辑器。

c. 使用 **vi** 编辑器命令访问历史文件中的命令。

d. 使用 **vi** 编辑器命令编辑修改命令行。

e. 重复最后的命令。

f. 显示以前命令的列表。

g. 重复历史文件中的第一个命令。

h. 开始一个新的历史文件。

i. 为带确认选项的删除命令 (**rm-i**) 建立别名。

j. 创建 `.kshrc` 文件, 包含 **ksh** 变量和别名的设置。

第9章 UNIX 通信

本章将主要集中介绍 UNIX 的通信工具,描述如何与系统中其他用户通信,如何读取系统的新闻,如何广播消息给所有用户,解释 UNIX 电子邮件(e-mail)的功能,展示使用电子邮件的命令和选项。本章还描述 shell 和其他变量如何影响 e-mail 的环境,告诉用户如何做一个启动文件来定制 e-mail 工具集的使用。

9.1 通信方式

UNIX 提供了一组与其他用户通信的命令和工具。用户可以用一种非常简单的方式同其他用户进行通信,如通过邮件传递系统发送和接收邮件,或者广播消息给系统上的每个人。

与系统上的其他用户进行通信时请遵循以下一些准则:

- 尊重对方,不要有亵渎的言行
- 在发送前要认真考虑。不要等发送完后后悔
- 对发送出去的邮件做备份

9.1.1 全双工通信:write 命令

用户可以使用 `write` 命令与其他用户通信。这种通信方式是交互地从一个终端到另一个终端进行,因此接收方也必须是一个已登录的用户。用户发送的消息会出现在接收用户的屏幕上。那么接收用户可以在他的终端发一个 `write` 命令来回复消息。通过 `write` 命令,两个用户之间可以通过各自的终端进行有效的交谈。

下面通过一个例子一步一步地看看 `write` 是如何工作的。假定用户的 ID 是 `david`,想和 Daniel 交谈, Daniel 的 ID 是 `daniel`。

【实例】

键入 `write daniel`,然后按回车。

如果 Daniel 没有登录,在屏幕上会看到以下信息:

```
daniel not logged on.
```

如果 Daniel 已经登录了,他将在屏幕上看到一条类似下面的消息:

```
Message from david on (tty06) [Thu Nov 9:30:30]
```

在你的屏幕上,光标处于下一行,系统正等待键入消息。你也许要发送多行消息,在按回车键后,每行的消息就会发送给 Daniel。你可以在空行开头键入 `[Ctrl-d]` 表示消息的结束。这样就完成 `write` 命令,同时发送 EOT(结束传送)消息给 Daniel。

图 9.1 显示两个屏幕,描述一次典型的对话。上面是 David 的终端显示,下面是 Daniel 的终端显示。

```

$ write daniel [Return]
Hello Dan [Return]
Is today's meeting still on? [Return]
[Ctrl-d]
< EOT >
$

Message from david on (tty06) [Thu Nov 9:30:30]...
Hello Dan
Is today's meeting still on?
< EOT >
$

```

图 9.1 典型的屏幕对话:上面的屏幕是 David 的,下面的屏幕是 Daniel 的

【说明】

使用 `write` 命令,需要知道你想交谈的对方的 ID。通过 `who` 命令(请参考第 3 章)可以得到当前登录到系统上的用户 ID。

Daniel 可以使用 `write` 命令从他的终端上答复,但是他不必等到用户发完消息以后。当 Daniel 看到用户发给他的第一条消息后,就可以键入 `write david`,按回车键,回复消息,这时也许用户又在给 Daniel 发第二或者更多条消息。

由于 `write` 命令同时激活两个终端,用户和 Daniel 之间可以展开全双工的对话。有时这种全双工的对话容易混淆,因此有必要建立使用 `write` 的协议。UNIX 用户之间一般就是规定发送者在消息发送结束时加上字符 `o`(结束),通知对方一条消息发送结束,也许用户会等待回答。如果用户想结束谈话,就键入 `oo`(结束,退出)。

如果用户收到来自另外一个用户的 `write` 消息,那么这时无论用户正在干什么,消息会出现在屏幕上。如果用户正在使用 `vi` 编辑器,编辑工作做到一半,那么 `write` 消息就显示在光标处。不过别惊慌,这是一个端到端的通信,不会破坏用户正在编辑的任务。`write` 消息仅仅是覆盖在屏幕上的信息,用户可以继续编辑或者继续其他任何正在进行的工作。

无论如何,用户正在专注于一项工作的时候,收到一条消息总是不太方便,更不用提还会造成屏幕的混乱。用户可以禁止终端接收 `write` 消息。

9.1.2 禁止消息:mesg 命令

用户可以使用 `mesg` 命令作为开关,禁止终端接收来自 `write` 命令的消息,或者重新打开接收功能。`mesg` 命令不带参数显示的是系统这时处于何种状态。

【实例】

下面的命令顺序告诉用户如何保护自己不受干扰。

```

$ mesg [Return]..... 检查终端的状态
is y..... 处于 YES 状态,可以接收消息
$ mesg n [Return]..... 设置为 NO,停止接收消息
$ mesg [Return]..... 复核
is n..... 处于 NO 状态

```

\$..... 回到提示符

为了训练通信命令的使用,通常还需要另外一个用户参加练习。然而,用户可以独立完成大多数命令,使用自己的 ID 发送和接收消息。

9.1.3 显示新闻:news 命令

可以使用 `news` 命令看看系统正在发生什么。`news` 从系统的 `news` 目录中(通常在 `/usr/news` 下)读出信息。不带选择项时,`news` 命令显示 `news` 目录下用户还没有看过的所有文件。`news` 命令参考并且更新用户 `home` 目录下的 `.news_time` 文件,该文件是用户第一次使用 `news` 命令时创建的空文件。`news` 命令根据该文件的存取时间得到用户上次是何时使用 `news` 命令的。

【说明】

1. 可以按下中断键(通常是[Del]键)停止显示一条新闻,开始显示下一条新闻。
2. 键入中断键两次就退出 `news` 命令。

【实例】

为了检查最新新闻,键入 `news`,按回车键。图 9.2 显示了几条新闻样例。

```
$ news [Return]
david (root) Wed Nov      28   14:14:14 2001
  Let's congratulate david; he got his B.S. degree.
  Friday night party on second floor. Be there!

books (root) Wed Nov      28   14:14:14 2001
  Our technical library is growing.
  New set of UNIX books is now available.

$_
```

图 9.2 news 命令

每条新闻项有一个头,包含文件名、文件的创建者、文件放入 `news` 目录的时间。

news 选项

`news` 选项的总结在表 9.1 中。这些选项不更新用户的 `.news_time` 文件。

表 9.1 news 命令选项

选 项	操 作
-a	显示所有的新闻项,无论是新的还是旧的
-n	仅仅列出新闻名
-s	显示当前新闻的总数

【实例】

下面的命令顺序显示 `news` 命令加上选项是如何工作的。

- 列出系统当前的新闻项,使用 `-n` 选项:键入 `news -n`,然后按回车键。UNIX 仅仅显示这些文件的头。文件头包含文件名、文件的创建者、文件放入 `news` 目录的时间。图 9.3 显示命令的输出。

□ 为了显示具体新闻的内容,键入 `news david`,按回车键。例如,图 9.4 显示了新闻样例的输出。

```
$ news -n [Return]
david (root) Wed Nov    28  14:14:14  2001
books (root) Wed Nov    28  14:14:14  2001
$_
```

图 9.3 news 命令加-n 选项

```
$ news david [Return]
david (root) Wed Nov    28  14:14:14  2001
Let's congratulate david; he got his B.S. degree.
Friday night party on second floor. Be there!
$_
```

图 9.4 news 命令加特定的新闻项

【说明】

新闻名就是 news 目录下的文件名。

9.1.4 广播消息:wall 命令

用户可以使用 `wall`(write all)命令给当前登录到系统上的所有用户发送消息。`wall` 命令从键盘(标准输入)读入消息,直到用户在空行的开头键入[Ctrl-d],表示消息的结束。`wall` 的可执行文件放在 `/etc` 目录下,没有定义到 `PATH` 标准变量中,这意味着用户需要键入整个目录路径才能激活 `wall`。

`wall` 命令通常由系统管理员使用,用于通知用户一些紧急事件。也许有些用户没有权限使用该命令。

【实例】

假定用户的登录名为 david(而且有使用 `wall` 的权限),给所有用户发送一条消息:

- 输入 `/etc/wall`,然后按回车键。
- 输入 `Alert...`,然后按回车键。
- 输入 `Lab will be closed in 5 minutes. Time to log out.`,然后按回车键。
- 键入[Ctrl-d]。系统的反应如图 9.5。

```
$ /etc/wall [Return]
Alert ... [Return]
Lab will be closed in 5 minutes. Time to log out. [Return]
[Ctrl-d]
$_

Broadcast message from david
Alert ...
Lab will be closed in 5 minutes. Time to log out.
```

图 9.5 使用 wall 命令

【说明】

1. 广播的消息也会发送给发送者本身,因此用户会看见自己广播出去的消息。
2. `wall` 命令发送出去的消息不会被那些将 `mesg` 设为 `n` 的用户接收。
3. 系统管理员不受权限的限制。
4. 用户发送的广播消息中带有用户的 ID,因此不能发送匿名消息。

9.1.5 全双工通信:talk 命令

用户可以使用 `talk` 命令与系统中已登录的其他用户通信。这个命令和 `write` 命令很相似。键入 `talk`,用户的屏幕就会分成两半。上面显示用户自己输入的信息,下面显示另外一方输入的信息。举例来说,如果用户输入 `talk daniel`,按回车键后,用户的屏幕就会分成两半,屏幕上显示如下信息:

```
[waiting for your party to respond]
```

在 Daniel 的屏幕上会显示这样的消息:

```
Message from Talk_daemon@xyz at 11:22 ..
talk:Connection requested by david@xyz
talk:Respond with:talk david@xyz
```

如果 Daniel 如此回应:

```
talk david@xyz
```

那么 Daniel 的屏幕就会分成两半,双方就可以对话。

为了结束对话,任何一方只要键入[`Ctrl-c`],双方的终端上就会显示如下信息:

```
[Connection closing. Exiting]
```

如果 Daniel 使用 `mesg` 命令拒绝消息,那么用户屏幕上会显示如下的消息:

```
[Your party is refusing messages]
```

如果 Daniel 没有登录,那么屏幕上会显示如下信息:

```
[Your party is not logged on]
```

如果用户是 David,想和 Daniel 通信,图 9.6 显示了对话的内容。

```
[Connection established]                                david's screen
Hi dan!
Is today's meeting still on?
-----
Hi Dave
Yes. Be there!
Bye.
```

```

[Connection established]                                daniel's screen
Hi Dave!
Yes. Be there!
Bye.
-----
Hi Dan
Is today's meeting still on?

```

图 9.6 屏幕对话:上面是 David 的,下面是 Daniel 的

9.2 电子邮件

电子邮件(e-mail)是现代办公环境中必不可少的一部分。e-mail 使得用户能够给其他用户发送或者从其他用户接收消息、便笺和文档。用 e-mail 服务发送邮件和用 `write` 命令的主要区别在于:只有用户登录到系统上时,由 `write` 命令发送来的消息用户才能收到。但是对于 e-mail 来说,系统会自动帮你保存邮件直到用户读取。e-mail 服务比传统的信笺传递快捷了许多,而且也不会像电话一样,马上打断对方的活动。

在 UNIX 下,`mail` 和 `mailx` 命令都可以接收或者发送邮件。`mailx` 是基于 Berkeley UNIX 的 `mail`,支持用户简单而有效地操作(浏览、保存、删除等)邮件,比 `mail` 的功能强大。本书讨论 `mailx`,它有许多特性和选项,使用 `mailx` 需要用户有丰富的 UNIX 经验。本章将详细讲解如何使用 `mailx`,使用户能很流利地使用 `mailx`,而且会很有兴趣得到更多关于使用 `mailx` 的信息。

【说明】

从哪儿能得到更多的信息? `man` 命令如何?(如果忘记了,请参考第 3 章的讨论)

使用 `mailx` 命令可以读取别人发来的邮件,也可以发送邮件给别人。`mailx` 的操作需要很多文件,它显示的方式和功能依赖于文件中设定的环境变量,而且邮件也是存储到文件中的。

9.2.1 使用邮箱

在 UNIX 邮件系统中,有两类邮箱:系统邮箱和私人邮箱。

系统邮箱

系统中的每个用户都有一个 `mailbox`,它是一个文件,文件名就是用户的登录 ID。这个文件一般放在 `/usr/mail` 下。所有发送给用户的邮件都储存在这个文件里。用户读取消息时,`mailx` 就读这个文件。假定用户的 ID 是 `david`,那么邮箱的路径和名字可能是 `/usr/mail/student/david`。

可以使用 `set` 命令(参见第 8 章)查看系统的邮箱路径和名字。变量 `MAIL` 设置接收用户邮件的文件名。

私人邮箱

等用户读完邮件后,`mailx` 自动把邮件复制一份,追加到 `home` 目录下的 `mbox` 文件中。如

果 `mbox` 文件不存在, `mailx` 会在用户第一次读取邮件时生成这个文件。这个文件包含所有用户读过的邮件, 只要 `$HOME/mbox` 没有删除或者保存到其他地方就行。变量 `MBOX` 控制文件名, 默认值是 `HOME/mbox`。例如, 下面的命令改变默认值, 设置私人邮箱位置到 `e-mail` 目录下。

```
MBOX = $HOME/EMAIL/mbox
```

【注意】

明确地保存消息或者使用 `x(exit)` 命令退出 `mailx`, 都禁止了消息的自动保存。

定制 `mailx` 环境

用户可以通过设定两个启动文件中适当的变量, 定制自己的 `mailx` 环境。这两个启动文件分别为: 系统目录下的 `mail.rc` 和用户 `home` 目录下的 `.mailrc` 文件。

调用 `mailx`, 首先检查 `mail.rc`。 `mail.rc` 的路径名类似于:

```
/usr/share/lib/mailx/mail.rc
```

这个文件是由系统管理员创建维护的。这个文件中变量的设置适用于系统中的所有用户。

`mailx` 查找的另外一个文件就是用户 `home` 目录下的 `mailrc` 文件。用户可以通过设置 `.mailrc` 文件中的变量来改变由系统管理员在 `mail.rc` 设置的变量, 从而改变 `mailx` 的环境。这个文件不是必需的, 只要用户对系统管理员设置的环境满意, 没有它 `mailx` 也可以正常工作。定制 `mailx` 环境的方法将在 9.5 节详细介绍。

9.2.2 发送邮件

为了给另外一个用户发送邮件, 需要那个用户的 ID。例如, 如果想给登录 ID 为 `daniel` 的用户发送邮件, 输入 `mailx daniel`, 按回车键。

默认情况下, `mailx` 发送的消息来自键盘输入(标准输入)。根据系统环境变量的设置, `mailx` 可能会显示 `Subject:`。如果是这样, 用户需要输入邮件的主题, 然后 `mailx` 切换到输入方式, 等待用户输入消息的其余部分。用户可以在最后一行键入 `[Ctrl-d]` 结束消息。 `mailx` 显示 `<EOT>` (end of transmission), 消息发送出去。

在输入模式下, `mailx` 给用户提供了很多简单而有效地组织消息的命令。所有命令以 `~` 符号开头, 称为 `~` 转义命令(本章后面介绍 `~` 转义命令)。

【实例】

如下是给 `Daniel`(登录名 `daniel`) 发消息。

```
$ mailx daniel[Return]..... 给 daniel 发消息
Subject: meeting[Return]..... 输入主题
Hi, Dan [Return]
Let me know if tomorrow's meeting is still on.[Return]
Dave[Return]
[Ctrl-d]..... 结束消息
EOT..... mailx 显示传输结束
```

\$_. 回到提示符

【说明】

1. 主题栏是可选的。可以直接按回车键跳过 **Subject:** 提示。
2. 在空行键入 [Ctrl-d] 结束消息。
3. 邮件被转发到其他用户的信箱中, 而不考虑该用户当前是否登录。
4. 只要接收者登录, 系统会提示他有邮件。在终端上显示以下消息:

```
You have mail.
```

9.2.3 读邮件

用户使用不带参数的 **mailx** 命令, 就可以阅读邮件。如果用户的邮箱中有邮件, **mailx** 显示两行信息, 后面加上所有消息的头列表, 最后显示 **mailx** 提示符, 该提示符默认为问号。此时, **mailx** 处于命令模式, 用户可以使用删除消息、保存消息或者回复消息命令。在 ? 提示符后输入 **q** 退出 **mailx**。

头列表由多行组成, 一行代表一条消息, 格式如下:

```
> status message # sender date lines/characters subject
```

每行表达邮件的一些信息:

- 代表该消息是当前消息
- **status** 是 **N**, 表示消息是新的, 意味着用户还没有读过
- 如果消息不是新的, **status** 是 **U** (**unread**), 表示用户以前已经看到这条消息头, 但是还没有读过这条消息
- **message #** 表示邮件在邮箱中的顺序号
- **sender** 是发送者的登录 ID
- **date** 显示了邮件到达邮箱的日期和时间
- **line/character** 显示了邮件的行数和字符数

【实例】

假定用户是 **Daniel**, 登录到系统, 想读邮件。

- 系统提示用户有新的邮件:

```
You have mail
```

- 为了读邮件, 键入 **mailx**, 按回车键, UNIX 响应:

```
mailx version 4.0 Type ? for help.
"/usr/students/mail/daniel": 1 message 1 new
> N 1 david .... Thu Nov 28 14:14 8:126 meeting
```

【说明】

1. 头行显示了 **mailx** 的版本号, 提示可以按下 ? 求助。
2. 第 2 行显示系统邮箱的路径名, 后面跟着消息的数目和状态。在这里, 用户有一条消

息, N 代表这是用户第一次读这条消息头。

? 提示符表示 **mailx** 处于命令模式。输入邮件在邮箱中的顺序号就可以阅读指定的邮件内容。用户也可以直接按回车键, 开始阅读当前的邮件(在行头上有 > 标志), 然后在 ? 提示符后按回车键就可以顺序阅读邮件。

```
?.....mailx 处于命令模式
? 1[Return].....显示消息 1, 邮箱中仅有的邮件
Message 1:
From :david Thu, 28 Nov 01 14:14 EDT 2001
To:daniel
Subject: meeting
Status: R
Hi, Dan
Let me know if tomorrow's meeting is still on.
Dave
?.....准备接收下一条命令
? q[Return].....退出 mailx
Saved 1 message in /usr/student/david/mbox
$_.....回到 shell
```

【说明】

如果使用 **q** 退出 **mailx**, 系统会把邮件复制一份保存到用户 home 目录下私人邮箱 **mbox** 中(这样, 此时系统邮箱就空了)。

【实例】

假定用户此时想再次读取邮件。键入 **mailx**, 按回车键, UNIX 系统反应如下:

```
No mail for daniel
```

【实例】

查看邮箱 **mbox** 的内容。

```
$ cat mbox[Return].....查看邮箱 mbox 的内容
From :david Thu, 28 Nov 01 14:14 EDT 2001
To: daniel
Subject: meeting
Status: RO
Hi, Dan
Let me know if tomorrow's meeting is still on.
Dave
```

正如前述, **mbox** 包含邮件的备份。

9.2.4 退出 **mailx:q** 和 **x** 命令

退出 **mailx** 只要在 ? 提示符后输入 **q**(quit) 或 **x**(exit) 即可。但是, 尽管两个命令都能退出 **mailx**, 但是它们在退出方式上却有很大区别。

q 命令自动把用户读过的邮件从系统信箱中移走。默认情况下, 系统复制一份邮件, 然后保存在用户的私人邮箱中(文件名由环境变量 **MAIL** 指定)。

x 命令不会把用户读过的邮件从系统信箱中移走。实际上,如果用户使用 x 命令,邮箱中什么也不变,即使是已删除的邮件也完好无损。

mailx 选项:表 9.2 总结了 mailx 选项。这些选项使用在命令行,当用户调用 mailx 发送或者接收邮件时。下面的命令顺序地展示如何使用 mailx 选项。

```

$ mailx -H[Return].....仅仅显示消息头
N 1 daniel      Thu ESP 30 12:26 6/103 Room
N 2 susan      Thu Sep 30 12:30 6/107 Project
N 3 marie      Thu Sep 30 12:30 6/70 Welcome
$_.....回到 shell

```

表 9.2 mailx 选项

选 项	操 作
-f 文件名	从指定的文件中而不是系统邮箱中读取消息。如果没有指定文件名,就从 mbox 中读取
-H	显示消息头列表
-s 主题	设置邮件的主题

【实例】

输入 mailx -f mymail ,按回车键,从指定的文件 mymail 中而不是系统邮箱中读取邮件。UNIX 响应如下:

```

/usr/students/daniel/mymail:No such file or directory

```

mailx 默认情况下从系统邮箱中读取邮件。带上-f 选项,它就从指定的文件中读,譬如说用户的旧邮件文件。在这里,用户指定从 mymail 中读取消息,但是 mymail 不在当前目录下。

如果用户输完 mailx -f 后,就按回车键,没有指定文件名,UNIX 默认就从用户的私人信箱中读,因此会看到下面的信息:

```

mailx version 4.0 Type ? for help.
"/usr/students/mail/daniel": 1 message 1 new
> N 1 david .... Thu Nov 28 14:14 17/32 meeting

```

【实例】

为了使用-s 选项,把主题作为命令行的部分,给 Daniel(ID 为 daniel)发送邮件:

```

$ mailx -s meeting daniel[Return].....给 daniel 发信,主题设置为 meeting
.....输入消息内容
[Ctrl-d].....结束消息
EOT.....显示传送结束
$_.....回到 shell

```

当 Daniel 读取消息时,主题栏显示:Subject: meeting。

【说明】

如果主题字之间有空格,需要用引号引起来。

9.3 mailx 输入模式

当 mailx 处于输入模式(输入邮件内容)时,也有许多命令可供使用。这些命令以(~)开

头,使得用户暂时与输入模式区别开,能够调用命令,这就是为什么被称为转义命令。表 9.3 总结了部分转义命令。

表 9.3 mailx 转义命令

命 令	操 作
~?	显示所有转义符命令列表
~! 命令	在编辑邮件时,可以调用指定的 shell 命令
~e	调用编辑器,编辑器的名字可以由变量 EDITOR 定义。默认为 vi
~p	显示目前正在编辑的消息
~q	退出输入模式,把编辑了部分的消息保存到 dead.letter 文件中
~r 文件名	读取指定的文件,把它的内容插入到消息中
~< 文件名	读取指定的文件(使用重定向符),把它的内容插入到消息中
~<! 文件名	执行指定的命令,把它的输出插入到消息中
~v	调用默认编辑器 vi,或者使用邮件变量 VISUAL,指定为其他编辑器
~w 文件名	把正在编辑的消息写到指定的文件中

上面有些命令很重要。假定用户想编辑一个数行的消息,使用 mailx 提供的编辑器很不方便。不用这个编辑器,用户可以调用 vi 编辑器。使用 vi 简单而强大的功能编辑消息,等消息编辑完后,退出 vi,回到 mailx 的输入模式。然后用户可以调用其他命令或者直接发送消息。

【注意】

1. 转义命令仅使用在 mailx 处于输入模式时。
2. 所有的命令必须在一行的开头输入。

【实例】

下面的命令顺序地显示 mailx 处在输入模式时,如何使用转义命令。假定用户的 ID 为 david,给自己发送邮件。

```
$ mailx -s "Just a Test" david[Return].....给自己发邮件
```

mailx 带着-s 选项。引号是因为主题字之间有空格。

```
$.....mailx 处于输入模式
~! date[Return].....调用 date 命令
Wed, Nov 28 16:16 EDT 2001
..... mailx 等待输入
```

这里用户使用了 ~!, 执行了 date 命令。当然用户可以任意执行他想执行的命令。命令的输出不会插入到正在编辑的消息中。

```
$~ <! date[Return].....执行 date 命令,将输出重定向到正在编辑的消息中
"date" 1/29.....返回的信息
.....准备好了,等待用户的输入
```

返回的信息指一行有 29 个字符插入到消息中(date 命令的输出)。

```

This is a test message to explore mailx.[Return]
$~ v[Return].....现在调用 vi 编辑器编辑消息的剩余部分

```

在此, vi 编辑器运行了, 用户已经编辑的消息输入到 vi 中。

```

Wed, Nov 28 16:16 EDT 2001
This is a test message to explore mailx capabilities.
~
~
~/tmp/Re26485" 2 line, 69 characters

```

现在用户可以使用 vi 提供的一切功能。用户可以删除、修改或者保存文件。可以执行命令或者导入另外一个文件, 然后继续编辑消息。

```

Wed, Nov 28 16:16 EDT 2001
This is a test message to explore mailx capabilities.
This message is composed using the vi editor.
:wq[Return].....退出 vi
3 lines, 115 characters..... 给出 vi 的反馈信息
(continue).....返回消息
..... 回到 mailx 的输入模式
~ w first.mail[Return].....把邮件保存到 first.mail 文件中
"first.mail" 3/115.....得到反馈信息

```

反馈信息表示 first.mail 文件的大小为 3 行, 115 个字符。

~w 命令将正在编辑的消息保存到指定的文件 first.mail 中。这种练习是好的习惯, 这样用户就有发送出去邮件的备份。如果用户设置了 mailx 中的记录变量, 那么发送出去的邮件自动作备份。

```

~q[Return].....退出 mailx 的输入模式
$_.....回到 shell

```

使用 ~q 命令退出 mailx 输入模式, 会把编辑好的消息保存到用户 home 目录下的 dead.letter 文件中(或者其他由 DEAD 变量指定的文件)。

【实例】

下面重新开始, 完成邮件的发送。情况如下: 用户是 david, 想给自己发送邮件。用户有一份消息的备份在文件 first.mail 中(~w 命令保存), 而且还有一份备份在 dead.letter 中(~q 退出输入模式产生的)。

发送 first.mail 给 david, 在命令行使用输入重定向符(<)。

```

$ mailx david < first.mail[Return].....发送邮件
$_.....完成

```

< 符号要求 shell 将指定的文件 first.mail 作为 mailx 命令的输入。

【说明】

用户可以指定更多的输入文件。

默认情况下, Bourne shell 每 10 分钟检查新邮件一次, 因此用户需要等待一会儿才能读到

刚才发给自己的邮件。如果用户邮箱中有新邮件,UNIX 会在提示符的上面显示 you have mail 消息。

【实例】

为了发送 first.mail 给 david,使用下面的转义命令:

```
$ mailx david[Return].....给自己发送邮件
Subject:.....等待输入主题
~ < first.mail[Return].....从 first.mail 读
"first.mail" 3/115.....得到反馈消息
[Ctrl-d].....发送
EOT.....mailx 显示 EOT
$_.....回到 shell
```

【说明】

~ < 从指定的文件中读,这里是 first.mail,然后将内容插入到正在编辑的消息中。也可以使用 ~r 命令得到同样的效果。

【实例】

把 mailx 保存在 dead.letter 文件中部分编辑好的消息打开,完成编辑,然后发送给 david。

```
$ mailx david[Return].....给 david 发送邮件
Subject:.....等待输入主题
~ r dead.letter[Return].....从 dead.letter 读
~ v[Return].....调用 vi 编辑器
Wed, 28 Nov 01 16:16 EDT 2001
This is a test message to explore mailx capabilities.
This message is composed using the vi editor.
~
~
"/tmp/Re265" 3 line and 115 characters.....vi 编辑器反馈消息
```

假定用户想插入几行到消息中。

```
Wed,28 Nov 01 16:16 EDT 2001.....完成消息
This is a test message to explore mailx capabilities.
This message is composed using the vi editor.
This is the first time I am using email. Maybe I should save this
text, get a copy of it, and frame it!
~
~
:wq[Return].....保存并退出 vi
(continue).....得到反馈信息
```

这里没有显示用户的消息,用户在发送之前可以用 ~p 命令看看自己编辑的邮件内容。UNIX 一次显示整个邮件的内容。

```
~ p[Return].....显示邮件
Wed, 28 Nov 01 16:16 EDT 2001
This is a test message to explore mailx capabilities.
This message is composed using the vi editor.
This is the first time I am using email. Maybe I should save
```

```

this text, get a copy of it, and frame it!
[Ctrl-d]..... 结束
EOT..... 传送结束
$_..... 回到 shell

```

9.3.1 发送已有的文件

其实用户根本就没必要使用 **mailx** 编辑器编辑要发送的消息。也许用户已经有一份写好的便笺想发送给另外一个用户。这样,可以使用 shell 重定向把 **mailx** 的默认输入从原来设备(键盘)重新定向到指定的文件。

【实例】

把 memo 文件发送给 Daniel,其 ID 是 daniel。

```

$ mailx daniel < memo[Return]..... 邮寄 memo 给 daniel
$_..... 回到 shell

```

假定用户想发送的邮件在文件 memo 中,想给 ID 为 daniel 的用户发送,上面的命令就完成这样的工作。

9.3.2 给一群用户发送邮件

假如想把 memo 文件发送给 daniel 和其他几个用户该怎么办呢?把同一个消息发送给 10 个不同的用户,使用 **mailx** 命令很不方便。那样的话,用户给出接收邮件的用户 ID 列表,然后 **mailx** 把邮件发送给所有人。

【实例】

把 memo 发送给 ID 分别为 daniel、susan、emma 的用户。输入 **mailx daniel susan emma < memo**,然后按回车键。

【说明】

用户 ID 之间用空格隔开。

如果用户经常给一群人发送邮件,可以定义一个用户列表,然后每次使用定义名而不用敲入所有接收者,这样会节省很多输入工作。**alias** 命令可以定义用户列表,其格式如下:

```
alias [name] [userID01] [userID02] [userID03] [userID04] [userID05] [userID06]
```

这里 name 是给列表中的用户发送邮件时输入的名字。

例如,假如用户输入 **alias friends daniel david marie gabe emma steve**,然后按回车键,UNIX 设置 friends 代表用户列表。现在,不需要输入所有的 ID,只要输入 **friends**。如果将 **alias** 命令写入 .mailrc 文件中,那么就成为 **mailx** 环境的一部分,这样就没必要每次在使用 **mailx** 时,都需要使用 **alias** 设置一次。

【实例】

假定已经设置 friends 代表用户朋友们的 ID,把 memo 文件发送给朋友们。输入 **mailx friends < memo**,然后按回车键。

9.4 mailx 命令模式

用户读取邮件时,mailx 处于命令模式。? 提示符说明系统在等待输入命令。当 mailx 处于命令模式时,有许多命令可供调用,可以复制、保存或者删除邮件。用户不用离开命令模式就可以回复发送者或者给指定的用户发送邮件。表 9.4 总结了部分命令。

表 9.4 命令模式下的 mailx 命令

命 令	功 能
!	执行 shell 命令(shell 转义)
cd [目录]	进入指定的目录,如果没有指定目录,就进入 home 目录
d	删除指定的消息
f	显示消息的头行
q	退出 mailx,把消息从系统邮箱中移去
h	显示当前消息的头
m 用户	发送邮件给指定的用户
R 消息	给消息的发送者回复
r 消息	给消息的发送者和同一消息的其他所有接收者回复
s 文件名	保存消息到指定的文件中
t 消息	显示指定的消息
u 消息	恢复被删除的消息
x	退出 mailx,不把消息从系统邮箱中移去

场景:下面顺序地显示如何使用 mailx 读邮件以及命令模式下有哪些命令和选项。假定用户的 ID 是 david,系统邮箱中有 3 条消息,用户想阅读、显示、删除邮件。mailx 命令提供给用户对邮件各种不同的操作。仅存的问题就是用户如何根据自己的需要使用合适的命令。

9.4.1 阅读/显示邮件

mailx 命令提供了几种不同的方法阅读或者显示邮件:每次显示一条消息,指定范围内的消息或者指定的单个消息。一般来说,无论用户打算怎么阅读邮件,都要显示邮件的列表,这可以简单地通过 mailx 实现。在? 提示符后直接按回车键显示当前的消息。

【实例】

阅读/显示邮件。

```
$ mailx[Return].....调用 mailx
mailx version 4.0 Type ? for help.
"/usr/student/mail/david": 3 messages 3 new
> N1 daniel          Thu ESP 30 12:26 6/103 Room
  N2 susan           Thu Sep 30 12:30 6/107 Project
  N3 marie           Thu Sep 30 12:30 6/70 Welcome
? [Return].....显示当前消息
Message 1:
```

```

From: daniel Thu Sep 30 12:26 EDT 2001
To: david
Subject: Room
Status: R
Room 707 is reserved for your meetings.
?.....等待下一条命令

```

在? 提示符后直接按回车键显示当前的消息,此处是消息 1。

```

? 3[Return].....显示消息 3
Message 3:
From: marie Thu Sep 30 12:30 EDT 2001
To: david
Subject: Welcome
Status: R
Welcome back!
?.....等待下一条命令
? t 1-3[Return].....显示消息 1 到 3
Message 1:
From: daniel Thu Sep 30 12:26 EDT 2001
To: david
Subject: Room
Status: R
Room 707 is reserved for your meetings.
Message 2:
From: susan Thu Sep 30 12:26 EDT 2001
To: david
Subject: Project
Status: R
Your project is in trouble! See me ASAP.
Message 3:
From: marie Thu Sep 30 12:30 EDT 2001
To: david
Subject: Welcome
Status: R
Welcome back!
?.....等待下一条命令

```

t(type)命令逐个显示消息。消息的范围是一个低数和一个高数,中间加上一个连字号决定。这里,t 1-3 表示显示消息 1、2、3。

```

? t 1 3[Return].....显示消息 1 和 3
Message 1:
From: daniel Thu Sep 30 12:26 EDT 2001
To: david
Subject: Room
Status: R
Room 007 is reserved for your meetings.
Message 3:
From: marie Thu Sep 30 12:30 EDT 2001
To: david
Subject: Welcome
Status: R

```

```
Welcome back!
? _.....等待下一条命令
```

【说明】

t命令还可以通过指定消息的序号显示消息。如果多于1条消息,消息序号之间用空格隔开。这里,**t 1 3**显示消息1和3。

```
? n[Return].....显示下一条消息
At EOF.....反馈信息
? _.....提示等待下一条命令
```

n(next)命令显示邮箱中的下一条消息,与直接按回车键功能一样。因为没有下一条消息存在,所以,提示 **End of the File**。

```
? n10[Return].....显示第10条消息
10:.....无效的消息号
```

可以直接指定消息号显示指定的消息。**n10**意思是显示消息10。但是消息10不存在,因此出现了错误。

```
? f [Return].....显示当前消息的头行
. 3 marie Thu Sep 30 12:30 EDT 2001
? _.....回到提示符
```

f命令显示当前消息的头行,此处显示的是消息3。

```
? x [Return].....退出 mailx
$_.....回到 shell 提示符
```

如果使用 **x** 退出 **mailx**, 邮箱中的消息还保留在那。

9.4.2 删除邮件

mailx 给用户提供几种删除命令:一次只删除一条消息,一次删除所有消息,一次删除指定范围的消息,恢复因为不小心而误删的消息。下面分别举例说明。场景与上面的一样,用户是 **david**, 系统邮箱中有3条消息。

【实例】

删除邮件

```
$ mailx[Return].....读取邮件
mailx version 4.0 Type ? for help.
"/usr/student/mail/david": 3 messages 3 new
> N1 daniel Thu ESP 30 12:26 6/103 Room
  N2 susan Thu Sep 30 12:30 6/107 Project
  N3 marie Thu Sep 30 12:30 6/70 Welcome
? d[Return].....删除当前消息
? d 3[Return].....删除消息 3
? h.....仅显示消息头行
> N2 susan Thu Sep 30 12:30 6/107 Project
```

h命令显示邮箱中消息的头行。用户已经删除了消息1和3,那么就只显示了剩下的消息

2 的头行。

```
? u 1[Return].....恢复删除的消息 1
? u 3[Return].....恢复删除的消息 3
```

【说明】

u 命令恢复被删除的指定消息,此处指消息 1 和 3。

```
? h[Return].....检查所有的消息是否在邮箱中
1 daniel      Thu ESP 30 12:26 6/103 Room
2 susan       Thu Sep 30 12:30 6/107 Project
3 marie       Thu Sep 30 12:30 6/70 Welcome
? d 1-3[Return].....删除消息 1、2、3
? h[Return].....检查所有的消息是否被删除了
No applicable messages
? u *[Return].....恢复所有被删除的消息
```

u 命令加上通配符 * 恢复所有被删除的消息。

```
? d /vacation[Return].....删除主题中有 vacation 的消息
No applicable messages
```

删除命令带上/string,指删除那些主题中含有字符串 string 的消息。此处,邮箱中消息的主题中没有含 vacation,因此没有消息被删除。

```
? d daniel[Return].....删除所有来自 daniel 的消息
```

删除命令带上指定的发送者的 ID,就是删除所有来自指定用户的消息。

```
? x [Return].....退出 mailx
$_.....回到 shell 提示符
```

从上可以知道,有多种删除消息的方法。然而,如果使用 **x** 退出 **mailx**,那么所有的消息(包括那些已被用户删除的消息)都保存在邮箱里。如果使用 **q** 命令退出 **mailx**,那么邮箱将根据用户已经使用的命令永久更新。

【注意】

如果使用 **q** 命令退出 **mailx**,那么用户将再也不可能恢复被删除的邮件了。因此在退出之前,要确保用户已经删除的邮件确实是用户想删除的。

9.4.3 保存邮件

mailx 命令使得用户在阅读消息时,可以将消息保存到指定的文件中。用户可以保存所有消息,也可以只保存一条消息,或者可以保存指定范围的消息。下面看一些例子。场景和前面一样:登录名 david,系统邮箱中有 3 条消息。

```
$ mailx[Return].....读取邮件
mailx version 4.0 Type ? for help.
"/usr/student/mail/david": 3 messages 3 new
> N1 daniel      Thu ESP 30 12:26 6/103 Room
   N2 susan       Thu Sep 30 12:30 6/107 Project
```

```

N 3 marie                Thu Sep 30 12:30 6/70 Welcome
? s mfile[Return]..... 把当前消息追加到文件 mfile
"mfile" [New file] 12/86..... 反馈消息

```

s 命令将指定的消息保存到指定的文件中。此处当前消息 1 (> 标志) 将保存到文件 mfile 中。

```

? s 2 3 mfile[Return]..... 把消息 2,3 追加到文件 mfile
"mfile" [Appended] 18/286..... 反馈消息
? s 1-3 mfile[Return]..... 把消息 1,2,3 追加到文件 mfile
"mfile" [Appended] 18/286..... 反馈消息
? x[Return]..... 退出 mailx, 邮箱保持不变
$_..... 回到 shell 提示符

```

此处,所有的消息都追加到文件 mfile 中,用户可以使用 mailx 命令带上 -f 选项阅读 mfile 的内容。

【说明】

1. 不要把 mailx 命令带上 -f 选项和 mailx 命令带 f 选项混淆。前者是加上一个文件名,阅读指定文件的内容;后者是仅显示当前消息的头行。

2. 不要把 mailx 命令带上 -s 选项和 mailx 命令带 s 选项混淆。前者是把主题设置为指定的字符串;后者是把消息保存到指定的文件中。

9.4.4 回复邮件

用户可以在阅读消息时给消息的发送者回复。这给用户带来了很大的便利,因为在读完消息后,马上就可以答复消息。

【实例】

在 mailx 的读取模式下,发送一个回复。操作如下:

```

$ mailx[Return]..... 读取邮件
mailx version 4.0 Type ? for help.
"/usr/student/mail/david": 3 messages 3 new
> N 1 daniel                Thu ESP 30 12:26 6/103 Room
  N 2 susan                 Thu Sep 30 12:30 6/107 Project
  N 3 marie                 Thu Sep 30 12:30 6/70 Welcome
? R[Return]..... 回复当前消息
Subject: RE: Room..... 主题是关于 Room
_..... 光标处在当前行的开头,等待用户的输入
Thank you..... 编辑回复的消息
[Ctrl-d]..... 结束消息
EOT..... 消息发送结束
?..... 等待下一个命令
? R 3[Return]..... 回复消息 3
? r 3[Return]..... 回复消息 3 和所有其他接收到 3 的用户

```

回复命令 R 仅仅把消息回复给消息的发送者或者指定的用户。而回复命令 r 不仅把消息回复给消息的发送者,而且还回复给所有接收到这条消息的用户。

用户还可以使用 m 命令给其他用户发送邮件。m 命令把 mailx 置为输入模式,这样用户

可以编辑邮件。使用 `m` 命令,用户可以指定一个用户或者一系列用户接收邮件。

【实例】

在 `mailx` 命令模式,给指定的用户发送邮件的步骤如下:

```
? m daniel[Return].....给 daniel 发送邮件
? m daniel susan[Return].....给 daniel 和 susan 发送邮件
? x[Return].....退出 mailx
$_.....出现 shell 提示符
```

9.5 定制 mailx 环境

用户可以通过设定 `.mailrc` 文件中的 `mailx` 变量来设置 `mailx` 的环境。为了定义这些变量,使用 `mailx set` 命令。

9.5.1 mailx 使用的 shell 变量

有一些 shell 变量同时也被 `mailx` 使用着,这些变量值影响 `mailx` 的行为。如 `HOME`(定义用户的 home 目录)和 `MAILCHECK`(定义 `mailx` 检查邮件的频率)。举例来说,如果用户想每一分钟就检查一次邮箱中的邮件到达情况,可以如此定义 `MAILCHECK`:

```
MAILCHECK = 60
```

`MAILRC` 是 `mailx` 使用的另外一个变量,这个变量定义每次 `mailx` 启动时检查的文件。如果这个变量没有定义,那么就使用默认值 `$HOME/.mailrc`。用户可以在 `.profile` 文件(登录 shell 的初始化文件)中这样定义 `MAILRC`:

```
MAILRC = $HOME/E-mail/.mailrc
```

【注意】

`HOME`、`MAILCHECK`、`MAILRC` 等 shell 变量被 `mailx` 使用,但是在 `mailx` 中不能改变它们的值。

有许多 `mailx` 变量可以被用户使用,根据用户的爱好剪裁 `mailx` 的环境。用户既可以在 `mailx` 中设置它们,也可以在 `.mailrc` 中设置。`set` 命令设置 `mailx` 变量,`unset` 命令保留原来的设置。`set` 命令的格式,创建、设置 `.mailrc` 文件的方法和 `.exrc`(vi 编辑器的初始化文件)相似。

【说明】

这些变量既可以在初始化文件 `.mailrc` 中的设置,也可以在 `mailx` 中设置。

`append`:在结束阅读邮件时,如果设置了 `append`,`mailx` 就会将消息追加到 `mbox` 文件的最后,而不是开头。

【实例】

设置 `append` 有两种选择:

- 键入 `set append`,按回车键,把消息追加到 `mbox` 文件的尾部。
- 键入 `unset append`,按回车键,把消息追加到 `mbox` 文件的开头,这是默认的设置。

asksub: 如果设置了 **asksub**, **mailx** 会提示用户输入主题, 这是默认设置。

crt 和 **PAGER**: **crt** 变量设置了屏幕的行数, 如果消息的行数超过 **crt** 设置的行数, 那么消息就由 **PAGER** 定义的变量发给管道命令。

【实例】

设置 **mailx**, 使得消息的长度如果超过 15 行, 就传送给 **pg** 命令, 一次显示一页。步骤如下:

- 键入 **set PAGER = 'pg'**, 按回车键(这是 **PAGER** 变量的默认设置, 除非系统管理员改了它)。
- 键入 **set crt = 15**, 按回车键。设置屏幕的行数为 15。

用户可以使用 **pg** 命令的选项上下浏览消息(见第 7 章)。

DEAD: 有部分编辑好的消息, 被中断(因为一些这样或者那样的原因没有发送), 保存到指定的文件中。默认的设置如下:

```
set DEAD = $HOME/dead.letter
```

【实例】

为了改变 **DEAD** 变量的默认设置, 键入 **set DEAD = \$HOME/E-mail/dead.mail**, 按回车键。

EDITOR: **EDITOR** 变量设置用户使用编辑命令(或者 **~e**)时编辑器的类型。默认设置如下:

```
set EDITOR = ed
```

【实例】

为了改变 **EDITOR** 的默认值, 键入 **set EDITOR = ex**, 按回车键。随后改变成另一个编辑器。

escape: **escape** 变量改变 **mailx** 的转义字符。默认设置为 **~**。

【实例】

为了改变 **escape** 变量值为 **@**, 键入 **set escape = @**, 按回车键。

folder: **folder** 变量指定一个目录为 **mailx** 的标准目录。所有的邮件文件都保存在 **folder** 变量指定的目录下。**folder** 变量没有默认设置的值。

【实例】

指定 **EMAIL** 文件保存到 **home** 目录下, 键入 **set folder = \$HOME/EMAIL**, 按回车键。

【说明】

这个命令不创建 **EMAIL** 目录。它仅仅给 **folder** 变量分配指定的目录。用户可以使用 **mkdir** 命令创建目录。

header: 如果 **header** 变量设为默认设置, 用户阅读消息时 **mailx** 显示消息的头。

【实例】

如果不设置 header 变量,键入 `unset header`,按回车键。

【说明】

此命令包含在一对后引号内。

MBOX: MBOX 指明把已经读过的消息自动保存到哪个文件中。默认的文件为 `$ HOME/mbox`。

【实例】

为了改变 MBOX 变量的默认值,键入 `MBOX = $ HOME/EMAIL/mbox`,按回车键。

【说明】

把消息保存到一个文件中或者使用 `x` 命令退出 `mailx`,就跳过了自动保存的过程。

PAGER: PAGER 变量设置为页命令,和 `ert` 变量联合使用。默认的页命令是 `pg`。

【实例】

为了把页命令改为 `more`,键入 `PAGER = `more``,按回车键。

【说明】

命令用一对后引号括起来。

record: record 变量指明文件名,该文件保存所有发送出去的消息。record 变量没有默认值。

【实例】

为了把所有发送出去的消息保存到文件 `keep` 中,键入 `record = $ HOME/EMAIL/keep`,按回车键。

【说明】

这个命令不创建 EMAIL 目录。它仅仅为 record 变量分配指定的目录。用户可以使用 `mkdir` 命令创建目录。

SHELL: SHELL 变量设置用户想使用的 shell 程序。这适用于当用户在 `mailx` 环境下,使用 `!` 或者 `~!` 命令调用 shell 命令。默认值为 `sh`。

【实例】

为了改变 SHELL 的默认值,键入 `set SHELL = csh`,按回车键。现在 shell 就变为 C shell。

VISUAL: VISUAL 变量设置 `mailx` 处于输入模式时,用户使用 `~v` 命令调用的全屏编辑器。默认值为 `vi` 编辑器。

9.5.2 设置 .mailrc 文件

用户 home 目录下的 `.mailrc` 初始化文件包含用户根据自己的喜好设置或者裁减 `mailx` 的变量和命令。用户可以使用 `vi` 或者 `cat` 命令创建 `.mailrc` 文件。

图 9.7 显示 `.mailrc` 的样本文件。在这个文件中,首先, `friends` 和 `chess` 分别被指定为两组用户 ID。然后,文件中还设置如果消息的长度超过 20 行,就使用 `pg` 命令(默认的 PAGER)显

示。最后,用户的私人信箱设置在 EMAIL 目录下,发送出去的邮件保存在 EMAIL 目录下的 record 文件中。

```
$ cat .mailrc
alias friends daniel david marie gabe emma stev
alias chess emma susan gabe
set crt = 20
set MBOX = $ HOME/EMAIL/mbx
set record = $ HOME/EMAIL/record
$ _
```

图 9.7 .mailrc 样本文件

9.6 与系统外的用户通信

本章已经讨论了如何与同一系统中的用户进行通信。用户也可以给其他 UNIX 系统的用户发送邮件。如果用户处于 UNIX 网络中,如大公司或者大学之间,用户可以使用同样的命令。然而,用户在发送邮件时可能需要给出更多的信息。例如,消息的目的地除了包含用户的 ID,还需要加上计算机(网络中的节点名)的名字。如果在网络中用户和 david 之间的计算机节点名是 X、Y 和 Z,那么用户给 david 发送消息时需要键入以下内容:

```
mailx X\!Y\!Z\!david
```

与其他 UNIX 系统中的用户通信细节超出了本书的范围。本章介绍的通信内容都是用户需要掌握的基本技巧,其他问题就需要在参考书中查找命令的使用。附录 F 列出一些对查找命令可能有用的书。

命令小结

本章主要集中讨论了 UNIX 通信工具,具体讨论了如下命令和选项的使用。

msg

如果用户不想接收 write 消息,该命令设置为 n。如果接收 write 消息,设置为 y。

mailx

这个程序给用户提供了电子邮件功能。用户可以给系统中的用户发送邮件,而不考虑该用户是否登录。

选项	功能
-f [文件名]	从指定的文件中而不是系统邮箱中读取消息。如果没有指定文件名,就从 mbox 中读取
-H	显示消息头列表
-s 主题	设置邮件的主题

news

这个命令用来查看系统中最新的新闻。可以被系统管理员用来通知用户发生的事情。

选项	功 能
-a	显示所有的新闻项,无论是新的还是旧的
-n	仅仅列出新闻名
-s	显示当前新闻的总数

wall

这个命令一般被系统管理员用来警告大家一些紧急的事件。

write

这个命令用来进行端到端的通信。接收方必须已经登录到系统。

mailx ~ 转义命令

启动 mailx 给其他用户发送邮件时,mailx 处于输入模式,等待用户编辑消息。在这种模式下,使用命令要加上~,称为~转义命令。

命令	功 能
-?	显示所有转义符命令列表
~! 命令	在编辑邮件时,可以调用指定的 shell 命令
~e	调用编辑器,编辑器的名字可以由变量 EDITOR 定义
~q	显示输入模式,把编辑了部分的消息保存到 dead.letter 文件中
~r 文件名	读取指定的文件,把它的内容插入到消息中
~< 文件名	读取指定的文件(使用重定向符),把它的内容插入到消息中
~<! 命令	执行指定的命令,把它的输出插入到消息中
~v	调用默认编辑器 vi,或者使用邮件变量 VISUAL,指定为其他的编辑器
~w 文件名	把正在编辑的消息写到指定的文件中

talk

这个命令用来进行端到端的通信。接收方必须已经登录到系统。

mailx 命令模式

启动 mailx 读取邮件时,mailx 处于命令模式,命令模式下的提示符是?。

命 令	功 能
!	执行 shell 命令(shell 转义)
cd [目录]	进入指定的目录,如果没有指定目录,就进入 home 目录
d	删除指定的消息
f	显示消息的头行
q	退出 mailx,把消息从系统邮箱中移去

(续表)

命 令	功 能
h	显示当前消息的头
m 用户	发送邮件给指定的用户
R 消息	给消息的发送者回复
r 消息	给消息的发送者和同一消息的其他所有接收者回复
s 文件名	保存消息到指定的文件中
t 消息	显示指定的消息
u 消息	恢复被删除的消息
x	退出 <code>mailx</code> , 不把消息从系统邮箱中移去

习题

1. 哪些命令用来进行端到端的通信? 什么键表示通信的结束?
2. 什么命令一般被系统管理员用来通知用户日常事务?
3. 什么命令作广播消息用?
4. 怎么设置终端屏蔽不想要的消息?
5. 读取邮件的命令有哪些?
6. 用户怎么知道自己有邮件?
7. `.mailrc` 文件有什么功能?
8. `mailx` 处于命令模式下, 哪些命令用来:
 - a. 显示消息列表?
 - b. 从指定的文件读取消息?
 - c. 改变主题的内容?
 - d. 显示 ~ 转义命令的列表?
 - e. 从指定的文件读取, 并插入到消息中?
 - f. 启动默认的编辑器?
9. 下列 shell 变量的作用是什么?
 - a. DEAD
 - b. RECORD
 - c. PAGER
 - d. MBOX
 - e. LISTER
 - f. header
10. 哪些命令能用来与另外一个登录的用户通信?

上机练习

这次的上机练习主要训练给其他用户发送消息。先给自己发送消息, 锻炼如何使用命令。

然后,当用户自己感到使用命令已经很顺畅的时候,可以找一个合作伙伴,练习发送邮件。

下面的练习推荐给大家。为了掌握 UNIX 通信工具的命令,希望大家多花点时间练习,试验各种命令的组合作用。

1. 在 home 目录下创建 EMAIL 目录。
2. 在 home 目录下创建 .mailrc 文件。如果该文件已经存在,修改它。
3. 如下设置 **mailx** 的初始化文件:
 - a. 使用 **alias** 命令给一群用户分配一个名称。
 - b. 在 EMAIL 目录下生成一个文件,自动保存用户发送出去的消息。
 - c. 在 EMAIL 目录下,设置用户的 mbox。
4. 把下面的消息发送给自己。

```
So little time and so much to do.  
Could it be reversed?  
So much time and so little to do.
```

5. 当 **mailx** 处于输入模式时,使用 **vi** 编辑消息。
6. 读取当前的日期和时间,追加到消息的结尾。
7. 在发送消息之前,把消息保存到一个文件中。
8. 用 **vi** 和其他命令以同样的方式编辑多条消息发送给自己。这样在练习读取邮件时,就有足够多的消息。
9. 读取自己的邮件。
10. 使用 **mailx** 命令模式下的所有命令,包括删除、恢复、保存等等。
11. 阅读邮件,使用 **x** 命令退出 **mailx**。然后再阅读邮件,使用 **q** 命令退出 **mailx**。观察 UNIX 的提示消息。
12. 使用 **mailx**,看看 mbox 文件。
13. 现在找一位合作伙伴,使用 **write** 命令进行交谈。
14. 把 **mesg** 设置为 **n**,然后再设置为 **y**,和你的伙伴一起观察效果。
15. 练习给其他用户发送邮件,也许在学习 UNIX 的过程中不是一件太令人厌烦的事情。多加练习,其乐无穷。

第 10 章 程序开发

本章描述了程序开发的要点。首先解释创建一个程序的步骤,对常用的计算机编程语言进行概要描述。然后给出一个简单的 C 程序示例,带着读者一步步从编写源代码到建立可执行程序。本章还解释了利用 shell 重定向操作符来重定向程序输出和错误信息。

10.1 程序开发

第 1 章简要讨论了软件和计算机编程。读者已经知道软件的重要作用,它能够让计算机做许多神奇的事情。读者也知道软件分成两类:应用软件和系统软件。

程序由一组指令组成,指示计算机进行基本数学和逻辑操作。每条指令告诉机器实现一个基本功能,通常由操作代码和一个或多个操作数组成。操作代码指定要实现的功能,操作数指定所操作的位置或数据元素。图 10.1 显示了一条典型的指令。

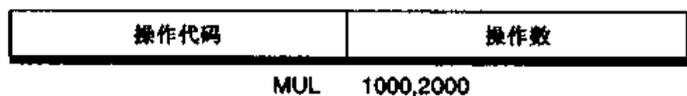


图 10.1 简单指令的格式

计算机由内存中的程序所控制。内存只能保存 0 和 1 串,所以内存中的程序必须是二进制形式。这意味着程序必须写成 01 串的形式或者转化成 01 串。可是程序员怎么写 01 串呢?幸好,程序员们不再需要这样做了。早期的程序员没有选择,他们不得不直接编写 01 串的二进制程序,所以很值得尊敬。程序员编写程序,生成前述两类软件。为了编写程序,需要一种编程语言,也确实有不少语言可以选择。

10.2 编程语言

编程是用计算机解决问题而编写指令(程序)的过程,这些指令必须用计算机编程语言来写。一个程序可以从最简单的把几个数字相加的指令列表到具有庞大和复杂结构的大公司工资单计算。

像计算机硬件一样,编程语言进化了好几代。每种新一代的语言改进了以前的语言,并为程序员集成了更多功能。

图 10.2 显示出编程语言的层次和换代。下面来看看不同级别的语言。

10.2.1 低级语言

机器语言:机器语言中,指令按照 01 序列编码,很麻烦而且难于编写程序。机器语言程序写于计算机操作中的最底层,是计算机惟一能够理解和执行的语言。用其他编程语言编写的

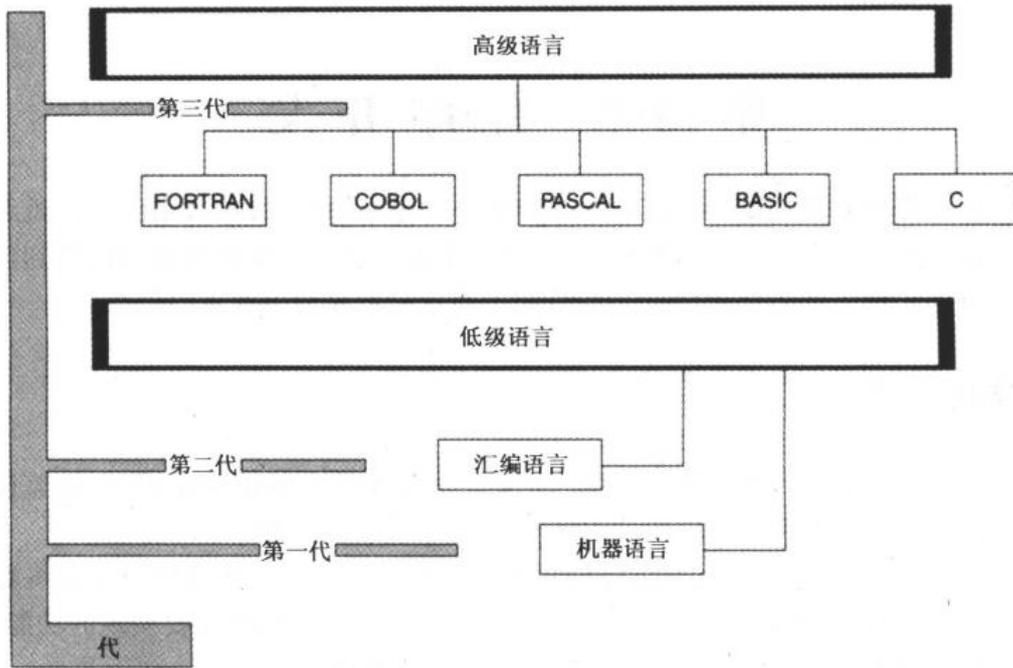


图 10.2 编程语言层次图

程序必须翻译成机器语言,计算机才能执行(称作编译器的程序完成翻译,下一节进一步讨论)。

汇编语言:和机器语言一样,汇编语言对一台计算机而言是惟一的,但是指令有所不同。不是 01 序列,汇编语言使用助记符号(称为助记符)来表示指令。例如助记符 MUL 用来表示乘法指令。因为计算机只理解 0 和 1,必须把汇编语言的程序改成机器语言格式才能执行。这个翻译过程通过调用汇编器来完成。汇编器把程序中的助记符翻译回 0 和 1 串。

10.2.2 高级语言

无论使用什么计算机,选择何种编程语言,程序必须翻译成机器语言才能执行。从高级编程语言到低层机器语言的转化是软件程序编译器和解释器的工作。高级编程语言是为了程序员编写方便;它们的原始形式(源代码)不能被执行。这一小节描述几种主要的编程语言。

COBOL:COBOL(通用面向商务语言)编程语言产生于 1959 年。对应于商业团体的需求而开发,主要用在大型机上,进行大公司的数据处理服务。尽管市场上有许多效率更高更快的通用计算机语言,但是由于超过一半的商业应用程序用 COBOL 写成,因此 COBOL 仍然在使用着。

FORTRAN:FORTRAN(公式翻译器)编程语言 1955 年开发,主要适合于科学或工程方面的编程,仍然是最广泛的科学编程语言。当然,FORTRAN 为了适用于新型计算机也进行了更改。当前的 FORTRAN 版本是 FORTRAN 77,是一种通用语言,可以处理数字和符号问题。和 COBOL 一样,大量已有的代码是用 FORTRAN 编写的,因而它仍然用在工程和科学上。

Pascal:Pascal 编程语言开发于 1968 年,以 17 世纪法国数学家 Blais Pascal 的名字命名。出于培养良好风格和编程习惯的想法,以及结构化编程也无意中形成了,Pascal 集成了结构化编程的思想。Pascal 作为一种帮助学生学习结构化编程和培养良好编程风格的语言而设计,但

其使用不限于教学方面,它也用在工业中创建易读易维护的代码。

BASIC: BASIC(初学者通用符号指令代码)编程语言开发于 1964 年,用于帮助没有计算机背景的学生学习计算机和编程。它被认为是用于教学目的的最有效的编程语言。后来, BASIC 进化成通用语言,使用不限于教育领域。BASIC 被广泛使用于商业应用。

C: C 编程语言 1972 年开发,基于 Pascal 编程的实际规则。主要用于系统编程,开发操作系统、编译器等等。多数 UNIX 操作系统用 C 写成。C 是快速、高效的通用语言。它也是一种可移植的语言,与机器无关。一种计算机上编写的 C 程序在另一种计算机上不加修改或少许修改就可以运行。C 是目前许多程序员开发商用、科学和其他应用的首选。

C++: 在 20 世纪 80 年代,C 语言中增加了必要的部分形成了 C++,它成为一种面向对象语言。面向对象编程(OOP)是编程中相对新的技术。C++ 减少了程序开发时间。

10.3 编程机制

为了编写程序,应该选择一种编程语言。编程语言的选择按照应用程序的需要而定。现在有许多通用目的和特殊目的的编程语言,可以适合各种要求。用户必须按照下面的具体步骤生成程序。

10.3.1 建立可执行程序步骤

与操作系统和编程语言无关,建立可执行程序都需要下面的步骤:

1. 建立源文件(源代码)。
2. 建立目标文件(目标代码/目标模块)。
3. 建立可执行文件(可执行代码/载入模块)

源代码: 用户通常用编辑器(例如 vi 编辑器)编写程序,然后把写好的程序保存在文件中。这个文件称作源代码。源代码是由用户选择的编程语言写成的,计算机不能直接理解。源代码文件最终要转换成可执行文件。

【说明】

源文件是文本文件,也称为 ASCII 文件。可以在屏幕上显示、用编辑器修改或者发送到打印机以获得源代码的硬拷贝。

目标代码: 源代码对计算机而言不可理解。计算机只能理解机器语言(0 和 1 格式)。因此,源代码要翻译成机器能理解的语言,这就是编译器和解释器的工作。它们生成目标代码。目标代码是源代码的机器语言翻译。但是,它还不是可执行文件;它缺乏一些必要的部分。这些必要部分是为用户的程序和操作系统之间提供接口的程序。它们一起放在库文件中。

【注意】

不能把目标文件发送给打印机。它是 0 和 1 组成的文件。如果非要显示,用户的键盘可能会锁住或者听到响铃,因为一些 0 和 1 串被翻译成的 ASCII 代码对应着键盘上锁、响铃、停止滚屏等等。

可执行代码: 目标代码可能依赖于并非目标模块的其他程序。在程序被执行之前,对其他

程序的依赖关系需要进行解析。这是链接器或链接编辑器的工作。它创建可执行代码,也就是载入模块。载入模块是完整的可以执行的程序。

【注意】

和目标文件一样,载入模块不能被送到打印机或者终端显示。

一些系统中,用户先调用编译器。当编译过程完成之后,用户再调用链接器建立可执行文件。另一些系统中,用户只需要给出编译命令,如果程序没有任何编译错误,链接器会被调用来链接程序。

【说明】

具体怎样工作要随着系统而变化,它依赖于管理员所做的系统环境配置。

图 10.3 描绘了从源文件开始以建立可执行文件结束的过程。

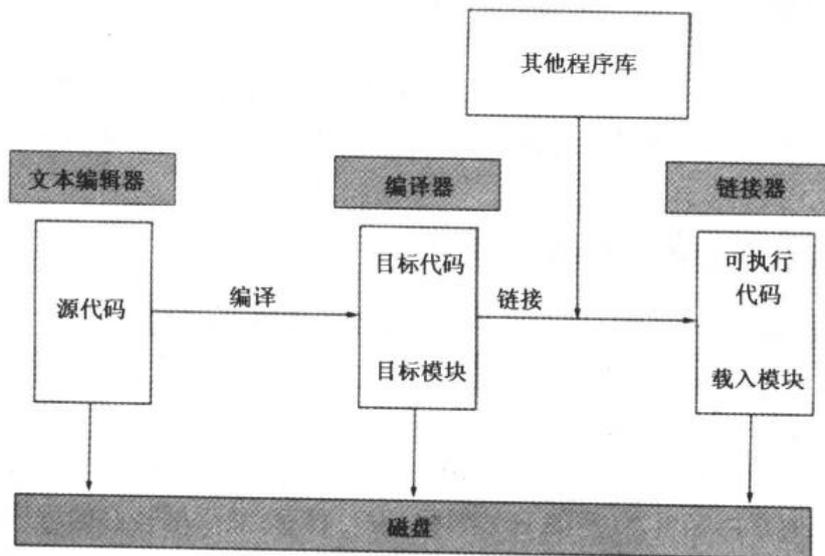


图 10.3 程序开发步骤

10.3.2 编译器/解释器

编译器或解释器的主要功能是把源代码(程序指令)翻译成机器代码,以便计算机能够理解用户的指令。编译语言和解释语言代表计算机语言的两个类别,每一种都有优点和缺点。

编译器:编译器是系统软件程序,它把高级程序指令(如 Pascal 程序)翻译成计算机能够解释和执行的机器语言。它一次编译所有的程序代码,在编译完成之前没有任何反馈。

每种编程语言都需要单独的编译器。为了执行 C 和 Pascal 程序,必须要有一个 C 和一个 Pascal 编译器。与解释器相比,编译器产生更好且效率更高的目标代码,所以编译的程序运行起来更快,需要更少的空间。

解释器:和编译器一样,解释器把高级语言翻译成机器语言。但是,不是翻译整个源程序,它一次翻译一行代码。解释器立刻给用户反馈。如果代码包含错误,只要用户按回车键结束一行代码,解释器就立刻能找出来。因此,可以在程序开发过程中改正错误。解释器并不产生单独的目标代码文件,每次程序执行都需要进行一次解释过程。解释器通常用在教育环境中,

生成的可执行代码比编译器产生的效率低。

10.4 简单的 C 程序

我们来编写一个简单的 C 程序,按步骤看看如何完成这个过程。目的是理解和练习编译过程,而不是学习 C 语言。但是,为了能够成功编写和编译 C 程序,读者需要理解一些 C 语言最基本的特征。

用 vi 编辑器(或者其他编辑器),创建一个名为 first.c 的源文件。图 10.4 显示了文件内容。

```
$ cat first.c
/* my first C program */
# include <stdio.h>
main()
|
|   printf ("Hi there! \n");
|   printf ("This is my first C program \n");
|
|_ $
```

图 10.4 简单的 C 程序

【注意】

1. 用小写字母输入源代码。和 UNIX 一样,C 语言是小写字母语言,所有关键字必须用小写字母。
2. 在大多数系统中,源代码文件必须用 .c 扩展名。

下一步是编译源代码。命令如下:

```
$ cc first.c [Return]
```

cc 命令编译源代码,如果没有任何错误,会自动调用链接器。最后的结果是生成名为 a.out 的可执行文件。默认情况下,a.out 是可执行文件的名称。现在如果想要执行文件,查看程序输出,输入:

```
$ a.out [Return].....执行程序
Hi there!
This is my first C program.
$_. ..... 出现提示符
```

【说明】

a.out 的输出显示在标准输出设备终端上。

如果不想把可执行文件命名为 a.out 该怎么办呢? 可以用带 -o 参数的 cc 命令来设置输出文件名。

【实例】

编译 first.c,指定输出文件名为 first。

```
$ cc first.c -o first [Return].....用带-o 参数的 cc 命令
$_..... 没有错误,回到提示符下
```

【注意】

确保指定的输出文件名和源代码文件名不同;否则,源代码文件会被覆盖,变成可执行文件——这意味着源文件被破坏了。

现在可执行文件称作 `first`,在提示符下输入 `first`,可以看到程序输出。

```
$ first [Return].....执行 first
Hi there!
This is my first C program.
$_.....等待其他命令
```

如果并不想在屏幕上输出怎么办?或许用户希望把程序输出保存到文件中。利用 shell 的重定向功能,可以把输出重定向到别的文件。

【实例】

运行 `first`,把输出保存到文件。

```
$ first > first.out [Return].....输出重定向到 first.out,所以没有显示在屏幕上
$ cat first.out [Return].....查看 first.out 的内容
Hi there!
This is my first C program.
$_.....出现提示符
```

【说明】

`first.out` 的内容是 C 程序的输出。重定向程序输出时创建文件 `first.out`。

【实例】

执行 `first`,输出显示在屏幕上,同时保存在文件里。

```
$ first | tee -a first.out [Return].....输出显示在屏幕上,同时保存在 first.out 中
Hi there!
This is my first C program.
$_.....出现 shell 提示符
```

用带管道操作符的 `tee`(复制输出)命令,程序输出显示在屏幕上,同时也保存到 `first.out` 里。`tee` 命令的 `-a` 参数把输出附加到 `first.out` 的文件末尾。

10.4.1 改正错误

如果没有任何错误就完成了第一个 C 程序,事情很简单。用户直接编译和执行程序。但是,当编写大程序的时候,不可能总是如此。在一个程序能成功运行之前,用户必须改正程序中的语法或逻辑错误。C 编译器能识别语法错误,并用行号索引把错误显示在屏幕上。

假设像图 10.5 那样更改的 `first.c` 文件,在第一个 `first printf()` 语句后,丢了分号(;).现在,编译 `first.c` 时,编译器就会报错。

```

$ cat first.c
/* my first C program */
# include <stdio.h>c
main()
{
    printf("Hi there! \n") /* semi colon omitted intentionally. */
    printf("this is my first C program \n");
}
$ _

```

图 10.5 有语法错误的 C 程序

【实例】

编译 first.c。

```

$ cc first.c -o first [Return].....编译,产生输出文件 first
"first.c", line 6: syntax error.
"first.c", line 6: illegal character: 134 (octal)
"first.c", line 6: cannot recover from earlier error; goodbye!
$ _.....提示符

```

错误消息表示在第 6 行有某种语法错误。编译器不能准确识别错误和位置,它只能指出源代码错误的大概位置。

这种情况下,编译器不会产生目标代码文件。为了改正错误,用户必须回到源代码 first.c,添加分号。然后,要想得到正确的可执行文件,应该重新编译文件。

屏幕上查看一两行错误很容易,但是如果代码很多,可能很多行都有错。想要都记住这些错误和行号从而修改源文件就不那么容易了。因此,为了方便查看,可以把编译错误信息保存到文件中。shell 的重定向功能用起来很方便。

默认错误设备通常与输出设备(终端)相同,所以,错误显示在终端上。假设用户想要把错误重定向到指定的文件。

【实例】

编译 first.c,如果有编译错误的话,把它保存到另一个文件里。

```

$ cc first.c -o first 2> error [Return].....重编译
$ _.....出现 shell 提示符

```

【注意】

“>”符号前的数字 2 是必要的,表明标准错误设备的重定向。

数字 2 来自哪里? 这个命令还需要进一步解释。下面回到 UNIX 重定向概念,找出数字 2 的来源。

10.4.2 重定向标准错误

shell 把“>”解释成标准输出重定向。符号 1> 与 > 一样,告诉 shell 重定向标准输出。“1>”里的数字 1 是文件描述符;默认情况下,文件描述符 1 分配给了标准输出设备。例如,下面的两个命令做同样的工作:把 ls 命令的输出重定向到一个文件。

```
$ ls -C > list [Return]
或者
$ ls -C 1 > list [Return]
```

文件描述符 2 分配给标准错误输出。shell 把符号 2> 解释成重定向到错误输出。例如, 假设当前目录没有文件 Y, 执行如下命令:

```
$ cat Y > Z [Return].....将 Y 拷贝到 Z
cat: cannot open Y.....错误信息
```

错误信息出现在屏幕上。现在把错误输出重定向到文件。

```
$ cat Y > Z 2> error [Return].....把 Y 复制到 Z。把错误保存到当前目录的 error 文件中
$.....没有错误信息, 返回提示符
$ cat error [Return].....显示文件 error 的内容
cat: cannot open Y.....正如读者所期待的
$.....出现提示符
```

现在回到 C 程序和它的编译错误; 把编译错误重定向到另一个文件:

```
$ cc first.c -o first 2> error [Return].....错误重定向到文件 error
$.....不显示错误
$ cat error [Return].....查看编译错误
"first.c", line 6 : Syntax error.
"first.c", line 6 : Cannot recover from earlier error : goodbye!
$.....提示符
```

10.5 UNIX 编程工具

在 UNIX 环境下也可以获得其他计算机语言的编译器。几乎每种语言都有运行在 UNIX 下的编译器。

本章的目的是介绍 UNIX 下的程序开发, 因而并不打算对语言、编译器和 UNIX 编程搞一个综合列表。但是, 用户应该知道 UNIX 提供了一些工具, 以帮助用户组织程序开发。开发大型软件的时候, 这些工具就变得很有用也很重要了。下面是对这些工具和功能的非常简单的介绍。

10.5.1 make 工具

如果程序由多个文件组成, **make** 工具将非常有用。**make** 自动记录进行了修改并需重新编译的源文件, 必要的话进行重新链接。**make** 程序从控制文件中得到信息。控制文件中包含着指定源文件依赖关系以及其他信息的规则。

10.5.2 SCCS 工具

SCCS(Source Code Control System, 源代码控制系统)是一组程序, 用来帮助用户维护和管理程序开发。程序在 **SCCS** 的控制下, 用户可以方便地生成程序的不同版本。**SCCS** 可以清楚记录不同版本之间的所有改变。

习题

1. 解释编写程序和产生可执行文件的必要步骤。
2. 什么是源代码?
3. 编译器的功能是什么?
4. 编译器和解释器的区别在哪里?
5. 编译 C 程序的命令是什么? 可执行文件的默认名是什么?
6. 为什么不能把可执行文件发送到打印机?

上机练习

本章的练习是编写一个 C 程序例子。这里并不要求用户懂得 C 语言。复制本章中的简单 C 程序例子,或者任何 C 语言编程书中的程序。练习的目的是让用户熟悉程序开发的过程。

1. 编写简单的 C 程序。
2. 编译。
3. 运行程序。
4. 重新编译,指定可执行文件。
5. 再次运行。
6. 把程序输出保存到另一个文件。
7. 更改源代码,故意产生语法错误。
8. 编译。
9. 观察错误信息,看看能否明白。
10. 重新编译程序,把错误信息保存到文件中。
11. 查看包含编译错误的文件。

第 11 章 shell 编程

本章讨论 shell 编程。在本章中将阐述 shell 作为高级解释性语言的能力,说明 shell 编程的构造和细节,探索 shell 编程中变量和流控命令等方面的问题。本章将演示创建、调试和运行 shell 程序,并继续介绍一些 shell 命令。

11.1 理解 UNIX shell 编程语言:介绍

命令语言提供了一种通过组合一系列命令来编写程序的方法,而这些命令与用户在命令提示符状态下键入的命令完全相同。如今,绝大多数命令语言的能力远远不止于一系列命令的顺序执行。这些命令语言拥有高级语言的许多特性,如循环结构、决策声明等。这为用户在编程时提供了选择:使用高级语言或者使用命令语言。命令语言与编译语言(如 C 或 FORTRAN 等)不同,它是一种解释语言。命令语言程序比编译语言程序易于调试和修改。但是,用户必须为这样的便利付出一定代价:命令语言程序的执行时间比编译语言程序要长。

UNIX shell 拥有自己的内部编程语言,并且所有的 shell(无论 Bourne shell 还是 C shell 等)都提供这种编程能力。shell 语言是一种命令语言,拥有许多计算机程序语言的一般特性,包括结构化语言构成:顺序、选择和重复。使用 shell 编程使程序易于编写、修改和调试,并且不需编译。用户可以写完程序后立即运行。

shell 的程序文件称为 shell 过程、shell 脚本或只是简单地称为脚本。shell 脚本是包含将由 shell 运行的一系列命令的文件。运行一个 shell 脚本时,文件中的每条命令被送往 shell 执行,一次运行一个命令。当所有命令执行完毕或出现错误时,脚本运行结束。

用户并不一定必须写 shell 程序。但是,在使用 UNIX 时,用户会发现有时希望 UNIX 完成的功能并没有相应的命令,或者希望 UNIX 同时完成几个命令。UNIX 有许多命令,难于记忆并且需要键入许多字符。如果用户不想记忆所有那些 UNIX 奇怪的语法,或者不擅长于键盘输入,那么写脚本文件可能是一个很好的选择。

【说明】

cat 命令可以用来显示大多数脚本程序。用户可以用 vi 来创建这些文件,当然也可以用 cat 创建它们。记住,cat 命令非常适合于创建只有几行的小巧的程序。

11.1.1 写一个简单脚本

写一个简单的脚本并不要求你是一位程序员。例如,假设用户想知道当前有多少个用户登录到系统上。有关命令及输出的情况如下:

```
who | wc -l [Return].....键入的命令  
6 .....UNIX 显示的结果
```

who 命令的输出被送往 wc 命令的输入,-l 选项使用 wc 命令只计算输入文件的行数,而这

些行数正是代表了当前登录到系统的用户数。

读者可以写一个能完成同样工作的脚本程序。如图 11.1 所示,一个名为 `won` 的脚本程序,使用 `who` 和 `wc` 命令来得到当前系统用户数。`shell` 脚本以 UNIX 文本文件方式保存,因此,用户可以使用 `vi` 编辑器(或自己喜欢的编辑器)或 `cat` 命令创建它们。

```
$ cat won
#
# won
# display # of users currently logged in
#
who | wc -l
$_
```

图 11.1 一个简单的 shell 脚本

【说明】

在图 11.1 的例子中,一些行首的 `#` 号代表该行是注释。`shell` 会忽略以 `#` 开始的行。

11.1.2 执行脚本

有两种方式执行 `shell` 脚本:使用 `sh` 命令或将该脚本程序转变成可执行文件。

用 `sh` 调用脚本

用户可以使用 `sh` 命令执行脚本文件。每当键入 `sh` 时,即调用了另一个 `shell`。由于上例中的脚本文件 `won` 不是一个可执行文件,因此用户必须调用另一个 `shell` 来执行它。用户指定脚本文件名,新 `shell` 读取该文件,执行其中的命令,并在所有命令执行完毕时(或出错时)结束。

图 11.2 显示使用这种方法如何执行 `won` 脚本文件。每次执行脚本文件时键入 `sh` 来调用另一个 `shell` 并不是很方便。但是,这种方法有它的优点,特别是用户写了复杂的脚本程序并且需要调试和跟踪工具时(这些工具将在本章后面讨论)。通常情况下,推荐使用第二种方法,即将脚本文件转换成可执行文件再执行。毕竟,键入越少越好。

```
$ sh won
6
$_
```

图 11.2 使用 `sh` 命令执行一个脚本文件

将文件变成可执行文件:使用 `chmod` 改变文件权限

运行一个 `shell` 程序的第二种方法是将该文件变为可执行文件。在这种情况下,用户不需要调用另一个 `shell`,如同执行其他 `shell` 命令一样,只需键入要执行的脚本文件的名字。这是推荐的方法。

要使一个文件变成可执行文件,必须修改该文件的访问权限。使用 **chmod** 改变指定文件的模式(关于 **chmod** 命令的更多信息见第 13 章)。表 11.1 显示 **chmod** 选项。假设用户有一个名为 **myfile** 的文件,下面例子显示如何改变该文件的访问模式。

表 11.1 **chmod** 命令选项

字符	作用对象
u	用户/所有者
g	组
o	其他用户
a	所有用户;可以用来替换 ugo 的结合
字符	权限分类
r	读权限
w	写权限
x	执行权限
-	无权限
字符	权限分类操作
+	赋予权限
-	拒绝权限
=	为特定用户设置权限

【实例】

下列命令使 **myfile** 成为可执行文件。

```
$ ls -l myfile [Return].....查看 myfile 的模式
-rw-rw-r--1 david student 64 Oct 18 15:45 myfile
$ chmod u+x myfile [Return].....改变 myfile 的模式
$ ls -l myfile [Return].....再次查看 myfile 的模式
-rwxrw-r--1 david student 64 Oct 18 15:45 myfile
$_ .....命令提示符
```

ls 命令用来核实该文件的访问权限是否改变。**u** 指示文件的所有者, **+x** 指示该文件的访问模式变为可执行的。

【实例】

下面命令取消所有其他用户对 **myfile** 文件的写权限。

```
$ chmod o-w myfile [Return].....改变 myfile 的模式
$_ .....命令提示符
```

用户及所在组的所有成员还拥有对该文件的读写许可,但其他用户只能读取该文件(用户可以用 **ls** 命令来核实这一改变)。

字母 **o** 指示其他用户, **-w** 表示取消对该文件的写权限。

【实例】

下面命令改变文件的访问许可为所有用户(包括所有者、与所有者同组的成员和其他用

户)读和写。

```
$ chmod a = rw myfile [Return].....改变 myfile 的模式
$_.....命令提示符
```

现在,所有人可以读写 myfile(同样,可以用 **ls** 命令来验证这一改变)。字母 **a** 指示所有用户, **= rw** 指示读写访问权限。

【实例】

下面命令取消所有非文件所有者(同组成员或其他用户)对文件的访问权限。

```
$ ls -l myfile [Return].....查看 myfile 的模式
-rwx rw-r--1 david student 64 Oct 18 15:45 myfile
$ chmod go = myfile [Return].....改变 myfile 的模式。注意,在 = 与
                                     myfile 之间必须有空格
$ ls -l myfile [Return].....再次查看 myfile 的模式
-rwx -----1 david student 64 Oct 18 15:45 myfile
$_.....命令提示符
```

所有者是惟一有权访问文件 myfile 的用户。**ls** 命令验证了这一改变。**go** 指示所有者同组成员和其他用户,所以 **go =** 表明取消他们对该文件的所有访问许可。

【实例】

回到 won shell 脚本,把它变成可执行文件。

```
$ ls -l won [Return].....检查 won 文件的权限
-rw-rw-r--1 david student 64 Oct 18 15:45 won
$ chmod u + x won [Return].....改变模式
$ ls -l won [Return].....验证改变
-rwx rw-r--1 david student 64 Oct 18 15:45 won
$_.....命令提示符
```

现在,该脚本文件是一个可执行文件了。要执行它并不需要调用另一个 shell 程序,而像所有其他命令一样,只需要键入该文件的名字,然后按回车键。

```
$ won [Return].....执行 won
6.....运行结果
$_.....命令提示符
```

11.2 写更多的 shell 脚本

shell 编程相当简单,但在用户工作中是功能强大的工具。用户可以将任何命令按任何顺序放在一个文件中,将该文件变成可执行文件,然后在 **\$** 命令提示符下简单地键入其文件名来执行它。

【说明】

1. 绝大多数 UNIX 系统包含多个 shell。本书中的命令和脚本文件(程序)应当可以很好地工作在 sh 和 ksh 中。但是,系统安装会有很多微妙的区别。因此,注意使用 **man** 命令来了解如何使各种命令更好地工作于用户使用的系统中。

2. 用户可以通过键入 `sh`、`ksh` 或 `csk` 来分别调用 Bourne shell、Korn shell 或 C shell 三者之一，以改变当前工作的 shell，但这并不会改变用户的登录 shell。键入 `exit` 可以终止当前 shell，返回用户登录的 shell。

3. 标准变量 SHELL 用来设定用户的登录 shell。

4. 如果用户登录 shell 是 Korn shell，则用 `ksh` 命令代替 `sh` 执行脚本文件。

现在修改一下 `won` 文件，为其增加一些命令。图 11.3 显示修改后第二个版本的 `won` 文件内容。

```
$ cat won
#
# won second version
# displays the current date and time, # of users currently logged
# in, and your current working directory
#
date
who | wc -l
pwd
$_
```

图 11.3 一个简单的 shell 脚本程序

假定 `won` 文件的访问权限已经被修改，该文件已是一个可执行文件。图 11.4 显示第二个版本的 `won` 文件的运行结果。

```
$ won
Wed Nov 29 14:00:52 EDT 2001
  14
/usr/students/david
$_
```

图 11.4 第二版本 `won` 程序的输出

该 `won` 程序的第二个版本的输出结果含义模糊，当然可以更好些。可以使用一些 `echo` 命令来使这些输出内容更加有意义。图 11.5 为第三个版本的 `won` 脚本。

```
$ cat won
#
# won third version. The user friendly version
# displays the current date and time, # of users currently
# logged in and your current working directory.
#
echo
echo "Date and Time: \c"
date
echo "Number of users on the system: \c"
who | wc -l
echo "Your current directory: \c"
```

```
pwd
echo
$_
```

图 11.5 第三个版本的 won 程序

没有参数的 `echo` 可以用来产生空行。`echo` 命令以一个新行作为其输出的结尾。`\n` 选项禁止默认的新行符。

图 11.6 显示第三个版本的 `won` 程序的输出。现在看着更有意义而且更舒服些。

```
$ won
Date and time: Wed Nov 29 15:00:52 EDT 2001
Number of users on the system:14
Your current directory:/usr/students/david

$_
```

图 11.6 第三版本 `won` 程序的输出

11.2.1 使用特殊字符

就像在第 8 章中提到的, `echo` 命令可以辨认出被称为扩展符的特殊字符。它们都以反斜杠打头。用户可以将这些扩展符当作 `echo` 命令参数字符串的部分来使用。这些字符使用户可以更好地控制输出格式。例如, 下面命令产生 4 个新行。

```
$ echo "\n\n\n" [Return].....产生 4 个空行
3 个空行由 3 个 \n 字符产生, 另一个空行由 echo 命令和回车键默认产生。
表 11.2 总结了这些扩展符。下面命令显示使用这些扩展符的例子。
```

表 11.2 `echo` 命令特殊字符

扩展符	含 义
<code>\b</code>	一个退格
<code>\c</code>	在输出的结尾禁止产生新行([回车键])
<code>\n</code>	回车并产生一个新行
<code>\r</code>	回车, 但不产生新行
<code>\t</code>	一个制表符
<code>\0n</code>	0 后面跟 1, 2, 3 位八进制数字, 代表字符的 ASCII 码

【实例】

使用 `\n` 扩展符。

```
$ echo "\nHello\n" [Return].....使用 \n 扩展符
.....第 1 个 \n 产生一个空行
```

```

hello .....字 hello
.....第 2 个 \n 产生一个空行
.....echo 命令自身产生一个空行
$_ .....命令提示符

```

发出响声:[Ctrl-G]会在绝大多数终端产生一个响声,该响声对应的 ASCII 码是八进制数 7。用户可以用 \0n 格式在终端产生一个响声。

【实例】

使用 \0n 扩展符产生蜂鸣声。

```

$ echo "\07\07WARNING" [Return].....用 \07 使 PC 喇叭发声
WARNING
$_ .....命令提示符

```

【说明】

用户将会听到终端喇叭发出两声蜂鸣,随后,WARNING 信息显示在终端上。

清屏:在写脚本程序,特别是写交互式脚本时,用户通常在进行下一步操作前先清屏。方法之一就是使用 echo 命令的 \0n 扩展符将清屏符送往终端。清屏符随着不同的终端而不同,因此,用户可能需要向系统管理员查询终端技术手册来确定用户终端的清屏符。

【实例】

这里假设清屏符为 [Ctrl-z], ASCII 代码为八进制的 32,用 echo 命令清屏。

```

$ echo "\032" [Return].....清屏
$_ .....命令提示符

```

【说明】

这时屏幕被清空,命令提示符出现在屏幕左上角。

11.2.2 退出系统的风格

通常,用户使用 [Ctrl-d]或 exit 命令结束会话并退出系统。假如现在用户希望以自己的方式退出系统,如键入 bye 来退出。

【实例】

写一个名为 bye 的脚本,只包含一条命令 exit。

```

$ cat bye [Return].....显示 bye 脚本的内容
exit .....只有一条命令
$_ .....命令提示符

```

【实例】

将 bye 文件改为可执行文件,然后运行它以退出系统。

```

$ chmod u+x bye [Return].....改变文件模式
$ bye [Return].....退出
$ _ .....用户还在 UNIX 系统中

```

为什么 exit 命令没有起作用? 当用户给 shell 发出一个命令时,它创建一个子进程来执行

该命令(见第 8 章)。用户登录的 shell 是父进程,脚本文件是子进程。子进程(**bye**)在取得系统控制权后执行 **exit** 命令,结果终止了子进程(**bye**)。当子进程死亡后,控制权回到父进程(用户登录的 shell),因此,显示命令提示符。

要想使 **bye** 脚本工作,用户必须阻止 shell 创建子进程(使用下节描述的 **dot** 命令)。这里,shell 在自身环境中运行用户程序。这样,**exit** 命令会终止当前 shell(用户登录的 shell),于是退出系统。

11.2.3 执行命令:dot 命令

dot 命令(.)是一个 shell 内部命令。它可以使用户在当前 shell 中运行程序,而不创建新的子进程。这一点对用户试验脚本程序如启动文件 .profile 等非常有用。.profile 在用户的主目录中,它包含用户希望在每次登录时执行的任何命令。用户不必先退出系统然后再登录来激活 .profile 文件。使用 **dot** 命令,用户可以执行 .profile 文件,并将它的执行结果直接作用于当前 shell——用户登录 shell。

【实例】

现在回到用户的 **bye** 脚本程序,再次运行它,这次使用 **dot** 命令。

```
$ . bye [Return].....使用 dot 命令
UNIX System V Release 4.0
login:_ .....收到登录命令提示符
```

【注意】

与运行 UNIX 的其他命令一样,在 **dot** 与其参数之间必须有一个空格(本例中是 **bye** 程序)。

现在尝试一下另一个版本的 **bye** 脚本,这个版本不需要 **dot** 命令,因此用户可以简单地键入 **bye**,然后按回车键即可退出系统。

在第 8 章,我们学习了 **kill** 命令。如下脚本程序中的 **kill** 命令会终止所有进程,包括登录的 shell。

```
$ cat bye [Return].....新版本的 bye 脚本
kill -9 0 .....该命令终止所有进程
$_ .....命令提示符
```

现在只剩下一个问题——使用 **bye** 脚本而不考虑当前所处的目录。如果 **bye** 处于主目录,用户要么改变目录到主目录,要么在执行 **bye** 时键入完整路径名。

路径修改:用户可以修改 shell 变量 **PATH**,并将主目录加入其中。这样在执行命令时,shell 也会搜索主目录来查找要执行的命令。

【实例】

改变 **PATH** 变量,将主目录加入其中。

```
$ echo $ PATH [Return].....查看 PATH 变量
PATH = :/bin:/usr/bin
$ PATH = $ PATH: $ HOME [Return].....加入用户主目录
$ echo $ PATH [Return].....再次查看 PATH 变量
```

```
PATH = ./bin:/usr/bin:/usr/students/david
$_ .....命令提示符
```

现在一切都准备好了,在 \$ 命令提示符下键入 **bye**,然后按回车键,用户就会退出系统。

【说明】

要想使用 PATH 变量的改变长期有效,将改变后的 PATH 放到 .profile 文件中。这样,每次登录后,PATH 变量就被设为想要的路径。

命令置换:在 UNIX 中可以将一个命令的输出用作另一个命令的参数。shell 将执行被后引号()引起来的命令,并将该命令的输出替换在 **echo** 命令的参数串中。例如:

```
$ echo your current directory: `pwd` [Return]
```

输出结果将会是:

```
Your current directory: /usr/students/david
```

【说明】

在这种情况下,**pwd** 被执行,它的输出(即用户当前路径名)被放在 **echo** 命令的参数串中 **pwd`** 的位置,并替换掉 **pwd`**。

11.2.4 读取输入:read 命令

给变量赋值的一种方法是使用赋值符号——等号(=)。用户也可以将从标准输入得到的值存到变量中。

用户可以使用 **read** 命令读取用户输入,并将其值存放在一个自己定义的变量中。这是用户定义变量的最常使用的方法之一,特别是在交互式程序中,程序提示用户一定信息,然后读取用户的响应。在执行 **read** 时,shell 将一直等待,直到用户输入一行文本,然后将输入的文本存入到一个或多个变量中。变量放在 **read** 命令之后,并用按回车键来标志输入结束。

图 11.7 是一个显示 **read** 命令如何工作的脚本,名为 **kb_read**。

```
$ cat kb_read
#
# The read command example
#
echo "Enter your name: \c"      # prompt the user
read name                      # read from keyboard and save the input in name
echo "Your name is $name"     # echo back the inputted data
$_
```

图 11.7 一个 shell 脚本例子 kb_read

read 命令通常与 **echo** 命令结合使用:用 **echo** 命令提示用户需要输入什么,**read** 命令等待用户输入。

【实例】

调用 **kb_read** 脚本。

```

$ kb_read [Return].....运行
Enter your name : .....用户被提示
david [Return].....输入一个名字
Your name is david .....名字被回显
$ .....提示符

```

【说明】

1. 用户输入的串被存到变量 name 中,并被显示出来。
2. 把变量用引号引起来是一个好办法,因为无法预料用户会输入什么字符,因此要防止 shell 将特殊字符如 [*], [?] 等解释成元字符。

read 命令从输入设备读取一行。输入的第一个字存在第一个变量中,第二个字存在第二个变量中,依此类推。如果输入包含多于变量的字,则所有剩余的将存在最后一个变量中。

【说明】

赋给 shell 变量 IFS 的字符(见第 8 章)决定字间的分隔,大多数情况下,使用空格做分隔符。

图 11.8 显示一个名为 read_test 的简单脚本,它读取用户的响应并显示在终端上。

```

$ cat read_test
#
# A simple script to test the read command
#
echo "Give me a long sentence: \c"      # use prompt
read Word1 Word2 Rest                  # read user response
echo "$ Word1 \n $ Word2 \n $ Rest"    # display the contents of the variables
echo "End of my act"                   # end the act
$

```

图 11.8 简单脚本 read_test,测试 read 命令

【实例】

现在运行该 read_test 脚本。

```

$ read_test [Return].....假设 read_test 是一个可执行文件
Give me a long sentence;.....显示提示信息
Let's test the read command. [Return].....用户输入
Let's .....变量 $ word1 中的内容
test.....变量 $ word2 中的内容
the read command.....变量 $ Rest 中的内容
End of my act
$ .....命令提示符

```

上例中,用户输入的串由 5 个单词组成。read 命令有 3 个参数(变量 Word1、Word2 和 Rest)来存储用户的所有输入。前 2 个单词存入前 2 个变量,剩下的所有输入存到第三个变量 Rest 中。如果 read 命令的参数只有一个,比如只有 Word1,则所有的输入都将存入变量 Word1 中。

11.3 探索 shell 编程基础

现在,读者已经掌握了将一定顺序的命令放在一个文件中来创建一个简单的脚本文件。让我们进一步研究 shell 脚本编程,以掌握这一具有全部能力的编程语言来编写应用程序。

与任何完全的编程语言一样,shell 提供了命令和结构,可以使用户写出具有良好结构的、可读的并且可维护的程序。本节讲解命令和结构的语法。

11.3.1 注释

在编写计算机程序时文档是很重要的,这在编写 shell 脚本程序时也不例外。文档是用来解释程序的目的和逻辑以及在程序中使用不明显的命令所必需的。文档是写给作者自己或其他任何读程序的人的。如果读者在自己写完程序一周后再去读它,就会发现竟然有那么多代码不记得了。

【说明】

shell 将 # 认作注释符号,因此将忽略 # 后的字符。

下面命令显示注释行的使用:

```
# .....这是一个注释
# program version 3 .....这也是一个注释行
date # show the current date .....这是一个在行中间的注释
```

11.3.2 变量

与其他编程语言一样,UNIX shell 允许用户创建变量存储数据。要将值存入变量,只需要键入变量名,后面紧跟等号和希望存入该变量的值即可,如下所示:

```
variable = value
```

【注意】

等号两边不允许有空格。

与其他编程语言不同,UNIX shell 不支持数据类型(如整数、字符、浮点类型等)。它将任何赋给变量的值都解释为一串字符。例如, `count = 1` 意味着将字符 1 存入变量 count 中。

变量名遵守与文件名同样的语法规则(见第 7 章)。简言之,变量名只能以字母或下划线(_)开头,其后部分可以使用字母或数字。

shell 是一种解释性语言,用户放在脚本文件中的命令和变量可以在 \$ 命令提示符下直接键入。当然,这只是一次性的过程;如果想再次执行就得再次键入那些命令。

【实例】

下面例子是有效的变量赋值,除非用户改变或退出系统,否则它们将一直有效。

```
$ count = 1 [Return].....将字符 1 赋给 count
$ header = "The Main Menu" [Return].....将该字符串赋给 header
$ BUG = insect [Return].....将字符串 insect 赋给 BUG
```

【注意】

如果所赋的串中包含空格,则必须用引号引起来。

shell 脚本中的变量将保存在内存中,直到该 shell 脚本结束或被终止。用户也可以用 `unset` 命令清除它们,用法是:键入 `unset` 命令,指定要删除的变量名,然后按回车键。例如,下面命令删除名为 XYZ 的变量:

```
$ unset XYZ [Return]
```

显示变量

从第 7 章我们知道,用 `echo` 命令可以显示变量的内容。格式如下:

```
echo $ variable
```

【实例】

现在看一下上面例子中被赋值的 3 个变量的值。

```
$ echo $ count $ header $ BUG[Return].....显示存在这些变量中的值
1 The Main Menu insect
$_ .....命令提示符
```

命令替换

用户可以将一个命令的输出存入一个变量中。当用一对后单引号(`'`)引住一个命令时,shell 执行该命令,然后用该命令的输出替换该命令,如下所示:

```
$ DATE=`date` [Return].....将 date 命令的输出存入变量 DATE 中
$ echo $ DATE [Return].....现在查看一下 DATE 的值
Wed Nov 29 14:30:52 EDT 2001
$_ .....命令提示符
```

【说明】

`date` 命令的输出存入变量 DATE 中,`echo` 命令被用来显示 DATE 的内容。

11.3.3 命令行参数

shell 脚本可以从命令行读取最多 10 个命令行参数(也叫变元)送到指定的变量(也称为位置变量或参量)。命令行参数是用户在键入命令后所跟的项目,通常用空格分隔。这些参数被送给程序并改变程序的行为或使程序按特定的顺序执行。这些特殊变量(位置变量)以 0 至 9 计数,并被命名为 \$0、\$1、\$2 等等。表 11.3 为位置变量。

表 11.3 shell 位置变量

变 量	含 义
\$0	包含脚本的文件名,与命令键入的一样
\$1, \$2... \$9	分别包含第 1 到第 9 个命令行参数
\$#	包含命令行参数的个数

(续表)

变 量	含 义
\$@	包含所有命令行参数:"\$1 \$2 ... \$9"
\$?	包含最后一个命令的退出状态
\$*	包含所有命令行参数:"\$1 \$2 ... \$9"
\$\$	包含正在执行进程的进程 ID(PID)

使用特殊 shell 变量(位置变量)

现在看一下有助于理解这些特殊变量使用和安排的例子。假定有一个名为 BOX 的脚本文件,其模式已经用 **chmod** 命令改变为可执行文件。图 11.9 为该脚本文件。

```
$ cat BOX
echo "The following is the output of $ 0 script:"
echo "Total number of command line arguments: $# "
echo "The first parameter: $1"
echo "The second parameter: $2"
echo "This is the list of all the parameters: $*"
$ _
```

图 11.9 shell 脚本文件 BOX

1. 特殊变量 \$0 总是记录命令键入的脚本文件的名称。
2. 特殊变量 \$1、\$2、\$3、\$4、\$5、\$6、\$7、\$8 和 \$9 分别保存参数 1 至 9。多于 9 个命令行参数被忽略。
3. 特殊变量 \$* 包含所有传送给程序的命令行参数,把所有参数保存在一个串中。它可以包含多于 9 个参数。
4. 特殊变量 @\$ 与 \$* 一样包含所有命令行参数。但是,它将每个参数用引号引起来的形式保存。
5. 特殊变量 \$# 保存命令行参数的个数。
6. 特殊变量 \$? 保存脚本文件 **exit** 命令的退出状态。如果脚本文件中没有 **exit** 命令,则该变量保存文件中最后一个执行的非后台命令。
7. 特殊变量 \$\$ 保存当前进程的进程 ID。

下面各命令显示对 BOX 程序不同调用(包括有或没有命令行参数)以及运行结果。

【实例】

无参数调用 BOX,只键入文件名。

```
$ BOX [Return].....没有参数
The following is the output of the BOX script:
Total number of command line arguments: 0
The first parameter is:
The second parameter is:
This is the list of all parameters:
$ _ ..... 命令提示符
```

变量 \$0 包含被调用脚本的名字。在本例中,BOX 被存入变量 \$0 中。由于没有命令行参数,因此,\$# 的值为 0,并且 \$1、\$2 和 \$* 中的值为空。echo 命令显示出命令提示符。

【实例】

调用 BOX,包含 2 个命令行参数。

```
$ BOX IS EMPTY [Return].....没有参数
The following is the output of the BOX script:
Total number of command line arguments: 2
The first parameter is: IS
The second parameter is: EMPTY
This is the list of all parameters: IS EMPTY
$_ .....命令提示符
```

变量 \$0 包含被调用脚本的名字。在本例中,BOX 被存入变量 \$0 中。第一个参数保存在 \$1 中,第二个参数保存在 \$2 中。在本例中,IS 是第一个参数(保存于 \$1 中),第二个参数是 EMPTY(保存于 \$2 中)。 $*$ 中保存所有命令行参数,本例中为 IS EMPTY。

如果被调用的脚本文件有多于 9 个命令行参数,则第 9 个参数之后的参数将被忽略。但是,可以用特殊变量 \$* 得到它们。

赋值

给位置变量赋值的另一种方法是使用 set 命令。在 set 命令键入的参数被赋给位置变量。

【实例】

看下面的例子。

```
$ set One Two Three [Return].....赋三个值
$ echo $1 $2 $3 [Return].....显示赋给变量 $1 至 $3 的值
One Two Three
$ set `date` [Return].....另一个例子
$ echo $1 $2 $3 [Return].....显示位置变量
Wed Nov 29
$_ .....命令提示符
```

date 命令被执行,它的输出被送给 set 命令的参数。如果 date 命令的输出为:

```
Wed Nov 29 14:00:52 EDT 2001
```

则 set 命令有 6 个参数(空格是分隔符),\$1 保存了 Wed,\$2 保存了 Nov,依此类推。

【情景】现在写一个脚本文件并使用这些位置变量。假设用户想写一个脚本文件,先将当前目录下的指定文件保存到主目录下的 keep 目录中,然后调用 vi 编辑该文件(仍在当前目录下)。完成该工作的命令为:

```
$ cp xyz $HOME/keep [Return].....将指定文件 xyz 复制到 keep 目录
$ vi xyz [Return].....调用 vi 编辑
```

图 11.10 完成上述工作的脚本文件,名为 svi。在 svi 文件中的命令与前面用户在命令提示符下键入的命令相同。文件中使用了位置变量使之成为一个通用程序。它保存并编辑任何在它的命令行参数指定的文件。

```

$ cat svi
#
# svi:Save and invoke vi (svi) program.
#
DIR = $HOME/keep          # assign keep pathname to DIR
cp $1 $DIR                # copy specified file to keep
vi $1                     # invoke vi with the specified filename
exit 0                    # end of the program,exit
$

```

图 11.10 svi 脚本文件

如果 svi 工作,则 xyz 文件被存往 keep 目录。然后调用 vi 编辑器,用户可编辑 xyz 文件。用 ls 命令可以查看是否有一个 xyz 的副本存在了 keep 目录下。

【情景】假设 svi 的模式已改为可执行文件,而用户有一个名为 xyz 的文件在当前目录,用户想编辑它。

【实例】

现在运行 svi 并看一下结果。

```

$ svi xyz [Return].....运行 svi 并指定文件名
hello .....现在在 vi 编辑器中
~ .....文件 xyz 的内容被显示
~ .....vi 编辑器屏幕显示的其他部分
"xyz" 1 Line, 5 characters .....vi 编辑器的状态行
$ .....用户退出 vi 编辑后,显示命令提示符

```

如果用户当前目录并没有文件 xyz 会怎么样? 如果用户不指定文件,svi 程序会怎样工作? svi 将会工作,但可能不是按用户所希望的那样进行。在第一种情况下,cp 命令无法在当前目录中找到 xyz 文件,然后调用 vi 编辑器并创建一个新的名为 xyz 的文件。在第二种情况下,cp 命令同样会失败,然后 vi 被不带文件名地调用。两种情况下,用户都没有看到 cp 命令的出错信息,因为在出错信息出现之前 vi 即被调用。

该 svi 程序需要修改。它必须能够认识到出错,显示相应的出错信息,并且如果指定文件未在当前目录中或者用户未在命令行指定文件,则不调用 cp 或 vi。

在修改 svi 代码解决上述问题前,必须了解 shell 语言的一些其他命令和结构。

终止程序:exit 命令

exit 命令是一个 shell 内部命令,用户可以用来立即终止 shell 程序的运行。该命令的格式如下:

```
exit n
```

n 是退出状态,也称为返回码(RC)。如果没有提供退出值,则使用 shell 执行的最后一个命令的退出值。为了与其他 UNIX 程序(命令)通常在完成时返回一个状态值的做法一致,用户可以在编程时使脚本给父进程返回一个退出状态。如同大家知道的,在 \$ 命令提示符下键入 exit 来终止登录 shell,最终导致退出系统。

11.3.4 条件和试验

用户可以指定某命令的执行依赖于另外命令的执行结果。在写 shell 脚本时总会需要这种控制。

UNIX 系统中运行的每个命令都返回一个数值,用户可以通过对它的测试来控制程序流。一个命令要么返回一个 0,表示执行成功(条件为真),要么返回一个其他值,表示失败(条件为假)。这些真或假条件被用在 shell 编程构造中,通过对它们的判断来决定程序流。现在看一下这种结构。

if-then 结构

if 语句为检验某件事发生的条件是真还是假提供了一种机制。根据 if 判断的结果,可以改变程序中命令执行的顺序。下面是 if 语句(命令)的语法:

```
if [ condition ]
then
  commands
  ...
  last-command
fi
```

【说明】

1. if 语句以将 if 反写的字 fi 结束。
2. 程序中的缩进并不是必需的,但它们可以使程序更易于阅读。
3. 如果条件为真,则在 then 与 fi 之间的所有代码(也被称为 if 体)将被执行。如果条件为假,if 体将被忽略,而 fi 之后的程序将开始执行。

【注意】

条件外面的方括号是必不可少的,条件必须用空格(也可是制表符)包围起来。

【实例】

修改 svi 脚本文件使之包含 if 语句,如图 11.11 所示。if 语句为 svi 的输出增加了一些控制。

```
$ cat svi
#
# svi; save and invoke vi (svi) program.
# Adding the if statement
#
if [ $# = 1 ]          # check for the number of command line arguments
then
  cp $1 $HOME/keep    # copy specified file to keep
fi                    # end of the if statement
vi $1                 # invoke vi with the specified filename
exit 0                # end of the program,exit
$_
```

图 11.11 另一个版本的 svi 脚本文件

如果命令行参数的个数(\$#)为 1(即指定了一个文件),则条件为真,并且 **if** 体(**cp** 命令)被执行,接着调用 **vi** 编辑器编辑指定的文件。与前面最后一次执行结果相同。

如果命令行参数的个数(\$#)为 0(即未在命令指定文件),则条件为假,**if** 体(**cp** 命令)被忽略,而 **vi** 以未指定文件名方式调用。

if-then-else 结构

通过给 **if** 结构增加 **else** 子句,可以在判断条件为假时执行相应的命令。相应的语法如下所示:

```
if [ condition ]
then
    true-commands
    ...
    last-true-command
else
    false-commands
    ...
    last-false-command
fi
```

【说明】

1. 如果条件为真,则 **then** 与 **else** 之间的所有命令(即 **if** 体)将被执行。
2. 如果条件为假,**if** 体被忽略,**else** 与 **fi** 之间的命令,即 **else** 体被执行。

【实例】

修改 **svi** 脚本文件以包括 **if-then-else** 语句。图 11.12 显示这次修改的情况。

```
$ cat svi
#
# svi:save and invoke vi (svi) program.
# Add if and else statements
#
if [ $# = 1 ]          # check for the number of command line arguments
then
    cp $1 $HOME/keep  # copy specified file to keep
    vi $1              # invoke vi with the specified filename
else
    echo "You must specify a filename. Try again." # display error message
fi                    # end of the if statement
exit 0                # end of the program, exit
$_
```

图 11.12 另一版本的 **svi** 脚本文件

如果命令行参数的个数(\$#)为 1(即指定了一个文件),则条件为真,并且 **if** 体(**cp** 命令和 **vi** 命令)被执行。下面 **else** 体被忽略,**exit** 命令被执行,与前面最后一次执行的结果一样。

如果命令行参数的个数(\$#)不是 1,则条件为假,**if** 体(**cp** 命令和 **vi** 命令)被忽略,而 **vi** 以未指定文件名方式调用。**else** 体(**echo** 命令)被执行,**echo** 显示了错误信息,然后执行 **exit** 命

令。

【实例】

现在运行一下新版本的 svi 文件。

```
$ svi [Return].....无命令行参数
You must specify a filename. Try again.
$_ ..... 命令提示符
```

【说明】

1. 必须在命令行参数指定文件名,否则会出现错误提示信息,并且不会调用 vi。
2. 如果未指定文件名,则参数个数(位置变量 \$# 的值)为 0。因此,if 条件为假并且 if 体被忽略,而 else 体(echo 命令)被执行。

在这个版本中,svi 并不检查文件是否存在。如果指定的文件名并不存在于指定目录中,则复制命令会出错,但 vi 仍会被调用,而指定的文件名被 vi 用来创建一个新文件。

if-then-elif 结构

当在脚本文件中用到嵌套的 if 和 else 结构时,可以使用 elif(else if 的缩写)语句。elif 是 else 和 if 语句的结合。完整的语法形式如下:

```
if [ condition_1 ]
then
    command_1
elif [condition_2]
then
    command_2
elif [condition_3]
then
    command_3
...
...
else
    command_n
fi
```

图 11.13 所示 greetings 脚本文件根据一天的不同时间显示不同的问候语:在中午前显示 Good Morning,在 12:00 至 18:00 之间显示 Good Afternoon 等等。使用 if-then-elif 结构来决定早上、中午或晚上时间。

【说明】

1. 使用 date 命令其参数的 set 命令设定位置变量。
2. date 命令输出的日期时间串中 hour:minute:second 是第 4 个字段,它被赋给了位置变量 \$4。
3. 必须使用标准 shell(sh),该版本的 greetings 脚本才能正常工作。如果用户登录的是 ksh,则键入 sh 然后按回车键来调用 sh 的副本,之后执行 greetings 文件。用户可以键入 exit 返回登录 shell。

```

$ cat greetings
#
# greetings
# A program sample using the if-then-elif construct
# This program displays greetins according to the time of the day
#
set `date`           # set the positional variables to the data string
hour = $4            # store the part of the date string that shows the hour
if ["$hour"-lt 12]  # check for the morning hours
then
    echo "GOOD MORNING"
elif ["$hour"-lt 18] # check for the afternoon hours
then
    echo "GOOD AFTERNOON"
else
    echo "GOOD EVENING"
fi
$_

```

图 11.13 一个使用 if-then-elif 结构的例子

用户可以根据自己所掌握和希望使用的 shell 命令的不同,以多种形式编写 `greetings` 程序。例如,可以不使用 `set` 命令和位置变量,而利用 `date` 命令的功能只从它的输出中获得小时 (hour) 字段。

图 11.14 显示另一个版本的 `greetings` 程序。图中只显示了需要改变的部分,其他代码与原来的版本相同。

```

$ cat greetings
#
# greetings Version 2
# A sample program using the if-then-elif construct.
# This program displays greetings according to the time of the day.
#
hour=`date + %H`    # stores the part of the date string that shows
.....            # the hour the rest of the program

```

图 11.14 使用 if-then-elif 结构的例子脚本

其中, `hour = `date + %H`` 一行需要额外解释。`%H` 限制 `date` 命令的输出为日期串,只显示小时 (hour) 部分。

【实例】

使用 `date` 命令的功能。

```

$ date [Return].....显示日期和时间
Wed Nov 29 14:30:52 EDT 2001
$ date + %H [Return].....只显示小时部分
14
$_.....命令提示符

```

`date`命令有多个字段描述符可使用户限制输出或控制输出格式。在参数的开始位置键入`+`,其后紧跟字段描述符如`%H`、`%M`等。用户可以用`man`命令来获得字段描述符的全部列表。下面多看几个例子。

【实例】

使用 `date` 命令字段描述符。

```
$ date '+DATE : %m-%d-%y'[Return].....显示用连字符分隔的日期
DATE : 05-10-99
$ date 'TIME : +%H:%M:%S'[Return].....显示用冒号分隔的时间
TIME : 16:10:52
```

【说明】

1. 参数必须以加号开始,意思是输出格式由用户控制。每个字段描述符前面有一个百分号(`%`)。
2. 如果参数包含空格,则必须用引号引住。
3. 读者可以把 `greetings` 程序加入到自己的 `.profile` 文件中,这样,每次登录时系统会显示相应的问候语。

真或假: `test` 命令

`test` 是 shell 内部命令,用来评估作为参数给它的表达式为真还是假。如果表达式为真,则 `test` 返回 0,否则返回非 0 值。表达式可能很简单,比如只是比较两个数值是否相等;也可能很复杂,比如判断用逻辑操作符相关的多个命令。`test` 命令对编写脚本程序特别有用。实际上, `if` 语句中括住条件的方括号是 `test` 命令的种特殊形式。图 11.15 显示一个用 `test` 命令判断 `if` 条件的脚本例子。

```
$ cat test
echo "Are you OK?"          # user prompt
echo "input Y for yes or N for no; \c"  # user prompt
read answer                 # stores user response in answer
if test "$answer" = Y      # test if user entered Y
then
    echo "Glad to hear that"# user entered Y
else
    echo "Go home!"        # user did not enter Y
fi
$_
```

图 11.15 一个名为 `test` 的脚本文件

【说明】

1. `test` 脚本提示用户输入 Y(是)或 N(否),然后读取用户的输入。
2. 如果用户响应 Y,则 `test` 命令返回 0,这时 `if` 条件值为真,于是 `if` 体被执行,显示信息 `Glad to hear that`,其后的 `else` 体被忽略。
3. 用户的任何非 Y 输入都会使 `if` 条件为假,于是 `if` 体被忽略,而 `else` 体被执行,显示 `Go`

home!。

用方括号调用 `test` 命令:shell 为用户提供了另外一种调用 `test` 命令的方法。用户可以使用方括号([和])来代替单词 `test`。因此,if 语句可以写为如下形式:

```
if test "$variable" 5 value
或
if [ "$variable" > value ]
```

11.3.5 不同类别的判断

使用 `test` 命令可以判断多种类型的事物,包括:数值、串值和文件。下面分别解释。

数值

使用 `test` 命令可以判断(比较)2 个整数值。用户也可以用逻辑运算符组合比较表达式。格式如下所示:

```
test expression_1 logical operator expression_2
```

逻辑运算符有:

- 逻辑与(and)运算符(-a):如果 2 个表达式都为真,则 `test` 命令返回 0(条件码为真)
- 逻辑或(or)运算符(-o):如果其中 1 个表达式为真或 2 个都为真,则 `test` 命令返回 0(条件码为真)
- 逻辑非(not)运算符(!):如果表达式都为假,则 `test` 命令返回 0(条件码为真)

可以用来比较保存数值的的操作符总结在表 11.4 中。

表 11.4 test 命令的数值判断操作符

操作符	例示	含义 P
-eq	number1 -eq number2	number1 与 number2 是否相等
-ne	number1 -ne number2	number1 与 number2 是否不等
-gt	number1 -gt number2	number1 是否大于 number2
-ge	number1 -ge number2	number1 是否大于或等于 number2
-lt	number1 -lt number2	number1 是否小于 number2
-le	number1 -le number2	number1 是否小于或等于 number2

【情景】假设用户想要写一个脚本:读取 3 个输入 number(命令行参数),然后显示最大的一个数。使用 `cat` 命令可以看到图 11.16 为该程序的一种实现方法,名为 `largest`。

```
$ cat largest
#
# largest:
# This program accepts three numbers and shows the largest of them
#
echo "Enter three numbers and I will show you the largest of them > > \c"
read num1 num2 num3
```

```

if test "$ num1" -gt "$ num2" -a "$ num1" -gt "$ num3"
then
    echo "The largest number is : $ num1"
elif test "$ num2" -gt "$ num1" -a "$ num2" -gt "$ num3"
then
    echo "The largest number is: $ num2"
else
    echo "The largest number is: $ num3"
fi
exit 0
$_

```

图 11.16 shell 脚本文件 largest

【说明】

1. 在运行程序 largest 时,它会等待用户输入 3 个 number。之后,用户的输入被送给 if-then-elif 结构来找到最大值。

2. 如果输入的前 2 个数中没有最大值,则第一个 if 语句失效,接着,elif 也失效,该程序执行 else 语句。这里并不需要更多比较:如果最大值既不是第一个值也不是第二个值,则它一定是第三个值。

【实例】

试试运行 largest 程序。

```

$ chmod +x largest [Return].....将其变为可执行文件
$ largest 100 10 400 [Return].....带 3 个参数运行它
The largest number is: 400
$_ ..... 命令提示符

```

现在考虑一下如何改进 largest 程序。例如,能否使代码行更少些?(比如是否需要在 if、elif 和 else 体都要重复 echo 命令?)能否进行错误检验?(比如只输入 2 个数怎么办?)

串值

用户也可用 test 命令比较(判断)字符串。test 命令为字符串比较专门提供了一组操作符。这些操作符总结在表 11.5 中。下面例子为带串比较参数的 test 命令的使用。

表 11.5 test 命令串比较操作符

操作符	例示	含义
=	string1 = string2	string1 是否与 string2 匹配
!=	string1 != string2	string1 是否与 string2 不匹配
-n	-n string	string 是否包含字符(长度非 0)
-z	-z string	string 是否为空串

【实例】

使用空串的第 1 个试验。

```

$ STRING5 [Return].....声明一个空串
$ test -z $ STRING [Return].....判断串长度是否为 0

```

```
test: Argument expected ..... 错误信息
$_ ..... 命令提示符
```

【说明】

1. shell 用空字符替换了 \$STRING, 因此出现错误信息。
2. 用引号将字符串变量引起来可以保证正确的检验, 即使是字符串包含空格或制表符也如此。

【实例】

现在把 \$STRING5 用引号引起来再试一下。

```
$ test -z "$STRING" [Return] ..... 判断串长度是否为 0
```

【说明】

这次 test 命令将返回 0(真), 说明变量 \$STRING 包含一个长度为 0 的字符串。

【实例】

下面例子中, 直接在 \$ 命令提示符下键入命令。如果命令未结束, shell 将显示第二命令提示符以等待用户完成命令。

```
$ DATE1 = `date` [Return] ..... 初始化 DATE1
$ DATE2 = `date` [Return] ..... 初始化 DATE2
$ if test "$DATE1" = "$DATE2" [Return] ..... 判断是否相等
> then [Return] ..... 因为 if 命令并未结束, shell 显示第二命令提示符
> echo "STOP! The computer clock is dead!" [Return]
> else [Return] ..... else 体开始
> echo "Everything is fine." [Return]
> fi [Return] ..... if 结束。这时, 按回车键后命令即被执行
Everything is fine. .... 程序输出
$_ ..... 出现命令提示符
```

【说明】

该 test 命令(if 条件)结果为假。因此, else 体被执行并显示相应的信息。保存在 DATE1 与 DATE2 中的值会相等吗?

文件

用户可以用 test 命令检验文件特性, 如文件长度、文件类型和文件权限等。可以检验多达 10 个文件特性, 这里仅介绍其中的一部分。表 11.6 总结了文件检验操作符。下面例子显示 test 在检验文件中的用法。命令均在 \$ 命令提示符下键入。

表 11.6 test 命令文件检验操作符

操作符	示例	含义
-r	-r filename	文件 filename 是否存在并可读
-w	-w filename	文件 filename 是否存在并可写
-s	-s filename	文件 filename 是否存在并且长度非 0
-f	-f filename	文件 filename 是否存在但不是目录
-d	-d filename	文件 filename 是否存在并且是一个目录文件

【实例】

假设用户有一个名为 `myfile` 的文件,其权限为只读。现在键入下面命令并看一下输出结果。

```

$ FILE=myfile [Return].....初始化 FILE 变量
$ if test -r "$FILE" [Return]..... 查看 myfile 是否可读
> then[Return].....第二命令提示符
> echo "READABLE" [Return].....显示信息
> elif test -w "$FILE" [Return]..... 查看文件是否可写
> then[Return]
> echo "WRITEABLE" [Return].....显示信息
> else[Return].
> echo "Read and Write Access Denied" [Return].
> fi [Return].....if 结构结束
READABLE
$.....回到基本命令提示符

```

【说明】

1. 用户一旦键入 `fi`,`if` 命令的结尾,shell 即执行该命令,生成输出并显示命令提示符。
2. 第一个 `if` 检验 `myfile` 的读访问权限。由于用户有读权限,因此 `test` 命令返回 0(真),`if` 条件为真。因此,只有第一个 `if` 体被执行,显示信息 `READABLE`。
3. 如果用户对 `myfile` 有写权限,则信息 `WRITEABLE` 也会显示。
4. 如果用户对 `myfile` 既没有读权限也没有写权限,则信息 `Read and Write Access Denied` 将被显示,这时 `if` 条件和 `elif` 条件都为假。

图 11.17 显示了 `svi` 脚本文件的另一个版本。在该版本中,检验指定文件是否存在。只有该文件存在时,`cp` 和 `vi` 命令才会执行。

```

$ cat svi
#
# The save and invoke vi (svi) program.
# This version checks for the file's existence.
#
if test $# = 1                # check for number of arguments.
then
    if test -f $1             # check for the file existence
    then
        cp $1 $HOME/keep     # copy and invoke vi
        vi $1
    fi
    else                       # file not found
        echo "You must specify a filename. Try again."
    fi                         # wrong number of arguments
else
    echo "You must specify a filename. Try Again."
fi
$_

```

图 11.17 另一个版本的 `svi` 脚本文件

【实例】

运行该新版本的 svi 程序。

```
$ svi xyz [Return].....运行它,指定一个不存在的文件 xyz
File not found. Try again.
$ svi [Return].....再次运行,不指定文件
You must specify a filename. Try again.
$_.....回到命令提示符
```

【说明】

1. 判断 if 条件时,使用的是 test 命令来代替方括号。
2. 该程序使用了嵌套 if 结构(一个 if 包含在另一个 if 中)。
3. 如果未找到指定文件,则显示 **File not found. Try again.** 信息。
4. 如果找到了指定文件,则将该文件复制到 keep 目录,并调用 vi 编辑器。
5. 如果命令行未指定文件名,则第一个 if 条件为假,于是显示 **You must specify a filename. Try again.** 信息。

11.3.6 参数替换

shell 提供一种参数替换功能,使得用户可以检验参数的值并根据选项改变它的值。如果用户在编程时希望检查某一变量是否设成某值时,这一功能非常有用。例如,用户在脚本文件有一个 read 命令,用户希望确认使用者在进行下一动作之前输入一些信息。

其格式为一个美元符号(\$)、一对大括号({})、一个变量、一个冒号(:)、一个选项符号和一个字,如下所示:

```
$ {parameter:Option character word}
```

选项字符(Option character)决定用户希望如何处理字(word)。有 4 个符:-、+、? 和 =。这 4 个选项根据变量是否为空完成不同的动作。

如果一个变量的值是一个空串,则这个变量是一个空变量(null 变量)。例如,下面 3 个变量值都被设置为空,因而是空变量。

```
EMPTY = .....名为 EMPTY 的空变量
EMPTY = "" .....名为 EMPTY 的空变量
EMPTY = ' ' .....名为 EMPTY 的空变量
```

【注意】

在创建空变量时,引号中不能有空格或制表符。

表 11.7 总结了 shell 的变量替换(评估)选项。每个选项都有一个简单的解释和例子。

表 11.7 shell 变量评估选项

变量选项	含 义
\$ 变量	保存在变量中的值
\${变量}	保存在变量中的值
\${变量:-字符串}	如果变量被设定并非空,则是变量的值,否则是字符串的值

(续表)

变量选项	含 义
\$ {变量:+ 字符串}	如果变量被设定并非空,则是字符串,否则是变量的值
\$ {变量:= 字符串}	如果变量被设定并非空,则是变量的值,否则变量被设为字符串
\$ {变量:? 字符串}	如果变量被设定并非空,则是变量的值,否则显示字符串并退出

\$ {parameter}:将变量(参数)放入大括号中可以防止任何由变量名后面紧跟的字符引起的冲突。下面例子更清楚地解释了这一问题。

【实例】

假设现在用户想把一个名为 memo 的文件改变名为 memoX,文件名 memo 已经存入变量 FILE 中。

```
$ echo $ FILE [Return]..... 查看一下 FILE 中存了什么
memo
$ mv $ FILE $ FILEX [Return]..... 将 memo 改为 memoX
Usage: mv
$_ ..... 命令提示符
```

该命令没有工作,因为 shell 认为 \$ FILEX 是一个变量的名字,而这个变量并不存在。

```
$ mv $ FILE $ {FILE}X [Return]..... 将 memo 改为 memoX
$_ ..... 命令提示符
```

这一次命令正确工作。这是因为 shell 认为 \$ FILE 是变量名,并将它替换为该变量的值,在本例中是 memo。

\$ {parameter:-word}:连字符-选项意味着,如果列出的变量(parameter)已被设置并且非空,则使用该变量的值。否则,如果该变量为空或未设置,用 word 替换它的值。例如:

```
$ FILES [Return]..... 这是一个空变量
$ echo $ {FILE:-/usr/david/xfile} [Return].. 显示 FILE 的值
/usr/david/xfile
$ echo "$ FILE" [Return]..... 查看 FILE 变量
..... 其值仍为空
$_ ..... 命令提示符
```

【说明】

1. shell 对 FILE 变量进行判断,该变量值为空,因此-选项导致用/usr/david/xfile 代替 FILE 变量送往 echo 命令并被显示。

2. FILE 变量仍旧为空。

\$ {parameter:+ word}:+ 选项与-选项相反。它意味着,如果列出的变量(parameter)已设定并且不空,则用 word 代替它的值。否则,该变量的值保持不变。例如:

```
$ HELP="wanted" [Return]..... 设置 HELP 变量
$ echo $ {HELP:+ "Help is on the way"} [Return]
Help is on the way
$ echo "$ HELP" [Return]..... 查看变量 HELP 的值
```

```
wanted.....保持不变
$_ .....命令提示符
```

【说明】

1. shell 评估变量 HELP 的值,已设为 wanted。于是 + 选项导致用 **Help is on the way** 串代替 HELP 的值送往 **echo** 命令显示。

2. HELP 的值保持不变。

\$ {parameter:=word};= 选项意味着如果列出的变量(parameter)未设置或值为空,则用 word 替换它。否则,如果变量值非空,则保持不变。例如:

```
$ MSG= [Return].....空变量
$ echo $ {MSG:="Hello There!"}.....显示 MSG 的值
Hello There!
$ echo "$ MSG" [Return].....查看 MSG 变量
Hello There!.....已经被赋给值
$_ .....命令提示符
```

【注意】

word 可以是包含空格的串,但要用引号引住。

【说明】

1. shell 评估 MSG 的值,为空,所以 = 选项将 MSG 的值赋为 **Hello There!**。替换结束,echo 命令显示保存在 MSG 中的值。

2. MSG 的值已变,并且不再是空变量。

\$ {parameter:? word};? 选项意味着如果列出的参数(parameter)已设置且值不为空,则取其值。否则,如果该变量为空,显示 word 并退出该脚本。如果 word 被省略,则预先设定的信息 **parameter null or not set**。例如:

```
$ MSG= [Return].....MSG 是一个空变量
$ echo $ {MSG:? "ERROR!"} [Return].....根据选项完成替换
ERROR!
$_ .....命令提示符
```

【说明】

shell 评估 MSG 的值,是一个空变量。于是? 选项导致 ERROR 串代替 MSG 变量的值被送往 echo 命令,并显示。

当脚本程序使用者对 read 命令只简单地用[Return]来响应时,用户可以用该选项显示一个错误信息,然后终止该 shell 脚本。图 11.18 显示一个名为 name 的脚本例子。

【实例】

运行 name 文件。

```
$ name [Return].....运行该脚本文件
Enter your name: _ .....提示信息
[Return].....假设程序使用者对 read 仅做回车响应
You must enter your name.....显示错误信息
$_ .....命令提示符
```

```

$ cat name
#
# Program to test parameter substitution
#
echo "Enter your name: \c" # prompts the user
read name
echo ${name:? "You must enter your name"}
echo "Thank you. That'all."
exit 0
$_

```

图 11.18 一个名为 name 的脚本文件

【说明】

1. 如果对 `read` 命令仅响应一个回车, 则 `name` 变量仍为空。? 选项判断该变量, 发现它的值为空, 于是指定的信息被显示, 同时 shell 脚本终止。

2. 但是, 如果键入一个名字或一个单词, 则脚本文件会继续, 而信息 `Thank you. That's all.` 将显示出来。

11.4 算术运算符

shell 没有提供简单的内部算术运算符来完成算术运算。例如, 如果想把一个 shell 变量的值加 1, 操作的结果可能并不是用户希望的那样。

```

$ x = 10[Return].....初始化, 将 10 赋值给 x
$ echo $ x[Return].....显示 x 的值
10.
$ x = $ x + 1[Return].....将 x 的值加 1
$ echo $ x[Return].....显示 x 的值
10 + 1
$_.....命令提示符

```

【说明】

shell(sh) 并没有将 `x` 的值加 1。它只是将字符串 `+ 1` 追加到 `x` 的串值之后。

11.4.1 算术运算 (sh): `expr` 命令

使用 `expr` 命令可以计算表达式的值。该命令提供算术运算符的功能, 并能够计算数字或非数字字符串。`expr` 命令以参数为表达式, 计算该表达式并将结果送往标准输出。

算术运算符

下面各命令介绍算术运算符并示范 `expr` 命令对常量运算的使用。

【实例】

下面例子示范常量的加、减运算。

```

$ expr 1 + 2[Return]..... + 号是加运算符
3
$ expr 15 - 6[Return]..... -号是减运算符
9
$ expr 6 - 15 [Return]
-9
$ expr 2.4 - 1[Return]..... 错误参数(2.4),只能是整数
expr: nonnumeric argument.....UNIX 错误信息
$ expr 1 + 1[Return]..... 也是一个错误参数
1+1..... 该命令仅显示串 1+1
$_

```

【注意】

运算符与运算元素之间必须有空格。正确的形式是 `expr 1 + 1`。注意 + 两边都有空格。

【实例】

下面例子显示乘、除和取余数运算。

```

$ expr 10 / 2[Return]...../号是除运算符
5
$ expr 10 \* 2[Return]..... * 号是乘运算符
20
$ expr 10 \% 3[Return]..... % 是取余运算符
1
$_

```

【注意】

由于 * (乘)和 % (取余)在 shell 中有特殊含义,因此它们前面必须有前导字符 \ (反斜号),以使 shell 忽略它们的特殊意义。

【实例】

下面命令示范将一个整数与 shell 变量相加。

```

$ x=10..... 初始化 x 为 10
$ x=`expr $x + 1`..... 将 x 加 1
$ echo $x[Return]..... 显示 x
11
$_..... 命令提示符

```

【注意】

必须用后单引号[']将命令行起来,以使 `expr` 的执行结果替换为输出。

关系运算符

`expr` 命令提供既可用于数字也可用于非数字的关系运算符。如果两个参数都是数字,则进行数字比较。如果有一个参数是非数字的,则进行非数字比较,并使用 ASCII 值。关系运算符有:

```

= ..... 相等
!= ..... 不等

```

```

< .....小于
< = .....小于等于
> .....大于
> = .....大于等于

```

【说明】

1. 比较结果相等时, `expr` 显示 1, 表示真。
2. 比较结果不等时, `expr` 显示 0, 表示假。

【实例】

下面各命令显示 `expr` 使用关系运算符的情况。

```

$ expr Gabe = Gabe[Return].....字符比较
1.....显示 1,表明为真
$ expr Gabe = Daniel[Return].....字符比较
0.....显示 0,表明为假
$ expr 10 \ < 20[Return].....数字比较
1.....显示 1,表明为真
$ expr 10 \ > 20[Return].....数字比较
0.....显示 0,表明为假
$_.....命令提示符

```

【注意】

再一次,由于 `>` (大于号)和 `<` (小于号)对 shell 有特殊含义,因此必须用反斜号 `[\]` 作前导符,以使 shell 忽略它们的特殊意义。

```

$ expr 6 \> A[Return].....混合比较(数字与字母)
0.....显示 0,表明为假
$_.....命令提示符

```

【说明】

这里 `expr` 命令将 6 认作一个字符参数,并且比较 6(54)与 A(65)的 ASCII 值。

11.4.2 算术操作(ksh):let 命令

用户可以使用 `let` 命令处理整数。`let` 命令与 `expr` 命令相似,并可互相替换。它包括了 `expr` 命令的所有基本算术运算,包括加、减、乘和除。例如:

```

$ x=100[Return].....初始化 x 为 100
$ let x=x+1[Return].....将 x 加 1
$ echo $x[Return].....显示 x 的值
101
$ let y=x*2[Return].....用 let 命令进行乘法
$ echo $y[Return].....显示 y 的值
202
$_

```

【说明】

1. `let` 命令自动使用变量的值。在本例中,直接键入 `x` 而不需 `$x` 即可取得 `x` 的值。
2. `let` 命令将 `*` 和 `%` 符号分别解释为乘和取余运算符,因而不需要使用 `*` 或 `\%` 来消除

它们的特殊含义。

11.5 循环结构

当程序中需要重复一系列声明或命令时,应当使用循环结构。循环结构可以为程序员节省大量时间。为了将一个简单信息显示 100 次就写 100 行代码是无法想像的。shell 提供了三种循环结构:for 循环、while 循环和 until 循环。这此循环可以使用户按一定的次数或直到某一条件到来而重复执行某些命令。

11.5.1 for 循环:for-in-done 结构

for 循环用于将一系列命令执行指定的次数。其基本格式如下:

```
for variable
in list-of-values
do
  commands
  . . .
  last-command
done
```

shell 扫描 list-of-values,将第一个字存在循环变量(variable)中,然后执行 do 与 done 之间的命令(即循环体)。接着,将第二个字保存在循环变量中并再次执行循环体。list-of-values 包含循环体中的命令被执行的次数。

【实例】

下面各命令显示 for 循环如何工作。

```
$ for count in 1 2 3[Return].....设置循环头
> do[Return].....等待用户完成命令行
> echo "In the loop for $ count times" [Return]...显示信息
> done [Return].....循环结束
In the loop for 1 times
In the loop for 2 times
In the loop for 3 times
$_.....返回命令提示符
```

【说明】

1. 在本例中,count 是循环变量,list-of-values 由三个数字(1、2 和 3)组成,循环体只有一个 echo 命令。
2. 由于 list-of-values 中包含三个值,因此,循环体被执行了三次。
3. list-of-values 中的值被依次赋给变量 count,每次赋一个值,直到 list-of-values 中的值全部赋完。

【情景】现在假设希望打印文件,在打印的同时把文件名保存在一个文件中。可以写一个名为 slp(Super LP)的脚本文件,该程序打印用户的文件并把文件的信息保存在一个名为 pfile 的文件中。图 11.19 为该脚本文件的一种实现方式。

```

$ cat slp
#
# The super line printer program (slp).
#
echo "Enter the name of the file(s). \c"           # prompt the user
read filename                                     # read input
for FILE in $ filename
do echo "nFilename: $ FILE \n Printed: `date`> > pfile # save in pfile
lp $ FILE                                         # print the file
done
echo "\n \07Job done"                             # inform user
exit 0
$ _

```

图 11.19 slp 脚本文件

【说明】

1. 用户可以输入多个文件名。每个文件被依次从变量 filename 中赋给循环变量 FILE。**echo** 命令负责将该文件的信息保存到文件 pfile 中，**lp** 命令打印相应的文件。
2. 第一次使用该程序时，**echo** 命令创建 pfile 文件。此后，新打印文件的信息被追加(> > 重定向符)到 pfile 中。

【实例】

运行 slp 程序。

```

$ slp [Return].....运行程序
Enter the name of the file(s).....等待输入
myfile[Return].....键入 myfile
lp: request id is lp 1-9223 (f file) ....lp 命令的信息
job done .....蜂鸣声和最后显示信息
$ _ .....命令提示符
$ cat pfile [Return] .....查看 pfile
Filename: myfile
Printed: Mon Dec 6 12:01:35 EST 2001
$ _ .....命令提示符

```

11.5.2 While 循环: while-do-done 结构

这里介绍的第二个循环结构是 **while** 循环。与 **for** 循环不同之处在于：**for** 循环的重复次数由 list-of-values 值的个数决定，而 **while** 循环是只要循环条件为真即继续循环。其格式如下：

```

while [ condition ]
do
  commands
  ...
  last-command
done

```

只要循环条件为真(非 0)，循环体(在 **do** 与 **done** 之间部分)的命令即重复执行。循环条件

必须最终变为假(0),否则该循环会成为一个无限循环而不停地循环下去。这里用户必须了解系统的 **kill** 命令以便终止这样的程序。

【实例】

下面命令显示 **while** 循环是如何工作的。

```
$ carry_on=Y [Return].....初始化 carry_on 变量
$ while [ $ carry_on = Y ] [Return].....建立 while 循环
> do [Return]
> echo "I do the job as long as you type Y:_ \b" [Return]
> read carry_on [Return]
> done[Return].....读取输入
I do the job as long as you type Y:_ .....程序等待输入
Y[Return].....键入 Y 并按回车键
I do the job as long as you type Y:_ .....程序再次等待输入
N[Return].....键入 Y 并按回车键
$_ .....出现 shell 命令提示符
```

【说明】

在这个程序中,只要用户键入 Y,循环条件即为真,循环体(**echo** 和 **read** 命令)就重复执行。要停止该程序只需键入非 Y 的任何字符。

图 11.20 为一个名为 COUNTER 的脚本文件的源代码,该 COUNTER 程序是 **while** 循环结构的另一个例子。它的功能是显示 0 到 9 的数字。程序中使用了 **expr** 命令来计算将要显示的下一个值。**cat** 命令用来显示源代码。

```
$ cat COUNTER
#
# COUNTER: counts from 1-9 using while loop
#
count = 1                # initialize the count variable
while [ $ count -lt 10 ]
do
    echo $ count         # display count value
    count=`expr $ count + 1` # increment count
done
$_
```

图 11.20 COUNTER 脚本文件

【说明】

要注意命令 **expr \$ count + 1** 被后单引号引起来。在运行该程序时,该命令的输出保存到变量 **count** 中。

【实例】

运行该 COUNTER 程序。

```
$ COUNTER [Return].....运行该 COUNTER 程序
1
```

```

2
3
4
5
6
7
8
9
$ _ .....命令提示符

```

图 11.21 为该 COUNTER 脚本文件的一个新版本,其中用 `let` 命令代替 `expr` 命令。`cat` 命令用来显示源代码。

```

$ cat COUNTER
#
# COUNTER: counts from 1-9 using while loop and the let command
#
count = 1                               # initialize the count variable
while [ $count -lt 10 ]
do
    echo $count                          # display count value
    let count = count + 1                 # increment count
done
$ _

```

图 11.21 COUNTER 脚本文件的新版本

【说明】

下面版本的 COUNTER 程序只适用于 ksh shell。

let 命令缩写

`let` 命令可以缩写为双括号,即 `(())`。图 11.22 为另一版本的 COUNTER 程序。在该版本中,用 `let` 命令的缩写形式来代替 `expr` 命令。

```

$ cat counter
#
# COUNTER: counts from 1-9 using while loop and the let command
#      abbreviation (( ))
#
count + 1                               # initialize the count variable
while (( count < 10 ))
do
    echo $count                          # display count value
    (( count = count + 1 ))               # increment count
done
$ _

```

图 11.22 COUNTER 脚本文件的另一版本

11.5.3 Until 循环:until-do-done 结构

这里介绍的第三种循环结构是 **until** 循环。**until** 循环与 **while** 循环类似,所不同的是 **until** 循环只要循环条件为假(非 0 值)就执行循环体。**until** 循环在编写需要由其他事件的发生作为执行条件的 shell 脚本程序时非常有用。其格式如下:

```
until [ condition ]
do
    commands
    ...
    last-command
done
```

【注意】

如果在第一次执行时循环条件即为真,则循环体可能会永远不执行。

【情景】现在假设要写一个完成如下功能的脚本:查看指定用户当前是否登录到系统上,如果没有,则在他登录时进行报告。图 11.23 为该脚本的一种实现方式,名为 uon(User ON)。

```
$ cat uon
#
# uon: Let me know if xxx is on the system.
#
until who | grep "$ 1" >/dev/null          # redirect the output of grep
do
    sleep 30                               # wait half a minute
done                                       # display count value
echo "\07\07$ 1 is logged on."          # inform the user
exit 0                                     # end of the if statement
$_
```

图 11.23 uon 脚本文件

在 uon 脚本中,只要循环条件为真,**until** 循环就终止。如果 **grep** 命令(详见第 8 章)未能在 **who** 命令送给它的用户列表中发现指定的用户 ID,则循环条件保持为假(非 0 值),并且 **until** 循环继续执行循环体——**sleep** 命令。一旦 **grep** 命令在用户列表中找到指定的用户 ID,循环条件即变为真(0),循环终止。接着,循环之后的命令开始执行,即通过两声蜂鸣通知程序使用者:指定的用户已登录到系统上了。

【说明】

1. **grep** 命令的输出被重定向到空设备。也就是说,用户既不想看到该输出,也不想保存它。
2. **sleep** 命令使该程序停止 30 秒钟。从网络的效果来看,就是 **grep** 命令每半分钟检查一次用户列表。

现在只剩下一个问题。如果用户在前台运行该程序,而指定的用户一直未登录,则用户无法使用该终端,直到指定的用户登录到系统才行。解决的方法是在后台运行 uon 程序。

【实例】

后台运行 uon 程序。

```
$ uon emma&[Return].....在后台运行该程序检查 emma 是否登录到系统
4483 ..... 进程 ID
$_ ..... 出现 shell 命令提示符
```

现在用户可以做自己想做的其他事情。如果 emma 登录,则系统会用两声蜂鸣通知用户:

```
emma is on the system. ....收到通知
```

现在用户可以继续做自己的事情。

11.6 调试 shell 程序

在编写较长并且较复杂的脚本文件时很容易出错。由于脚本文件是非编译程序,所以不会有编译器进行查错。因此,用户只能运行程序并尝试解释屏幕显示的错误信息。但是,千万不要绝望。

11.6.1 sh 命令

使用带参数的 sh 命令可以使脚本文件的调试变得简单。例如,-x 选项使得 shell 显示它所执行的每个命令。这样跟踪脚本的执行可以帮助用户发现程序中的 bug。

sh 选项

表 11.8 总结了 sh 命令的选项,下面例子显示如何使用它们。

表 11.8 sh 命令选项

选项	功能
-n	读取命令但不执行
-v	在读取输入时显示 shell 输入行
-x	在执行命令时显示命令的参数

如果想要调试 BOX 脚本文件,键入如下命令:

```
$ sh -x BOX [Return]
```

用户也可认把 sh 选项用作命令放入脚本文件中。使用 set 命令把下面命令放到脚本文件的开始或用户希望开始调试的任何位置:

```
set -x
```

-x 选项:-x 选项显示脚本文件中的命令,包括命令的参数以及完成的命令替换。现在我们使用带-x 选项的 sh 命令,再次执行 BOX 脚本文件,并探索一下可能的情况。

【实例】

带-x 选项运行 BOX 脚本,但不带命令行参数。

```
$ sh -x BOX [Return].....使用-x 选项
+ echo The following is the output of the BOX script.
The following is the output of the BOX script.
+ echo Total number of command line arguments:0
Total number of command line arguments:0
+ echo The first paramter is:
The first paramter is:
+ echo The second parameter is:
The second parameter is:
+ echo This is the list of all parameters:
This is the list of all parameters:
$_ ..... 返回命令提示符
```

【说明】

1. 以加号(+)开头的行是 shell 执行的命令行,其下面的行是命令的输出。
2. echo 命令的输出是在变量替换完成之后进行的。

【实例】

再次带 sh 命令及-x 参数运行 BOX 脚本,并加上一些命令行参数。

```
$ sh -x BOX of candy [Return].....使用-x 选项
+ echo The folling is the output of the BOX script.
The following is the output of the BOX script.
+ echo Total number of command line arguments:2
Total number of command line arguments:2
+ echo The first parameter is :of
The first parameter is :of
echo The second parameter is ;candy
The second parameter is ;candy
+ echo This is the list of all parameters: of candy
This is the list of all parameters:of candy
$_ ..... 返回命令提示符
```

-v 选项:-v 选项与-x 选项相似。但是,它显示的是在变量替换和命令完成前的命令行。

【实例】

带-v 选项运行 BOX 脚本文件,带有命令行参数。

```
$ sh -v BOX is full of gold nuggets [Return].....使用-v 选项
echo "The following is the output of the $0 script."
The following is the output of the BOX script.
echo "The total number of command line arguments: $# "
Total number of command line arguments:5
echo "The first parameter is: $1"
The first parameter is :is
echo "The second parameter is: $2"
The second parameter is: full
echo "This is the list of all parameters: $*"
This is the list of all parameters: is full of gold nuggets
$_ ..... 返回命令提示符
```

【说明】

1. 一行显示被 shell 执行的命令,下面一行显示该命令的输出。
2. 显示的 `echo` 命令未进行变量替换(这是该选项与 `-x` 选项不同之处,后者显示完成变量替换后的命令)。

用户可以将 `-x` 和 `-v` 选项用在同一个命令行。同时使用 2 个选项使用户能够在执行后分别查看文件中的命令及输出结果。下面命令行为调用 `sh` 命令同时带这两个选项。

```
$ sh -xv BOX [Return]
```

-n 选项: `-n` 选项用来检查用户脚本文件中的语法错误。使用这个选项,用户可以在运行程序前先检查所写的程序是否有语法错误。

例如,假设现有一个名为 `check_syntax` 的脚本文件。图 11.24 为该程序的源代码,其中有意包含了语法错误。

```
# cat check_syntax
#
# check_syntax: Sample program to show the output of the sh -n option.
#
echo "$0:Checking the program syntax"
if [ $# -gt 0]
echo "Number of the command line arguments: $# "
else
echo "no command line arguments"
fi
echo "GOODBYE"
$_
```

图 11.24 `check_syntax` 脚本文件源代码

【实例】

带 `-n` 选项运行 `check_syntax` 程序,观察其输出结果。

```
$ sh -n check_syntax [Return].....运行
check_syntax: syntax error at line 8 'else' unexpected
$_ ..... 返回命令提示符
```

【说明】

1. 使用 `-n` 选项不执行任何文件中的命令。该选项只是找到并指出语法错误。
2. 如果使用 `-x` 或 `-v` 选项,程序将执行到出现语法错误的地方。然后显示错误终止程序。

在第 8 行出现了什么错误呢? 在 `if-then-else` 结构中, `if` 语句后面必须有 `then`。

纠正该错误后该程序的输出会是什么呢? 位置变量 `$0` 包含该程序的名字,本例中即 `check_syntax`。 `$#` 包含命令行参数的个数。如果有命令行参数,则 `if` 体将被执行,否则将执行 `else` 体。

【实例】

再次运行 check_syntax。

```

$ check_syntax one two three [Return].....使用 3 个命令行参数
check_syntax: checking the program syntax
Number of the command line arguments: 3
$_ .....返回命令提示符
$ check_syntax [Return].....无命令行参数
check_syntax: checking the program syntax
No command line arguments
$_ .....返回命令提示符

```

在编写长而且复杂的 shell 程序时,sh 调试选项通常是很有用的。它们使读者在探索第 12 章中出现的脚本文件时倍感便利。

命令小结

本章中讨论了如下命令及其选项。

chmod

该命令改变选项字母指定类别的用户对某特定文件的访问权限。用户类别有:u(使用者/所有者)、g(所得者的同组成员)、o(非所有者及其同组的其他用户)和 a(所有用户)。访问权限有:r(读)、w(写)和 x(执行)。

sh

该命令调用一个新的 shell。用户可以用该命令运行自己的脚本文件。本章只介绍了 3 个选项。

选项	功 能
-n	读取命令但不执行
-v	在读取输入时显示 shell 输入行
-x	在执行命令时显示命令行及其参数。该选项多用于程序调试

.(点)

该命令使得用户在当前 shell 环境中执行命令,而不使 shell 创建子进程。

exit

该命令终止当前 shell 程序。它也可以返回一个状态码(RC)来指示程序执行的成功与否。如果用户在 \$ 命令提示符下键入该命令,它将终止用户登录 shell 并使用户退出系统。

read

该命令从输入设备读取输入,并存入一个或多个命令参数指定的变量中。

test

该命令判断作为其参数的表达式值为真还是为假,并返回相应的值。它使得用户能够判断不同类型的表达式。

expr

该命令是一个内部的算术运算操作符,提供算术和逻辑运算。

let(ksh)

该命令在 Korn shell 中有效,提供算术运算。

习题

1. 如何执行一个 shell 脚本文件?
2. 什么命令使一个文件成为可执行文件?
3. 什么时候使用.(点)命令?
4. 什么命令从键盘读取输出?
5. 解释命令行参数。
6. 什么是 shell 位置变量?
7. 位置变量与命令行参数如何相关?
8. 如何调试 shell 脚本?
9. 说出用来终止 shell 脚本的命令。
10. shell 语言中有哪些结构?
11. 循环结构的用途是什么?
12. while 循环与 until 循环的区别是什么?
13. 实现算术运算的命令是什么?
14. 两个整数相加的命令是什么?
15. 使两个变量值相乘的命令是什么?

上机练习

【实例】

在本次终端会话中,读者需要编写几个脚本文件,并改进本章中出现的部分脚本文件。

1. 创建一个名为 LL 的脚本文件,以长格式列出用户目录。
 - a. 用 sh 命令运行 LL。
 - b. 将 LL 变为一个可执行文件。
 - c. 再次执行 LL。
2. 创建一个脚本文件完成如下功能:
 - a. 清屏。
 - b. 空 2 行。

- c. 显示当前的日期和时间。
- d. 显示当前在系统上的用户数。
- e. 发出一小会儿蜂鸣声,然后显示:**Now at your service**
3. 修改本章中的 `largest` 脚本文件,使之能够辨认出输入的数字并显示适当的信息。
4. 写一个与 `largest` 相似的脚本文件,计算三个从键盘读入的整数的最小值。要求能够辨认出部分输入错误。
5. 为本章中在 `$` 命令提示符下键入的脚本例子创建一个相应的脚本文件。必要时做相应修改,然后运行它们。使用 `sh` 带 `-x` 和其他选项进行调试,并研究 shell 脚本执行的方式。
6. 写一个脚本文件,对在命令行参数传给它的数字求和。程序中要求使用 `for` 循环。例如,该程序名为 `SUM`,键入:

```
$ SUM 10 20 30 [Return]
```

程序显示:

```
10 1 20 1 30 5 60
```

7. 用 `while` 循环重写 `SUM` 脚本。
8. 用 `until` 循环重写 `SUM` 脚本。
9. 修改 `largest` 脚本,使之从命令行接收数字。例如,键入 `largest 1 2 3`,程序显示:

```
The largest number is : 3
```
10. 修改本章中的脚本文件 `greetings`,其中用 `cut` 命令来计算小时。

第 12 章 shell 脚本:编写应用程序

本章以前一章的命令和概念为基础,讨论其他 shell 编程命令和技巧。通过简单的应用程序例子,显示如何用 shell 语言开发程序的过程。当使用 shell 脚本的时候,介绍一些新的 shell 命令。

12.1 编写应用程序

第 11 章介绍了 shell 编程基础,提到用 shell 命令语言编写应用程序。本章解释开发应用程序的过程。例子是完整的程序。当需要时,介绍并使用新的命令和语法结构。

【情景】用户希望保护终端的访问(假如用户需要离开几分钟),但是用不着重复注销和登录。用户可以编写一个脚本文件,当激活该脚本时,屏幕上显示消息,直到用户输入正确的密码才退出。你真的想要锁定终端吗?其实这里的目的只是学习命令和编程技巧,并将其应用到脚本文件中,而不考虑是否真要锁定终端。

12.1.1 程序 lock1

图 12.1 显示了一个名为 lock1(锁屏,版本 1)的脚本,通过锁定用户键盘来实现其功能。lock1 程序用 vi 编辑器显示,行号并非代码的一部分。用 vi 编辑器的 `set nu` 参数可以产生行号。

```
1 #
2 # name: lock1 (lock version 1)
3 # definition: This program locks the keyboard, and you must type the
4 # specified password to unlock it.
5 # logic:
6 #     1- Ask the user to enter a password.
7 #     2- Lock the keyboard until the correct password is entered.
8 #
9 echo "\032"                # clear screen
10 echo "\n\nEnter your PASSWORD>" # ask for password
11 read pword_1              # read password
12 echo "\032"                # clear screen again
13 echo "\n\nTHIS SYSTEM IS LOCKED..."
14 pword_2=                    # declare an empty variable
15 until ["$pword_1" = "pword_2"] # start of the loop
16 do
17 read pword_2              # body of the loop
18 done                      # end of the until loop
19 exit 0                    # end of the program, exit
```

```
file "lock1" 20 lines 166 characters
```

图 12.1 lock1 脚本文件

我们逐行来看程序 lock1 如何工作。

1~8 行:这些行以 # 号开始,是文档说明。

9 和 12 行:这几行的命令是清屏。除了用 `echo` 和清屏代码,也可以用 `tput` 命令(本章稍后介绍)。

10 行:这一行提示用户输入口令。字符、数字和嵌入白字符的任何序列都可以接受。这里输入的口令与用户的登录口令无关。

11 行:读取键盘输入。按回车键后,用户输入的口令被保存到变量 `pwd_1` 中。

13 行:屏幕上显示合适的消息。可以改成任何其他信息。

14 行:声明变量 `pwd_2`,将用来存储从键盘的输入。

15~18 行:这几行建立了 `until` 循环。循环条件是比较变量 `pwd_1` 的内容(口令)和变量 `pwd_2` 的内容(键盘输入)。第一次循环,两个变量的内容不同(`pwd_1` 包含口令,而 `pwd_2` 为空)。因此,`until` 循环体被执行。注意,当循环条件为假,执行 `until` 循环体。循环体为 `read` 命令,等待从键盘读取。这里没有任何提示符,仅显示光标。(用户毕竟不希望告诉入侵者系统正等待口令)如果输入不正确的口令,`until` 循环继续,并读取键盘。`read` 命令的重复有效地锁定了键盘。读取输入、比较、比较失败,然后继续读取。当用户输入了正确的口令后,循环条件为真(`pwd_1` 与 `pwd_2` 相等)。循环停止,键盘锁定被释放。

19 行:以 0 状态标志退出 lock1 脚本,表明程序正常结束。

程序 lock1 的缺陷

lock1 程序运行良好,但有一些小问题,可以改进。下面进行说明。

lock1 并非一个很安全的程序:

- 当键盘被锁定时,如果用户按[Del](中断键),脚本终止,显示出 \$ 提示符,系统等待接收命令。为了有效避免他人入侵系统,lock1 程序必须能够忽略正常的中断信号。
- 口令被明文显示出来。通常,口令不应该显示在屏幕上。在第 10 行,当要求口令时,不论用户输入什么,都回显到屏幕上,任何人都可以看到。必须避免在口令提示符下,用户的响应被显示。
- 提示信息是固定的,总是显示 **The system is locked**。能够在命令行设定提示信息会更友好。

为了解决这些问题,需要用到另外几个命令。

12.2 UNIX 内部:信号(SIGNAL)

如何终止一个进程?用户可以通过产生一个终端信号来终止进程。“信号”(signal)向进程报告特定状况。例如,[Del]、[Break]和[Ctrl-c]被用来向进程发送中断信号,终止进程。

注意,进程只认为在与文件打交道,并不知道用户终端。那么,从键盘输入的中断信号如何才能终止进程呢?其实,用户中断信号送到 UNIX 内核,而非用户进程。内核具有设备信息,当用户按下任何中断键,消息送到了内核。UNIX 发送给进程一个信号,告诉它发生了中断。对应这个信号,进程将终止或进行其他操作。

有几类不同的事件导致内核向进程发送信号。对这些信号进行了编号,以表明它们代表

着指定什么事件。表 12.1 总结了一些脚本文件中常用的信号。用户系统中信号的编号有可能不同。若希望了解,可以请教系统管理员,或者查找该系统的参考手册。

表 12.1 一些 shell 信号

信号数	信号名	含义
1	挂起	失去终端联系
2	中断	任一中断键被按下
3	退出	任一退出键被按下
9	杀死	发出 kill -9 命令
15	终止	发出 kill 命令

挂起信号:信号 1 被用来通知进程系统已经与终端失去联系。当用户终端和计算机的连线断开,或者电话线(用调制解调器相连)断开时,产生该信号。或者,如果用户关闭终端,一些系统也会产生挂起信号。

中断信号:当任何一个中断键被按下,产生信号 2。中断键可以是[Ctrl-c]、[Del]或者[Break]。

【注意】

系统的中断键可能不同。每个特定的系统只有某个键起作用。

退出信号:当按下[Ctrl-\]时,键盘产生信号 3。在终止之前,引起进程进行 core dump。

终止信号:用 kill 命令(见第 8 章)生成信号 9 与信号 15。信号 15 是默认的信号。当 kill 命令带参数-9 时,产生信号 9。这两者都使收到信号的进程终止。

12.2.1 信号数俘获信号:trap 命令

当收到任何信号时,进程的默认响应是立刻终止。用户能用 trap 命令改变进程的默认行为,执行指定的动作。例如,用户能指示进程忽略中断信号,或代替终止而继续执行指定的命令。下面来说明其方法。trap 命令的格式如下:

```
trap "optional commands" signal numbers
```

commands 部分可选。如果有这一部分,当进程收到任一指定俘获的信号时,执行那些命令。

【注意】

trap 指定的命令必须用单引号或者双引号括起来。

【说明】

1. 可以指定俘获多个信号数。
2. 信号数是与用户希望用 trap 命令捕捉的信号相联系的数字。

【实例】

下面的命令显示如何使用 trap 命令。

```
trap "echo I refuse to die!" 15
```

当收到简单的 **kill** 命令时,这条命令执行 **echo** 命令,显示 **I refuse to die!** 消息。而脚本会继续运行。

```
trap "echo Killed by a signal!; exit" 15
```

如果进程收到 **kill** 命令(信号 15),**echo** 命令执行,显示 **Killed by a signal!** 消息。接着执行 **exit** 命令,脚本终止。

```
trap "" 15
```

没有命令指定。现在,如果进程收到 **kill** 命令,就会忽略它,继续运行。

【注意】

如果没有命令指定,必须有引号。如果没有引号,**trap** 命令理解成复位指定信号。

12.2.2 陷阱复位

脚本中运行 **trap** 命令改变了进程收到信号的默认行为。而当执行不带可选命令部分的 **trap** 命令时,又恢复指定信号的默认行为。当用户在脚本的一部分希望俘获信号,而另一部分不希望设陷阱的时候,这个命令就很有用。

例如,脚本文件中输入如下命令:

```
$ trap "" 2 3 15
```

中断、退出和终止信号都被忽略了,如果这些键中的任一个被按下,脚本都继续保持运行。如果输入如下命令:

```
$ trap 2 3 15
```

指定的信号被复位,也就是说,恢复了中断、退出和终止键。如果按下其中任何键,运行着的脚本就会终止。

12.2.3 设置终端参数:stty 命令

stty 命令用来设置和显示终端属性。用户可以控制终端的各种属性,例如波特率(终端与计算机之间的传输率)和特定键(杀死、中断等)的功能。不带参数的 **stty** 命令显示选定的一组设置。用 **-a** 参数列出所有终端设置。

图 12.2 显示终端设置的例子。用户系统的设置可能不同。

```
$ stty
speed 9600 baud;-parity
erase = '^h';kill = '^u';
echo
$_
```

图 12.2 终端设置的例子

终端的性能可能变化很大,**stty** 支持改变超过 100 种不同的设置。一些设置改变终端的联系方式,一些改变指定的键值,一些具有组合效果。

表 12.2 列出了可用的参数中很小的部分,也是最常用的。用 `man` 命令得到参数更详细的列表和功能的解释。

表 12.2 终端参数的简单列表

选 项	功 能
<code>echo [-echo]</code>	回显[不回显]输入字符;默认回显
<code>raw [-raw]</code>	禁止[启用]特殊意义的元字符;默认为启用
<code>intr</code>	生成中断信号;通常用[Del]键
<code>erase</code>	(回退)擦除前一个字符;通常用#键
<code>kill</code>	删除整行;通常用@或者[Ctrl-u]键
<code>eof</code>	从终端产生文件结束(eof)信号;通常用[Ctrl-d]键
<code>ek</code>	用#和@键分别复位 erase 和 kill 键
<code>sane</code>	用合理的默认值设置终端属性

【说明】

1. 默认设置对大多数选项来说通常已经是最佳的。
2. 一些参数可以通过输入参数名启用,参数名前输入连字符(-)禁用。

【实例】

下面看一些例子。

```
$ stty -echo [Return].....关闭回显
$ stty echo [Return].....打开回显
```

【注意】

`stty echo` 命令并不显示在终端上,因为前面的 `stty -echo` 命令已经起作用了。

【实例】

把 `kill` 键设为[Ctrl-u]。

```
$ stty kill \^u [Return].....现在[Ctrl-u]是 kill 键
$_.....回到命令提示符
```

【说明】

要设置一个特殊键,可以按三个字符([\]、[^]和特殊字母)或者直接输入组合键。在这个例子中,可以输入 `\^u` 或者按[Ctrl-u]。

```
$ stty sane [Return].....恢复参数的合理值
$_.....出现命令提示符
```

如果用户把参数改变了太多次,以致不知道现在的状态了,可以用 `sane` 参数来恢复。

【实例】

把 **kill** 和 **erase** 键恢复成默认值。

```
$ stty ek [Return].....设置 kill 和 erase 键
$_.....出现命令提示符
```

此命令将 **kill** 键设成 @ , **erase** 键设成 # 。

12.3 终端的进一步讨论

UNIX 操作系统支持各种类型的终端。每一种终端都有各自的性能和特性。终端用户/技术手册中记录了这些性能,一组转义字符使用户得以操纵终端的这些特性。每种终端类型有各自的一组转义字符。在第 11 章里,转义字符 \032 被用来清屏。 \032 是 vt100 类型终端的清屏代码。输入以下命令清屏:

```
$ echo "\032" [Return].....在 vt100 型终端上清屏
```

终端的功能并不限于清屏。还有许多其他特性,例如粗体字、闪烁、下划线等等。用户利用它们能使显示更有意义、组织更高或者更美观。

12.3.1 终端数据库:terminfo 文件

系统支持的每一类终端在终端数据库(名为 terminfo 的文件)有一个条目。terminfo 数据库是一个简单文本文件,包含许多类型终端的描述。数据库里,有一个功能列表与每一类终端相关连。

12.3.2 设置终端功能:tput 命令

作为所有系统与 terminfo 数据库相连的标准命令, **tput** 工具允许用户打印出任一功能的值。这样就可以在 shell 编程中使用终端的功能。例如,如下输入清屏:

```
$ tput clear [Return]
```

【说明】

无论终端类是哪种,这条命令都能工作,只要系统中包含着 terminfo 数据库,并且终端类型在数据库中即可。

表 12.3 显示了一些终端功能,可以用 **tput** 命令激活。**tput** 程序与 **terminfo** 数据库相连,允许用户选择特定的终端性能,显示它们的值或者将其保存在 shell 变量里。默认情况下, **tput** 假设用户使用的终端类型设置在 shell 变量 **TERM** 中。用 **-T** 选项可以覆盖它。例如,输入如下命令,指定终端类型:

```
$ tput -T wy50 [Return]
```

【说明】

1. 通常情况下,如果启动一种模式,直到删除之前,它一直起作用。
2. 可以把字符串存在变量里,然后使用这些变量。

表 12.3 终端功能的简单列表

选项	功 能	选项	功 能
bel	回显终端的响铃字符	el	从光标位置到行末清除字符
blink	闪烁显示	smso	启动突出模式
bold	粗体显示	smul	启动下划线模式
clear	清屏	rmso	结束突出模式
cup r c	把光标移到 r 行 c 列	rmul	结束下划线模式
dim	显示变暗	rev	反白显示
ed	从光标位置到屏幕底清屏	sgr0	关闭所有属性

【注意】

用户能使用 `sgr0` 参数关闭所有用户定义的终端属性。这是一个非常重要的选项,因为它是关闭一些属性(比如闪烁)的惟一方法。

下面的命令显示用 `tput` 改变终端属性。

【实例】

用 `tput` 命令先清屏,然后在 10 行 20 列的位置显示信息“The terminfo database”。

```
$ tput clear [Return].....清屏
$ tput cup 10 20 [Return].....置光标位置
$ echo "The terminfo database" [Return].....显示消息
The terminfo database
```

【说明】

在 `echo` 命令之前设定光标位置,信息就显示在屏幕指定的位置处。

几条命令可以放在一行。命令之间用冒号隔开,如下:

```
$ tput clear: tput cup 10 20: echo "The terminfo database" [Return]
```

【实例】

把字符串保存在变量里,用来控制屏幕显示。

```
$ bell = `tput bel` [Return].....把响铃的设置信息保存在 bell 变量中
```

shell 执行后引号之间的命令,把命令输出(这里是终端响铃的字符序列)赋给变量 `bell`。

```
$ s_uline = `tput smul` [Return].....保存启用下划线的代码
$ e_uline = `tput rmul` [Return].....保存禁用下划线的代码
$ tput clear [Return].....清屏
$ tput cup 10 20 [Return].....设置光标位置
$ echo $ bell [Return].....响铃
$ echo $ s_uline [Return].....启用下划线显示
$ echo "The terminfo database" [Return].....显示带下划线的信息
The terminfo database
```

```
$ echo $ e _uline [Return].....结束下划线显示
```

echo 命令能够把已赋值的变量结合到命令中。如下：

```
$ echo "$ bell$ |s _uline|The terminfo database $ e _uline" [Return]
```

【注意】

\$ |s _uline|的大括号是必需的,shell 用来识别变量名(见第 11 章,变量替换)。

12.3.3 解决 lock1 程序的问题

现在用 stty、trap 和 tput 命令,修改 lock1 脚本中的问题,并建立一个更好的用户接口。图 12.3 显示 lock 脚本的新版本,下面解释修改与增加的行。

```
1 #
2 # name: lock2 (lock version 2)
3 # definition: This program locks the keyboard, and you must type the
4 # specified password to unlock it.
5 # logic:
6 #     1-Ask the user to enter a password.
7 #     2-Lock the keyboard until the correct password is entered.
8 #
9 trap ""2 3          # ignore the listed signals
10 stty -echo # prohibit echoing the input
11 tput clear # clear the screen
12 tput cup 5 10; echo "Enter your PASSWORD>" \c          # ask for password
13 read pword_1      # read password
14 tput clear # screen again
15 tput cup 10 20 ;echo "THIS SYSTEM IS LOCKED..."
16 pword_2 =          declare an empty variable
17 until [" $pword_1" = "$pword_2"]          # start of the loop
18 do
19 read pword_2      # body of the loop
20 done              # end of until
21 stty echo         # enable echo of input characters
22 exit 0            end of program, exit

file "lock2" 23 lines 166 characters
```

图 12.3 lock2 脚本

9 行:这一行为信号 2 和信号 3 设置陷阱。脚本忽视中断和退出键,当用户按下这些键时,脚本继续运行。

10 行:禁用中断回显的能力。因而接下来输入的字符(这里是口令)不会显示。

11 和 14 行:清屏。用 tput 命令代替了 echo 命令。

12 行:此行有两个命令,用分号(;)隔开。第一个命令在 5 行 10 列设置光标,第二个命令在光标位置显示信息。

15 行:这一行与 12 行类似,有两个命令。前一个命令在 10 行 20 列设置光标,第二行在光标位置显示信息。

23 行:将终端的回显能力复位。当用户输入了正确的口令解开键盘后,触发该事件。现在,如果用户把 lock2 修改成可执行文件并执行,输入如下命令:

```
# chmod +x lock2 [Return].....修改程序访问模式
# lock2 [Return]..... 执行
```

屏幕如图 12.4 所示,提示用户输入口令。

```
Enter your PASSWORD> _
```

图 12.4 lock2 程序的提示符

输入字符或数字序列(并不显示)作为口令。用户必须记住它,来解锁键盘。接着,清除终端屏幕,屏幕如图 12.5 显示。

```
THIS SYSTEM IS LOCKED...
```

图 12.5 lock2 程序的信息

现在,锁定了键盘,只有口令才可以将其解锁。如果忘了口令会怎样呢?此时不能用 [Del]、[Ctrl-c]或者其他键终止 lock2 程序。trap 命令忽略这些信号,程序继续保持键盘锁定状态。程序没有为 kill 信号(9 和 15)设置陷阱,但是用户并不能用键盘执行 kill 命令。解决办法很简单:用另一个终端登录,执行 kill 命令来结束这个终端的 lock2 程序。

设定显示信息:如前所述,我们可以改进 lock2 程序,给用户更多的自由以设定屏幕上的提示信息,或者选择默认的信息。指定信息必须通过命令行传递给程序。例如,输入:

```
# lock3 Coffee Break.Will be back in 5 minutes [Return]
```

用户要修改 lock2 程序以适应新的改变。新版本中(称作 lock3),用户可以在命令行输入信息,或者和以前一样不指定信息而执行。程序必须能识别这两种情况,显示合适的信息。图 12.6 显示了 lock3 程序的代码,下面解释它如何工作。

```
1 #
2 # name: lock3 (lock program version 3)
3 # definition:This program locks the keyboard,and you must type the
4 # specified password to unlock it.
5 # logic:
6 #     1:Ask the user to enter a password.
7 #     2:Lock the keyboard until the correct password is entered.
8 #
9 trap ""2 3 4                # ignore the listed signals
10 stty -echo                  # prohibit echoing the input
11 if [ $# -gt 0 ]             # on-line message is specified
12 then
```

```

13      MMSG = "$@"          # store the specified message
14 else
15      MMSG = "THIS SYSTEM IS LOCKED"
16 fi
17 tput clear                # clear the screen
18 tput 5 10 ; echo "Enter your PASSWORD >" # ask for password
19 read pword_1             # read password
20 tput clear                # clear screen again
21 tput .cup 10 20 ; echo "$ MMSG"         # display the message
22 pword_2 =                 # declare an empty variable
23 until ["$ pword_1" = "$ pord_2" ]      # start of the loop
24 do
25 read pword_2              # body of the loop
26 done                      # end of until
27 stty echo                 # enable echo of input characters
28 exit 0 # end of program,exit

"lock3" 28 lines 166 characters

```

图 12.6 lock 程序,第3版

11~16行:这几行是 **if-then-else** 结构。**if** 条件检查记录参数个数的位置变量 **\$#** 的取值。如果 **\$#** 大于 0,说明用户在命令行设置了信息,**if** 体执行。位置变量 **\$@** 保存着设置的信息,保存到变量 **MMSG** 中。

如果 **\$#** 不大于 0,**else** 体执行。变量 **MMSG** 保存着信息。

21行:显示变量 **MMSG**(用户指定或者默认)的信息。

其余行与 **lock2** 版本相同,实现同样的功能。和前面一样,要改变 **lock3** 的访问模式为可执行模式,然后可以在命令行以设定或不设定信息的方式执行该命令。

12.4 更多的命令

下一节把所有的 **shell** 编程技巧放到一起,编写多个脚本组成的应用程序。然而,在描述这个应用程序的使用场景之前,有必要介绍几个命令。

12.4.1 多路分支:case 结构

shell 提供了 **case** 结构,用来从命令列表中有选择地执行一组命令。用 **if-elif-else** 结构可以达到同样效果,但是当需要使用许多 **elif** 语句(例如超过 2 个或 3 个)时,用 **case** 结构更好。采用 **case** 结构,而非多个 **elif** 语句,是更好的编程方式。

case 结构的语法如下:

```

case variable in
  parttern_1
    commands_1;;
  pattern_2

```

```

    commands_2;;
    ...
    ...
    *)
    default_commands ;;
esac

```

`case`、`in` 和 `esac` (`case` 的反序) 是保留字(关键字)。在 `case` 和 `esac` 之间的语句称为 `case` 结构体。

shell 执行 `case` 语句时,与每一个模式比较变量的内容,直到发现一个匹配或者 shell 到达关键字 `esac`。shell 执行与匹配的模式相联系的命令。默认情况用 `*` 表示,必须是程序的最后一种情况。每一条 `case` 语句的结束用两个分号(;;)表示。

我们来看一个用 `case` 结构编写的简单菜单程序。菜单系统给用户提供了容易和简便的接口,大多数计算机用户都熟悉菜单系统。用 shell 脚本很容易实现菜单。

用户通常希望菜单程序具有下面的功能:

- 显示可能的选项
- 提示用户选择一个功能
- 读取用户输入
- 根据用户输入调用其他程序
- 如果输入有误,提示错误信息

用户输入一个选择,菜单程序必须识别出用户的选择,并做出相应的动作。用 `case` 结构很容易实现这种识别。例如,假设有个名为 `MENU` 的脚本,源代码如图 12.7 所示。下面的命令显示程序的运行和输出。

```

$ cat MENU
#
# A simple menu program to demonstrate the use of the case construct.
#
echo "0:Exit"
echo "1:Show date and Time "
echo "2:List my HOME directory"
echo "3:Display Calendar"
echo "Enter your choice:"
read option
case $ option in
    0) echo good bye ;;
    1) date ;;
    2) ls $ HOME ;;
    3) cal ;;
    *) echo "Invalid input. Good bye.;"
esac
$

```

图 12.7 简单 MENU 程序的代码

【实例】

运行 MENU 程序。

```

$ MENU(Return).....Execute the menu program
0: Exit
1: Show Date and Time
2: List my HOME directory
3: Display Calendar
Enter your choice:.....Wait for input

```

菜单显示,提示用户进行选择。输入保存在变量 `$option` 中。`$option` 被传递给 `case` 结构,它识别用户选择,并执行相应命令。

如果输入 0: `$option` 包含 0,在 `case` 结构中模式 0 匹配。因而,执行 `echo` 命令,显示消息“good bye”。没有其他模式匹配,程序结束。

如果输入 1: `$option` 包含 1,在 `case` 结构中模式 1 匹配。因而,执行 `date` 命令,显示日期和时间。没有其他模式匹配,程序结束。

如果输入 2: `$option` 包含 2,在 `case` 结构中模式 2 匹配。因而,执行 `ls` 命令,显示用户主目录的内容。没有其他模式匹配,程序结束。

如果输入 3:与前面的选择相似。变量 `$option` 匹配情况 3,显示 `calendar` 命令的输出。

如果输入 5:如果用户错输了 5 或者 7,会怎样呢? 如果 `case` 结构有默认情况(* 匹配所有的情况),当其他情况匹配失败时,就会匹配默认并执行与之相关的命令。这里,显示“Invalid input. Good bye.”。

12.4.2 回顾 greetings 程序

现在来编写另一个版本的 `greetings` 程序(第 11 章介绍过)。这个新版本里,用 `case` 结构代替了 `if-elif-else`。图 12.8 显示了程序的一种写法。

```

$ cat greetings2
# greetings2: greeting program version 2
# This version uses the case construct to check the hour of the day
# and to display the appropriate greetings
#
bell = `tput bel`           # Store the code for the bell sound
echo $bell$bell            # two beeps
hour=`date + %H`           # obtain hour of the day
case $hour in
  0?1[0-1] ) echo "Good Morning";;
  1[2-7] ) echo "Good Afternoon";;
  * ) echo "Good Evening";;
esac
exit 0
$_

```

图 12.8 greetings 程序的新版本

输出与前一个版本相同。响铃两次,根据时间,显示合适的问候。`hour` 变量(包含着小时)传递给 `case` 语句。保存在 `hour` 里的值与 `case` 语句的模式进行匹配,直到匹配成功。下面进一步解释这些模式。

第一种模式 `0?|1[0-1]` 检查上午的小时(从 01 到 11 点)。`0?` 代表着 0 和另一个数字(0 到 9)组成的两位数。因此 `0?` 匹配从 00 到 09。`1[0-1]` 代表 1 和另一个数字(0 到 1)组成的两位数。因此, `1[0-1]` 匹配 10 和 11。`|`(管道)符号表示逻辑或,因此如果两个模式中的任何一个与变量 `hour` 的值相匹配,整个表达式为真。如果 `hour` 的值在 01 到 11 之间, `echo` 命令显示 **Good Morning**。

第二种模式 `1[2-7]` 检查下午的小时(从 12 到 17 点)。它表示 1 和范围在 2 到 7 的另一个数字组成的两位数。因此, `1[2-7]` 匹配 12 到 17 的值。如果 `hour` 在 12 到 17 之间取值, `echo` 命令显示 **Good Afternoon**。

第三种模式 `*` (默认)匹配任何值。如果 `hour` 的值在 18 到 23 之间,前两种模式都不匹配, `echo` 命令显示 **Good Evening**。

12.5 菜单驱动的应用程序

【情景】 假设用户想要编写一个菜单驱动的应用程序,以管理 UNIX 书籍。用户希望能够更新图书列表,了解指定的图书是否在库中,是否出借,谁以及何时借的。

当然,这是一个典型的数据库应用,但是我们的目标是练习使用一些 UNIX 命令,给用户一些如何把命令组成有用程序的感受。

12.5.1 层次图

图 12.9 显示菜单驱动的 UNIX 图书库程序(称作 ULIB)的层次图。当调用 ULIB 时,显示主菜单,等待输入选择。每一个条目进入另一级菜单。如同多数菜单驱动的用户接口,ULIB 层次图的顶级代表用户接口。这里,ULIB、EDIT 和 REPORTS 三个程序显示合适的菜单,等待用户选择菜单项。

层次图的每一个方框代表一个程序。先来看第一个程序,图的顶层 ULIB。图 12.10 显示了该程序的一种写法。行号并不是代码的一部分。必要时,行号方便对程序中的命令进行解释。

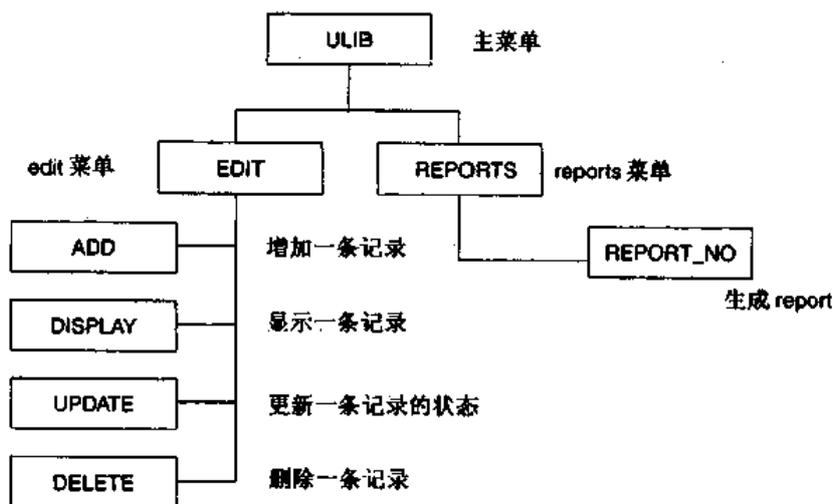


图 12.9 ULIB 程序层次图

```

1 #
2 # UNIX library
3 # ULIB: This program is the main driver for the UNIX library application
4 #   program. It shows a brief startup message and then displays the main
5 #   menu. Invokes the appropriate program according to the user selection.
6 #
7 BOLD=`tput smso`           # store code for bold mode in BOLD
8 NORMAL=`tput rmso`        # store code for end of the bold in NORMAL
9 export BOLD NORMAL        # make them recognized by subshells
10 #
11 # show the title and a brief message before showing the main menu
12 #
13 tput clear                # clear screen
14 tput cup 5 15            # place the cursor on line 5, column 15
15 echo "$BOLD|Super Duper UNIX Library" # show the title in bold
16 tput cup 12 10          # place the cursor on line 12, column 10
17 echo "$NORMAL|This is the UNIX library application." # the rest of the title
18 tput cup 14 10 ; echo "Please enter any key to continue..._ \b \c"
19 read answer              # read user input
20 error_flag=0             # initialize the error flag, indicating no error
21 while true                # loop forever
22 do
23   if [ $error_flag -eq 0 ] # check for the error
24   then
25     tput clear            # clear screen
26     tput cup 5 10
27     echo "UNIX Library - $BOLD| MAIN MENU $NORMAL|"
28     tput cup 7 20 ; echo "0: $BOLD|EXIT $NORMAL| this program"
29     tput cup 9 20 ; echo "1: $BOLD|exit $NORMAL| Menu"
30     tput cup 11 20 ; echo "2: $BOLD|REPORTS $NORMAL| Menu"
31     error_flag=0         # reset error flag
32   fi
33   tput cup 13 10 ; echo "Enter your choice >_ \b \c"
34   read choice            # read user choice
35 #
36 # case construct for checking the user selection
37 #
38   case $choice in        # check user input
39     0 ) tput clear ; exit 0 ;;
40     1 ) EDIT ;;          # call EDIT program
41     2 ) REPORTS ;;      # call REPORT program
42     * ) ERROR 20 10     # call ERROR program
43     tput cut 20 1 ; tput ed # clear the reset of the screen
44     error_flag=1 ;;     # set error flag to indicate error
45     esac                 # end of the case construct
46 done                    # end of the while construct

```

图 12.10 ULIB 程序代码

1~6行:以#开始,是文件的说明。

7行:把粗体显示的终端代码保存在变量 BOLD 中。

8 行:把正常显示的终端代码保存在变量 `NORMAL` 中。

9 行:使输出变量 `BOLD` 和 `NORMAL` 在子 shell 程序中也可以使用。

10~12 行:以 `#` 开始,是文件的说明。

13 行:清屏。

14 行:置光标于 5 行 15 列。

15 行:屏幕上显示消息 **Super Duper UNIX Library**。`BOLD` 变量使得消息以粗体显示。

16 行:置光标于 12 行 10 列。

17 行:在屏幕上显示其余的消息 **This is the UNIX library application**。`NORMAL` 变量取消了终端的粗体模式,以正常方式显示消息。

【注意】

当终端模式设置后,直到被撤消,模式一直有效。因此,如果使用了命令 `tput smso`,终端显示粗体文本。直到用户输入命令 `tput rmso`,才撤消粗体模式。

18 行:此行有两个命令。`tput` 命令设置光标位置为 14 行 10 列,`echo` 命令显示提示消息。

19 行:等待键盘输入。当按下下一个键后,程序继续。

现在暂停逐行解释,可以运行程序,看看这一部分(1~19 行)的输出。当输入了如下命令时,ULIB 程序显示初始屏幕,等待用户输入。

```
$ ULIB [Return]
```

图 12.11 描绘了开始屏。在显示初始屏幕之后,程序显示主菜单,等待用户从菜单中指定选择。根据用户选择,本程序激活其他程序。菜单一直显示,直到用户选择了退出功能。

```
Super Duper UNIX Library
This is the UNIX library Application
Please enter any key to contnue...> _
```

图 12.11 ULIB 程序初始屏幕

20 行:把 `error_flag` 初始化为 0,说明现在还没有错误。这个标志的目的是指示用户是否有错误输入。根据这个标志的取值,程序显示整个菜单或者需要的部分。这样可以避免每次用户犯错时都清屏而使得用户不知所措。

21、22 和 46 行:这几行实现了 `while` 循环。`while` 循环条件设置为真。条件真总是成立的,因此,`while` 一直循环,循环体永远执行。在选择退出功能前,它会不断显示主菜单。

23、24 和 33 行:这几行实现 `if` 结构。`if` 条件检查 `error_flag` 的状态。如果 `error_flag` 为 0,说明没有错误,条件为真,因此 `if` 体被执行。如果 `error_flag` 不为 0,`if` 条件为假,跳过 `if` 体。

26 行:清屏显示主菜单。注意,初始屏幕在主菜单显示之前显示。

28~31 行:显示主菜单。

32 行:把 `error_flag` 复位,为下次循环做初始化。

33 和 34 行:显示提示文字,等待用户选择。

继续运行程序。图 12.12 显示主菜单,提示符出现,程序等待用户输入。

```

UNIX Library - MAIN MENU
0: EXIT this program
1: EDIT menu
2: REPORTS Menu

Enter your choice> _

```

图 12.12 ULIB 主菜单屏幕

【说明】

屏幕上的粗体文字是因为显示了(`echo` 命令)BOLD 变量中保存的代码。

输入保存在 `choice` 变量里, `choice` 传递给 `case` 结构, 根据输入确定下面的动作。

35 ~ 37 行: 以 # 开始, 是文件的说明。

38 ~ 45 行: 实现 `case` 结构。

如果选择 0: `choice` 变量值为 0; 匹配 0 模式, 执行相关的命令。这里, 清屏并退出程序。主菜单的 0 选项通常是结束程序。

如果选择 1: `choice` 变量值为 1; 匹配 1 模式, 执行相关的命令。这里, 执行名为 EDIT 的程序。当 EDIT 程序终止时, 控制权回到这个程序, 程序接着执行下面的语句。45 行指示 `case` 结构的结束。46 行指示 `while` 循环的结束; 因此, 它回到 21 行检查 `while` 循环条件。条件为真; 因此, 执行循环体, 再次显示主菜单。直到用户在主菜单输入 0, 这一过程不断重复。

如果选择 2: `choice` 变量值为 2; 匹配 2 模式, 执行相关的命令。这里, 执行名为 REPORTS 的程序。与选择 1 时的解释相同, 当执行完 REPORTS 程序后, 控制权回到程序菜单, 显示主菜单, 等待用户的下一次输入。

如果输入错误: 用户的选择是 0、1 或者 2 以外的错误值, `choice` 变量值匹配默认模式(星号), 执行与之相关的命令。这里, 执行程序 ERROR。ERROR 程序在屏幕的指定行列显示错误信息。行列值可以在命令行设定。因此, **ERROR 20 10** 意味着在 20 行 10 列显示错误信息。和上边的解释相同, 当 ERROR 程序结束后, 控制权返回菜单程序。

43 行: 由两个 `tput` 命令组成。`tput cup 20 1` 把光标置于 20 行 1 列, `tput ed` 清除从光标位置到屏幕底部的显示。这里, 20 到 24 行被清除。这样擦除了错误消息, 但是主菜单仍然保持在屏幕上。

44 行: 设置 `error_flag` 为 1, 表明有错误发生。

【说明】

`error_flag` 在 `if` 条件中被检查。因为它的值是 1, `if` 条件失败, `if` 程序体被跳过。这意味着主菜单文本已经在屏幕上了, 不需要重新显示。

图 12.13 显示用户进行了错误选择时的屏幕。错误消息和提示文字是 ERROR 程序的输出。用户按任意键继续执行程序, 错误消息擦除, 提示请求从菜单进行选择(如图 12.12 所示)。

```

UNIX Liabary - MAIN MENU
0:EXIT this program
1:EDIT menu
3:REPORTS Menu

Enter your choice>_6
Wrong Input. Try again
Press any dey to continue...>_

```

图 12.13 ULIB 程序的错误信息屏

12.5.2 ERROR 程序

ERROR 程序显示固定的错误消息,从命令行接受指定光标位置的值。图 12.14 显示这段程序的代码。

```

1 #
2 # ERROR: This program displays an error message and waits for user
3 #           input to continue. It displays the message at the
4 #           specified row and column.
5 #
6 # tput cup $1 $2           # place the cursor on the screen
7 # echo "Wrong Input. Try again." # show the error message
8 # echo "Press any key to continue...> \b \c" # display the prompt
9 # read answer             # read user input
10 # exit 0                 # indicate normal exit

```

图 12.14 ERROR 程序代码

6行:在屏幕的指定位置设置光标。行列值被保存在两个位置变量 \$1 和 \$2 中。这些参数包含从命令行传递给程序的前两个参数。因此,如果输入:

```
$ ERROR 10 15 [Return]
```

位置变量 \$1 和 \$2 分别包含值 10 和 15,错误信息显示在 10 行 15 列。如果用户输入错误,ULIB 程序调用 ERROR 程序,在 20 行 10 列显示错误信息。

7行:显示错误信息。

8行:显示提示。

9行:读用户输入。用户输入不进行错误检查,按任何键都满足 read 命令。因此,用户可以控制程序,按下任意键继续运行程序。

12.5.3 EDIT 程序

从主菜单选择了 1,EDIT 程序被激活。这个程序驱动编辑菜单,与主菜单相似。它显示编辑菜单,根据用户选择激活合适的程序。覆盖整个程序的 while 循环构成这个程序。while 循环体由编辑菜单显示代码和一个 case 语句组成。case 语句决定根据用户选择执行什么命令。其实,这是一般菜单程序的通常逻辑:显示菜单、读取选择,然后根据选择完成动作。

图 12.15 显示 EDIT 程序的代码。这个程序与主程序(ULIB)相似,前边一节详细解释了主程序。图 12.16 显示 EDIT 菜单。当用户从主菜单选择了功能 1 时,显示 EDIT 菜单。

```

1 #
2 # UNIX Library
3 # EDIT:This program is the main driver for the EDIT program.
4 #     It shows the edit menu and invokes the appropriate program
5 #     according to the user selection.
6 #
7 error_flag=0          # initialize the error flag, indicating no error
8 while true            # loop forever
9     do
10        if [ $error_flag-eq 0 ]      # check for the error
11            then
12                tput clear ; tpup cut 5 10      # clear screen and place the cursor
13                echo "UNIX Library - ${BOLD}EDITMENU ${NORMAL}"
14                tput cup 7 20                # place the cursor
15                echo "0: ${BOLD}RETURN ${NORMAL}To the Main Menu"
16                tput cup 9 20 ; echo "1: ${BOLD}ADD ${NORMAL}"
17                tput cup 11 20 ; echo "2: ${BOLD}UPDATESTATUS ${NORMAL}"
18                tput cup 13 20 ; echo "3: ${BOLD}DISPLAY ${NORMAL}"
19                tput cup 15 20 ; echo "4: ${BOLD}DELETE ${NORMAL}"
20            fi
21        error_flag=0          # reset error flag
22        tput cup 17 10 ; echo " Enter your choice>_ \b \c:"
23        read choice          # read user choice
24 #
25 # case construct for checking the user selection
26 #
27 case $choice in
28     0 ) exit 0 ;;          # check user input
29     1 ) ADD ;;            # return to the mian menu
30     2 ) UPDATE ;;        # call UPDATE program
31     3 ) DISPLAY ;;       # call DISPALY program
32     4 ) DELETE ;;        # call DELETE program
33     * ) ERROR 20 10      # call ERROR program
34     tput cup 20 1 ; tput ed # clear the rest of the screen
35     error_flag=1 ;;      # set error flag to indicate
36 esac                    # end of the case construct
37 done                    # end of the while construct

```

图 12.15 EDIT 程序代码

```
UNIX Library - EDIT MENU

0: RETURN To The Main Menu
1: ADD
2: DISPLAY
3: UPDATE STATUS
4: DELETE

Enter your choice > _
```

图 12.16 EDIT 菜单

12.5.4 ADD 程序

当用户从 EDIT 菜单中选择添加功能时,激活这个程序,向库文件增加记录。它提示用户输入信息,把新记录增加到库文件尾,然后请求是否继续添加更多记录。回答 **yes** 继续程序,增加记录。回答 **no** 终止程序,控制权回到调用它的程序,这里是 EDIT 程序。这样,用户回到 EDIT 菜单,等待下次选择。程序假设记录所有 UNIX 图书信息的库文件名为 `ULIB_FILE`;它也假设在 `ULIB_FILE` 中每本书按照记录格式保存下面的信息。每一项作为记录的一个字段保存。用一本教科书作为例子,下面列出各项字段名。

- 书名:UNIX Unbound

- 作者:Afzal Amir

- 种类:教科书

假设有三种有效的类别:

- 系统书:简写为 `sys`

- 参考书:简写为 `ref`

- 教科书:简写为 `tb`

- 状态:`in`

状态表明书被借出还是在库中。图书状态由程序决定,当增加一本书时,状态字段自动设成 `in`。当表明有人借走该书时,状态改成 `out`。

- 借阅者姓名:

指定的书在库中时,这个字段保持为空。如果状态字段为 `out`(借出),该字段设置成借阅者的姓名。

- 日期:

状态字段(`in`)显示指定的书在库中时,该字段为空。如果状态字段为 `out`(借出),该字段设置成借出的时间。

【说明】

1. 一本书的记录第一次被添加到 `ULIB_FILE` 时,状态字段被设置成 `in`,借阅者姓名和日期字段都为空。

2. 接下来,当有人借书时,选择主菜单中的更新功能更新库。然后,用户被请求输入借书者的姓名。程序把状态字段改成 `out`(借出),日期字段设置为当前时间。

图 12.17 显示程序代码,下面逐行解释。

```

1 #
2 # UNIX library
3 # ADD: This program adds a record to the library file (U_LIB). It asks for the
4 #     title ,author, and category of the book. After adding the information
5 #     to the ULIB_FILE file, it prompts the user for the next record.
6 #
7 answer = y                # initialize the answer to indicate yes
8 while ["$answer" = y]    # as long as the answer is yes
9 do
10  tput clear
11  tput cup 5 10 ; echo "UNIX Library - ${BOLD}ADDMODE"
12  tput cup 7 23 ; echo "Title:"
13  tput cup 9 22 ; echo "Author:"
14  tput cup 11 20 ; echo "Category:"
15  tput cup 12 20 ; echo "sys; system, ref; reference, tb; textbook"
16  echo "${NORMAL}"
17  tput cup 7 30 ; read title
18  tput cup 9 30 ; read author
19  tput cup 11 30 ; read category
20  status = in             # set the status to indicate book is in
21  echo "$title: $author: $category: $status: $bname: $date" >> ULIB_FILE
22  tput cup 14 10 ; echo "Any more to add? (Y)es or (N)o>_ \ b \ c"
23  read answer
24  case $answer in        # check user answer
25      [Yy]* ) answer = y ;;      # any word starting with Y or y is yes
26      * ) answer = n ;;         # any other word indicates no
27  esac                   # end of the case construct
28 done

```

图 12.17 ADD 程序代码

1~6行:以#开始,是文件的说明。

7行:用字母y初始化answer变量,表示yes。只要这个变量表示yes,while循环继续执行,每次重复向ULIB_FILE中增加一个记录。

8,9和28行:实现while循环。循环条件为真(这里answer保存着字符y)时,循环体一直执行。

10行:清屏。

11行:在指定光标位置显示屏幕标题。

12~14行:指定光标位置显示提示信息。

15行:显示帮助信息,解释用户应该在图书种类字段填写什么缩写。

16~19行:把光标置于指定的位置(跟在提示信息后),读取用户输入,存储在合适的变量中。

20行:设置状态变量为in,表示书在库中。

到这里为止,所有必需的信息都已经获得,下一步是保存在ULIB_FILE中。如果从EDIT菜单中选择ADD,ADD程序被激活。图12.18显示ADD程序产生的屏幕。

```

UNIX Library - ADD MODE

Title:
Author:
Category:
sys: system,ref: reference,tb:textbook

```

图 12.18 ADD 程序屏幕格式

光标置于书名字段。用户输入书名,然后按回车键。光标移动到作者字段。当用户输入了最后一个字段时,程序保存信息并问用户是否添加另一个记录。以这本书作为例子,图 12.19 显示了添加信息的屏幕,记录已经都被保存。

```

UNIX Library - ADD MODE

Title: UNIX Unbounded
Author: Afzal Amir
Category: tb
sys: system,ref: referencce,tb: textbook

Any more to add? (Y)es or (N)o> _

```

图 12.19 添加信息的 ADD 屏幕示例

21 行:把信息保存到 ULIB_FILE 文件。在 ADD 程序执行到这一行之前,变量得到如下值:

- title 变量包含 UNIX Unbounded
- author 变量包含 Afzal Amir
- category 变量包含 tb
- status 变量包含 in
- bname 变量为空
- date 变量为空

默认情况下,echo 命令把输出显示在屏幕上。但是,在 21 行,echo 命令的输出重定向到了 ULIB_FILE 文件。因此,指定变量里的信息被保存到了 ULIB_FILE 文件中。

【说明】

1. 增加一个记录的时候,变量 bname(借阅者姓名)和 date(借出日期)都为空。
2. 添加符号(>>)用来把信息写到 ULIB_FILE 中。每次添加一个记录的时候,内容被附加到文件末尾,文件的其他记录保持不变。

分隔符

每个记录的各个字段按顺序保存。但是,当用户想要读取和显示一个记录时,这种保存方法难以获取每个字段。必须确定字段分隔符,用选定的字符隔开每个字段。选择 UNIX 用来

作为字段分隔符的键([Tab]、[Return]和空格键)不是好的选择。需要注意,书名和作者名都可能包含空格。如果选择空格键作为分隔符,在作者的姓和名之间带有的空格会使作者名被认为是一个字段,尽管用户仅仅希望程序把它们认为是一个字段。

用户可以选择 ~ 或者 ^ 符号作为分隔键。在 ULIB 程序中,“:”用来做分隔符。

ULIB_FILE 是文本文件,可以用 `cat` 命令显示其内容,或者用 `vi` 编辑器更改。如果刚才的记录是文件中惟一的记录,ULIB_FILE 文件的内容看起来如图 12.20。

```
$ cat ULIB_FILE file
    UNIX Unbounded:Afzal Amir:tb:in::
$ _
```

图 12.20 ULIB_FILE 文件内容

【说明】

1. 文件的每条记录是一行,每个字段用冒号分隔。

2. 这一行结尾的两个冒号是变量 `bname` 和 `date` 的占位符,如果这两个变量不空的话,其内容会包含在冒号之间。

22 和 23 行:`echo` 命令显示提示符,`read` 命令等待用户的回答输入,结果保存在 `answer` 中。

23 ~ 27 行:实现 `case` 结构。`case` 语句检查用户的回答是否和 `case` 的模式匹配。

第一个模式是 `[Yy]*`,与所有以 `Y` 或者 `y` 开头的字符串相匹配。用户可以输入任何以字母 `y` 开头的单词,程序都解释成 `yes` 回答。当 `answer` 被设置成 `y` 时,循环条件(第 8 行)为真,程序继续。

第二个模式是 `[*]`,与所有不以 `Y` 或者 `y` 开头的字符串相匹配,程序解释成 `no` 回答。当 `answer` 被设置成 `n` 时,循环条件(第 8 行)为假,程序终止。

12.5.5 获取记录

该图书库应用程序的其余程序中,都包含了显示现存记录的代码。为了显示记录,必须指定哪些记录需要显示。例如,如果指定作者名,程序检查库文件中是否有指定的作者名。如果发现了,读取记录并显示。如果没有发现记录,显示错误信息。

12.5.6 DISPLAY 程序

DISPLAY 程序把 ULIB_FILE 库文件中的指定记录显示在屏幕上。当用户从 EDIT 菜单选择显示功能时,激活该程序。程序先提示用户输入书名或者作者名。然后从 ULIB_FILE 中查找指定的书。如果发现了记录,用合适的格式显示。如果没有发现记录,显示错误信息并提示下一次输入。这个过程直到用户退出该程序时才终止。当 DISPLAY 程序结束时,控制权回到调用它的程序,这里是 EDIT 程序,用户回到 EDIT 菜单进行下一次选择。

图 12.21 显示了 DISPLAY 程序代码。下面逐行解释该程序。

```
1 #
2 # UNIX library
```

```

3 # DISPLAY: This program displays a specified record from the ULIB_FILE.
4 #       It asks the Author/Title of the book, and displays the specified
5 #       book is not found in the file.
6 #
7 old_IFS="$IFS"           # save the IFS settings
8 answer = y              # initialize the answer to indicate yes
9 while [ "$answer" = y ]  # as long as the answer is yes
10 do
11  tput clear ; tput cup 3 5 ; echo "Enter the Title/Author >_ \b \c"
12  read response
13  grep -i "$response" ULIB_FILE > TEMP # find the specified book in the library
14  if [ -s TEMP ]          # if it is found
15  then
16      IFS=":"             # set the IFS to colon
17      read title author category status bname date < TEMP
18      tput cup = 10
19      echo "UNIX Library - ${BOLD} | DISPLAYMODE ${NORMAL}|"
20      tput cup 7 23 ; echo "Title: $title"
21      tput cup 8 22 ; echo "Author: $author"
22      case $category in   # check the category
23          [Tt][Bb] ) word = textbook ;;
24          [Ss][Yy][Ss] ) word = system ;;
25          [Rr][Ee][Ff] ) word = reference ;;
26          * ) word = undefined ;;
27      esac
28      tput cup 9 20 ; echo "Category: $word" # display the category
29      tput cup 10 22 ; echo "Status: $status" # display the status
30      if [ "$status" = "out" ]             # if it is checked out
31      then                                  # then show the rest of the information
32          tput cup 11 14 ; echo "Checked out by: $bname"
33          tput cup 12 24 ; echo "Date: $date"
34      fi
35  else                                     # if book not found
36      tput cup 7 10 ; echo "$response not found"
37  fi
38  tput cup 15 10 ; ECHO "Any more to look? (Y)es or (N)o >_ \b \c"
39  read answer                             # read user's answer
40  case $answer in                          # check user's answer
41      [Yy]* ) answer = y ;;                # any word starting with Y or y is yes
42      * ) answer = n ;;                    # any other word indicates no
43  esac
44 done                                     # end of the while loop
45 IFS="$OLD_IFS"                           # restore the IFS to its original value
46 exit 0                                    # exit

```

图 12.21 DISPLAY 程序代码

1~6行:以#开始,程序的说明文档。

7行:把环境变量 IFS 的值复制到 OLD_IFS 中。在分配新的值之前,要保存原来的值。以后可以把初始设置恢复到变量 IFS 中。

8 行:用字母 `y` 初始化变量 `answer`。于是,可以进入 `while` 循环。

9、10 和 45 行:`while` 循环。循环体在 10 行和 45 行之间。只要 `while` 循环条件为真(`answer` 等于字母 `y`),就执行循环体。

11 行:该行由三个命令组成。清屏,置光标于 5 行 6 列,然后显示提示信息让用户输入书名或者作者。

12 行:等待用户输入,将输入保存在变量 `response` 中。

13 行:用 `grep` 命令(见第 8 章)从文件 `ULIB_FILE` 中寻找包含变量 `response` 所保存的模式的所有行,输出重定向到 `TEMP` 文件中。如果 `TEMP` 文件不空,表明存在指定书的记录;如果 `TEMP` 为空,说明没找到指定的书。

14、15、35 和 37 行:`if-then-else` 结构。`if` 条件测试 `TEMP` 文件长度是否非 0。如果条件为真(`TEMP` 存在,其中有内容),执行 `if` 体(16~18 行)。如果条件为假(`TEMP` 是空文件),执行 `else` 体(36 行)。

16 行:设置分隔符为冒号(:)。这个程序里,冒号是分隔记录的字段的符号。

17 行:从 `TEMP` 中读取指定书的每个字段。

18 和 19 行:把光标置于指定位置,显示记录头。

20 行:该行由两个命令组成。置光标于 7 行 23 列,然后显示书名。`title` 变量保存着从 `TEMP` 读出来的书名。

21 行:该行由两个命令组成。置光标于 8 行 22 列,然后显示作者名。`author` 变量保存着从 `TEMP` 读出来的作者名。

22~25 行:`case` 结构。`case` 语句检查书的种类(保存在 `category` 变量里),把种类简写改成完整的单词。例如,把种类的简写 `sys` 改成 `system`。

模式 `[Tt][Bb]` 与表示种类缩写 `tb` 的任何大小写字母的组合相匹配。字符串 `textbook` 存在 `word` 变量中。

模式 `[Ss][Yy][Ss]` 与表示种类缩写 `sys` 的任何大小写字母的组合相匹配。字符串 `system` 存在 `word` 变量中。

模式 `[Rr][Ee][Ff]` 与表示种类缩写 `ref` 的任何大小写字母的组合相匹配。字符串 `Reference` 存在 `word` 变量中。

如果不是以上类别,匹配默认情况(`*`),字符串 `undefined` 存在 `word` 变量中。

28 行:置光标于 9 行 20 列,显示指定书的种类(`word` 的内容)。

29 行:置光标于 10 行 22 列,显示指定书的状态(`status` 的内容)。

30 和 34 行:实现 `if-then` 结构。如果 `if` 条件为真(书被借出),执行 `if` 体(31~33 行)。否则,状态显示书在库中,`if` 体被跳过。

32 行:置光标于 11 行 14 列,显示借书者姓名(`bname` 的内容)。

33 行:置光标于 12 行 24 列,显示借书时间(`date` 的内容)。

36 行:当 `TEMP` 为空,执行该行。置光标于 7 行 10 列,显示错误信息。

和 `ADD` 程序类似,余下的代码检查用户是否还要显示其他记录或者返回 `EDIT` 菜单。

【实例】

如果运行 ULIB 程序,选择 EDIT 菜单的 DISPLAY 功能,显示提示符。图 12.22 显示此时的屏幕。

```
Enter the Author/Title>_
```

图 12.22 DISPLAY 程序提示屏幕

用户可以输入书名或者作者名,显示想要的记录。假如输入了 **UNIX Unbounded**,图 12.23 为指定记录的显示屏幕。该屏幕和图 12.19 增加记录的屏幕类似。程序提问用户是否继续查看记录还是退出 DISPLAY 程序。当程序结束时,控制权回到 EDIT 程序,重新显示 EDIT 菜单,继续选择。

```
UNIX Library -DISPLAY MODE

    Title : UNIX Unbounded
    Author : Afzal Amir
    Category : texbook
    Status : in

Any more to look for? (Y)es or (N)o>_
```

图 12.23 DISPLAY 程序显示结果

如果没有发现指定的记录,显示错误消息。例如,如果没有名为 XYZ 的书,图 12.24 显示了错误消息和提示符。

```
Enter the Author/Title>XYZ

XYZ not found

Any more to look for? (Y)es or (N)o>_
```

图 12.24 错误信息的显示屏幕

12.5.7 UPDATE 程序

UPDATE 程序更改 ULIB_FILE 中指定记录的状态。当用户从 EDIT 菜单中选择更新状态功能时,激活该程序。当借出一本书,或者归还借出的书时,选择该项。首先,UPDATE 程序提示用户输入想要修改状态的的书的书名或者作者。然后,从 ULIB_FILE 中查找指定的内容。如果发现了记录,以合适的格式显示。如果没有发现记录,显示错误信息,提示用户下一次输入。直到用户表明要退出程序,该过程一直继续。当 UPDATE 程序结束时,控制权回到调用的程序,即 EDIT 程序,用户回到编辑菜单。

图 12.25 显示 UPDATE 程序的源代码,下面逐行解释。UPDATE 程序的代码比前边的程序

都长,但是多数代码都是从 DISPLAY 复制过来的。DISPLAY、UPDATE 和 DELETE 程序的开头和结尾都是类似的。这些程序提示用户指定作者或者书名,程序找到记录或者没有找到记录显示错误消息。有程序结尾,提示用户继续该程序还是结束。因此,这里只解释新增加或者改变的那些行。

```

1 #
2 # UNIX library
3 # UPDATE; This program updates the status of a specified book. It asks for the
4 #     Author/Title of the book, and changes the status of the specified
5 #     book from in (checked in) to out (checked out), or from out to in.
6 #     If the book is not found in the file, an error message is displayed.
7 #
8 OLD_IFS="$IFS"           # save the IFS settings
9 answer=y                 # initialize the answer to indicate yes
10 while ["$answer" = y]   # as long as the answer is yes
11 do
12 new_status = ; new_bname = ; new_date =      # declare empty variables
13 tput clear                               # clear screen
14 tput clear ; tput cup 3 5 ; echo "Enter the Title/Author>_ \b \c"
15 read response
16 grep -i "$response" ULIB_FILE > TEMP        # find the specified book
17 if [-s TEMP]                               # if it is found
18 then                                       # then
19   IFS=":"                                  # set the IFS to colon
20   read title author category status bname date < TEMP
21   tput cup 5 10
22   echo "UNIX Library - ${BOLD}UPDATESTATUSMODE${NORMAL}:"
23   tput cup 7 23 ; echo "Title: $title"
24   tput cup 8 22 ; echo "Author: $author"
25   case $category in                       # check the category
26     [Tt][Bb] ) word = textbook ;;
27     [Ss][Yy][Ss] ) word = system ;;
28     [Rr][Ee][Ff] ) word = reference ;;
29     * ) word = undefined ;;
30   esac
31   tput cup 9 20 ; echo "Category: $word"     # display the category
32   tput cup 10 22 ; echo "Status: $status"    # display the status
33   if ["$status" = "in"]                    # if it is checked in
34   then                                       # then show the rest of the information
35     new_status = out                        # indicate the new status
36     tput cup 11 18 ; echo "New status: $new_status"
37     tput cup 12 14 ; echo "Checked out by:_ \b \c"
38     read new_bname
39     new_date=`date` + %D
40     tput cup 13 24 ; echo "Date: $new_date"
41   else
42     new_status = in
43     tput cup 11 14 ; echo "Checked out by: "$ bname"

```

```

44     tput cup 12 24 ; echo "Date: $ date"
45     tput cup 15 18 ; echo "New status: $ new_status"
46 fi
47 grep -iv "$ title: $ author: $ category: $ status: $ bname: $ date" \
48 ULIB_FILE > \TEMP
49 my TEMPULIB FILE
50 echo "$ title: $ author: $ category: > > ULIB_FILE"
51 else                                     # if book not found
52 tput cup 7 10 ; echo "$ response not found"
53 fi
54 tput cup 16 10; echo "Any more to Update? (Y)es or (N)o > _ \ b \ c"
55 read naswer                               # read user answer
56 case $ answer in                          # check user answer
57 [Yy]* ) answer = y ;;                    # any word starting with Y or y is yes
58 * ) answer = n ;;                       # any other word indicates no
59 esac
60 done                                       # end of the while loop
61 IFS = "$ OLD_IFS"                        # restore the IFS to its original value
62 exit 0

```

图 12.25 UPDATE 程序代码

12 行:声明三个空变量。当书的状态从 **in** 改变到 **out**(出借)时,这些变量保存新的值;当书的状态从 **out** 改变到 **in**(归还)时,它们被初始化成空。

- **new_bname** 保存书名,或为空
- **new_status** 保存新状态(**in** 或者 **out**)
- **new_date** 保存现在的时间,或为空

33~45 行:实现 **if-then-else** 结构。如果 **if** 条件为真(**status** 变量保存着 **in**,表明书在库中),那么执行 **if** 体(35~39 行)。如果 **if** 条件为假(**status** 变量保存着 **out**,表明书被借出),执行 **else** 体(41~44 行)。

35~39 行:包含 **if** 体,把记录状态从 **in** 改成 **out**。惟一需要的信息是借阅者的姓名。

35 行:在 **new_status** 中保存单词 **out**,表明书被借出。

36 行:置光标于指定位置,显示新状态。这里,显示 **out**。

37 行:提示用户输入借阅者姓名。

38 行:读用户输入,保存在变量 **new_bname** 中。

39 行:把当前日期保存在 **new_date** 中,参数 **%D** 说明时间字符串中只记日期,不记时间(参见第 11 章)。

42~45 行:**else** 体,把记录状态从 **out** 改成 **in**。

42 行:在 **new_status** 中保存单词 **in**,表明书已归还。

43 行:置光标于指定位置,显示 **bname** 的内容借阅者姓名。

44 行:置光标于指定位置,显示 **date** 的内容借出时间。

45 行:置光标于指定位置,显示 **new_status** 内容,这里是单词 **in**,表明书已归还入库。

47 行:除了正在显示的指定记录,把 **ULIB_FILE** 文件中的每个记录复制到 **TEMP** 中。用带 **!** 和

v 参数的 **grep** 命令(见第 8 章)实现该功能,然后把 **grep** 的输出从屏幕重定向到 **TEMP** 文件。参数 i 使 **grep** 命令忽略大小写差别。参数 v 使 **grep** 命令挑出与指定模式不匹配的所有行。

48 行:用 **mv** 命令(见第 5 章)把 **TEMP** 文件重命名成 **ULIB_FILE**。现在 **ULIB_FILE** 文件包含除了状态改变记录之外的所有记录。

49 行:这一行把修改过的记录附加到 **ULIB_FILE** 中。现在, **ULIB_FILE** 包含更新状态的指定记录。

【实例】

继续运行这个程序。终端的显示有助于用户把前边的解释与实际输出联系起来。

假设从 **EDIT** 菜单中选择了更新状态,指定的书是《UNIX Unbounded》。UPDATE 程序的显示如图 12.26。

```
UNIX Library - UPDATE STATUS MODE

      Title: UNIX Unbounded
      Author: Afzal Amir
      Category: textbook
      Status: in
      New status: out
      Checked out by: Steve Fraser
      Date: 12/12/99

Any more to update? (Y)es or (N)o>_
```

图 12.26 UPDATE 程序的显示

【说明】

1. 通过显示旧状态和新状态,在屏幕上反映了状态的改变。
2. 系统提示用户输入借阅者的姓名。假设用户在提示符 **Checked out by: _**后输入 **Steve Fraser**。

图 12.27 显示了在状态改变后的 **ULIB_FILE**。如果与图 12.20 进行比较,可以注意到几个字段的改变。

```
$ cat ULIB_FILE file
  UNIX Unbounded:Afzal Amir:tb:out:Steve Fraser:12/12/99
$_
```

图 12.27 ULIB_FILE 文件的内容

图 12.28 为程序 **DISPLAY** 产生的屏幕显示。

```
UNIX Library - UPDATE STATUS MODE

      Title: UNIX Unbounded
      Author: Afzal Amir
```

```

Category: textbook
Status: in
New status: out
Checked out by: Steve Fraser
Date: 12/12/99
Any more to update? (Y)es or (N)o >

```

图 12.28 显示记录的例子

【说明】

比较本图(status 显示已经出借)和图 12.23 的显示(status 显示在库中)。

12.5.8 DELETE 程序

DELETE 程序从 ULIB _ FILE 文件中删除指定的记录,当用户从 EDIT 菜单选择删除功能时,激活该程序。程序首先提示用户输入书名或作者名。然后,在 ULIB _ FILE 文件中查找与指定书相匹配的记录。如果发现了记录,用合适的方式显示记录,并显示确认提示。用户可以确定是否真的想要删除该记录。如果没有发现记录,显示错误信息,提示下一次输入。该过程一直运行到用户准备退出。当 DELETE 程序结束时,控制权回到调用的程序 EDIT,用户回到 EDIT 菜单,准备新的选择。

图 12.29 显示 DELETE 程序的代码。程序的长度应该不是问题,因为 DELETE 的开头结尾部分与 DISPLAY 和 UPDATE 程序的基本相同。主要差别在于删除记录前的确认代码和从 ULIB _ FILE 文件中实际删除指定记录。

```

1 #
2 # UNIX library
3 # Delete: This program deletes a specified record from the ULIB _ FILE.
4 #     It asks for the Author/Title of the book, and displays the specified
5 #     book, and deletes it after confirmation, or it shows an error
6 #     message if the book is not found in the file.
7 #
8 OLD_IFS = "$IFS"           # save the IFS setting
9 answer = y                 # initialize the answer to indicate yes
10 while ["$answer" = y]    # as long as the answer is yes
11 do
12     tput clear ; tput cup 3 5 ; echo "Enter the Title/Author >_ \b \c"
13     read response
14     grep -i "$response" ULIB _ FILE > TEMP # find the specified book in the library
15     if [ -s TEMP]         # if it is found
16     then                 # then
17         IFS = ":"        # set the IFS colon
18         read title author category status bname date < TEMP
19         tput cup 5 10
20         echo "UNIX Library - ${BOLD} DELETEMODE ${NORMAL}"
21         tput cup 7 23 ; echo "Title: $title"

```

```

22  tput cup 8 22 ; echo "Author: $ author"
23  case $ category in                # check the category
24      [Tt][Bb] ) word = textbook ;;
25      [Ss][Yy][Ss] ) word = system ;;
26      [Rr][Ee][Ff] ) word = reference ;;
27      * ) word = undefined ;;
28  esac
29  tput cup 9 20 ; echo "Category: $ word"      # display the category
30  tput cup 10 22 ; echo "Status: $ status"     # display the status
31  if [ "$ status" = "out" ]                # if it is checked out
32  then                                     # then show the rest of the information
33      tput cup 11 14 ; echo "Checked out by: $ bname"
34      tput cup 12 24 ; echo "Date: $ date"
35  fi
36  tput cup 9 20 ; echo "Delete this book? (Y)es or (N)o>_\b\c"
37  read answer
38  if [ $ answer = y -o $ answer = Y ]      # test for Y or y
39  then
40      grep -iv "$ title: $ author: $ category: $ status: $ bname: $ date"
41      ULIB_FILE > TEMP \
42      mv TEMP ULIB_FILE
43  fi          # if book not found
44  esle
45  tput cup 7 10 ; echo "$ response not found"
46  fi
47  tput cup 15 10 ; echo "Any more to delete? (Y)es or (N)o>_\b\c"
48  read answer                            # read user answer
49  case $ answer in                       # check user answer
50      [Yy]* ) answer = y ;;              # any word starting with Y or y is yes
51      * ) answer = n ;;                  # any other word indicates no
52  esac
53 done                                     # end of the while loop
54 IFS = "$ OLD_ IFS"                      # restore the IFS to its original value
55 exit 0                                    # exit

```

图 12.29 DELETE 程序代码

发现和显示记录以后,出现确认提示。此刻,程序在 36 行。

36 行:置光标于指定行列,显示提示。

37 行:读用户响应,保存在变量 `answer` 中。

38~42 行:实现 `if` 结构。如果 `if` 条件为真,执行 `if` 体(40 和 41 行)。如果 `if` 条件测试为 `Y` 或者 `y`,记录被删除。任何其他字母都使判断结果为假,`if` 体被跳过,不删除记录。

40 行:除了正在显示的指定记录,把 `ULIB_FILE` 文件中的每个记录复制到 `TEMP` 中。用带 `i` 和 `v` 参数的 `grep` 命令实现该功能,然后把 `grep` 的输出从屏幕重定向到 `TEMP` 文件。

41 行:用 `mv` 命令把 `TEMP` 文件重命名成 `ULIB_FILE`。现在 `ULIB_FILE` 文件包含除了被删除的记录之外的所有记录。

假设从 `EDIT` 菜单选择了删除功能,而且 `ULIB_FILE` 中存在指定的书,图 12.30 显示 `DELETE` 程序输出的例子。

```

UNIX Library - DELETE MODE

      Title: UNIX Unbounded
      Author: Afzal Amir
      Category: textbook
      Status: in
      Checked out by: Steve Fraser
      Date: 12/12/99

Delete this book? (Y)es or (N)o > Y

Any more to update? (Y)es or (N)o >

```

图 12.30 DELETE 的显示

12.5.9 REPORTS 程序

层次图(见图 12.9)的另一个分支是通过利用 ULIB_FILE 中的信息生成报表。当用户从主菜单中选择功能 2 时,REPORTS 程序被激活。和 EDIT 程序类似,该程序驱动 REPORTS 菜单。它显示 REPORTS 菜单,根据用户选择激活对应的程序,生成需要的报表。一个 while 循环覆盖了整个程序。while 循环体由 REPORTS 菜单和 case 语句组成,case 语句决定根据用户选择,执行什么命令。

图 12.31 显示了 REPORTS 程序的代码。现在不需要逐行解释了,因为读者已经知道了一切。但是,这里调用了别的程序。程序 REPORT_NO 用来生成要求的报表。每次根据选择,在命令行中设定不同的报表号。报表号(1、2 或 3)保存在位置变量 \$1 中,被程序 REPORT_NO 访问。

```

1 #
2 # UNIX Library
3 # Reports: This program is the main driver for the REPORTS menu.
4 #           It shows the reports menu, and invokes the appropriate program
5 #           according to the user's selection.
6 #
7 error_flag=0           # initialize the error flag, indicating no error
8 while true             # loop forever
9     do
10    if [ $error_flag -eq 0 ]      # check for the error
11    then
12        tput clear ; tput cup 5 10    # clear screen and place the cursor
13        echo "UNIX Library - ${BOLD}REPORTSMENU${NORMAL}"

```

```

14         tput cup 7 20                # place the cursor
15         echo "0: ${BOLD}|RETURN${NORMAL}| To The Main Menu"
16         tput cup 9 20 ; echo "1: Sorted by ${BOLD}|TITLES ${NORMAL}|"
17         tput cup 11 20 ; echo "2: Sorted by ${BOLD}|AUTHOR ${NORMAL}|"
18         tput cup 13 20 ; echo "3: Sorted by ${BOLD}|CATEGORY ${NORMAL}|"
19     fi
20     error_flag=0                      # reset error flag
21     tput cup 17 10 ; echo "Enter your choice>_ \b \c"
22     read choice                        # read user choice
23 #
24 # case construct for checking the user selection
25 #
26     case $ choice in                  # check user input
27         0 ) exit 0 ;;                 # return to the main menu
28         1 ) REPORT_NO 1 ;;           # call REPOT_NO program,passing 1 to it
29         2 ) REPORT_NO 2 ;;           # call REPOT_NO program,passing 2 to it
30         3 ) REPORT_NO 3 ;;           # call REPOT_NO program,passing 3 to it
31         * ) ERROR 20 10              # call ERROR program
32             tput cup 20 1 ; tput ed # clear the rest of the screen
33             error_flag=1;;           # set error flag to indicate error
34     esac                              # end of the case construct
35 done                                  # end of the while construct

```

图 12.31 REPORTS 程序代码

假设从主菜单中选择了报表功能,图 12.32 显示报表菜单。

```

UNIX Library - REPORTS MENU
0:RETURN To The Main Menu
1:Sorted by the TITLE
2:Sorted by the AUTHOR
3:Sorted by the CATEGORY
Enter your choice>_

```

图 12.32 REPORTS 菜单

12.5.10 REPORT_NO 程序

当用户从 REPORTS 菜单选择了一个报表号后,REPORT_NO 程序被激活。这个程序检查位置变量 \$1 的值,根据这个值,对 ULIB_FILE 进行排序。如果 \$1 的值为 1,ULIB_FILE 根据书名字段(记录的第 1 个字段)排序。如果 \$1 的值为 2,ULIB_FILE 根据作者字段(记录的第 2 个字段)排序,等等。

排序的记录保存在文件 TEMP 中,这个文件传递给 pg 或者 pr 命令(见第 8 章)进行显示。但是,如图 12.27,每个记录用冒号分开,这样的报表不够美观。为了使输出更加美观,记录从 TEMP 文件中读出来,经过格式化,然后存储到 PTEMP 文件中。PTEMP 文件再传递给 pg 命令显示。

图 12.33 显示 REPORT_NO 程序的代码,下面逐行解释一些新的代码。

```

1 #
2 # UNIX library
3 # REPORT_NO: This program produces report from the ULIB_FILE file.
4 #           It checks for the report number passed to it on the command
5 #           line, sorts, and produces reports accordingly.
6 #
7 IFS = ;" # set delimiter to;
8 case $1 in # check the contents of the $ 1
9   1) sort -f -d -t:ULIB_FILE > TEMP ;; # sort on title field
10  2) sort -f -d -t:11 ULIB_FILE > TEMP ;; # sort on author field
11  3) sort -f -d -t:12 ULIB_FILE > TEMP ;; # sort on category field
12 esac
13 #
14 # read records from the sorted file TEMP. Format and store them in
15 # PTEMP.
16 #
17 while read title author category status bname date # read a record
18 do
19   echo "Title: $title" >> PTEMP # format title
20   echo "Author: $author" >> PTEMP # format author
21   case $category in # check the category
22     [Tt][Bb]) word = textbook ;;
23     [Ss][Yy][Ss]) word = system ;;
24     [Rr][Ee][Ff]) word = reference ;;
25     *) word = undefined ;;
26   esac
27   echo "   Category: $word" >> PTEMP # format category
28   echo "   Status: $status\n" >> PTEMP # format status
29   if [ "$status" = "out" ] # if it is checked out
30   then # then
31     echo "   Checked out by: $bname" >> PTEMP # format bname
32     echo "   Date: $date\n" >> PTEMP # format date
33   fi
34   echo >> PTEMP
35 done < TEMP
36 #
37 # ready to display the formatted records in the PTEMP
38 pg -c -p "Page %d:" PTEMP # display PTEMP page by page
39 rm TEMP PTEMP # remove files
40 exit 0 # end of the program_

```

图 12.33 REPORT_NO 程序代码

9、10 和 11 行:这几行是 **case** 结构体。每一行执行一个 **sort** 命令(见第 8 章),根据指定字段对 ULIB_FILE 排序。**sort** 的输出保存在 TEMP 中。下面来看看 **sort** 命令的参数。

- **-f** 参数把小写字母都认为是大写字母
- **-d** 参数忽略非字母字符
- 数字参数(+2 和 +3)按指定字段号对文件排序。在 ULIB_FILE 中,字段 1 是 title 字段,字段 2 是 author 字段,字段 3 是 category 字段。这是 REPORT_NO 感兴趣的三个字段

17~33 行:做 **while** 循环把 TEMP 文件中的所有记录都读取出来。这里测试 **read** 的返回状态:只要有记录则返回为 0,到达文件尾部则返回 1。循环体和 DISPLAY 程序相似,但是 **echo** 命令的输出重定向到 PTEMP 文件。

33 行:**while** 循环的结尾, < **TEMP** 重定向,输入从标准输入设备变成了 TEMP 文件。这被称作循环重定向,Bourne shell 在子 shell 中运行重定向循环。当循环结束后,PTEMP 存储了经过排序和格式化的报表待显示。下面用 **pg** 命令显示。

38 行:显示 PTEMP 文件。**pg** 命令一次一屏显示报表,可以用 **pg** 的参数扫描报表。

下面来看 **pg** 命令的参数。

- **-c** 参数在显示之前清屏
- **-p** 参数在屏幕底部用指定的字符串代替默认的冒号提示。这里,指定的字符串是 **Page %d:**。字符串 **%d** 代表当前是第几屏

39 行:删除 TEMP 和 PTEMP 这两个文件。

图 12.34 显示了从 REPORTS 菜单选择第 2 项生成的报表例子。报表根据第 2 个字段 Author 进行了排序。

```

      Title: UNIX unbound
      Author: Afzal Amir
      Category: textbook
      Status: out
      Checked out by: Steve Fraser
      Date: 1/12/99

      Title: UNIX For All
      Author: Brown David
      Category: reference
      Status: in

      Title: A Brief UNIX Guide
      Author: Redd Emma
      Category: reference
      Status: out
      Checked out by: Steve Fraser
      Date: 1/12/00

```

Page 1:

图 12.34 REPORT_NO 生成的报表例子

【说明】

屏幕底部的提示显示页号。用户可以按回车键显示下一页,或者输入其他的 **pg** 动作命令查看需要的部分。

命令小结

下面是本章介绍的命令。

trap

设置和复位终端信号。下面的表显示了一些用来控制程序终止的信号。

信号数	信号名	含义
1	挂起	失去终端联系
2	中断	任一中断键被按下
3	退出	任一退出键被按下
9	杀死	发出 kill -9 命令
15	终止	发出 kill 命令

stty

用来设置和显示终端属性,支持改变超过 100 种不同的设置,下表列出一些参数。

选项	功能
echo [-echo]	回显[不回显]输入字符;默认回显
raw [-raw]	禁止[启用]特殊意义的元字符;默认为启用
intr	生成中断信号;通常用[Del]键
erase	(回退)擦除前一个字符;通常用#键
kill	删除整行;通常用@或者[Ctrl-u]键
eof	从终端产生文件结束(eof)信号;通常用[Ctrl-d]键
ek	用#和@键分别复位 erase 和 kill 键
same	用合理的默认值设置终端属性

tput

该命令与包含着终端特性的 terminfo 数据库相连,方便进行终端属性的设置(如粗体、清屏等)。

选项	功能
bel	回显终端的响铃字符
blink	闪烁显示
bold	粗体显示
clear	清屏
cup r c	把光标移到 r 行 c 列
dim	显示变暗
ed	从光标位置到屏幕底清屏
el	从光标位置到行末清除字符
smsc	启动突出模式

(续表)

选 项	功 能
<code>rmso</code>	结束突出模式
<code>smul</code>	启动下划线模式
<code>rmul</code>	结束下划线模式
<code>rev</code>	反白显示
<code>sgr0</code>	关闭所有属性

习题

1. `trap` 命令的作用是什么?
2. 如何终止进程?
3. `trap` 命令用在哪里?
4. 显示终端设置的命令是什么?
5. `terminfo` 数据库是什么? 里边存储了什么信息?

上机练习

下面的上机练习提供了本章提到的终端命令和更改程序的练习。

1. 你的终端类型是什么?
2. 显示终端设置的部分列表。
3. 显示终端设置的完全列表。
4. 改变 `kill` 键的设置。检查是否确实改变了。
5. 改变 `erase` 键的设置。检查是否确实改变了。
6. 分别把 `erase` 和 `kill` 键恢复成 # 和 @。
7. 检查系统是否有 `tput` 工具。
8. 用 `tput` 命令清屏。
9. 写一个名为 `CLS` 的脚本文件用来清屏。
10. 编写一个与 `MENU` 程序相似的脚本文件。该菜单用来显示一些命令,比如是用户经常不记得确切语法的命令,或者是懒得输入的很长的命令。
11. 修改本章的 UNIX 书库的程序,以存储更多信息;例如价格、出版日期等。
12. 改进 `REPORTS` 程序。例如,用户可以选择打印或者显示报告。
13. `ULIB` 程序有许多可改进之处。比如,目前不能解决同一个作者有多本书,或者不同作者的书具有相同的书名等问题。如何解决这些问题?
14. UNIX 书库程序可以作为其他类似程序的原型。例如,可以改成保存朋友的姓名、地址和电话号码的通信簿,或者为音乐 CD 和磁带建立一个数据库来管理。请发挥想像力,编写代码实现一个与 UNIX 书库相似的系统工具。

第 13 章 告别 UNIX

前面我们已讨论了 UNIX 系统的基本内容,了解了如何使用 shell 编程,现在可讨论一些 UNIX 系统的独特之处。本章讨论的命令包括磁盘命令、文件管理命令和拼写检查命令。前面章节已讨论过一些文件管理命令,本章的文件管理命令是对以前章节的补充。本章最后还要讨论一些权限控制命令和系统管理命令,以帮助用户深入理解 UNIX 系统,感觉 UNIX 系统的方便之处。

13.1 磁盘空间

磁盘或文件中可保存的文件数目是有限的。这个最大文件数目由以下 2 个因素决定:

- 未用存储空间大小
- 用于保存索引节点的存储空间大小

文件系统中每个文件都有一个索引节点号(在第 7 章中有详细讨论),这些信息保存在索引节点表中。每个索引节点包括一些特定的文件信息,如文件在磁盘上的存储位置、文件大小等。

13.1.1 df 命令:显示未用磁盘空间

df(disk free)命令可得到指定文件系统的磁盘空间总量或未用空间。如果在命令行中没有指定文件系统,**df** 命令会报告所有文件系统的未用空间。

【实例】

显示未用磁盘空间。

```
$ df [Return].....未指定文件系统
/ (/dev/dsk/c0d0s0)          14534 blocks 2965 i-nodes
/usr (/dev/dsk/c0d0s2)      203028 blocks 51007 i-nodes
$_.....命令输入提示
```

命令显示说明系统中有 2 个文件系统,第一个值表示未用磁盘块数目,第二个值表示未用索引节点数目。每个磁盘块通常为 512 字节,一些系统磁盘块大小为 1024 字节。

-t选项:在 **df** 命令中使用 **-t** 选项可在命令显示输出中给出文件系统的磁盘块总数。

【实例】

使用带 **-t** 选项的 **df** 命令。

```
$ df -t [Return].....使用-t 选项
/ (/dev/dsk/c0d0s0)          14534 blocks      2965 i-nod
                             total: 31552 blocks  3936 i-nodes
/usr (/dev/dsk/c0d0s2).     203028 blocks     51007 i-nodes
                             total:539136 blocks  65488 i-nodes
$_.....命令输入提示
```

【说明】

可在用户的 .profile 文件中加入 **df** 命令。这样,用户在每次登录时就可见到磁盘空间使用情况报告。

13.1.2 统计磁盘空间使用情况:du 命令

利用 **du**(disk usage)命令可得到文件系统中每个目录和文件占用的存储块数目。这个命令可方便地获得一个文件系统的存储空间使用情况。

【实例】

查询当前目录及其子目录的存储空间使用情况。

```
$ du [Return]..... 当前目录的存储空间报告
22          ./source/basic
35          ./source/c
26          ./source
122         ./memos
997         .
$_..... 命令输入提示
```

当前目录用“.”表示。这里,./source/basic 文件占用 22 个存储块,./source/c 文件占用 35 个存储块……

【说明】

du 命令可用于查询任何目录的存储空间使用情况。

du 命令选项

表 13.1 列出了 **du** 命令的选项,并给出了每个选项的解释。

表 13.1 du 命令选项

选项	功能
-a	显示目录和文件的大小
-s	只显示指定目录占用的存储块数目,不列出子目录

-a 选项: -a 选项显示指定目录和该目录中每个文件占用的存储空间。

13.2 高级 UNIX 命令

本节介绍一些新命令,帮助用户操作 UNIX 环境、制作大标题、查询程序状态信息等。

13.2.1 banner 命令:显示标题

banner 命令可在标准输出设备上输出大字符,在标准输出上显示它的命令行参数(不多于 10 个字符),每个参数占一行。该命令可用于制作标语、符号、标题等。

【实例】

制作一个生日快乐标语。

```
$banner happy birthday | lp [Return].....生成一个标语,并输出到打印机。
$_.....命令提示符
```

管道命令(|)把输出传送到打印机。每个单词(参数)打印成一行。用引号可把两个或多个单词作为一个参数。

【实例】

制作一个回家标语,两个单词打印在一行上。

```
$banner "GO HOME" | lp [Return]
$_.....命令提示符
```

13.2.2 at 命令:在指定时间执行程序

使用 **at** 命令可在指定的时间开始执行一个或一组程序。当用户希望在机器不繁忙时执行自己的程序或在指定时间发送电子邮件时,该命令十分有用。用户可用多种格式来指定日期和时间,语法十分灵活。

【注意】

在各种 UNIX 系统中,可能会严格限制能使用 **at** 命令的用户。系统管理员可限制只有少数用户能使用 **at** 命令。如果没有权限使用 **at** 命令,系统会在使用时显示如下信息:

```
at: You are not authorized to use at. Sorry. (at: 抱歉,你没有权限使用 at 命令。)
```

【说明】

1. 用户要在命令行中指定希望系统执行用户程序的日期和时间。
2. 当系统在指定的时间执行程序时,不需要用户登录系统。

指定时间

当 **at** 命令中的时间参数为 1 个或 2 个数字时(HH 格式),表示整点时间,于是 04 和 4 都表示凌晨 4 点整。如果时间参数为 4 个数字时(HHMM 格式),表示小时和分钟;如 0811 表示 8 点 11 分。用户也可用 noon(中午)、midnight(午夜)或 now(现在)来表示时间。

【说明】

UNIX 系统默认使用 24 进制时钟,除非使用 am(上午)或 pm(下午)后缀。如 2011 表示下午 8 点 11 分。

指定日期

at 命令中的日期参数可为星期,如 Wednesday 或 Wed(星期三);也可为月日年格式,如 Aug 10, 2001(2001 年 8 月 10 日);还可用 today(今天)或 tomorrow(明天)表示日期。下面是 **at** 命令中的一些合法日期和时间格式。

```
$at 1345 wed [Return].....在星期三下午 1 点 45 分执行一个作业
```

```

$at 0145 pm Wed [Return].....在星期三下午 1 点 45 分执行一个作业
$at 0925 am Sep 18 [Return].....在 9 月 18 日上午 9 点 25 分执行一个作业
$at 11:00 pm tomorrow [Return].....在明天晚上 11 点整执行一个作业

```

下面命令序列显示如何使用 `at` 命令和各种指定日期和时间的格式。

【实例】

在明天凌晨 4 点开始执行一个作业。

```

$at 04 tomorrow [Return].....指定日期和时间
sort BIG_FILE [Return].....对文件 BIG_FILE 的内容进行排序
[Ctrl-d].....表示输入结束
user = david.....75969600.a.....Wed Jan 26 14:32:00
$_.....命令行提示

```

`sort` 命令将在明天凌晨 4 点开始执行,作业标识号为(75969600.a)。

`at` 命令将如何处理在指定时间执行用户程序时可能产生的输出信息——如在上面例子中 `sort` 命令的输出信息? 如果没有指定输出文件,系统会通过电子邮件把输出信息传给用户,用户可在自己的邮箱中见到结果。

【实例】

执行一个带输出重定向的命令。

```

$at 04 tomorrow [Return].....指定日期和时间
sort BIG_FILE > BIG_SORT[Return].....输出结果保存在文件 BIG_SORT 中
[Ctrl-d].....表示输入结束
user = david.....75969600.a.....Wed Jan 26 14:32:00
$_.....命令行提示

```

`sort` 命令的输出被重定向到文件 `BIG_SORT`。

【说明】

可利用 `cat`、`vi` 等分页显示命令来查看 `BIG_FILE` 的内容。

【实例】

执行一个带输入重定向的命令。

```

$at noon Wed[Return].....在星期三中午执行
mailx david < memo[Return].....通过电子邮件把 memo 文件传给 david
[Ctrl-d].....表示命令列表结束
user = david.....75969600.a.....Wed Jan 26 14:32:00 2001
$_.....命令行提示再次出现

```

这个命令在星期三中午通过电子邮件把备忘录传送给 `david`。星期三中午前,在用户的当前目录中必须存在一个名为 `memo` 的文件。

【实例】

在星期五下午 3 点 30 分执行一个名为 `cmd_file` 的脚本文件。

```

$at 1530 Fri < cmd_file[Return].....从文件 cmd_file 读入命令
user = david.....75969601.a.....Fri Jan 28 14:32:00 2001
$_.....提示可输入下一个命令

```

at 命令选项:表 13.2 列出了 at 命令的选项。

表 13.2 at 命令选项

选项	功 能
-l	显示所有用 at 命令提交的任务的列表
-m	任务完成时向用户发一个简短的确认信息
-r	从未执行任务队列中删除指定的任务号

【实例】

显示用 at 命令提交的未执行任务列表。

```
$at -l[Return].....列出所有未执行任务
75969601.a.....at Fri Jan 28 14:32:00 2001
$_.....再次出现命令提示符
```

【实例】

从 at 命令的任务队列中删除一个任务。

```
$at -r 75969601.a[Return].....删除指定任务
$at -l[Return].....检查 at 命令的任务队列
$_.....任务队列为空
```

用户必须指定要从队列中删除的任务号。通过在输入多个用空格隔开的任务号,可实现一次删除多个任务。

13.2.3 type 命令:显示命令类型

type 命令可用于查询命令类型。它可判断一个命令是 shell 的内部命令还是外部命令。shell 内部命令是指定该命令是 shell 命令解释程序的一部分,在调用时不产生子进程。

下面是一些 type 命令的应用实例。

```
$type pwd[Return].....查询 pwd 命令的类型
pwd is a shell built-in
$type ls[Return].....查询 ls 命令的类型
ls is /bin/ls
$type cat[Return].....查询 cat 命令的类型
cat is a shell built-in
$_.....命令提示符
```

13.2.4 time 命令:显示命令执行时间

利用 time 命令可得到系统执行用户程序时的时间信息,如实际时间(real time)、用户态时间(user time)和系统态时间(system time)。time 命令的参数为需要计时的命令名。

实际时间是指从用户输入命令行的回车键开始到命令执行完毕的时间。实际时间中包括 I/O 时间、等待时间等等。实际时间可能比命令执行时占用的 CPU 时间长几倍。

用户态时间是指用于执行用户命令中用户态代码的 CPU 时间。

系统态时间是指为了完成用户命令而执行 UNIX 系统例程的时间。
CPU 时间是指 CPU 用于执行用户命令的用户态和系统态时间的总和。

【实例】

查询对文件 BIG_FILE 排序所占用的时间。

```
$time sort BIG_FILE > BIG_FILE.SORT[Return]
60.8 real    11.4 user    4.6 sys
$_.....可输入下一个命令
```

结果显示执行该命令占用的实际时间为 60.8 秒、用户态时间为 11.4 秒、系统态时间为 4.6 秒,共计占用 CPU 时间为 16 秒。

13.2.5 calendar 命令:提醒服务

calendar 命令可提醒用户自己有关约会或做某事的时间。要使用提醒服务,用户必须在主目录或当前目录创建一个日历文件。该命令显示日历文件中今明两天的日程安排。如果设置自动运行 **calendar** 命令,则系统会在相应日期把日程安排用电子邮件通知用户。用户也可在命令提示符 \$ 下直接输入 **calendar** 命令来得到自己的日程安排。

日历文件中的日期可有多种表示格式。图 13.1 是一个日历文件的实例,其中各行的时间格式是不同的。

```
$cat calendar
1/20 Call David
Meet with your advisor on January 22.
1/22 You have a dental appointment.
The BIG MEETING is on Jan. 23.
3/21 Time to clean up your desk.
$_
```

图 13.1 日历文件实例

【实例】

假设当前日期为 1 月 22 日,执行 **calendar** 命令。

```
$calendar[Return].....可输入下一个命令
1/22 You have a dental appointment. ....(1月22日;去看牙医)
Meet with your advisor on January 22. ....(1月22日;与导师讨论)
The BIG MEETING is on Jan.23.....(1月23日;开全体会)
$_.....命令提示符
```

该命令显示日历文件中所有日期为 1 月 22 或 23 日的行。

【说明】

可把 **calendar** 命令放在启动脚本 .profile 中。这样,用户在每次登录时可见到相应的日程安排。

13.2.6 显示详细用户信息:finger 命令

第 3 章介绍的 **who** 命令可显示系统的当前用户信息,而 **finger** 命令可得到当前用户更详

细和广泛的信息。默认时, `finger` 命令按多列格式显示用户信息。图 13.2 给出了一个 `finger` 命令的输出实例。

```
$ finger
Login      Name          TTY          Idle        When        Where
dan        Daniel Knee  * console    5:08        Mon 14:14
gabe       Gabriel Smart p0           1:33        Mon 17:15 205.130.70
emma       Emma Good    p2           8:21        Mon 11:02 205.130.71
$_
```

图 13.2 `finger` 命令

- 登录名(Login): 显示用户的登录名
- 用户全名(Name): 显示用户全名
- 终端名(TTY): 显示用户所用终端的设备名。终端名前的星号(如 * console)表示向该终端发送的消息被阻塞(参见第 9 章 `mesg` 命令)
- 空闲时间(Idle): 显示用户最后一次敲击键盘至今的时间
- 登录时间(When): 显示用户的本次登录时间
- 终端地址(Where): 显示用户所用终端的地址

如果 `finger` 命令后有一个作为用户名的参数, 不论该用户是否正在使用系统, 都显示该用户的相关信息。图 13.3 显示用户 `gabe` 的相关信息。

```
$ finger gabe
Login name: gabe      (message off)   In real life: Gabriel Smart
Directory: /home/students/gabe    shell: /bin/ksh
On since July 30 18:45:38 On p0
Mail last read Tue Jul 23 09:08:06 1997
No Plan.
$_
```

图 13.3 带参数的 `finger` 命令

.plan 和 .project 文件

上面显示信息中有一行奇怪的内容“`No plan`”。`finger` 命令会显示用户主目录中两个隐含文件 `.plan` 和 `.project` 的内容。在上面例子中, “`No plan`”表示在用户 `gabe` 的主目录中没有文件 `.plan`。假设在 `gabe` 的主目录中有文件 `.plan` 和 `.project`, 图 13.4 给出了这时 `finger` 命令的输出。

用户 `gabe` 的主目录中文件 `.project` 的内容为:

```
C++ Application Project (C++ 应用项目)
This project is assigned to the Gold Team. (该项目由 Gold Team 承担。)
For more information about this project contact Gabe Smart. (如需该项目的详细信息,
请与 Gabe Smart 联系。)
```

用户 `gabe` 的主目录中文件 `.plan` 的内容为:

```

Hi,
I am all smiles these days! (微笑每一天。)
Gabe :-)

$finger gabe
Login name: gabe      (message off)      In real life:Gabriel Smart
Directory: /home/students/gabe          shell: /bin/ksh
On since July 30 18:45:38 On p0
Mail last read Tue Jul 23 09:08:06 1997
Project: C11 Application Project
Plan:
Hi,
I am all smiles these days!
Gabe :-)
$_

```

图 13.4 带有文件 .project 和 .plan 内容的 finger 命令输出

【说明】

如果用户主目录 (\$HOME) 中有文件 .project 和 .plan, finger 命令将显示文件 .project 的第 1 行和文件 .plan 的全部内容。

finger 命令选项

finger 命令的选项主要用于控制输出格式。表 13.3 列出了一些输出格式控制选项。

表 13.3 finger 命令的选项

选项	功 能
-b	不在长格式中输出用户主目录和 shell 程序名
-f	不在短格式中输出标题行
-h	不在长格式中输出文件 .project 的内容
-l	强制使用长格式输出
-p	不在长格式中输出文件 .plan 的内容
-s	强制使用短格式输出

【说明】

长格式选项是用类似于图 13.3 中单个用户的信息格式来显示所有用户的信息。

13.2.7 tar 命令:存档和分发文件

用 tar 命令(Tape Archive)可把一组文件复制到一个文件中,称为存档文件(tarfile)。存档文件通常放在磁带上,但也可存在软盘等其他介质上。tar 命令把多个文件打包在一起,以 tar 格式存放在一个存档文件中;tar 格式的存档文件可用 tar 命令来解包,还原成多个文件。例如:可用下面命令行把当前目录中的文件和子目录打包成一个存档文件。

```
$tar -cvf /dev/ctape1 .[Return]
```

在上面例子中,命令行的最后一个参数“.”表示要打包的对象是当前目录;生成的存档文件送至设备“/dev/ctape1”。

tar 命令的选项

表 13.4 列出了 tar 命令的一些常用选项。

表 13.4 tar 命令的选项

选项	功 能
c	(create)创建一个新的存档文件,该文件中原来的内容被覆盖
r	(replace)添加文件到存档文件中,该文件中的原来内容保留
t	(table of contents)列出存档文件中的所有被打包文件的文件名
x	(extract)从存档文件中还原出被打包文件
f	(file)用下一个参数作为存档文件的存放位置
v	(verbose)显示详细打包过程信息

【实例】

下面命令行实例说明了如何利用 tar 命令对一组文件进行存档。

假设当前目录为 projects,其中包括 2 个文件,文件名分别为 head 和 tail。下面命令行把当前目录打包成一个存档文件 project.tar 存放在目录 tarfiles 中。

```
$tar -cvf ./tarfiles/project.tar .[Return].....对当前目录进行存档
```

选项 c 表示创建一个新的存档文件,选项 v 表示在 tar 命令打包的过程中显示详细信息,选项 f 说明下一个参数为生成的存档文件的路径。

```
$tar -cvf ./tarfiles/project.tar .
a ./projects/      0K
a ./projects/head  3K
a ./projects/tail  6K
$_
```

选项 v 提供打包文件的一些附加信息。这些信息分成 3 部分:第一项为表示存档的字母 a,第二项为打包文件的路径,第三项为打包文件的大小。

【说明】

存档文件的名称可为任何有效的 UNIX 文件名。这里的后缀.tar 是为了表示文件的意义,不是必需的。

下面例子是不带 v 选项的 tar 命令输出。注意,显示信息要少一些。

```
$tar -cf ./tarfiles/project.tar .
a ./projects
a ./projects/head
a ./projects/tail
```

\$_

【实例】

下面命令行实例说明了如何利用 `tar` 命令查看存档文件中的内容。

假设目录 `tarfiles` 中有一个存档文件 `project.tar`。下面命令显示文件 `project.tar` 中的打包文件列表。

```
$ tar -tvf ./tarfiles/project.tar[Return].....列出文件 project.tar 中的内容
```

选项 `t` 表示列出被打包文件的目录,选项 `v` 表示显示与打包时一样的详细目录,选项 `f` 表示下一个参数为存档文件的路径。

```
$ tar -tvf ./tarfiles/project.tar
drwx r-- r-- 2029/1005 0 Sep 3 10:33 2001 ./tarfiles/project/
-rw- r-- r-- 2029/1005 2350 Sep 3 10:33 2001 ./tarfiles/project/head
-rw- r-- r-- 2029/1005 5374 Sep 3 10:33 2001 ./tarfiles/project/tail
$_
```

【说明】

上面的显示输出格式与带 `-l` 选项的 `ls` 命令类似。

显示输出中各列的含义为:

第 1 列	第 2 列	第 3 列	第 4 列	第 5 列
<code>-rw- r-- r--</code>	<code>2029/1005</code>	<code>2350</code>	<code>Sep 3 10:33 2001</code>	<code>./tarfiles/project/head</code>
文件访问权限	用户标识/组标识	文件字节数	文件修改时间	文件名

现在可用如下命令行还原出存档文件中的所有被打包文件:

```
$ tar -xvf ./tarfiles/projects.tar[Return]...从 project.tar 中还原所有被打包文件
```

选项 `v` 表示还原存档文件 `projects.tar` 中的被打包文件;选项 `v` 表示在还原过程中显示详细信息。

```
$ tar -cvf ./tarfiles/projects.tar
x ./projects, 0 bytes, 0 tape blocks
x ./projects/head, 2350 bytes, 5 tape blocks
x ./project/tail, 5374 bytes, 11 tape blocks
$_
```

【说明】

上面的显示输出提供了每个文件的字节数和占用的存储块数。

通过指定文件名,`tar` 命令也可从备份介质中只还原出特定的文件。例如:

```
$ tar -xvf ./tarfiles/projects.tar tail[Return].... 从 projects.tar 中只还原文件 tail
x ./project/tail, 5374 bytes, 11 tape blocks.... 显示选项 v 对应的详细信息
$_..... 命令提示符
```

13.3 拼写错误更正

用 `spell` 命令可检查用户文档中的拼写错误。该命令把指定文件中的单词与字典文件进行比较,显示在字典文件中没有找到的单词。用户可指定多个要比较的文件,但在没有指定比较文件时,`spell` 命令会从默认输入设备(键盘)读入要比较的内容。下面是一些 `spell` 命令的应用实例。

【实例】

运行没有指定比较文件的 `spell` 命令。

```
$spell [Return]..... 没有指定任何参数
lookin good[Return].....输入是从键盘进行的
[Ctrl-d]..... 表示输入结束
lookin
good
$_..... 命令提示符
```

在新行的开始处按[Ctrl-d]表示输入结束。`spell` 命令并不支持拼写错误的更正,而仅仅是显示可疑的单词,每个单词占一行。`spell` 命令对大小写敏感,它认为,David 是正确的,而 david 是错误的。

【实例】

假定有一个名为 `my_doc` 的文件,检查它的拼写错误,并把结果保存在另一个文件中。

```
$spell my_doc > bad_words[Return]..... 对文件 my_doc 的内容进行拼写检查
$_..... 命令提示符
```

上面命令行实例中,`spell` 命令的输出被重定向到文件 `bad_words`,可用 `cat` 命令看到这个文件的内容。

用户可利用 `spell` 命令的参数来指定多个要进行拼写检查的文件,文件名间用空格分开。

【实例】

假设在 `vi` 文本编辑器中调用拼写检查命令。

```
:! spell[Return]..... 调用拼写检查命令
pervious..... 输入被怀疑可能有错的单词
[Ctrl-d]..... 表示输入结束
pervious..... 拼写可能有错的单词
[Hit Return to continue]..... 按回车键返回 vi 文本编辑器
```

【说明】

通过在冒号提示符下输入感叹号(!)和命令名,可实现在 `vi` 文本编辑器中调用任何命令(参见第 6 章)。

`spell` 命令选项

表 13.5 列出了 `spell` 命令的选项、相应解释和实例。

表 13.5 spell 命令的选项

选项	功 能
-b	按英国英语进行拼写检查
-v	显示字典中没有的派生单词
-x	显示被查单词的原形

-b 选项:该选项使 **spell** 命令按英国英语进行单词检查。在英国英语中, colour、centre、programme 等拼写都是正确的。

-v 选项:该选项可显示字典文件中没有精确对应的单词和派生方法。派生方法用一个加号和词缀表示。

【实例】

运行带 **-v** 选项的 **spell** 命令,从键盘输入被检查单词列表。

```
$ spell -v[Return]..... 要求从键盘输入被检查单词列表
appointment looking worked preprogrammed[Return]
[Ctrl-d]..... 表示输入被检查单词列表的结束
+ ment      appointment
+ ing       looking
+ ed        worked
+ pre       preprogrammed
$_.         回到命令提示符
```

-x 选项:该选项显示单词匹配时的原形查找过程,每个匹配单词前有一个等号。

【实例】

运行带 **-x** 选项的 **spell** 命令,从键盘输入被检查单词列表。

```
$ spell -x[Return]..... 要求从键盘输入被检查单词列表
appointment looking worked preprogrammed[Return]
[Ctrl-d]..... 表示输入被检查单词列表的结束
= appointment
= appoint
= looking
= looke
= look
= preprogrammed
= programmed
= worked
= worke
= work
$_.         回到命令提示符
```

13.3.1 创建用户词汇文件

在大多数 UNIX 系统中,用户都可创建自己的词汇文件,用于补充标准字典中词汇的不足。例如,可创建一个包括用户专业领域的专用单词和短语的文件。利用加号选项,可在命令行指定用于拼写检查的用户词汇表。**spell** 命令先用系统的标准字典进行检查,再与用户词汇

表对比。可用 vi 文本编辑器创建用户词汇文件。

【说明】

1. 用户词汇文件必须一个单词占一行；
2. 用户词汇文件必须按字母排序。

【实例】

假定用户词汇文件名为 my_owm, 使用加号选项指定用该用户词汇文件进行拼写检查。

```
$ spell 1my_owm BIG_FILE > MISSPELLED_WORDS[Return]
$.....命令提示符
```

这里的 1my_owm 表示要使用一个名为 my_owm 的用户词汇文件。spell 命令将显示系统标准字典文件和用户词汇文件这两个字典文件中都没有的单词。命令输出被重定向到文件 MISSPELLED WORDS。

【情景】 默认的标准字典文件中不包括 shell 命令或其他的 UNIX 词汇。因此, 如果被查文本中包括 grep 和 mkdir 等单词, 系统会把它们视为可疑单词显示出来。

用户可创建一个类似于图 13.5 中所示的文件作为用户词汇文件, 在其中列出各 shell 命令和 UNIX 单词。

```
$ cat U_DICTIONARY
grep
i-node
ls
mkdir
pwd
```

图 13.5 字典文件举例

【实例】

分别运行带指定用户词汇文件的 spell 命令和不带指定用户词汇文件的 spell 命令。

```
$ spell 1U_DICTIONARY[Return]..... 有用户词汇文件的 spell 命令
grep pwd mkdir ls[Return]..... 被检查的单词列表
$..... 没有拼写错误, 返回命令提示符
$ spell[Return]..... 没有用户词汇文件的 spell 命令
grep pwd mkdir ls[Return]..... 被检查的单词列表
grep
ls
mkdir
pwd
$..... 回到命令提示符
```

13.4 UNIX 系统安全

信息和计算机机时都是需要保护的有价资源。系统安全模块是多用户系统的重要组成部分, 需要从多种角度来保证系统安全。例如:

- 确保未经授权的人员不能得到系统的使用权限
- 确保未经授权的用户不能修改系统或其他用户的文件
- 允许某些用户具有一定特权

在 UNIX 系统中,可通过简单的命令来实施系统安全控制,可按用户的要求采用严格的或宽松的系统安全管理。下面介绍几种 UNIX 系统安全控制手段。

13.4.1 口令保护

系统管理的用户信息保存在文件 `/etc/passwd` 中。该文件中包括加密的用户口令,所采用的加密编码算法保证破解用户口令是十分困难的。

每个系统用户在 `passwd` 文件中有一条对应记录,每条记录由若干用冒号分开的字段构成,在文件中占一行。下面介绍 `passwd` 文件中每行的格式,并解释每个字段的含义。

```
Login-name:passwd:user-ID:group-ID:user-info:directory:program
```

login-name :这是用户的登录名。作为对系统的响应,在登录提示符下需输入登录名。

password:这是加密的用户口令。在 SVR4 中,加密的用户口令并不保存在 `passwd` 文件中,而是存储在文件 `/etc/shadow` 中;在 `passwd` 文件中的口令字段仅填一个字母 `x` 作为占位符。

【说明】

所有用户都可读出 `/etc/passwd` 文件,但普通用户不能读文件 `/etc/shadow`。

user-ID:该字段为用户标识号。系统为每个用户分配一个唯一的用户标识号,用户标识号 0 是指超级用户。

group-ID:该字段为组标识号。组标识号表示该用户是相应用户组的成员。

user-info:该字段用于进一步描述用户信息,通常包含用户的全名。

directory:该字段表示用户主目录的绝对路径。

program:该字段指定用户登录后要运行的程序,通常是一个 shell 程序。如果在该字段中没有指定要运行的程序,默认运行的程序为 `/usr/bin/sh`。通过修改该字段,可把 C shell 作为用户登录后的 shell 程序,也可在用户登录后运行其他程序。

图 13.6 是 `passwd` 文件的一个实例,系统中任何用户都可看到这个文件的内容。

```
$ cat /etc/passwd
root:x:0:1:admin:/:usr/bin/sh
david:x:110:255:David Brown:/home/david:/usr/bin/sh
emma:x:120:255:Emma Redd:/home/emma:/usr/bin/sh
steve:x:130:255:Steve Fraser:/home/steve:/usr/bin/csh
$_
```

图 13.6 `passwd` 文件实例

13.4.2 文件保护

文件保护就是控制各用户对文件的访问权限。UNIX 系统为用户提供了相应命令,指定可

访问文件的用户列表和可访问用户的访问操作种类。

在第 11 章讨论过 `chmod` 命令,现在我们探讨另一种设置文件访问权限的方法。用带 `-l` 选项的 `ls` 命令(参见第 5 章)可列出用户目录的详细列表,其中的一个重要内容是每一行中代表访问权限的 `rwX` 信息。`chmod` 命令可用于修改文件的访问模式(权限)。例如,要允许所有用户拥有文件 `myfile` 的执行权限,可输入如下命令行:

```
$ chmod a = x myfile[Return]
```

另一种设置访问权限的方法是用一个 3 位八进制整数来代表新设置的权限。表 13.6 列出了各种访问权限对应的等价整数。

表 13.6 文件访问权限的等价整数

所有者	同组用户	其他用户
<code>rwX</code>	<code>rwX</code>	<code>rwX</code>
<code>421</code>	<code>421</code>	<code>421</code>

假设当前目录中有一个文件 `mayflies`,下面例子说明如何使用 `chmod` 命令修改文件访问权限,用 `ls -l` 命令可确认修改结果是否正确。

【实例】

把文件 `mayflies` 的访问权限修改为允许所有用户读、写和执行。

```
$ chmod 777 mayflies[Return]..... 修改文件 mayflies 的访问模式
$_..... 完成修改,返回命令提示符
```

【说明】

整数 777 表示文件所有者、同组用户和其他用户都拥有读、写和执行权限,其中每个二进制位对应一种权限。

【实例】

修改文件 `mayflies` 访问权限,使所有用户有读权限,但只有所有者有写和执行权限。

```
$ chmod 744 mayflies[Return]..... 修改文件 mayflies 的访问模式
$_..... 完成修改,返回命令提示符
```

数字 7 表示所有者有读、写和执行权限,第一个 4 表示同组用户只有读权限,第二个 4 表示其他用户只有读权限。

【实例】

允许所有者和同组用户拥有所有读、写和执行权限,但只允许其他用户有执行权限。

```
$ chmod 771 mayflies[Return]..... 修改文件 mayflies 的访问模式
$_..... 完成修改,返回命令提示符
```

数字 1 表示其他用户只有执行权限。

13.4.3 目录访问权限

目录的访问权限控制与文件类似,但对于目录,各权限位的含义有变化。

读(**read**):目录的读权限(**r**)表示用户可用 **ls** 命令列出目录中的文件名;

写(**write**):目录的写权限(**w**)表示用户可在目录中添加和删除文件;

执行(**execute**):目录的执行权限(**x**)表示用户可用 **cd** 命令进入该目录或使用该目录作为路径名的一部分。

【实例】

允许所有用户拥有目录 **mybin** 的所有读、写和执行权限。

```
$ chmod 777 mybin[Return]..... 修改目录的访问模式
$_..... 完成修改,返回命令提示符
```

与文件访问权限相同,命令中每个数字代表一组用户(所有者、同组用户和其他用户)。数字 7 表示允许相应的用户群体对目录进行读、写和执行操作。

13.4.4 超级用户

本章中的一些命令只有系统的超级用户可使用,那么谁是超级用户呢?超级用户是一个拥有特权的用户,不受文件访问权限的限制。系统管理员必须是超级用户,以便进行创建用户账号以及修改口令等管理工作。超级用户通常以登录名 **root** 进行登录,可读、写和删除系统中的任何文件,甚至可关闭系统。超级用户身份通常分配给系统管理人员。

13.4.5 文件加密:crypt 命令

系统允许超级用户忽视文件访问权限设置而访问任何文件。如何保护用户的敏感文件不被其他用户(可能是或不是超级用户)访问?UNIX 系统提供了 **crypt** 命令,可对用户文件进行加密,从而使其他用户不能读出该文件。**crypt** 命令按一种可逆方式对用户文件中的每一个字符进行变换,使用户以后可以还原出原来的用户文件。加密编码机制是基于简单的置换方法,如把文件中的字符 A 置换成字符 ~。**crypt** 命令利用密码来打乱从标准输入到密文的变换,变换后的密文被送到标准输出。

crypt 命令既可用于加密,也可用于解密。在解密时,用户必须使用与加密时相同的密码。下面是一些实例。

【实例】

加密当前目录中的一个文件 **names**。

```
$ cat names[Return]..... 显示文件 names 的内容
David Emma Daniel Gabriel Susan Maria
$ crypt xyz < names > names.crypt[Return]..... 加密口令为 xyz
$ rm names[Return]..... 删除文件 names
$ crypt xyz < names.crypt > names[Return]..... 对加密文件 names.crypt 解码
$ cat names[Return]..... 检查文件 names 的内容
David Emma Daniel Gabriel Susan Maria
$_..... 返回命令提示符
```

xyz 是加密文件时使用的密码,这个密码是以后解码时需要的口令。

实例中使用了输入/输出重定向,命令的输入来自文件 **names**,输出保存在文件 **names.crypt** 中。

在对文件进行加密后,通常要删除明文,而只保留密文。这样可保证只有知道密码的人才可得到文件中的信息。

【实例】

对文件 `names` 进行加密,但在命令行指定密码。

```
$ crypt < names > names.crypt[Return]..... 没有在命令行指定密码
Key..... 提示输入密码
$_..... 返回命令提示符
```

【说明】

如果在命令行没有指定密码, `crypt` 会提醒用户输入密码。密码不会在屏幕回显,这种方法比在命令行指定密码要安全。

【注意】

如果在对文件进行加密时输入了错误的密码,以后将无法把密文还原成明文。为了保险起见,最好先尝试对密文进行解码,在确保可正确解码后再删除明文。

13.5 使用 FTP

文件传输协议(FTP, File Transfer Protocol)是 UNIX 系统提供的最常用服务。使用 FTP 命令可把文件从一台机器传送到另一台机器。虽然用户要指定文件是二进制文件或 ASCII 文件,FTP 传送的文件可为任何类型。

【说明】

1. FTP 不仅是一个协议名字,同时还是一个程序名或命令名。
2. FTP 是因特网上一种流行的信息共享方法。

FTP 命令格式为先输入 `ftp`,再输入另一台机器的地址并按回车键。

```
$ ftp server2[Return]..... 发出 FTP 命令
```

接下来,系统会提示输入用户名和口令。用户必须在 `server2` 系统上有登录权限。

```
Username: daniel..... 输入用户名,这里使用的用户名为 daniel
Password:..... 输入口令,这里没有回显口令
```

如果用户在 `server2` 系统中有一个登录名,则可在当前系统与 `server2` 系统间传送文件。FTP 提供一组用于从一个系统向另一个系统传输文件的命令,包括列目录和任意方向的文件复制等。

13.5.1 FTP 的工作原理

FTP 是以客户/服务器方式工作的。可用如下方式使用带一个远端机器地址的 `ftp` 命令:

```
$ ftp server2[Return]..... 发出与 server2 系统通信的 ftp 命令
```

这里 `server2` 是用户要进行通信的远端机器名。`ftp` 程序立即尝试与用户在命令行中指定的 FTP 服务器建立连接,例子中的 FTP 服务器在 `server2` 系统上。

用户发出 **ftp** 命令时,在本地系统上执行的 FTP 进程是一个客户进程,它与 server2 系统上的 FTP 服务器进程相对应。用户从 FTP 客户进程向 FTP 服务器进程发出命令,由服务器进程作出适当的响应。

【说明】

在一个 FTP 会话中,用户使用的本地系统称为本地主机,与用户建立连接的系统称为远程主机。

可用 FTP 的交互模式与远程主机建立连接。如果没有指定 FTP 服务器的名字,ftp 客户进程进入交互模式并提示用户输入下一步命令。例如:

```
$ ftp[Return]..... 没有指定要通信的远程主机
ftp> ..... ftp 客户进程进入交互模式并显示相应提示符
```

要查见全部 **ftp** 命令列表,可在一个 FTP 会话期间输入问号(?)并按回车键。例如:

```
ftp> ..... 提示符表示正在处于一个 FTP 会话
ftp> ? [Return]..... 列出所有可用命令
```

在 FTP 会话中输入 **bye** 命令可终止当前会话,回到 shell 的命令提示符。

```
ftp> bye[Return].....终止 ftp 会话
```

用户也可使用如下的退出(**quit**)命令:

```
ftp> quit[Return].....终止 ftp 会话
```

在这两种方式下,系统都会显示会话结束信息。

```
221 Goodbye.
```

图 13.7 给出了在 **ftp** 会话中使用? 命令时显示的命令列表。

```
FTP> ?
Commands may be abbreviated. Commands are:
!          cr          macdef     proxy      send
$          delete      mdelete   sendport   status
account   debug       mdir      put        struct
append    dir         mget      pwd        sunique
ascii     disconnect  mkdir     quit       tenex
bell      form        mls       quote      trace
binary    get         mode      recv       type
bye       glob        mput      remotehelp user
case      hash        nmap      rename     verbose
cd        help        ntrans    reset      ?
cdup      lcd         open      rmdir
close     ls          prompt    runique
ftp>
```

图 13.7 FTP 命令列表

在用户使用 FTP 时,可能通常只需要很少的几个命令。下面表格分类列出了大多数 **ftp**

命令。

FTP 连接命令

与 FTP 连接相关的命令用于与远程主机建立连接和在 FTP 会话结尾的终断连接。表 13.7 简要说明了这些命令。

表 13.7 FTP 连接命令

命令	功 能
open 远程主机名	创建一个与指定主机上 FTP 服务器进程的连接。该命令将提示用户输入用于在远程主机上登录的用户名和口令
close	关闭已创建的当前连接,返回本地 FTP 命令状态。这时可发出与另一台远程主机进行连接的 open 命令
quit (bye)	关闭与远程主机进行当前的 FTP 会话,退出 FTP 客户进程。也就是返回 UNIX 系统的 shell 程序

FTP 文件传输命令

与 FTP 文件传输相关的命令用于在本地主机和远程主机间相互传送文件。有两种常用的文件传输模式:文本模式和二进制模式,默认传输模式为文本模式。用文本模式可传输任何文本(ASCII)文件,而目标文件或可执行文件等二进制文件必须用二进制模式进行传输。表 13.8 简要说明了这些命令。

表 13.8 FTP 文件传输命令

命令	功 能
ascii	设置文件传输模式为文本模式。这是系统的默认文件传输模式
binary	设置文件传输模式为二进制模式
get 远程文件名[本地文件名]	从远程主机复制一个文件到本地主机。如果没有指定本地文件名,复制到本地主机的文件名与远程文件名相同
mget 远程文件名	从远程主机复制多个文件到本地主机
put 本地文件名 [远程文件名]	从本地主机复制一个文件到远程主机。如果没有指定远程文件名,复制到远程主机的文件名与本地文件名相同
mput 本地文件名	从本地主机复制多个文件到远程主机

FTP 文件和目录控制命令

与 FTP 文件和目录控制相关的命令用于 FTP 会话中的文件管理和控制,这些命令与 UNIX 文件和目录控制命令类似。表 13.9 简要说明了这些命令。

表 13.9 FTP 文件和目录控制命令

命令	功 能
cd 远程目录名	修改远程主机上的当前目录为指定目录
lcd 本地目录名	修改本地主机上的当前目录为指定目录
dir	显示远程主机上当前目录的文件列表
pwd	显示远程主机上的当前目录名
mkdir 远程目录名	在远程主机上创建一个目录。要求用户必须在远程主机上拥有相应的权限
delete 远程文件名	删除远程主机上的一个指定文件
mdelete 远程文件名	删除远程主机上的多个指定文件

其他命令

问号(?)命令可提供 FTP 命令的帮助信息。叹号(!)是 shell 出口命令,可用于在本地主机上运行 shell 命令。**hash** 命令可在文件传输过程中提供反馈信息,该命令在传送大文件时十分有用。表 13.10 简要说明了这些命令。

表 13.10 其他 FTP 命令

命令	描 述
? 或 help	显示指定命令的解释信息。如果没有指定参数,显示所有命令列表
!	切换到临时 shell 命令方式
hash	作为反馈信息,每传送一个数据块显示一个 # 号。这里数据块的大小为 8192 字节

下面通过一些 FTP 会话实例说明 **ftp** 命令的使用方法。

创建一个 FTP 会话

ftp open 命令用于创建与远程主机的 FTP 会话。下面命令序列说明如何创建与远程主机 **server2** 的 **ftp** 会话。

```

$ ftp[Return]..... 发出 ftp 命令
ftp> ..... 显示 ftp 提示符
ftp> open[Return]..... 发出 open 命令
(to)..... 显示 open 提示符
(to)server2[Return]..... 输入远程主机名
Responses from FTP server on the remote host (server2)
Connected to server2
220 server2 FTP server (UNIX(r) System V Release 4.0) ready.
Name (server2:gabe)..... 显示登录名输入提示
Name (server2:gabe)gabe[Return]..... 输入远程主机上的登录名

```

```

331 password requires for gabe
password:..... 显示口令输入提示
password:[Return]..... 输入口令(没有回显)
230 User gabe logged in.
ftp> bye[Return]..... 结束 FTP 会话
221 Goodbye.....ftp 会话结束信息

```

用户也可在命令行指定远程主机名。例如,要创建与远程主机 server2 的 FTP 会话,可输入:

```

$ ftp server2[Return].....在命令行指定远程主机名
Connected to server2
220 server2 FTP server (UNIX(r) System V Release 4.0) ready.

```

与前面的 FTP 会话例子类似,在建立 FTP 会话前系统会提示用户输入远程主机上的登录名和口令。

下面命令序列说明如何使用帮助信息和其他 ftp 命令。这里假设用户已与远程主机 server2 建立了 FTP 会话。

【实例】

bell 命令

```

ftp> help bell[Return]..... 显示 bell 命令的描述信息
bell    beep when command completed(在命令结束时发出嘟声)
ftp> _..... 显示 ftp 提示符
ftp> bell[Return]..... 显示发声设置
Bell mode on. (发声设置为打开状态)
ftp> bell[Return]..... 关闭发声
Bell mode off. (发声设置为关闭状态)
ftp> _..... 显示 ftp 提示符

```

【实例】

hash 命令

hash 命令用于在文件传送中每传送完一个数据块就给出一个反馈信息,数据块的大小为 8192 字节。如下所示,每传送完一个数据块,系统会显示一个 # 号。通常在传送大文件前要打开反馈显示设置。

```
#####
```

通常在传送大文件前要打开反馈显示设置。

```

ftp> help hash[Return]..... 显示 hash 命令的描述信息
hash    toggle printing '#' for each buffer transferred (每传送一个缓冲块显示一个 #号)
ftp> _..... 显示 ftp 提示符
ftp> hash[Return]..... 打开反馈显示设置
Hash mark printing on (8192 bytes/hash mark).hash (反馈显示设置为打开,每传送 8192 字节显示一个符号)
ftp> hash[Return]..... 关闭反馈显示设置
Hash mark printing off. (反馈显示设置为关闭)
ftp> _..... 显示 ftp 提示符

```

【实例】

文件传送模式命令

```
ftp> help bin[Return]..... 显示 bin 命令的描述信息
binary      set binary transfer type      (设置文件传送模式为二进制模式)
ftp> bin[Return]..... 设置文件传送模式为二进制模式
200 Type set to I.
ftp> ascii[Return]..... 设置文件传送模式为文本模式
200 Type set to A.
ftp> _..... 显示 ftp 提示符
```

【实例】

从远程主机复制文件

假设已创建了与远程主机 server2 的 FTP 会话,并且在远程主机的目录 Memos 中有一个文件 Notes。下面命令序列说明如何从远程主机取到文件 Notes。用户发出命令后,可收到服务器给出的一些反馈信息,表示它正在执行用户命令。

cd 命令用于修改远程主机上的当前目录为目录 Memos。

```
ftp> cd Memos[Return]..... 进入目录 Memos
ftp> _..... 显示 ftp 提示符
```

dir 命令用于列出远程主机(server2)上当前目录的文件列表,检查是否有文件 Notes。

```
ftp> dir[Return]..... 给出当前目录的文件列表
-rwx-x-x 1 tuser other 3346 Aug 9 11:51 Notes
ftp> _..... 显示 ftp 提示符
```

也可用 **ls** 命令来列出文件名。

```
ftp> ls[Return]..... 给出当前目录的文件列表
(local-file)..... 提示输入要显示的文件名
(local-file)november.rpt[Return]..... 输入本地主机上的文件名
(remote-file)..... 提示输入要显示的文件名
(remote-file)november.rpt[Return]..... 输入远程主机上的文件名
```

远程主机从本地主机的当前目录传送文件时,用户可收到一些关于传送字节数和传送速率的反馈信息。

```
200 PORT command successful.
150 ASCII data connection for november.rpt
(192.246.52.166, 37175).
226 Transfer complete.
ftp> _
```

【实例】

其他命令

如前所述,? 命令可列出所有 ftp 命令。通过指定一个命令名作为? 命令的第一个参数,我们也可用? 命令获得一个 ftp 命令的帮助信息。例如:

```
ftp> ? close[Return]..... close 命令的帮助信息
```

Close Terminate FTP session.

(终止 FTP 会话)

shell 出口命令(!)可在本地主机上启动一个子 shell 进程。如果用户正在与一个远程的 ftp 服务器通信时要在本地主机上执行其他操作,该命令是很有用的。用户完成本地主机上的操作并退出子 shell 进程时,系统会返回原来的 FTP 会话。下面命令序列演示! 命令的用法。

假设用户正在进行一个 FTP 会话,下面命令可启动一个子 shell 进程。

```
ftp> ! [Return]..... 发出 shell 出口命令
$_..... 本地主机的命令提示符
```

现在位于本地主机的当前目录下,调用的任何命令都是在本地主机上执行的。用户可使用任何有效的 UNIX 命令,这些命令作用于本地主机的文件。例如:

```
$ cd [Return]..... 修改本地主机的当前目录为主目录
$_..... 本地主机的命令提示符
```

修改本地主机的当前目录为用户主目录。可用退出命令来终止子 shell 进程,并返回原来的 FTP 会话。例如:

```
$ exit [Return]..... 终止子 shell 进程并返回 FTP 会话
ftp> _..... 返回 FTP 会话
```

13.5.2 匿名 FTP

匿名 FTP 是使用 ftp 程序和远程主机上指定的受限账号 anonymous。利用匿名 FTP 可在因特网上访问和共享文档、源代码和可执行文件等各种文件。匿名 FTP 是一个允许匿名访问远程主机文件的特殊账号。下面讨论的匿名 FTP 使用步骤与其他 FTP 账号的使用类似。

1. 需要了解允许使用匿名 FTP 的主机名或 IP 地址。
2. 使用 anonymous 作为用户名。
3. 输入一个非空字符串作为口令。一些系统要对匿名登录进行口令验证,这时需要用电子邮件地址作为口令。

一旦用匿名 FTP 登录成功,可获得远程主机上匿名 ftp 目录的有限访问能力,本小节所描述的所有命令都可使用。

FTP 站点通常把任何人都可访问的文件放在目录/public 中。一些站点的 ftp 目录下还有一个称为 incoming 的目录,可用于匿名 FTP 的文件上载。

【实例】

匿名 FTP 会话

下面命令序列是一个匿名 FTP 会话的实例,演示 FTP 命令和相应反馈信息。

假设匿名 FTP 站点为 ftp.xyz.net,要下载的文件为/pub/services/example.tar:

```
$ ftp .xyz.net [Return]..... 发出与远程主机 ftp.xyz.net 创建 FTP 会话的命令
220 ftp.xyz.net FTP server (Version 1.1 Nov 17 2001) ready.
Name (ftp.xyz.net:xyz):anonymous [Return] 用 anonymous 作为用户名
331 Guest login ok, send ident as password. (欢迎访问,请输入标识作为口令)
password: [Return].....输入电子邮件地址作为口令(没有回显)
230 Guest login ok,access restrictions apply. (欢迎登录,访问权限控制生效)
```

```
ftp> cd/pub/services[Return]..... 进入要访问的目录
250 CWD command successful.         (目录修改命令完成)
ftp> get examples.tar[Return]..... 下载文件
200 PORT command successful.        (找到指定文件)
266 Transfer complete.              (下载成功完成)
```

【说明】

1. 许多 FTP 站点都有一个名为 README 的文件, 该文件包含各目录内容的信息。下面命令可在不完全下载文件的条件下阅读 README 文件或任何文本文件的内容。

```
ftp> get README/pg [Return]
```

2. 一旦以匿名 FTP 账号成功登录, 可使用本小节所讲的所有 FTP 命令。

用 help 命令可获得 get 命令的功能描述。

```
ftp> helpget[Return]..... get 命令的功能描述信息
get      receive file          (文件下载)
ftp> _..... 显示 ftp 提示符
```

下面 get 命令可从远程主机上下载文件 Notes。

```
ftp> get[Return]..... 发出 get 命令
(remote-file)..... 提示输入文件名
(remote-file)Notes[Return]..... 输入远程主机上的文件名
(local-file)..... 提示输入文件名
(local-file)Notes.txt[Return]..... 输入本地主机上的文件名
```

在完成从远程主机到本地主机的当前目录的传送文件后, 系统会显示传送字节数和传送速率的信息。

```
200 PORT command successful.
150 ASCII data connection for Notes.txt (193.246.52.166, 37176) (0 bytes).
226 ASCII Transfer complete.
```

也可在命令行输入文件名。例如:

```
ftp> get Notes Notes.txt[Return]..... 下载文件 Notes
```

这样, 远程主机上的文件 Notes 被传送到本地主机的当前目录, 下载后的文件名为 Notes.txt。

【实例】

传送到远程主机

用 put 命令可把本地主机上的文件传送到远程主机上。假设已创建了与远程主机 server2 的 FTP 会话, 现在要把本地主机当前目录的子目录 Reports 中的文件 november.rpt 传送到远程主机上。下面命令序列说明如何从本地主机传送文件 november.rpt。用户发出命令后会收到服务器的一些响应信息, 表明它正在执行命令。

要把本地主机当前目录的子目录 Reports 中的文件 november.rpt 传送到远程主机, 可先创建一个与远程主机的 FTP 会话, 再执行下面步骤:

用 `lcd` 命令修改本地主机的当前目录。

```
ftp> lcd reports[Return].....修改当前目录
```

用 `help` 命令获取 `put` 命令的在线帮助信息。

```
ftp> help put[Return].....put 命令的帮助信息
put send one file                (上传一个文件)
```

用 `put` 命令把本地主机上的文件 `november.rpt` 上传到远程主机。

```
ftp> put[Return].....发出 put 命令
```

13.6 使用压缩文件

在匿名 FTP 站点上,大的文件或软件包通常要进行压缩,以节约存储空间。这些压缩文件的格式通常为 `tar` 格式,相应文件名后缀为 `.tar.Z`;下载时必须用二进制模式。在使用前要首先对这些文件进行解压缩,再用 `tar` 命令还原出原来的文件。

13.6.1 `compress` 和 `uncompressed` 命令

用 `compress` 命令可减少文件的大小,以节约存储空间。当用 `compress` 命令压缩一个文件时,被压缩文件会被一个后缀为 `.Z` 的同名文件代替。作为一种安全措施,压缩命令也可与加密命令一起使用。在这种情况下,先压缩文件,后用 `crypt` 命令进行加密。下面命令序列说明 `compress` 命令的用法。

【实例】

假设当前目录中有一个名为 `important` 的文件。

```
$compress important[Return]..... 发出压缩命令
$ls important *[Return].....      显示文件名
important.z.....                  压缩后的文件
$_.....                            命令提示符
```

可使用 `compress` 命令的 `-v` 选项,以显示文件的压缩比。

```
$compress -v important[Return]..... 使用-v 选项
important: compression: 49.18% -replaced with important.Z
(压缩比为 49.18%, 压缩后的文件名为 important.Z)
$_.....                            命令提示符
```

用 `uncompress` 命令可把压缩文件还原成原来的文件;该命令对压缩文件进行解压缩,并删除压缩文件。继续上面的例子,假设在当前目录中有一个名为 `important.z` 的压缩文件。

```
$uncompress important[Return]..... 对文件 important.z 文件进行解压缩
$_.....                            命令提示符
```

命令小结

下面对本章所讨论的命令进行小结。

df

该命令可显示指定文件系统的磁盘空间总量或 unused 磁盘空间。

du

该命令可统计每个目录、子目录和文件占用的存储空间。

选项	功 能
-a	显示目录和文件的大小
-s	只显示指定目录占用的存储块数目,不列出子目录

finger

该命令显示详细的用户信息。

选项	功 能
-b	不在长格式中输出用户主目录和 shell 程序名
-f	不在短格式中输出标题行
-h	不在长格式中输出文件 .project 的内容
-l	强制使用长格式输出
-p	不在长格式中输出文件 .plan 的内容
-s	强制使用短格式输出

banner

该命令用大号字符显示它的参数——一个指定的字符串。

spell

该命令可检查指定文档或键盘输入单词中的拼写错误。它只显示在字典文件中没有找到的单词,并不提供更正建议。

选项	功 能
-b	按英国英语进行拼写检查
-v	显示字典中没有的派生单词
-x	显示被查单词的原形

calendar

该命令是一种提醒服务,可从当前目录的日历文件中读取日程安排。

compress

该命令用于压缩指定文件,以减少文件尺寸并节约存储空间。解压缩命令可还原出压缩前的文件,并删除压缩文件。

tar

该命令可把一组文件复制到一个称为存档文件的单个文件中。存档文件通常放在磁带上,但也可存在软盘等其他介质上。它把多个文件打包成一个可解包的 tar 格式的单个文件。

选项	功 能
c	(create) 创建一个新的存档文件, 该文件中的原来内容被覆盖
r	(replace) 添加文件到存档文件中, 该文件中的原来内容保留
t	(table of contents) 列出存档文件中的所有被打包文件的文件名
x	(extract) 从存档文件中还原出被打包文件
f	(file) 用下一个参数作为存档文件的存放位置
v	(verbose) 显示详细打包过程信息

FTP

该命令可把文件从一台机器传送到另一台机器。被传送文件可为任何类型, 用户可指定被传送文件是二进制文件或 ASCII 文件。输入 ftp 可启动一个 FTP 会话。

FTP 的连接命令

命令	功 能
open 远程主机名	创建一个与指定主机上 FTP 服务器的连接。该命令将提示用户输入在远程主机上登录的用户名和口令
close	关闭已创建的当前连接, 返回本地 FTP 命令状态。这时可发出与另一台远程主机进行连接的打开命令
quit (bye)	关闭与远程主机进行当前的 FTP 会话, 退出 FTP 客户进程。也就是返回 UNIX 系统的 shell 程序

FTP 的文件和目录命令

命令	功 能
cd 远程目录名	修改远程主机上的当前目录为指定目录
lcd 本地目录名	修改本地主机上的当前目录为指定目录
dir	显示远程主机上当前目录的文件列表
pwd	显示远程主机上的当前目录名
mkdir	在远程主机上创建一个目录。要求用户在远程主机上拥有相应的权限
delete 远程文件名	删除远程主机上的一个指定文件
mdelete 远程文件名	删除远程主机上的多个指定文件

其他 FTP 命令

命令	功 能
? 或 help	显示指定命令的解释信息。如果没有指定参数, 显示所有命令列表
!	切换到临时 shell 命令方式
hash	作为反馈信息, 每传送一个数据块显示一个 # 号。这里数据块的大小为 8192 字节

习题

1. 简要说明 UNIX 系统的安全机制。用什么方法来保护文件系统?
2. 超级用户可删除其他用户的文件吗?
3. 超级用户可读取其他用户的加密文件吗?
4. `cd` 是 shell 内部命令吗? 你是如何知道的?
5. 解释实际时间、用户态时间和系统态时间。
6. 用什么命令可获得磁盘空间信息?
7. 在 vi 中可调用 `spell` 命令吗? 如果可以,怎么调用?
8. 如果要在以后的某时刻运行一个程序,用什么命令?
9. 对于文件和目录的访问权限,其含义相同吗?
10. `tar` 命令有什么用?
11. 什么是存档文件?
12. 用 `tar` 命令可将一组文件备份到磁带以外的介质上吗?
13. 用什么命令可列出一个名为 `save.tar` 的存档文件中的文件列表?
14. `compress` 命令有什么用?

上机练习

在 UNIX 系统上给出下列操作的命令。

1. 在明天 13:00 点钟对一个文件排序。
2. 在星期三下午 6 点向另一个用户发送一个电子邮件。
3. 用 `time` 命令统计 `sort`、`spell` 等命令的运行时间信息。
4. 修改用户目录的访问权限为只有所有者有读、写和执行权限。
5. 检查一个文本文件的拼写错误。
6. 创建一个用户字典文件,使 `spell` 命令用它作为附加的词汇表。
7. 查询磁盘上的可用空间。
8. 查询用户的主目录及其中的文件和子目录占用的存储块数。
9. 在初始化脚本文件 `.profile` 中加入 `du` 和 `df` 命令,以在每次登录时统计磁盘使用情况。
10. 如果有相应权限,请对一个文件进行加密。先在终端上显示加密后的文件,再对该文件进行解密和显示。
11. 在屏幕上作一条标语,显示自己的姓名。
12. 在打印机上打印一条标语,内容为自己的姓名。
13. 能让系统在用户登录时显示用户姓名吗?
14. 修改第 12 章中的 `greetings` 程序,用大号字符显示欢迎词。
15. 在主目录中创建一个日历文件,输入日程安排。
16. 用 `calendar` 命令显示现在的日程安排。
17. 在初始化脚本文件 `.profile` 中加入 `calendar` 命令,在登录时显示日程安排。
18. 用 `tar` 命令把当前目录中的文件备份到另一个名为 `my_tar_files` 的目录中。

-
19. 使用 `tar` 命令的 `-t` 和 `-v` 等选项,观察其输出信息。
 20. 用 `tar` 命令的 `-x` 选项从存档文件中还原出一个指定的文件。
 21. 用 `compress` 命令压缩一个大文件,观察显示压缩比的反馈信息,列出文件并检查其扩展名。
 22. 解压缩一个文件,并检查压缩文件是否被删除。

附录 A 命令索引

本附录是本书所涉及命令的快速索引。命令按字母顺序排列。

alias	为命令创建别名(ksh)	mkdir	创建一个目录
at	在指定时间执行程序	more	分屏显示文件
banner	显示标题(用大字体)	mv	移动/重命名文件
cal	提供日历服务	news	查看本机系统的新闻
calendar	提供日程提示服务	nohup	用户退出系统后,后台命令继续执行
cancel	删除(终止)打印请求	passwd	改变用户登录口令
cat	显示文件	paste	合并文件
cd	改变当前目录	pg	显示文件,每次一屏
chmod	改变文件/目录权限	pr	在打印前格式化文件
compress	文件压缩	ps	提交进程状态报告
cp	复制文件	pwd	显示当前/工作目录名
crypt	文件加密和解密	r(redo)	重复历史文件中的命令
cut	从文件中选择指定域/列	read	从输入设备读取输入
date	显示日期和时间	rm	移去(删除)文件
df	显示空闲磁盘空间大小	rmdir	移去(删除)空目录
.(点)	在当前 shell 环境中运行进程	set	设置/显示 shell 变量
du	报告磁盘使用情况	sh	调用另一个 shell
echo	显示(回显)参数	sleep	使进程等待指定的时间(按秒计)
exit	终止当前 shell 程序	sort	按特定的顺序对文件排序
export	向其他 shell 传送变量	spell	提供拼写检查
expr	提供算术运算操作	stty	设置终端选项
find	查找指定文件	tail	显示指定文件的最后部分
finger	显示用户信息	talk	提供终端到终端通信
grep	在文件中查找指定字符串	tar	将一系列文件存档到磁带上 的一个磁带文件上
head	显示指定文件的第一部分	tee	分离输出
help	调用菜单驱动帮助系统工具	test	判断表达式为真/假
history	显示所有键入命令的清单(ksh)	time	统计命令执行时间
kill	终止进程	tput	提供对 terminfo 数据库的访问
learn	调用课程系统工具	trap	设置/取消中断信号
let	提供算术运算操作(ksh)	type	显示指定命令的类型
ln	链接文件	unset	删除 shell 变量
lp	在行打印机上打印文件	vi	调用标准 UNIX 编辑器
lpstat	提供打印请求的状态	view	调用只读 vi 编辑器
ls	显示目录内容	wall	给当前所有登录的终端发消息(write all)
mailx	提供电子邮件处理系统(email)	wc	计算指定文件的行数、字数或字符数
man	从电子手册中查找信息	who	显示哪些用户登录到系统上
msg	禁止/否定来自 write 命令的消息	write	提供终端到终端通信

附录 B 分类命令索引

本索引是本书中所涉及命令的快速索引。本索引中的命令根据功能进行分类,然后按字母顺序排列。数字排在字母之后。

B.1 文件与目录命令

> << >>	重定向操作符	ls	列出目录内容
	管道操作符	mkdir	创建目录
cat	显示文件	mv	重命名/移动文件
cd	改变当前目录	paste	合并文件
chmod	改变文件/目录权限	pwd	显示当前目录
compress	文件压缩	rm	删除文件或者目录
cp	复制文件	rmdir	删除空目录
cut	从文件中选择指定域/列	tar	把一组文件存档为一个 tar 文件
ln	链接文件		

B.2 通信命令

mailx	提供电子邮件服务	wall	给当前所有登录的终端发消息 (write all)
mesg	禁止/否定来自 write 命令的消息	write	提供端到端的通信
news	查看本机系统的新闻		
talk	提供端到端的通信		

B.3 帮助命令

help	菜单式帮助程序	man	电子手册信息查找
learn	课程式帮助程序		

B.4 进程控制命令

alias	创建命令别名(ksh)	ps	提交进程状态报告
at	在指定时间执行程序	r(redo)	重复历史文件中的命令
kill	终止进程	sleep	使进程等待指定的时间(按秒计)6
nohup	用户退出系统后,后台命令继续执行		

B.5 行打印命令

cancel	取消打印请求	lpstat	显示打印请求状态
lp	打印文件	pr	打印前格式化文件

B.6 信息处理命令

df	显示空闲磁盘空间大小	pwd	显示当前/工作目录
du	报告磁盘使用情况	set	设置/显示 shell 变量
expr	提供算术运算操作	sort	按特定的顺序对文件排序
find	查找指定文件, 并进行操作	spell	提供拼写检查
finger	显示用户信息	tail	显示指定文件的最后部分
grep	在文件中查找指定字符串	time	统计命令执行时间
head	显示指定文件的开始部分	type	显示指定命令的类型
history	显示所有键入的命令的清单(ksh)	unset	删除 shell 变量
let	提供算术运算操作(ksh)	wc	计算指定文件的行数、字数或字符数
more	分屏显示文件	who	显示哪些用户登录到系统上
ps	提交进程状态报告		

B.7 终端命令

more	分屏显示文件	stty	设置终端选项
pg	显示文件, 每次一屏	tput	提供对终端信息数据库的访问

B.8 安全命令

chmod	改变文件/目录权限	passwd	改变用户登录口令
crypt	文件加密和解密		

B.9 开始/结束会话

[Ctrl-d]	结束会话	login	登录提示
exit	结束会话(退出系统)	passwd	改变用户登录口令

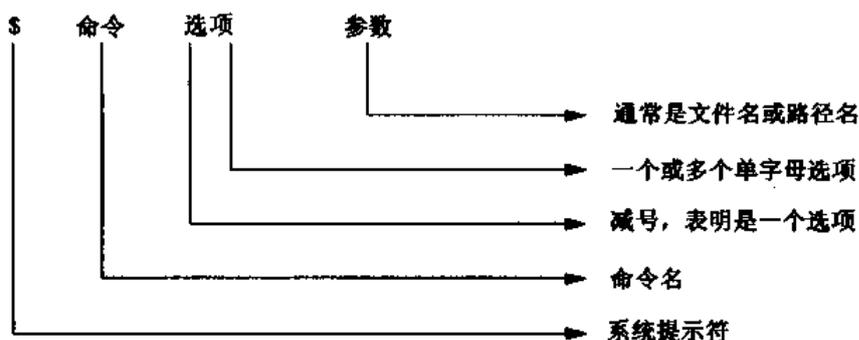
B.10 UNIX 编辑器

ed	UNIX 支持的行编辑器	vi	标准的 UNIX 全屏编辑器
emacs	emacs 编辑器	view	只读模式的 vi
ex	标准的 UNIX 行编辑器		

附录 C 命令小结

下面是按字母序排列的 UNIX 命令(系统工具)。为了强化记忆,这里再重复一下命令行格式。

命令行格式:



alias (ksh)

为命令创建别名。

at

该命令在以后的日期或时间运行用户程序。用户可以用不同的格式来指定时间。该命令在指定时间运行用户命令,即使用户没有登录到系统上。

选项	操 作
-l	列出 at 命令提交的所有任务
-m	完成任务后,给用户发送一个短信息进行确认
-r	从 at 任务表中删除指定的任务

banner

该命令以大写字母显示其参数——指定的字符串。

cal(calendar)

该命令显示指定的年或指定的年月的日历。

calendar

提醒服务,可从当前目录中的日历文件读取日程安排。

cancel(cancel print requests)

该命令用来取消不想要的打印请求。

cat (catenate)

显示文件。

cd (change directory)

这条命令把当前目录改到其他目录。

chmod (change mode)

该命令改变选项字母指定类别的用户对某特定文件的访问权限。用户类别有：**u**(使用者/所有者)、**g**(所有者的同组成员)、**o**(非所有者及其同组的其他用户)和**a**(所有用户)。访问权限有：**r**(读)、**w**(写)和**x**(执行)。

cp

把文件从某个目录复制到其他地方。

选项	功 能
-i	如果目标文件存在,请求确认
-r	复制目录到新的目录

crypt

按指定密码对指定文件进行加密或解密。加密文件是无法读懂的,必须使用相同密码才能对加密文件解密。

cut

从文件中“切掉”指定的列(或域)。

选项	功 能
-f	指定域位置
-c	指定字符位置
-d	指定域分隔符

date

显示星期、月份、日期和时间。

df

该命令报告指定的文件系统的磁盘空间总数或可用磁盘空间。

.(点)

使用户在当前 shell 环境中运行进程,而不为之创建子进程。

du

该命令可统计每个目录、子目录和文件占用的存储空间。

选项	功 能
-a	显示目录和文件的大小
-s	只显示指定目录占用的存储块数目,不列出子目录

echo

该命令显示(回显)其参数到输出设备。

扩展符	含 义
n	一个回车并生成新行
f	一个制表符
b	一个退格符
r	回车但不产生新行
t	禁止回车

exit

该命令终止当前 shell 程序。它也可以返回一个状态码(RC)来指示程序执行的成功与否。如果用户在 \$提示符下键入该命令,它将终止用户登录 shell 使用户退出系统。

export

向其他 shell 传送一系列变量。

expr

该命令是一个内部操作,提供算术运算和逻辑运算。

find

在目录层次中定位与一组给定的标准相匹配的文件。动作选项指示 find 命令发现文件之后如何操作。

查找选项	描 述
-name 文件名	寻找给定 filename 的文件
-size ± n	寻找大小 n 的文件
-type 文件类型	寻找具有给定访问模式的文件
-atime ± n	寻找 n 天以前访问的文件
-mtime + n	寻找 n 天以前更改的文件
-newer 文件名	寻找比 filename 更近更新的文件

动作选项	描 述
-print	显示每个发现的文件的路径名
-exec command/;	对找到的文件执行 <code>command</code> 命令
-ok command/;	执行命令前请求确认

finger

该命令显示详细的用户信息。

选项	功 能
-b	不在长格式中输出用户主目录和 shell 程序名
-f	不在短格式中输出标题行
-h	不在长格式中输出文件 .project 的内容
-l	强制使用长格式输出
-p	不在长格式中输出文件 .plan 的内容
-s	强制使用短格式输出

grep(Global Regular Expression Print)

在文件中查找指定样式。如果查找成功,在用户终端显示指定样式所在的行。

选项	操 作
-c	只显示每个文件中包含匹配样式的行数
-i	匹配时忽略大小写
-l	只显示包含匹配样式的文件名,不显示具体的行
-n	在输出的每行前显示行号
-v	只显示与样式不匹配的行

head

显示指定文件的头部,这样,用户能很快捷地查看文件的开始几行。可以在命令行中指定行数和多于一个的文件名。

help

该命令提供一系列菜单和问题来引导用户找到最常见的 UNIX 命令的描述。

history(ksh)

该命令是 Korn shell 所特有的。它保存用户会话期间所有键入命令的清单。

kill

该命令终止用户不想要的或不正常的进程。用户必须指定进程 ID。使用进程 ID0 将终止与用户终端有关的所有进程。

learn

该计算机辅助教学程序被安排为一系列教程与课程。它显示教程的菜单,然后由用户选择所希望的教程及课程。

let(ksh)

该命令在 Korn shell 中可用,提供算术运算。

ln

在存在的文件和一个新文件名之间建立新链接,使文件有多于一个的文件名。

lp(line printer)

打印指定的文件。

选项	功 能
-d	在指定的打印机上打印
-m	完成打印请求后,向用户邮箱发邮件
-n	打印指定份数
-s	取消反馈消息
-t	输出的标题页打印指定标题
-w	完成打印请求后,在用户终端显示消息

lpstat(line print status)

该命令用来获得打印请求和打印机状态的信息。

ls(list)

命令用来显示指定目录的内容。

选项	功 能
-a	列出所有文件,包括隐藏文件
-C	以多列的格式列表,按列排序
-F	如果是目录,文件名后加斜杠(/);如果是可执行文件,加星号(*)表示
-l	按照长格式列表,显示文件的详细信息
-m	按页宽列文件,以逗号分隔
-p	如果是目录,文件名后加斜杠(/)
-r	以字母反序列表
-R	循环列出子目录的内容
-s	以文件块为单位显示文件大小
-x	以多列的格式列表,按行排序

mailx

这个软件给用户提供了电子邮件功能。用户可以给系统中的用户发送邮件,而不考虑该用户是否登录。

选项	功能
-f [文件名]	从指定的文件中而不是系统邮箱中读取消息。如果没有指定文件名,就从 mbox 中读取
-H	显示消息头列表
-s 主题	设置邮件的主题

mailx ~ 转义命令

启动 mailx 给其他用户发送邮件时,mailx 处于输入模式,等待用户编辑消息。在这种模式下,使用命令要加上~,称为~转义命令。

命令	功能
~?	显示所有转义符命令列表
~! 命令	在编辑邮件时,可以调用指定的 shell 命令
~e	调用编辑器,编辑器的名字可以由变量 EDITOR 定义。默认为 vi
~p	显示目前正在编辑的消息
~q	退出输入模式,把编辑了部分的消息保存到 dead.letter 文件中
~r 文件名	读取指定的文件,把它的内容插入到消息中
~< 文件名	读取指定的文件(使用重定向符),把它的内容插入到消息中
~<! 命令	执行指定的命令,把它的输出插入到消息中
~v	调用缺省编辑器 vi,或者使用邮件变量 VISUAL,指定为其他的编辑器
~w 文件名	把正在编辑的消息写到指定的文件中

man

该命令显示系统文档信息。

mesg

如果用户不想接收 write 消息,该命令设置为 n。如果接收 write 消息,设置为 y。

mkdir

该命令在工作目录或者其他任何命令中指定的目录下创建一个新的子目录。

选项	功能
-p	在一个命令行中创建多层目录

more

一次一屏显示文件。对于看大文件很有用。

mv

该命令重命名文件,或将文件从一个位置移动到另一位置。

news

这个命令用来看系统中最新的新闻。可以被系统管理员用来通知用户发生的事情。

选项	功 能
- a	显示所有的新闻项,无论是新的还是旧的
- n	仅仅列出新闻名
- s	显示当前新闻的总数

nohup

该命令使得用户退出系统后,后台命令继续执行。

passwd

该命令改变用户登录密码。

paste

一行接一行把文件连接在一起,或者把两个或多个文件的域连到一个新文件。

pg

一次查看一屏文件内容。提示符显示在屏幕底部时,可以输入参数或其他命令。

选项	功 能
- n	不需要回车键完成单字母命令
- s	用反白显示消息和提示符
- num	设置每一屏的行数为 num,默认值是 23 行
- pstr	把提示符:(冒号)改成设定的字符串 str
+ line - num	从设定的 line - num 行开始显示文件
+ /pattern	从包含设定模式的第一处开始显示

Pr

在显示或者打印之前,格式化文件。

选项	功 能
+ page	从指定页开始显示。默认是第 1 页
- columns	用指定的列数显示输出。默认是 1 列
- a	以横跨页面的方式显示输出,每列一行
- d	双倍行距显示输出
- hstr	用指定字符串 str 代替页眉的文件名
- lnumber	用指定的行数设置页长。默认值是 66 行
- m	用多列显示所有指定文件
- p	在每页末暂停,并响铃
- character	用单个指定的字符分割列。如果没有指定字符,那么用 [tab]
- t	取消 5 行页眉和 5 行页脚
- wnumber	用指定字符数设置行宽。默认值为 72

ps(process status)

该命令提交与用户终端有关程序的进程状态报告。

选项	操 作
-a	显示所有活动进程的状态,不仅仅是用户自己的
-f	显示完整信息,包括完整命令行

pwd(print working directory)

这条命令显示用户当前所在目录的绝对路径名,或者任何指定的目录。

r(redo)

这是一个 Korn shell 命令,重复最后一次执行的命令或历史文件中的命令。

read

该命令从输入设备读取输入,并将输入的串存入一个或多个作为命令行参数而指定的变量。

Rm(remove)

该命令删除当前目录或者其他指定目录不再需要的文件。

选项	功 能
-i	删除文件前,请求确认
-r	删除指定的目录及目录中的所有文件和子目录

rmdir(remove)

该命令从用户当前目录或用户指定的其他目录中移去(删除)文件。

选项	操 作
-i	在删除任何文件前要求确认
-r	删除一个指定的目录及其中的所有子目录

set

该命令在输出设备显示环境/shell 变量。unset 命令消除不想要的变量。

sh

该命令调用一个新的 shell。用户可以用该命令运行自己的脚本文件。本章中只介绍了 3 个选项。

选项	操 作
-n	读取命令但不执行
-v	在读取输入时显示 shell 输入行
-x	在执行命令时显示命令行及其参数。该选项多用于程序调试

sleep

该命令使一个进程休眠(等待)指定的时间(按秒计)。

sort

按特定的顺序对文本文件排序。

选项	操 作
-b	忽略前导空格
-d	按字典序排序。忽略标点符号和控制字符
-f	忽略大小写
-n	数字按数值大小排序

spell

该命令可检查指定文档或键盘输入单词中的拼写错误。它只显示在字典文件中没有找到的单词,并不提供更正建议。

选项	功 能
-b	按英国英语进行拼写检查
-v	显示字典中没有的派生单词

stty

用来设置和显示终端属性,支持改变超过 100 种不同的设置,下表列出一些参数。

选项	功 能
echo [-echo]	回显[不回显]输入字符;默认回显
raw [-raw]	禁止[启用]特殊意义的元字符;默认为启用
intr	生成中断信号;通常用[Del]键
erase	(回退)擦除前一个字符;通常用#键
kill	删除整行;通常用@或者[Ctrl-u]键
eof	从终端产生文件结束(eof)信号;通常用[Ctrl-d]键
ek	用#和@键分别复位 erase 和 kill 键
same	用合理的默认值设置终端属性

tail

显示指定文件的尾部。用户能快捷地查看文件内容。选项灵活地设定查看方式。

选项	功 能
b	以文件块为单位来计算
c	以字符为单位计算
l	以行为单位计算

talk

这个命令用来进行端到端的通信。接收方必须已经登录到系统。

tar

该命令将一系列文件存档到磁带上的一个存档文件上。

tee

分离输出。一个副本显示在用户终端上,另一个副本保存在一个文件中。

选项	操作
-a	将输出追加到一个文件中,对已有文件不覆盖
-i	忽略中断。不响应中断信号

test

该命令判断作为其参数的表达式值为真还是为假,并返回相应的值。它使得用户能够判断不同类型的表达式。

time

统计系统执行用户程序时的时间信息,报告指定程序的实际时间、用户态时间和系统态时间。

tput

该命令与包含着终端特性的 terminfo 数据库相连,方便进行终端属性的设置(如粗体、清屏等)。

选项	功 能
bel	回显终端的响铃字符
blink	闪烁显示
bold	粗体显示
clear	清屏
cup r c	把光标移到 r 行 c 列
dim	显示变暗
ed	从光标位置到屏幕底清屏
el	从光标位置到行末清除字符
smso	启动突出模式
rmso	结束突出模式
smul	启动下划线模式
rmul	结束下划线模式
rev	反白显示

trap

设置和复位终端信号。下面的表显示了一些用来控制程序终止的信号。

信号数	信号名	含义
1	挂起	失去终端联系
2	中断	任一中断键被按下
3	退出	任一退出键被按下
9	杀死	发出 kill -9 命令
15	终止	发出 kill 命令

type

该命令告知用户一个指定的命令是一个内部命令还是一个 UNIX 程序。

wall

这个命令一般被系统管理员用来警告大家一些紧急事件。

wc

计算一个文件或者指定的多个文件中的行数、单词数和字符数。

选项	功 能
-l	报告行数
-w	报告单词数
-c	报告字符数

who

该命令可列出当前登录到系统的所有用户的登录名、终端号和登录时间。

选项	功 能
-q	简要信息功能,仅显示各用户名和用户总数
-H	显示各列信息的标题
-b	显示系统的启动日期和时间
-s	只显示用户名、终端号和登录时间

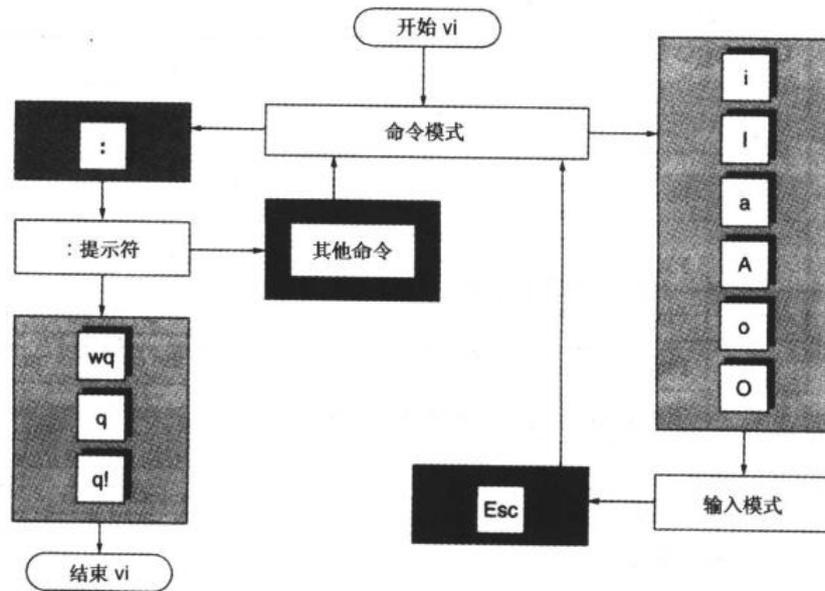
write

这个命令用来进行端到端的通信。接收方必须已经登录到系统。

附录 D vi 命令小结

这部分附录总结了本书中介绍的 vi 编辑器的命令。详细的信息请参见第 4 章和第 6 章。

vi 编辑器的工作模式



vi 编辑器

vi 是一个全屏编辑器, 可用来生成文件。vi 有两种工作模式: 命令模式和文本输入模式。用户键入 vi、空格键、文件名就可以启动 vi, 有几个键可以切换 vi 到文本输入模式, [Esc] 键使得 vi 返回到命令模式。

wq	保存缓冲区中的内容, 退出 vi
w	保存缓冲区中的内容, 但不退出 vi

保存与退出 vi 的命令

除了 ZZ 命令外, 其他的命令都以: 开头, 以 [Return] 结尾。

键	功能
wq	保存文件, 退出 vi
w	保存文件, 但不退出 vi
q	退出编辑器
q!	退出编辑器, 同时放弃所作的修改
ZZ	保存文件, 退出 vi

模式切换键

这些键把 vi 从命令工作模式切换到文本输入模式。不同的键使 vi 进入不同的文本输入模式, [Esc] 键使 vi 返回到命令工作模式。

命令键	功 能
i	在光标左侧输入正文
I	在光标所在行的开头输入正文
a	在光标右侧输入正文
A	在光标所在行的末尾输入正文
o	在光标所在行的下一行增添新行, 并且光标位于新行的开头
O	在光标所在行的上一行增添新行, 并且光标位于新行的开头

光标移动命令

这些键只在命令模式下使用。

键	功 能
h 或 [左箭头]	把光标左移一个空格
j 或 [下箭头]	把光标下移一行
k 或 [上箭头]	把光标上移一行
l 或 [右箭头]	把光标右移一个空格
\$	把光标移到当前行的末尾
w	右移光标, 到下一个字的开头
b	左移光标, 到前一个字的开头
e	右移光标, 到一个字的末尾
0	数字 0, 左移光标, 到本行的开头
[回车键]	移动光标到下一行的开头
[空格键]	把光标右移一个空格
[回退键]	把光标左移一个空格

搜索命令

这些键允许用户在文件中向前或者向后搜索指定的字符串。

键	功 能
/	从指定位置向后开始搜索指定的字符
?	从指定位置向前开始搜索指定的字符

剪切和粘贴键

这些键用来重新安排用户文件中的文本。这些键在 vi 命令模式下可用。

键	功 能
d	删除一段指定的文本, 并将它们保存在临时缓冲区中, 这些缓冲区可利用放置运算操作符

y	复制一段指定的文本到临时缓冲区中,这些缓冲区可利用放置操作符访问
p	将临时缓冲区中的内容放置到光标的前面位置
P	将临时缓冲区中的内容放置到光标的后面位置

域控制键

使用 vi 命令结合域控制键使用户在进行编辑任务中具有更多控制能力。

键	功 能
\$	域为从光标位置到所在行尾
0(零)	域为从光标前一个位置到所在行首
e	域为从光标位置到所在字的尾部
b	域为从光标前一个位置到所在字的头部

修改文本的键

这些键仅适用于命令模式。

键	功 能
x	从指定位置开始删除字符
dd	从指定位置删除行
u	放弃最近的修改
U	放弃对当前行做的所有修改
r	替换当前光标所在的字符
R	从当前光标的位置开始替换字符,并且使 vi 进入文本输入模式
.(点)	重复上次的修改

翻页键

翻页键用来一次滚动多行文本。

键	功 能
[Ctrl-d]	使光标向文件尾移动,通常一次 12 行
[Ctrl-u]	使光标向文件头移动,通常一次 12 行
[Ctrl-f]	使光标向文件尾移动,通常一次 24 行
[Ctrl-b]	使光标向文件头移动,通常一次 24 行

设置 vi 环境

用户可以通过设置 vi 环境选项来定义 vi 编辑器的行为。用户使用 set 命令改变选项的值。

选项	缩写	功 能
autoindent	ai	将新行与前一行的开始对准
ignorecase	ic	在搜索选项下,忽略大小写
magic	-	在搜索时,允许使用特殊字符
number	nu	显示行号
report	-	告知用户,用户最后一个命令作用行的行号
scroll	-	设定使用[Ctrl-d]命令翻滚的行数
shiftwidth	sw	设定缩进空格数。与 autoindent 一同使用
showmode	smd	在屏幕右角显示 vi 编辑器模式
terse	-	缩短错误信息
wrapmargin	wm	将右边界设为一定的字符个数

附录 E ASCII 表

字符/键	十进制	十六进制	八进制	二进制
CTRL - 1 (空)	0	00	000	0000 0000
CTRL - A	1	01	001	0000 0001
CTRL - B	2	02	002	0000 0010
CTRL - C	3	03	003	0000 0011
CTRL - D	4	04	004	0000 0100
CTRL - E	5	05	005	0000 0101
CTRL - F	6	06	006	0000 0110
CTRL - G (响铃)	7	07	007	0000 0111
CTRL - H (回退)	8	08	010	0000 1000
CTRL - I (制表)	9	09	011	0000 1001
CTRL - J (新行)	10	0A	012	0000 1010
CTRL - K	11	0B	013	0000 1011
CTRL - L	12	0C	014	0000 1100
CTRL - M(回车)	13	0D	015	0000 1101
CTRL - N	14	0E	016	0000 1110
CTRL - O	15	0F	017	0000 1111
CTRL - P	16	10	020	0001 0000
CTRL - Q	17	11	021	0001 0001
CTRL - R	18	12	022	0001 0010
CTRL - S	19	13	023	0001 0011
CTRL - T	20	14	024	0001 0100
CTRL - U	21	15	025	0001 0101
CTRL - V	22	16	026	0001 0110
CTRL - W	23	17	027	0001 0111
CTRL - X	24	18	030	0001 1000
CTRL - Y	25	19	031	0001 1001
CTRL - Z	26	1A	032	0001 1010
CTRL - [(ESCAPE)	27	1B	033	0001 1011
CTRL - \	28	1C	034	0001 1100
CTRL -]	29	1D	035	0001 1101
CTRL - ^	30	1E	036	0001 1110
CTRL - _	31	1F	037	0001 1111

续表

字符/键	十进制	十六进制	八进制	二进制
SP (空格)	32	20	040	0010 0000
! (惊叹号)	33	21	041	0010 0001
" (双引号)	34	22	042	0010 0010
# (数字符)	35	23	043	0010 0011
\$ (美元符)	36	24	044	0010 0100
% (百分号)	37	25	045	0010 0101
& (and 符)	38	26	046	0010 0110
' (单引号)	39	27	047	0010 0111
((左括号)	40	28	050	0010 1000
) (右括号)	41	29	051	0010 1001
* (星号)	42	2A	052	0010 1010
+ (加号)	43	2B	053	0010 1001
, (逗号)	44	2C	054	0010 1100
- (连字符/减号)	45	2D	055	0010 1101
. (句号)	46	2E	056	0010 1110
/ (斜杠)	47	2F	057	0010 1111
0	48	30	060	0011 0000
1	49	31	061	0011 0001
2	50	32	062	0011 0010
3	51	33	063	0011 0011
4	52	34	064	0011 0100
5	53	35	065	0011 0101
6	54	36	066	0011 0110
7	55	37	067	0011 0111
8	56	38	070	0011 1000
9	57	39	071	0011 1001
:(冒号)	58	3A	072	0011 1010
;(分号)	59	3B	073	0011 1011
< (小于号)	60	3C	074	0011 1100
= (等号)	61	3D	075	0011 1101
> (大于号)	62	3E	076	0011 1110
? (问号)	63	3F	077	0011 1111
@ (at 符号)	64	40	100	0100 0000
A	65	41	101	0100 0001
B	66	42	102	0100 0010
C	67	43	103	0100 0011
D	68	44	104	0100 0100

续表

字符/键	十进制	十六进制	八进制	二进制
E	69	45	105	0100 0101
F	70	46	106	0100 0110
G	71	47	107	0100 0111
H	72	48	110	0100 1000
I	73	49	111	0100 1001
J	74	4A	112	0100 1010
K	75	4B	113	0100 1011
L	76	4C	114	0100 1100
M	77	4D	115	0100 1101
N	78	4E	116	0100 1110
O	79	4F	117	0100 1111
P	80	50	120	0101 0000
Q	81	51	121	0101 0001
R	82	52	122	0101 0010
S	83	53	123	0101 0011
T	84	54	124	0101 0100
U	85	55	125	0101 0101
V	86	56	126	0101 0110
W	87	57	127	0101 0111
X	88	58	130	0101 1000
Y	89	59	131	0101 1001
Z	90	5A	132	0101 1010
[(左方括号)	91	5B	133	0101 1011
\(反斜杠)	92	5C	134	0101 1100
](右方括号)	93	5D	135	0101 1101
^(发音符)	94	5E	136	01011110
_(加重符)	95	5F	137	0101 1111
'(反单引号)	96	60	140	0110 0000
a	97	61	141	0110 0001
b	98	62	142	0110 0010
c	99	63	143	0110 0011
d	100	64	144	0110 0100
e	101	65	145	0110 0101
f	102	66	146	0110 0110
g	103	67	147	0110 0111
h	104	68	150	0110 1000

续表

字符/键	十进制	十六进制	八进制	二进制
i	105	69	151	0110 1001
j	106	6A	152	0110 1010
k	107	6B	153	0110 1011
l	108	6C	154	0110 1100
m	109	6D	155	0110 1101
n	110	6E	156	0110 1110
o	111	6F	157	0110 1111
p	112	70	160	0111 0000
q	113	71	161	0111 0001
r	114	72	162	0111 0010
s	115	73	163	0111 0011
t	116	74	164	0111 0100
u	117	75	165	0111 0101
v	118	76	166	0111 0110
w	119	77	167	0111 0111
x	120	78	170	0111 1000
y	121	79	171	0111 1001
z	122	7A	172	0111 1010
{(左大括号)	123	7B	173	0111 1011
(垂直线)	124	7C	174	0111 1100
}(右大括号)	125	7D	175	0111 1101
~(波浪号)	126	7E	176	0111 1110
DEL(删除键)	127	7F	177	0111 1111

附录 F 参考文献

进一步阅读以下文献将有助于精通 UNIX。

A Practical Guide to the Unix System V, second edition, by Mark G. Sobell (The Benjamin/Cummings Publishing Company, Inc., ISBN 0-8053-7560-0).

UNIX for Programmers and Users: A Complete Guide by Graham Glass (Prentice-Hall, Inc., ISBN 0-13-480880-0).

LIFE WITH UNIX: A Guide For Everyone by Done Libes and Sandy Ressler (Prentice-Hall, Inc., ISBN 0-13-536657-7).

UNIX POWER TOOLS by Jerry Peek, Tim O'Reilly, and Mike Loukides (O'Reilly & Associates, Inc., ISBN 0-553-35402-7).

UNIX System V Bible: Commands and Utilities by The Waite Group Stephen Prata and Donald Martin (SAMS, ISBN 0-672-22562-X).

UNIX System Architecture by Prabhat K. Andleigh (Prentice-Hall, Inc., ISBN 0-13-949843-5).

UNIX Shell Programming, revised edition, by Stephen G. Kochan and Patrick H. Wood (Hayden Books, 06-672-48448-X).

UNIX Application Programming: Mastering the Shell by Ray Swart (SAMS, ISBN 0-672-22715-0).

UNIX Made Easy, second edition, by John Muster and Associates (McGraw-Hill, ISBN 0-07-882173-8).