

LINUX



- ❖ 344个命令，覆盖Linux系统常用命令，每个命令均配以实例进行讲解
- ❖ 实例讲解Vim、Emacs等编译器及其相关命令的使用
- ❖ 25个典型Shell程序，完全掌握Shell编程的精髓

命令、编辑器、Shell编程 实例大全

杨明华 谭励 于重重 编著



Linux



人民邮电出版社
POSTS & TELECOM PRESS

TP316.81
237

Linux

命令、编辑器、Shell编程 实例大全

杨明华 谭励 于重重 编著

Linux

人民邮电出版社
北京

图书在版编目 (C I P) 数据

Linux命令、编辑器、Shell编程实例大全 / 杨明华,
谭励, 于重重编著. —北京: 人民邮电出版社, 2009.4
ISBN 978-7-115-19468-8

I. L… II. ①杨…②谭…③于… III. Linux操作系统
IV. TP316.89

中国版本图书馆CIP数据核字 (2008) 第211007号

内 容 提 要

本书全面、系统地介绍了 Linux 命令、编辑器及 Shell 编程的相关内容。既便于读者深入了解 Linux 命令强大的功能, 又可以帮助 Linux 用户在较短的时间内学习和掌握 Shell 编程, 开发出自己需要的实用程序。全书分上、中、下 3 篇, 共 20 章, 内容涵盖了 Linux 简介、Red Hat Linux 基础知识、系统管理与设置、用户和用户组管理、磁盘管理、文件和目录管理、备份与压缩、网络管理、正则表达式、vim 编辑器、emacs、grep 命令、awk 与 sed、常用的文本编辑命令、Shell 编程基础、Shell 变量、条件测试表达式、Shell 的控制结构、函数以及 Shell 编辑实例。

本书内容丰富, 语言通俗易懂, 叙述深入浅出, 适合于各层次 Linux 用户。既可以作为各类院校相关专业的教材, 也可作为 Linux 培训的教程及广大 Linux 爱好者的专业参考书。

Linux 命令、编辑器、Shell 编程实例大全

- ◆ 编 著 杨明华 谭 励 于重重
责任编辑 屈艳莲
责任编辑 黄 焱
- ◆ 人民邮电出版社出版发行 北京市崇文区夕照寺街 14 号
邮编 100061 电子函件 315@ptpress.com.cn
网址 <http://www.ptpress.com.cn>
北京鑫正大印刷有限公司印刷
- ◆ 开本: 787×1092 1/16
印张: 51
字数: 1 207 千字 2009 年 4 月第 1 版
印数: 1—4 000 册 2009 年 4 月北京第 1 次印刷

ISBN 978-7-115-19468-8/TP

定价: 89.00 元

读者服务热线: (010)67132692 印装质量热线: (010)67129223
反盗版热线: (010)67171154

前 言

Linux 是一种开源的操作系统，因其开源性而得到广大开发者的青睐。经过十几年的发展，Linux 已经拥有了大量用户并在政务、军事、商业等众多领域得到了广泛的应用。与 Windows 相比，Linux 具有更好的安全性、稳定性以及更高的执行效率和更低的价格。

如今，学习 Linux 已经成为一种潮流，而学习 Linux 命令并掌握 Shell 编程是一名 Linux 初学者成长为中、高级 Linux 用户的标志。为了满足众多 Linux 初学者、爱好者及专业人员的使用需要，笔者在多年从事 Linux 研究、教学及开发的基础上，精心编写了本书。本书遵循命令、编辑器、Shell 编程这一知识脉络组织各章节内容，各知识点前后贯穿，循序渐进，依次进阶。在 Linux 命令的讲解中融入了基础知识的解析及相关应用示例，尽量避免手册、指南式的枯燥讲解；在 Shell 编程部分通过对代码的逐行剖析及执行范例的讲评，使读者能够深入掌握 Shell 编辑技巧，达到举一反三、触类旁通的目的。

为了降低学习和理解的难度，本书共分为 3 个独立的部分，共 20 章。各个部分既前后铺垫又自成体系，适于读者按主题进行跳跃式阅读。同时，本书在命令讲解及 Shell 编程中附有大量实例，便于读者边学习边实践，在实践中体会要领，达到学习、实践、应用一气呵成。

和其他书籍相比，本书有何特点

1. 示例教学，寓学于练

本书力图通过实践提高读者的水平，在讲解每个命令时均配有示例说明，对每一 Shell 程序也给出了详细的代码及执行示例。代码讲解和注释精细到行，读者只需按步阅读、操作即可快捷地提高编程水平。

2. 由浅入深，循序渐进

本书充分考虑读者的学习规律，从最基本的 Linux 命令入手，由浅入深地展开知识脉络。在完成对编辑器的学习之后，过渡到 Shell 编程，最后达到能够熟练编写 Shell 程序的目的。

3. 命令详尽，环境多样

本书涉及了 Linux 中绝大部分常用的命令，对于一些不太常用但是对于读者来说有一定使用价值的命令，本书也进行了较详细的介绍。对于一些已经不再使用的命令或已经被其他命令取代的命令，本书不再介绍。本书以 Red Hat Linux 9 为平台，但其中绝大部分内容也适用于其他 Linux 版本，包括 Red Hat Enterprise Linux、Fedora Core 等，但是由于运行环境不同，得到的输出结果可能与本书的例子略有不同。

4. 内容全面，重点突出

本书内容丰富，覆盖面广，内容从基本的系统命令扩展到 Linux 系统管理、网络配置等诸多方面，每一方面的阐述又从多个角度进行了延伸。对于重点、难点则给出常见问题的分析。



5. 结构合理，适用面广

本书在章节的编排和内容的深度、广度设置方面，能够满足大多数 Linux 爱好者学习和使用的需要。

6. 语言简练，通俗易懂

本书尽量避免冗长的知识点讲解，采用概念和实例相结合的方法，在实例中验证概念，在概念中拓展实例。每个知识点都以通俗易懂的语言阐述，力求深入浅出。

本书包括的内容

本书共分上、中、下 3 篇。上篇包括第 1 章~第 8 章，重点讲述 Linux 常用的命令；中篇包括第 9 章~第 14 章，重点讲述 Linux 编辑器及相关命令；下篇包括第 15 章~第 20 章，重点讲述 Shell 编程的相关内容。各章节具体编排如下。

第 1 章简要介绍了 Linux 系统的起源和特点，以及 Linux 的各种发行版本。

第 2 章详细介绍了 Red Hat Linux 的发展史、Red Hat Linux 的安装、登录 Linux、Linux 的终端命令窗口、命令行自动补全、退出 Linux 以及联机帮助命令。

第 3 章介绍了常用的信息显示命令、系统管理命令、系统维护命令、自动作业处理以及系统日志相关命令。

第 4 章介绍了普通用户管理、根用户管理、用户组管理、用户和用户组管理的安全防范措施以及用户、用户组管理的综合应用。

第 5 章介绍了 Linux 磁盘分区工具、常用的磁盘管理命令、磁盘配额管理、逻辑卷管理以及磁盘管理命令的综合应用。

第 6 章介绍了文件系统的概念、文件系统的组织方式、文件系统的类型、创建 Linux 文件系统、文件和目录的权限、mount 与 umount 命令以及文件和目录管理命令的综合应用。

第 7 章介绍了 tar 命令、gzip 与 gunzip 命令、zip 与 unzip 命令、bzip2 与 bunzip2 命令、dump 与 restore 命令等。

第 8 章介绍了网络的基本类型、体系结构、网络管理常用命令、FTP 常用命令、UUCP 常用命令以及电子邮件相关命令。

第 9 章介绍了正则表达式，包括元字符、字符集和类、重复、子表达式、分支等。

第 10 章介绍了 vim 编辑器，包括启动与退出、运行模式、常用命令、查找与替换、窗口操作、使用 Shell 命令等。

第 11 章介绍了 emacs 编辑器，包括 emacs 概述、文件编辑、缓冲区和窗口、emacs 的扩展工具。

第 12 章介绍了 grep 命令，包括 grep 与正则表达式、grep 的选项、egrep 与 fgrep 等。

第 13 章介绍了过滤器 gawk 与流编辑器 sed。

第 14 章介绍了一些常用的文本编辑命令，包括 aspell、cat、csplit、ed、head 等。

第 15 章介绍了 Shell 编辑的基础知识，包括 Shell 的分类、Shell 通配符、如何运行 Shell 脚本等。

第 16 章介绍了 Shell 变量，包括变量的类型、变量的赋值、变量的输出、数组变量、算术运算等。

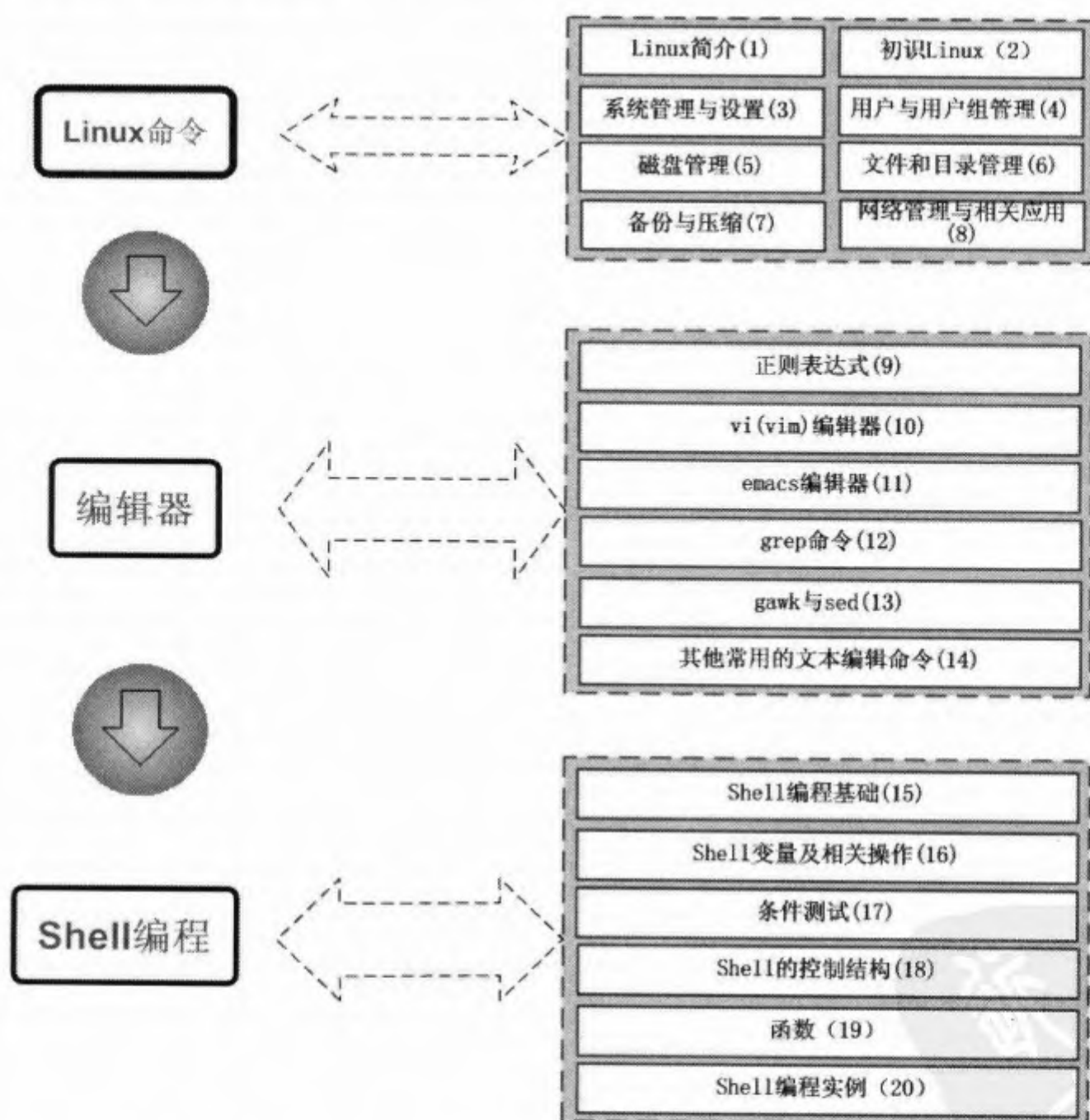
第 17 章介绍了 Shell 的条件测试表达式，包括测试文件属性、测试数值、测试字符串、测试逻辑运算符等。

第 18 章介绍了 Shell 的控件结构，包括 if 结构、case 结构、select 结构、while 循环、for 循环、until 循环以及 break 和 continue 语句。

第 19 章介绍了 Shell 函数，包括函数的定义、函数的调用、函数的作用域、函数的嵌套以及函数的递归。

第 20 章综合运用 Shell 命令编写了 17 个典型的 Shell 程序，包括制作菜单、创建多个用户、对多个目录进行备份、检测系统资源、对数值进行排序等。

各章节具体内容如图所示



适合阅读本书的读者

本书适合于各层次 Linux 用户，既可作为各类院校相关专业的教材，也可作为 Linux 培训的教程及广大 Linux 爱好者的专业参考书。



本书使用的约定

(1) 为了更好地使读者理解本书中涉及的代码和命令，在示例中加入了大量注释。注释以“//”开始。

(2) 对于图形界面中的对话框、文本框、下拉列表框、按钮及下拉菜单等的名称，均使用“【】”加以标注。

本书由杨明华、谭励、于重重组织编写。由于时间仓促，加之编者的水平有限，书中的不足之处在所难免，敬请读者批评指正。本书责任编辑的电子邮箱是 huangyan@ptpress.com.cn，欢迎来信交流。

编 者

2009 年 1 月



目 录

上篇 Linux 命令

第 1 章 Linux 简介	2	终端	25
1.1 Linux 起源及特点	3	2.4.2 终端窗口的常规操作	26
1.1.1 Linux 起源	3	2.5 命令行自动补全	28
1.1.2 Linux 特点及主要优势	3	2.5.1 环境变量名补全	28
1.2 Linux 版本发展	5	2.5.2 用户名补全	28
1.2.1 Linux 内核版本	5	2.5.3 命令、别名或函数的名字 补全	29
1.2.2 Linux 发行版本	6	2.5.4 主机名补全	29
1.3 Linux 与自由软件	8	2.6 退出 Linux	29
1.3.1 GNU 与 FSF	8	2.6.1 shutdown 命令: 关机或 重新启动	29
1.3.2 GPL 与 Copyright	8	2.6.2 halt 命令: 关闭系统	31
第 2 章 初识 Linux	10	2.6.3 reboot 命令: 重新启动 系统	31
2.1 Red Hat Linux 的发展历史	11	2.6.4 exit 命令: 重新登录	32
2.2 Red Hat Linux 的安装	11	2.6.5 logout 命令: 注销用户	32
2.2.1 硬件需求	11	2.7 联机帮助命令 man	32
2.2.2 安装方式	11	2.7.1 命令格式	32
2.2.3 安装过程	12	2.7.2 常规操作	33
2.3 登录 Linux	20	2.7.3 按章节查询	34
2.3.1 Linux 的启动过程	20	2.7.4 man 文件的输出	36
2.3.2 以图形方式登录 Linux	20	2.8 其他联机帮助命令	36
2.3.3 设置文本登录模式	21	2.8.1 apropos 命令: 基于字符串 查找 whatis 数据库	36
2.3.4 以文本方式登录 Linux	23	2.8.2 help 命令: 显示内建命令 的帮助信息	36
2.3.5 login 命令: 登录系统	23	2.8.3 info 命令: 查看 info 格式 帮助信息	38
2.3.6 startx 命令: 启动图形 界面	23	2.8.4 whereis 命令: 查询文件 存储位置	40
2.3.7 xset 命令: 设置 X-windows	24	2.8.5 whatis 命令: 查找简洁的 帮助信息	40
2.4 Linux 的终端命令窗口	25		
2.4.1 在桌面系统中使用虚拟			



第 3 章 系统管理与设置	41
3.1 常用的信息显示命令	42
3.1.1 arch 命令: 显示主机的 体系结构	42
3.1.2 cal 命令: 显示日历	42
3.1.3 cat /proc/meminfo: 查看 内存使用情况	43
3.1.4 cat /proc/modules: 查看 当前系统模块	44
3.1.5 cat /proc/version: 查看当 前系统版本	45
3.1.6 cat /proc/partitions: 查看 当前系统分区	46
3.1.7 cat /etc/issue 命令: 显示 当前系统的发行版本	46
3.1.8 cat /proc/cpuinfo 命令: 显示 CPU 信息	46
3.1.9 cat /proc/interrupts 命令: 显示中断信息	47
3.1.10 cat /proc/filesystems 命令: 显示文件系统信息	47
3.1.11 date 命令: 显示或设置 系统时间	47
3.1.12 dmesg: 显示启动信息	50
3.1.13 echo 命令: 显示文本行	51
3.1.14 free 命令: 显示内存 信息	52
3.1.15 fgconsole 命令: 显示 活动的虚拟终端数目	53
3.1.16 fuser 命令: 用文件或 套接口识别进程	54
3.1.17 gdialog 命令: 从 Shell 创建对话框	55
3.1.18 history 命令: 显示历史 指令	62
3.1.19 hostname 命令: 显示或 设置当前系统的主机名	63
3.1.20 hash 命令: 显示、添加 或清除哈希表	64
3.1.21 hwclock 命令: 显示和 设置硬件时钟	65
3.1.22 ipcs 命令: 显示 IPC 设备 信息	66
3.1.23 isosize 命令: 显示 iso9660 文件系统大小	67
3.1.24 jobs 命令: 显示工作 清单	67
3.1.25 lsmod 命令: 显示所有 已加载的模块	68
3.1.26 locale 命令: 显示当前 系统的语言设置	69
3.1.27 mesg 命令: 显示信息	70
3.1.28 minfo 命令: 显示 ms-dos 文件系统的各项参数	70
3.1.29 modinfo 命令: 显示内核 模块信息	71
3.1.30 mev 命令: 监视鼠标 情况	71
3.1.31 pstree 命令: 以树状图 显示程序	72
3.1.32 pwd 命令: 显示当前 目录	73
3.1.33 proc 文件系统: 查看 系统内核信息	73
3.1.34 ps 命令: 报告程序 状况	74
3.1.35 pmap 命令: 显示程序的 内存信息	77
3.1.36 procinfo 命令: 显示系统 状态信息	77
3.1.37 pwd 命令: 显示当前 目录	79
3.1.38 rulevel 命令: 显示系统 运行级别	80
3.1.39 sysctl 命令: 设置系统	

核心参数	80	3.2.11	gcov 命令: coverage 测试工具	94
3.1.40	stat 命令: 显示文件/文件 系统的详细信息	81	3.2.12	init 命令: 所有进程的 父进程
3.1.41	top 命令: 显示进程 信息	82	3.2.13	ipcrm 命令: 删除消息队 列、信号量或共享内存的 ID
3.1.42	time 命令: 显示进程 执行时间	83	3.2.14	kill 命令: 杀死进程
3.1.43	tload 命令: 显示系统 负载	84	3.2.15	killall 命令: 杀死同名的 所有进程
3.1.44	uptime 命令: 显示系统 的运行信息	84	3.2.16	logrotate 命令: 处理 log 文件
3.1.45	uname 命令: 显示操作 系统信息	85	3.2.17	mtools: 软盘操作 命令集
3.1.46	vmstat 命令: 显示虚拟 内存的统计信息	85	3.2.18	nice 命令: 设定进程 优先级
3.1.47	yes 命令: 连续输出给定 的字符串	86	3.2.19	nohup 命令: 退出系统 继续执行命令
3.2	常用的系统管理命令	87	3.2.20	ntsysv 命令: 设置系统的 各种服务
3.2.1	apm 命令: 高级电源 管理	87	3.2.21	open 命令: 开启虚拟 终端
3.2.2	apmd 命令: 高级电源 管理守护进程	88	3.2.22	pgrep 命令: 查找符合 条件的进程
3.2.3	apmsleep 命令: 设置挂起、 待机及唤醒	89	3.2.23	pidof 命令: 显示当前正 在运行程序的进程 ID
3.2.4	bg 命令: 将程序放在后台 执行	89	3.2.24	pkill 命令: 杀死进程
3.2.5	chsh 命令: 改变登录系统 时的 Shell	90	3.2.25	pwck 命令: 检查密码 文件
3.2.6	clear 命令: 清除终端 屏幕	91	3.2.26	rlogin 命令: 远端登录
3.2.7	depmod 命令: 检查模块 依赖关系	91	3.2.27	renice 命令: 对进程 优先级进行调整
3.2.8	exec 命令: 执行完命令后, 交出控制权	92	3.2.28	rmmod 命令: 删除 模块
3.2.9	fc 命令: 修改并执行 命令	92	3.2.29	rsh 命令: 远端登录的 Shell
3.2.10	fg 命令: 将后台任务 放到前台	93	3.2.30	screen 命令: 多重视窗 管理程序



- 3.2.31 sleep 命令: 休眠113
- 3.2.32 sudo 命令: 以其他身份
来执行命令.....114
- 3.2.33 suspend 命令: 暂停执行
Shell115
- 3.2.34 symlinks 命令: 管理和
维护符号链接.....115
- 3.2.35 vlock 命令: 锁定终端....116
- 3.2.36 wait 命令: 等待程序返回
状态.....116
- 3.2.37 watch 命令: 重复执行
命令并将结果输出116
- 3.2.38 &命令: 将任务放到
后台执行.....118
- 3.3 常用的系统维护命令.....118
 - 3.3.1 alias 命令: 设置命令的
别名118
 - 3.3.2 aumix 命令: 设置音效
装置.....119
 - 3.3.3 authconfig 命令: 配置
系统的认证资源.....121
 - 3.3.4 bind 命令: 显示或设置
按键组合.....124
 - 3.3.5 chkconfig 命令: 设置和
检查系统的服务设置124
 - 3.3.6 declare 命令: 显示或
设置 Shell 变量126
 - 3.3.7 dircolors 命令: 设置 ls
命令显示的颜色.....128
 - 3.3.8 enable 命令: 开动或关闭
Shell 内置命令131
 - 3.3.9 eval 命令: 连续执行多个
命令.....134
 - 3.3.10 export 命令: 设置或显示
环境变量.....134
 - 3.3.11 false 命令: 返回值为
失败.....137
 - 3.3.12 fbset 命令: 显示或
设置帧缓存137
 - 3.3.13 hostid 命令: 显示出当前
主机的数字标识139
 - 3.3.14 insmod 命令: 载入
模块139
 - 3.3.15 ldconfig 命令: 配置
动态链接绑定141
 - 3.3.16 ldd 命令: 显示共享库
文件的相互依赖关系...143
 - 3.3.17 lilo 命令: 安装启动装载
程序144
 - 3.3.18 losetup 命令: 设置和
控制循环设备145
 - 3.3.19 mkraid 命令: 初始化
或升级 raid.....145
 - 3.3.20 modprobe 命令: 自动
处理可载入模块.....146
 - 3.3.21 mouseconfig 命令: 设置
鼠标相关参数147
 - 3.3.22 resize 命令: 设置终端
视窗的大小149
 - 3.3.23 rpm 命令: 管理
RPM 包149
 - 3.3.24 setserial 命令: 设置或
显示串口的信息153
 - 3.3.25 setup 命令: 设置公用
程序154
 - 3.3.26 swapoff 命令: 卸载交换
区 (swap area)157
 - 3.3.27 swapon 命令: 加载交换
区 (swap area)158
 - 3.3.28 set 命令: 设置 Shell 执行
方式158
 - 3.3.29 telinit 命令: 设置系统的
运行等级160
 - 3.3.30 true 命令: 返回值为
成功160
 - 3.3.31 ulimit 命令: 控制 Shell

及进程的资源	161	文件	183
3.3.32 unalias 命令: 取消 指定的别名	162	4.1.4 pwunconv 命令: 从影子 文件恢复密文	184
3.3.33 unset 命令: 删除变量 或函数	163	4.1.5 useradd 命令: 添加新 用户	185
3.3.34 up2date 命令: 升级 Red Hat Linux 系统	163	4.1.6 usermod 命令: 修改用户 的账号	188
3.4 自动作业处理	166	4.1.7 userdel 命令: 删除用户 ..	189
3.4.1 crontab 命令: 设置定时 执行	166	4.1.8 passwd 命令: 修改用户 密码	189
3.4.2 anacron 命令: 任务调度 守护进程	169	4.1.9 newusers 命令: 批量 创建用户	191
3.4.3 at 命令: 系统将执行的 任务排队	170	4.1.10 chpasswd 命令: 批量 创建用户密码	192
3.4.4 atd 命令: 执行已经排队的 任务	172	4.1.11 chage 命令: 查看账户 密码的有效期	192
3.4.5 atq 命令: 检查排队的 任务	172	4.2 根用户的管理	193
3.4.6 atrm 命令: 删除已经 排队的任务	172	4.2.1 su 命令: 变更用户身份 ..	193
3.4.7 atrun 命令: 执行已排队 的任务	173	4.2.2 以单用户模式解决 root 密码丢失问题	194
3.4.8 batch 命令: 批处理	173	4.2.3 以救援模式解决 root 密码 丢失问题	196
3.5 系统日志	174	4.2.4 chroot 命令: 改变根 目录	198
3.5.1 syslogd 命令: 启动 syslog 日志进程	174	4.3 用户组的管理	198
3.5.2 syslog.conf: 系统日志 配置文件	175	4.3.1 解析/etc/group 与 /etc/gshadow 文件	198
3.5.3 logger 命令: 测试 syslog.conf	177	4.3.2 groupadd 命令: 添加 新用户组	202
3.5.4 使用 echo 命令清空 日志	177	4.3.3 groupmod 命令: 修改用户 组属性	204
第 4 章 用户和用户组管理	178	4.3.4 groupdel 命令: 删除 用户组	204
4.1 普通用户的管理	179	4.3.5 newgrp 命令: 切换所属 用户组	205
4.1.1 解析/etc/passwd 文件	179	4.4 用户和用户组管理的安全 防范措施	205
4.1.2 解析/etc/shadow 文件	181	4.4.1 ac 命令: 统计用户连接	
4.1.3 pwconv 命令: 创建影子			



时间.....	205	导致用户密码丢失.....	219
4.4.2 last 命令: 查询最近的 使用者登录时间.....	206	第 5 章 磁盘管理.....	220
4.4.3 lastb 命令: 显示最近登录 系统失败的用户信息.....	208	5.1 Linux 磁盘分区工具.....	221
4.4.4 who 命令: 查询当前系统 中登录的用户.....	209	5.1.1 Linux 磁盘分区简介.....	221
4.4.5 w 命令: 查看登录用户及 其进行的操作.....	210	5.1.2 fdisk 命令: 磁盘分区 工具.....	223
4.4.6 使用 userdel 和 groupdel 命令删除不需要的用户和 用户组.....	212	5.1.3 fdisk 命令交互模式.....	224
4.5 其他用户和用户组管理的 常用命令.....	213	5.1.4 使用 fdisk 划分磁盘分区 实例.....	228
4.5.1 finger 命令: 查找并显示 用户信息.....	213	5.1.5 parted 命令: 磁盘分区 工具.....	231
4.5.2 groups 命令: 显示指定用 户所属的用户组.....	214	5.1.6 parted 命令交互模式.....	231
4.5.3 id 命令: 显示用户的 UID、 GID.....	214	5.1.7 sfdisk 命令: Linux 分区 工具.....	234
4.5.4 logname 命令: 显示登录 用户的用户名.....	215	5.2 常用的磁盘管理命令.....	235
4.5.5 newgrp 命令: 修改用户的 用户组.....	215	5.2.1 badblocks 命令: 检查设备 坏块.....	235
4.5.6 rwho 命令: 查看登录本地 的用户信息.....	216	5.2.2 du 命令: 统计文件和目录 占用的磁盘空间.....	236
4.5.7 users 命令: 显示当前登录 的用户.....	216	5.2.3 df 命令: 显示文件系统 磁盘空间使用情况.....	237
4.5.8 write 命令: 将信息实时传 递给登录的用户或终端.....	216	5.2.4 fdformat 命令: 格式化 磁盘.....	239
4.5.9 whoami 命令: 显示当前 用户的信息.....	217	5.2.5 hdparm 命令: 显示和设定 硬盘参数.....	239
4.6 用户及用户组管理命令的综合 应用.....	217	5.2.6 mkswap 命令: 设置交换 分区 (swap area).....	241
4.6.1 解决/etc/nologin 文件引起的 普通用户无法登录问题.....	217	5.2.7 mkisofs 命令: 建立 ISO9660/JOLIET/HFS 文件系统.....	242
4.6.2 创建用户共享目录.....	218	5.2.8 mkinitrd 命令: 创建 ramdisk 的镜像文件.....	242
4.6.3 对/etc/shadow 文件的编辑		5.2.9 mt 命令: 磁带驱动 操作.....	243
		5.2.10 mzip 命令: 弹出 zip/jaz 设备中的磁盘.....	244
		5.2.11 sync 命令: 将缓冲区内	

的数据写入磁盘	244	6.4 文件系统的类型	260
5.2.12 rsync 命令: 远程同步	245	6.5 文件系统的组成	262
5.3 磁盘配额管理	246	6.6 文件的类型	263
5.3.1 磁盘配额的系统配置	247	6.7 创建 Linux 文件系统	264
5.3.2 edquota 命令: 对用户或 用户组设置磁盘配额	247	6.7.1 mkfs 命令: 创建文件 系统	264
5.3.3 quota 命令: 显示磁盘已使 用的空间与配额限制	249	6.7.2 mkbootdisk 命令: 建立 目前系统的启动盘	265
5.3.4 quotacheck 命令: 检查 磁盘的使用空间与限制	249	6.8 文件和目录的权限	266
5.3.5 quotaoff 命令: 关闭磁盘 空间限制	251	6.8.1 权限的常规表示	266
5.3.6 quotaon 命令: 启动磁盘 空间配额限制	252	6.8.2 chmod 命令: 权限设置 与修改	268
5.3.7 quotastats 命令: 显示磁 盘空间的限制	252	6.8.3 使用 chomd 命令设置 特殊权限	270
5.3.8 repquota 命令: 对文件 系统的磁盘配额情况进 行统计	253	6.8.4 umask 命令: 设置文件 或目录的默认权限	271
5.4 逻辑卷管理 (LVM)	253	6.8.5 setfacl 命令: 设置访问 控制列表 ACL	272
5.4.1 pvdisplay 命令: 查看组成 LVM 卷的物理卷	254	6.9 修复受损的文件系统	274
5.4.2 pvcreate 命令: 创建 物理卷	254	6.9.1 fsck 命令: 检测文件系统 并尝试修复错误	274
5.4.3 vgcreate 命令: 创建 卷组	254	6.9.2 e2fsck 命令: 检查 ex2 文件系统	275
5.4.4 vgextend 命令: 在卷组里 添加一个新的物理卷	254	6.10 mount 与 umount 命令	277
5.4.5 lvcreate 命令: 创建一个 逻辑卷	254	6.10.1 mount 命令格式	277
5.5 磁盘管理命令的综合应用	255	6.10.2 umount 命令格式	278
5.5.1 添加新磁盘	255	6.10.3 软盘的挂载与卸载	278
5.5.2 无法使用全部硬盘空间	256	6.10.4 CDROM 的挂载与 卸载	278
第 6 章 文件和目录管理	257	6.10.5 USB 存储设备的挂载 与卸载	279
6.1 文件系统的概念	258	6.11 其他文件和目录管理常用 命令	280
6.2 文件系统的组织方式	258	6.11.1 文件和目录操作中常用 的通配符	280
6.3 系统的默认安装目录	258	6.11.2 attr 命令: XFS 文件系统 对象的扩展属性	281
		6.11.3 basename 命令: 显示文 件名	281



- 件或目录的基本名称...281
- 6.11.4 cd 命令: 改变目录.....282
- 6.11.5 cp 命令: 复制文件.....283
- 6.11.6 chattr 命令: 改变文件的属性.....285
- 6.11.7 cksum 命令: 文件的 CRC 校验.....287
- 6.11.8 chown 命令: 改变所有者.....287
- 6.11.9 chgrp 命令: 更换用户组.....289
- 6.11.10 dirname 命令: 显示文件的除名字外的路径290
- 6.11.11 fold 命令: 自动换行以适应指定列宽.....291
- 6.11.12 find 命令: 查找目录或文件.....292
- 6.11.13 file 命令: 显示文件类型.....293
- 6.11.14 filterdiff 命令: 从 diff 文件中提取或去除差异.....294
- 6.11.15 findfs 命令: 通过卷标或 UUID 查找文件系统295
- 6.11.16 locate 命令: 定位文件或目录.....295
- 6.11.17 ls 或 dir 命令: 列目录...296
- 6.11.18 ln 命令: 链接文件或目录.....300
- 6.11.19 lndir 命令: 建立到目录的符号链接.....302
- 6.11.20 lsattr 命令: 显示文件属性.....303
- 6.11.21 mkfontdir 命令: 创建 x 字体文件目录.....303
- 6.11.22 msgcat 命令: 合并消息.....304
- 6.11.23 msgcomm 命令: 匹配消息.....305
- 6.11.24 msgen 命令: 创建英语消息目录.....306
- 6.11.25 msginit 命令: 初始化消息目录.....307
- 6.11.26 mkdir 命令: 创建目录...307
- 6.11.27 mv 命令: 移动或更名现有的文件或目录.....308
- 6.11.28 mattrib 命令: 修改或显示 MS-DOS 文件的属性.....310
- 6.11.29 mc 命令: 可视化文件管理程序.....310
- 6.11.30 mcopy 命令: 复制 MS-DOS 文件到 UNIX 或从 UNIX 复制 MS-DOS 文件.....312
- 6.11.31 md5sum 命令: 计算和检测 MD5 信息摘要...312
- 6.11.32 mdel 命令: MS-DOS 文件删除.....313
- 6.11.33 mdir 命令: 显示 MS-DOS 目录.....314
- 6.11.34 mktemp 命令: 建立暂存文件.....314
- 6.11.35 mmove 命令: 移动 MS-DOS 文件.....314
- 6.11.36 mren 命令: 更改 MS-DOS 文件名称.....315
- 6.11.37 mshowfat 命令: 显示分配给文件的 FAT 簇信息...315
- 6.11.38 mtools 命令: 显示 mtools 支持的命令316
- 6.11.39 mtoolstest 命令: 测试并显示 mtools 的相关设置...316
- 6.11.40 paste 命令: 对文件进行纵向合并.....316
- 6.11.41 rcp 命令: 远端复制文件或目录.....317

6.11.42	rm 命令: 删除文件 或目录	318	7.3.1	常规压缩与解压缩操作 ..	346
6.11.43	rmdir 命令: 删除目录 ..	319	7.3.2	查看.gz 压缩包中的 内容	346
6.11.44	slocate 命令: 安全查找 文件或目录	320	7.3.3	自定义压缩包后缀	346
6.11.45	split 命令: 切割文件 ..	321	7.3.4	zcat 命令: 显示压缩 文件内容	347
6.11.46	stat 命令: 显示 inode 内容	323	7.4	zip 与 unzip 命令	347
6.11.47	scp 命令: 远程复制 文件	324	7.4.1	使用 zip 生成压缩包	348
6.11.48	sum 命令: 计算文件的 校验码	327	7.4.2	使用 unzip 进行解压缩 ..	352
6.11.49	tmpwatch 命令: 删除 临时文件	327	7.4.3	zipinfo 命令: 显示压缩 文件信息	353
6.11.50	touch 命令: 更新文件或 目录时间	328	7.5	bzip2 与 bunzip2 命令	356
6.11.51	tree 命令: 以树状图 显示目录内容	329	7.5.1	使用 bzip2 生成压缩包 ..	356
6.11.52	which 命令: 查找 文件	333	7.5.2	使用 bunzip2 进行 解压缩	357
6.12	文件和目录管理命令的综合 应用	333	7.5.3	bzip2recover 命令: 恢复 损坏的 bzip2 文件	359
6.12.1	无法卸载已挂载的文件 系统	334	7.6	dump 与 restore 命令	359
6.12.2	查询设备上采用的未知 文件系统	334	7.6.1	使用 dump 制作备份	360
6.12.3	恢复已删除的文件	334	7.6.2	使用 restore 恢复备份 ..	361
6.12.4	无法引导文件系统	335	7.7	compress 与 uncompress 命令 ..	363
第 7 章	备份与压缩	339	7.7.1	使用 compress 生成压缩 文件	363
7.1	Linux 备份与压缩的策略	340	7.7.2	使用 uncompress 进行 解压缩	364
7.2	tar 命令	341	7.8	其他常用备份与压缩命令	365
7.2.1	打包和解包的常规操作 ..	342	7.8.1	ar 命令: 从归档文件中建 立、修改或提取文件	365
7.2.2	查看 tar 包中的内容	343	7.8.2	cpio 命令: 备份文件	366
7.2.3	打包链接文件	343	7.8.3	dd 命令: 磁盘复制 命令	367
7.2.4	向包中添加新文件	344	7.8.4	使用 cp 命令制作软盘 镜像	367
7.2.5	生成.tar.gz 压缩包	344	7.8.5	使用 cp 命令制作光盘 镜像	368
7.3	gzip 与 gunzip 命令	345	7.8.6	lha 命令: 压缩或解压缩 .lhz 文件	368
			7.8.7	unarj 命令: 解压文件 ..	370



第 8 章 网络管理与相关应用	371
8.1 计算机网络的发展	372
8.2 网络基本类型	373
8.2.1 按地理覆盖范围	373
8.2.2 按拓扑结构	375
8.3 网络体系结构	377
8.3.1 OSI/RM 参考模型	377
8.3.2 TCP/IP 参考模型	379
8.4 网络配置基本内容	382
8.4.1 主机名	382
8.4.2 IP 地址	382
8.4.3 子网与子网掩码 (subnet mask)	385
8.4.4 广播地址 (broadcast address)	385
8.4.5 网关地址 (gateway)	385
8.4.6 域名服务器地址 (DNS)	385
8.4.7 DHCP 服务器	386
8.5 网络管理常用命令	386
8.5.1 apachectl 命令: apache HTTP 服务器控制接口 ..	386
8.5.2 arp 命令: 控制系统 ARP 缓存	387
8.5.3 arpwatch 命令: 监听 ARP 记录	388
8.5.4 arping 命令: 向邻居主机发送 ARP 请求	388
8.5.5 cu 命令: 调用其他主机系统	390
8.5.6 gaim 命令: 即时信息传输客户端	391
8.5.7 hostname 命令	391
8.5.8 host 命令: dns 查询	393
8.5.9 httpd 命令: apache http 服务器程序	395
8.5.10 ifup 命令	396
8.5.11 ifdown 命令	396
8.5.12 ifconfig 命令: 显示或配置网络设备	397
8.5.13 iptables 命令: ipv4 的包过滤和 nat 的管理	400
8.5.14 iptables-save 命令: 保存 ip 表	403
8.5.15 iwconfig 命令: 配置无线网络设备	405
8.5.16 mesg 命令: 控制终端的写入操作	406
8.5.17 mingetty 命令: 精简版的agetty	407
8.5.18 minicom 命令: 串口通信程序	407
8.5.19 mkfifo 命令: 创建管道	408
8.5.20 mtr 命令: 网络诊断工具	409
8.5.21 nc 命令: TCP/UDP 连接与侦听	409
8.5.22 netconfig 命令: 设置各项网络功能	412
8.5.23 netstat 命令: 显示网络状态	413
8.5.24 nslookup 命令: 查询 DNS	417
8.5.25 route 命令: 显示并设置路由	418
8.5.26 ping 命令: 检测主机 (ipv4)	420
8.5.27 ping6 命令: 检测主机 (ipv6)	422
8.5.28 pppd 命令: 点对点连接的守护进程	423
8.5.29 pppstats 命令: 显示 PPP 统计信息	424
8.5.30 rdate 命令: 通过网络	

获取时间	424	8.6.6	tftp 命令: 传输文件	458
8.5.31	smbd 命令: Samba 服务端	8.7	UUCP 常用命令	460
8.5.32	ssh 命令: 远程登录程序 (ssh 客户端)	8.7.1	uucico 命令: UUCP 文件传输守护进程	460
8.5.33	statserial 命令: 显示串口连接状态	8.7.2	uucp 命令: 在 UNIX 系统间传送文件	461
8.5.34	slrn 命令: 新闻阅读程序	8.7.3	uupick 命令: 处理 UUCP 文件	462
8.5.35	talk 命令: 与其他用户交谈	8.7.4	uuto 命令: 文件传送到远端主机	462
8.5.36	tcpdump 命令: 显示网络中的 TCP 数据包	8.7.5	uulog 命令: 显示 uucp 日志信息	463
8.5.37	telnet 命令: 使用 telnet 进行远程登录	8.7.6	uuname 命令: 显示 uucp 远端主机	464
8.5.38	testparm 命令: 测试 samba 配置文件	8.7.7	uustat 命令: 显示 UUCP 状态	464
8.5.39	tracepath 命令: 追踪路径	8.7.8	uux 命令: 通过 UUCP 在远端主机上执行命令	465
8.5.40	traceroute 命令: 显示到达目标网络主机的路由包	8.8	电子邮件	466
8.5.41	tty 命令: 显示标准输入设备名称	8.8.1	fetchmail 命令: 收取邮件	467
8.5.42	wall 命令: 发送信息	8.8.2	mail 命令: 收发邮件	470
8.5.43	wget 命令: 从互联网上下载资源	8.8.3	mailq 命令: 显示发件箱的邮件	474
8.5.44	ytalk 命令: 多用户聊天程序	8.8.4	mutt 命令: Mutt 用户邮件代理	474
8.6	FTP 常用命令	8.8.5	pine 命令: 收发邮件	477
8.6.1	ftp 命令: 文件传输协议	8.9	利用常用命令分析局域网连通故障	480
8.6.2	ncftp 命令: 传输文件			
8.6.3	ncftpget 命令: 下载文件			
8.6.4	ncftpls 命令: 显示文件目录			
8.6.5	ncftpput 命令: 上传文件			

中篇 编辑器

第 9 章	正则表达式	482
9.1	正则表达式介绍	483
9.2	元字符	483
9.2.1	元字符的定义	483
9.2.2	元字符的应用示例	484
9.3	字符集和类	486



9.3.1 字符集和类的定义.....	486	10.4.12 查找字符命令.....	506
9.3.2 元字集和类的应用示例...	487	10.4.13 读入文件命令.....	506
9.4 重复.....	487	10.4.14 写入文件命令.....	506
9.4.1 重复的定义.....	488	10.4.15 定位到指定行命令.....	507
9.4.2 重复的应用示例.....	488	10.4.16 复制命令.....	507
9.5 子表达式.....	489	10.4.17 粘贴命令.....	507
9.5.1 子表达式的定义.....	489	10.4.18 重复命令.....	507
9.5.2 子表达式的应用示例.....	489	10.4.19 显示状态信息命令.....	508
9.6 定位字符串的开始和末尾.....	490	10.4.20 保存和退出命令.....	508
9.6.1 开始和末尾的定义.....	490	10.5 查找与替换字符串.....	509
9.6.2 开始和末尾的应用 示例.....	491	10.5.1 普通查找与增量查找...	509
9.7 分支.....	491	10.5.2 替换字符串.....	509
9.7.1 分支的定义.....	491	10.6 窗口操作.....	510
9.7.2 分支的应用示例.....	492	10.6.1 窗口操作的快捷方式...	510
9.8 匹配特殊字符.....	492	10.6.2 窗口水平拆分.....	511
第 10 章 vi (vim) 编辑器.....	494	10.6.3 窗口垂直拆分.....	512
10.1 vi (vim) 编辑器的历史.....	495	10.6.4 关闭窗口.....	513
10.2 启动与退出 vi (vim).....	496	10.7 使用 Shell 命令.....	513
10.2.1 启动 vi (vim).....	496	10.7.1 启动一个新 Shell.....	513
10.2.2 退出 vi (vim).....	498	10.7.2 在 vi (vim) 中执行 Shell.....	513
10.3 vi (vim) 的运行模式.....	498	10.8 其他常用操作.....	514
10.3.1 命令模式.....	499	10.8.1 屏幕重绘.....	514
10.3.2 插入模式.....	500	10.8.2 滚动屏幕.....	514
10.3.3 末行模式.....	500	10.8.3 建立和使用标记.....	515
10.4 vi (vim) 中的常用命令.....	500	10.9 获得帮助.....	515
10.4.1 插入文本命令.....	501	第 11 章 emacs 编辑器.....	517
10.4.2 附加文本命令.....	501	11.1 emacs 概述.....	518
10.4.3 添加新行命令.....	501	11.1.1 emacs 简介.....	518
10.4.4 光标移动命令.....	501	11.1.2 启动和退出.....	519
10.4.5 删除命令.....	502	11.1.3 新建一个文件.....	521
10.4.6 修改命令.....	503	11.1.4 保存文件.....	522
10.4.7 更改大小写命令.....	504	11.2 文件编辑.....	522
10.4.8 替换命令.....	504	11.2.1 按键的表示与使用.....	522
10.4.9 合并命令.....	505	11.2.2 光标的移动.....	523
10.4.10 句点命令.....	505	11.2.3 输入文本.....	524
10.4.11 撤销命令.....	505	11.2.4 文本的删除.....	524

11.2.5 查找和替换	525	13.1.11 函数	557
11.3 缓冲区和窗口	525	13.1.12 关联数组	557
11.3.1 拆分窗口	526	13.1.13 printf 命令: 格式化 输出	559
11.3.2 同时使用多个缓冲区 进行工作	527	13.1.14 关系运算符	560
11.3.3 在缓冲区中滚动	528	13.1.15 算术运算符	561
11.4 emacs 的扩展工具	528	13.1.16 gawk 的流程控制	563
11.4.1 显示日历	529	13.1.17 getline: 控制输入	565
11.4.2 执行 Shell 命令	529	13.1.18 协进程	566
11.4.3 调用游戏	530	13.1.19 从网络获取数据	567
第 12 章 grep 命令	532	13.1.20 gawk 的应用示例	568
12.1 grep 介绍	533	13.2 流编辑器 sed	569
12.1.1 grep 的含义	533	13.2.1 sed 的基本语法	570
12.1.2 grep 的工作原理	533	13.2.2 sed 编辑器基础	571
12.2 grep 与正则表达式	534	13.2.3 地址	572
12.2.1 grep 的正则表达式元 字符	534	13.2.4 指令	572
12.2.2 grep 正则实例	537	13.2.5 打印: p 指令	573
12.3 grep 的选项	539	13.2.6 删除: d 指令	574
12.4 使用管道的 grep	542	13.2.7 追加: a 指令	574
12.5 grep 的退出状态	542	13.2.8 插入: i 指令	575
12.6 egrep 与 fgrep	543	13.2.9 修改: c 指令	575
12.6.1 egrep	543	13.2.10 退出: q 指令	576
12.6.2 fgrep	545	13.2.11 替换: s 指令	576
第 13 章 gawk 与 sed	546	13.2.12 下一条: n 指令	578
13.1 过滤器 gawk	547	13.2.13 从文件读入: r 指令 ..	579
13.1.1 gawk 概述	547	13.2.14 写入文件: w 指令	579
13.1.2 gawk 的语法	549	13.2.15 sed 的控制结构	580
13.1.3 awk 的语言基础	550	13.2.16 Pattern 区和 Hold 区 ..	580
13.1.4 模式	551	13.2.17 sed 的应用示例	583
13.1.5 动作	553	第 14 章 其他常用的文本编辑命令	585
13.1.6 注释	554	14.1 aspell 命令: 检查文件的拼写 错误	586
13.1.7 变量	554	14.2 cat 命令: 显示文本内容	588
13.1.8 记录和字段	555	14.3 cmp 命令: 比较文件差异	590
13.1.9 输出字段分隔符	556	14.4 csplit 命令: 按指定行数分割 文件	591
13.1.10 重定向输出	556	14.5 cut 命令: 文件剪切	592



14.6	col 命令: 过滤控制字符.....	594
14.7	colrm 命令: 删除指定的列	594
14.8	comm 命令: 对已排序文件 逐行进行比较	595
14.9	diff 命令: 文件比较.....	596
14.10	diffstat 命令: diff 结果的统计 信息.....	599
14.11	ed 命令: 行文本编辑器.....	600
14.12	fmt 命令: 编排文本文件.....	601
14.13	gedit 命令: gnome 的文本 编辑器	602
14.14	head 命令: 从开始显示 指定行	604
14.15	indent 命令: 调整 c 原始代码 文件的格式	605
14.16	jed 命令: 编辑文本文件.....	609
14.17	joe 命令: 编辑文本文件.....	611
14.18	join 命令: 基于相同字段连接 两个文件	615
14.19	less 命令: 前后翻阅文本 内容.....	616
14.20	more 命令: 分屏显示文本 内容.....	617
14.21	od 命令: 输出文件内容	618
14.22	pr 命令: 编排文件格式	620
14.23	patch 命令: 修补文件.....	622
14.24	pico 命令: 编辑文字文件.....	624
14.25	sort 命令: 文件排序.....	626
14.26	spell 命令: 简单的拼写检查 程序.....	629
14.27	tail 命令: 显示文件的末尾 部分.....	629
14.28	tac 命令: 反序输出文件.....	631
14.29	tee 命令: 读取标准输入到 标准输出或文件	632
14.30	tr 命令: 转换文件中的 字符.....	633
14.31	uniq 命令: 检查文件中重复	

出现的行..... 634

14.32 view 命令: 查看文件内容... 635

14.33 wc 命令: 计算文件的字数、
字节数、行数..... 636

下篇 Shell 编程

第 15 章 Shell 编程基础..... 640

15.1 Shell 概述..... 641

15.1.1 Shell 的功能与定义..... 641

15.1.2 Shell 的启动和退出..... 641

15.1.3 Shell 脚本介绍..... 641

15.2 Shell 分类..... 644

15.2.1 Bourne Shell..... 644

15.2.2 Bourne Again Shell 644

15.2.3 C Shell..... 645

15.2.4 Korn Shell..... 645

15.2.5 Perl Shell 645 |

15.3 Shell 基础知识..... 645

15.3.1 区分大小写 645 |

15.3.2 Shell 脚本中的注释..... 646

15.3.3 I/O 重定向..... 646

15.3.4 管道 (Pipe) 649 |

15.3.5 前台与后台 650 |

15.3.6 命令执行顺序 651 |

15.3.7 Shell 通配符..... 652

15.3.8 单引号、双引号及
反引号 653 |

15.3.9 指定使用的 Shell..... 654

15.3.10 运行脚本 655 |

第 16 章 Shell 变量及相关操作..... 658

16.1 Shell 变量类型..... 659

16.1.1 环境变量 659 |

16.1.2 位置变量 660 |

16.1.3 预定义的特殊变量..... 662

16.1.4 用户自定义的变量..... 663

16.2 变量的操作 663 |

14

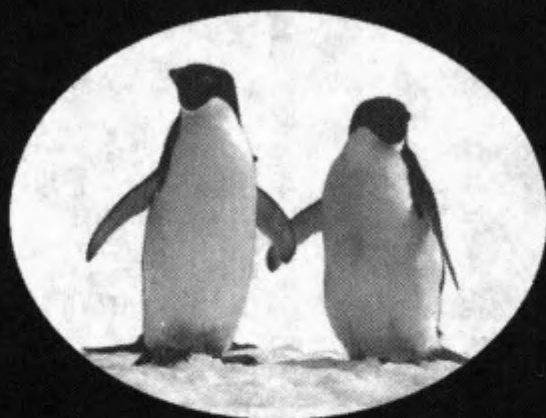
LINUX

16.2.1 创建变量	663	17.3 测试数值	681
16.2.2 区分变量名	664	17.3.1 相等测试	682
16.2.3 删除变量	664	17.3.2 不相等测试	682
16.3 变量的赋值	665	17.3.3 大于测试	683
16.3.1 使用 read 命令赋值	665	17.3.4 大于等于测试	684
16.3.2 直接给变量赋值	667	17.3.5 小于测试	684
16.3.3 使用命令行参数赋值	667	17.3.6 小于等于测试	685
16.3.4 利用命令的输出结果 赋值	668	17.4 测试字符串	686
16.3.5 从文件读入数据实现 赋值	668	17.4.1 相等测试	686
16.4 变量的输出	669	17.4.2 不相等测试	687
16.4.1 echo 命令: 直接输出 变量的值	669	17.4.3 长度为零测试	688
16.4.2 printf 命令: 格式化输出 变量	669	17.4.4 长度不为零测试	688
16.5 数组变量	671	17.4.5 非空测试	689
16.5.1 数组的定义	671	17.5 测试逻辑运算符	690
16.5.2 数组的复制	672	17.5.1 逻辑与测试	690
16.5.3 数组元素的个数	673	17.5.2 逻辑或测试	691
16.5.4 数组元素的长度	674	17.5.3 逻辑非测试	692
16.6 算术运算	675	第 18 章 Shell 的控制结构	693
16.6.1 expr 命令	675	18.1 if 结构	694
16.6.2 let 命令	675	18.1.1 if 结构的语法格式	694
16.6.3 浮点算术运算	676	18.1.2 if...then...fi 示例	695
第 17 章 条件测试	677	18.1.3 if...then...else...fi 示例	696
17.1 Shell 的测试命令	678	18.1.4 if...then...elif...示例	697
17.2 测试文件属性	678	18.1.5 if 嵌套	699
17.2.1 测试文件存在并且是 块设备	678	18.2 case 结构	701
17.2.2 测试文件存在并且是 字符设备	679	18.2.1 case 结构的语法格式	701
17.2.3 测试文件存在并且是 目录	679	18.2.2 case 结构示例	702
17.2.4 测试文件存在	680	18.2.3 case 嵌套	707
17.2.5 测试文件存在并且 可执行	681	18.3 select 结构	709
		18.3.1 select 结构的语法格式	709
		18.3.2 select 结构示例	709
		18.4 while 循环	711
		18.4.1 while 语句的语法格式	711
		18.4.2 while 循环示例	712
		18.4.3 while 的嵌套	713
		18.4.4 避免死循环	714



18.5	for 循环	716	20.4.1	系统设计	750
18.5.1	for 循环语法结构	716	20.4.2	代码实现	751
18.5.2	for 循环示例	717	20.5	对指定目录定时备份	754
18.6	until 循环	719	20.5.1	系统设计	754
18.6.1	until 循环的语法结构	719	20.5.2	代码实现	754
18.6.2	until 循环示例	720	20.6	连续对多个目录逐个备份	755
18.7	break 和 continue 语句	721	20.6.1	系统设计	755
18.7.1	break 语句	721	20.6.2	代码实现	755
18.7.2	continue 语句	723	20.7	自动下载指定文档	756
第 19 章	函数	728	20.7.1	系统设计	757
19.1	函数的定义	729	20.7.2	代码实现	757
19.2	函数的调用	729	20.8	自动上传指定文档	757
19.2.1	执行函数	729	20.8.1	系统设计	758
19.2.2	函数参数传递	731	20.8.2	代码实现	758
19.2.3	函数的返回值	734	20.9	通过命令连续创建 50 个用户	
19.2.4	载入函数	735	账号		760
19.2.5	删除函数	736	20.9.1	系统设计	760
19.3	函数的作用域	737	20.9.2	代码实现	760
19.3.1	全局作用域	737	20.10	通过操作文件创建多个	
19.3.2	局部作用域	738	用户		761
19.3.3	应用示例	739	20.10.1	系统设计	761
19.4	函数的嵌套	740	20.10.2	代码实现	762
19.5	函数的递归	741	20.11	连续创建多个目录	764
第 20 章	Shell 编程实例	745	20.11.1	系统设计	764
20.1	判断文件类型	746	20.11.2	代码实现	764
20.1.1	系统设计	746	20.12	检查系统资源使用情况	765
20.1.2	代码实现	746	20.12.1	系统设计	766
20.2	使用 if 语句编写选择菜单	746	20.12.2	代码实现	766
20.2.1	系统设计	747	20.13	账户的一致性检查	769
20.2.2	代码实现	747	20.13.1	系统设计	769
20.3	使用 while 循环编写选择		20.13.2	代码实现	770
菜单		748	20.14	文件类型转换	773
20.3.1	系统设计	748	20.14.1	系统设计	774
20.3.2	代码实现	748	20.14.2	代码实现	774
20.4	使用 until 循环编写选择		20.15	对输入的 3 个数字进行	
菜单		750	排序		775
			20.15.1	系统设计	775
			20.15.2	代码实现	775

20.16 计算奇数之和	778	20.21 使用 sed 打印指定行	785
20.16.1 系统设计	778	20.21.1 系统设计	785
20.16.2 代码实现	778	20.21.2 代码实现	785
20.17 使用 select 命令生成菜单	779	20.22 使用 sed 筛选普通文件	786
20.17.1 系统设计	779	20.22.1 系统设计	786
20.17.2 代码实现	780	20.22.2 代码实现	786
20.18 打印图形	781	20.23 使用 gawk 反序输出各列	786
20.18.1 系统设计	781	20.23.1 系统设计	787
20.18.2 代码实现	781	20.23.2 代码实现	787
20.19 处理压缩文件	782	20.24 对文件进行大小写转换	787
20.19.1 系统设计	782	20.24.1 系统设计	788
20.19.2 代码实现	782	20.24.2 代码实现	788
20.20 数值进制转换	784	20.25 统计文件中的文本信息	789
20.20.1 系统设计	784	20.25.1 系统设计	789
20.20.2 代码实现	784	20.25.2 代码实现	789



LINUX

上篇 Linux 命令

Linux 系统下包含了大约 2 000 个经常用到的命令。虽然这些命令大部分可以通过图形界面的操作来完成，但命令行快速、高效的优点会给系统的使用、管理和维护提供极大的便捷。

本篇详细阐述了 Linux 系统常用命令的使用方法与示例，主要包括 Linux 系统简介、Red Hat Linux 系统的安装与基本操作、系统管理与配置、用户和用户组管理、磁盘管理、文件和目录管理、备份与压缩以及网络管理与相关应用。

LINUX

第1章 Linux 简介

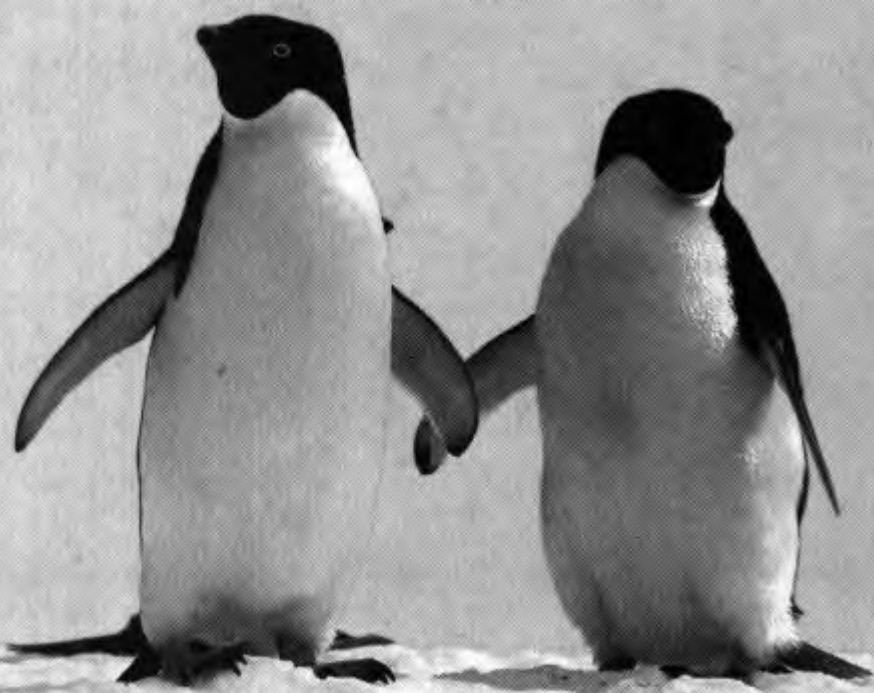
Linux 以其系统简明、功能强大、性能稳定以及扩展性和安全性高著称，可以支持多用户、多任务环境，具有较好的实时性和广泛的协议支持。同时，Linux 在系统兼容性和可移植性方面也有上佳表现，可以广泛应用到 x86、Sun Sparc、Digital、Alpha、MIPS、PowerPC 等平台。

Linux 是一套遵从 POSIX (Portable Operating System Interface for Computer Environment, 即可移植操作系统环境接口) 规范的操作系统，兼容 UNIX System V 及 BSD UNIX，其发行遵守 GPL (GNU General Public License, 即 GNU 的通用公共许可协议)。



POSIX 是一套由 IEEE (电气和电子工程师学会) 制订的标准。POSIX 的意思是计算机环境的可移植操作系统界面。

UNIX System V 和 BSD UNIX 是 UNIX 操作系统的两大主流，目前绝大多数的 UNIX 系统都由这两种系统衍生而来。UNIX System V 系统下的源代码可以在 Linux 下编译后执行，而 BSD UNIX 下的可执行文件也可以直接在 Linux 下运行。



1.1 Linux 起源及特点

Linux 起源于“古老”的 UNIX。1969 年, Bell 实验室的 Ken Thompson 利用一台闲置的 PDP-7 计算机设计了一种多用户、多任务的操作系统。不久, Dennis Richie 加入了这个项目, 他们共同努力开发了最早的 UNIX。早期的 UNIX 由汇编语言编写, 在其第 3 个版本用 C 语言进行了重写。之后, UNIX 得以移植到更为强大的 DEC PDP-11/45 与 11/70 计算机上运行。后来, UNIX 逐渐走出实验室并成为了主流操作系统之一。

但 UNIX 通常是在企业级服务器或工作站等级的服务器上使用的操作系统, 而这些较大型的计算机系统一般价格不菲, 因此得不到普及。由于 UNIX 的功能强大, 有许多开发者希望在相对廉价的 PC 上开发出具有相同功能而且是免费的类似 UNIX 的系统, 其中比较成功的是 Andre S.Tanenbaum 教授开发的 Minix 系统。随后, 有许多人参考 Minix 系统来开发自己的操作系统, Linux 就是在此背景下出现的。

1.1.1 Linux 起源

Linux 因其创始人是 Linus Torvalds 而得名 (Linux 的发音为[ˈlinəks])。Linus Torvalds 是芬兰赫尔辛基大学技术科学系的学生。出于学习和研究的需要, Linus 希望能够做出“比 Minix 更好的 Minix”。1991 年, Linus 在 Minix 的基础上开发了 Linux, 并将 0.02 版放到 Internet 上, 使其成为自由和开放源代码的自由软件。

Linux 随着 Internet 的传播而得到了快速成长, 来自世界各地的编程人员对其进行了修订和扩充。1994 年, 在与互联网上的志愿开发者协同工作的基础上, Linux 发布了标志性的 1.0 版本。值得注意的是, Linux 只是参考了 Minix, 而并不是 Minix 的改良。Minix 采用微内核技术, 而 Linux 采用具有动态加载模块特性的单内核技术。同时, Linux 具备 UNIX 系统所具备的全部特征, 包括多任务、虚拟内存、共享库、需求装载及 TCP/IP 网络支持等。Linux 的成功并没有为 Linus Torvalds 带来巨额财富, 但他的成就使其在计算机发展史上占有了一席之地。

1.1.2 Linux 特点及主要优势

作为操作系统, Linux 在短短几年内得到了如此迅猛的发展, 与 Linux 自身所具有的良好特性是分不开的。简单地说, Linux 具有以下主要特性。

1. 免费的专业级操作系统

Linux 具备服务器级操作系统的强大功能。同时, 由于 Linux 遵守通用公共许可 GPL, 因此任何人都有共享和修改的自由, 并且可在不需要额外费用的条件下得到 Linux 的源代码。用户可以放心地免费使用 Linux, 而不必担心成为盗版用户。

2. 良好的可移植性

可移植性是指将操作系统从一个硬件平台转移到另一个硬件平台, 而无需改变其自身



的运行方式。Linux 是一种可移植的操作系统，到目前为止，几乎能够在所有的计算机平台上运行，包括笔记本电脑、PC 机、工作站，甚至大型机，支持 x86、MIPS、PowerPc、SPARC 等主流系统架构，并同时支持 32 位和 64 位操作系统。在其平台下编写的程序，不用经过太多的修改就可以在各个平台上顺利运行，很好地继承了 UNIX 系统宣称的硬件平台无关性。

3. 良好的用户界面

Linux 具有类似 Windows 图形界面的 X-Windows 系统，用户可以通过使用鼠标，方便灵活地进行操作。X-Windows 系统是源于 UNIX 系统的标准图形界面，最早由 MIT 开发，可以为用户提供一个具有多种窗口管理功能的对象集成环境。经过多年的发展，这种基于 X-Windows 系统的 Linux 图形界面技术已经非常成熟，其用户友好性不逊于 Windows。

4. 低成本

Linux 是开放源代码的操作系统，不仅其内核是免费的，其许多系统程序及应用程序也是自由软件，用户可以从网上免费获得。由于 Linux 拥有众多来自互联网上的志愿开发者在协同工作，所以其功能的完善和漏洞的发现及修改的速度非常快，极大地降低了使用和管理风险。同时，由于 Linux 具有良好的可移植性，不同平台之间软件的移植会变得简单容易，从而可以进一步降低成本。

5. 内核的定制和剪裁

Linux 内核负责管理计算机的各种资源，如处理器和内存，而且必须保证合理地分配资源。当 Linux 启动时，内核被调入内存，并一直驻留在内存中直到关机断电。Linux 内核采用了动态加载技术，用户可以根据需要将内核设计得很小，许多暂不需要的模块可以从内核中剪裁掉，在需要时再重新进行加载。通过利用 Linux 的这个特点，用户在安装 Linux 的时候可以定制出很小的内核，甚至可以在一张容量为 1.44MB 的软盘上安装一个 Linux 操作系统。

6. 广泛的协议支持

可以说，网络就是 Linux 的生命。Linux 在网络方面具备与生俱来的优势，其内核支持的主要协议包括：

- TCP/IP 通信协议。
- IPX/SPX 通信协议。
- Apple Talk 通信协议，包括 X.25 及 Frame-relay。
- ISDN 通信协议。
- PPP、SLIP、PLIP 等通信协议。
- ATM 通信协议。

7. 丰富的应用程序和开发工具支持

由于 Linux 系统具有良好的可移植性，目前大部分在 UNIX 系统下的工具已经被成功移植到 Linux 系统上，包括几乎所有的 GNU 软件库。加上 IBM、Intel、Dell、AMD、Oracle、Sysbase 等国际知名企业的支持，Linux 获得了越来越多的应用程序和开发工具，如下所示。

- 语言及编程环境：C、C++、Java、Perl 和 Fortran 等。
- 数据库：MySQL、PostgreSQL 及 Oracle 等。
- Shell：bash、tcsh、ash 及 csh 等。
- 编辑器：emacs、Gedit、VIM 及 pico 等。
- 图形环境：GNOME、KDE、GIMP、IceWM 等。
- 文字处理软件：Openoffice、Kword、abiWord 等。
- 浏览器：Firefox 等。

8. 良好的安全性和稳定性

Linux 的安全性和稳定性是其另外一个比较明显的特性。Linux 是多任务、多用户操作系统，可以支持多个用户同时使用系统的处理器、内存、磁盘和外设等资源。Linux 的保护机制使每个用户、每个应用程序可以独立地工作。一个用户的某个任务崩溃了，其他用户的任务依然可以正常运行。为了给网络多用户环境中的用户提供必要的安全保障，Linux 采取了多种安全技术措施，包括对读、写进行权限控制，带保护的子系统，审计跟踪，核心授权，等等。由于 Linux 本身的设计就对病毒攻击提供了非常好的防御机制，因此 Linux 系统基本上不用安装什么防毒杀毒软件。

Linux 内核具有极强的稳定性。除非硬件出问题，系统出现死机的概率是很小的，可以长年累月地运行，因此 Linux 被广泛应用在网关和防火墙。

1.2 Linux 版本发展

Linux 继承了 UNIX 版本制定的规则，分为内核版本和发行版本两类。内核版本是指 Linux 系统内核自身的版本号，而发行版本是指由不同的公司或组织将 Linux 内核与应用程序、文档组织在一起，构成的一个发行套装。各个公司或组织通常会使用 CD-ROM 发布它们的 Linux 发行套装。

1.2.1 Linux 内核版本

内核是系统的核心，是运行程序和管理如磁盘和打印机等硬件设备的核心程序。Linux 内核的开发规范一直由 Linus 领导下的开发小组控制着。开发小组每隔一段时间会公布新的内核版本或修订版本。内核具有两种不同的版本号：实验版本和产品化版本。版本的格式通常为“主版本号.次版本号.修正号”。其中主版本号和次版本号表示有重要的功能变动，修正号则表示较小的变动。次版本号如果是偶数则说明这种版本相对稳定，是一个可以产品化的版本，如果是奇数则说明是实验版本，内核中可能加入了一些测试功能，是一个内部可以存在 BUG 的测试版。例如，2.6.34 是产品化版本，2.5.73 表示是一个实验版本。Linux 的两种版本是相互联系的。实验版本是产品化版本的最初副本，产品化版本只修订存在的错误，而实验版本继续对增加的新功能进行测试，直到实验版本被证明是稳定的，才能转换为产品化



版本，开始新的循环。用户可以到 Linux 内核官方网站下载最新的内核代码 (<http://www.kernel.org>)，如图 1-1 所示。

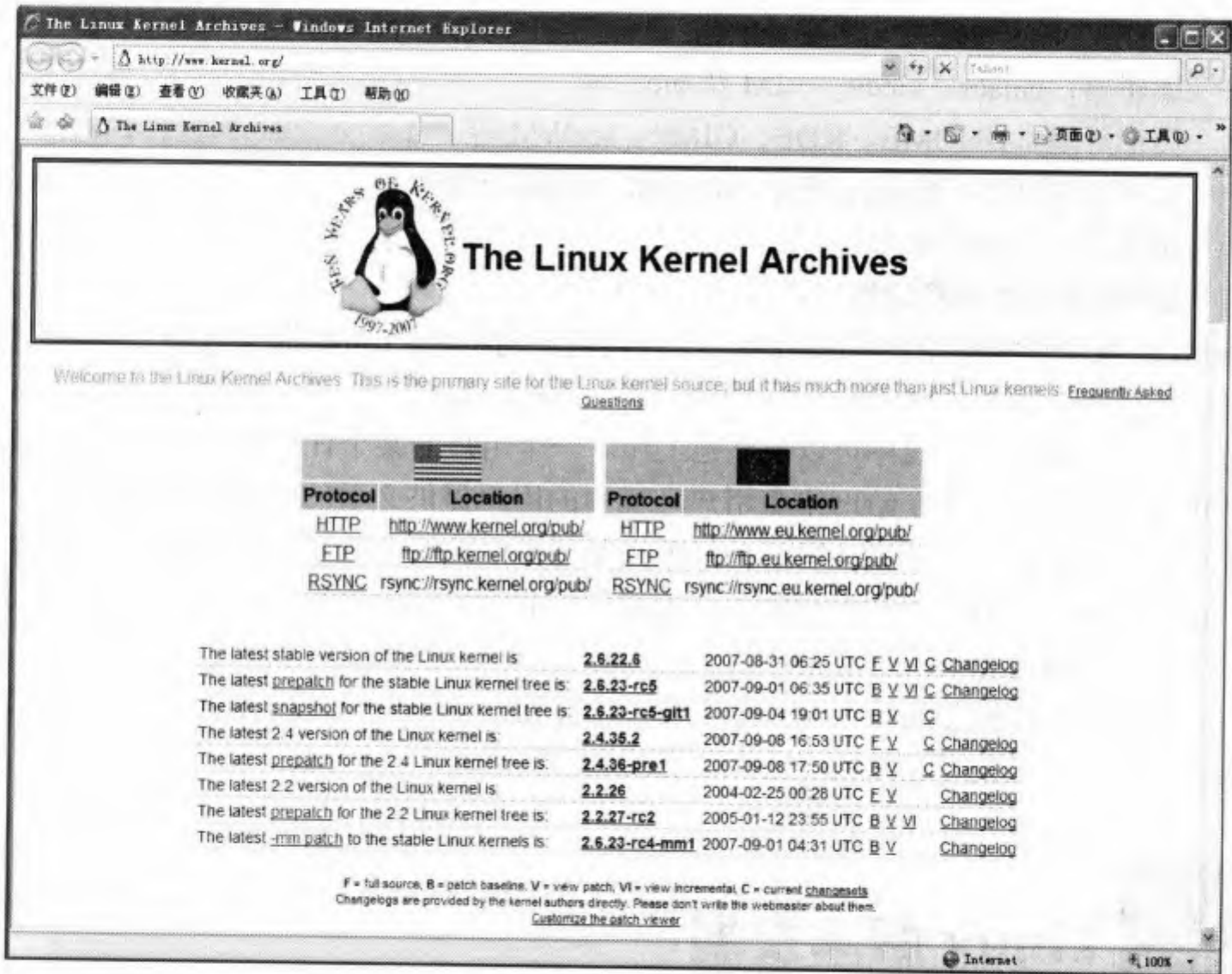


图 1-1 Linux 内核官方网站

1.2.2 Linux 发行版本

由于 Linux 的内核源代码和大量的 Linux 应用程序可以自由获得，很多公司或组织开发了属于自己的 Linux 发行版本。每个发行版本都有自己的特性，目前全球有超过 100 种的 Linux 发行版本，较知名的有 Red Hat、Slackware、Debian、Mandrake、SuSE、Xlinux、Turbo Linux、Blue Point、Red Flag、Xteam 等。

1. Red Hat Linux

Red Hat Linux 是目前流行最广的发行版，几乎成了 Linux 的代名词。其主要特点集中在方便简易的安装和操作使用上，用户可以免去繁杂的安装和设置工作，尽快开始使用 Linux 强大的功能，其图形化的操作环境与 Windows 不相上下。Red Hat Linux 曾被权威的计算机杂志 InfoWorld 评为最佳 Linux。

官方网站为 <http://www.redhat.com/>。

2. Slackware Linux

Slackware Linux 由 Patrick Volkerding 创建于 1992 年，是历史最悠久的 Linux 发行版。其

主要特点是尽量采用原版的软件包而不进行任何修改,并且一直坚持 KISS (Keep It Simple and Stupid) 原则。Slackware 曾经非常流行,但当大多数主流发行版本强调易用性的时候,Slackware 仍然为了追求效率而使用配置文件进行管理,而这对于 Linux 的广大新用户来说是十分困难的。

Slackware 提供了更多的透明性和灵活性,更适合比较有经验的使用者。如果用户希望深入学习 Linux 或者希望安装、编译自己的软件程序,Slackware 是最佳选择。有人曾经这样评价:学会了 Red Hat Linux,则只学会了 Red Hat Linux;而学会了 Slackware Linux,则学会了 Linux。

官方网站为 <http://www.slackware.com/>。

3. Mandriva Linux

Mandriva 原名是 Mandrake,最早由 Gal Duval 创建并于 1998 年 7 月发布。其特点是集成了图形化的桌面环境及图形化的配置工具。在早期 Linux 普遍比较难安装阶段,Mandriva 图形化的安装和配置方式为 Linux 的易用性带来了很大改进。Mandrake 最早是基于 Red Hat 进行开发的,因此继承了许多 Red Hat 的优点,但 Red Hat 默认的桌面是 GNOME,而 Mandriva 则采用 KDE。

官方网站为 <http://www.mandrivaLinux.com>。

4. Debian Linux

Debian Linux 最早由 Ian Murdock 于 1993 年创建,是迄今为止最遵循 GNU 规范的 Linux 系统。其特点是使用了 Debian 特有的软件包管理工具 dpkg,使得在 Debian 上安装、升级、删除和管理软件包变得非常容易。

在 Debian 内部有一套很特别的版本分发制度,分别为 stable、unstable 和 testing。其中,stable 是 Debian 的外部发行版本,该版本在稳定性和安全性方面要求都非常高。而 unstable 则是开发中的版本,更新速度快,因而风险也比较高。testing 版本大多是 unstable 版本经过维护、开发人员不断测试后的版本,实际上已经很接近 stable 版本了。

官方网站为 <http://www.debian.org>。

5. SUSE Linux

SUSE Linux 最早是由德国的 SUSE linux AG 公司发行维护的 Linux 发行版,其特点是使用了自主开发的软件包管理系统 YaST 并受到用户的普遍欢迎。2003 年 11 月,Novell 收购了 SUSE,并对 SUSE Linux 进行了改进,使 SUSE 迅速成长为 Red Hat 有力的竞争对手。目前,SUSE 面向企业或高级桌面的 Linux 版本包括:SUSE Linux Enterprise Server、Novell Open Enterprise Server 和 Novell Linux Desktop。

官方网站为 <http://www.suse.com>。

6. 红旗 Linux

红旗 Linux 是中国人自行研发的 Linux,在中国民族软件产业化过程中具有里程碑意义。其特点是提供了良好的中文支持,其界面和操作设计也更符合中国人的习惯。

官方网站为 <http://www.redflag-linux.com>。



1.3 Linux 与自由软件

Linux 是一种开放源代码的操作系统，它的出现打破了传统商业操作系统长久以来形成的技术垄断与壁垒，进一步推动了人类的信息技术进步。尤为可贵的是，Linux 树立了“自由开放之路”的成功典范。

1.3.1 GNU 与 FSF

自由软件是相对于商业软件和共享软件而言的，自由软件的创始人是 Richard Stallman，他于 1984 年开创了 GNU Project（GNU 是采用递归方式定义的，是“GNU, s Not UNIX”的首字母缩写），他认为自由软件是全人类共同的财富，应该自由传播。

自由软件是指允许任何人使用、复制、修改、分发（免费/少许收费）、改进的软件。“自由”实际上指的是使用上的自由、获得源程序的自由、修改的自由、复制和推广的自由。自由软件从诞生之日起就表现出一种旺盛的生命力，得到业界认可并迅速遍及全球，许许多多年轻的志愿者在 GNU 的激励下，编写了大量自由软件，Richard Stallman 本人也开发了很多为人们广泛使用的著名的自由软件，包括 GNU Compiler Collection (GCC)、GNU symbolic debugger (GDB) 和 GNU Emacs。

由于 GNU Emacs 的成功，Richard Stallman 在 1985 年建立了自由软件基金会 FSF (Free Software Foundation)。

1.3.2 GPL 与 Copyright

GPL 是由自由软件基金会 (FSF) 发行的用于计算机软件的证书，取得该证书的软件被称为自由软件。GPL 与传统商业软件许可协议 Copyright 相对立的，所以又被称为 Copyleft。

GPL 保证任何人有共享和修改自由软件的自由，并且规定，在不增加附加费用的条件下可以得到自由软件的源代码。同时，还规定自由软件的衍生作品必须以 GPL 作为它重新发布的许可协议，即 GPL 在给予用户各种权利的同时，也要求用户通过修改其自由软件而得到的软件也必须成为自由软件，从而从整体上保证了自由软件的衍生和传播。

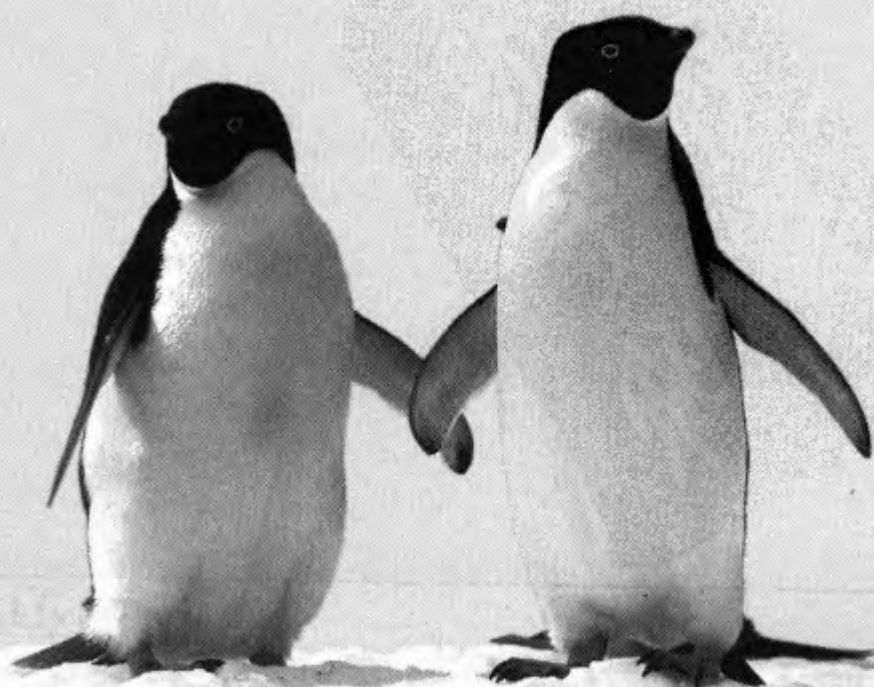
在 GNU 创立之初，Richard Stallman 一直希望能有一个完全自由的、兼容 UNIX 但功能更强大的操作系统，以便为所有的计算机使用者提供一个功能齐全、性能良好的基本系统。1991 年，芬兰人 Linus Torvalds 开发了一个 Linux 内核。1993 年 Linux 软件正式加盟“自由软件联盟”。Richard Stallman 表示：“Linux 是 GPL 协议下众多软件中的一种，正是基于 GPL 自由公开的机制，使得众多的电脑高手有机会参与其内核的开发，从而使其在短短几年内取得飞速的发展。目前，Linux 已经发展得非常成熟。从另一方面来说，Linux 的发展也将带动整个自由软件的发展”。严格地说，Linux 是在 GPL (GNU General Public License) 版权协议下发行的操作系统内核，其版权属于 Linus Torvalds。

Linux 出现至今已经近 20 年, 随着 Internet 的飞速发展, Linux 已经从一个学生的试验软件演变为成熟、实用的操作系统, 并逐渐进入商业、办公、军事等各个领域。目前, 国外的 Linux 软件提供商, 如 RedHat、SUSE、Slackware、TurboLinux、Debian Linux 等, 一直致力于提供对 Linux 的应用系统支持和用户界面的开发和改进, 国内的如红旗 Linux、蓝点 Linux、Xterm Linux 等, 也一直致力于 Linux 的本地化发展。总之, Linux 的发展壮大以事实证明了自由软件发展道路的成功。

LINUX

第2章 初识 Linux

熟练掌握 Linux 是一个循序渐进的系统学习过程,需要花费一定的时间和精力。但对于每一位 Linux 初学者, Linux 大门却是始终敞开的,只需通过简单的学习就可顺利地踏上 Linux 之路。



2.1 Red Hat Linux 的发展历史

Red Hat 公司最早由 Bob Young 和 Marc Ewing 在 1995 年创建。1998 年, RedHat 5.0 获得了 InfoWorld 的操作系统奖项。1998 年 10 月, Intel 和 Netscape 宣布小额投资 Red Hat 软件, 这被业界视作 Linux 获得商业认同的信号。

1999 年, IBM 宣布与 Red Hat 公司建立伙伴关系, 以确保 Red Hat 在 IBM 机器上正确运行。同年, 第一届 LinuxWorld 大会召开, IBM、Compaq 和 Novell 宣布投资 Red Hat 公司, 而以前一直对 Linux 持否定态度的 Oracle 公司也宣布投资。Linux 的时代正式到来。

Red Hat 公司开始只有一个 Red Hat 版本, 但由于被越来越多的用户所接受, 单一的 Red Hat 版本已经无法满足用户的需求。为此, Red Hat 公司在 2002 年推出了收费的 Red Hat Enterprise Linux (简称 RHEL, 即 Red Hat 的企业版), 而对普通的 Red Hat Linux, 在 9.0 之后 Red Hat 公司就停止了技术支持。因此, 许多 Linux 爱好者们指责 Red Hat 为了利润而抛弃了自由软件精神。

目前 Red Hat Linux 分为两个系列: Red Hat Enterprise Linux 适用于企业级服务器, 由 Red Hat 公司提供收费的技术支持和更新, 最新版本为 Red Hat Enterprise Linux 5; Red Hat Fedora Core 定位于桌面用户, 适用于非关键性的计算环境, 由 Fedora 社区开发并提供免费的支持, 最新版本为 Red Hat Fedora Core 9。

2.2 Red Hat Linux 的安装

与 Windows 系统安装类似, Linux 的安装也需要考虑软、硬件两方面的需求。本节将以 Red Hat Linux 9 为例, 详细讲述 Linux 的安装。

2.2.1 硬件需求

Linux 内核运行对硬件要求很低, 在很多嵌入式系统中使用的 Linux 内核的大小很多不到 100KB。当然, 普通用户使用的 Linux 服务器版或桌面版就比较庞大, 以 Red Hat Linux 为例, 其 9.0 版完全安装大概要占用硬盘空间 3.5GB, 默认安装占用硬盘空间 1.3GB 左右。推荐使用 128MB 以上内存。至于其 Advanced Server 则至少需要 256MB 内存。如果要在 Linux 下运行其他程序, 如 Oracle 数据库, 那么这些程序可能对系统配置还有额外要求。

2.2.2 安装方式

下面将以 Red Hat Linux 9 为例说明 Linux 的安装方式。一般来说, Red Hat Linux 9 可以使用 3 种安装方式: 光盘安装、硬盘安装和网络安装。

- 光盘安装: 光盘安装是最理想和最简单的安装方式。下一小节将详细讲述这种安装



方式。

- 硬盘安装：如果机器没有光驱，或者没有 Red Hat Linux 9 安装光盘，那么可以考虑硬盘安装方式。硬盘安装前首先要将安装光盘的 ISO 镜像文件复制到硬盘的某个目录下（Red Hat Linux 9 安装光盘有 5 个 ISO 文件，一般安装过程使用前 3 个 ISO 文件即可）。解压缩第一个 ISO 文件（假定解压到 e:\ shrike-i386-disc1 目录下），接下来就是制作 Linux 安装软盘。在 DOS 下安装软盘制作过程如下。

```
cd e:\ shrike-i386-disc1\dosutils //改变当前目录到 rawrite 所在目录
rawrite //执行 rawrite 程序
..\images\boot.img //指定 Linux 映像文件名
a: //指定安装软盘所在的驱动器
```

有了安装软盘后，用该软盘做系统启动盘启动机器，即可见到如图 2-1 所示界面。按回车键后系统会询问 Red Hat Linux 9 安装 ISO 文件所在目录，正确指定该目录后的安装过程和光盘安装过程完全一样。

- 网络安装：Red Hat 支持 HTTP 和 FTP 两种协议的网络安装。适合于本地机器没有光驱，而知道网上 ISO 文件所在 URL 的 Linux 安装。网络安装也要制作系统安装软盘，并用该软盘启动机器。机器启动后选择图像安装或文本安装均可，在接下来的安装方式中选择“网络安装”，指定安装 ISO 文件所在的 URL，再接下来和光盘安装类似。

2.2.3 安装过程

下面以 Red Hat Linux 9 光盘安装为例，详细讲述 Linux 的安装过程。

(1) 用光盘引导机器：首先用 Red Hat 的安装光盘引导机器，正确引导后将出现图 2-1 所示的界面。直接按回车键将进入图形安装模式，如果想要采用字符安装模式，输入 linux text 并回车即可。由于图形安装模式更直观、更简单，这里直接回车进入图形安装模式。

(2) 介质检查：选择是否需要介质检查，如图 2-2 所示。这是事先检查安装光盘文件是否完整，是否没有损坏。如果确信光盘没有问题，可以直接单击【Skip】按钮，如果确实想检查可以单击【OK】按钮。

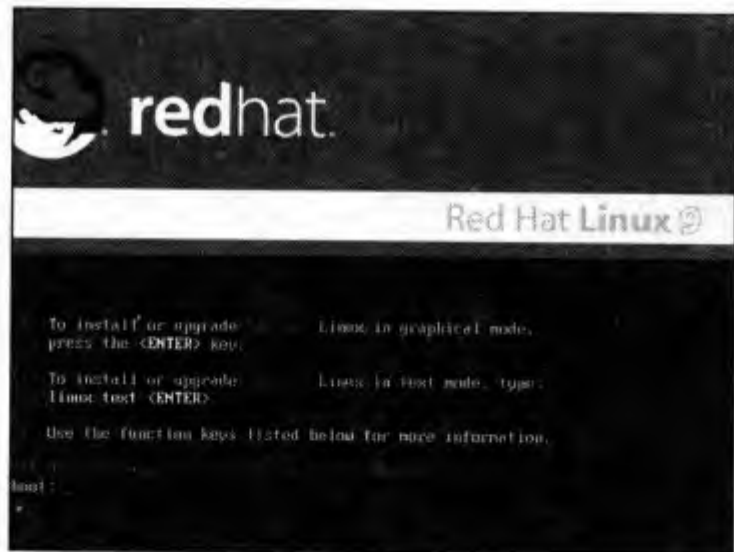


图 2-1 选择安装界面

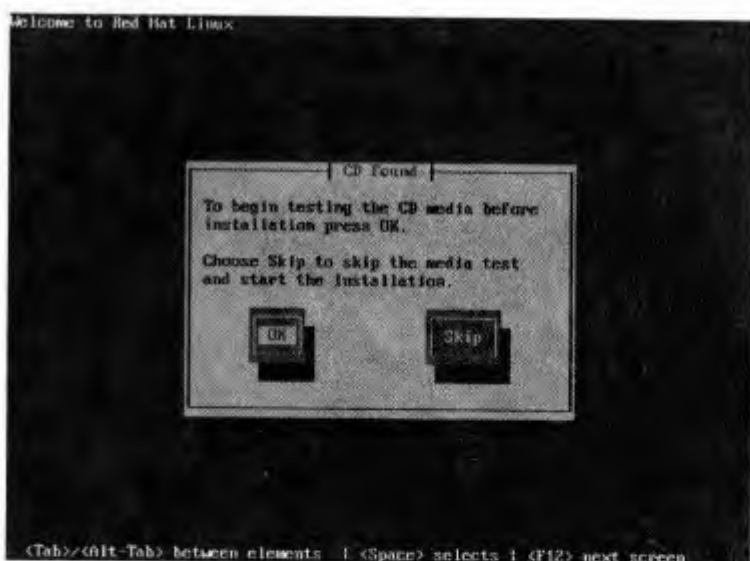


图 2-2 介质检查

(3) 开始安装：如图 2-3 所示，系统显示 Red Hat Linux 9 安装欢迎信息，直接单击【Next】

按钮进入下一画面。

(4) 选择安装过程中的界面语言：如图 2-4 所示，选择安装过程中的界面语言。此处笔者选择的是“Chinese Simplified (简体中文)”。单击【Next】按钮进入下一个画面。

(5) 选择键盘类型：如图 2-5 所示，选择键盘类型，一般使用 US English 103 键盘。如果系统使用的不是默认的键盘，请选择其他类型。系统一般都能够识别正确的键盘类型。选择好键盘类型后，单击【下一步】按钮进入下一个画面。

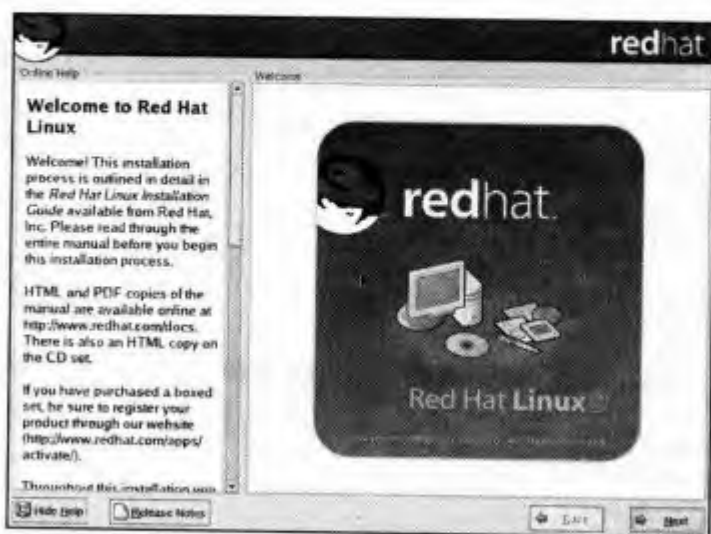


图 2-3 开始安装



图 2-4 选择安装过程中的界面语言



图 2-5 选择系统键盘类型

(6) 选择鼠标类型：接下来的工作是选择所使用的鼠标类型，如图 2-6 所示。现在最常用的鼠标有两种接口，一种是旧式的 PS/2 鼠标，通过一个 6 针微型 DIN 接口与计算机相连；另一种是现在很流行的 USB 接口鼠标。系统一般都能够正确识别用户所使用的鼠标。在屏幕最下面有一个【模拟 3 键】复选框，这是为老式 2 键鼠标提供的功能。如果勾选该复选框，那么同时按下鼠标左键和右键就等同于按下 3 键鼠标的中键。选择好鼠标类型后单击【下一步】按钮进入下一个画面。

(7) 选择安装类型：如图 2-7 所示，有 4 种安装类型，即个人桌面、工作站、服务器和定制。个人桌面一般适合于家庭使用，工作站一般适合于办公使用，服务器类型一般都会默认安装许多服务器软件，定制则完全是用户自己选择需要安装的软件。

(8) 选择分区设置：接下来是选择分区设置。如图 2-8 所示，用户可以从“自动分区 (A)”和“用 Disk Druid 手工分区 (D)”两种方式中选择一个。如果选择“自动分区 (A)”，系统将根据用户选定的安装类型来建立分区，建立分区后可以再进行定制分区；如果选择“用 Disk Druid 手工分区 (D)”将进入手工分区界面。一般选择“用 Disk Druid 手工分区”，单击【下一步】按钮。

(9) 分区：如图 2-9 所示，系统显示检测到的所有驱动器及其所有分区。用户可以在空闲空间新建分区，也可以选择某个分区进行编辑，还可以删除某些分区。

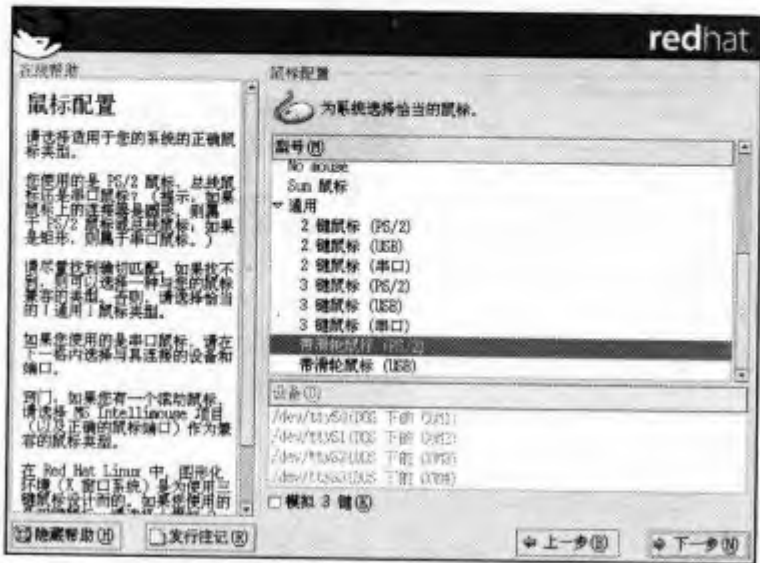


图 2-6 选择鼠标类型

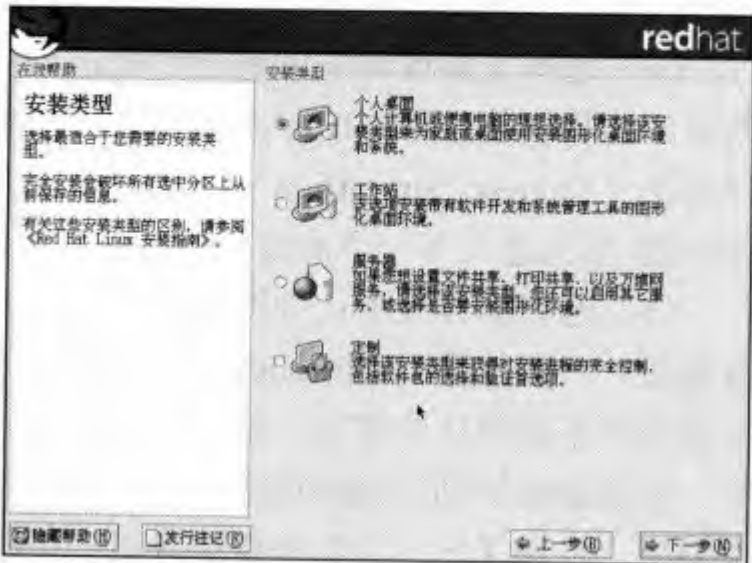


图 2-7 选择安装类型

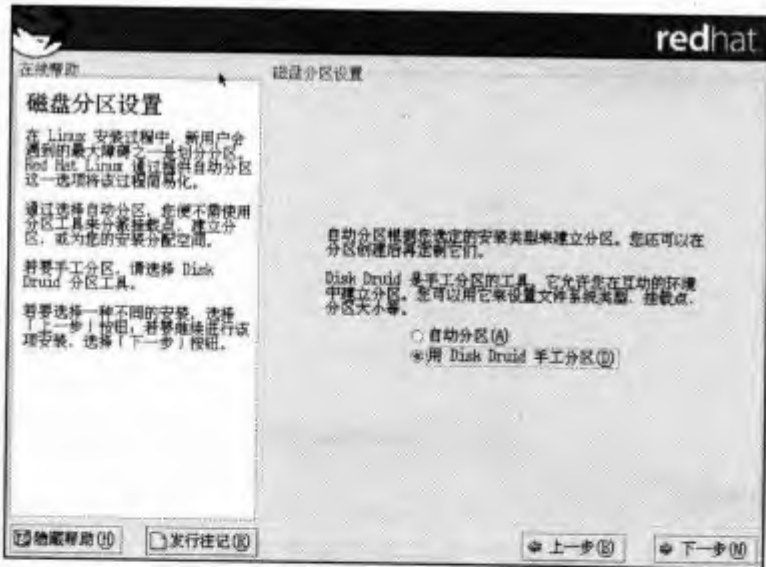


图 2-8 选择分区设置

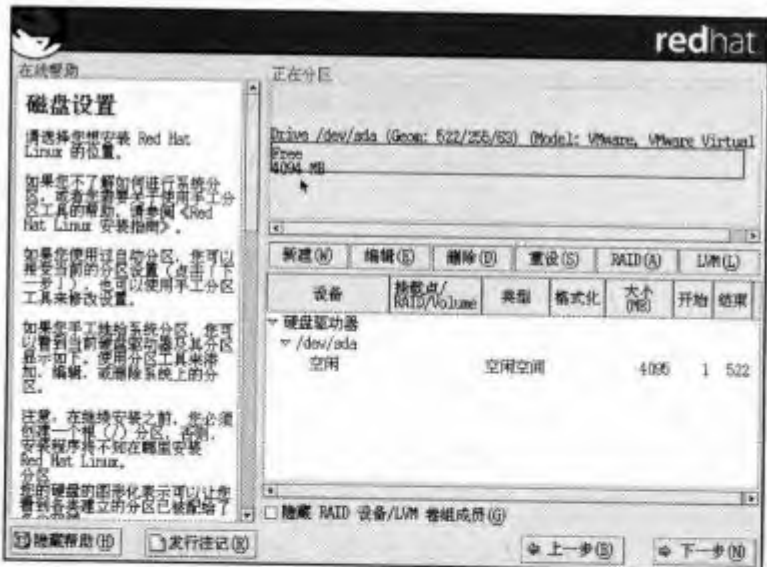


图 2-9 定制系统分区

一般安装 Linux 至少需要两种分区，一个就是 Swap 分区，用于虚拟内存的交换分区，由操作系统管理，用户不能使用。其大小一般设置为物理内存大小的两倍比较合适，太小了不能较好地起到虚拟内存的作用，太大了则浪费硬盘空间。另外一个分区就是安装 Linux 文件系统的分区。Ext2 是 Linux 专用的文件系统，Ext3 是 Ext2 的升级版，所以一般采用 Ext3 作为 Linux 的文件系统。单击【新建】按钮将出现如图 2-10 的界面。

(10) 新建文件分区：如果选择的文件系统类型是“Swap”，那么只需要指定其分区大小即可。如果选择的文件系统类型不是“Swap”，还有一个指定挂载点的问题。如图 2-10 所示，挂载点指定分区将用来保存 Linux 操作系统目录树中的某个目录，至少应该指定一个挂载点“/”。单击【确定】按钮。

(11) 配置 Linux 引导装载程序：Red Hat Linux 9 默认使用 GRUB 作为其引导装载程序，也可以在这里更改引导装载程序。常用的引导装载程序有两种，即 GRUB 和 LILO，在以后的章节中将会对其进行详细说明。这里使用默认的 GRUB 作为引导装载程序。也可以通过本界面配置引导程序引导其他操作系统（如 Windows 等）。如图 2-11 所示，单击【添加】按钮可以在开机引导装载程序运行时的操作系统选择菜单中添加一个操作系统选项，单击【编辑】按钮可以编辑开机引导装载程序运行时的操作系统选择菜单的显示结果，单击【删除】按钮

可以删除一个已有的操作系统选项。

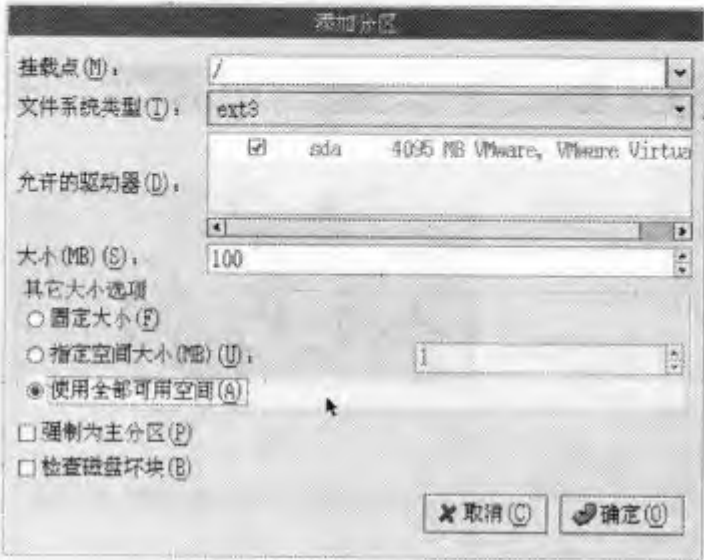


图 2-10 新建文件系统

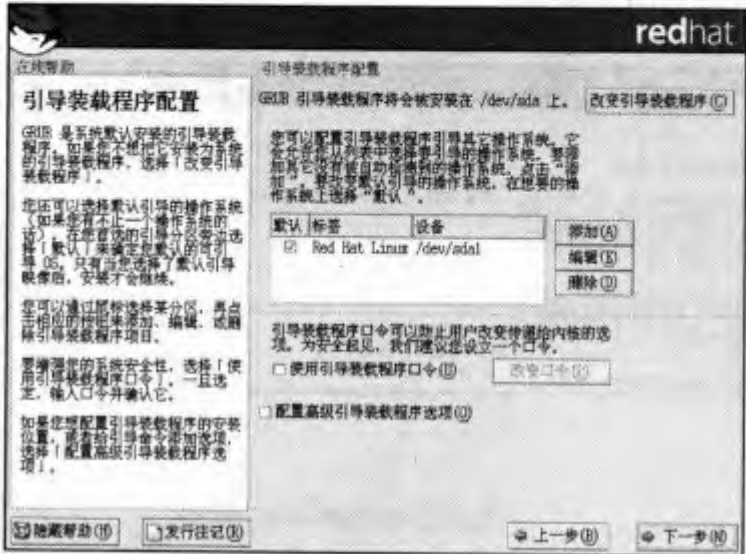


图 2-11 配置 Linux 引导装载程序

(12) 配置网络：系统将自动检测机器中的网络适配卡。如图 2-12 所示，单击【编辑】按钮可以配置本机 IP 地址和子网掩码，或者指定为采用 DHCP 动态获取 IP 地址。如果不是采用 DHCP 动态获取 IP 地址，则还要通过本界面设置网关、主要 DNS、次要 DNS 和第三 DNS 的 IP 地址。完成这些操作后单击【下一步】按钮进入下一环节。

(13) 防火墙设置：如图 2-13 所示，可以指定系统是否启用防火墙。如果选择启用防火墙，则还可以选择允许哪些服务通过防火墙，或者哪些网络适配器是信任设备。

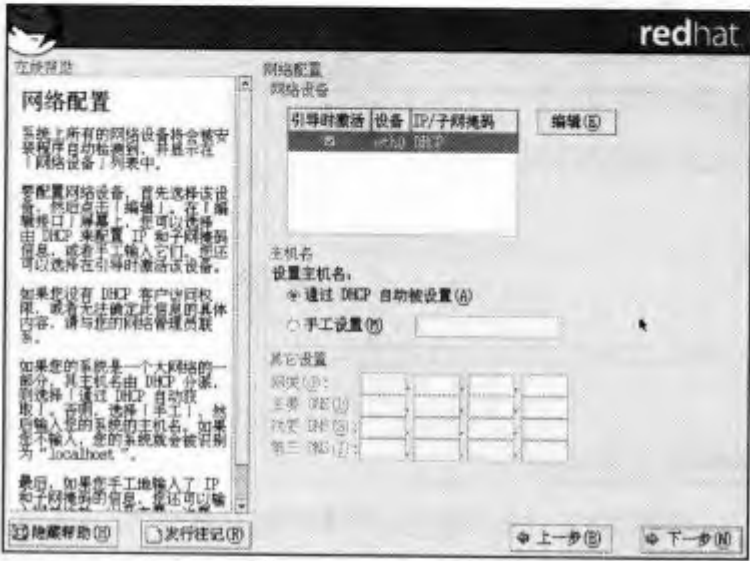


图 2-12 配置网络

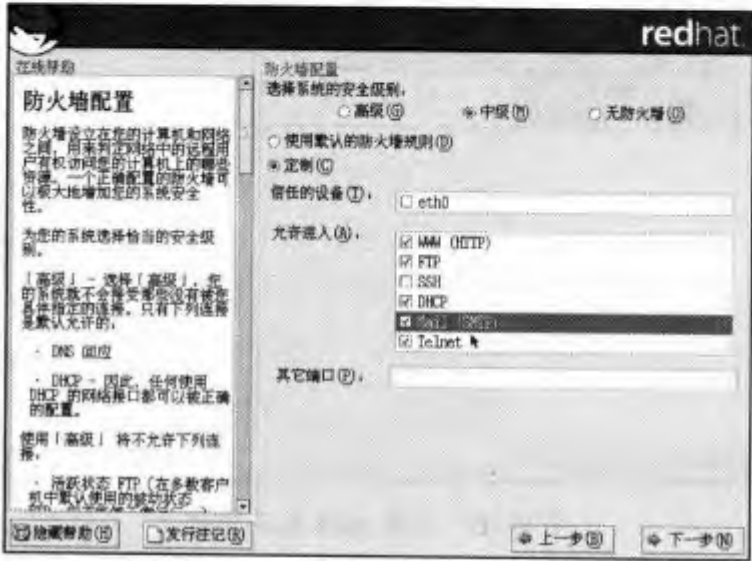


图 2-13 配置系统防火墙

(14) 选择系统支持语言：通过这个界面可以选择操作系统所支持的语言，也可以通过这里指定系统的默认语言。一般建议选择支持简体中文 (Chinese P.R China) 和美国英语 (English U.S.A)，如图 2-14 所示。

(15) 选择用户所在时区：这里一般应该选择“亚洲/上海”，因为我国都使用北京时间。也可以通过这里设置时钟服务器，使得本机时钟保持和所设置的服务器的时钟同步，如图 2-15 所示。

(16) 设置 root 口令：如图 2-16 所示，这里用来设置系统 root 用户密码。root 拥有执行所有程序、查看所有文件和系统配置的权限。如果 root 密码被人破解，则将对系统造成很大



的危害，因此 root 密码应该设置得复杂一点，最好是数字与字母相结合，而且尽量不要用字典里面的词语。



图 2-14 选择操作系统支持的语言



图 2-15 选择用户所在时区

(17) 选择是否接受系统默认软件包：如图 2-17 所示。如果对系统不是很熟悉，或者系统默认软件包已经符合需求，用户可以直接接受系统默认软件包；如果要定制软件包，可以选择“定制要安装的软件包集合”并选择需要的软件集合。

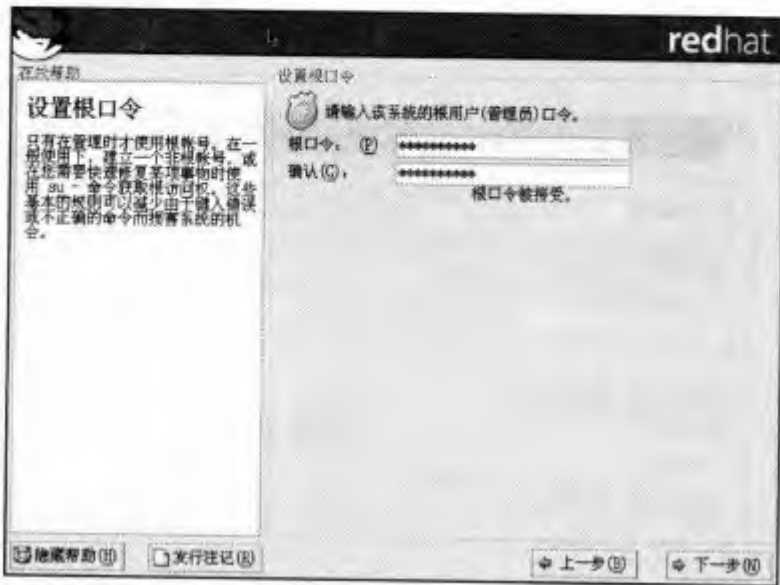


图 2-16 设置 root 用户密码

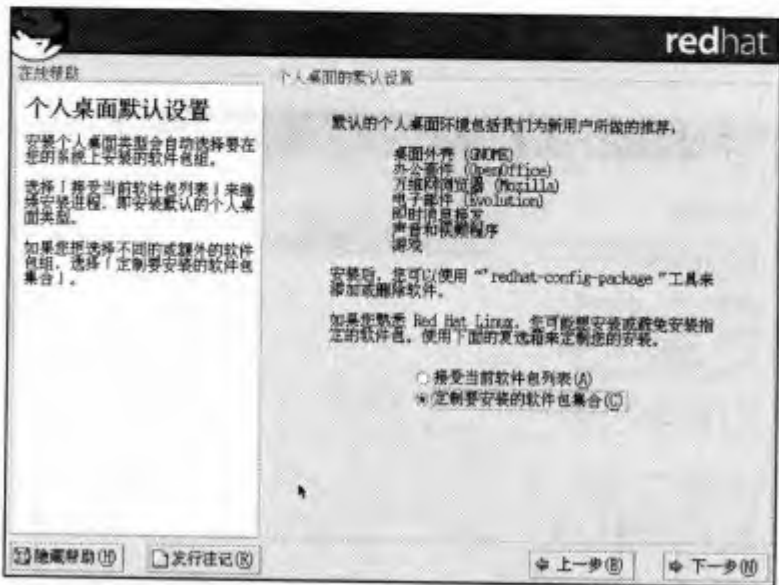


图 2-17 选择是否接受系统默认软件包

(18) 即将开始安装：如图 2-18 所示，系统提示即将开始安装，并且提示安装日志和安装选项记录所在的文件。

(19) 安装软件包：到这里，系统已经收集好了用户的各项信息，下面就是安装软件包的过程。依据机器硬件配置和用户选择安装的软件集合大小，这个过程需要的时间会有较大差异，大致需要 40 分钟到 1 个小时不等，如图 2-19 所示。

(20) 配置图形化界面：如图 2-20 所示，选择系统显卡类型和显存大小。一般系统可以自动检测到合适的显卡类型和显存大小。如果确信系统能够正确检测到这些配置，可以在这里勾选【跳过 X 配置 (S)】。如果没有选择【跳过 X 配置 (S)】，单击【下一步】按钮将开始进行显示器的配置，包括显卡型号、显示器刷新率、分辨率和颜色深度等。

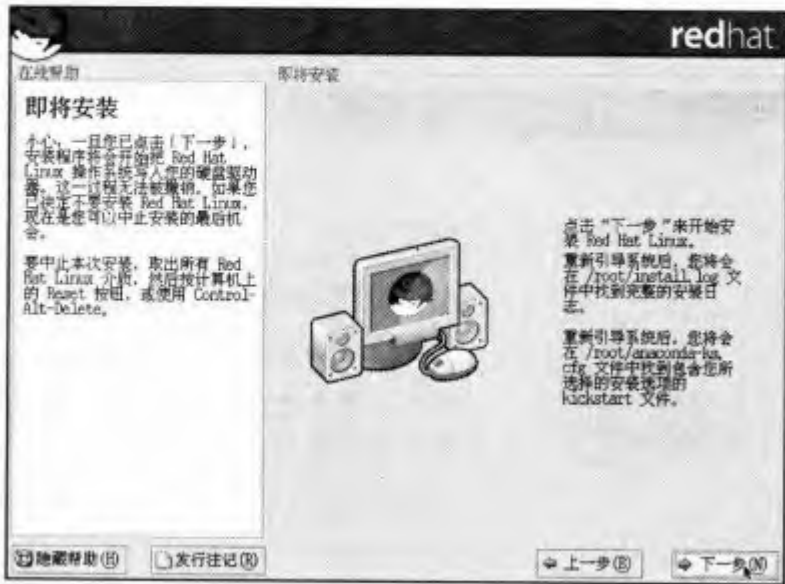


图 2-18 即将开始安装

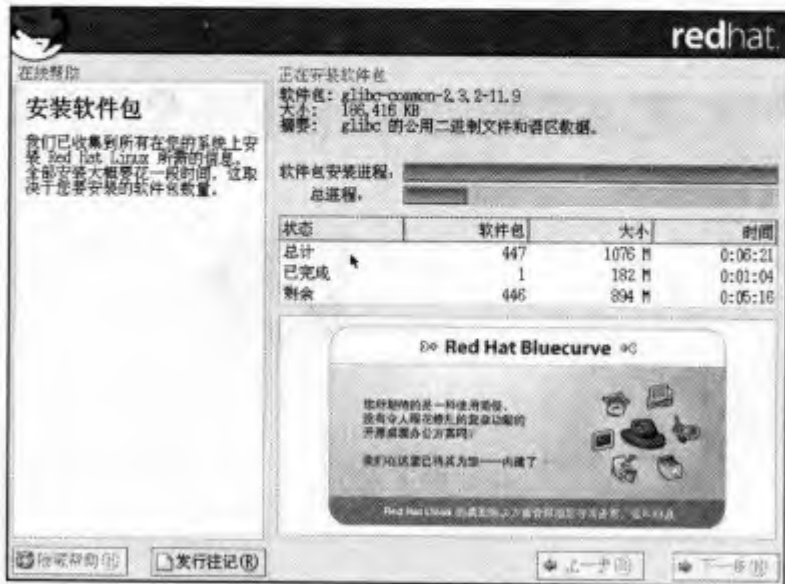


图 2-19 安装软件包

(21) 配置显示器：配置显示器包括配置显示器的品牌、型号、水平频率和垂直频率。一般系统可以自动检测到合适的显示器配置，如图 2-21 所示。

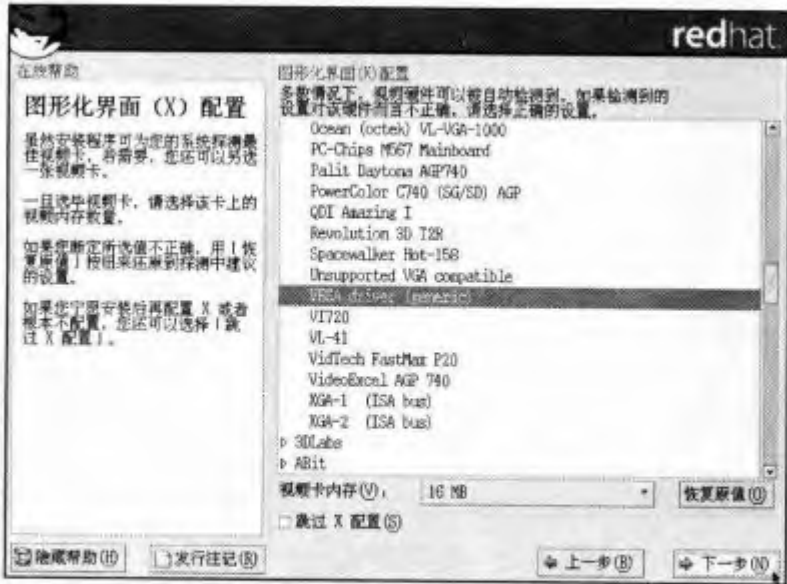


图 2-20 配置图形化界面

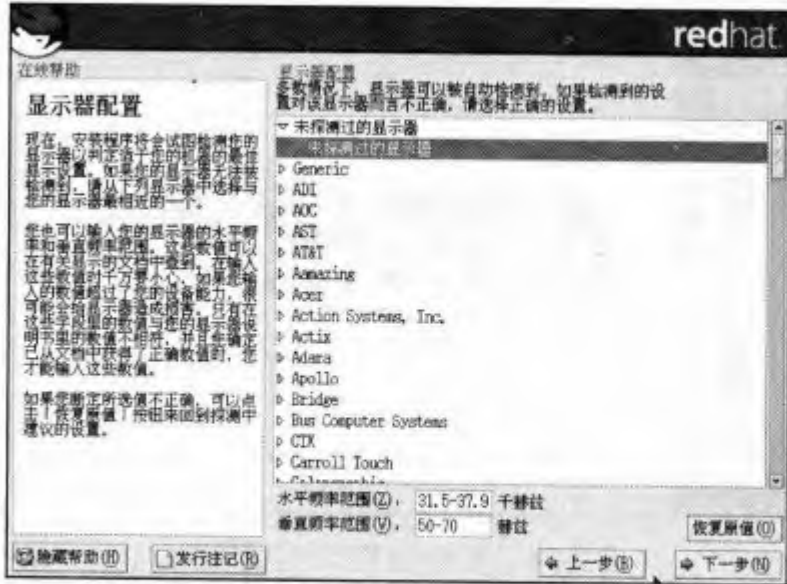


图 2-21 配置显示器

(22) 定制图形化配置：这里配置显示的分辨率和色彩深度，如图 2-22 所示。系统一般可以自动检测到合适的色彩深度和屏幕分辨率。建议用户选择图形化登录类型。

(23) 安装完成：到这里，系统已经初步安装完毕。如图 2-23 所示，系统提示取出安装过程中使用的安装介质，并给出了一些关于 Red Hat 的网站。单击【退出】按钮，重新启动后将进入首次使用前的一些其他配置界面。

(24) 首次运行前的欢迎信息：这是首次运行前的欢迎信息，如图 2-24 所示，直接单击【前进】按钮即可。

(25) 新建普通用户账号：Linux 操作系统 root 权限太大，一般情况下不建议采用 root 登录系统，所以有必要新建普通用户账号。输入普通用户的用户名、用户全名、口令及确认口令即可新建普通用户，如图 1-25 所示。

(26) 设置日期和时间：如图 2-26 所示，可以在这里设置系统时钟。这里勾选【启用网络时间协议】，并配置相应时间服务器，启用网络时间协议。用户可以根据具体情况选择是否



启用网络时间协议。

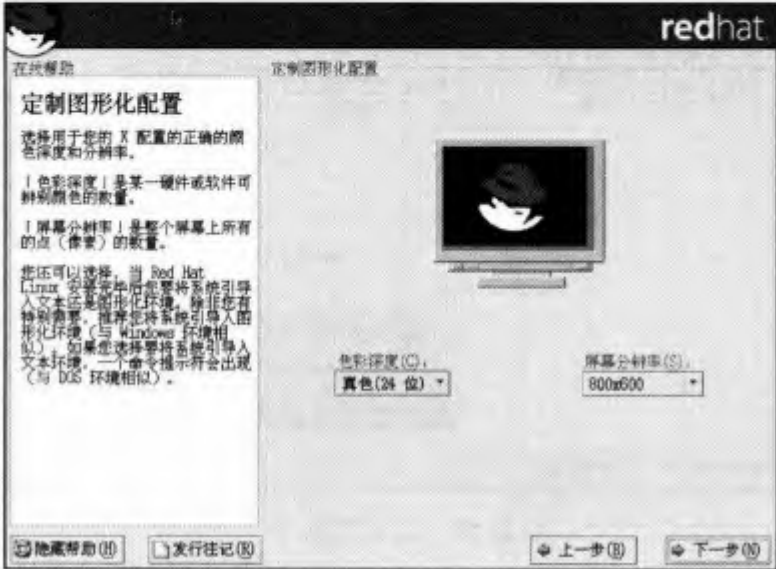


图 2-22 配置显示分辨率和色彩深度

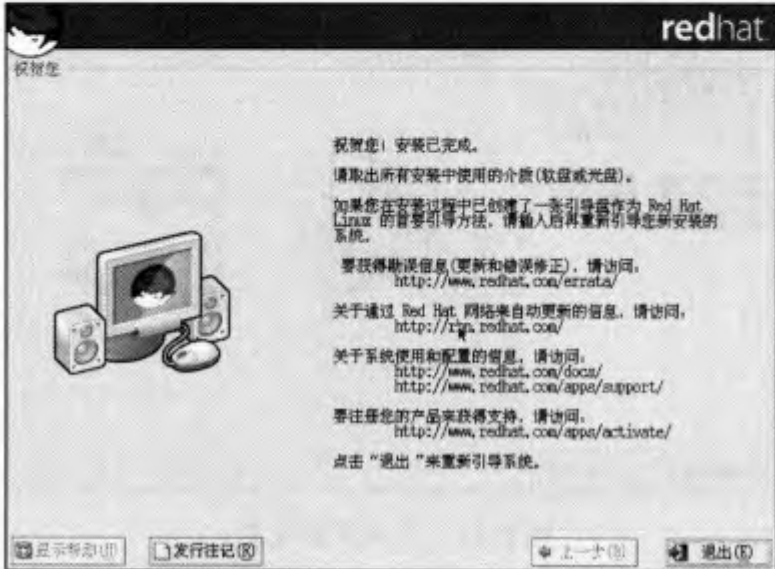


图 2-23 安装完成



图 2-24 首次运行前的欢迎信息

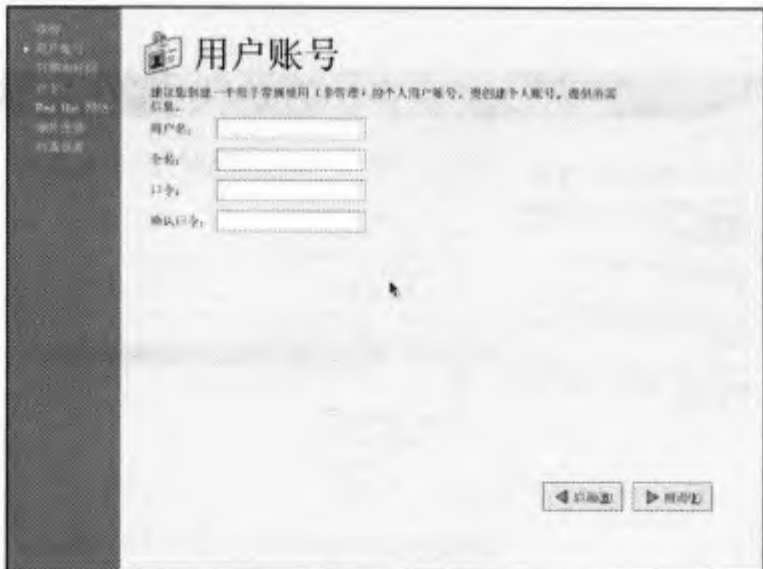


图 2-25 新建普通用户账号

- (27) 测试声卡：如果系统检测到安装有声卡，将显示如图 2-27 所示界面。如果系统正确检测到声卡，单击【播放测试声音】按钮则可以听到 3 个测试声音，第一个从右声道播放，第二个从左声道播放，第三个从两个声道同时播放。系统一般都能够正常检测到声卡。
- (28) 选择是否注册系统：如图 2-28 所示，可以选择注册，也可以选择不注册。如果在 Red Hat 网络注册系统，Red Hat 公司将定期向用户发送漏洞报告，并告诉用户如何下载补丁。
- (29) 额外安装：如果有其他光盘程序要安装，可以在这里安装；若没有，直接单击【前进】按钮，如图 2-29 所示。
- (30) 结束设置：如果能够看到如图 2-30 所示界面，则已经成功完成 Linux 的安装。
- (31) 选择操作系统：如图 2-31 所示界面，这是用户选择的 Linux 装载机 GRUB。如果用户还安装了其他操作系统，如 Windows 系列的操作系统，则也会在启动菜单中显示。选择【Red Hat Linux (2.4.20-8)】再按回车键，即可进入刚安装好的 Red Hat Linux 9。

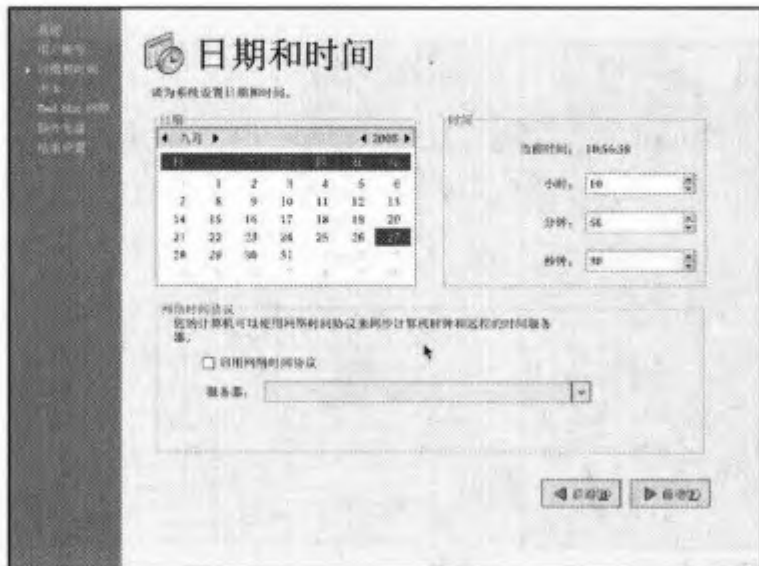


图 2-26 设置系统时钟



图 2-27 测试声卡



图 2-28 选择是否需要注册系统

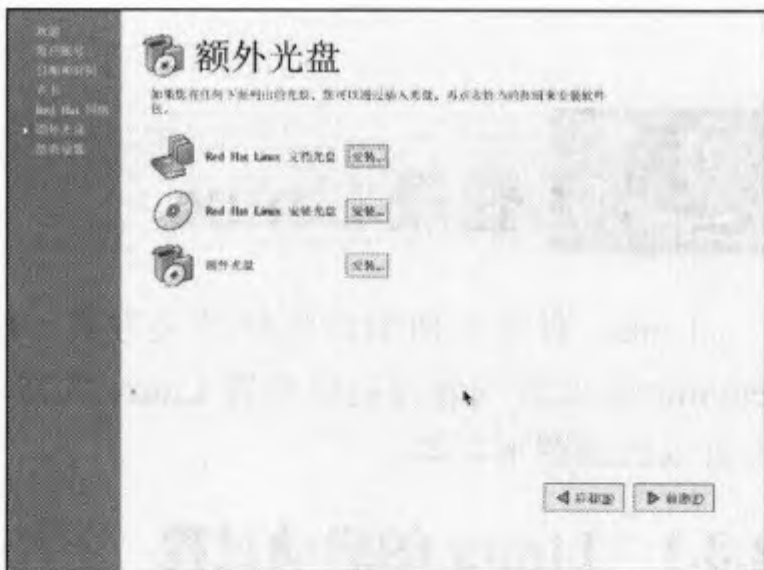


图 2-29 额外安装



图 2-30 安装结束



图 2-31 选择操作系统

(32) 输入用户名及其密码：界面如图 2-32 所示。用户可以输入 root 用户名及其相应密码进入 Red Hat Linux 9，也可以输入在安装过程中添加的某个普通用户账号及其相应密码进入系统。

(33) 开始使用操作系统：界面如图 2-33 所示。用户成功登录后，系统即显示 Red Hat Linux 9 的桌面，用户就可开始使用刚刚安装的操作系統了。



图 2-32 输入用户名及其密码

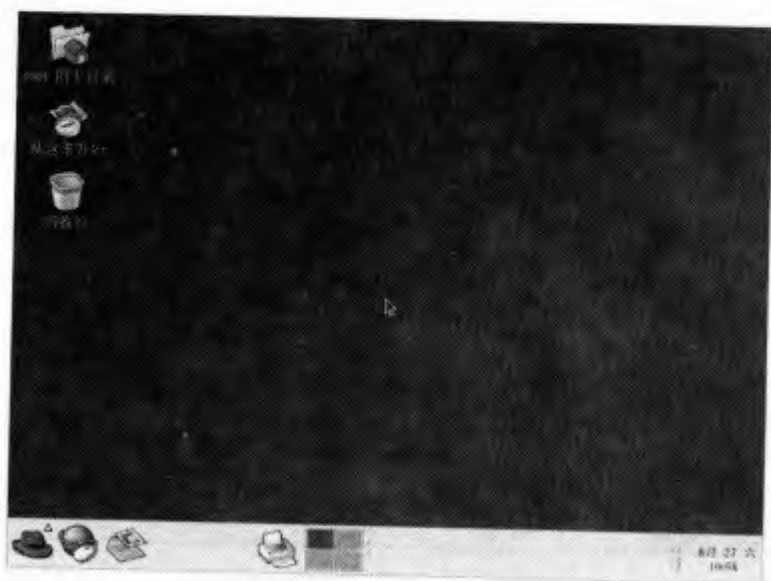


图 2-33 开始使用操作系统

2.3 登录 Linux

Linux 有文本和图形两种登录方式。Red Hat Linux 9 默认以图形方式登录，通过对 `/etc/inittab` 文件的修改可以设置 Linux 以文本方式登录。在文本方式下，通过执行命令 `startx` 也可以启动图形界面。

2.3.1 Linux 的启动过程

启动 Linux 操作系统，须安装 Linux 启动器，常见的 Linux 启动器有两种：LILO (Linux LOader) 和 GRUB (GRand Unified Bootloader)。Red Hat Linux 9 默认采用 GRUB 作为其启动器。LILO 或 GRUB 不仅可以引导 Linux，也可以引导其他操作系统，包括 MS-DOS 或 Windows XP、Windows Server 2003、Windows 2000 等。在安装 Linux 时，启动器可以装在硬盘的 MBR (主引导记录) 中，也可以安装在活动分区的引导扇区中。

计算机加电启动时，BIOS 装载 MBR，然后从当前活动分区启动，LILO 或 GRUB 获得引导过程的控制权，并显示相应提示符，提示用户选择需要启动的操作系统。图 2-31 显示了使用 GRUB 时的引导界面。

当引导程序成功完成引导任务后，Linux 从其手中接管了 CPU 的控制权，然后 CPU 就开始执行 Linux 的核心映像代码，开始 Linux 启动过程。对于以文本方式登录的用户，Linux 将显示登录提示“`linux login:`”，等待用户输入用户名、密码登录系统。对于以图形方式登录的用户来说，Linux 将显示一个图形化的登录界面，登录成功后可以直接进入 KDE 或 Gnome 等窗口管理器。

2.3.2 以图形方式登录 Linux

对于 Red Hat Linux 9，系统默认以图形方式启动。在系统登录界面单击屏幕下方的【会话】按钮，可以选择使用的桌面系统：GNOME 或 KDE，如图 2-34 所示。



图 2-34 选择使用的桌面系统

2.3.3 设置文本登录模式

当系统引导时，启动的第一个进程是 `init`，即执行 `/sbin/init`。`init` 程序会根据 `/etc/inittab` 文件中的配置内容执行相应的脚本，完成系统的初始化工作。`/etc/inittab` 文件内容如下所示。

```
# cat /etc/inittab
# inittab      This file describes how the INIT process should set up
#              the system in a certain run-level.
#
# Author:      Miquel van Smoorenburg, <miquels@drinkel.nl.mugnet.org>
#              Modified for RHS Linux by Marc Ewing and Donnie Barnes
#
# Default runlevel. The runlevels used by RHS are:          //运行等级
# 0 - halt (Do NOT set initdefault to this)
# 1 - Single user mode
# 2 - Multiuser, without NFS (The same as 3, if you do not have networking)
# 3 - Full multiuser mode
# 4 - unused
# 5 - X11
# 6 - reboot (Do NOT set initdefault to this)
#
id:5:initdefault:
# System initialization.
si::sysinit:/etc/rc.d/rc.sysinit
10:0:wait:/etc/rc.d/rc 0
11:1:wait:/etc/rc.d/rc 1
12:2:wait:/etc/rc.d/rc 2
13:3:wait:/etc/rc.d/rc 3
14:4:wait:/etc/rc.d/rc 4
15:5:wait:/etc/rc.d/rc 5
16:6:wait:/etc/rc.d/rc 6
# Trap CTRL-ALT-DELETE                                //设置热启动
```



```
ca::ctrlaltdel:/sbin/shutdown -t3 -r now
# When our UPS tells us power has failed, assume we have a few minutes
# of power left. Schedule a shutdown for 2 minutes from now.
# This does, of course, assume you have powerd installed and your
# UPS connected and working correctly.
pf::powerfail:/sbin/shutdown -f -h +2 "Power Failure; System Shutting Down"
# If power was restored before the shutdown kicked in, cancel it.
pr:12345:powerokwait:/sbin/shutdown -c "Power Restored; Shutdown Cancelled"
# Run gettys in standard runlevels // 设置 getty
1:2345:respawn:/sbin/mingetty tty1
2:2345:respawn:/sbin/mingetty tty2
3:2345:respawn:/sbin/mingetty tty3
4:2345:respawn:/sbin/mingetty tty4
5:2345:respawn:/sbin/mingetty tty5
6:2345:respawn:/sbin/mingetty tty6
# Run xdm in runlevel 5
x:5:respawn:/etc/X11/prefdm -nodaemon
```

其中“#”表示注释行。可以看到，`/etc/inittab` 文件中每一行包括 4 个字段，字段之间以冒号分隔，格式如下所示。

id : runlevels : action : process

其中“id”为标识符，通常由 1~2 字符组成；“runlevels”表示运行级别，共 7 种，如表 2-1 所示；“action”表示执行的动作，如表 2-2 所示；“process”表示执行的进程。

表 2-1 inittab 中的 runlevels 选项说明

运 行 级 别	说 明
0	关机（不可将 <code>initdefault</code> 的值设置为 0，否则系统无法启动）
1	单用户模式
2	多用户模式，但不支持 NFS（当用户不需要网络支持时，可选用此项）
3	多用户模式，支持网络
4	保留
5	图形启动方式
6	重新启动（不可将 <code>initdefault</code> 的值设置为 6，否则系统无法正常完成启动）

表 2-2 inittab 中 action 选项说明

执 行 动 作	说 明
<code>initdefault</code>	指定默认的运行模式
<code>sysinit</code>	指定运行的初始化脚本，默认为 <code>/etc/rc.d/rc.sysinit</code>
<code>respawn</code>	若指定的进程结束，则重新执行该进程
<code>ctrlaltdel</code>	用户按下 <code>Ctrl+Alt+Del</code>

为了让系统启动时直接进入文本登录方式，可以直接修改 `/etc/inittab` 文件，将其中的“`id:5:initdefault:`”修改为“`id:3:initdefault:`”。重新启动系统后，即可以文本登录模式登录系统。

2.3.4 以文本方式登录 Linux

对于文本方式登录的用户，启动 Linux 系统后，屏幕上将快速闪过一串串包含启动信息的文字提示，在出现一系列绿色的 “[OK]” 后，系统显示如下内容提示用户登录。

```
linux login:
```

如果用户想要以超级用户的身份登录，在 “login:” 后面输入 “root”，然后回车，系统提示如下。

```
Password:
```

该提示提醒用户输入在安装时设定的系统管理员密码并回车。如果密码正确，系统将完成登录。成功登录后系统出现提示符 “#”。

一般情况下，为了安全考虑不推荐使用 root 账号登录系统。因为 root 账号权限太大，很容易由于误操作导致系统不正常。所以一般以普通用户账号登录系统。假设用户以账号 Zhangsan 登录系统，登录成功后系统显示：

```
[Zhangsan@localhost Zhangsan]$
```

可以看到，对于普通用户，系统提示符为 “\$”。其中 “@” 之前的 “Zhangsan” 表示当前登录的用户名，localhost 表示主机名，“@” 之后的 “Zhangsan” 表示当前所在的工作目录。在使用 Zhangsan 账号的过程中，如果要完成某些只有 root 账号才能执行的操作，需要临时切换为 root 账号，Linux 系统为此提供了 su 命令。su 命令的基本格式如下：

```
su 用户名
```

如果在前面所述的 Zhangsan 账号登录的情况下，希望临时以 root 身份登录，可以输入 “su root” 命令并回车，系统提示用户输入 root 的密码，输入正确后系统进入 root 登录模式。这样用户就可以完成 root 账号才能完成的操作了。

如果要退出 root 用户的临时登录，可以使用 exit 命令或者按 Ctrl+d。如前面所述，在临时用 root 身份完成希望的操作后，如果用户想要重新回到 Zhangsan 的账号下继续操作系统，输入 exit 命令并回车即可。

2.3.5 login 命令：登录系统

login 命令允许用户登录系统或者使当前用户离开现有的身份，重新登录系统。例如，用户使用新的身份重新登录系统，命令如下所示。

```
# login
```

2.3.6 startx 命令：启动图形界面

startx 是启动 X-Windows 的脚本文件，实际上是通过调用 xinit 启动 X-Windows。对于以文本方式登录的用户，可以在提示符下输入命令 startx 进入图形界面。startx 命令的语法格式如下所示。

```
startx [ [ client ] options ... ] [ -- [ server ] options ... ]
```

其中 “--” 用于分隔客户端参数与服务器参数。例如，启动服务器端具有 16 位色彩的 X-Windows，命令如下。

```
startx -- -depth 16
```



例如，设置服务器端 X-Windows 的 dpi 为 100，命令行如下所示。

```
startx -- dpi 100
```



startx 命令通常不带参数运行。

当用户以文本方式登录系统后，只要计算机的显卡、显示器、键盘和鼠标选择比较通用的型号，即可直接通过不带参数的 startx 命令启动 X-Windows，如下所示。

```
#startx
Fatal server error:
Server is already active for display 0
    If this server is no longer running, remove /tmp/.X0-lock
    and start again.
When reporting a problem related to a server crash, please send
the full server output, not just the last messages.
Please report problems to xfree86@xfree86.org.
```

因为 X-Windows 已经启动，所以会出现以上的错误，正常时将启动 X-Windows。

2.3.7 xset 命令：设置 X-windows

以文本方式登录的用户，可以在提示符下输入命令 xset 来设置 X-windows 的各项参数。xset 命令的语法格式如下所示。

```
xset [c][q][s][r][p][m][ -dpms][ +dpms][ -display][ .led]
```

其中部分常用选项的具体说明如表 2-3 所示。

表 2-3 xset 命令的常用选项说明

选 项	说 明
c	键盘点击设置
q	显示现在的设置
s	屏幕保护参数设置
r	设置自动重复
p	设置像素颜色值
m	鼠标设置
+dpms	启用电源 DPMS 特征（能源之星）
-dpms	关闭电源 DPMS 特征（能源之星）
-display	指定使用的服务器
led	设置键盘 led 灯

通过命令“xset q”可显示当前 X-Windows 的系统设置，如下所示。

```
# xset q //显示当前设置
Keyboard Control: //键盘控制
    auto repeat: on    key click percent: 0    LED mask: 00000000
```



```

auto repeat delay: 500    repeat rate: 30
auto repeating keys: 00ffffffdffffbbf
                    fadfffffdfe5ff
                    ffffffff
                    ffffffff

bell percent: 50    bell pitch: 400    bell duration: 100
Pointer Control:
  acceleration: 2/1    threshold: 4
Screen Saver:                                //屏幕保护
  prefer blanking: yes    allow exposures: yes
  timeout: 600    cycle: 600
Colors:                                        //颜色
  default colormap: 0x20    BlackPixel: 0    WhitePixel: 16777215
Font Path:                                    //字体路径
  /root/.gnome2/share/cursor-fonts,unix/:7100,/root/.gnome2/share/fonts
Bug Mode: compatibility mode is disabled
DPMS (Energy Star):
  Standby: 1200    Suspend: 1800    Off: 2400
  DPMS is Enabled
  Monitor is On
Font cache:                                    //字体缓存
  hi-mark (KB): 5120    low-mark (KB): 3840    balance (%): 70
File paths:                                    //文件路径
  Config file: /etc/X11/XF86Config
  Modules path: /usr/X11R6/lib/modules
  Log file: /var/log/XFree86.0.log

```

2.4 Linux 的终端命令窗口

早期的 Linux 系统并没有现在的 Linux 系统通常所具备的 X-Window 图形化管理界面，而只有命令行终端模式来实现人机交互。只是后来由于 Linux 系统影响力越来越大，使用的用户也越来越多，为了方便普通用户使用 Linux 系统，才设计并开发出了 X-Window 图形化界面。但是原来的命令行终端模式仍然被保留，并且发挥着非常重要的作用。

可以采用以下三种方法进入终端工作方式。

- 在图形界面下通过启动终端窗口，进入命令行工作方式。
- 通过设置 inittab 文件，在系统启动时直接进入命令行终端方式。
- 使用远程登录方式。

2.4.1 在桌面系统中使用虚拟终端

当用户登录桌面系统后，可以通过虚拟终端输入要执行的命令。要打开虚拟终端，只需打开主菜单，然后选择【系统工具】，单击【终端】选项，如图 2-35 所示。

也可以在桌面单击右键，然后在弹出的快捷菜单中选择【终端】选项，启动虚拟终端。



图 2-35 在桌面系统中启动虚拟终端

2.4.2 终端窗口的常规操作

在图 2-35 所示的终端窗口中选择【文件】|【新建窗口】或按快捷键“Shift+Ctrl+N”，可以新建一个终端窗口。选择【文件】|【新建标签】或按快捷键“Shift+Ctrl+T”可以新建一个标签，如图 2-36 所示。

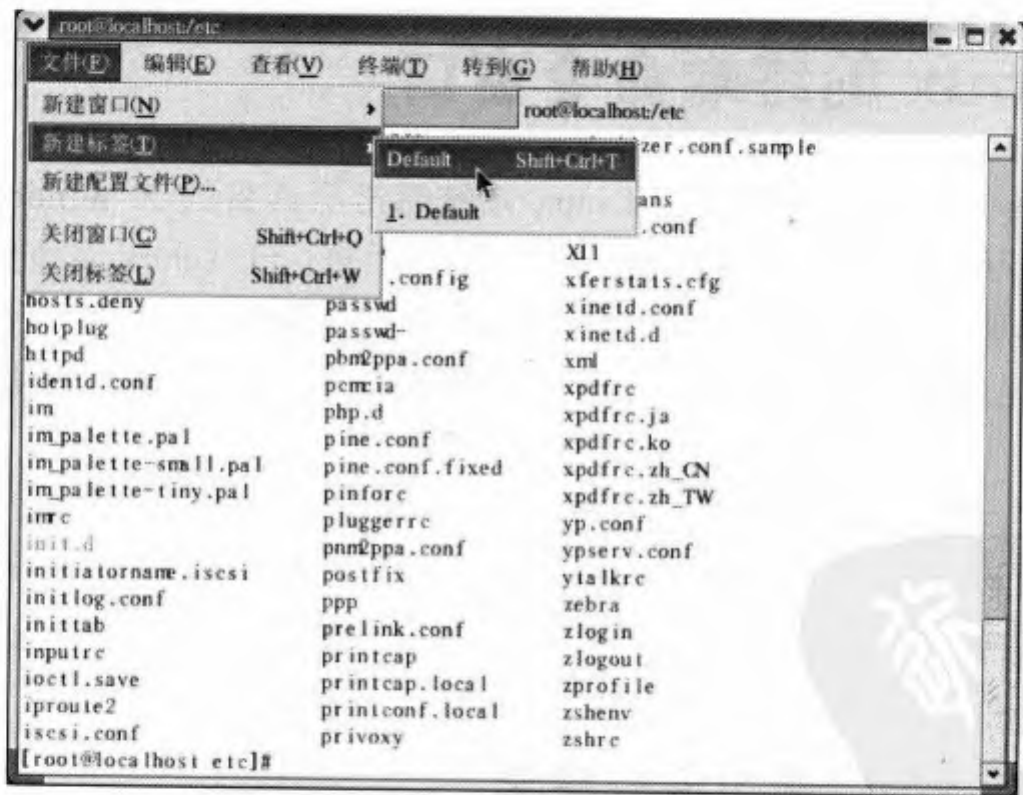


图 2-36 新建一个标签

终端窗口具有一些文本编辑器的特征，窗口的右边是一个滚动条，用户可以通过它来查看以前输入的命令及产生的执行结果。利用图 2-36 中【编辑】菜单的【复制】与【粘贴】功能，可以复制任何以前显示过的文本，编辑后还可以组成更复杂的命令。利用复制与粘贴命

令还可以在不同标签、不同终端窗口中完成复制。

在编辑命令行时，新键入的字符会出现在光标所在的位置，可以使用左右方向键把光标从命令行的一端移动到另一端，也可以按上下方向键在不同行之间移动。表 2-4 列出了可以用来移动光标的按键组合。

表 2-4 移动光标的快捷键

快 捷 键	功 能	说 明
Ctrl+f	字符前移	向前移动一个字符
Ctrl+b	字符后移	向后移动一个字符
Alt+f	单词前移	向前移动一个单词
Alt+b	单词后移	向后移动一个单词
Ctrl+a	行起始	移动到当前行的行首
Ctrl+e	行结尾	移动到当前行的行尾
Ctrl+l	清屏	清屏，并在屏幕的最上面开始一个新行

表 2-5 列出了编辑命令行输入的一些快捷方式。

表 2-5 编辑命令行输入的快捷方式

快 捷 键	功 能	说 明
Ctrl+d	删除当前位	删除当前的字符
Backspace	删除前一位	删除前一个字符
Ctrl+t	调换字符	交换当前字符和前一个字符的位置
Alt+t	调换单词	交换当前单词和前一单词的位置
Alt+u	大写单词	把当前单词变成大写
Alt+l	小写单词	把当前单词变成小写
Alt+c	首字母大写单词	把当前单词变成首字母大写的单词
Ctrl+v	插入特殊字符	添加一个特殊字符。例如，要添加一个制表符，按 Ctrl+v+Tab

表 2-6 列出了在命令行中实现剪切和粘贴的快捷方式。

表 2-6 命令行中剪切和粘贴的快捷方式

快 捷 键	功 能	说 明
Ctrl+k	剪切至行末	剪切文本直到行的末尾
Ctrl+u	剪切至行首	剪切文本直到行的起始
Ctrl+w	剪切前个单词	剪切光标前的单词
Alt+d	剪切下一个单词	剪切光标后的单词
Ctrl+y	粘贴新近文本	粘贴最近剪切的文本
Alt+y	粘贴先前的文本	回退到先前剪切的文体并粘贴它
Ctrl+c	删除整行	删除整行



在 Red Hat Linux 中，命令行是区分大小写的。例如，系统会认为“ls”与“Ls”是不同的两个名字。

2.5 命令行自动补全

为了简化打字工作，Bash Shell 提供了几种可以对输入不完整的文本进行自动补全的方法。如果想对输入进行自动补全，只需输入初始的几个字符，然后按“Tab”键，系统会自动匹配所需的其余输入。当有多种匹配时，系统会给出警告提示，按“ESC+?”组合键或按两次“Tab”键，可以列出所有可能的匹配。自动补全可以应用在下面 4 类输入工作中。

2.5.1 环境变量名补全

如果输入的文本以“\$”开始，Shell 就以当前 Shell 的一个环境变量补全文本，例如如下所示代码。

```
# echo $P <Tab><Tab> //按两次"Tab"键，系统列出环境变量中所有第一个字母为
                        //“P”的可能匹配
                        //在显示所有的可能性之后，返回原来命令行，等待用户的
                        //选择
$PATH                  $PRELINK_OPTS
$PIPESTATUS            $PROMPT_COMMAND
$PPID                  $PS1
$PRELINK_FULL_TIME_INTERVAL $PS2
$PRELINKING            $PS4
$PRELINK_NONRPM_CHECK_INTERVAL $PWD
# echo $PA <Tab> //匹配了 PATH 环境变量
/usr/lib/qt-3.3/bin:/usr/kerberos/sbin:/usr/kerberos/bin:/usr/local/sbin:/usr/local
/bin:/sbin:/bin:/usr/sbin:/usr/bin:/usr/X11R6/bin:/root/bin
```

如果仅输入“\$”，则系统会按字母顺序列出所有环境变量，如下所示。

```
$ $ <Tab><Tab> //按两次"Tab"键，系统列出所有环境变量
$_
$BASH
$BASH_ARGC
$BASH_ARGV
$BASH_COMMAND
$BASH_LINENO
$BASH_SOURCE
$BASH_SUBSHELL
$BASH_VERSION
$COLORS
$COLORTERM
...
```

2.5.2 用户名补全

如果输入的文本以波浪线“~”开始，Shell 会以用户名补全文本，例如下列代码。


```
# cd ~s <Tab><Tab> //按两次"Tab", 系统列出所有以字母"s"开头的用户名
~sabayon ~smmsp/ ~sshd/
~shutdown/ ~squid/ ~student1/ ~sync/
# cd ~st <Tab><Tab> //按两次"Tab", 匹配了用户名 student1
#pwd
/student1
```

2.5.3 命令、别名或函数的名字补全

如果文本以常规字符开始，Shell 将尝试利用命令、别名或函数名来补全文本，例如下列代码。

```
# ls
chage_li man_chage student1 student3 teacher1
# cd t <Tab> //按一次"Tab", 匹配了 teacher1 目录名
#pwd
/home/teacher1
```

2.5.4 主机名补全

如果输入的文本以“@”符号开始，系统会利用/etc/hosts 文件中的主机名来补全文本，例如下列代码。

```
# mail root@ <Tab><Tab> //按两次"Tab", 系统列出所有可用的主机名
@::1 @localhost @localhost.localdomain
@localhost @localhost6.localhost6
```

表 2-7 列出了自动补全的快捷方式及其说明。

表 2-7 自动补全快捷方式及说明

快 捷 键	说 明
Alt+~	以一个用户名补全此处的文本
Alt+\$	以一个变量补全此处的文本
Alt+@	以一个主机名补全此处的文本
Ctrl+x+~	列出可能的用户名补全
Ctrl+x+\$	列出可能的环境变量补全
Ctrl+x+@	列出可能的主机名补全
Ctrl+x+!	列出可能的命令名补全

2.6 退出 Linux

Linux 提供了 3 个退出系统的命令：shutdown、halt 和 reboot。这 3 个命令在一般情况下只有系统的超级用户（一般是指 root）才可以执行。命令 exit 及 logout 用于重新登录，类似于 Windows 下的用户注销。

2.6.1 shutdown 命令：关机或重新启动

shutdown 命令可以用来关机或重新启动系统。直接输入没有参数的 shutdown 命令，两



分钟之后即可关闭系统。在这段时间，Linux 将提示所有已经登录系统的用户系统将要退出，该命令的一般格式如下。

```
shutdown [-akrhfnc] [-t secs] [time] [warning message]
```

其中所使用的选项含义如表 2-8 所示。

表 2-8 shutdown 命令的可用选项说明

选 项	说 明
-a	权限设置
-c	中断关机
-f	重启时不进行磁盘检测
-F	重启时进行磁盘检测
-h	关机
-n	不调用 init 关机
-H	关机后关闭电源
-k	测试
-r	重启
-t secs	延迟指定的时间
time	指定时间后关机
warning message	关机时显示警告信息

其中，参数“time”可以使用的选项说明如下。

- now: 立即退出系统。
- mins: 在指定的分钟之后退出系统。
- hh:ss: 在指定的时间退出系统。

例如，下面命令表示系统在 3 分钟之后关闭，并在屏幕显示字符串“I am going to poweroff, byebye”。

```
# shutdown 3 "I am going to poweroff" //3min 后关闭系统
Broadcast message from root (pts/1) (Thu May 29 02:55:04 2008):
I am going to poweroff
The system is going DOWN to maintenance mode in 3 minutes!
Shutdown cancelled. //按 Ctrl+C 组合键取消 shutdown 命令
```

例如，下面命令表示系统在 3min 后重新启动，在重启之前给所有用户发送提示信息“I am going to reboot”。

```
# shutdown 3 -r "I am going to reboot" //3min 后系统重启
Broadcast message from root (pts/1) (Thu May 29 03:18:25 2008):
I am going to reboot
The system is going DOWN for reboot in 3 minutes!
```

在关掉系统之前，系统会自动产生一个/etc/nologin 文件，用于说明系统即将关闭，普通用户不能再登录系统。在这段时间中，只有系统管理员可以进入。若要获得有关 shutdown 命令更详细的帮助，可以使用命令“man shutdown”。



在关机尚未真正执行时，通过按 Ctrl+C，可以随时终止命令的执行。

2.6.2 halt 命令：关闭系统

halt 命令相当于“shutdown -h now”，表示立刻关机，通过选项还可以设置是否重启或关闭电源。halt 命令的语法格式如下所示。

```
halt [-n] [-w] [-d] [-f] [-i] [-p]
```

其中可用选项的具体说明如表 2-9 所示。

表 2-9 halt 命令的可选参数

选 项	说 明
-d	不在 wtmp 中记录
-f	强制关机或重启，不调用 shutdown
-i	关闭系统前先关闭网络系统
-n	不执行 sync
-p	halt 后执行 poweroff
-w	在 wtmp 中记录，不关闭系统

例如，直接使用命令 halt 关闭系统，命令如下。

```
# halt //关闭系统
```

下面的命令在关闭系统的同时关闭电源。

```
# halt -p //关闭系统并关闭电源
```

2.6.3 reboot 命令：重新启动系统

reboot 命令相当于“shutdown -r now”，表示立刻重启计算机。reboot 命令的语法格式如下。

```
reboot [-n] [-w] [-d] [-f] [-i] [-p]
```

其中可用选项的具体说明如表 2-10 所示。

表 2-10 reboot 命令的可选参数

选 项	说 明
-d	重启时不把数据写入/var/tmp/wtmp
-f	强制执行命令
-i	关闭网络设置之后再重启
-n	保存数据后再重启
-w	测试
-p	关闭系统

例如，下面命令将重启计算机。



```
-sh-2.05b# reboot //重启
Broadcast message from root (pts/0) (Sun May 27 22:47:48 2007):
The system is going down for reboot NOW!
-sh-2.05b#
```



对于 Red Hat Linux 9，使用命令“init 6”也可以立即重启主机。

2.6.4 exit 命令：重新登录

exit 命令用来退出当前的 Shell 环境，即重新登录系统。该命令与 logout 命令功能类似，也等同于 Ctrl+d 组合键。例如，退出当前 Shell 并重新登录系统，命令如下所示。

```
$ exit //退出
logout
Connection closed by foreign host.
```

2.6.5 logout 命令：注销用户

logout 命令让当前用户退出系统，功能与命令 login 相对应，与命令 exit 类似。例如，注销当前用户，重新登录系统，命令如下所示。

```
#logout //退出
Red Hat Linux release 9 (Shrike)
Kernel 2.4.20-8 on an i686
login:
```

2.7 联机帮助命令 man

Red Hat Linux 9 系统提供了丰富的联机帮助（包括手册和信息文档页），可以使用 man 和 info 等命令对其进行查询。

2.7.1 命令格式

man 命令用于查看 Linux 系统的手册。手册是 Linux 中广泛使用的联机帮助形式，其中不仅包括了常用的命令帮助说明，还包括配置文件、设备文件、协议和库函数等多种信息。man 命令的语法格式如下所示。

```
man [-adfhktwW] [section] [-M path] [-P pager] [-S list] [-m system] [-p string] [name]
```

其中各选项的具体说明如表 2-11 所示。

表 2-11 man 命令选项及其说明

选 项	说 明
-a	man 默认在显示第一个搜索到的手册之后自动退出，如果使用该选项会强制 man 继续显示所有匹配的内容，即找出所有符合的匹配
-d	不显示手册，只显示错误信息
-c	强制对手册重新排版

续表

选 项	说 明
-K	在所有的联机手册中搜寻指定的字符串。一般会比较慢，大约每搜索 500 页需要 1min 的时间，如果指定章节会加快搜索速度
-k	同 apropos
-h	打印帮助信息并退出
-f	同 whatis
-t	将联机手册重新排版，并将结果输出到标准输出设备
-w	不显示联机手册内容，只显示手册的位置
-W	与-w 相同，但每行打印一个文件名，不显示其他信息
--path	同-w
-H	设定以文本方式显示 HTML 文件时所使用的命令，其设定值的优先级高于环境变量 HTMLPAGER，man 默认使用/bin/cat
-m system	按照给定的 system 名，指定另一组联机手册
-C config_file	设定使用的配置文件，系统默认是/etc/man.config，有关配置文件的详细说明可以查看 man.config(5)
-D	显示并打印 debug 信息
-M pathlist	设定搜索手册的路径，多个路径之间用冒号分隔。如果搜索路径为空，则“-M”不起作用。如果没有指定路径，使用环境变量 MANPATH 的设定值；如果 MANPATH 没有指定，则使用/usr/lib/man.conf 内的设定值
-P pager	设定使用何种 page，其优先级高于环境变量 MANPAGER 和 PAGE 的设定值。man 默认使用/usr/bin/less -is
-S section_list	设定搜索的章节列表，用冒号分隔，其设定值的优先级高于环境变量 MANSECT
-B	设定使用 HTML 文件时的浏览器，其优先级高于环境变量 BROWSER 的设定值。man 默认使用/usr/bin/less -is
name	指定查询的命令名

2.7.2 常规操作

例如，使用 man 查看 clear 命令的帮助，如下所示。

```
# man clear
clear(1)
NAME
    clear - clear the terminal screen //清空终端屏幕命令——clear
SYNOPSIS
    clear //语法
DESCRIPTION
    clear clears your screen if this is possible. It looks in the environ-
    ment for the terminal type and then in the terminfo database to figure
    out how to clear the screen.
    clear ignores any command-line parameters that may be present.
SEE ALSO
    tput(1), terminfo(5) //也可以查看man1中的tput和man5中的terminfo命令
    This describes ncurses version 5.5 (patch 20060715).
```

也可以查看 man 命令自身的手册帮助信息，如下所示。

```
#man man
man(1)
NAME
```



```
man - format and display the on-line manual pages      //在线手册页——man
SYNOPSIS                                              //man 格式
man [-acdfFhkKtwW] [--path] [-m system] [-p string] [-C config_file]
  [-M pathlist] [-P pager] [-B browser] [-H htmlpager] [-S section_list]
  [section] name ...
DESCRIPTION
man formats and displays the on-line manual pages. If you specify sec-
tion, man only looks in that section of the manual. name is normally
the name of the manual page, which is typically the name of a command,
function, or file. However, if name contains a slash (/) then man
interprets it as a file specification, so that you can do man ./foo.5
or even man /cd/foo/bar.1.gz.
See below for a description of where man looks for the manual page
files.
OPTIONS                                              //man 选项
  -C config_file
... ..
```

可以看到，手册一般包括 NAME、DESCRIPTION、FILES 和 SEE ALSO 等几个部分，按“q”键可以退出 man 命令的交互界面。

2.7.3 按章节查询

man 手册的存储位置定义在/etc/man.config 文件中，一般按类型存放在/usr/share/man 目录下，组成不同的章节，如下所示。

```
# ls -d /usr/share/man/man?
/usr/share/man/man1 /usr/share/man/man5 /usr/share/man/man9
/usr/share/man/man2 /usr/share/man/man6 /usr/share/man/mann
/usr/share/man/man3 /usr/share/man/man7
/usr/share/man/man4 /usr/share/man/man8
```

其中，每一个目录下都存放着对应类型的手册文件，手册文件大多为“.gz”格式的压缩文件，采用“手册名称.章节.gz”的命名规则。例如，查看第 2 章所包括的手册文件，命令如下所示。

```
#ls /usr/share/man/man2
accept.2.gz          ioprio_get.2.gz      sched_setparam.2.gz
access.2.gz          ioprio_set.2.gz      sched_setscheduler.2.gz
acct.2.gz            io_setup.2.gz        sched_yield.2.gz
add_key.2.gz         io_submit.2.gz       security.2.gz
... ..
```

手册分为 man1~man9 共 9 个章节，对应 9 种类型，其说明如表 2-12 所示。

表 2-12 手册章节说明

章 节	说 明
man1	提供给普通用户使用的可执行命令说明
man2	系统调用、内核函数的说明
man3	子程序、库函数说明
man4	系统设备手册，包括“/dev”目录中的设备文件的参考说明
man5	配置文件格式手册，包括“/etc”目录下各种配置文件的格式说明

续表

章 节	说 明
man6	游戏的说明手册
man7	协议转换手册
man8	系统管理工具手册，这些工具只有 root 用户才可以使用
man9	Linux 系统例程手册

如果指定章节，man 命令只在指定的章节中寻找，可以使用“man N intro”命令查看某一章手册的说明信息，其中“N”取值范围为 1~9，与手册章节相对应。例如，查看第 4 章手册的说明信息如下。

```
#man 4 intro
INTRO(4)                                Linux Programmer's Manual          INTRO(4)
NAME
    intro - Introduction to special files
DESCRIPTION
    This chapter describes special files.
FILES
    /dev/* - device files                //设备文件，包括/dev
AUTHORS
    Look at the header of the manual page for the author(s) and copyright
    conditions. Note that these can be different from page to page!
SEE ALSO
    standards(7)
Linux                                1993-07-24                                INTRO(4)
(END)
```

如果在不同的章节中有相同的说明项，可以在使用 man 命令的同时指定手册章节。例如，passwd 命令在 man1 和 man5 中均有帮助说明，若查看 passwd 命令在手册第 5 章中的帮助说明，可以使用如下命令。

```
#man 5 passwd
PASSWD(5)                                Linux Programmer's Manual          PASSWD(5)
NAME
    passwd - password file
DESCRIPTION
    //passwd(5) 描述内容如下
    Passwd is a text file, that contains a list of the system's accounts,
    giving for each account some useful information like user ID, group ID,
    home directory, shell, etc. Often, it also contains the encrypted
    passwords for each account. It should have general read permission
    (many utilities, like ls(1) use it to map user IDs to user names), but
    write access only for the superuser.
    In the good old days there was no great problem with this general read
    permission. Everybody could read the encrypted passwords, but the
    hardware was too slow to crack a well-chosen password, and moreover,
    the basic assumption used to be that of a friendly user-community.
    These days many people run some version of the shadow password suite,
    where /etc/passwd has asterisks (*) instead of encrypted passwords, and
    the encrypted passwords are in /etc/shadow which is readable by the
    superuser only.
    Regardless of whether shadow passwords are used, many sysadmins use an
    asterisk in the encrypted password field to make sure that this user
    can not authenticate him- or herself using a password. (But see the
    Notes below.)
```



If you create a new login, first put an asterisk in the password field, then use passwd(1) to set it. //在创建一个新的用户时,应在password字段中添加一个“*”符//号,并用man1中介绍的passwd命令设置密码

2.7.4 man 文件的输出

man 虽然具有强大的在线查询功能,但由于 man 文件的格式不是一般的文本文件,所以很难直接将帮助信息进行打印。通常采用的方法是将 man 的执行结果通过输出重定向,导入到另一个文本文件中进行编辑打印。例如,将 ls 命令的帮助信息输出到 ls_help 文件中,命令行为:

```
#man ls > /ls_help
```

2.8 其他联机帮助命令

除了 man 命令以外,Linux 还提供了其他一些常用联机帮助命令,包括 info、help、whereis、whatis 及 apropos 等。

2.8.1 apropos 命令: 基于字符串查找 whatis 数据库

apropos 命令在 whatis 数据库中进行搜索,找出并显示包含所输入的字符串的所有数据。apropos 命令与“man -k”命令是完全等同的,显示的结果也完全相同。

apropos 命令基于字符串,而 whatis 命令基于关键词,因而 apropos 命令通常会显示比 whatis 命令更多的信息。例如,使用 apropos 命令查询命令 cd 的帮助信息,命令如下所示。

```
# apropos cd
/etc/nscd.conf [nscd] (5) - name service cache daemon configuration file
/usr/sbin/nscd [nscd] (8) - name service cache daemon
Encode::EBCDIC (3pm) - EBCDIC Encodings
Inline::C::ParseRecDescent (3pm) - The Classic Inline::C Parser
...
#man -k cd
/etc/nscd.conf [nscd] (5) - name service cache daemon configuration file
/usr/sbin/nscd [nscd] (8) - name service cache daemon
Encode::EBCDIC (3pm) - EBCDIC Encodings
Inline::C::ParseRecDescent (3pm) - The Classic Inline::C Parser
...

```

2.8.2 help 命令: 显示内建命令的帮助信息

Linux 内建命令数量众多,但没有独立的帮助文件。help 命令提供了对这些内建命令的在线帮助支持。help 命令格式如下。

```
help [-s] [pattern...]
```

其中,选项“-s”表示只显示命令的格式信息,“pattern”表示要查询的内容。例如,查询命令 cd 的命令格式,可以输入如下命令。

```
#help -s cd
cd: cd [-L|-P] [dir]
```

如果要查看命令 cd 的详细帮助信息,在命令行输入如下命令。

```
# help cd
cd: cd [-L|-P] [dir]
```


Change the current directory to DIR. The variable \$HOME is the default DIR. The variable CDPATH defines the search path for the directory containing DIR. Alternative directory names in CDPATH are separated by a colon (:). A null directory name is the same as the current directory, i.e. `.`. If DIR begins with a slash (/), then CDPATH is not used. If the directory is not found, and the shell option `cdable_vars' is set, then try the word as a variable name. If that variable has a value, then cd to the value of that variable. The -P option says to use the physical directory structure instead of following symbolic links; the -L option forces symbolic links to be followed.

直接输入 help 命令, 则显示所有支持的帮助信息, 如下所示。

```
# help
GNU bash, version 2.05b.0(1)-release (i386-redhat-linux-gnu)
These shell commands are defined internally. Type 'help' to see this list.
Type 'help name' to find out more about the function 'name'.
Use 'info bash' to find out more about the shell in general.
Use 'man -k' or 'info' to find out more about commands not in this list.
```

A star (*) next to a name means that the command is disabled.

//命令名前面的

“*” 标志, 表示该命令不可用

```
%[DIGITS | WORD] [&]          (( expression ))
. filename                      :
[ arg... ]                      [[ expression ]]
alias [-p] [name[=value] ...]  bg [job_spec]
bind [-lpvsPVS] [-m keymap] [-f fi break [n]
builtin [shell-builtin [arg ...]] case WORD in [PATTERN [| PATTERN]].
cd [-L|-P] [dir]                command [-pVv] command [arg ...]
compgen [-abdefghjksuv] [-o option complete [-abdefghjksuv] [-pr] [-o
continue [n]                    declare [-afFirtx] [-p] name[=valu
dirs [-clpv] [+N] [-N]          disown [-h] [-ar] [jobspec ...]
echo [-neE] [arg ...]           enable [-pnds] [-a] [-f filename]
eval [arg ...]                  exec [-cl] [-a name] file [redirec
exit [n]                        export [-nf] [name[=value] ...] or
false                           fc [-e ename] [-nlr] [first] [last
fg [job_spec]                   for NAME [in WORDS ... ;] do COMMA
for (( exp1; exp2; exp3 )); do COM function NAME { COMMANDS ; } or NA
getopts optstring name [arg]     hash [-lr] [-p pathname] [-dt] [na
help [-s] [pattern ...]         history [-c] [-d offset] [n] or hi
if COMMANDS; then COMMANDS; [ elif jobs [-lnprs] [jobspec ...] or job
kill [-s sigspec | -n signum | -si let arg [arg ...]
local name[=value] ...          logout
popd [+N | -N] [-n]             printf format [arguments]
pushd [dir | +N | -N] [-n]      pwd [-PL]
read [-ers] [-u fd] [-t timeout] [ readonly [-anf] [name[=value] ...]
return [n]                      select NAME [in WORDS ... ;] do CO
set [--abefhkmnptuvxBCHP] [-o opti shift [n]
shopt [-pqsu] [-o long-option] opt source filename
suspend [-f]                    test [expr]
time [-p] PIPELINE              times
trap [arg] [signal_spec ...] or tr true
type [-afptP] name [name ...]   typeset [-afFirtx] [-p] name[=valu
ulimit [-SHacdflmnpstuv] [limit] umask [-p] [-S] [mode]
unalias [-a] [name ...]         unset [-f] [-v] [name ...]
until COMMANDS; do COMMANDS; done variables - Some variable names an
wait [n]                        while COMMANDS; do COMMANDS; done
{ COMMANDS ; }
```



直接在待查询的命令后带上选项 “--help”，也可查询该命令的帮助信息。例如，查询命令 `mkdir` 的帮助信息，命令行如下所示。

```
# mkdir --help
用法: mkdir [选项] 目录...
若目录不是已经存在则创建目录。
  -Z, --context=CONTEXT (SELinux) set security context to CONTEXT
长选项必须用的参数在使用短选项时也是必须的。
  -m, --mode=模式      设定权限<模式> (类似 chmod)，而不是 rwxrwxrwx 减 umask
  -p, --parents        需要时创建上层目录，如目录早已存在则不当作错误
  -v, --verbose        每次创建新目录都显示信息
      --help          显示此帮助信息并退出
      --version       输出版本信息并退出
请向 <bug-coreutils@gnu.org> 报告错误。
```

`help` 命令也可以查询自身的帮助信息，如下所示。

```
# help help
help: help [-s] [pattern ...]
    Display helpful information about builtin commands.  If PATTERN is
    specified, gives detailed help on all commands matching PATTERN,
    otherwise a list of the builtins is printed.  The -s option
    restricts the output for each builtin command matching PATTERN to
    a short usage synopsis.
```

2.8.3 info 命令：查看 info 格式帮助信息

`info` 文档是 Linux 系统提供的另一种格式的帮助信息，与 `man` 手册相比有更强的交互性。使用 `info` 命令可以查看 Texinfo 格式的帮助文档。`info` 文档通常存放在 `/usr/share/info` 目录中，如下所示。

```
#ls
a2ps.info.gz      emacs-16.gz      grub.info.gz
accounting.info.gz  emacs-17.gz      gzip.info.gz
ada-mode.gz        emacs-18.gz      history.info.gz
annotate.info.gz   emacs-19.gz      idlwave.gz
```

`info` 命令语法格式如下所示。

```
info
[--apropos=STRING] [--directory=DIR] [--dribble=FILENAME] [--file=FILENAME] [--help]
[--vi-keys] [--version]
[--index-search=STRING] [--node=NODONAME] [--output=FILENAME] [--restore=FILENAME]
[--subnodes] [--show-options, --usage]
```

命令 `info` 的选项说明如表 2-13 所示。

表 2-13 info 的选项说明

选 项	说 明
--apropos=STRING	在所有手册的所有索引中搜索 STRING
-d, --directory=DIR	将目录 DIR 添加到 INFOPATH 中
--dribble=FILENAME	将用户的按键记录在文件 FILENAME 中
-f, --file=FILENAME	指定要阅读的 Info 文件
-h, --help	显示该求助信息并退出

续表

选 项	说 明
--index-search=STRING	转移到索引条目 STRING 所指向的节点
-n, --node=NODENAME	指定首先阅读的 Info 文件中的节点
-o, --output=FILENAME	将选中的节点输出到文件 FILENAME 中
-R, --raw-escapes	不要删除来自 man 页面的 ANSI 转义符
--restore=FILENAME	从文件 FILENAME 中读取初始按键
-O, --show-options, --usage	转移到命令行选项节点
--subnodes	递归输出菜单条目
--vi-keys	使用类 vi 和类 less 的按键绑定
--version	显示版本信息并退出

例如，查找 passwd 的帮助信息，命令行如下。

```
#info passwd
PASSWD(1)                                User utilities                                PASSWD(1)
NAME
    passwd - update a user's authentication tokens(s)
SYNOPSIS
    passwd [-k] [-l] [-u [-f]] [-d] [-n mindays] [-x maxdays] [-w warndays]
    [-i inactivedays] [-S] [--stdin] [username]
... ..
```

info 命令支持文件的链接跳转，使用方向键在显示的帮助文档中选择需要进一步查看的文件名，并按回车键，被选择的文件就会自动打开。

除了 info 命令外，还可以使用 pinfo 命令查看 info 文档。pinfo 命令采用了 Lynx 浏览器风格，使得查看文档的操作更加简单。pinfo 支持彩色显示链接文件，支持鼠标功能。可以直接使用 pinfo 命令查看帮助文件列表，如图 2-37 所示。



图 2-37 使用 pinfo 命令查看帮助



2.8.4 whereis 命令：查询文件存储位置

whereis 命令顾名思义是用来查询文件存储位置的命令，通常用来查找一个命令的二进制文件、源文件或帮助文件在系统中的位置。whereis 命令的语法格式如下所示。

```
whereis [-s][-b][-m] [-S dir] [-M dir][][-f ] [name]
```

其中 whereis 命令的选项说明如表 2-14 所示。

表 2-14 whereis 命令的选项说明

选 项	说 明
-b	只匹配二进制文件
-f	只显示文件的真实名称
-m	只查找帮助文件
-s	只查找 source 文件
-M dir	设置查找目录
-S dir	设置查找目录
name	指定查询的文件

如果不带任何选项，则查找并显示所有相关的文件。例如，查找命令 mkdir 的相关文件如下所示。

```
# whereis mkdir
mkdir: /bin/mkdir /usr/share/man/man1p/mkdir.1p.gz /usr/share/man/man3p/ mkdir.3p.gz
/usr/share/man/man1/mkdir.1.gz /usr/share/man/man2/mkdir.2.gz
```

只查找与命令 mkdir 相关的二进制文件，如下所示。

```
# whereis -b mkdir
mkdir: /bin/mkdir
```

只查找与命令 mkdir 相关的帮助文件，如下所示。

```
# whereis -m mkdir
mkdir: /usr/share/man/man1p/mkdir.1p.gz /usr/share/man/man3p/mkdir.3p.gz /usr/share/man/man1
/mkdir.1.gz
/usr/share/man/man2/mkdir.2.gz
```

2.8.5 whatis 命令：查找简洁的帮助信息

与 man 或 info 相比，whatis 命令可以提供更加简洁的帮助信息。whatis 命令在 whatis 数据库中进行查找，并显示与所输入的关键词相关的信息。whatis 命令格式如下。

```
whatis [keyword...]
```

其中“keyword”表示待查询的关键词。在使用 whatis 命令前应建立 whatis 数据库。该数据库只有系统管理员才能建立，建立所需的时间视系统软硬件性能而定。建立 whatis 数据库的命令如下所示。

```
#makewhatis
```

whatis 数据库建立后即可使用 whatis 命令进行查询。例如，查询 cd 命令的帮助信息，如下所示。

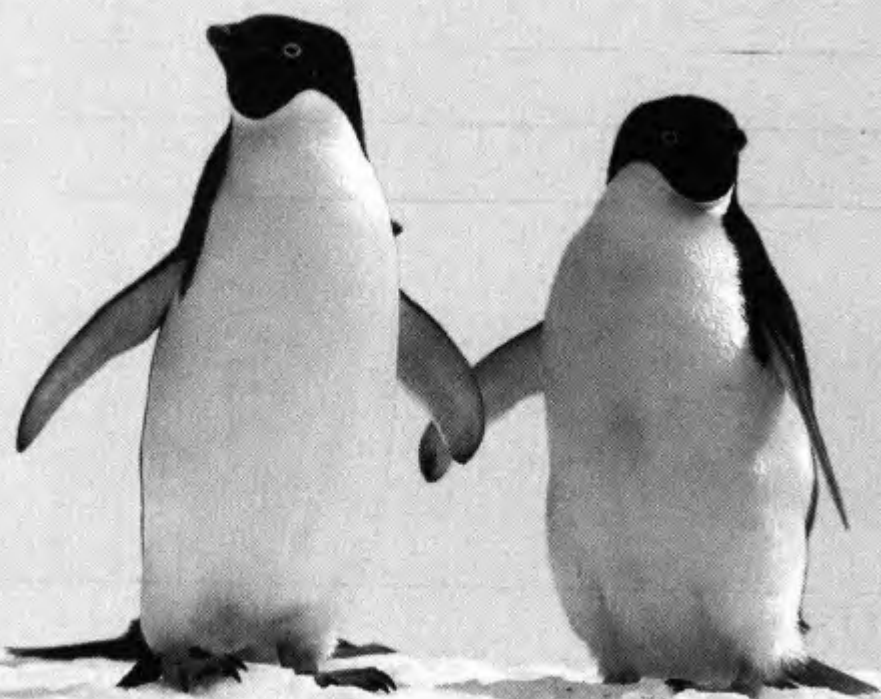
```
# whatis cd
cd (n) - Change working directory
cd [builtins] (1) - bash built-in commands, see bash(1)
```


LINUX

第3章

系统管理与设置

系统管理员的一项重要工作就是尽可能提高系统的性能，最大化资源的使用效率，并及时对系统进行更新和维护。Linux 系统提供了一些系统资源管理与配置的命令和工具，综合利用这些工具可以有效提高系统的运行效率，降低管理成本。





3.1 常用的信息显示命令

在 Red Hat Linux 9 中，常用的信息显示命令大约有 50 多个。通过信息显示命令，系统管理员可以直接了解系统的工作状态。

3.1.1 arch 命令：显示主机的体系结构

arch 命令等同于命令“uname -m”，用于查询并显示主机的体系结构。在当前的 Linux 系统下，arch 命令的输出结果有：i386、i486、i586、alpha、sparc、arm、m68k、mips、ppc、i686 等。

例如，显示当前系统的体系结构，命令如下所示。

```
# arch //打印当前系统的体系结构
i686
```

3.1.2 cal 命令：显示日历

cal 命令用于显示日历，配合选项“-y”可以显示全年的年历。cal 命令的语法格式如下所示。

```
cal [-l][-3][-s][-m][-j][-y] [[month] year]
```

其中各选项说明如表 3-1 所示。

表 3-1 cal 命令的选项说明

选 项	说 明
-l	显示单月份信息，默认设置
-3	显示前一月、当前月及下一月月历信息
-s	显示输出时将星期日作为每周的第一天，默认设置
-m	显示输出将星期一作为每周的第一天
-j	按 Julian 日期进行显示，显示距 1 月 1 日的天数，即从 1 月 1 日开始计算
-y	显示整年日历
month	月份
year	年份

例如，显示当前月份，使用如下命令。

```
#cal
七月 2007
日 一 二 三 四 五 六
          1  2
3  4  5  6  7  8  9
```



```
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
31
```

例如，按 Julian 日期表示形式（显示距 1 月 1 日的天数）显示 2007 年 8 月的日历，如下所示。

```
$ cal -j 8 2007
      八月 2007
日 一 二 三 四 五 六
    213 214 215 216 217 218
219 220 221 222 223 224 225
226 227 228 229 230 231 232
233 234 235 236 237 238 239
240 241 242 243
```

例如，显示前一月、当前月以及下一月份的月历，如下所示。

```
# cal -3
      四月 2008          五月 2008          六月 2008
日 一 二 三 四 五 六  日 一 二 三 四 五 六  日 一 二 三 四 五 六
    1 2 3 4 5          1 2 3          1 2 3 4 5 6 7
6 7 8 9 10 11 12      4 5 6 7 8 9 10      8 9 10 11 12 13 14
13 14 15 16 17 18 19  11 12 13 14 15 16 17  15 16 17 18 19 20 21
20 21 22 23 24 25 26  18 19 20 21 22 23 24  22 23 24 25 26 27 28
27 28 29 30          25 26 27 28 29 30 31  29 30
```

3.1.3 cat /proc/meminfo: 查看内存使用情况

文件/proc/meminfo 保存了当前系统内存使用情况的相关信息。配合使用 cat 命令可以查看内存使用情况。

例如，查看系统内存使用情况，命令如下所示。

```
# cat /proc/meminfo
MemTotal:      252444 kB          //总内存 256MB
MemFree:       6428 kB
Buffers:       1828 kB
Cached:        63032 kB
SwapCached:    0 kB
Active:        189776 kB
Inactive:      17136 kB
HighTotal:     0 kB
HighFree:      0 kB
LowTotal:      252444 kB
LowFree:       6428 kB
SwapTotal:     589816 kB          //交换分区约为内存的 2~2.5 倍
SwapFree:      589708 kB
Dirty:         36 kB
Writeback:     0 kB
AnonPages:     142076 kB
Mapped:        45352 kB
Slab:          19172 kB
PageTables:    5960 kB
NFS_Unstable:  0 kB
Bounce:        0 kB
CommitLimit:   716036 kB
```




```
Committed_AS: 526868 kB
VmallocTotal: 598008 kB
VmallocUsed: 4356 kB
VmallocChunk: 593420 kB
```



/proc 目录下的文件不是真正意义上的文件，关闭系统后，该目录中的文件不再存在。该目录文件通常用来帮助查看内核运行信息。

3.1.4 cat /proc/modules: 查看当前系统模块

文件/proc/modules 保存了当前系统中被加载模块的相关信息。配合使用 cat 命令可以查看当前系统加载模块的情况。

例如，查看系统当前加载的模块，命令如下所示。

```
# cat /proc/modules
bridge 53725 0 - Live 0xd0c6c000
netloop 10817 0 - Live 0xd0bd3000
netbk 78017 0 [permanent], Live 0xd0c46000
blktap 385125 2 [permanent], Live 0xd0bdc000
blkbk 21089 0 [permanent], Live 0xd0b9c000
autofs4 23749 2 - Live 0xd0b84000
hidp 23105 2 - Live 0xd0ace000
rfcomm 42457 0 - Live 0xd0bc7000
l2cap 29505 10 hidp,rfcomm, Live 0xd0b65000
bluetooth 53925 5 hidp,rfcomm,l2cap, Live 0xd0b8d000 //蓝牙设备支持
sunrpc 142973 1 - Live 0xd0ba3000
ip_conntrack_ftp 11697 0 - Live 0xd0b61000
ip_conntrack_netbios_ns 6977 0 - Live 0xd0abc000
ipt_REJECT 9537 1 - Live 0xd0b5d000
xt_state 6209 12 - Live 0xd0ac5000
ip_conntrack 53153 3 ip_conntrack_ftp,ip_conntrack_netbios_ns,xt_state, Live
0xd0b6e000
nfnetlink 10713 1 ip_conntrack, Live 0xd0aca000
iptable_filter 7105 1 - Live 0xd0ab9000 //iptalbe 包过滤防火墙
ip_tables 17029 1 iptable_filter, Live 0xd0abf000
ip6t_REJECT 9409 1 - Live 0xd0ab5000
xt_tcpudp 7105 30 - Live 0xd0a9a000
ip6table_filter 6849 1 - Live 0xd0a45000 //ip6talbe 防火墙
ip6_tables 18181 1 ip6table_filter, Live 0xd0aaf000
x_tables 17349 6 ipt_REJECT,xt_state,ip_tables,ip6t_REJECT,xt_tcpudp,ip6_tables, Live
0xd0aa9000
video 19269 0 - Live 0xd0aa3000
sbs 18533 0 - Live 0xd0a9d000
i2c_ec 9025 1 sbs, Live 0xd0a30000
button 10705 0 - Live 0xd0a24000
battery 13637 0 - Live 0xd0a91000
asus_acpi 19289 0 - Live 0xd0a8b000
ac 9157 0 - Live 0xd0a28000
ipv6 251137 15 ip6t_REJECT, Live 0xd0ad5000
lp 15849 0 - Live 0xd093b000 //打印机 15849
sg 35933 0 - Live 0xd09ad000
floppy 54949 0 - Live 0xd0a36000 //软磁盘
```



```

snd_ens1371 28385 1 - Live 0xd09fe000
gameport 18889 1 snd_ens1371, Live 0xd09f8000
snd_rawmidi 26945 1 snd_ens1371, Live 0xd09f0000
snd_ac97_codec 87137 1 snd_ens1371, Live 0xd0a0d000 //ac97 声卡模块
snd_ac97_bus 6337 1 snd_ac97_codec, Live 0xd0994000
snd_seq_dummy 7877 0 - Live 0xd098b000
snd_seq_oss 32705 0 - Live 0xd09b8000
snd_seq_midi_event 11073 1 snd_seq_oss, Live 0xd0987000
snd_seq 49841 5 snd_seq_dummy,snd_seq_oss,snd_seq_midi_event, Live 0xd09e2000
snd_seq_device 11853 4 snd_rawmidi,snd_seq_dummy,snd_seq_oss,snd_seq, Live 0xd095d000
snd_pcm_oss 42849 0 - Live 0xd09d6000
snd_mixer_oss 19137 1 snd_pcm_oss, Live 0xd098e000
snd_pcm 72005 3 snd_ens1371,snd_ac97_codec,snd_pcm_oss, Live 0xd09c3000
snd_timer 25029 2 snd_seq,snd_pcm, Live 0xd09a5000
pcspkr 7105 0 - Live 0xd095a000
snd_ens1371,snd_rawmidi,snd_ac97_codec,snd_seq_oss,snd_seq,snd_seq_device,snd_pcm_oss,snd_mixer_oss,snd_pcm,snd_timer, Live 0xd0997000
i2c_piix4 12109 0 - Live 0xd0929000
soundcore 13217 1 snd, Live 0xd0955000
pcnet32 36805 0 - Live 0xd097d000
snd_page_alloc 13769 1 snd_pcm, Live 0xd087b000
i2c_core 23745 2 i2c_ec,i2c_piix4, Live 0xd0976000
mii 9409 1 pcnet32, Live 0xd0925000
parport_pc 29157 1 - Live 0xd096d000
parport 37641 2 lp,parport_pc, Live 0xd0962000
serial_core 23617 0 - Live 0xd0934000
ide_cd 40033 0 - Live 0xd094a000
cdrom 36705 1 ide_cd, Live 0xd0940000 //光盘驱动器
serio_raw 10693 0 - Live 0xd08e2000
dm_snapshot 20581 0 - Live 0xd092d000
dm_zero 6209 0 - Live 0xd0863000
dm_mirror 29840 0 - Live 0xd0826000
dm_mod 56537 8 dm_snapshot,dm_zero,dm_mirror, Live 0xd0906000
mptspi 20169 2 - Live 0xd0875000
mptscsih 26177 1 mptspi, Live 0xd085b000
mptbase 53089 2 mptspi,mptscsih, Live 0xd0917000
scsi_transport_spi 26177 1 mptspi, Live 0xd0853000
sd_mod 22977 3 - Live 0xd083e000
scsi_mod 130893 5 sg,mptspi,mptscsih,scsi_transport_spi,sd_mod, Live 0xd08c1000//SCSI 设备
ext3 123081 2 - Live 0xd08e6000 //ext3 设备
jbd 56553 1 ext3, Live 0xd0866000
ehci_hcd 33229 0 - Live 0xd0849000
ohci_hcd 23645 0 - Live 0xd0837000
uhci_hcd 25677 0 - Live 0xd082f000

```

3.1.5 cat /proc/version: 查看当前系统版本

文件/proc/version 保存了当前系统版本的相关信息, 配合使用 cat 命令可以查看当前系统版本情况。

例如, 查看当前系统的版本, 命令如下所示。

```

# cat /proc/version
Linux version 2.4.20-8(bhcompile@porky.devel.redhat.com) (gcc version 3.2.2 20030222 (Red Hat Linux 3.2.2-5)) #1 Thu Mar 13 17:54:28 EST 2003

```



3.1.6 cat /proc/partitions: 查看当前系统分区

文件/proc/partitions 保存了当前系统分区的相关信息，配合使用 cat 命令可以查看当前系统分区情况，命令如下所示。

```
# cat /proc/partitions
major minor #blocks name      rio rmerge rsect ruse wio wmerge wsect wuse running use aveq
      8     0   8388608 sda 73457 64861 1105512 227870 116402 157529 2192566 2003200 0
1557960 2231070          //SCSI 设备
      8     1    104391 sda1 38 85 246 40 17 10 54 290 0 250 330          // SCSI 设备分区 1
      8     2   7759395 sda2 73126 64733 1102866 227650 115461 154459 2160640 806890 0 376610
1034540          // SCSI 设备分区 2
      8     3    522112 sda3 290 38 2384 150 924 3060 31872 1196020 0 1187600 1196170
      22     0    496672 hdc 1 3 16 0 0 0 0 0 -14 26781161 11612690
```

3.1.7 cat /etc/issue 命令: 显示当前系统的发行版本

文件/etc/issue 中保存了当前系统的发行版本，配合使用 cat 命令可以显示输出版本信息，如下所示。

```
# cat /etc/issue
Red Hat Linux release 9 (Shrike)
Kernel \r on an \m
```

可以看到，当前的发行版本为 Red Hat Linux 9。

3.1.8 cat /proc/cpuinfo 命令: 显示 CPU 信息

文件/proc/cpuinfo 中保存了处理器的相关信息，通过 cat 命令查看/proc/cpuinfo，可以查询 CPU 的相关信息，命令如下所示。

```
#cat /proc/cpuinfo
processor       : 0
vendor_id      : GenuineIntel
cpu family     : 6
model          : 13
model name     : Intel(R) Pentium(R) M processor 2.26GHz
stepping      : 8
cpu MHz        : 798.069
cache size     : 0 KB
fdiv_bug      : no
hlt_bug       : no
f00f_bug      : no
coma_bug      : no
fpu           : yes
fpu_exception : yes
cpuid level    : 2
wp            : yes
flags          : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat clflush
dts acpi
mmx fxsr sse sse2 ss
bogomips      : 1582.69
```


3.1.9 cat /proc/interrupts 命令：显示中断信息

文件/proc/interrupts 中保存了系统中断的相关信息。显示系统中正在使用的中断号 (IRQ)，可以使用 cat 命令查看/proc/interrupts 文件，如下所示。

```
# cat /proc/interrupts
CPU0
0: 15604769 XT-PIC timer
1: 263 XT-PIC keyboard
2: 0 XT-PIC cascade
5: 0 XT-PIC ehci-hcd
8: 1 XT-PIC rtc
9: 0 XT-PIC usb-uhci
10: 67653 XT-PIC PCnet/PCI II 79C970A
11: 173832 XT-PIC BusLogic BT-958
12: 10975 XT-PIC PS/2 Mouse
15: 24 XT-PIC ide1
NMI: 0
ERR: 0
```

3.1.10 cat /proc/filesystems 命令：显示文件系统信息

文件/proc/filesystems 中保存了当前使用的文件系统信息，使用 cat 命令查看文件 /proc/filesystems，可以获得文件系统相关信息，如下所示。

```
# cat /proc/filesystems
nodev rootfs //未加载
nodev bdev
nodev proc
nodev sockfs
nodev tmpfs
nodev shm
nodev pipefs
ext2 //Linux 的 ext2 文件系统
nodev ramfs
iso9660 //光盘所使用的文件系统
nodev devpts
ext3 //Linux 的 ext3 文件系统
nodev usbdevfs
nodev usbfs
nodev autofs
```

3.2.11 date 命令：显示或设置系统时间

date 命令可以根据给定的格式显示当前的时间，也可以设置系统时间。该命令的语法格式如下所示。

```
date [-u, --utc,--universal] [-d, --date=STRING] [-s, --set=STRING] [-f, --file=
DATEFILE] [-R,
--rfc-822]
[-r, --reference=FILE] [--help] [--version] [MMDDhhmm[[CC]YY][.ss]] [+FORMAT]
```

其中各选项的说明如表 3-2 所示。



表 3-2 date 命令的选项说明

选 项	说 明
-d, --date=STRING	显示字符串所代表的时间
-s, --set=STRING	按字符串代表的时间进行设置
-f, --file=DATEFILE	从指定的文件中读取要显示的时间
-R, --rfc-822	按 RFC 822 兼容的时间格式输出
-r, --reference=FILE	显示上次文件的修改时间
-u, --utc, --universal	按 UTC 时间进行显示
--help	显示帮助信息
--version	显示版本信息
MM	月份
DD	日期
hh	小时
mm	分钟
CC	年的前两位
YY	年的后两位
ss	秒
+FORMAT	按指定的控制格式进行输出，控制格式见表 3-3

表 3-3 date 命令的输出格式设置

选 项	说 明
%H	小时（00~23）
%I	小时（01~12）
%K	小时（0~23）
%l	小时（0~12）
%M	分钟（00~59）
%P	以本地语言显示 AM 或 PM
%p	以本地语言显示 am 或 pm
%r	包含时分秒的时间，12h 进制（hh:mm:ss [AP]M）
%s	UTC 时间，总秒数，自“00:00:00 1970-01-01 UTC”开始计算
%V	年中的周数，按星期一为一周的开始计算（01..53）
%v	年中的天数，按星期日为一周的开始计算（00..53）
%S	秒数（00..60）
%T	包含时分秒的时间，按 24h 进制（hh:mm:ss）
%X	以本地语言显示时间，按（%H:%M:%S）格式
%Z	时区
%a	以本地语言显示星期，使用缩写（如 Sun..Sat）

续表

选 项	说 明
%A	以本地语言显示星期，使用完整的写法（如 Sunday..Saturday）
%b	以本地语言显示月份，缩写（如 Jan..Dec）
%B	以本地语言显示月份的完整写法（如 January..December）
%c	以本地语言显示包含年月日和时分秒的时间（如“2008 年 05 月 29 日 星期四 21 时 52 分 53 秒”）
%C	世纪（00~99）
%R	包含时分秒的时间，24h 进制（hh:mm）
%d	月中的天数（01..31）
%D	包含年月日的日期（mm/dd/yy）
%e	月中的天数（1~31）
%F	与%Y-%m-%d 相同
%j	天数，从本年第一天开始计算（001..366）
%m	显示月份（01..12）
%U	周数，从年初开始算起
%w	天数，从本周周日开始算起，周日为 0
%x	以本地语言显示日期，按（mm/dd/yy）格式
%y	包含年的日期（00~99）
%Y	包含年的日期，4 位数表示
%n	输出时，插入回车输出一新行
%t	输出时，插入水平 Tab

例如，显示当前时间，使用下面命令。

```
#date //显示当前时间、日期
四 5月29 22:15:28 CST 2008
```

例如，按本地语言设置（中文）以完整格式显示当前的时间、日期，命令如下。

```
# date +%c //按完整格式显示当前时间、日期
2008 年 05 月 29 日 星期四 22 时 15 分 45 秒
```

例如，以“mm/dd/yy”格式显示当前的年月日信息，命令如下所示。

```
# date +%D
05/29/08
```

例如，以“YYYY-MM-DD”格式显示当前的日期，命令如下所示。

```
# date +%F
2008-05-29
```

例如，以本地语言显示当前的时间，命令如下所示。

```
# date +%X
22 时 33 分 23 秒
```

例如，按 24h 进制显示包含时分秒的时间（hh:mm:ss 格式），命令如下所示。

```
# date +%T
22:33:34
```



例如，在时间输出中加入自己的格式，命令如下所示。

```
# date "+Today is %A ,Tomorrow is weekend."
Today is 星期四 ,Tomorrow is weekend.
```

例如，配合使用选项“-d”显示明天的时间，命令如下所示。

```
# date -d tomorrow
五 5月 30 20:56:17 CST 2008
```

例如，使用选项同“-d”显示昨天的系统时间，命令如下所示。

```
# date -d yesterday
三 5月 28 22:42:40 CST 2008
```

例如，使用选项“-d”显示当天的系统时间，命令如下所示。

```
# date -d today
四 5月 29 20:56:32 CST 2008
```

3.1.12 dmesg: 显示启动信息

dmesg 命令可以检测和控制内核的缓存，通常用来显示输出系统最后一次启动时的内核内部缓存信息。dmesg 命令的语法格式如下所示。

```
dmesg [-c] [-n level] [-s bufsize]
```

其中各选项的具体说明如表 3-4 所示。

表 3-4 dmesg 命令选项说明

选 项	说 明
-c	显示信息后清空缓冲区中的信息
-s bufsize	设置查询所需的缓冲区大小，默认是 16 392 字节
-n level	设置将日志信息进行输出的级别。例如，设置为“-n 1”将只显示输出 panic（恐慌）级别的信息

例如，直接使用 dmesg 信息显示上次系统启动时的信息，如下所示。

```
# dmesg
Linux version 2.4.20-8 (bhcompile@porky.devel.redhat.com) (gcc version 3.2.2 20030222
(Red Hat Linux 3.2.2-5)) #1 Thu Mar 13 17:54:28 EST 2003
BIOS-provided physical RAM map: //BIOS 物理内存地址映射
 BIOS-e820: 0000000000000000 - 000000000009f800 (usable)
 BIOS-e820: 000000000009f800 - 00000000000a0000 (reserved)
 BIOS-e820: 00000000000ca000 - 00000000000cc000 (reserved)
 BIOS-e820: 00000000000dc000 - 0000000000100000 (reserved)
 BIOS-e820: 0000000000100000 - 0000000000fef0000 (usable)
 BIOS-e820: 0000000000fef0000 - 0000000000feff000 (ACPI data)
 BIOS-e820: 0000000000feff000 - 0000000000ff00000 (ACPI NVS)
 BIOS-e820: 0000000000ff00000 - 00000000010000000 (usable)
 BIOS-e820: 0000000000fec00000 - 0000000000fec10000 (reserved)
 BIOS-e820: 0000000000fee00000 - 0000000000fee01000 (reserved)
 BIOS-e820: 0000000000fffe0000 - 000000000100000000 (reserved)
0MB HIGHMEM available. //可用高端内存为 0
256MB LOWMEM available. //256MB 低端内存
On node 0 totalpages: 65536
zone(0): 4096 pages.
zone(1): 61440 pages.
zone(2): 0 pages.
Kernel command line: ro root=LABEL=/
```



```
Initializing CPU#0
Detected 2259.329 MHz processor.
Console: colour VGA+ 80x25      //显示器为彩色 VGA+ 80x25
Calibrating delay loop... 4521.98 BogoMIPS
... ..
```

3.1.13 echo 命令：显示文本行

echo 命令可以将参数输出到标准输出设备上，多个参数之间默认用空格分隔，在输出完成后自动添加换行符。echo 命令通常用于显示输出文本行，该命令的语法格式如下所示。

```
echo [-n][-e][-E][--help][--version][string]
```

其中各选项的语法格式如表 3-5 所示。

表 3-5 echo 命令选项说明

选 项	说 明
-n	不输出最后的换行符 “\n”
-e	对转义字符进行解析
-E	不解析转义字符
string	指定显示的字符串
--help	显示输出帮助信息
--version	显示输出版本信息



不同的 Shell 所支持的 echo 命令略有不同。

echo 命令可以通过使用选项 “-e” 或 “-E” 设定是否对转义字符进行解析，默认不进行解析。转义字符是一些以 “\” 开头的字符组合，当开启选项 “-e” 后，echo 命令会将转义字符解析为控制字符，而不会将其当作普通字符进行输出。表 3-6 列出了一些常用的转义字符。

表 3-6 转义字符及含义说明

选 项	说 明
\a	响铃警告
\b	删除前一个字符
\c	相当于 “-n” 选项，不输出末尾的换行符 “\n”
\f	换页
\n	换行
\r	回车，即光标移到行首
\t	插入水平 Tab
\v	插入垂直 Tab



续表

选 项	说 明
\\	插入反斜杠
\'	插入单引号
\"	插入双引号
\nnn (八进制)	显示 nnn (八进制 ASCII) 表示的字符

例如，输出指定的字符串，命令如下所示。

```
# echo "I am a student and I like my school" //输出指定字符
I am a student and I like my school
```

例如，当输出的字符串中含有转义字符时，使用选项“-e”对转义字符进行解析，命令如下所示。

```
# echo -e "hello\t\t world"
hello          world
```

从输出结果可以看出，“\t”被转义为水平 Tab，因此在“hello”与“world”之间插入了两个 Tab。如果不希望对转义字符进行转义，可以使用选项“-E”，命令如下所示。

```
# echo -E "hello\t\t world"
hello\t\t world
```

可以看出，“\t”被当成普通字符进行输出。echo 命令默认不对转义字符进行解析，因此直接使用 echo 命令可以取得与上例相同的效果，如下所示。

```
# echo "hello\t\t world"
hello\t\t world
```

3.1.14 free 命令：显示内存信息

free 命令用于显示系统使用和空闲的内存情况，涉及的内存包括物理内存、交换分区内存 (swap)、内核缓冲区内存以及高速缓存。free 命令的语法格式如下所示。

```
free [-b][-k][-m][-g] [-o] [-t] [-s delay] [-c count] [-V]
```

其中各选项的具体说明如表 3-7 所示。

表 3-7 命令 free 的选项说明

选 项	说 明
-b	显示内存单位为字节
-k	显示内存的单位为 KB
-m	显示内存的单位为 MB
-g	显示内存的单位为 GB
-o	按旧格式进行显示，即不显示 “-/+buffers/cache” 行
-t	显示 RAM 及 swap 的总和
-s delay	每间隔指定时间刷新一次命令，单位为 s
-c count	设定刷新的次数
-V	显示输出版本信息并退出该命令

例如，直接使用 free 命令显示内存的使用情况，如下所示。

```
#free //显示内存使用信息
total      used      free      shared  buffers  cached
Mem:      255264    244192    11072         0     65576    75144
-/+ buffers/cache:  103472    151792
Swap:      522104    17752     504352
```

例如，以总和的形式显示内存使用信息，命令如下所示。

```
# free -t //以总和的形式显示内存使用信息
total      used      free      shared  buffers  cached
Mem:      255264    244196    11068         0     65576    75144
-/+ buffers/cache:  103476    151788
Swap:      522104    17752     504352
Total:     777368    261948    515420
```

例如，每隔 3s，周期性地查询内存使用信息，命令如下所示。

```
# free -s 3 //每 5 秒钟执行一次 free 命令
total      used      free      shared  buffers  cached
Mem:      255264    244272    10992         0     65648    75148
-/+ buffers/cache:  103476    151788
Swap:      522104    17752     504352

total      used      free      shared  buffers  cached
Mem:      255264    244280    10984         0     65652    75148
-/+ buffers/cache:  103480    151784
Swap:      522104    17752     504352
... ..
```

例如，每隔 2s 刷新一次内存信息，共刷新 3 次，命令如下所示。

```
# free -s 2 -c 3
total      used      free      shared  buffers  cached
Mem:      255264    244288    10976         0     65660    75152
-/+ buffers/cache:  103476    151788
Swap:      522104    17752     504352

total      used      free      shared  buffers  cached
Mem:      255264    244292    10972         0     65660    75152
-/+ buffers/cache:  103480    151784
Swap:      522104    17752     504352

total      used      free      shared  buffers  cached
Mem:      255264    244292    10972         0     65660    75152
-/+ buffers/cache:  103480    151784
Swap:      522104    17752     504352
#
```

例如，以旧格式显示内存信息，命令如下所示。

```
# free -o
total      used      free      shared  buffers  cached
Mem:      255264    243964    11300         0     65660    74828
Swap:      522104    17752     504352
```

可以看到，所显示的内容中不包含缓存信息。

3.1.15 fgconsole 命令：显示活动的虚拟终端数目

fgconsole 命令用来显示活动的虚拟终端（vt）的数目，如下所示。



```
# fgconsole //显示 vt 数目
7
```

3.1.16 fuser 命令：用文件或套接口识别进程

fuser 命令可以利用指定的文件或套接口对进程进行识别，该命令的语法格式如下所示。
fuser [-] [-a] [-s] [-n space] [-signal] [-k][-i][-m][u][v] [-V][-n space][name ...]
其中各选项说明如表 3-8 所示。默认情况下，每一个文件名后会跟着一个字母来表示类型，具体含义如下。

表 3-8 fuser 命令的选项说明

选 项	说 明
-l	显示信号列表
-4	IPv4
-6	IPv6
-a	显示与指定文件或目录相关的进程，包括未正在使用的文件
-k	杀死访问文件的进程
-i	交互模式，在杀死进程之前进行询问
-s	静默模式，不显示处理信息
-u	显示用户 ID
-v	运行时，显示详细的处理信息
-V	显示版本信息
-	清零
-m	挂载文件系统
-n space	搜索指定的名字空间（包括文件、UDP 或 TCP）
-signal	发送信号

- c: 当前目录。
- e: 扩展。
- f: 打开的文件。
- r: 根目录。
- m: 共享库。

例如，显示信号列表，命令如下所示。

```
# fuser -l //显示信号
HUP INT QUIT ILL TRAP ABRT IOT BUS FPE KILL USR1 SEGV USR2 PIPE ALRM TERM
STKFLT CHLD CONT STOP TSTP TTIN TTOU URG XCPU XFSZ VTALRM PROF WINCH IO PWR SYS
UNUSED
```

例如，显示与 /root 目录相关的进程（包括正在使用和未被使用的文件），命令如下所示。

```
# fuser -a /root //显示与 /root 目录相关的进程
/root: 2582c 2604c 2648c 2650c 2652c 2666c 2668c 2670c 2672c 2675c
2679c 2688c 2967c 2968 3003c 3004c 3005c
```


例如，杀死所有与/root 目录相关的进程，命令如下所示。

```
# # fuser -k /root //杀死所有与/root 目录相关的进程
/root:          9050c 9086c 9116c 9118c 9121c 9127 9134c 9136c 9138c  9140c  9142c
9144c 9151c 9157c 9158c 9159c 9194c 9198c
```

3.1.17 gdialog 命令：从 Shell 创建对话框

gdialog 命令用来从 Shell 脚本创建对话框。gdialog 命令目前支持的对话框包括：yes/no 对话框、menu 对话框、input 对话框、message 对话框、text 对话框、info 对话框、checklist 对话框、radiolist 对话框和 gauge 对话框。gdialog 命令的语法格式如下所示。

```
gdialog [--create-rc <file>][--title <title>] [--backtitle <backtitle>] [Box options]
其中各选项的具体说明如表 3-9 所示。
```

表 3-9 gdialog 命令的选项说明

选 项	说 明
--title	设置对话框的标题
--backtitle <backtitle>	设置背景标题
--create-rc <file>	将对话框的基本配置信息输出到指定的文件
Box options	对话框选项说明，见表 3-10

表 3-10 gdialog 命令的 Box option 选项说明

选 项	说 明
--yesno <text><height><width>	创建一个 yes/no 对话框，对话框高<height>（0~40），宽<width>（0~80）。<text>为对话框中显示的文本内容。当文本长度超出单行的最大长度，文本自动换行，文本中也可以使用“\n”控制换行。该对话框需要用户选择【Yes】或【No】按钮，按钮之间使用 Tab 键或 RIGHT/LEFT 键切换
--msgbox<text><height><width>	创建一个 msgbox 对话框，对话框高<height>（0~40），宽<width>（0~80）。<text>为对话框中显示的文本内容，当文本长度超出单行的最大长度，文本自动换行，文本中也可以使用“\n”控制换行。该对话框只有【OK】按钮，用户只需按回车键即可关闭该对话框
--infobox<text><height><width>	创建一个 info 对话框，对话框高<height>（0~40），宽<width>（0~80）。info 对话框具有 message 对话框的基本特征。<text>为对话框中显示的文本内容，当文本长度超出单行的最大长度，文本自动换行，文本中也可以使用“\n”控制换行。与 message 对话框不同的是，该对话框在显示信息后很快会自动退出
--inputbox<text><height><width>	创建一个 inputbox 对话框，对话框高<height>（0~40），宽<width>（0~80）。<text>为对话框中显示的文本内容，当文本长度超出单行的最大长度，文本自动换行，文本中也可以使用“\n”控制换行。该对话框允许用户输入文本，当输入的文本超出单行最大长度，文本自动换行。可以使用“BACKSPACE”键对输入的错误进行删除。退出对话框时，所输入的文本将从标准错误设备输出
--textbox<file><height><width>	创建一个 textbox 对话框，该对话框允许将一个文本文件的内容显示在对话框中，类似于一个简单的文本浏览器。用户可以使用“UP/DOWN”、“PGUP/PGDN”以及“HOME/END”键对文本进行翻阅。当一行的文本超出了单行最大长度，可以使用“LEFT/RIGHT”键进行滚动
--menu <text> <height> <width> <menu-height> [<tag> <item> ...]	创建一个菜单对话框，对话框高<height>（0~40），宽<width>（0~80）。<text>为对话框中显示的文本内容，当文本长度超出单行的最大长度，文本自动换行，文本中也可以使用“\n”控制换行。<menu-height>用于设置菜单的高度，当菜单项超出菜单高度，可以使用“UP/DOWN”上下滚动。<tag>为菜单项的名称，<item>为对应菜单项的说明。当退出对话框时，所选择的<tag>将被输出到标准错误输出设备



续表

选 项	说 明
<code>--checklist<text><height><width></code> <code><list-height></code> <code>[<tag><item><status>...]</code>	创建一个多选对话框，类似于 <code>menu</code> ，但可以选择多个选项。对话框高<height> (0~40)，宽<width> (0~80)。<text>为对话框中显示的文本内容，当文本长度超出单行的最大长度，文本自动换行，文本中也可以使用“\n”控制换行。<list-height>用于设置列表的高度，当列表项超出列表高度时，可以使用“UP/DOWN”上下滚动。<tag>为选项的名称，<item>为对应选项的说明，<status>为选项的默认状态，取值 on/off。当退出对话框时，所选择的<tag>将被输出到标准错误输出设备
<code>--radiolist<text><height><width></code> <code><list-height></code> <code>[<tag><item><status>...]</code>	创建一个单选对话框，类似于 <code>checklist</code> ，但只能选择一个选项。对话框高<height> (0~40)，宽<width> (0~80)。<text>为对话框中显示的文本内容，当文本长度超出单行的最大长度，文本自动换行，文本中也可以使用“\n”控制换行。<list-height>用于设置列表的高度，当列表项超出列表高度时，可以使用“UP/DOWN”上下滚动。<tag>为选项的名称，<item>为对应选项的说明，<status>为选项的默认状态，取值 on/off。当退出对话框时，所选择的<tag>将被输出到标准错误输出设备

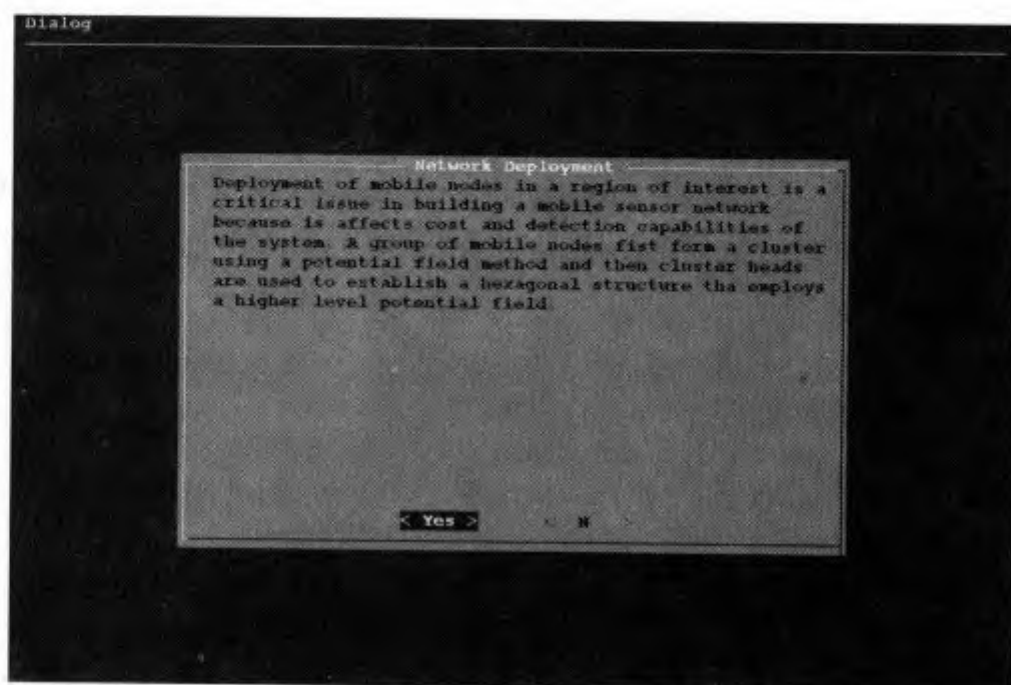
例如，使用选项“`create-rc`”将对话框的配置信息保存到文件 `config` 中，命令如下所示。

```
#gdialog --create-rc config
# cat config                                     //显示 config 文件内容
#
# Run-time configuration file for dialog
#
# Automatically generated by "dialog --create-rc <file>"
#
#
# Types of values:                               //数值类型
#
# Number      - <number>
# String      - "string"
# Boolean      - <ON|OFF>
# Attribute   - (foreground,background,highlight?)
# Shadow dialog boxes? This also turns on color.
use_shadow = OFF                                //是否使用阴影
# Turn color support ON or OFF
use_colors = ON
# Screen color
screen_color = (WHITE,BLUE,OFF)
```

例如，创建一个 20×60 的 `yesno` 对话框，对话框标题为“`Network Deployment`”，背景标题为“`Dialog`”，对话框内显示文本为“`Deployment of mobile nodes in a region of interest is a critical issue in building a mobile sensor network because is affects cost and detection capabilities of the system. A group of mobile nodes fist form a cluster using a potential field method and then cluster heads are used to establish a hexagonal structure tha employs a higher level potential field.`”，命令如下所示。

```
# gdialog --title "Network Deployment" --backtitle Dialog --yesno "Deployment of
mobile nodes in a region of interest is a critical issue in building a mobile sensor network
because is affects cost and detection capabilities of the system. A group of mobile nodes
fist form a cluster using a potential field method and then cluster heads are used to establish
a hexagonal structure tha employs a higher level potential field." 20 60
```

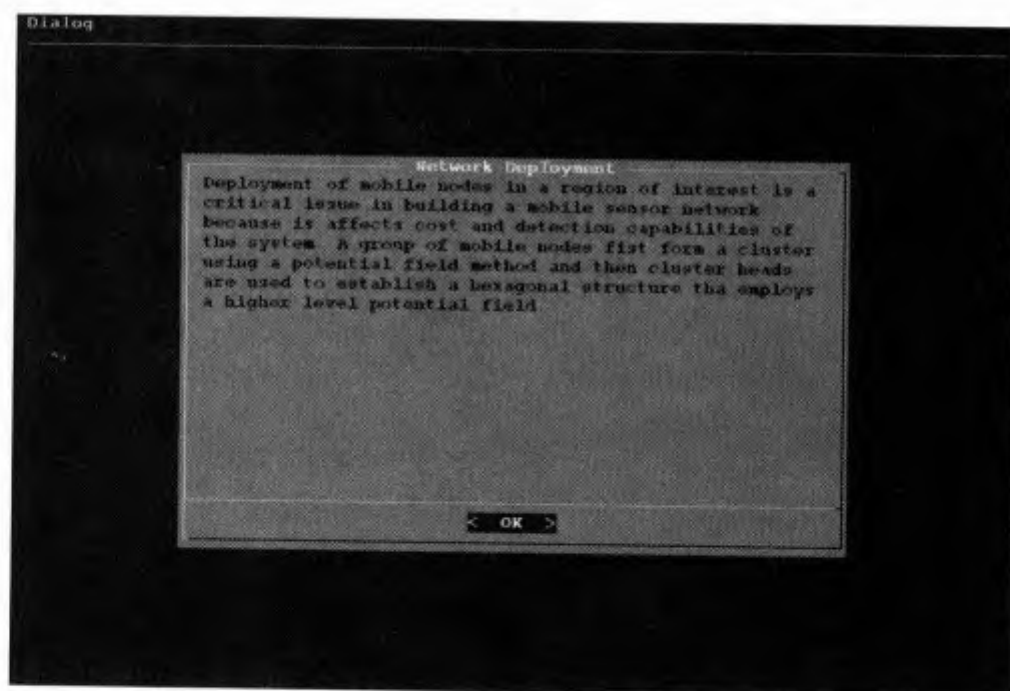
执行上述命令后，将出现如图 3-1 所示的对话框界面。该对话框需要用户选择【`Yes`】或【`No`】按钮，按钮之间切换使用 `Tab` 键或 `LEFT/RIGHT` 方向键。

图 3-1 使用 `gdialog` 命令创建 `yes/no` 对话框

例如，创建一个 20×60 的 `msgbox` 对话框，对话框标题为“Network Deployment”，背景标题为“Dialog”，对话框内显示文本为“Deployment of mobile nodes in a region of interest is a critical issue in building a mobile sensor network because is affects cost and detection capabilities of the system. A group of mobile nodes fist form a cluster using a potential field method and then cluster heads are used to establish a hexagonal structure tha employs a higher level potential field.”，命令如下所示。

```
# gdialog --title "Network Deployment" --backtitle Dialog --msgbox "Deployment of
mobile nodes in a region of interest is a critical issue in building a mobile sensor network
because is affects cost and detection capabilities of the system. A group of mobile nodes
fist form a cluster using a potential field method and then cluster heads are used to establish
a hexagonal structure tha employs a higher level potential field." 20 60
```

执行上述命令后，将出现图 3-2 所示的 `msgbox` 对话框，该对话框只有【OK】按钮，用户只需按回车键即可关闭该对话框。

图 3-2 使用 `gdialog` 命令创建 `msgbox` 对话框



例如，创建一个 20×60 的 inputbox 对话框，对话框标题为“Network Deployment”，背景标题为“Dialog”，对话框内显示文本为“Deployment of mobile nodes in a region of interest is a critical issue in building a mobile sensor network because is affects cost and detection capabilities of the system. A group of mobile nodes fist form a cluster using a potential field method and then cluster heads are used to establish a hexagonal structure tha employs a higher level potential field.”，命令如下所示。

```
# gdialog --title "Network Deployment" --backtitle Dialog --inputbox "Deployment of mobile nodes in a region of interest is a critical issue in building a mobile sensor network because is affects cost and detection capabilities of the system. A group of mobile nodes fist form a cluster using a potential field method and then cluster heads are used to establish a hexagonal structure tha employs a higher level potential field." 20 60
```

执行上述命令后，将出现图 3-3 所示的 inputbox 对话框，该对话框允许用户输入文本。若输入的文本超出单行最大长度，文本将自动换行，可以使用“BACKSPACE”键（回退键）对输入的错误进行删除。在输出框中输入“the work is very perfect!!!”后，选择【OK】按钮并按回车后，退出对话框。退出时，所输入的文本将从标准错误设备输出。

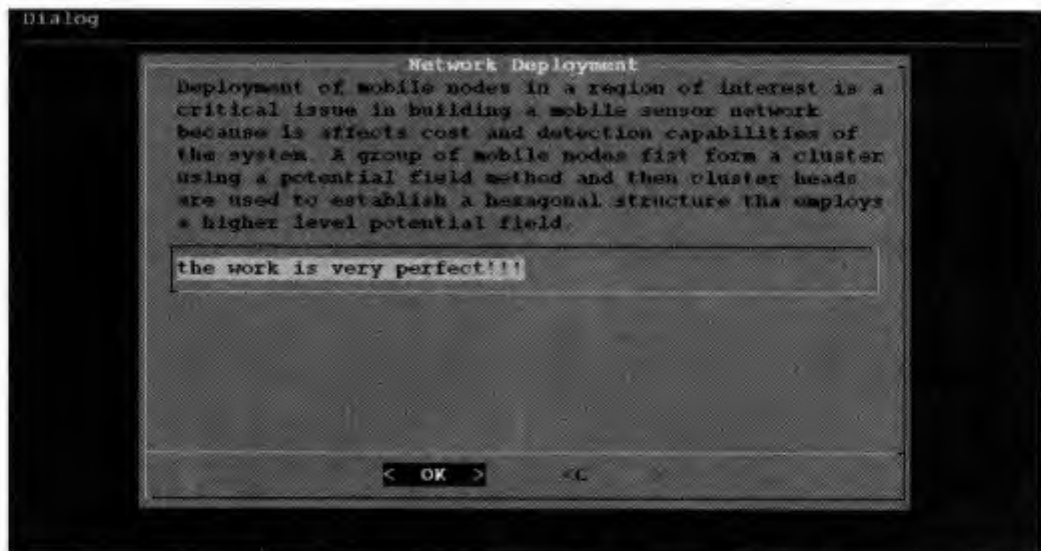


图 3-3 使用 gdialog 命令创建 inputbox 对话框

例如，创建一个 20×60 的 textbox 对话框，对话框标题为“Network Deployment”，背景标题为“Dialog”，对话框内显示来自文件 textbox.txt 的文本，命令如下所示。

```
# cat textbox.txt
Deployment of mobile nodes in a region of interest is
a critical issue in building a mobile sensor network
because is affects cost and detection capabilities of
the system. A group of mobile nodes fist form a cluster
using a potential field method and then cluster heads
are used to establish a hexagonal structure tha employs
a higher level potential field.
# gdialog --title "Network Deployment" --backtitle Dialog --
textbox textbox.txt 20 60
```

执行该命令后，将出现图 3-4 所示的 textbox 对话框，该对话框类似于一个简单的文本阅读器。用户可以使用“UP/DOWN”、“PGUP/PGDN”及“HOME/END”键对文本进行翻阅。当一行的文本超出了单行最大长度，可以使用“LEFT/RIGHT”键进行滚动。按回车键即可退出对话框。

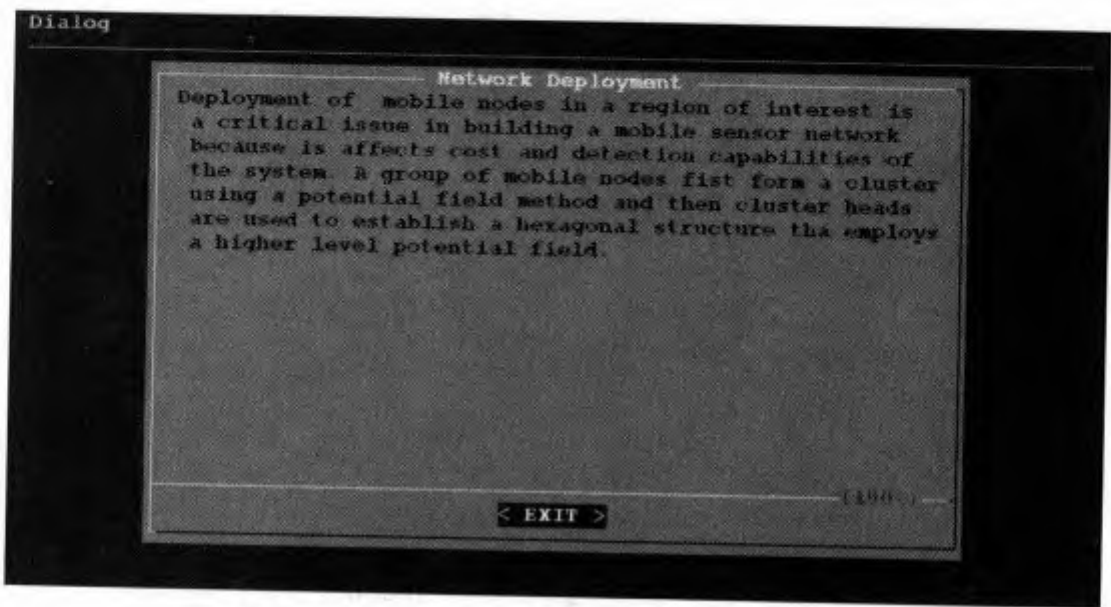


图 3-4 使用 gdialog 命令创建 textbox 对话框

例如，创建一个 20×60 的 menu 对话框，对话框标题为 “Name VS. ID”，背景标题为 “Dialog”，对话框内显示的菜单为 “姓名—学号” 列表，菜单高为 10 行，“姓名—学号” 列表具体内容如表 3-11 所示。

表 3-11 memu 对话框的 “姓名—学号” 列表

姓 名	学 号
John	10000345
Tom	10000245
Zhangsan	10000768
sullivan	10002353
kelly	10000875
sum	20000342
Jaeyone	20004523
Liliang	10000328
mark	10023458
shanks	1000043
kitty	20003213
Jony	10000875

执行如下命令。

```
# gdialog --title "Name VS. ID" --backtitle Dialog --menu "This is a DEMO of menu box"
20 60 10 John 10000345 Tom 10000245 Zhangsan 10000768 sullivan 10002353 kelly
10000875 sum 20000342 Jaeyone 20004523 Liliang 10000328 mark 10023458 shanks
20000432 kitty 20003213 Jony 10000875
```

执行该命令后，将出现图 3-5 所示的 menu 对话框。由于所输入的 “姓名—学号” 列表超出了菜单的高度，可以使用 “UP/DOWN” 键上下滚动菜单选项。选择【OK】按钮并按回车后，所选择的菜单项将被输出到标准错误输出设备。

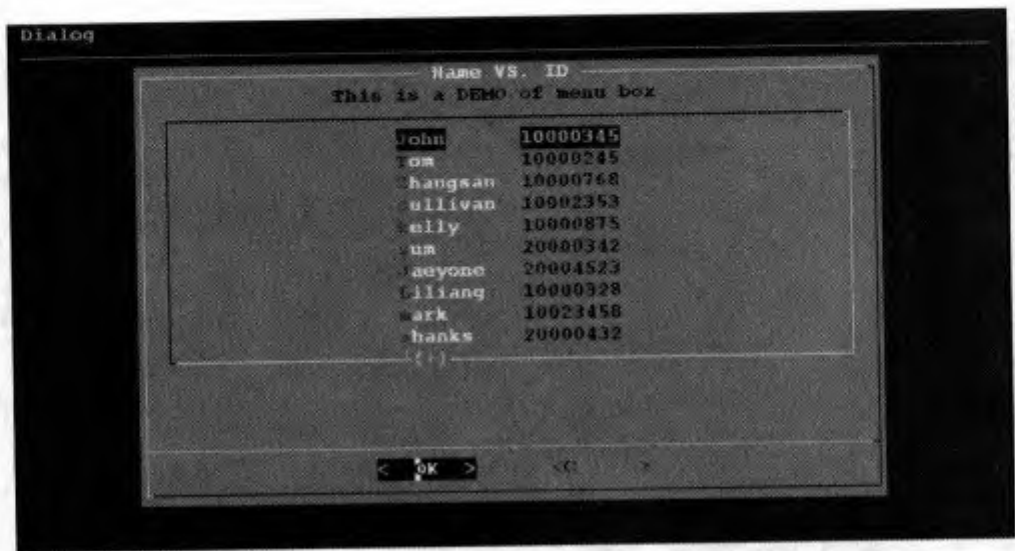


图 3-5 使用 gdialog 命令创建 menu 对话框

例如，创建一个 20×60 的 checklist 对话框，对话框标题为“Name VS. ID”，背景标题为“Dialog”，对话框内显示的多选列表为“姓名—学号”列表，多选列表高为 10 行，“姓名—学号”列表具体内容及默认设置如表 3-12 所示。

表 3-12 checklist 对话框的“姓名—学号”列表

姓 名	学 号	默 认
John	10000345	未选
Tom	10000245	未选
Zhangsan	10000768	未选
sullivan	10002353	已选
kelly	10000875	未选
sum	20000342	已选
Jaeyone	20004523	未选
Liliang	10000328	未选
mark	10023458	未选
shanks	1000043	未选
kitty	20003213	未选
Jony	10000875	未选

执行如下所示命令。

```
# gdialog --title "Name VS. ID" --backtitle Dialog --checklist "This is a DEMO of checklist box" 20 60 10 John 10000345 off Tom 10000245 off Zhangsan 10000768 off sullivan 10002353 on kelly 10000875 off sum 20000342 on Jaeyone 20004523 off Liliang 10000328 off mark 10023458 off shanks 20000432 off kitty 20003213 off Jony 10000875 off
```

执行该命令后，将出现图 3-6 所示的 checklist 对话框，由于所输入的“姓名—学号”列表超出了菜单的高度，可以使用“UP/DOWN”键上下滚动菜单选项。可以看到其中“sullivan”及“sum”默认已经是选择状态，按空格键可以选择或取消选择。选择【OK】按钮并按回车后，所选择的菜单项将被输出到标准错误输出设备。

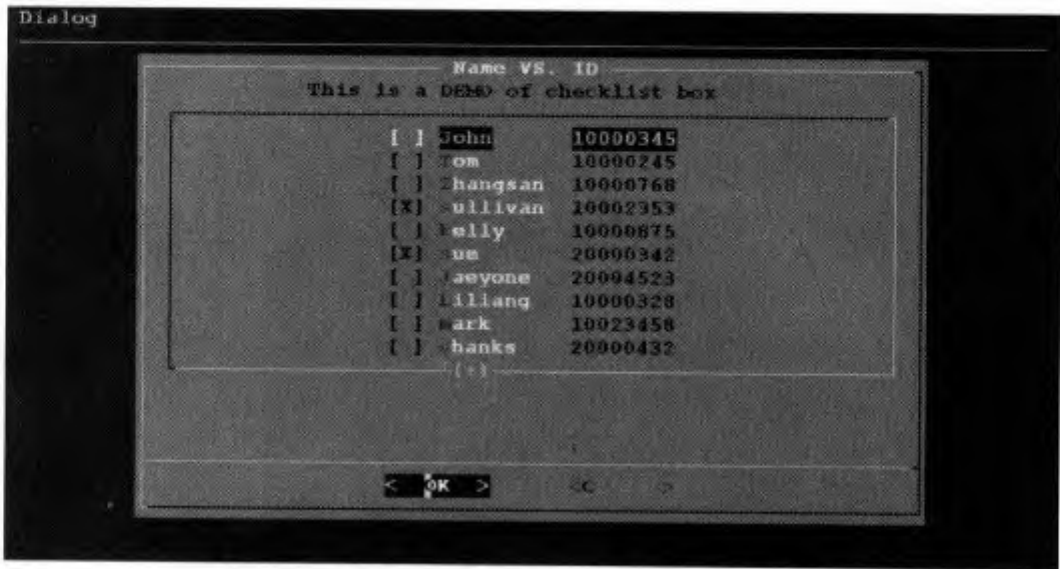


图 3-6 使用 gdialog 命令创建 checklist 对话框

例如，创建一个 20×60 的 radiolist 对话框，对话框标题为“Name VS. ID”，背景标题为“Dialog”，对话框内显示的列表为“姓名—学号”列表，列表高为 10 行，“姓名—学号”列表具体内容及默认设置如表 3-13 所示。

表 3-13 radiolist 对话框的“姓名—学号”列表

姓 名	学 号	默 认
John	10000345	已选
Tom	10000245	未选
Zhangsan	10000768	未选
sullivan	10002353	未选
kelly	10000875	未选
sum	20000342	未选
Jaeyone	20004523	未选
Liliang	10000328	未选
mark	10023458	未选
Shanks	1000043	未选
kitty	20003213	未选
Jony	10000875	未选

执行如下所示命令。

```
# gdialog --title "Name VS. ID" --backtitle Dialog --radiolist "This is a DEMO of radiolist box" 20 60 10 John 10000345 on Tom 10000245 off Zhangsan 10000768 off sullivan 10002353 off kelly 10000875 off sum 20000342 off Jaeyone 20004523 off Liliang 10000328 off mark 10023458 off shanks 20000432 off kitty 20003213 off Jony 10000875 off
```

执行该命令后，将出现图 3-7 所示的 radiolist 对话框，由于所输入的“姓名—学号”列表超出了菜单的高度，可以使用“UP/DOWN”键上下滚动菜单选项。可以看到其中“John”默认已经是选择状态，按空格键可以选择或取消选择。选择【OK】按钮并按回车后，所选择的菜单项将被输出到标准错误输出设备。

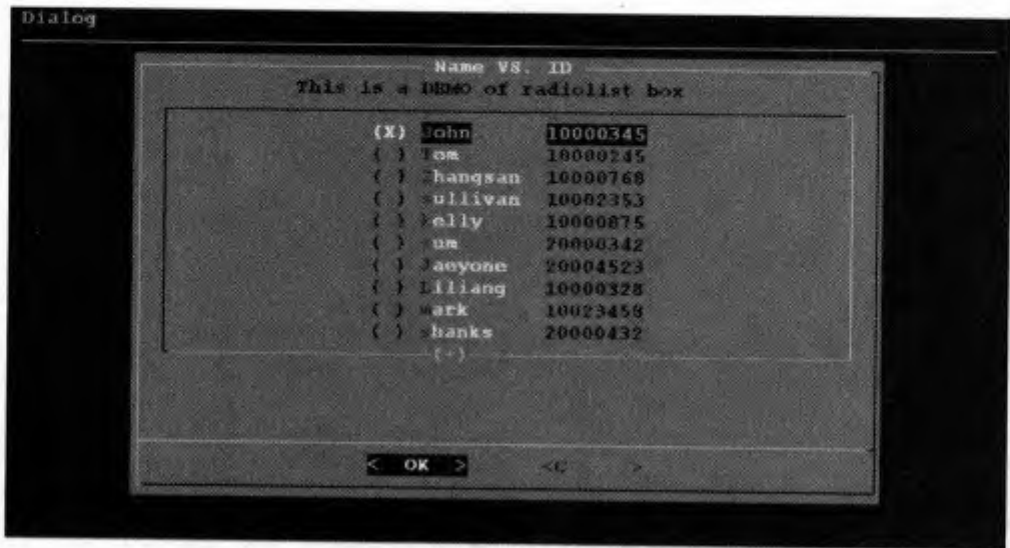


图 3-7 使用 gdialog 命令创建 radiolist 对话框

3.1.18 history 命令：显示历史指令

history 命令保留了最近执行的命令记录，默认可以保留 500 个。历史清单从 0 开始编号，直到保留的最大值。history 命令的语法格式如下所示。

```
history [-c] [-d offset] [n] [-a][w][r] [-n filename]
```

其中各选项说明如图 3-14 所示。

表 3-14 history 命令的选项说明

选 项	说 明
N	显示历史记录中最近的 N 个记录
-a	将当前命令添加到历史记录中
-r	读取历史记录，但不会添加到历史记录中
-w	覆盖原有的 history 文件
-c	清除所有的历史记录
-d<offset>	删除历史记录中第 offset 个命令
-n<filename>	读取指定文件

快速重新执行已经执行过的命令可以使用“! <命令事件号>”，如下所示。

```
#history
... ..
281 cd student1
282 ls
283 ls -al
284 cd ..
285 history
# !282                //重复执行第 282 号命令
ls
anaconda-ks.cfg chage_li Desktop install.log install.log.syslog man_chage
```

如果要检查某用户在系统上都运行了什么命令，可以以 root 用户身份登录系统，检查该用户主目录下的“.bash_history”文件。该文件中记录了用户所使用的命令和历史信息，例如下列命令所示。


```
#cd /home/teacher1
# cat .bash_history
rm /etc/test
mkdir homework
ls -al
more
... ..
```

例如，显示最近执行的 5 个命令，命令如下所示。

```
# history 5
1120 pstree root
1121 pstree -h
1122 history --help
1123 man history
1124 history 5
```

例如，使用选项 “-c” 消空历史记录，命令如下所示。

```
# history -c
# history
126 history
```

3.1.19 hostname 命令：显示或设置当前系统的主机名

hostname 命令用于显示或设置系统的主机名称。hostname 命令的语法格式如下所示。

```
hostname [-v][-s, --short] [-a, --alias] [-i, --ip-address] [-f, --fqdn, --long] [-d]
[-y, --yp, --nis] [-F, --file][-V, --version] [-h, --help]
```

其中各选项的具体说明如表 3-15 所示。

表 3-15 hostname 命令选项说明

选 项	说 明
-s, --short	短主机名
-a, --alias	别名
-i, --ip-address	IP 地址
-f, --fqdn, --long	长主机名
-d, --domain	DNS 域名
-y, --yp, --nis	NIS/YP 域名
-v	运行时，显示详细的处理过程
-F, --filename	读取指定文件
-h, --help	帮助信息
-V, --version	版本信息

例如，直接使用 hostname 命令将显示主机名称（包含域名），命令如下所示。

```
# hostname //显示主机名
localhost.localdomain
```

例如，使用选项 “-s” 以简短格式显示主机名，命令如下所示。

```
# hostname -s //显示短主机名
localhost
```

例如，使用选项 “-a” 显示主机别名，命令如下所示。

```
# hostname -a //显示主机别名
localhost
```



例如，使用选项 “-i” 显示主机 IP 地址，命令如下所示。

```
# hostname -i                //显示主机 IP 地址
127.0.0.1
```

例如，使用选项 “-d” 显示主机域名，命令如下所示。

```
# hostname -d                //显示主机域名
localdomain
```

例如，使用选项 “--yp” 显示 NIS/YP 域名，命令如下所示。

```
# hostname --yp              //显示 NIS/YP 域名
(none)
```

例如，设置主机名为 teacher，命令如下所示。

```
# hostname teacher          //设置主机名称
# hostname
Teacher
```

3.1.20 hash 命令：显示、添加或清除哈希表

hash 命令用来显示、添加和清除哈希表。该命令的语法格式如下所示。

```
hash [-l][-r] [-p <path> <name>] [-d <name>] [-t<command>]
```

其中各选项的具体说明如表 3-16 所示。

表 3-16 hash 命令的选项说明

选 项	说 明
-l	显示哈希表，包括路径
-r	清除哈希表
-p<path><name>	向哈希表中增加内容
-t<command>	显示指定命令的完整路径

例如显示哈希表，命令如下所示。

```
# hash -l                    //显示哈希表
builtin hash -p /usr/bin/tload tload
builtin hash -p /usr/bin/procinfo procinfo
builtin hash -p /usr/bin/ipcs ipcs
builtin hash -p /usr/bin/vim vim
builtin hash -p /bin/cat cat
builtin hash -p /usr/bin/pmap pmap
builtin hash -p /bin/ps ps
builtin hash -p /usr/bin/gdialog gdialog
builtin hash -p /usr/bin/man man
builtin hash -p /usr/bin/info info
builtin hash -p /bin/ls ls
builtin hash -p /sbin/fuser fuser
```

例如，显示指定命令的完整路径，命令如下所示。

```
# hash -t info              //显示 info 命令的完整路径
/usr/bin/info
```

例如，向哈希表中增加记录 aclocal，其路径为 “/usr/bin/aclocal”，命令如下所示。

```
# hash -p /usr/bin/aclocal aclocal    //向哈希表中增加 aclocal 条目
# hash -l                            //显示哈希表
builtin hash -p /usr/bin/tload tload
builtin hash -p /usr/bin/procinfo procinfo
```



```
builtin hash -p /usr/bin/ipcs ipcs
builtin hash -p /usr/bin/vim vim
builtin hash -p /bin/cat cat
builtin hash -p /usr/bin/pmap pmap
builtin hash -p /bin/ps ps
builtin hash -p /usr/bin/gdialog gdialog
builtin hash -p /usr/bin/aclocal aclocal //已成功添加
builtin hash -p /usr/bin/man man
builtin hash -p /usr/bin/info info
builtin hash -p /bin/ls ls
builtin hash -p /sbin/fuser fuser
```

例如，删除哈希表内的所有内容，命令如下所示。

```
# hash -r //删除哈希表内容
# hash -l //显示哈希表内容
hash: hash table empty //哈希表已为空
```

3.1.21 hwclock 命令：显示和设置硬件时钟

hwclock 命令用于显示和设置系统硬件时钟。通常将硬件时钟设定为一个指定的时间，也可以设置为系统时间。hwclock 命令的语法格式如下所示。

```
hwclock [--adjust][--debug][--directisa][--directisa][--show][--hctosys][--se--date]
[--show][--systohc][--test]
[--utc][--setepoch,--epoch=year][--version][ --date][ --badyear][ --noadjfile][--help]
```

其中各选项的具体说明如表 3-17 所示。

表 3-17 hwclock 命令各选项说明

选 项	说 明
--adjust	自动调整硬件时钟时间
--debug	运行时，输出详细的调试信息
--directisa	不通过设备文件/etc/rtc，直接对硬件时钟进行存取
--getepoch	输出硬件时钟的数值到标准输出
--hctosys	时钟同步，使系统时钟和硬件时钟同步
--set	设置硬件时间
--show	显示硬件时钟到标准输出
--systohc	系统时间写入硬件时钟
--test	进行测试
--utc	使用 UTC 时间
--setepoch,--epoch=year	设置硬件时钟的年份
--version	版本信息
--localtime	使用本地时间
--noadjfile	不使用/etc/adjtime 文件对时间进行调整
--badyear	当 BIOS 失效时，忽略硬件时间中的年份
--version	显示版本信息
--date	对硬件时间进行设置
--help	显示帮助信息



UTC 时间是 Universal Time Coordinated 的缩写。在 GPS 系统中有两种时间概念，一为 UTC，另一为 LT（Local Time，地方时）。两者的区别为时区不同，UTC 是 0 时区的时间，地方时为本地时间。例如，北京为东 8 区，如果北京是早上 8 点，UTC 时间就为 0 点，比北京晚 8h。

例如，获取当前的硬件时间，默认采用本地时间，命令如下所示。

```
# hwclock //获取当前的时间
2008 年 05 月 30 日 星期五 14 时 35 分 52 秒 -0.609969 seconds
```

例如，获取当前的硬件时间，使用 UTC 时间，命令如下所示。

```
# clock -utc
# hwclock -utc //显示 UTC 时间
2008 年 05 月 30 日 星期五 22 时 37 分 36 秒 -0.670434 seconds
```

例如，显示 hwclock 命令的版本信息，命令如下所示。

```
# hwclock --version
hwclock from util-linux-2.11y
```

3.1.22 ipcs 命令：显示 IPC 设备信息

ipcs 命令将提供调用进程可以访问的 IPC 设备信息，对进程通信进行描述。如果不指定参数，那么输出的是当前系统中活跃的消息队列、共享内存段和标志信息设置。该命令的语法格式如下所示。

```
ipcs [ -a ][ -s ][ -m ][ -q ][ -c ][ -l ][ -u ][ -p ][ -t ]
```

其中各选项的具体说明如表 3-18 所示。

表 3-18 ipcs 命令的选项说明

选 项	说 明
-m	显示共享内存
-q	显示消息队列
-s	显示信号队列
-a	显示全部信息，该项为默认选项
-t	显示时间信息
-p	显示进程 PID 信息
-c	显示创建者信息
-l	显示限制信息，例如最大队列长度、段最小尺寸
-u	显示统计信息

例如，显示 IPC 信息，命令如下所示。

```
# ipcs //显示进程间通信状态

----- Shared Memory Segments -----
key          shmid    owner    perms    bytes    nattch    status
0x00000001  32768    root     600      655360    2

----- Semaphore Arrays -----
key          semid    owner    perms    nsems
```



```
----- Message Queues -----
key      msqid    owner    perms    used-bytes  messages
```

例如，显示进程 PID 及时间信息，命令如下所示。

```
# ipcs -p -t //显示进程 PID 及时间信息

----- Shared Memory Attach/Detach/Change Times -----
shmids   owner    attached      detached      changed
32768    root      May 30 12:08:28  May 30 12:08:28  May 29 00:13:44

----- Shared Memory Operation/Change Times -----
shmids   owner    last-op      last-changed

----- Message Queues Send/Recv/Change Times -----
msqids   owner    send      recv      change
```

例如，显示共享内存及创建者信息，命令如下所示。

```
# ipcs -m -c

----- Shared Memory Segment Creators/Owners -----
shmids   perms    cuid    cgid    uid    gid
32768    600      root    root    root    root
```

3.1.23 isosize 命令：显示 iso9660 文件系统大小

isosize 命令用于显示输出 iso9660 文件系统的大小。该文件可以是普通的文件，也可以是一个块设备，如/dev/hdd 或/dev/sr0。如果没有指定相关选项，默认以字节的方式显示大小。该命令支持大于 4GB 的文件系统。该命令的语法格式如下所示。

```
isosize [-x] [-d <num>] <iso9660_image_file>
```

其中各选项的具体说明如表 3-19 所示。

表 3-19 命令 isosize 的选项说明

选 项	说 明
-x	以用户易读的方式进行显示，例如，以“扇区”代替“块”
-d<num>	与-x 不同时使用。显示的结果是实际大小（以字节表示）除以指定的 num。所以，如果 num 表示“块”大小，则输出的结果将是块数
iso9660_image_file	指定的 iso9660 文件系统

例如，光驱设备属于 iso9660 文件系统，显示其大小（字节表示），使用的命令如下所示。

```
# isosize /dev/cdrom //显示文件大小
668991488
```

例如，设置 1 “块”为 1024 字节，以“块”为单位显示光驱设备大小，命令如下所示。

```
# isosize -d 1024 /dev/cdrom
653312
```

例如，以用户易读的方式显示光驱设备大小，命令如下所示。

```
# isosize -x /dev/cdrom
sector count: 326656, sector size: 2048
```

可以看到，扇区数为 326 656，扇区大小为 2 048 字节。

3.1.24 jobs 命令：显示工作清单

jobs 命令显示当前正在后台执行的作业。在得到作业的相关信息之后，可以对作业进行



进一步的操作，如使用 `fg` 命令将作业调到前台执行，或者使用 `kill` 命令杀死指定的作业。`jobs` 命令的语法格式如下所示。

```
jobs [-lnprs] [jobspec ...] or jobs -x command [args]
```

其中各选项的具体说明如表 3-20 所示。

表 3-20 jobs 命令的选项说明

选 项	说 明
-p	仅列出作业 ID
-h	仅列出已经更改状态的作业 ID
-l	列出后台作业的所有信息
-s	仅列出已经停止的作业
-r	仅列出正在运行的作业

例如，下面命令创建了两个后台作业并通过 `jobs` 命令对作业情况进行查询，如下所示。

```
# vi &                                //后台运行 vi
[1] 14152
# cat &                                //后台运行 cat
[2] 14153
[1]+ Stopped vim
# jobs                                  //列出当前作业
[1]- Stopped vim
[2]+ Stopped cat
# jobs -l                               //列出作业的详细信息
[1]- 14152 停止 (tty 输出) vim
[2]+ 14153 停止 (tty 输入) cat
# jobs -p                               //列出作业 ID
14152
14153
# jobs -s                               //列出停止执行的作业
[1]- Stopped vim
[2]+ Stopped cat
```

3.1.25 lsmod 命令：显示所有已加载的模块

`lsmod` 命令用于显示所有已加载的模块信息，其显示的信息与文件 `/proc/modules` 的内容相同。`lsmod` 命令的语法格式如下所示。

```
lsmod [-V,--version][-h, --help]
```

其中，选项 `-V, --version` 用于显示 `lsmod` 命令的版本信息，选项 `-h, --help` 用于显示帮助信息。例如，下面使用 `lsmod` 命令显示加载的模块。

```
# lsmod
Module              Size Used by Not tainted
ide-cd              35708  0 (autoclean)
cdrom               33728  0 (autoclean) [ide-cd]
parport_pc         19076  1 (autoclean)
lp                 8996  0 (autoclean)
parport            37056  1 (autoclean) [parport_pc lp]
autofs             13268  0 (autoclean) (unused)
pcnet32            18240  1
```



```
mii                3976  0 [pcnet32]
ipt_REJECT         3928  6 (autoclean)
iptables_filter    2412  1 (autoclean)
ip_tables          15096 2 [ipt_REJECT iptables_filter]
microcode          4668  0 (autoclean)
keybdev            2944  0 (unused)
mousedev           5492  1
hid                22148 0 (unused)
input              5856  0 [keybdev mousedev hid]
usb-uhci           26348 0 (unused)
ehci-hcd           19976 0 (unused)
usbcore            78784 1 [hid usb-uhci ehci-hcd]
ext3               70784 2
jbd                51892 2 [ext3]
BusLogic           100796 3
sd_mod             13452 6
scsi_mod           107128 2 [BusLogic sd_mod]
```

可以看到，所显示内容共分 4 个字段，其中第 1 个字段表示模块名称，第 2 个字段表示模块的大小，第 3 个字段表示使用的个数，第 4 个字段表示所涉及的模块列表。

3.1.26 locale 命令：显示当前系统的语言设置

locale 命令用于显示当前系统的语言设置，该命令的语法格式如下所示。

```
locale [-m,--charmaps] [-a,--all-locales] [-c,--category-name] [-k,--keyword-name] [-v,
--verbose]
[ -?, --help] [ --usage] [ -V, --version]
```

其中各选项的具体说明如表 3-21 所示。

表 3-21 locale 命令选项说明

选 项	说 明
系统信息	
-a, --all-locales	输出所有可用的语言名称
-m, --charmaps	输出所有可用的字符集
修改输出格式	
-c, --category-name	按指定的类别输出
-k, --keyword-name	按指定的关键字输出
-v, --verbose	输出更多信息
-?, --help	输出该命令的帮助列表
--usage	输出该命令简要用法信息
-V, --version	输出该命令的版本号

例如，下面命令显示输出系统当前的语言设置。

```
# locale
LANG=zh_CN.GB18030
LC_CTYPE="zh_CN.GB18030"
LC_NUMERIC="zh_CN.GB18030"
LC_TIME="zh_CN.GB18030"
LC_COLLATE="zh_CN.GB18030"
LC_MONETARY="zh_CN.GB18030"
```



```
LC_MESSAGES="zh_CN.GB18030"
LC_PAPER="zh_CN.GB18030"
LC_NAME="zh_CN.GB18030"
LC_ADDRESS="zh_CN.GB18030"
LC_TELEPHONE="zh_CN.GB18030"
LC_MEASUREMENT="zh_CN.GB18030"
LC_IDENTIFICATION="zh_CN.GB18030"
LC_ALL=
... ..
```

3.1.27 mesg 命令：显示信息

mesg 命令用于设置是否允许其他用户通过 write 命令给自己发送信息。mesg 命令的语法格式如下所示。

```
mesg [y][n]
```

其中各选项的说明如表 3-22 所示。

表 3-22

mesg 命令选项说明

选 项	说 明
y	允许其他用户发送信息到当前终端
n	禁止其他用户发送信息到当前终端

例如，下面的命令允许其他用户发送信息到当前终端。

```
[root@localhost root]# mesg y
[zhangsan@zhangsan etc]$ write root pts/4           //在另一个终端上写入
Hello I am Zhangsan
Who are you                                           //用 Ctrl+D 结束
[roo@yingzheng ~]#                                   //在被写入终端上的输出
Message from Zhangsan@ Zhangsan on pts/3 at 08:57 ...
Hello I am Zhangsan
Who are you EOF
```

例如，下面的命令禁止其他用户发送信息到当前终端。

```
[root@localhost root]# mesg n                       //首先在一个终端上设置为禁止
[zhangsan @ Zhangsan /etc]$ write root pts/4         //另一个终端上用 write 写入被禁止
write: root has messages disabled on pts/4
```

3.1.28 minfo 命令：显示 ms-dos 文件系统的各项参数

minfo 命令用于显示 MS-DOS 文件系统的各项参数，包括扇区数、磁头数、柱面数等。该命令的语法格式如下所示。

```
minfo [-v] drive
```

其中各选项的具体说明如表 3-23 所示。

表 3-23

minfo 命令的选项说明

选 项	说 明
drive	指定的驱动器
-v	除显示文件系统的各项参数外，以十六进制显示启动扇区内容
--help	显示帮助信息
--version	显示版本信息

例如，显示 DOS 系统 C 盘的参数信息并显示启动扇区内容，命令如下所示。

```
# minfo -v C: //显示系统参数
```

3.1.29 modinfo 命令：显示内核模块信息

modinfo 命令用来显示内核模块信息。modinfo 命令会检测与内核相关的模块文件，并显示检测的结果。该命令的语法格式如下所示。

```
modinfo[-a,--author][-d,--description][-l,--license][-n,--filename][-p,--parameters][-V,--version][-h,--help]
```

其中各选项的具体说明如表 3-24 所示。

表 3-24 modinfo 命令的选项说明

选 项	说 明
-a, --author	显示模块作者
-d, --description	显示模块描述信息
-l, --license	显示模块许可
-n, --filename	显示模块文件名称
-p, --parameters	显示模块支持的参数信息
-V, --version	显示版本信息
-h, --help	显示帮助信息

例如，显示模块 sg 的相关信息，命令如下所示。

```
# modinfo sg
/lib/modules/2.4.20-8/kernel/drivers/scsi/pci2000.o
[root@teacher scsi]# modinfo sg
filename:      /lib/modules/2.4.20-8/kernel/drivers/scsi/sg.o
description: "SCSI generic (sg) driver"
author:        "Douglas Gilbert"
license:       "GPL"
parm:          def_reserved_size int, description "size of buffer reserved for each fd"
```

例如，显示模块 pci2000 的模块文件，命令如下所示。

```
# modinfo -n pci2000
/lib/modules/2.4.20-8/kernel/drivers/scsi/pci2000.o
```

3.1.30 mev 命令：监视鼠标情况

mev 命令是 gpm 包的一部分，可以将鼠标控制的事件打印到标准控制台上。该命令的语法格式如下所示。

```
mev [-C<number>][-d<number>][-e<number>][-E][-i][-f][-m<number>][-M<number>][-p][-u]
```

其中各选项的具体说明如表 3-25 所示。

表 3-25 mev 命令的选项说明

选 项	说 明
-C<number>	选择虚拟控制台
-d<number>	选择默认掩码，所有不属于事件掩码的事件都不会被打印
-e<number>	选择事件掩码
-E	Emacs 模式。在 Emacs 模式下，鼠标事件不再以数字格式表示，而是以 lisp 格式表示



续表

选 项	说 明
-i	交互模式，接受从标准终端输入的命令，并依此改变连接参数
-f	识别屏幕内的拖曳事件
-m<number>	最小修正掩码，低于这个修正掩码的事件将不被上报
-M<number>	最大修正掩码，高于这个修正掩码的事件将不被上报
-p	在拖曳期间描绘出指针的轨迹
-u	用户模式，该项为默认项

例如，显示鼠标事件，命令如下所示。

```
#mev
H/#H/H/#H/b.#b.b.#b.B)#?*7-#-/./#./B(#B(='#=':(#:
```

 //单击鼠标后显示事件信息

3.1.31 pstree 命令：以树状图显示程序

pstree 命令将所有进程以树状图的方式显示。树状图默认会以 init 进程为根，如果指定了 pid 选项，则只显示指定进程为根的树状图。pstree 的命令格式如下。

```
pstree [-a] [-c] [-h|-Hpid] [-l] [-n] [-p] [-u] [-Z] [-A|-G|-U] [pid][user]
```

 其中各选项的具体说明如表 3-26 所示。

表 3-26 pstree 命令的选项说明

选 项	说 明
-a	显示进程完整的指令及参数
-p	显示进程的进程号
-c	不进行精简，即不对同等的子树进行压缩
-h	高亮显示当前正在执行的进程及其父进程
-p	显示进程标识号，即 PID
-u	显示用户名称
-l	不对过长的记录进行剪切
-n	以 PID 进行排序
--version	显示版本信息
--user	显示指定用户的子树

例如，以树状图方式显示进程及进程号，命令行如下。

```
#pstree -p
init(1)───/usr/bin/sealer(3454)
    │
    ├──acpid(2065)
    │
    ├──at-spi-registry(7998)
    │
    ├──atd(2255)
    │
    ├──auditd(1785)───python(1787)
    │               │
    │               └─{auditd}(1786)
    │
    ├──automount(2046)───{automount}(2047)
    │                   │
    │                   ├──{automount}(2048)
    │                   └─{automount}(2051)
```



```
      |
      |-----{automount} (2054)
      |
      |-----avahi-daemon (2286)-----avahi-daemon (2287)
... ..
```

例如，高亮显示当前正在执行的进程，命令如下所示。

```
# pstree -h
init--+-apmd
      |-atd
      |-smbd---smbd
      |-sshd
      |-syslogd
      |-xfs
      `--xinetd--+-fam
                  `--in.telnetd---login---bash---pstree
```

3.1.32 pwd 命令：显示当前目录

pwd 命令用于显示用户当前所在目录的绝对路径。pwd 命令的语法格式如下所示。

```
pwd [--help] [--version]
```

其中，选项 “--help” 用于显示帮助信息，选项 “--version” 用于显示版本信息，例如下列命令所示。

```
# cd /etc/rc.d           //将当前目录切换到/etc/rc.d
# pwd                    //显示当前的目录
/etc/rc.d
```

3.1.33 proc 文件系统：查看系统内核信息

proc 不是一个真实的文件系统，不占用外存空间，只是以文件的方式为访问 Linux 内核数据提供接口。用户和应用程序可以通过查看 /proc 目录中的相关文件得到系统的运行信息，并可以改变内核的某些参数。由于系统的信息总是动态变化的，所以用户或应用程序读取 proc 文件时所获得的数据也是瞬时的。许多应用程序依靠 proc 来访问 Linux 内核信息。查看 /proc 目录如下。

```
# cd /proc
# ls
1      1939  2230  307   3287  3377  7      fs      net
10     1952  2231  3072  3289  3379  723    ide     partitions
12     1964  2252  3073  3291  3393  74     interrupts schedstat
1225   2      2271  3074  3298  3395  78     iomem   scsi
13     2004  2283  3075  3299  3397  80     ioports self
131    2024  2284  3076  3301  3408  9      irq     slabinfo
132    2043  2295  3077  3313  3420  acpi    kallsyms stat
133    2062  2296  3078  3317  3428  asound  kcore   swaps
134    2073  2302  312   3319  3430  buddyinfo keys     sys
1770   2078  2305  317   3323  3431  bus     key-users sysrq-trigger
1782   2090  2315  3179  3327  3442  cmdline kmsg    sysvipc
1784   2104  2559  3183  3333  3444  cpuinfo loadavg tty
1798   2116  2564  3186  3336  3445  crypto  locks   uptime
1801   2129  2565  320   3343  3520  devices mdstat  version
1828   2149  2567  3211  3350  353   diskstats meminfo vmcore
1841   2158  2569  3240  3352  387   dma     misc    vmstat
1854   2170  276   3274  3354  4      driver  modules xen
1875   2181  2938  3277  3366  5      execdomains mounts  zoneinfo
```




```
1908 2216 3 3278 3368 5628 fb mpt
1927 2227 3060 3285 3371 6 filesystems mtrr
```

其中每一个以数字命名的目录对应系统中运行的一个进程，该数字即为进程的 PID 号。数字目录中存放了该进程的运行信息，可以通过相应的命令进行查询。例如查询当前系统中运行的 vi 程序的相关信息如下所示。

```
# ps aux | grep vi
root      5716  0.0  0.4  4956 1012 pts/1    T   11:55   0:00 vi      //5716 为 vi 的进程号
root      5778  0.0  0.2  4136  668 pts/1    R+  12:01   0:00 grep vi

# ls /proc          //查看 proc 目录下是否已存在 5716 目录
1      1939  2230  307  3287  3377  5717      fb      mpt
10     1952  2231  3072 3289  3379  6        filesystems mtrr
12     1964  2252  3073 3291  3393  7        fs      net
1225   2     2271  3074 3298  3395  723     ide     partitions
13     2004  2283  3075 3299  3397  74      interrupts schedstat
131    2024  2284  3076 3301  3408  78      iomem   scsi
132    2043  2295  3077 3313  3420  80      ioports self
133    2062  2296  3078 3317  3428  9       irq     slabinfo
134    2073  2302  312  3319  3430  acpi    kallsyms stat
1770   2078  2305  317  3323  3431  asound  kcore   swaps
1782   2090  2315  3179 3327  3442  buddyinfo keys     sys
1784   2104  2559  3183 3333  3444  bus     key-users sysrq-trigger
1798   2116  2564  3186 3336  3445  cmdline kmsg    sysvipc
1801   2129  2565  320  3343  3520  cpuinfo loadavg tty
1828   2149  2567  3211 3350  353  crypto  locks   uptime
1841   2158  2569  3240 3352  387  devices mdstat  version
1854   2170  276  3274 3354  4     diskstats meminfo vmcore
1875   2181  2938  3277 3366  5     dma     misc    vmstat
1908   2216  3     3278 3368  5685  driver  modules xen
1927   2227  3060  3285 3371  5716  execdomains mounts  zoneinfo

# ls -l /proc/5716/exe          //查询 vi 所执行的程序
lrwxrwxrwx 1 root root 0 07-25 11:57 /proc/5716/exe -> /bin/vi
# ls -l /proc/5716/cwd          //查询 vi 的当前目录
lrwxrwxrwx 1 root root 0 07-25 11:57 /proc/5716/cwd -> /proc
# ls -l /proc/5716/environ      //查询 vi 的运行环境
-r----- 1 root root 0 07-25 11:57 /proc/5716/environ
```



在 /proc 目录中，通过查看 `cpuinfo` 文件可以获得处理器的详细信息；通过 `interrupts` 文件可以查看当前系统使用的中断号；通过 `uptime` 文件可以查看系统运行的时间；通过 `filesystems` 文件，可以查看当前系统支持的文件系统类型。

3.1.34 ps 命令：报告程序状况

`ps` 命令用于监测进程的工作情况。进程是正在运行的程序，一直处于动态变化过程中，而 `ps` 命令所显示的进程工作状态是瞬时间的。如果试图连续查看某一进程的工作情况，必须连续使用该命令或者换用能够动态显示进程状态的 `top` 命令。`ps` 的命令格式如下。

```
ps [a][-a][-A][c][-N][-e][e][-H][T][-f][-h][-l][-w] [-m][x][r][-t][-U][ -C]
[--help]
[ --lines][ --width][--version]
```

其中各选项的具体说明如表 3-27 所示。

表 3-27

ps 命令各选项说明

选 项	说 明
a	显示所有的进程，包括其他用户的进程
-A	显示所有的进程
-a	所有的终端进程，但不包括 session leaders
c	显示命令名称
-N	反向选择
-e	显示所有进程，等同于-A
e	显示环境变量
-s	指定会话相关进程
-t	指定终端相关进程
-o,o	按用户自行定义的内容进行显示
-u	选择显示有效的用户的 UID 或用户名
x	显示没有控制终端的进程
-m	显示线程
-H	显示树状结构
r	仅显示正在运行的进程
T	显示当前终端的所有进程
-U	指定用户的进程
-C	指定命令相关的进程
--help	帮助信息
--version	版本信息
--lines	每页显示的行数
--width	每行显示的字符数

例如，查看当前所有进程，命令行如下。

```
# ps -e
  PID TTY          TIME CMD
    1 ?            00:00:04 init
    2 ?            00:00:00 migration/0
    3 ?            00:00:00 ksoftirqd/0
    4 ?            00:00:00 watchdog/0
... ..
```

例如，显示所有不是以 root 身份运行的进程，命令行如下。

```
# # ps -u root -N
  PID TTY          TIME CMD
 2084 ?            00:00:00 portmap
 2103 ?            00:00:00 rpc.statd
 2268 ?            00:00:00 mysqld
 2269 ?            00:00:00 mysqld
 2270 ?            00:00:00 mysqld
... ..
```



例如，根据用户指定的排列方式进行显示，命令行如下。

```
# ps -eo pid,tt,user,fname,f,tmout,wchan
PID TT      USER  COMMAND      F      TMOUT  WCHAN
  1 ?        root   init          -      4      -
  2 ?        root   migratio      1      -      migration_thread
  3 ?        root   ksoftirq      1      -      ksoftirqd
  4 ?        root   watchdog      5      -      watchdog
  5 ?        root   events/0      5      -      worker_thread
  6 ?        root   khelper       1      -      worker_thread
```

例如，显示所有进程状态和进程的基本信息，命令行如下。

```
# ps ax
PID TTY      STAT   TIME COMMAND
  1 ?        S      0:04 init
  2 ?        SW     0:00 [keventd]
  3 ?        SW     0:00 [kapmd]
  4 ?        SWN    0:00 [ksoftirqd_CPU0]
  9 ?        SW     0:06 [bdflush]
  5 ?        SW     0:03 [kswapd]
  6 ?        SW     0:00 [kscand/DMA]
```

例如，显示与 `syslogd` 进程相关的进程号，命令如下所示。

```
# ps -C syslogd -o pid=
2062
```

例如，显示与 `httpd` 进程相关的进程号，命令如下所示。

```
# ps -C httpd -o pid=
2522
2523
2524
2541
2542
2543
```

由于 `httpd` 进程启动后又启动了 5 个附加子进程，所以共列出了 6 个进程，但主进程号是 2522。如果希望杀死 `httpd` 进程，只需执行如下命令。

```
# kill 2522
# ps -C httpd -o pid=
#
```

可以看到，再次运行“`ps -C httpd -o pid=`”命令显示结果为空，因为 `httpd` 进程已经被杀死。但如果只杀死子进程，并不会杀死 `httpd` 进程，如下所示。

```
# kill 2543
# ps -C httpd -o pid=
2522
2523
2524
2541
2542
10425
```

可以看到，虽然 2543 进程被杀死了，但 `httpd` 相关进程依然是 6 个。这是因为 `httpd` 进程的配置文件中定义进程必须启动 5 个附加子进程，虽然 2522 被杀死，但很快 `httpd` 进程会再启动一个进程。

3.1.35 pmap 命令：显示程序的内存信息

pmap 命令用来显示指定程序的内存信息。该命令的语法格式如下所示。

```
pmap [pid] [/proc/pid]
```

pmap 命令可以使用的选项包括“pid”和“/proc/pid”，即可以通过进程的 PID 或/proc 目录下相关进程的子目录，对进程进行查询。

例如，首先使用 ps 命令查询 man 进程的 PID，然后查询 man 进程的详细内存信息，命令如下所示。

```
# ps -a //显示进程信息
  PID TTY          TIME CMD
 9855 pts/2    00:00:00 man
 9858 pts/2    00:00:00 sh
 9859 pts/2    00:00:00 sh
 9866 pts/2    00:00:00 less
10881 pts/0    00:00:00 ps

# pmap 9855 //显示进程 9855 的内存信息
man(9855)
08048000 (36 KB)    r-xp (08:02 340541)  /usr/bin/man
08051000 (4 KB)     rw-p (08:02 340541)  /usr/bin/man
08052000 (204 KB)   rwxp (00:00 0)
40000000 (84 KB)    r-xp (08:02 453447)  /lib/ld-2.3.2.so
40015000 (4 KB)     rw-p (08:02 453447)  /lib/ld-2.3.2.so
40016000 (4 KB)     rw-p (00:00 0)
40017000 (2048 KB)  r--p (08:02 469632)  /usr/lib/locale/locale-archive
40217000 (1060 KB)  r--p (08:02 469632)  /usr/lib/locale/locale-archive
40320000 (4 KB)     rw-p (00:00 0)
42000000 (1208 KB)  r-xp (08:02 340050)  /lib/tls/libc-2.3.2.so
4212e000 (12 KB)    rw-p (08:02 340050)  /lib/tls/libc-2.3.2.so
42131000 (8 KB)     rw-p (00:00 0)
bffff3000 (52 KB)   rwxp (00:00 0)
mapped: 4728 KB writable/private: 292 KB shared: 0 KB
```

3.1.36 procinfo 命令：显示系统状态信息

proinfo 命令用来显示系统的状态信息，其信息来源于/proc 目录。procinfo 命令可以显示如表 3-28 所示的信息。

表 3-28 procinfo 命令显示信息列表

显示信息内容	说 明
Memory（内存）	显示内存信息，与 free 命令显示内容相似
Bootup（启动）	系统的启动时间
Load Average（平均负载）	显示正在运行作业的平均数，正在运行的进程数和所有进程的数目
User（用户）	运行作业所消耗的用户时间
Nice（优化）	运行优化作业所消耗的用户时间
System（系统）	运行作业所消耗的系统时间
Idle（空闲）	空闲时间
Uptime（）	系统运行时间



续表

显示信息内容	说 明
Page In (块存入页)	磁盘存入 core (当某个程序在执行时发生错误, Linux 系统会把程序出错的内存内容映像下来供程序员调试查错用。通常是写在一个叫 core 的文件中) 的块数目 (通常 1 块为 1024 字节)
Page Out (块恢复页)	从 core 恢复到磁盘的块数目
Swap In (内存存入页)	从交换分区 (swapspace) 存入的内存页数目
Swap Out (内存换出页)	恢复到交换分区的内存页数目
Context (上下文切换)	系统自启动以来的上下文切换数目
Disk 1-4 (磁盘 1-4)	磁盘被访问的次数
Interrupts (中断)	IRQ 通道。对于 Intel 架构, 共有 16 个 IRQ 通道, 其含义如表 3-29 所示
Modules (模块)	显示系统已加载的设备驱动模块的大小, 以 KB 为单位
Character and Block Devices (字符设备和块设备)	所有可用设备的主设备号
File Systems (文件系统)	所有的文件系统

表 3-29 Intel 架构中的中断定义

通 道	说 明
0	计时器通道 0
1	键盘
2	控制器 2 的级联 (用于控制 IRQ8-15)
3	串口 2
4	串口 1
5	并口 2
6	软磁盘控制器
7	并口 1
8	时实时钟
9	重定向到 IRQ2
10	未定义
11	未定义
12	未定义
13	数字协处理器
14	硬盘控制器
15	未定义

procinfo 命令的语法格式如下所示。

```
procinfo [-f][-m][-d][-i][-D][-S][-b][-h][-v] [-nN] [-Ffile]
```

其中各选项的具体说明如表 3-30 所示。

表 3-30 procinfo 命令的选项说明

选 项	说 明
-m	显示模块和设备信息
-s	显示内存、磁盘、IRQ 及 DMA 信息，该项为默认选项
-a	显示所有信息
-f	全屏模式
-i	显示所有 IRQ 通信，但不包括正被使用的
-d	显示内存、CUP、交换分区、磁盘、中断等信息，每秒更新一次
-D	显示当前内存、交换分区等的使用情况。除了内存以总计方式进行显示外，与“-d”相同，
-S	如果和“-d”或“-D”配合使用，则每秒显示一次信息。如果同时指定了“-nN”选项，则按每 N 秒一次显示信息
-b	如果系统的内核版本足够新，则显示块数目，而不是 I/O 请求数
-r	显示内存使用情况，包括实际空闲内存
-F<file>	将信息结果输出到指定文件
-h	帮助信息
-n<N>	周期性更新的时间，即每隔 N 秒刷新一次
-v	版本信息

例如，显示系统状态，命令如下所示。

```
# procinfo //显示系统状态
Linux 2.4.20-8 (bhcompile@porky) (gcc 3.2.2 20030222 ) #1 1CPU [teacher]

Memory:      Total      Used      Free      Shared      Buffers      Cached
Mem:         255264      242420      12844      0           49096        107644
Swap:        522104      21476      500628

Bootup: Thu May 29 00:13:00 2008   Load average: 0.00 0.00 0.00 1/90 11057

user  :      0:05:12.94   0.2% page in :      871894 disk 1:  119426r 140623w
nice  :      0:00:03.21   0.0% page out:    1460982
system:      0:08:54.25   0.3% swap in :        449
idle  :  2d  2:50:39.57  99.5% swap out:      6980
uptime:  2d  3:04:49.97      context : 25084298

irq 0: 18388997 timer          irq 8:      10 rtc
irq 1:      1078 keyboard      irq 9:        0 usb-uhci
irq 2:        0 cascade [4]    irq 10:    63890 PCnet/PCI II 79C970A
irq 3:        3              irq 11:   234513 BusLogic BT-958
irq 4:        3              irq 12:   30501 PS/2 Mouse
irq 5:        0 ehci-hcd       irq 15:  263591 idel
irq 6:        4
```

3.1.37 pwd 命令：显示当前目录

pwd 命令用于显示用户当前所在目录，目录以绝对路径表示。pwd 的语法格式如下所示。



```
pwd [--help][--version]
```

其中，选项“--help”用于显示帮助信息，选项“--version”用于显示版本信息。例如，显示当前用户所在目录，命令如下所示。

```
[root@teacher root]# pwd //显示当前的目录
/root
```

3.1.38 rulevel 命令：显示系统运行级别

runlevel 命令读取系统的/var/run/utmp 文件，读入系统的运行级别记录，然后输出前一个和当前的运行级别，如果没有记录上一次系统的运行等级，则用 N 表示。

例如，显示系统运行等级，命令如下所示。

```
# runlevel //显示执行等级
N 5
#
```

系统运行级别相关说明见表 3-31。

表 3-31 运行级别说明

运行级别	说 明
0	关闭系统
1	单用户模式
2	多用户模式，但不支持 NFS
3	完全的多用户模式
4	保留
5	图形用户模式
6	重新启动



系统的初始运行级别设置存放在文件/etc/inittab 里，不要把系统初始运行级别设置为 0 或 6，否则系统无法正常启动。

3.1.39 sysctl 命令：设置系统核心参数

sysctl 命令可以用来实时设置系统的核心参数。该命令的语法格式如下所示。

```
sysctl [-n][-e][-p] [-a][-A] [ -w<variable=key>]
```

其中各选项的具体说明如表 3-32 所示。

表 3-32 sysctl 命令的选项说明

选 项	说 明
-a	显示当前所有的设置
-A	以表格的形式显示当前所有的设置
-e	模糊模式，忽略由于未知关键字引起的错误
-n	显示输出时忽略关键词名

续表

选 项	说 明
-w<variable=key>	设置变量值
-p	指定配置文件，默认为/etc/sysctl.conf
-h	显示帮助信息
-V	显示版本信息

例如，显示所有核心设置，命令如下所示。

```
# sysctl -a
abi.fake_utsname = 0
abi.trace = 0
abi.defhandler_libcso = 68157441
abi.defhandler_lcall7 = 68157441
abi.defhandler_elf = 0
abi.defhandler_coff = 117440515
dev.cdrom.check_media = 0
dev.cdrom.lock = 1
dev.cdrom.debug = 0
dev.cdrom.autoeject = 0           //设置自动弹出
dev.cdrom.autoclose = 1
dev.cdrom.info = CD-ROM information, Id: cdrom.c 3.12 2000/10/18
... ..
```

例如，设置变量 dev.cdrom.autoeject 的值为 1，命令如下所示。

```
# sysctl -w dev.cdrom.autoeject =1           //设置变量
dev.cdrom.autoeject =1
```

3.1.40 stat 命令：显示文件/文件系统的详细信息

stat 命令用于显示文件或文件系统的详细信息，该命令语法格式如下所示。

```
stat [-L,--dereference] [-f,--filesystem] [--version] [-t,--terse] [--help] [filename...]
```

其中各选项的具体说明如表 3-33 所示。

表 3-33 命令 stat 的选项说明

选 项	说 明
-f	不显示文件本身的信息，显示文件所在文件系统的信息
-L, --dereference	显示符号链接
-t, --terse	简洁模式，只显示摘要信息
--help	帮助信息
--version	版本信息
filename	待查询的文件名称



事实上，stat 命令显示的是文件的 I 节点信息。linux 文件系统以块（block）为单位存储信息，为了找到某一个文件在存储空间中存放的位置，用 I 节点对一个文件进行索引。所谓 I 节点是指文件系统管理的一个数据结构，是一个 64 字节长的表，它包含了描述一个文件所必需的全部信息，包括文件的大小、文件所有者、文件的存取权限、类型等重要信息。



例如，查询文件/etc/rc.d/rc0.d 文件的详细信息，命令如下所示。

```
# ll //以长格式显示文件列表
总用量 64
drwxr-xr-x 2 root root 4096 5月 28 23:58 init.d
-rwxr-xr-x 1 root root 2338 2003-02-19 rc
drwxr-xr-x 2 root root 4096 1月 22 05:29 rc0.d
drwxr-xr-x 2 root root 4096 1月 22 05:29 rc1.d
drwxr-xr-x 2 root root 4096 1月 22 05:29 rc2.d
drwxr-xr-x 2 root root 4096 1月 22 05:29 rc3.d
drwxr-xr-x 2 root root 4096 1月 22 05:29 rc4.d
drwxr-xr-x 2 root root 4096 1月 22 05:29 rc5.d
drwxr-xr-x 2 root root 4096 1月 22 05:29 rc6.d
-rwxr-xr-x 1 root root 258 1月 21 20:09 rc.local
-rwxr-xr-x 1 root root 23299 2003-02-25 rc.sysinit
# stat rc0.d //显示文件 rc0.d 的详细信息
  File: 'rc0.d'
  Size: 4096          Blocks: 8          IO Block: 4096   Directory
Device: 802h/2050d   Inode: 32459        Links: 2
Access: (0755/drwxr-xr-x)  Uid: ( 0/   root)   Gid: ( 0/   root)
Access: 2008-05-29 04:02:12.000000000 +0800
Modify: 2008-01-22 05:29:50.000000000 +0800
Change: 2008-01-22 05:29:50.000000000 +0800
```

例如，查询文件 rc0.d 所在的文件系统的详细信息，命令如下所示。

```
# stat -f rc0.d
  File: "rc0.d"
   ID: 0          Namelen: 255          Type: ext2/ext3
Blocks: Total: 1909350   Free: 357425          Available: 260433   Size: 4096
Inodes: Total: 971520   Free: 662420
```

直接查询/dev/sda 文件系统的详细信息，命令如下所示。

```
# stat /dev/sda
  File: '/dev/sda'
  Size: 0          Blocks: 0          IO Block: 4096   Block Device
Device: 802h/2050d   Inode: 66914        Links: 1          Device type: 8,0
Access: (0660/brw-rw----)  Uid: ( 0/   root)   Gid: ( 6/   disk)
Access: 2003-01-30 18:24:37.000000000 +0800
Modify: 2003-01-30 18:24:37.000000000 +0800
Change: 2007-12-21 17:03:53.000000000 +0800
```

3.1.41 top 命令：显示进程信息

top 命令可以动态显示当前系统中消耗资源最多的进程，如图 3-8 所示。top 命令与 ps 命令基本作用相同，都是用来显示当前的进程及其状态。但 top 能够实现动态显示，可以不断刷新当前状态，即 top 提供了对系统处理器状态的实时监测。如果在前台执行，该命令会占据整个前台，直接按“q”键即可退出。

默认情况下，该命令按照处理器占用时间、占用内存大小和进程的执行时间对进程进行排序，还可以根据用户需要，在定制文件中对其进行设定。top 命令的语法格式如下所示。

```
top [b][c][i][s][q][S] [d delay][H] [p pid] [n iterations]
```

其中各选项的说明如表 3-34 所示。

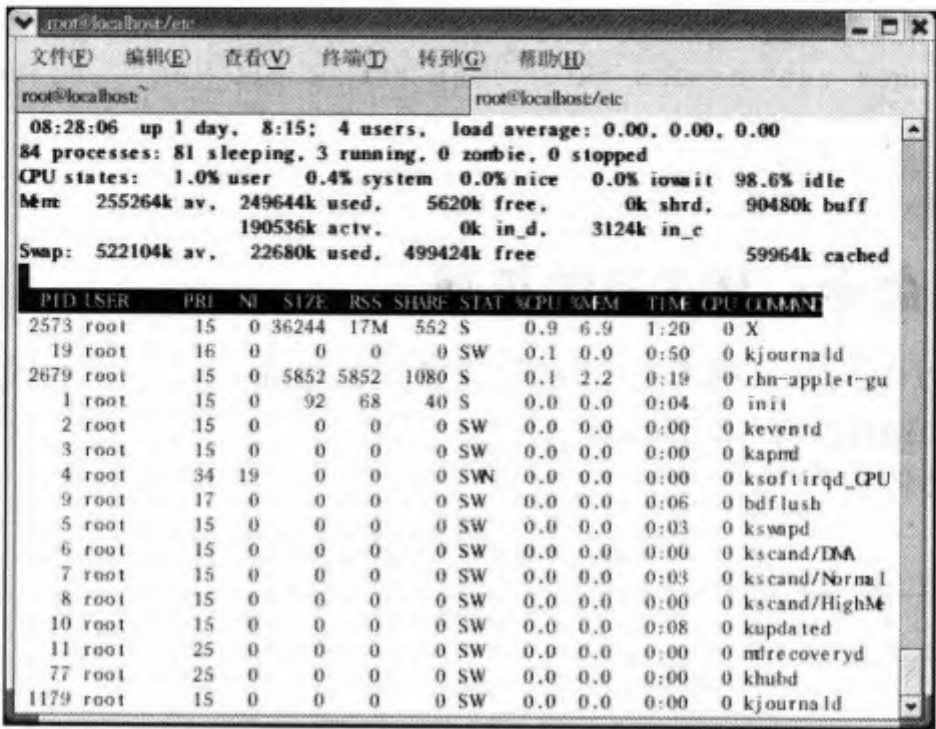


图 3-8 top 命令主界面

表 3-34 top 命令的选项说明

选 项	说 明
b	批处理模式
c	显示命令行，而不仅仅是命令名称
i	忽略空闲进程
s	保密模式
S	累积模式
H	显示所有线程
q	立即刷新
-p pid	监测指定进程
-n iterations	循环刷新的次数

例如，显示进程信息，刷新两次后退出，命令如下所示。

```
#top n 2
PID USER    PRI   NI  SIZE  RSS SHARE STAT  %CPU   %MEM    TIME CPU COMMAND
2573 root      15    0 36244  17M   552 S    0.3    6.9    1:26  0 X
2679 root      15    0 5852 5852 1080 S    0.1    2.2    0:19  0 rhn-applet-gui
  1 root      15    0   92   68   40 S    0.0    0.0    0:04  0 init
  2 root      15    0    0    0    0 SW    0.0    0.0    0:00  0 keventd
  3 root      15    0    0    0    0 SW    0.0    0.0    0:00  0 kapmd
... ..
```

3.1.42 time 命令：显示进程执行时间

time 命令可以计算执行一个进程所需要的时间。包括实际 CPU 时间、用户时间、系统时间。

例如，查看执行命令 ls 所需的时间，如下所示。



```
# time ls
cal_txt      finger3_txt groups_txt id_txt  newgrp_txt suple_txt~ w.txt
finger2_txt  finger_txt  groups_txt2 last_txt suple_txt  who.txt
real        0m0.156s
user        0m0.008s
sys         0m0.096s
```

3.1.43 tload 命令：显示系统负载

tload 命令用于以图形方式显示系统的负载信息，该命令的语法格式如下所示。

tload [-V] [-s scale] [-d delay] [tty]

其中各选项的具体说明如表 3-35 所示。

表 3-35 tload 命令的选项说明

选 项	说 明
-d<delay>	刷新的间隔时间，以秒为单位
-s<scale>	指定垂直方向的刻度，刻度越小，垂直方向的高度越大
tty	指定显示终端
-V	显示版本信息

例如，显示系统负载，执行如下所示命令。

```
# tload -d 2 //显示系统负载
```

执行该命令后，将出现图 3-9 所示的系统负载的图形表示，且每隔 2s 刷新一次。

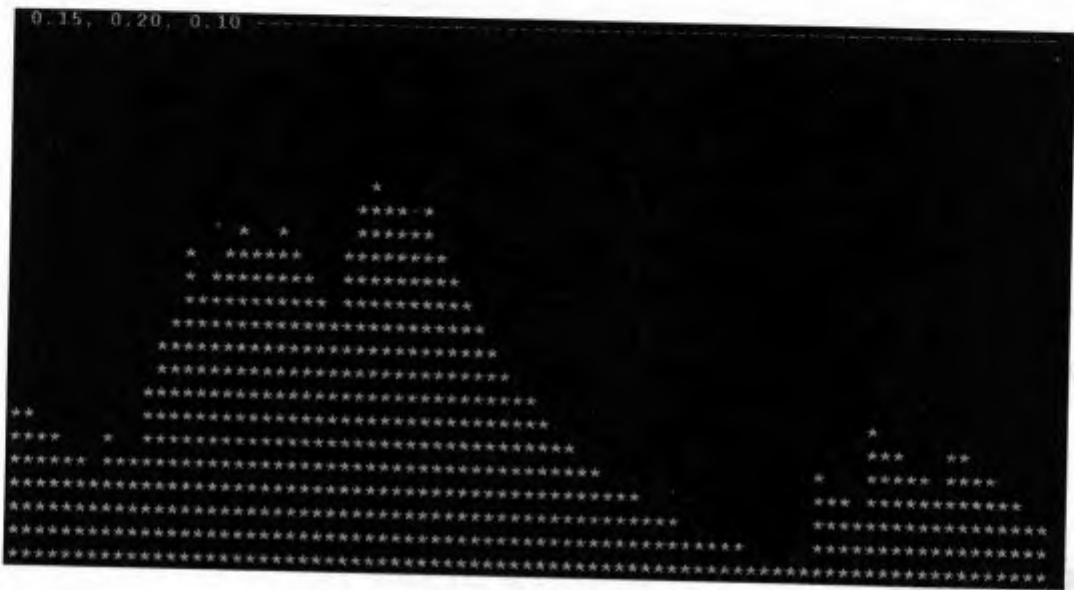


图 3-9 tload 命令显示系统负载

3.1.44 uptime 命令：显示系统的运行信息

uptime 命令用于显示系统的运行信息，包括当前系统时间、系统已经运行的时间、用户登录数以及在过去 1、5、15 分钟内的系统平均负载。uptime 命令的语法格式如下所示。

uptime [-V]

其中，选项“-V”用于显示版本信息。例如，查询系统自启动到现在总的运行时间，命令如下所示。


```
# uptime
07:53:39 up 1 day, 7:40, 4 users, load average: 0.82, 0.30, 0.11
```

可以看到，目前系统时间为 07:53:39，系统运行了 1 天 7 小时 40 分，有 4 个用户登录系统。在过去 1 分钟内的平均负载为 0.82，在过去 5 分钟内的平均负载为 0.30，在过去 15 分钟内的平均负载为 0.11。

3.1.45 uname 命令：显示操作系统信息

uname 命令用于显示操作系统信息，其语法格式如下所示。

```
uname [-a, --all] [-s, --kernel-name] [-n, --nodename] [-r, --kernel-release]
[-m, --machine] [-p,
--processor]
[-i, --hardware-platform] [-o, --operating-system] [--help] [--version]
```

uname 命令各选项的具体说明如表 3-36 所示。

表 3-36 uname 命令选项说明

选 项	说 明
-a, --all	显示全部信息（包含以下各项）
-s, --kernel-name	显示内核名称
-n, --nodename	显示网络主机名称
-r, --kernel-release	显示内核版本
-m, --machine	显示主机的硬件名称
-p, --processor	显示处理器类型
-i, --hardware-platform	显示主机的硬件平台
-o, --operating-system	显示所使用的操作系统名称
--help	帮助信息
--version	版本信息

例如，显示当前计算机的硬件平台、处理器类型，命令如下所示

```
# uname -i -p
i686 i386
```

例如，显示当前主机所有软硬件信息，命令如下所示。

```
# uname -a
Linux localhost.localdomain 2.4.20-8 #1 Thu Mar 13 17:54:28 EST 2003 i686 i686 x
```

可以看到，当前主机使用的操作系统是 Linux，内核版本为 2.4.20，主机名为 localhost.localdomain，使用的处理器为 i686。

3.1.46 vmstat 命令：显示虚拟内存的统计信息

vmstat 命令用来显示虚拟内存的统计信息，该命令的语法格式如下所示。

```
vmstat [-V] [-n] [-a] [delay [count]]
```

其中各选项的具体说明如表 3-37 所示。



表 3-37 vmstat 命令的选项说明

选 项	说 明
-a	显示所有虚拟内存的信息
delay	每隔指定时间进行刷新，以秒为单位
count	刷新的次数。如果没有指定刷新次数，并且指定了 delay，则无限次刷新
-n	字段头只显一次，而不是周期性地显示
-V	显示版本信息

例如，显示虚拟内存的使用信息，命令如下所示。

```
# vmstat //显示内存的使用信息
procs          memory      swap      io      system      cpu
r b w  swpd  free buff cache  si so  bi bo in  cs us sy id
0 0 0 30624 49756 21256 98152 0 0 6 8 104 138 0 0 99
```

例如，每隔 2s 刷新一次统计信息，且刷新 3 次，命令如下。

```
# vmstat 2 3
procs          memory      swap      io      system      cpu
r b w  swpd  free buff cache  si so  bi bo in  cs us sy id
1 0 0 30624 49756 21328 98148 0 0 6 8 104 138 0 0 99
0 0 0 30624 49756 21328 98148 0 0 0 0 104 142 0 0 100
0 0 0 30624 49756 21328 98148 0 0 0 0 103 139 0 0 100
```

3.1.47 yes 命令：连续输出给定的字符串

yes 命令用于连续输出指定的字符串，每行显示一个字符串，直到 yes 进程被杀死。该命令的语法格式如下所示。

```
yes [string][--help][--version]
```

其中各选项的具体说明如表 3-38 所示。

表 3-38 yes 命令的选项说明

选 项	说 明
string	设定重复显示的字符串
--help	显示帮助信息
--version	显示版本信息

例如，重复显示输出“hello everyone! I am Sullivan”，直到按下 Ctrl+C 组合键终止 yes 命令，命令如下所示。

```
#yes hello everyone! I am Sullivan
hello everyone! I am Sullivan
hello everyone! I am Sullivan
hello everyone! I am Sullivan
hello everyone! I am Sullivan
hello everyone! I am Sullivan
... ..
```

利用 yes 命令可以让一些交互问题自动得到应答。例如，使用“rm *”命令删除一系列文件，需要用户连续输入“yes”，命令如下所示。


```
# ls
1.txt 2.txt 3.txt 4.txt 5.txt 6.txt 7.txt 8.txt 9.txt
[root@teacher test1]# rm *
rm: 是否删除一般空文件 '1.txt' ? yes
rm: 是否删除一般空文件 '2.txt' ? yes
rm: 是否删除一般空文件 '3.txt' ? yes
rm: 是否删除一般空文件 '4.txt' ? yes
rm: 是否删除一般空文件 '5.txt' ? yes
rm: 是否删除一般空文件 '6.txt' ? yes
rm: 是否删除一般空文件 '7.txt' ? yes
rm: 是否删除一般空文件 '8.txt' ? yes
rm: 是否删除一般空文件 '9.txt' ? yes
# ls
#
```

使用“yes | rm *”命令可以自动回答 rm 命令提出的问题，无需用户再连续输入“yes”，命令如下所示。

```
# yes | rm *
rm: 是否删除一般空文件 '1.txt' ? rm: 是否删除一般空文件 '2.txt' ? rm: 是否删除一般空文件 '3.txt' ? rm:
是否删除一般空文件 '4.txt' ? rm: 是否删除一般空文件 '5.txt' ? rm: 是否删除一般空文件 '6.txt' ? rm:
是否
删除一般空文件 '7.txt' ? rm: 是否删除一般空文件 '8.txt' ? rm: 是否删除一般空文件 '9.txt' ?
# ls
#
```

3.2 常用的系统管理命令

在 Red Hat Linux 9 中，常用的系统管理命令大约有 40 多个。通过这些系统管理命令，管理员可以更为快捷地修改系统配置，高效、直接地实现系统管理。

3.2.1 apm 命令：高级电源管理

apm 命令可以进行 BIOS 高级电源管理查询，可以设置计算机的挂起、待机状态。当不输入参数直接输入 apm 命令时，系统将以用户可读的方式直接输出 /proc/apm 文件的内容。apm 命令的语法格式如下所示。

```
apm [-v][-m][-s][-S][-i][-n] [--verbose] [--minutes] [--verbose] [--suspend]
[--standby] [--version]
```

其中各选项的说明如表 3-39 所示。

表 3-39 apm 命令的选项说明

选 项	说 明
-V, --version	显示版本信息
-v, --verbose	显示 APM BIOS 版本信息及 Linux APM 驱动程序版本信息
-m, --minutes	取代 hh:mm 格式，换以分钟数的形式输出剩余的时间
-S, --standby	将设备设置为待机状态



续表

选 项	说 明
-s, --suspend	将设备设置为挂起状态
-i, --ignore	当使用交流电供电时，忽略系统产生的挂起、待机事件。该功能主要针对用户使用笔记本的情况，当使用电池供电时，应由 APM 进行管理；当使用交流电供电时，放弃其管理
-n, --noignore	即使使用交流电供电，也不允许忽略系统产生的挂起、待机事件。该项为默认设置。通常用于取消“-i”所进行的设置
BUGS	调试信息输出。该选项需要 post-1.3.57 以上内核。不兼容旧版本内核或仅对 APM 打过补丁的情况，因为新旧版本的/proc/apm 格式不同

例如，显示剩余电量，命令如下所示。

```
# apm -m //显示剩余电量
AC on-line, no system battery //由于使用的是交流电供电，显示没有使用电池
```

例如，使计算机处于挂机状态，命令如下所示。

```
#apm -s
```

3.2.2 apmd 命令：高级电源管理守护进程

apmd 命令是高级电源管理系统（apm）的守护进程，与系统内核中的 apm BIOS 驱动协同工作。当 apm 驱动通知守护进程将要来临的挂起或待机请求时，apmd 激活相应的命令并对事件、数据进行日志记录，最后通知 apm 驱动继续其操作。apmd 命令的语法格式如下所示。

```
apmd [-V, --version] [-c seconds, --check seconds] [-p percent_to_log] [-q, --quie]
[-W, --wall] [-v, --verbose]
[-w warn_percent, --warn warn_percent] [-?, --help]
```

其中各选项的具体说明如表 3-40 所示。

表 3-40 apmd 命令的选项说明

选 项	说 明
-v,--verbose	冗余模式，即记录所有的电源管理事件
-W,--wall	如果电源电量不足，向所有用户发出警告信息
-w warn_percent, --warn warn_percent	当电源不在充电状态时，如果电源电量低于 warn_percent，则发出警告信息。通常默认值为 10，如果设置的值小于 0，则关闭此项功能
-P proxy_cmd, --apmd_proxy proxy_cmd	当某些 APM 驱动事件被报告时，执行指定的命令 proxy_cmd
-q, --quie	关闭声音警告
-c seconds, --check seconds	封锁/dev/apm_bios 设备的时间，以秒为单位。通常只在有事件发生时解除封锁
-V, --version	版本信息
-, --help	帮助信息

例如，记录所有的电源管理事件，命令如下所示。

```
# apmd -v //记录所有的电源管理事件
```

例如，当电源电量低于 15%，则给所有用户发出警告信息，命令如下所示。

```
# apmd -w 15 -W
```


3.2.3 apmsleep 命令：设置挂起、待机及唤醒

有些电脑，特别是笔记本，能够利用 RTC（Real-time-clock，实时时钟）芯片将计算机从低功耗的挂起状态（挂机到 DRAM）唤醒到工作状态。apmsleep 命令用于设置唤醒时间，同时将计算机设置为挂起或待机状态，最终由来自 RTC 的中断请求唤醒计算机。

```
apmsleep [-sSnwhVd] [--suspend] [--standby] [--noapm] [--wait] [--pre-cise] [--help] [--version]
[--debug]
[+]hh:mm
```

其中各选项的具体说明如表 3-41 所示。

表 3-41 apmsleep 命令的选项说明

选 项	说 明
-s, suspend	进入挂起模式，对于笔记本来说，挂机模式将关闭除了内存外所有的程序
-S, --standby	进入待机模式，对于笔记本来说，待机模式将关闭显示器、硬盘和 CPU
-w, --wait	等待时间 hop
-p, --precise	等待唤醒时间与实际时间相匹配
-n, --noapm	不调用 APM BIOS 来挂起计算机，只是设置唤醒时钟，等待时间 hop 的到来
-d, --debug	输出正在进行的任务的一些信息
-V, version	显示版本信息
+hh:mm	指定相对当前时间的时间间隔。计算机在进入挂起或等待状态后，会在 hh 小时、mm 分钟以及若干秒后被唤醒。对于笔记本，由于定时器不一定完全准确，实际唤醒时间会有一些偏差
hh:mm	指定 24 小时进制的绝对本地时间。计算机在进入挂起或等待状态后，会在指定的绝对时间被唤醒
-V, --version	显示版本信息
-h, --help	显示帮助信息

例如，设定计算机挂机 1 小时 50 分钟，然后再被唤醒，命令如下所示。

```
# apmsleep -s +1: 50 //进入挂起状态
```

例如，设定计算机进入挂机状态，直到上午 8 点再被唤醒，命令如下所示。

```
# apmsleep -s 8: 00 //进入挂起状态
```

例如，设定计算机待机 1 小时 50 分钟，然后再被唤醒，命令如下所示。

```
# apmsleep -S +1: 50 //进入待机状态
```

例如，设定计算机进入待机状态，直到上午 9 点再被唤醒，命令如下所示。

```
# apmsleep -S 9: 00 //进入待机状态
```

3.2.4 bg 命令：将程序放在后台执行

bg 命令用来将程序放到后台执行，与命令后加“&”效果相同。该命令的语法格式如下所示。

```
bg [job_spec]
```

其中，选项“job_spec”用于指定放在后台执行的任务。例如，将任务 3 放在后台执行，执行如下所示命令。

```
# find / -name *.txt > temp.txt //从“/”开始查找，查找所有扩展名为 txt 的//文件名，保存到
//temp.txt 中
[2] Done find / -name l.txt >temp.txt
[3]+ Stopped find / -name *.txt >temp.txt //按 Ctrl+Z 暂停
```



```
# jobs                                //查看任务队列
[3]+  Stopped                        find / -name *.txt >temp.txt
# bg 3                               //将 find 命令放在后台继续执行
[3]+  find / -name *.txt >temp.txt &
# jobs,
[3]+  Running                        find / -name *.txt >temp.txt & //在后台正在执行
# cat temp.txt                       //find 命令执行完毕，查看 temp.txt 文件中的//内容
/var/www/html/disk/chap07/temp.txt
/etc/temp.txt
k/chap05/hello/1.txt
```

3.2.5 chsh 命令：改变登录系统时的 Shell

chsh 命令用来改变用户登录时使用的 Shell。如果命令行没有给定一个 Shell，那么 chsh 将提示指定一个。chsh 命令可以接受任何可执行文件路径，但如果指定的 Shell 没有在 /etc/shells 文件中列出，将给出警告。该命令的语法格式如下所示。

```
chsh [ -s,--shell ] [ -l, --list-shells ] [ -u, --help ] [ -v, --version ]
其中各选项的具体说明如表 3-42 所示。
```

表 3-42 chsh 命令的选项说明

选 项	说 明
-s, --shell	指定系统登录时使用的 Shell
-l, --list-shells	列出在/etc/shells 文件中记录的所有 Shell
-u, --help	显示帮助信息
-v, --version	显示版本信息

例如，列出当前所有可用的 Shell，命令如下所示。

```
# chsh -l                                //列出所有的 Shell
/bin/sh
/bin/bash
/sbin/nologin
/bin/bash2
/bin/ash
/bin/bsh
/bin/tcsh
/bin/csh
/bin/ksh
/bin/zsh
```

例如，当前的 Shell 为 /bin/bash，通过 chsh 命令，改变 Shell 的设置为 /bin/csh，命令如下所示。

```
# chsh                                //shell 设置
Changing shell for root.
New shell [/bin/bash]: /bin/csh       //输入新的指定的 shell
Shell changed.
```

例如，通过选项 “-s” 指定登录系统时默认的 Shell，命令如下所示。

```
# chsh -s /bin/bash                    //改变当前的 shell 设置为/bin/csh
Changing shell for root.
Shell changed.
```


3.2.6 clear 命令：清除终端屏幕

clear 命令用于清除终端屏幕。屏幕被清空后，光标回到屏幕左上角。例如，使用 clear 命令清除当前屏幕，执行该命令后屏幕如图 3-10 所示。



图 3-10 clear 命令清屏后光标回到左上角

3.2.7 depmod 命令：检查模块依赖关系

depmod 命令用来检查并显示模块之间的依赖关系。该命令的语法格式如下所示。

```
depmod      [-a, --all][ -A, --quick ] [-e, --errsyms] [-C configfile, --config
configfile] [-V, --version][ --help]
[-F kernelsyms,--filesyms kernelsyms] [-b basedirectory, --basedir basedirectory]
```

其中各选项的具体说明如表 3-43 所示。

表 3-43 depmod 命令的选项说明

选 项	说 明
-a, --all	搜索/etc/modules.conf 文件中指定目录中的模块
-A, --quick	比较文件的时间戳。如果需要，则搜索/etc/modules.conf 文件中指定目录中的模块。该选项仅在发生改变后，用于更新文件之间的依赖关系
-n, --show	在标准输出设备显示模块依赖信息，而不在保存在/lib/modules 目录中
-q, --quiet	保持静默模式，不输出错误信息
-e, --errsyms	显示每一模块中所有不能识别的字符
-s, --syslog	所有错误信息由 syslog 守护进程写入日志，而不再由标准错误输出进行显示
-b basedirectory, --basedir basedirectory	在指定的 basedirectory 目录中搜索/lib/modules 目录树的镜像
-C configfile, --config configfile	使用指定的配置文件 configfile，而不是默认的/etc/modules.conf
-F kernelsyms,--filesyms kernelsyms	使用指定的符号表
--help	显示帮助信息
-V, --version	显示版本信息

例如，显示输出模块之间的依赖关系，命令如下所示。



```
# depmod --show //显示输出模块依赖关系
/lib/modules/2.4.20-8/kernel/net/sched/sch_teql.o:

/lib/modules/2.4.20-8/kernel/net/sunrpc/sunrpc.o:

/lib/modules/2.4.20-8/kernel/net/tux/tux.o:

/lib/modules/2.4.20-8/kernel/net/wanrouter/wanrouter.o:

# module id=string
# pci module vendor device subvendor subdevice class class_mask driver_data
paep 0x00008086 0x00001200 0x0000172a 0x00000000 0x00000000 0x00000000
0x00000000
qla2200 0x00001077 0x00002200 0xffffffff 0xffffffff 0x00000000 0x00000000
0x00000000
qla2300 0x00001077 0x00002300 0xffffffff 0xffffffff 0x00000000 0x00000000
0x00000000
... ..
```

3.2.8 exec 命令：执行完命令后，交出控制权

exec 命令用来在执行完指定的命令后，交出控制权。由于控制权不会再交给原来的 Shell，所以将退出登录。例如，下面的命令行在执行完“ls -al”命令后，从系统退出，如下所示。

```
# exec ls
*# anaconda-ks.cfg install.log install.log.syslog kk showVar textbox.txt // 执
行完 ls 命令，后退出
Red Hat Linux release 9 (Shrike)
Kernel 2.4.20-8 on an i686
login: //重新登录
```

3.2.9 fc 命令：修改并执行命令

fc 命令用于修改并执行命令，可以从历史列表中选择命令并用文本编辑器对选择的命令进行编辑，并最终批处理的形式执行。fc 命令有助提高工作效率。该命令的语法格式如下所示。

fc [-e ename] [-nlr] [first] [last] or fc -s [pat=rep] [cmd]
其中各选项的具体说明如表 3-44 所示。

表 3-44 fc 命令的选项说明

选 项	说 明
-l	在标准输出设备上显示所有命令列表
-n	在列出命令列表时，不显示编号
-r	显示命令列表时，使用反向排序
-e<name>	指定使用 name 作为文本编辑器，默认是 vi
-s<pat=rep><cmd>	在执行 cmd 命令前，用 rep 代替 pat

例如，列出当前执行过的所有命令，如下所示。

```
root@localhost:~# fc -l //列出当前执行过的命令
$ fc -l
```



```

1      cd ~
2      pwd
3      ls /home
4      cd /home
5      ls
... ..

```

例如，修改并执行已经执行过的命令“pwd”及“cd /home”，命令如下所示。

```
# fc 2 4
```

fc 命令将自动打开 vi，并将命令“pwd”及“cd /home”自动写入 vi。在 vi 中添加一行命令“echo "ha,ha! It works."”，如下所示。

```

pwd
cd /home
echo "ha,ha! It works."

```

存盘并退出 vi，可以看到上述命令的执行结果，如下所示。

```

#fc 2 4
pwd
/home
cd /home
echo "ha,ha! It works."
ha,ha! It works.

```

例如，下面命令用“ll”替换历史命令中的 pwd，然后执行替换后的命令，如下所示。

```

#fc -l
34      fc -l
35      fc 20 22 23
36      pwd
37      cd home
38      cd /home
... ..
# fc -s pwd=ll pwd          //用“ll”替换历史命令中的 pwd，然后执行 pwd 命令
ll                          //实际执行的是替换后的 ll 命令
总用量 4
drwx----- 4 yang yang 4096 5月 28 21:45 yang

```

3.2.10 fg 命令：将后台任务放到前台

fg 命令可以把后台执行的任务放到前台执行。该命令的语法格式如下所示。

```
fg [job_spec]
```

其中，选项“job_spec”用于指定放在前台执行的任务。如果“job_spec”为空，fg 命令将当前默认的后台作业转到前台执行。例如，下面的命令依次将后台作业 vi 及 cat 放到前台执行。

```

# cat                                //使用 cat 命令将标准输入设备内容复制到标准输出设备上
Hello , I am Sullivan,I like Linux. //输入文本内容
Hello , I am Sullivan,I like Linux.
//按 Ctrl+Z 暂停
?1]+ Stopped                        cat
# vi &                             //在后台运行 vi 程序
[2] 14102
# jobs                              //查看当前挂起的作业
[1]- Stopped                        cat
[2]+ Stopped                        vim  //当前默认的作业
# fg                                //将默认作业 vi 放到前台执行

```



```
vim
//退出 vi
# jobs //查看当前挂起的作业
[1]+ Stopped cat
# fg 1 //将 cat 作业放到前台执行
cat
//按 Ctrl+d 结束 cat 命令
#
```

3.2.11 gcov 命令：coverage 测试工具

gcov 命令是一个 coverage 的测试工具，通常配合 gcc 使用，用于分析优化代码。gcov 命令可以给出以下几方面的测试结果。

- 每行代码被执行的频率。
- 哪些代码被实际执行。
- 代码的每一部分占用了多少 CPU 时间。

gcov 命令可以帮助用户决定对哪些代码进行优化。gcov 命令的语法格式如下所示。

```
gcov [-v, --version] [-h, --help] [-b, --branch-probabilities] [-c, --branch-counts]
[-n, --no-output]
[-l, --long-file-names] [-f, --function-summaries] [-o, --object-directory
directory] sourcefile
```

其中各选项的具体说明如表 3-45 所示。

表 3-45 gcov 命令的选项说明

选 项	说 明
-coverage	为 gcov 准备镜像
-b, --branch-probabilities	显示程序的分支频度
-c, --branch-counts	显示分支被执行次数
-f, --function-summaries	显示每一个函数的概要
-n, --no-output	不创建 gcov 输出文件
-o, --object-directory	目标文件存放目录，gcov 会在指定目录中搜索.bb、.bbg、.da 文件
Sourcefile	测试源文件
-h, --help	显示帮助信息
-v, --version	显示版本信息

例如，下面代码用于实现起泡排序。

```
$cat bubblesort.c
#include <stdio.h>
void bubbleSort( int list[], int size )
{
    int i, j, temp, flag = 1;
    while (flag)
    {
        flag = 0;
        for ( i = (size-1) ; i >= 0 ; i-- )
```



```

{
    for ( j = 1 ; j <= i ; j++ )
    {
        if ( list[j-1] > list[j] )
        {
            temp = list[j-1];
            list[j-1] = list[j];
            list[j] = temp;
            flag= 1;
        }
    }
}

int main( )
{
    int myList[10]={6, 9, 29, 7,18, 35, 4, 3, 14, 10};           //待排序的数组
    int i;
    /* 调用起泡排序算法 */
    bubbleSort(myList, 10 );
    /* 输出排序后的序列*/
    for (i = 0 ; i < 10 ; i++)
    {
        printf("%d, ", myList[i]);
    }
}

```

对 bubbleSort.c 文件进行编译，产生可执行文件 bubbleSort，命令如下所示。

```

#gcc bubblesort.c -o bubblesort -ftest-coverage -fprofile-arcs //产生可执行文件
# ls
bubbleSort          bubbleSort.bbg      bubbleSort.c.gcov
bubbleSort.bb       bubbleSort.c        bubbleSort.da

```

运行生成的 bubbleSort 可执行程序，代码如下所示。

```

# ./bubbleSort //运行 bubbleSort
3, 4, 6, 7, 9, 10, 14, 18, 29, 35,

```

例如，希望查看 bubbleSort.c 程序的分支情况，命令如下所示。

```

# gcov -b bubbleSort.c
100.00% of 17 source lines executed in file bubbleSort.c
100.00% of 12 branches executed in file bubbleSort.c
100.00% of 12 branches taken at least once in file bubbleSort.c
100.00% of 2 calls executed in file bubbleSort.c
Creating bubbleSort.c.gcov.

```

例如，希望查看 bubbleSort.c 程序中函数的使用情况，命令如下所示。

```

# gcov -f bubbleSort.c
100.00% of 11 source lines executed in function bubbleSort
100.00% of 6 source lines executed in function main
100.00% of 17 source lines executed in file bubbleSort.c
Creating bubbleSort.c.gcov.

```

3.2.12 init 命令：所有进程的父进程

init 命令是所有进程的父进程，init 命令从/etc/inittab 文件中读取脚本，然后执行这些脚本。init 命令的语法格式如下所示。



```
init [-a][-b][-z xxx ][0][1][2][3][4][5][6][-S][-s]
```

其中各选项的具体说明如表 3-46 所示。

表 3-46 init 命令选项说明

选 项	说 明
0~6	设定 init 以何种运行等级运行，其中 0 表示关机，6 表示重启系统，1 表示单用户模式，3 表示完全多用户模式
-S,s	切换到单用户模式
-b	直接进入单用户模式而不运行任何启动脚本
-a, auto	LILO 自动启动默认设置
-z XXX	设置给-z 的选项将被忽略

例如，将系统切换到运行等级 0（即关机模式），事实上，其他关闭计算机的命令也是调用 init 命令，命令如下所示。

```
#init 0 //关机
```

例如，重新启动系统，命令如下所示。

```
#init 6 //重启
```



init 命令还有一个链接命令 telinit。

3.2.13 ipcrm 命令：删除消息队列、信号量或共享内存的 ID

ipcrm 命令用来删除消息队列、信号量或共享内存段。该命令的语法格式如下所示。

```
ipcrm [-q msqid] [-m shmid] [-s semid][ -M shmkey][ -S semkey][ -Q msgkey]
```

其中各选项的具体含义如表 3-47 所示。

表 3-47 ipcrm 命令选择说明

选 项	说 明
-m shmid	删除由 shmid 标识的共享内存段
-q msqid	删除由 msqid 标识的消息队列
-Q msgkey	删除由 msgkey 创建的消息队列
-s semid	删除由 semid 标识的信号量
-S semkey	删除由 semkey 创建的信号量
-M shmkey	删除由 shmkey 指定的共享内存段

例如，使用 ipcs 显示 IPC 相关信息，命令如下所示。

```
# ipcs //显示进程间通信信息
----- Shared Memory Segments -----
key      shmid      owner      perms      bytes      nattch      status
0x00000001 32768      root      600      655360      2
----- Semaphore Arrays -----
key      semid      owner      perms      nsems
```



```
----- Message Queues -----
key      msqid      owner      perms      used-bytes      messages
例如，通过命令“ipcrm -m 32768”删除进程 32768 的共享内存，命令如下所示。
# ipcrm -m 32768
# ipcs
例如，删除进程号为 32768 的共享内存，命令如下所示。
# ipcrm -m 32768 //删除进程 32768 的共享内存
```

3.2.14 kill 命令：杀死进程

kill 命令可以发送指定的信号到相应进程或进程组。如果不指定信号，默认将向指定进程发送 SIGTERM 信号，即终止信号。若 SIGTERM 信号仍无法终止该进程，可以利用 kill 命令发送 SIGKILL (9) 信号，将指定进程强制杀死。root 用户可以利用 kill 杀死任何用户的进程，非 root 用户只能杀死自己创建的进程。kill 命令的语法格式如下所示。

```
kill [ -s signal ][ -p ] [ -a ] [ -- ] [-l signal ] [ pid ...]
其中各选项的具体说明如表 3-48 所示。
```

表 3-48 kill 命令的选项说明

选 项	说 明
-l signal	若不指定信号 signal，则-l 选项会列出全部的信号名称
-a	当处理当前进程时，不限制命令名、进程号以及用户 ID 之间的对应关系
-p	只显示指定进程的 PID，而不发送任何信号
-s signal	指定发送的信号，默认为 SIGTERM，信号可以通过名称指定，例如“-HUP”，也可以通过数字指定
pid	指定的进程 PID

通常，首先使用 ps 命令或 jobs 命令查看当前运行进程的进程号，然后使用 kill 命令杀死指定的进程。

例如，杀死 vi 进程，命令如下所示：

```
# ps -a
PID TTY      TIME CMD
15694 pts/0    00:00:00 vim           //进程号为 15694
15992 pts/2    00:00:00 man
15995 pts/2    00:00:00 sh
15996 pts/2    00:00:00 sh
16001 pts/2    00:00:00 less
16149 pts/3    00:00:00 ps
# kill 15694 //杀死进程 15694
# ps -a //vi 已被杀死
PID TTY      TIME CMD
15992 pts/2    00:00:00 man
15995 pts/2    00:00:00 sh
15996 pts/2    00:00:00 sh
16001 pts/2    00:00:00 less
16150 pts/3    00:00:00 ps
```

当使用 kill 命令不能杀死指定进程时，可以使用“-SIGKILL”信号，强制杀死相应进程。例如，下面的命令强制杀死 vi 进程。

```
# ps -a
PID TTY      TIME CMD
```



```

15992 pts/2    00:00:00 man
15995 pts/2    00:00:00 sh
15996 pts/2    00:00:00 sh
16001 pts/2    00:00:00 less
16152 pts/3    00:00:00 vim          //进程号为 16152
16154 pts/3    00:00:00 ps
# kill 16152          //杀死进程 16152
# ps -a
  PID TTY          TIME CMD
15992 pts/2    00:00:00 man
15995 pts/2    00:00:00 sh
15996 pts/2    00:00:00 sh
16001 pts/2    00:00:00 less
16152 pts/3    00:00:00 vim          //未能杀死 vi
16162 pts/3    00:00:00 ps
# kill -SIGKILL 16152 //强制杀死 vi
# ps -a
  PID TTY          TIME CMD
15992 pts/2    00:00:00 man
15995 pts/2    00:00:00 sh
15996 pts/2    00:00:00 sh
16001 pts/2    00:00:00 less
16163 pts/3    00:00:00 ps
[1]+  已杀死                vim PP

```



上述命令将强制杀死 vi 进程,但可能引起系统中被进程占用的某些资源不能被正确释放。

例如,使用 kill 命令列出所有可以使用的信号,如下所示。

```

# kill -l
 1) SIGHUP          2) SIGINT          3) SIGQUIT         4) SIGILL
 5) SIGTRAP         6) SIGABRT        7) SIGBUS          8) SIGFPE
 9) SIGKILL        10) SIGUSR1       11) SIGSEGV        12) SIGUSR2
13) SIGPIPE        14) SIGALRM       15) SIGTERM        17) SIGCHLD
18) SIGCONT        19) SIGSTOP       20) SIGTSTP        21) SIGTTIN
22) SIGTTOU        23) SIGURG        24) SIGXCPU        25) SIGXFSZ
26) SIGVTALRM      27) SIGPROF       28) SIGWINCH       29) SIGIO
30) SIGPWR         31) SIGSYS        33) SIGRTMIN       34) SIGRTMIN+1
35) SIGRTMIN+2     36) SIGRTMIN+3    37) SIGRTMIN+4     38) SIGRTMIN+5
39) SIGRTMIN+6     40) SIGRTMIN+7    41) SIGRTMIN+8     42) SIGRTMIN+9
43) SIGRTMIN+10    44) SIGRTMIN+11   45) SIGRTMIN+12    46) SIGRTMIN+13
47) SIGRTMIN+14    48) SIGRTMIN+15   49) SIGRTMAX-14    50) SIGRTMAX-13
51) SIGRTMAX-12    52) SIGRTMAX-11   53) SIGRTMAX-10    54) SIGRTMAX-9
55) SIGRTMAX-8     56) SIGRTMAX-7    57) SIGRTMAX-6     58) SIGRTMAX-5
59) SIGRTMAX-4     60) SIGRTMAX-3    61) SIGRTMAX-2     62) SIGRTMAX-1
63) SIGRTMAX

```

例如,发送 SIGHUP 信号(进程重启)给进程 23912,可以使用下面的命令。

```
$ kill -SIGHUP 23912
```

3.2.15 killall 命令: 杀死同名的所有进程

killall 命令将根据指定的名称向相关的所有进程发送信号。如果不指定信号,那么默认

的信号是 SIGTERM。信号可以通过名称指定，如“-HUP”，也可以通过数字指定。如果 killall 命令至少杀死了一个指定的进程，那么它将返回 0 值，否则，返回非 0 值。killall 进程不会杀死自己，但可以杀死别的 killall 进程。killall 命令的语法格式如下所示。

```
killall [-e,--exact] [-g,--process-group] [-i,--interactive] [-q,--quiet]
[-v,--verbose] [-w,--wait] [-V,--version]
[-s,--signal signal] [-l,--list] [-V,--version] [--help]
```

其中各选项的具体说明如表 3-49 所示。

表 3-49 killall 命令的选项说明

选 项	说 明
-e,--exact	要求精确匹配进程的全名。通常如果进程名称长于 15 个字符，超出 15 个字符的部分将被忽略，在这种情况下所有匹配前 15 个字符的进程都将被杀死
-g,--process-group	杀死进程所属的进程组。该选项将向进程所属的进程组发送 kill 信号，该信号每组只发送一次，即使该进程组有多个进程，也只会收到一个信号
-i,--interactive	交互模式，杀死进程前先询问用户
-l,--list	列出所有已知的信号名称
-q,--quiet	静默模式，不输出警告信息
-s,--signal	发送指定的信号
-v,--verbose	报告信号是否成功发送
-w,--wait	等待进程死亡
-V,--version	显示版本信息
--help	显示帮助信息

例如，下面的命令杀死与 vim 同名的所有进程。

```
# vi & //在后台运行 vi
[1] 16230
# vi & //在后台再次运行 vi
[2] 16231
[1]+ Stopped vim
# ps //显示所有的进程
PID TTY TIME CMD
16040 pts/3 00:00:00 bash
16230 pts/3 00:00:00 vim
16231 pts/3 00:00:00 vim
16232 pts/3 00:00:00 ps
# killall -9 vim //强制杀死所有名称为 vim 的进程
[1]- 已杀死 vim
[2]+ 已杀死 vim
# ps -a //已杀死
PID TTY TIME CMD
15992 pts/2 00:00:00 man
15995 pts/2 00:00:00 sh
15996 pts/2 00:00:00 sh
16001 pts/2 00:00:00 less
16235 pts/3 00:00:00 ps
```

例如，显示所有可用的信号名称，命令如下所示。



```
# killall -l
HUP INT QUIT ILL TRAP ABRT IOT BUS FPE KILL USR1 SEGV USR2 PIPE ALRM TERM STKFLT CHLD
CONT STOP TSTP TTIN TTOU URG XCPU XFSZ VTALRM PROF WINCH IO PWR SYS UNUSED
```

3.2.16 logrotate 命令：处理 log 文件

logrotate 命令用来管理系统的 log 文件。logrotate 命令可以替换、压缩、删除或通过邮件发送系统日志。系统日志可以根据需要每天处理一次，也可以每星期、每月处理一次，或者当日志文件过大时再对其进行处理。通常 logrotae 每天只处理一次，除非指定日志文件过大时才进行处理或使用了“-f,-force”选项。该命令的语法格式如下所示。

```
logrotate [-d, --debug] [-f, --force] [-m, --mail=command] [-s, --state=statefile] [-v,
--verbose] [ --usage]
[ -?, --help] [configfile]
```

其中各选项的具体说明如表 3-50 所示。

表 3-50 logrotate 命令的选项说明

选 项	说 明
-d, --debug	显示调试信息
-f, --force	强制对日志文件进行处理，即使 logrotae 命令认为没有必要进行处理。在往配置文件中添加了新的选项，或者手工删除了某些日志文件后，通常需要启动 logrotae 命令，产生新的日志文件
-m, --mail=command	发送电子邮件
-s, --state=statefile	指定状态文件的路径
-v, --verbose	显示详细的处理过程
--usage	显示用法摘要
-?, --help	显示帮助信息
configfile	指定配置文件

例如，指定 logrotate 命令的配置文件，命令如下所示。

```
# cat logrotate.conf //查看 logrotate.conf 配置文件
# see "man logrotate" for details
# rotate log files weekly
weekly

# keep 4 weeks worth of backlogs
rotate 4

# create new (empty) log files after rotating old ones
create

# uncomment this if you want your log files compressed //设定是否进行压缩
#compress

# RPM packages drop log rotation information into this directory
include /etc/logrotate.d

# no packages own wtmp -- we'll rotate them here
/var/log/wtmp {
    monthly
```



```
create 0664 root utmp
rotate 1
}
# system-specific logs may be also be configured here.
#logrotate /etc/logrotate.conf //对配置文件进行指定
```

3.2.17 mtools: 软盘操作命令集

在 Red Hat Linux 9 中，为了方便用户使用软盘，专门提供了 mtools 工具集。mtools 工具集由一组以字母“m”开头的命令组成，其中绝大多数命令格式与对应的 MS-DOS 命令兼容。用户只需要在相应的 MS-DOS 命令前加“m”，就可以像在 MS-DOS 中一样操作软盘，而无需再对软盘进行挂载和卸载。mtools 命令集只能对 FAT 格式的软盘进行操作。mtools 工具集包括的命令如下所示。

```
# mtools
Supported commands: //包括的命令
mattrib, mbadblocks, mcat, mcd, mclasserase, mcopy, mdel, mdeltree
mdir, mdoctorfat, mdu, mformat, minfo, mlabel, mmd, mmount
mpartition, mrd, mread, mmove, mren, mshowfat, mtoolstest, mtype
mwrite, mzip
```

表 3-51 列出了对 mtools 工具集中命令的说明。

表 3-51 mtools 工具集及说明

mtools 命令	对应的 DOS 命令	说 明
mdir	dir	列目录
mcopy	copy	复制文件
mdel	del	删除文件
mmd	md	创建目录
mformat	format	磁盘格式化
mtype	type	显示文件内容
mrd	rd	删除目录，只能删除空目录，不能删除文件
mmove	move	移动文件
mren	ren	重命名
mattrib	attrib	显示属性
mcomp	comp	对两个文件进行比较
mcd	cd	改变当前所在目录
mbadblock	chkdsk	软盘检测，并在 FAT 表中对坏块进行标识
mshowfat	showfat	显示软盘中分配给指定文件的簇
mtoolstest	toolstest	测试 mtools 设置
mlable	lable	设置卷标
mdeltree	deltree	删除目录树，连同目录下的子目录及文件一并删除
mmount		挂载软盘



续表

mttools 命令	对应的 DOS 命令	说 明
mdu		显示文件目录所占空间
mttools		显示 mttools 命令集
mzip		改变 Zip/Jaz 驱动器保护模式，弹出磁盘
mcheck		用于检测指定软盘中的文件是否正确

例如，列出软盘的目录，命令如下所示。

```
#mdir a:
Volume in drive A has no label
Volume Serial Number is 3D91-2134
Directory for A:/
passwd bak 223 08-26-2007 21:55
shadow      255 08-26-2007 21:55
  2files 478 bytes
  1 460 459 bytes free
```

例如，将上例中的 shadow 文件，复制到新创建的目录 private 下，命令如下所示。

```
#mmd a: private //创建目录 private
#mcopy a:shadow a:private/shadow //复制
```

例如，删除软盘中 passwd.bak 文件，命令如下所示。

```
#mdel a: passwd.bak
```

3.2.18 nice 命令：设定进程优先级

在 Linux 系统中，进程有运行、就绪和阻塞三种状态，进程会在三种状态间切换。每个正被执行的进程都会被赋予一定的使用 CPU 的优先级。系统会按着优先级顺序对进程进行调度。等级越高的进程在执行时就会有越多获得使用 CPU 的机会，其大部分时间都会处于运行态，总的执行时间就会缩短。与之相反，等级较低的进程就需要较长时间等待 CPU，因此总的执行时间就会较长。Red Hat Linux 中提供了 nice 命令，可以根据用户的需要设置进程执行的优先级。

nice 的命令的语法格式如下所示。

```
nice -N 命令
```

N 为-20~19 之间的整数，表示进程执行时的优先等级。-20 表示最高等级，19 代表最低等级。其中-20~-1 之间的等级只有系统管理员可以设置。如果没有使用 nice 命令对进程执行时的优先级进行设定，系统默认等级为 0；如果使用了 nice 命令，但没有指定等级，则系统默认为 10。表 3-52 是 nice 命令的使用范例及说明。

表 3-52 nice 命令使用范例及说明

命 令	优先级说明
vi test1 &	没有使用 nice 命令，因而系统默认赋予其优先级为 0
nice vi test2 &	使用 nice 命令，但没有指定优先级，系统默认赋予其优先级为 10
nice -25 vi test3 &	由于用 nice 命令指定的优先级 25，超出系统最大值，系统会以最低等级 19 运行
nice --25 vi test4 &	由于用 nice 命令指定的优先级-25，超出系统最小值，系统会以最高等级-20 运行
nice -15 vi test5 &	使用 nice 命令，并指定优先级为 15，所以该进程的等级为 15
nice --15 vi test6 &	使用 nice 命令，并指定优先级为-15，所以该进程的等级为-15

可以通过查看进程的优先级来验证上述结论，如下所示。

```
# ps -l | grep vi
0 T      0 6087 3796 0 77      0 - 1239 finish pts/1    00:00:00 vi
0 T      0 6088 3796 0 87      10 - 1239 finish pts/1    00:00:00 vi
0 T      0 6124 3796 0 97      19 - 1239 finish pts/1    00:00:00 vi
4 T      0 6125 3796 0 60     -20 - 1240 finish pts/1    00:00:00 vi
0 T      0 6128 3796 0 90      15 - 1240 finish pts/1    00:00:00 vi
4 T      0 6129 3796 0 62     -15 - 1239 finish pts/1    00:00:00 vi
```

3.2.19 nohup 命令：退出系统继续执行命令

nohup 命令允许用户在退出登录以后仍然可以继续执行命令。该命令的语法格式如下所示。

```
nohup [--help][--version][command]
```

其中各选项的具体说明如表 3-53 所示。

表 3-53 nohup 命令的选项说明

选 项	说 明
command	设置指定的命令在用户退出登录后仍继续执行
--help	显示帮助信息
--version	显示版本信息

例如，设定退出系统后继续在后台执行 ftp 下载任务，如下所示。

```
# nohup ftp get -u user -p 123456 192.168.20.127 /home/teacher/homework.c & // 退出系统后继续执行命令
[1] 2840
nohup: appending output to 'nohup.out'
#
```

通过命令 nohup，可以使得用户在退出系统之后仍然可以从指定的 ftp 服务器（192.168.20.127）上下载文件（/home/teacher/homework.c）。

3.2.20 ntsysv 命令：设置系统的各种服务

ntsysv 命令用来设置系统的各种服务，执行该命令后将进入图形化界面，其功能与 chkconfig 命令类似，但后者不提供图形界面。该命令的语法格式如下所示。

```
ntsysv [--back] [--level <levels>]
```

其中各选项的具体说明如表 3-54 所示。

表 3-54 ntsysv 命令的选项说明

选 项	说 明
-back	在互动界面中显示【返回】按钮，默认为【取消】按钮
--level <levels>	设置指定等级的系统服务，level 必须在 0~6 之间

例如，通过命令 ntsysv 设置系统开机时启动的各种服务。在命令行输入以下内容。

```
#ntsysv
```

回车后系统进入如图 3-11 所示图形界面，然后可以通过 UP/DOWN、Tab、SPACE 进行



相应的设置。



图 3-11 ntsysv 命令窗口

例如，在 ntsysv 界面中设置显示【返回】按钮，执行如下所示命令。

```
#ntsysv --back
```

回车后系统进入图 3-12 所示的图形界面，其中原界面中的【取消】按钮已被【返回】按钮取代。



图 3-12 具有【返回】按钮的 ntsysv 命令窗口

3.2.21 open 命令：开启虚拟终端

open 命令用来开启虚拟终端，然后执行指定的命令。该命令的语法格式如下所示。

```
open [-c vtnumber] [-l] [-u] [-s] [-v] [-w] [--help] [-- command_line]
```

其中各选项的具体说明如表 3-55 所示。

表 3-55 open 命令的选项说明

选 项	说 明
-l	指定登录所使用的 Shell
-s	执行命令时，切换到新的虚拟终端，命令将在新的虚拟终端执行
-u	查看当前终端的登录用户，然后以相同的用户名登录到新终端

续表

选 项	说 明
-v	运行时显示详细的处理过程
-w	等待命令结束。如果-w 与-s 选项同时使用，open 会在命令结束以后切换回原来的终端
-- command_line	--是 open 命令的选项与所执行命令的分隔符
-c vtnumber	指定打开的终端

例如，在虚拟终端 6 执行“ls-al”命令，同时切换到虚拟终端 6，命令如下所示。

```
#open -s -c 6 -- ls -al //在终端 6 执行 ls -al 指令
```

例如，开启新的终端 tty5 执行命令 tcpdump，并等待命令执行完毕后返回。命令如下所示。

```
# open -c 5 -sw -- tcpdump //开启终端 5 执行 tcpdump 指令
```

3.2.22 pgrep 命令：查找符合条件的进程

pgrep 命令会在当前所有进程中查找符合条件的进程，并将进程 ID 显示在标准输出设备上。该命令的语法格式如下所示。

```
pgrep [-f][-l][-n][-v][-x] [-d DELIM] [-P PPIDLIST] [-g PGRPLIST] [-s SIDLIST] [-u EUIDLIST] [-U UIDLIST] [-G GIDLIST] [-t TERMLIST] [PATTERN]
```

其中各选项的具体说明如表 3-56 所示。

表 3-56 pgrep 命令的选项说明

选 项	说 明
-d DELIM	设置显示在每个进程 ID 之间的分隔符
-f	默认情况下，pgrep 只搜索进程名称符合指定条件的进程，使用-f 选项指定 pgrep 命令以完整的命令行进行搜索
-l	列出进程名称的同时列出进程 ID
-n	只显示最近的一个符合条件的进程
-v	搜索与条件不符的进程
-x	进行全字匹配，即搜索进程名或命令行（如果指定选项-f）与条件严格符合的进程
-u EUIDLIST	搜索在 EUIDLIST 中列出的有效用户的进程，数字或字符串均可作为条件
-P PPIDLIST	搜索匹配父进程为指定进程的进程
-t TERMLIST	搜索由 TERMLIST 列出的终端下的所有进程
-U UIDLIST	搜索匹配由 UIDLIST 列出的指定用户的进程，数字或字符串均可作为条件
-G GIDLIST	搜索匹配由 GIDLIST 列出的指定组的进程，数字或字符串均可作为条件
PATTERN	匹配条件

例如，查找由 root 用户创建的进程，命令如下所示。

```
# pgrep -u root -l
1 init
2 keventd
3 kapmd
4 ksoftirqd_CPU0
9 bdflush
5 kswapd
... ..
```



例如，查找不是由 root 用户创建的进程，命令如下所示。

```
# pgrep -u root -l -v
2084 portmap
2103 rpc.statd
2268 mysqld
2269 mysqld
2270 mysqld
2271 mysqld
2272 mysqld
... ..
```

例如，查找与字符串 “nslookup” 相关的进程，命令如下所示。

```
# pgrep nslookup //显示与指定字符串相关的进程
16867
16869
16870
16871
16874
16876
... ..
```

例如，列出父进程为 init 进程（进程 ID 为 1）的所有进程，命令如下所示。

```
# pgrep -P 1 //列出父进程为 1 的所有进程
2
3
4
9
5
6
... ..
```

例如，列出父进程不为 init 的所有进程，命令如下所示。

```
# pgrep -v -P 1 //反向选择
1
2268
2269
2270
2271
2272
```

3.2.23 pidof 命令：显示当前正在运行程序的进程 ID

pidof 命令可以显示当前正在执行程序进程的 ID。该命令的语法格式如下所示。

```
pidof [-s] [-x] [-c] [-o omitpid] [-o omitpid..] [program..]
```

其中各选项的具体说明如表 3-57 所示。

表 3-57 pidof 命令的选项说明

选 项	说 明
-s	单次选中，只返回一个 PID
-c	只返回进程 ID
-x	同时返回命令运行的脚本
-o omitpid	忽略由 omitpid 指定的进程号
program	程序名称

例如，通过命令“pidof mingetty”，显示与“mingetty”相关的进程号，如下所示。

```
# pidof mingetty
2535 2534 2533 2532 2531 2530
```

例如，显示与“mingetty”相关的进程号，但忽略 PID 为 2535 的进程，命令如下所示。

```
# pidof mingetty -o 2535
2534 2533 2532 2531 2530
```

例如，只显示与“mingetty”相关的进程的首次匹配，命令如下所示。

```
# pidof mingetty -s //只返回首次匹配的进程号
2333
```

3.2.24 pkill 命令：杀死进程

pkill 命令是 ps 命令和 kill 命令的结合。pkill 命令会在当前所有进程中查找符合条件的进程，然后向这些进程发送 SIGTERM 信号，用来杀死指定的进程。其语法格式如下所示。

```
pkill [-signal] [-f][n][v][x] [-P ppid] [-g pgrp][--s sid] [-u euid] [--U uid] [--G gid]
[-t term] [pattern]
```

其中各选项的具体说明如表 3-58 所示。

表 3-58 pkill 命令的选项说明

选 项	说 明
-f	默认情况下，只搜索进程名称符合指定条件的进程，使用-f 选项后将以完整的命令行进行搜索
-n	只杀死最近的一个符合条件的进程
-v	搜索与条件不符的进程
-x	进行全字匹配，即搜索进程名或命令行（如果指定选项-f）与条件严格符合的进程
-s sid	搜索由 sid 列出的会话下的所有进程
-u euid	搜索在 euid 中列出的有效用户的进程，数字或字符串均可作为条件
-P ppid	搜索匹配父进程为指定进程的进程
-t term	搜索由 term 列出的终端下的所有进程
-U uid	搜索匹配由 uid 列出的指定用户的进程，数字或字符串均可作为条件
-G gid	搜索匹配由 gid 列出的指定组的进程，数字或字符串均可作为条件
pattern	匹配条件
-signal	指定发送的信号

例如杀死 vi 进程，命令如下所示。

```
# ps //显示当前的进程
PID TTY TIME CMD
16514 pts/0 00:00:00 bash
17620 pts/0 00:00:00 vim
17685 pts/0 00:00:00 vim
17687 pts/0 00:00:00 cat
17688 pts/0 00:00:00 ps
# pkill -9 vim //强制杀死进程 vim
```



```
[1] 已杀死      vim
[2]- 已杀死      vim
```

例如杀死终端 `tty1` 下的所有进程，命令如下所示。

```
# pkill -t tty1 //杀死所有终端 1 下的所有进程
```

例如，杀死指定用户为“teacher”的所有进程，命令如下所示。

```
# pkill -u teacher //杀死用户 teacher 的所有进程
```

例如，杀死不是由用户 `teahcer` 创建的进程，命令如下所示。

```
# pkill -vu teacher //杀死不属于用户 teacher 的所有进程
```

3.2.25 pwck 命令：检查密码文件

`pwck` 命令可以检测密码文件的正确性，其语法格式如下所示。

```
pwck [-r] [-s] [passwd [shadow]]
```

其中各选项的具体说明如表 3-59 所示。

表 3-59 pwck 命令的选项说明

选 项	说 明
-r	只读模式
-s	依据 UID 对/etc/passwd 或/etc/shadow 排序
Passwd	passwd 文件
Shadow	shadow 文件
--help	显示帮助信息

例如，检测 `/etc/passwd` 文件，查看其正确性，命令如下所示。

```
# pwck /etc/passwd //检测 passwd 文件
user adm: directory /var/adm does not exist
user gopher: directory /var/gopher does not exist
user wnn: directory /home/wnn does not exist
pwck: no changes
```

例如，检测 `/etc/shadow` 文件，查看其正确性，命令如下所示。

```
# pwck /etc/shadow //检测 shadow 文件
user root: directory 7 does not exist
user bin: directory 7 does not exist
user daemon: directory 7 does not exist
user adm: directory 7 does not exist
user lp: directory 7 does not exist
user sync: directory 7 does not exist
user shutdown: directory 7 does not exist
... ..
```

3.2.26 rlogin 命令：远端登录

`rlogin` 命令连接当前本地主机系统到远端的主机系统。`rlogin` 命令使用标准的 Berkeley `rhosts` 身份认证机制，该命令的语法格式如下所示。

```
rlogin [-ec] [-7] [-8] [-c] [-a] [-f] [-F] [-E] [-t termttype] [-n] [-d] [-k] [-x] [-L] [-l]
```

其中各选项的具体说明如表 3-60 所示。

表 3-60 rlogin 命令的选项说明

选 项	说 明
-E	忽略 escape 字符
-8	只识别 8 位的字符，否则 rlogin 将会忽略奇偶校验位
-L	允许 rlogin 会话运行在 litout 模式
-e c	设置 escape 字符为 c
-c	断开链接前要求确认
-a	通过发送一个空的本地用户名，强制要求远端主机发送密码请求
-f	向远端主机发送一个本地的认证副本
-F	向远端主机发送一个可转寄的本地认证的副本
-7	强制执行 7 位的传输
-d	打开用于与远端主机通信的 TCP 套接口的调试
-k	要求包含远端主机的 tickets 信息
-x	启动数据传输的 DES 加密机制
-t termtype	以 termtype 替换终端类型
-l	指定登录用户

首先在远端主机上开启 rlogin 服务，执行如下所示命令。

```
# chkconfig rlogin on //开启 rlogin 服务
```

使用 rlogin 命令登录远端主机 192.168.20.122，命令如下所示。

```
# rlogin 192.168.20.122 //登录主机 192.168.20.122
Password:
Last login: Tue Jun 3 06:14:31 on tty1
You have new mail.
```

例如，以指定用户名 (teacher) 登录远端主机 (192.168.20.122)，命令如下所示。

```
# rlogin 192.168.20.122 -l teacher //登录远端主机
Password:
Last login: Mon May 28 15:30:25 from 192.168.20.19
```

3.2.27 renice 命令：对进程优先级进行调整

nice 命令用于设置优先级并执行相应的程序。如果一个程序已经开始执行，可以使用 renice 命令对优先级进行调整。renice 命令格式如下。

```
renice N [[进程号] [-u 用户名] [-g 用户组]]
```

其中 N 为-20~19 之间的整数，表示进程的优先等级，[进程号]指进程的 PID 值，[-u]或[-g]用于改变用户或用户组中所有进程执行的优先级。



renice 使用进程号来对进程进行指定，而 nice 命令使用程序名。renice 命令的参数 N 前不需要加上“-”符号，而 nice 命令的参数 N 前需要加上“-”符号。

例如，修改 vi 进程的优先级，命令行如下。



```
# ps -l | grep vi
0 T      0 6087 3796 0 77  0 - 1239 finish pts/1    00:00:00 vi
# renice -10 6087
6087: old priority 0, new priority -10
```

如果重新修改的等级超出最小值，则该进程的优先级取最高等级-20，命令如下所示。

```
# renice -25 6087
6087: old priority -10, new priority -20
```

如果重新修改的等级超出最大值，则该进程的优先级取最低等级 19，命令如下所示。

```
# renice 25 6087
6087: old priority -20, new priority 19
```

例如，修改用户 teacher1 和 teacher2 所执行进程的优先级为-10，命令如下所示。

```
# renice -10 -u teacher1 teacher2
500: old priority -10, new priority -10
505: old priority 0, new priority -10
```

如果修改 teacher 用户组中所有用户执行的进程的优先级为-5，命令如下所示。

```
# renice -5 -g teacher
0: old priority -10, new priority -5
```

3.2.28 rmmod 命令：删除模块

rmmod 命令用于卸载可加载模块。该命令从当前内核中卸载可加载的模块，但这些模块必须没有被使用。rmmod 命令的语法格式如下所示。

```
rmmod [-a, --all][ -e, --persist][ -r, --stacks][ -s, --syslog][ -v, --verbos][ -h, --help][ -V, --version]
```

其中各选项的具体说明如表 3-61 所示。

表 3-61 rmmod 命令的选项说明

选 项	说 明
-a, --all	删除所有没有使用的模块
-e, --persist	保存数据，不卸载模块
-r, --stacks	从命名的模块开始删除堆栈
-s, --syslog	使用 syslog 记录错误信息
-v, --verbos	运行时，显示详细的处理信息
-h, --help	显示帮助信息
-V, --version	显示版本信息

例如，通过命令 lsmod，显示当前系统下已经安装的模块，如下所示。

```
# lsmod //显示已经安装的模块
Module          Size Used by Not tainted
nls_iso8859-1    3516  0 (autoclean)
ide-cd           35708  0 (autoclean)
cdrom            33728  0 (autoclean) [ide-cd]
parport_pc       19076  1 (autoclean)
mousedev         5492   0
input            5856   0 [keybdev mousedev hid]
usb-uhci          26348  0 (unused) //未使用
sd_mod           13452   6
```



```
scsi_mod          107128  2  [BusLogic sd_mod]
... ..
```

例如，卸载模块“usb-uhci”，命令如下所示。

```
# rmmod -v usb-uhci                                     //卸载模块
Checking usb-uhci for persistent data
# lsmod                                                    //显示已经安装的模块
Module              Size Used by  Not tainted
nls_iso8859-1       3516  0  (autoclean)
ide-cd              35708  0  (autoclean)
cdrom               33728  0  (autoclean) [ide-cd]
parport_pc         19076  1  (autoclean)
mousedev            5492  0
input              5856  0  [keybdev mousedev hid]
sd_mod              13452  6
scsi_mod            107128  2  [BusLogic sd_mod]
... ..
```

3.2.29 rsh 命令：远端登录的 Shell

rsh 命令向远程用户提供登录 Shell，其语法格式如下所示。

```
rsh host [-l username] [-n] [-d] [-F] [-x] [host][command]
```

其中各选项的具体说明如表 3-62 所示。

表 3-62 rsh 命令的选项说明

选 项	说 明
-d	在远端服务器的 TCP 端口上打开端口调试模式
-n	将输入的命令从/dev/null 重新定向
-l username	默认情况下，登录远端服务器所使用的用户名和本地用户名一致，该选项允许用户以其他的用户名登录远端服务器
-F	向远端主机发送一个可转寄的本地认证的副本
-x	网络通信被加密
host	登录的远端服务器
command	在远端服务器执行的命令

首先在远端服务主机上开启 rsh 服务，执行如下所示命令。

```
# chkconfig rsh on
```

例如，登录到远程主机 192.168.20.122，命令如下所示。

```
# rsh 192.168.20.122
Password:
Last login: Tue Jun 3 08:31:30 from 192.168.20.122
You have new mail.
#
```

例如，通过命令“rsh -l yang 192.168.20.122 /bin/ls”，以账号 yang 登录到远程主机 192.168.20.122，执行命令“/bin/ls”，如下所示。

```
# rsh -l yang 192.168.20.122 /bin/ls //远程命令执行
```

3.2.30 screen 命令：多重视窗管理程序

screen 命令用来管理视窗程序，是一个全屏的视窗管理工具，可以在一个视窗中同时运



行多个进程。screen 命令的语法格式如下所示。

```
screen[-a][-A][-m][-R][-ls,--list][-wipe][-h<num>][-r<pid.tty.host>][-s<shell>][-v][  
-d,pid.tty.host>][-c <file>]
```

其中各选项的具体说明如表 3-63 所示。

表 3-63 screen 命令的选项说明

选 项	说 明
-a	包括所有的功能
-A	所有 screen 调整为当前 screen 的大小
-m	强制建立新的 screen
-R	若 screen 已在运行则刷新，若没在运行，则进行创建
-ls, --list	显示当前的 screen 列表
-wipe	显示当前的 screen 列表，但会进行检查，删除失效的 screen
-h<num>	设置历史回滚缓冲区行数
-r<pid.tty.host>	恢复指定的作业
-s<shell>	指定默认的 Shell
-v	显示版本信息
-d<pid.tty.host>	让指定的 screen 离线
-c< file>	用指定文件重载默认的配置文件的\$HOME/.screenrc

screen 命令可以在一个终端上模拟多个终端。下面使用 screen 命令创建一个新的 screen，如下所示。

```
#screen
```

执行该命令后，出现如图 3-13 所示的 screen。



图 3-13 创建的 screen0

再次使用 screen 命令或者按 Ctrl+a+c，可以再创建一个新的 screen，如图 3-14 所示。

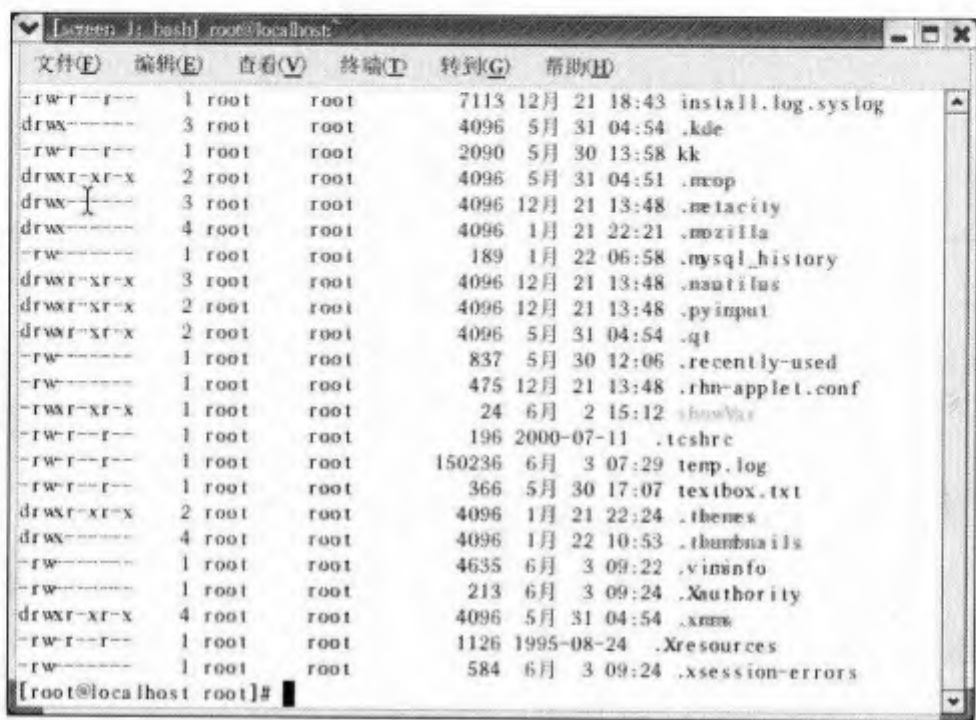


图 3-14 创建的 screen1

对于创建的多个 screen，可以使用 Ctrl+a 进行 screen 之间的切换。通过屏幕左上角可以看出 screen 的名称。

例如，显示当前的 screen 列表，命令如下所示。

```
# screen -ls //显示当前所有的 screen 列表
# screen -ls
There are screens on:
    2733.pts-0.localhost (Attached)
    2849.pts-3.localhost (Attached)
2 Sockets in /tmp/screens/S-root.
```

例如，通过命令“screen -d”让指定的 screen 离线，如下所示。

```
# screen -d 2733.pts-0.localhost //让 screen 作业离线
[2733.pts-0.localhost detached.]

# screen -ls
There are screens on:
    2733.pts-0.localhost (Detached) //显示 screen 作业
    2849.pts-3.localhost (Attached)
2 Sockets in /tmp/screens/S-root.
```

例如，通过命令“screen -r”恢复指定的作业，如下所示。

```
# screen -r 2733.pts-0.localhost //恢复指定的 screen
# screen -ls //显示 screen 作业
There are screens on:
    2733.pts-0.localhost (Attached)
    2849.pts-3.localhost (Attached)
2 Sockets in /tmp/screens/S-root.
```

3.2.31 sleep 命令：休眠

sleep 命令可以使系统进入休眠状态，即暂停一段时间。该命令的语法格式如下所示。

```
sleep [--help][--version][time]
```

其中各选项的具体说明如表 3-64 所示。



表 3-64 sleep 命令的选项说明

选 项	说 明
time	休眠的时间，单位为 s（秒），m（分），h（小时），d（天）。例如，3s 表示 3 秒
--help	显示帮助信息
--version	显示版本信息

例如，通过命令 sleep 让系统进入休眠状态，时间为 2 分钟，如下所示。

```
# sleep 2m //休眠 2 分钟
```

3.2.32 sudo 命令：以其他身份来执行命令

sudo 命令可以以系统其他用户的身份执行指定的命令，只有在/etc/sudoers 文件中允许的用户才可以执行 sudo 命令。在执行 sudo 时，必然首先输入密码，然后才能获得相应的身份，有效时间为 5 分钟。超过时间以后，需要用户再次输入密码才能继续使用其他命令。sudo 命令的语法格式如下所示。

```
sudo [-V][-h][-l][-L][-v ][-k] [ -H ] [-P ] [-S ] [ -b ] [ -p ] [ -a auth_type ] [ -u username|#uid ][ command]
```

其中各选项的具体说明如表 3-65 所示。

表 3-65 sudo 命令的选项说明

选 项	说 明
-a auth_type	指定用户认证的类型
-l	显示当前用户允许执行的命令列表
-b	后台执行指定的命令
-k	每次执行命令都要输入密码，即将密码的有效时间失效
-p	变更密码提示符
-L	显示每个允许的命令参数
-v	延长密码有效时间，默认为 5 分钟
-h	显示帮助信息
-H	使用指定的目录作为用户目录
-u username #uid	以指定的用户身份运行，默认是 root 身份。如果用户以 UID 指定，需使用#uid
command	指定执行的命令
-V	显示版本信息

例如，显示 sudo 命令可以使用的参数及其相关描述信息，如下所示。

```
# sudo -L
syslog: Syslog facility if syslog is being used for logging
syslog_goodpri: Syslog priority to use when user authenticates successfully
syslog_badpri: Syslog priority to use when user authenticates unsuccessfully
long_otp_prompt: Put OTP prompt on its own line
ignore_dot: Ignore '.' in $PATH
mail_always: Always send mail when sudo is run
... ..
```


例如，以用户 yang 的身份执行命令 ls，命令如下所示。

```
# sudo -u yang ls -l /home //以用户 yang 身份执行命令
总用量 4
drwx----- 4 yang yang 4096 6月 3 01:05 yang
```

3.2.33 suspend 命令：暂停执行 Shell

suspend 命令用于暂停正在执行的 Shell，直到收到 SIGCONT 信号为止。该命令的语法格式如下所示。

```
suspend [-f]
其中选项“-f”表示强制暂停，如下所示。
# suspend //暂停 shell
-bash: suspend: cannot suspend a login shell //无法暂停 Shell
# suspend -f //强制暂停 shell
Child has been stopped, restarting.
```

3.2.34 symlinks 命令：管理和维护符号链接

symlinks 命令将检查目录中的符号链接情况，是管理和维护符号链接的工具。symlinks 命令能够识别的符号链接类型如表 3-66 所示。

表 3-66 符号链接的类型

类 型	说 明
absolute	链接以绝对路径给出，绝对路径是从根目录开始表示的路径
dangling	源文件已经不存在的链接文件
messy	链接路径中存在无用的“.”或“/”
lengthy	链接路径中存在无用的“/..”
other_fs	所链接的源文件位于其他文件系统上
relative	链接以相对路径给出，相对路径是相对源文件的路径

symlinks 命令的语法格式如下所示。

```
symlinks [-c][-r][-s][-v][-d][-t]
```

其中各选项的具体说明如表 3-67 所示。

表 3-67 symlinks 命令的选项说明

选 项	说 明
-c	将使用绝对路径的符号链接转化为相对路径
-d	删除源文件已经不存在的符号链接（dangling）
-r	递归检查子目录，即检查指定目录及其子目录中的符号链接文件
-s	检查 lengthy 类型的符号链接
-t	如果使用了-c 选项，-t 选项将对转换进行检测，但并不进行实际的转换
-v	显示所有符号链接

例如，通过命令 symlinks 显示根目录下的所有符号链接文件，如下所示。



```
# symlinks -v / //显示所有符号链接
relative: /lamp -> lamp.txt
relative: /php -> php-5.2.5.tar.gz
```

3.2.35 vlock 命令：锁定终端

vlock 命令用来锁定虚拟终端，以避免别人使用。被锁定的终端需要输入用户密码才能取消锁定。该命令的语法格式如下所示。

```
vlock [ -a,--all ] [ -c,--current ] [ -h,--help ] [ -v,--version ]
```

其中各选项的具体说明如表 3-68 所示。

表 3-68 vlock 命令的选项说明

选 项	说 明
-a,--all	锁定所有虚拟终端
-c,--current	锁定当前虚拟终端，该项为默认选项
-h,--help	显示帮助信息
-v,--version	显示版本信息

例如，锁定当前虚拟终端，命令如下所示。

```
# vlock //锁定虚拟终端
*** This tty is not a VC (virtual console). ***
*** It may not be securely locked. ***

This TTY is now locked.
Please enter the password to unlock.
root's Password:
```

例如，通过输入密码对锁定的虚拟终端解锁，命令如下所示。

```
# vlock //锁定虚拟终端
*** This tty is not a VC (virtual console). ***
*** It may not be securely locked. ***
This TTY is now locked.
Please enter the password to unlock.
root's Password: //输入密码
root's Password:
#
```

3.2.36 wait 命令：等待程序返回状态

wait 命令是 Shell 内建命令，用来等待指定的进程结束，并返回状态值。如果不指定等待的进程，wait 将等待所有进程结束。

```
wait [n]
```

选项“n”用于指定等待的进程。例如，等待所有进程结束，命令如下。

```
#wait
```

3.2.37 watch 命令：重复执行命令并将结果输出

watch 命令可以以全屏方式重复执行指定命令，并将输出结果输出到标准输出设备。watch 命令的语法格式如下所示。


```
watch      [-d][-h][-v]  [-n <seconds>]  [--differences[=cumulative]]  [--help]
[--interval=<seconds>]
[--version] <command>
```

其中各选项的具体说明如表 3-69 所示。

表 3-69 watch 命令的选项说明

选 项	说 明
--interval=<seconds>	设定重复的时间间隔
-n <seconds>	周期性执行命令的间隔
-h,--help	显示帮助信息
-d, --differences[=cumulative]	高亮显示异常的差异，cumulative 表示对差异的处理是累积的
-v, --version	显示版本信息
command	执行的命令

例如，每隔 10 秒执行一次 ls 命令，命令如下所示。

```
# watch -n 10 ls //每隔 10 秒执行一次 who 命令
Every 5s: ls      Tue Jun 3 18:49:10 2008

18
bin
book3.txt
book3.txt~
boot
dev
etc
... ..
```

例如，每隔 10 秒执行一次 who 命令，如下所示。

```
#watch -n 10 who
Every 10s: who      Tue Jun 3 18:54:01 2008
root      :0      Jun 3 09:24
root      pts/0      Jun 3 09:25 (:0.0)
root      pts/1      Jun 3 09:25 (:pts/3:S.0)
root      pts/2      Jun 3 09:26 (:pts/3:S.1)
root      pts/3      Jun 3 09:39 (192.168.20.148)
root      pts/4      Jun 3 09:39 (:pts/3:S.0)
root      pts/5      Jun 3 09:39 (:pts/3:S.1)
```

例如，以高亮字符显示出现的差异，且差异累加，命令如下所示。

```
#watch -n 2 --difference=cumulative ls
Every 2s: ls      Tue Jun 3 19:30:28 2008

20
bin
book3.txt
book3.txt~
boot
dev
etc
homework.c      //新创建的文件
homework.c.bak  //新创建的文件
httpd-2.0.63
```



3.2.38 &命令：将任务放到后台执行

&命令指定程序在后台执行，即在执行的命令后加上符号“&”，可以使得该进程转到后台执行。有时用户需要执行的程序要花很多时间（例如从打印机输出），如果在前台运行，用户将无法进行其他工作。为此，应利用&将程序放到后台运行。

例如，下面的命令让ftp进程转到后台运行。

```
# ftp 192.168.20.122 &                                //让ftp进程在后台执行
[1] 9994
# Connected to 192.168.20.122
220 (vsFTPd 2.0.5)
530 Please login with USER and PASS.
530 Please login with USER and PASS.
```

3.3 常用的系统维护命令

系统维护命令是Linux中最常使用的一种命令，主要用来设置并修改系统的相关参数。常见的系统维护命令有30多个，包括音效设置、视窗大小设置、模块载入、系统升级等。

3.3.1 alias命令：设置命令的别名

alias命令为已经存在的命令设置别名。设置别名之后，用户可以运行自己设置的别名，来实现与已经存在命令相同的功能。这样就可以用一个较短的自定义的字符串来实现较长的完整的字符串命令，从而减少打字工作量。系统重启之后，本次设置的别名将不再生效。如果希望设置永久生效，需修改.profile或.cshrc文件。alias命令的语法格式如下所示。

```
alias [-p] [name[=value] ...]
```

其中各选项的说明如表3-70所示。

表 3-70

alisa 命令选项说明

选 项	说 明
-p	以“name=value”的形式列出所有设定的别名，该选项功能与不带任何选项的alisa命令相同
name=value	为value指定别名name
--help	显示帮助信息

例如，不输入任何参数，只输入alias，则显示当前系统下已经设置的命令别名的列表，如下所示。

```
# alias                                //列出已设置的别名
alias cp='cp -i'
alias l.='ls -d .* --color=tty'
alias ll='ls -l --color=tty'
alias ls='ls --color=tty'
alias mc='. /usr/share/mc/bin/mc-wrapper.sh'
alias mv='mv -i'
alias rm='rm -i'
```



```
alias vi='vim'
alias which='alias | /usr/bin/which --tty-only --read-alias --show-dot --show-tilde'
```

可以看到，依据当前系统下别名的设置，命令“ll”相当于命令“ls -l”，命令“ls”相当于命令“ls -color=tty”。

例如，设置命令 pwd 的别名为 p，如下所示。

```
# alias p=pwd                //设置别名
# alias                      //列出当前已经设置的别名
alias cp='cp -i'
alias l.='ls -d .* --color=tty'
alias ll='ls -l --color=tty'
alias ls='ls --color=tty'
alias mc='. /usr/share/mc/bin/mc-wrapper.sh'
alias mv='mv -i'
alias p='pwd'                //已包含设置的别名
alias rm='rm -i'
alias vi='vim'
alias which='alias | /usr/bin/which --tty-only --read-alias --show-dot --show-tilde'
# p
/
```

例如，不带参数的 rm 命令用于直接删除文件。通常为了防止误删除，都会采用参数“-i”。命令“rm -i”在删除每个文件时都会进行询问，增强了安全性。为了简化该命令，可以为“rm -i”指定别名，重载 rm 命令，如下所示。

```
#rm text1
#alias rm='rm -i'
# rm text2
rm: 是否删除 一般空文件 “text2”? y
```

3.3.2 aumix 命令：设置音效装置

aumix 命令用来调整音频设备的设置。该命令可以应用在命令行下，也可以与键盘或鼠标进行交互操作，其命令格式如下所示。

```
aumix [[+|-][<amount>] [-o][-p][-s][-w][-x][-t][-m][-l][-c][-i][-b][-v][-h][-I]
[-L][-q] [-S] [-f <rc file>][-d device_file]
[-C <color scheme file>]
```

其中各选项的具体说明如表 3-71 所示。

表 3-71 aumix 命令的选项说明

选 项	说 明
-v	主音量
-b	低音
-c	CD
-i	线路输入
-l	信号线
-m	麦克风
-o	线路输出
-p	PC 喇叭



续表

选 项	说 明
-s	合成器
-t	高音
-w	PCM 设置
-x	混声器
-h	显示帮助信息
-I	以交互模式运行 aumix，使用全屏
-L	从\$HOME/.aumixrc 或/etc/aumixrc 加载配置
-q	查询所有的设备，显示输出配置信息
-S	保存配置内容到\$HOME/.aumixrc 文件中
-1	线路 1，+/-选项增加/减少原来的强度
-2	线路 2，+/-选项增加/减少原来的强度
-3	线路 3，+/-选项增加/减少原来的强度
-C <color_scheme_file>	指定包含颜色设置摘要的文件
-d <device_file>	指定音频设备名称
-f< rc_file>	指定保存和载入配置的文件

例如，在图形方式下设置音效设备，命令如下所示。

```
# aumix //设置音效设备
```

输入命令 aumix 后，将会出现如图 3-15 所示的图形界面，用鼠标直接拖动上面的白色圆圈“O”，可以直接设置相应的选项。

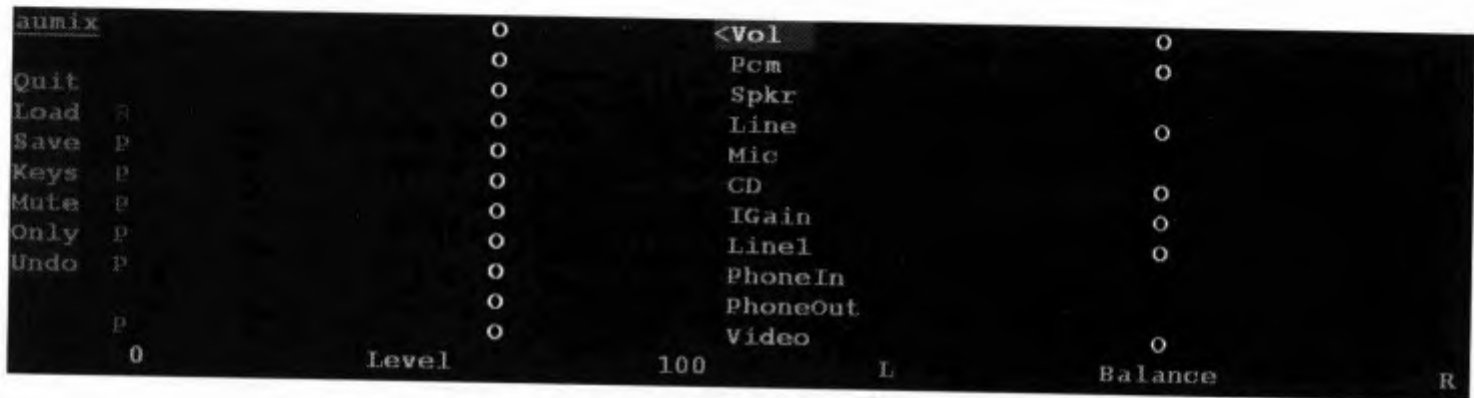


图 3-15 aumix 命令的图形设置方式

例如，将主音量设为 85，低音为 90，高音为 75，命令如下所示。

```
# aumix -v 85 -b 90 -t 75
aumix: MIXER_WRITE
```

例如，显示所有设备的设置值，命令如下所示。

```
# aumix -q
vol 85, 85
pcm 64, 64
speaker 64, 64
line 64, 64, R
mic 64, 64, P
```



```
cd 64, 64, P
igain 64, 64, P
linel 64, 64, P
phin 64, 64, P
phout 64, 64
video 64, 64, P
```

例如，将 CD 左右声道的音量设置为 80，然后显示所有设备及其设置值，命令如下所示。

```
# aumix cd -c 80 -q
vol 85, 85
pcm 64, 64
speaker 64, 64
line 64, 64, R
mic 64, 64, P
cd 80, 80, P           //CD 的左右声道音量已设备为 50
igain 64, 64, P
linel 64, 64, P
phin 64, 64, P
phout 64, 64
video 64, 64, P
```

3.3.3 authconfig 命令：配置系统的认证资源

通过 authconfig 命令可以对 NIS、/etc/passwd、/etc/shadow、LDAP、Kerberos5 以及 SMB 等资源的认证进行配置。该命令的语法格式如下所示。

```
authconfig [--back] [--test] [--nostart] [--kickstart] [--probe] [--enablecache] [--disablecache]
           [--enablenis] [--nisdomain<domain>] [--nisserver<nisserver [,nisserver...]>]]
[--disablekrb5]
           [--disablenis] [--enable-shadow] [--disablesshadow] [--enablemd5] ]
[--disablemd5] ] [--enablekrb5 ]
           [--enablel-dap] [--enableldapauth] [--enableldapssl] [--ldapserver
<ldapserver>] [--enablesmb]
           [--disableldap] [--krb5realm <realm>] [--krb5kdc<hostname[,host-name,...]>]
           [--krb5adminserver<hostname[,host-name,...]>] ] [--enablehesiod
[--hesiodlhs <lhs>]
           [--disablehesiod] [--smbworkgroup <workgroup>] [--smb-servers
<server[,server]>]]
           [--ldapbasedn <basedn>] [--disableldapauth] [--hesiodrhs <rhs>]
```

其中各选项的具体说明如表 3-72 所示。

表 3-72 authconfig 命令的选项说明

选 项	说 明
--enablesshadow	默认启用屏蔽口令
--disablesshadow	默认禁用屏蔽口令
--enablemd5	默认启用 MD5 口令
--disablemd5	默认禁用 MD5 口令
--enablenis	启用 NIS
--disablenis	禁用 NIS
--nisdomain=<domain>	默认的 NIS 域
--nisserver=<server>	默认的 NIS 服务器



续表

选 项	说 明
--enableldap	默认启用用户信息 LDAP
--disableldap	默认禁用用户信息 LDAP
--enableldaptls	允许和 LDAP 一齐使用 TLS
--disableldaptls	禁止和 LDAP 一齐使用 TLS
--enableldapauth	默认启用验证 LDAP
--disableldapauth	默认禁用验证 LDAP
--ldapserver=<server>	默认的 LDAP 服务器
--ldapbasedn=<dn>	默认的 LDAP 基点 DN
--enablekrb5	默认启用 erberos 验证
--disablekrb5	默认禁用 erberos 验证
--krb5kdc=<server>	默认的 erberos KDC
--krb5adminserver=<server>	默认的 erberos 管理服务器
--krb5realm=<realm>	默认的 erberos 领域
--enablesmbauth	默认启用 SMB 验证
--disablesmbauth	默认禁用 SMB 验证
--smbworkgroup=<workgroup>	指定 SMB 工作组
--smbservers=<server>	指定 SMB 服务器名称
--enablehesiod	默认启用用户信息 hesiod
--disablehesiod	默认禁用用户信息 hesiod
--hesiodlhs=<lhs>	默认的 hesiod LHS
--hesiodrhs=<rhs>	默认的 hesiod RHS
--enablecache	默认启用缓存用户信息
--disablecache	默认禁用缓存用户信息
--nostart	不要开始或停止 portmap、ypbind 及 nscd
--kickstart	不要显示用户界面
--probe	探测网络的默认值后打印它们
-, --help	显示帮助信息
--usage	显示用法摘要
--back	默认情况下，运行 authconfig 的图形界面包括一个【取消】按钮。如果使用 --back 选项，那么【取消】按钮将被【返回】按钮代替
--nostart	更改的设置在系统启动以后才生效
--kickstart	不显示用户界面
--probe	使用 DNS 以及其他方法猜测当前用户的配置信息，并显示输出猜测结果
--test	进行测试，设置的内容不会生效

例如，设置系统的认证信息，命令如下所示。


```
# authconfig //设置系统的认证信息
```

输入以上命令后，按下 Enter 键，将出现如图 3-16 所示的图形界面，可以进行用户信息的设置。

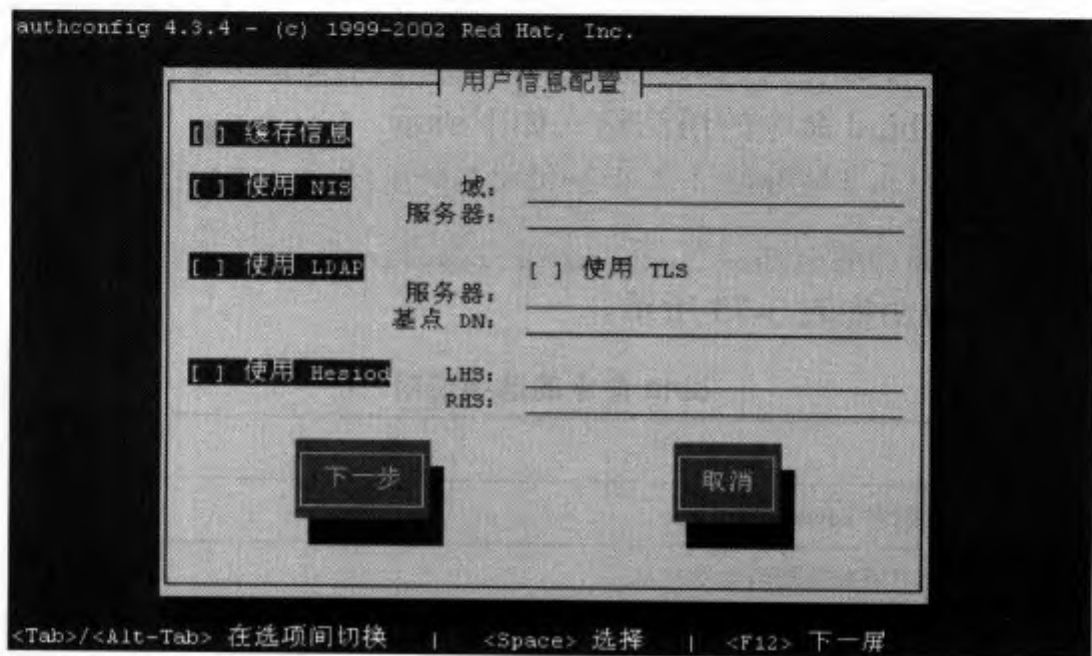


图 3-16 用户信息配置界面

通过 Tab 键选择【下一步】按钮，将出面如图 3-17 所示的验证配置界面。按 SPACE 键选择使用的验证类型。如果选项使用 LDAP 验证，需要输入服务器地址、基点 DN 的相关信息；如果选择使用 kerberos 5，则需要输入域名、KDC、管理服务器地址；如果使用 SMB 验证，则需要输入工作组、服务器的相关信息。

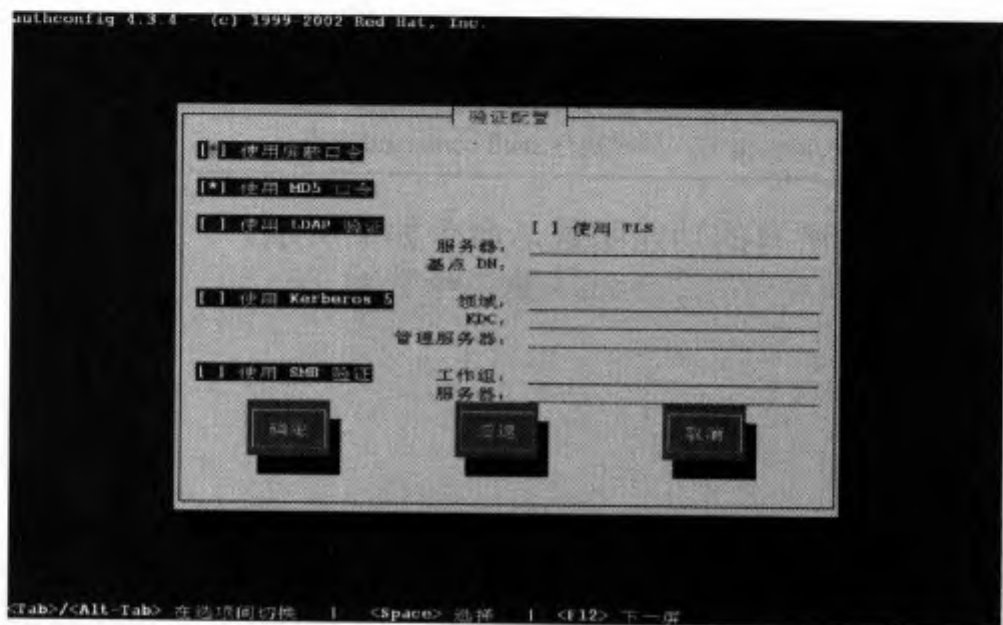


图 3-17 验证配置界面

例如，将【取消】按钮更改为【返回】按钮，命令如下所示。

```
#authconfig -back
```

以普通用户身份无法进行配置，只有 root 用户才有配置权限，如下所示。

```
# su yang
```

```
$ authconfig
```

```
authconfig: 只能作为根用户运行
```

但普通用户可以使用“--test”选项运行 authconfig，但所修改的设置无法生效，命令如



下所示。

```
#authconfig --test
```

3.3.4 bind 命令：显示或设置按键组合

bind 命令可以显示按键组合代表的功能，也可以对按钮组合进行设置，使得某些按键组合代表某些特殊的功能。bind 命令的语法格式如下所示。

```
bind [-v][-P][-V] [-m keymap] [-f filename] [-q function] [-u name] [-r keyseq] [-x keyseq:shell-command]
[keyseq:readline-function or readline-command] [-l][-p]
```

其中各选项的具体说明如表 3-73 所示。

表 3-73 bind 命令的选项说明

选 项	说 明
-r keyseq	删除对 keyseq 的绑定
-l	列出所有功能函数的名称
-P	以可重复读入方式列出所有功能函数及其绑定
-p	列出所有功能函数及其绑定
-v	以可重复读入方式列出当前设置的按键组合的名称及值
-V	列出当前设置的按键组合的名称及值
-u function	取消对指定功能的所有绑定
-f filename	从 filename 中读入指定的键盘绑定
-m keymap	设定特殊的按键组合,可以接受的 keymap 包括:emacs、emacs-standard 、emacs-meta、emacs-ctls、vi 、vi-command 及 vi-insert。其中, vi 等同于 vi-command, emacs 等同于 emacs-standard
-q function	列出指定功能的按键或按键组合
-x keyseq:shell-command	当输入 keyseq 时,同时执行 shell-command 命令

例如，显示按键组合的所有的功能函数，命令如下所示。

```
# bind -l //显示按键组合的内容
abort
accept-line
alias-expand-line
arrow-key-prefix
backward-byte
backward-char
backward-delete-char
backward-kill-line
backward-kill-word
backward-word
... ..
```

例如，查询 abort 上绑定的键，命令如下所示。

```
# bind -q abort
abort can be invoked via "\C-g", "\C-x\C-g", "\M-\C-g".
```

3.3.5 chkconfig 命令：设置和检查系统的服务设置

chkconfig 命令用于检查、更新系统的各种服务设置。chkconfig 提供了对/etc/rc[0~6].d

目录进行维护的简单方式，避免了采用复杂的符号链接进行管理带来的复杂性。Chkconfig 主要包括以下 5 个原始功能。

- 系统管理增加新的服务
- 为系统管理移除服务
- 列出当前服务的启动信息
- 改变服务的启动信息
- 检查特殊服务的启动状态

当不带任何选项运行 `chkconfig` 命令时，`chkconfig` 将显示使用方法说明。该命令的语法格式如下所示。

```
chkconfig [--list<name>] [--level levels][ --del <name>][--add <name>] [--level <levels>] [name <on|off|reset>]
```

其中各选项的具体说明如表 3-74 所示。

表 3-74 `chkconfig` 命令的选项说明

选 项	说 明
--level <levels>	指定该服务运行于哪个运行等级，运行等级范围是 0~6，如--level35 代表指定运行等级为 3 和 5
--add <name>	增加一个 <code>chkconfig</code> 命令可以管理的服务。对所增加的新服务， <code>chkconfig</code> 会检测该服务在每个运行等级是否有开启或关闭的记录。如果没有相关记录， <code>chkconfig</code> 会在该等级添加相应记录项
--del <name>	从 <code>chkconfig</code> 所管理的服务中删除指定的服务， <code>chkconfig</code> 将不再对其进行管理，同时/etc/rc[0~6].d 中指向该服务的符号链接也会被删除
--list <name>	如果指定了 <code>name</code> ，则列出 <code>name</code> 的各个运行等级的设置状态。如果没有指定 <code>name</code> ，则列出 <code>chkconfig</code> 所知道的所有服务
<on off reset>	其中“on”表示在指定的运行等级中启动指定服务，如果不指定运行等级，默认在 3、4、5 中启动服务；其中“off”表示在指定的运行等级中关闭指定服务，如果不指定运行等级，默认在 3、4、5 中关闭服务；其中“reset”表示在指定运行等级中重新启动设置的服务，使之回到脚本文件的初始值
--version	版本信息
--help	显示帮助信息

例如，列出 `chkconfig` 所知道的所有的服务设置状态，如下所示。

```
# chkconfig --list //列出 chkconfig 所知道的所有的服务
microcode_ctl 0:关闭 1:关闭 2:启用 3:启用 4:启用 5:启用 6:关闭
kudzu          0:关闭 1:关闭 2:关闭 3:启用 4:启用 5:启用 6:关闭
syslog         0:关闭 1:关闭 2:启用 3:启用 4:启用 5:启用 6:关闭
netfs          0:关闭 1:关闭 2:关闭 3:启用 4:启用 5:启用 6:关闭
network        0:关闭 1:关闭 2:启用 3:启用 4:启用 5:启用 6:关闭
random         0:关闭 1:关闭 2:启用 3:启用 4:启用 5:启用 6:关闭
rawdevices     0:关闭 1:关闭 2:关闭 3:启用 4:启用 5:启用 6:关闭
pcmcia         0:关闭 1:关闭 2:启用 3:启用 4:启用 5:启用 6:关闭
... ..
```

```
基于 xinetd 的服务:
  chargen-udp:  关闭
  rsync:        关闭
  chargen:      关闭
  daytime-udp:  关闭
  daytime:      关闭
  echo-udp:     关闭
  echo:        关闭
```



```
telnet: 关闭
...
例如，开启 telnet 服务，命令如下所示。
# chkconfig telnet on           //开启 telnet 服务
# chkconfig -list               //列出 chkconfig 所知的所有的服务的情况
microcode_ctl 0:关闭 1:关闭 2:启用 3:启用 4:启用 5:启用 6:关闭
kudzu          0:关闭 1:关闭 2:关闭 3:启用 4:启用 5:启用 6:关闭
syslog         0:关闭 1:关闭 2:启用 3:启用 4:启用 5:启用 6:关闭
...
基于 xinetd 的服务:
    chargen-udp: 关闭
    rsync: 关闭
    chargen: 关闭
    daytime-udp: 关闭
    daytime: 关闭
    echo-udp: 关闭
    echo: 关闭
    telnet: 启用 //已启用
...
```

例如，显示 telnet 服务的状态，命令如下所示。

```
# chkconfig --list telnet
telnet          启用
```

例如，关闭 telnet 服务，命令如下所示。

```
# chkconfig telnet off       //关闭 telnet 服务
# chkconfig -list           //列出 chkconfig 所知的所有的服务的情况
# chkconfig -list           //列出 chkconfig 所知的所有的服务的情况
microcode_ctl 0:关闭 1:关闭 2:启用 3:启用 4:启用 5:启用 6:关闭
kudzu          0:关闭 1:关闭 2:关闭 3:启用 4:启用 5:启用 6:关闭
syslog         0:关闭 1:关闭 2:启用 3:启用 4:启用 5:启用 6:关闭
...
基于 xinetd 的服务:
    chargen-udp: 关闭
    rsync: 关闭
    chargen: 关闭
    daytime-udp: 关闭
    daytime: 关闭
    echo-udp: 关闭
    echo: 关闭
    telnet 关闭 //已关闭
...
```

3.3.6 declare 命令：显示或设置 Shell 变量

declare 命令用来显示或设置 Shell 变量，该命令的语法格式如下所示。

```
declare [-f][-i][-r][-t][-x] [-p] [name=value]
```

其中各选项的具体说明如表 3-75 所示。

表 3-75 declare 命令的选项说明

选 项	说 明
+	关闭指定属性
-	启动指定属性
-p	显示每一变量的名称及属性值

续表

选 项	说 明
-r	只读模式。无法再对其指定值
-f	只显示函数名称
-i	变量默认为整数类型。当变量被赋予值时进行数值检验
-t	为每一变量设定追踪属性
-x	显示或设置环境变量
--help	帮助信息
--version	版本信息

例如，显示 Shell 的所有变量及值，命令如下所示。

```
# declare -p
BASH=/bin/bash                // bash 的路径
BASH_ENV=/root/.bashrc
BASH_VERSINFO=([0]="2"[1]="05b"[2]="0"[3]="1"[4]="release"[5]="i386-redhat-linux-gnu")
BASH_VERSION='2.05b.0(1)-release'
COLORS=/etc/DIR_COLORS.xterm
COLUMNS=96
DIRSTACK=()
EUID=0
GROUPS=()
G_BROKEN_FILENAMES=1
HISTFILE=/root/.bash_history
HISTFILESIZE=1000
HISTSIZE=1000
HOME=/root
HOSTNAME=localhost.localdomain
HOSTTYPE=i386
IFS=$' \t\n'
INPUTRC=/etc/inputrc
LAMHELPPFILE=/etc/lam/lam-helpfile
LANG=zh_CN.GB18030
LANGUAGE=zh_CN.GB18030:zh_CN.GB2312:zh_CN                //默认的语言设置
LESSOPEN='|/usr/bin/lesspipe.sh %s'
LINES=33
LOGNAME=root
LS_COLORS='no=00:fi=00:di=00;34:ln=00;36:pi=40;33:so=00;35:bd=40;33;01:cd=40;33;01:
or=01;05;37;41:mi=01;05;37;41:ex=00;32:*.cmd=00;32:*.exe=00;32:*.com=00;32:*.btm=00;32:
*.bat=00;32:*.sh=00;32:*.csh=00;32:*.tar=00;31:*.tgz=00;31:*.arj=00;31:*.taz=00;31:*
.lzh=00;31:*.zip=00;31:*.z=00;31:*.Z=00;31:*.gz=00;31:*.bz2=00;31:*.bz=00;31:*.tz=00;31
:*.rpm=00;31:*.cpio=00;31:*.jpg=00;35:*.gif=00;35:*.bmp=00;35:*.xbm=00;35:*.xpm=00;35:*
.png=00;35:*.tif=00;35:'
MACHTYPE=i386-redhat-linux-gnu
MAIL=/var/spool/mail/root
MAILCHECK=60
OPTERR=1
OPTIND=1
OSTYPE=linux-gnu
PATH=/usr/kerberos/sbin:/usr/kerberos/bin:/usr/local/sbin:/usr/local/bin:/sbin:/bin
```



```
:/usr/sbin:/usr/bin:/usr/X11R6/bin:/root/bin //变量 PATH, 默认的工作路径
PIPESTATUS=([0]="0")
PPID=18049
PROMPT_COMMAND='echo -ne "\033]0;${USER}@${HOSTNAME%%.*}:${PWD/#$HOME/~}\007"'
PS1='[\u@\h \W]\$ '
PS2='> '
PS4='+ '
PVM_ROOT=/usr/share/pvm3
PVM_RSH=/usr/bin/rsh
PWD=/root
QTDIR=/usr/lib/qt-3.1
REMOTEHOST=192.168.20.148
SHELL=/bin/bash //默认的 Shell
SHELLOPTS=braceexpand:emacs:hashall:histexpand:history:interactive-comments:monitor
SHLV=1
SSH_ASKPASS=/usr/libexec/openssh/gnome-ssh-askpass
SUPPORTED=zh_CN.GB18030:zh_CN:zh
TERM=xterm
UID=0
USER=root
USERNAME=root
XPVM_ROOT=/usr/share/pvm3/xpvm
_PATH
```

例如, 修改变量 BASH 的值为/bin/csh, 命令如下所示。

```
# declare BASH=/bin/csh
# declare //显示所有变量
BASH=/bin/csh
BASH_ENV=/root/.bashrc
BASH_VERSINFO=([0]="2" [1]="05b" [2]="0" [3]="1" [4]="release" [5]="i386-redhat-linux-gnu")
BASH_VERSION='2.05b.0(1)-release'
COLORS=/etc/DIR_COLORS.xterm
COLUMNS=96
```

例如, 显示所有环境变量的值, 命令如下所示。

```
# declare -x
declare -x BASH_ENV="/root/.bashrc"
declare -x G_BROKEN_FILENAMES="1"
declare -x HISTSIZE="1000"
declare -x HOME="/root"
declare -x HOSTNAME="localhost.localdomain"
declare -x INPUTRC="/etc/inputrc"
declare -x LAMHELPPFILE="/etc/lam/lam-helpfile"
declare -x LANG="zh_CN.GB18030"
declare -x LANGUAGE="zh_CN.GB18030:zh_CN.GB2312:zh_CN"
declare -x LESSOPEN="|/usr/bin/lesspipe.sh %s"
```

3.3.7 dircolors 命令: 设置 ls 命令显示的颜色

dircolors 命令用来设置 ls 命令对应的颜色配置, 该命令的语法格式如下所示。

```
dircolors [-p, --print-database] [-b, --sh, --bourne-shell] [-c, --csh,
--c-shell] [FILE] [--help] [--version]
```

其中各选项的具体说明如表 3-76 所示。

表 3-76 dircolors 命令的选项说明

选 项	说 明
-p, --print-database	显示默认值
-b, --sh, --bourne-shell	显示 Bourne Shell 颜色代码设置
-c, --csh, --c-shell	显示 C Shell 颜色代码设置
FILE	指定颜色配置文件
--help	帮助信息
--version	版本信息

例如，显示默认颜色设置，命令如下所示。

```
# dircolors -p                               //显示默认值
# Configuration file for dircolors, a utility to help you set the
# LS_COLORS environment variable used by GNU ls with the --color option.

# The keywords COLOR, OPTIONS, and EIGHTBIT (honored by the
# slackware version of dircolors) are recognized but ignored.//适用的终端类型列表
# Below, there should be one TERM entry for each termttype that is colorizable
TERM linux
TERM linux-c
TERM mach-color
TERM console
TERM con132x25
TERM con132x30
TERM con132x43
TERM con132x60
TERM con80x25
TERM con80x28
TERM con80x30
TERM con80x43
TERM con80x50
TERM con80x60
TERM dtterm
TERM xterm
TERM xterm-color
TERM xterm-debian
TERM rxvt
TERM screen
TERM screen-w
TERM vt100
TERM Eterm

# Below are the color init strings for the basic file types. A color init
# string consists of one or more of the following numeric codes:
# Attribute codes:
# 00=none 01=bold 04=underscore 05=blink 07=reverse 08=concealed
# Text color codes:
# 30=black 31=red 32=green 33=yellow 34=blue 35=magenta 36=cyan 37=white
# Background color codes:
# 40=black 41=red 42=green 43=yellow 44=blue 45=magenta 46=cyan 47=white
NORMAL 00      # global default, although everything should be something.
```




```
FILE 00          # normal file                //普通文件设置
DIR 01;34         # directory                  //目录文件设置
LINK 01;36        # symbolic link. (If you set this to 'target' instead of a //链接文件设置
                  # numerical value, the color is as for the file pointed to.)
FIFO 40;33        # pipe                      //管道文件设置
SOCK 01;35        # socket                    //socket 文件设置
DOOR 01;35        # door
BLK 40;33;01      # block device driver
CHR 40;33;01      # character device driver
ORPHAN 40;31;01   # symlink to nonexistent file

# This is for files with execute permission:    //可执行文件设置
EXEC 01;32

# List any file extensions like '.gz' or '.tar' that you would like ls
# to colorize below. Put the extension, a space, and the color init string.
# (and any comments you want to add after a '#')

# If you use DOS-style suffixes, you may want to uncomment the following:
#.cmd 01;32 # executables (bright green)
#.exe 01;32
#.com 01;32
#.btm 01;32
#.bat 01;32

.tar 01;31 # archives or compressed (bright red)    //归档或压缩文件设置 (亮红色)
.tgz 01;31
.arj 01;31
.taz 01;31
.lzh 01;31
.zip 01;31
.z 01;31
.Z 01;31
.gz 01;31
.bz2 01;31
.deb 01;31
.rpm 01;31
.jar 01;31

# image formats                                //图像设置
.jpg 01;35
.jpeg 01;35
.gif 01;35
.bmp 01;35
.pbm 01;35
.pgm 01;35
.ppm 01;35
.tga 01;35
.xbm 01;35
.xpm 01;35
.tif 01;35
.tiff 01;35
.png 01;35
.mpg 01;35
.mpeg 01;35
```



```
.avi 01;35
.fli 01;35
.gl 01;35
.dl 01;35
.xcf 01;35
.xwd 01;35

# audio formats //音频设置
.ogg 01;35
.mp3 01;35
.wav 01;35
```

例如，显示 Bourne Shell 颜色代码设置，命令如下所示。

```
# dircolors -b
LS_COLORS='no=00:fi=00:di=01;34:ln=01;36:pi=40;33:so=01;35:do=01;35:bd=40;33;01:cd=
40;33;01:or=40;31;01:ex=01;32:*.tar=01;31:*.tgz=01;31:*.arj=01;31:*.taz=01;31:*.lzh=01;
31:*.zip=01;31:*.z=01;31:*.Z=01;31:*.gz=01;31:*.bz2=01;31:*.deb=01;31:*.rpm=01;31:*.jar
=01;31:*.jpg=01;35:*.jpeg=01;35:*.gif=01;35:*.bmp=01;35:*.pbm=01;35:*.pgm=01;35:*.ppm=0
1;35:*.tga=01;35:*.xbm=01;35:*.xpm=01;35:*.tif=01;35:*.tiff=01;35:*.png=01;35:*.mpg=01;
35:*.mpeg=01;35:*.avi=01;35:*.fli=01;35:*.gl=01;35:*.dl=01;35:*.xcf=01;35:*.xwd=01;35:*.
ogg=01;35:*.mp3=01;35:*.wav=01;35:';
export LS_COLORS
```

例如，显示 C Shell 颜色代码设置，命令如下所示。

```
# dircolors -c
setenv LS_COLORS 'no=00:fi=00:di=01;34:ln=01;36:pi=40;33:so=01;35:do=01;35:bd=40;33;
01:cd=40;33;01:or=40;31;01:ex=01;32:*.tar=01;31:*.tgz=01;31:*.arj=01;31:*.taz=01;31:*.l
zh=01;31:*.zip=01;31:*.z=01;31:*.Z=01;31:*.gz=01;31:*.bz2=01;31:*.deb=01;31:*.rpm=01;31
:*.jar=01;31:*.jpg=01;35:*.jpeg=01;35:*.gif=01;35:*.bmp=01;35:*.pbm=01;35:*.pgm=01;35:*.
ppm=01;35:*.tga=01;35:*.xbm=01;35:*.xpm=01;35:*.tif=01;35:*.tiff=01;35:*.png=01;35:*.m
pg=01;35:*.mpeg=01;35:*.avi=01;35:*.fli=01;35:*.gl=01;35:*.dl=01;35:*.xcf=01;35:*.xwd=0
1;35:*.ogg=01;35:*.mp3=01;35:*.wav=01;35:'
```

3.3.8 enable 命令：开动或关闭 Shell 内置命令

enable 命令用来开启或关闭 Shell 内置的命令，该命令的语法格式如下所示。

```
enable [-p][-n][-d][-s] [-a] [-f filename] [name]
```

其中各选项的具体说明如表 3-77 所示。

表 3-77 enable 命令的选项说明

选 项	说 明
-a	显示所有的内置命令，同时显示该命令是否已经加载
name	指定加载的命令
-n<name>	关闭指定的命令
-f filename	从指定的文件加载内置命令
-d	删除采用-f 命令加载的内置命令
-p	显示已加载的内置命令列表
-s	仅显示 POSIX 定义的内置命令

例如，显示所有的内置命令以及该命令是否被开启，命令如下所示。



```
# enable -a //显示 Shell 指令
enable .
enable :
enable [
enable alias
enable bg
enable bind
enable break
enable builtin
enable cd //cd 命令, 已加载
enable command
enable compgen
enable complete
enable continue
enable declare
enable dirs
enable disown
enable echo //echo 命令, 已加载
enable enable
enable eval
enable exec
enable exit
enable export
enable false
enable fc
enable fg
enable getopts
enable hash
enable help
enable history //history 命令, 已加载
enable jobs
enable kill
enable let
enable local
enable logout
enable popd
enable printf
enable pushd
enable pwd //pwd 命令, 已加载
enable read
enable readonly
enable return
enable set
enable shift
enable shopt
enable source
enable suspend
enable test
enable times
enable trap
enable true
enable type
enable typeset
enable ulimit
enable umask
```



```
enable unalias
enable unset
enable wait
```

//wait 命令, 已加载

例如, 关闭内部命令 bg、fg、break、pwd、unalias 和 alias, 命令如下所示。

```
# enable -n bg break alias
```

```
# enable -a
```

//显示所有命令及状态

```
enable .
```

```
enable :
```

```
enable [
```

```
enable -n alias
```

//alias 命令, 已关闭

```
enable -n bg
```

//bg 命令, 已关闭

```
enable bind
```

```
enable -n break
```

//break 命令, 已关闭

```
enable builtin
```

```
enable cd
```

```
enable command
```

```
enable compgen
```

```
enable complete
```

```
enable continue
```

```
enable declare
```

```
enable dirs
```

```
enable disown
```

```
enable echo
```

```
enable enable
```

```
enable eval
```

```
enable exec
```

```
enable exit
```

```
enable export
```

```
enable false
```

```
enable fc
```

```
enable fg
```

//fg 命令, 已关闭

```
enable getopts
```

```
enable hash
```

```
enable help
```

```
enable history
```

```
enable jobs
```

```
enable kill
```

```
enable let
```

```
enable local
```

```
enable logout
```

```
enable popd
```

```
enable printf
```

```
enable pushd
```

```
enable pwd
```

//pwd 命令, 已关闭

```
enable read
```

```
enable readonly
```

```
enable return
```

```
enable set
```

```
enable shift
```

```
enable shopt
```

```
enable source
```

```
enable suspend
```

```
enable test
```

```
enable times
```




```
enable trap
enable true
enable type
enable typeset
enable ulimit
enable umask
enable unalias //unalias 命令, 已关闭
enable unset
enable wait
# alias
-bash: alias: command not found //无效的 alias 命令
# unalias
-bash: unalias: command not found //无效的 unalias 命令
# break
-bash: break: command not found //无效的 break 命令
# pwd
-bash: pwd: command not found //无效的 pwd 命令
# bg
-bash: bg: command not found //无效的 bg 命令
# fg
-bash: fg: command not found //无效的 fg 命令
```

例如, 加载内部命令 `alias`, 命令如下所示。

```
#enable alias
# alias //alias 命令已加载
alias cp='cp -i'
alias l.='ls -d .* --color=tty'
alias ll='ls -l --color=tty'
alias ls='ls --color=tty'
alias mc='. /usr/share/mc/bin/mc-wrapper.sh'
alias mv='mv -i'
alias rm='rm -i'
alias vi='vim'
alias which='alias | /usr/bin/which --tty-only --read-alias --show-dot --show-tilde'
```

3.3.9 eval 命令：连续执行多个命令

`eval` 命令用来连接多个命令, 被连接的多个命令将依次执行。该命令的语法格式如下所示。

```
eval [command ; command...]
```

其中选项“`command`”表示连接的多个命令, 命令之间以分号分隔。例如, 连接 `pwd` 命令和 `ls` 命令, 命令及执行结果如下所示。

```
# eval pwd; ls //连接多个命令
/root
*#          bubble      bubble.c      install.log      kk2  showVar
anaconda-ks.cfg bubble.bb  bubble.c.gcov  install.log.syslog kk3  temp.log
b           bubble.bbg  bubble.da      kk               kk4  textbox.txt
```

可以看到, 通过命令 `eval`, `pwd` 命令和 `ls` 命令依次被执行。

3.3.10 export 命令：设置或显示环境变量

通过 `export` 命令, 可以设置或显示环境变量。该命令的语法格式如下所示。

```
export [-f][-n][-p] [name[=word]]
```


其中各选项的具体说明如表 3-78 所示。

表 3-78 export 命令的选项说明

选 项	说 明
-f	代表环境变量为函数名称
-n	删除指定的环境变量
-p	列出所有的环境变量
name	环境变量名
word	环境变量值

例如，列出当前所有的环境变量，命令如下所示。

```
# export -p //列出当前的环境变量值
declare -x BASH_ENV="/root/.bashrc"
declare -x G_BROKEN_FILENAMES="1"
declare -x HISTSIZE="1000"
declare -x HOME="/root"
declare -x HOSTNAME="localhost.localdomain"
declare -x INPUTRC="/etc/inputrc"
declare -x LAMHELPPFILE="/etc/lam/lam-helpfile"
declare -x LANG="zh_CN.GB18030"
declare -x LANGUAGE="zh_CN.GB18030:zh_CN.GB2312:zh_CN"
declare -x LESSOPEN="|/usr/bin/lesspipe.sh %s"
declare -x LOGNAME="root"
declare -x LS_COLORS="no=00:fi=00:di=00;34:ln=00;36:pi=40;33:so=00;35:bd=40;33;01:cd
=40;33;01:or=01;05;37;41:mi=01;05;37;41:ex=00;32:*.cmd=00;32:*.exe=00;32:*.com=00;32:*.
btm=00;32:*.bat=00;32:*.sh=00;32:*.csh=00;32:*.tar=00;31:*.tgz=00;31:*.arj=00;31:*.taz=
00;31:*.lzh=00;31:*.zip=00;31:*.z=00;31:*.Z=00;31:*.gz=00;31:*.bz2=00;31:*.bz=00;31:*.t
z=00;31:*.rpm=00;31:*.cpio=00;31:*.jpg=00;35:*.gif=00;35:*.bmp=00;35:*.xbm=00;35:*.xpm=
00;35:*.png=00;35:*.tif=00;35:"
declare -x MAIL="/var/spool/mail/root"
declare -x OLDPWD="/tmp/empty"
declare -x PATH="/usr/kerberos/sbin:/usr/kerberos/bin:/usr/local/sbin:/usr/local/
bin:/sbin:/bin:/usr/sbin:/usr/bin:/usr/X11R6/bin:/root/bin"
declare -x PVM_ROOT="/usr/share/pvm3"
declare -x PVM_RSH="/usr/bin/rsh"
declare -x PWD="/tmp"
declare -x QTDIR="/usr/lib/qt-3.1"
declare -x REMOTEHOST="192.168.20.148"
declare -x SHELL="/bin/bash"
declare -x SHLVL="1"
declare -x SSH_ASKPASS="/usr/libexec/openssh/gnome-ssh-askpass"
declare -x TERM="xterm"
declare -x USER="root"
declare -x USERNAME="root"
declare -x XPVM_ROOT="/usr/share/pvm3/xpvm"
```

例如，自定义一个环境变量 MYENVIRON，命令如下所示。

```
# export MYENVIRON //定义环境变量
# export -p //列出当前的环境变量
declare -x BASH_ENV="/root/.bashrc"
declare -x G_BROKEN_FILENAMES="1"
declare -x HISTSIZE="1000"
```



```

declare -x HOME="/root"
declare -x HOSTNAME="localhost.localdomain"
declare -x INPUTRC="/etc/inputrc"
declare -x LAMHELPPFILE="/etc/lam/lam-helpfile"
declare -x LANG="zh_CN.GB18030"
declare -x LANGUAGE="zh_CN.GB18030:zh_CN.GB2312:zh_CN"
declare -x LESSOPEN="|/usr/bin/lesspipe.sh %s"
declare -x LOGNAME="root"
declare -x LS_COLORS="no=00:fi=00:di=00;34:ln=00;36:pi=40;33:so=00;35:bd=40;33;01:cd
=40;33;01:or=01;05;37;41:mi=01;05;37;41:ex=00;32:*.cmd=00;32:*.exe=00;32:*.com=00;32:*.
btm=00;32:*.bat=00;32:*.sh=00;32:*.csh=00;32:*.tar=00;31:*.tgz=00;31:*.arj=00;31:*.taz=
00;31:*.lzh=00;31:*.zip=00;31:*.z=00;31:*.Z=00;31:*.gz=00;31:*.bz2=00;31:*.bz=00;31:*.t
z=00;31:*.rpm=00;31:*.cpio=00;31:*.jpg=00;35:*.gif=00;35:*.bmp=00;35:*.xbm=00;35:*.xpm=
00;35:*.png=00;35:*.tif=00;35:"
declare -x MAIL="/var/spool/mail/root"
declare -x MYENVIRON //自定义的变量 MYENVIRON
declare -x OLDPWD="/tmp/empty"
declare -x PATH="/usr/kerberos/sbin:/usr/kerberos/bin:/usr/local/sbin:/usr/local/bin:
/sbin:/bin:/usr/sbin:/usr/bin:/usr/X11R6/bin:/root/bin"
declare -x PVM_ROOT="/usr/share/pvm3"
declare -x PVM_RSH="/usr/bin/rsh"
declare -x PWD="/tmp"
declare -x QTDIR="/usr/lib/qt-3.1"
declare -x REMOTEHOST="192.168.20.148"
declare -x SHELL="/bin/bash"
declare -x SHLVL="1"
declare -x SSH_ASKPASS="/usr/libexec/openssh/gnome-ssh-askpass"
declare -x TERM="xterm"
declare -x USER="root"
declare -x USERNAME="root"
declare -x XPVM_ROOT="/usr/share/pvm3/xpvm"

```

可以看到，通过命令 `export` 定义了新的环境变量 `MYENVIRON`；通过运行命令“`export -p`”，发现当前的环境变量列表中多了一个环境变量 `MYENVIRON`，即如上所示的“`declare -x MYENVIRON`”。

下面的命令给自定义环境变量 `MYENVIRON` 赋值为“Red Hat Linux 9”，如下所示。

```

# export MYENVIRON="Red Hat Linux 9" //定义环境变量并赋值
# export -p
declare -x BASH_ENV="/root/.bashrc"
declare -x G_BROKEN_FILENAMES="1"
declare -x HISTSIZE="1000"
declare -x HOME="/root"
declare -x HOSTNAME="localhost.localdomain"
declare -x Hat
declare -x INPUTRC="/etc/inputrc"
declare -x LAMHELPPFILE="/etc/lam/lam-helpfile"
declare -x LANG="zh_CN.GB18030"
declare -x LANGUAGE="zh_CN.GB18030:zh_CN.GB2312:zh_CN"
declare -x LESSOPEN="|/usr/bin/lesspipe.sh %s"
declare -x LOGNAME="root"
declare -x LS_COLORS="no=00:fi=00:di=00;34:ln=00;36:pi=40;33:so=00;35:bd=40;33;01:cd
=40;33;01:or=01;05;37;41:mi=01;05;37;41:ex=00;32:*.cmd=00;32:*.exe=00;32:*.com=00;32:
*.btm=00;32:*.bat=00;32:*.sh=00;32:*.csh=00;32:*.tar=00;31:*.tgz=00;31:*.arj=00;31:*.
taz=00;31:*.lzh=00;31:*.zip=00;31:*.z=00;31:*.Z=00;31:*.gz=00;31:*.bz2=00;31:*.bz=00;

```



```

31:*.tz=00;31:*.rpm=00;31:*.cpio=00;31:*.jpg=00;35:*.gif=00;35:*.bmp=00;35:*.xbm=00;35:
*.xpm=00;35:*.png=00;35:*.tif=00;35:"
declare -x Linux
declare -x MAIL="/var/spool/mail/root"
declare -x MYENVIRON="Red Hat Linux 9" //环境变量MYENVIRON的值为Red Hat Linux 9
declare -x OLDPWD="/tmp/empty"
declare -x PATH="/usr/kerberos/sbin:/usr/kerberos/bin:/usr/local/sbin:/usr/local/bin:
/sbin:/bin:/usr/sbin:/usr/bin:/usr/X11R6/bin:/root/bin"
declare -x PVM_ROOT="/usr/share/pvm3"
declare -x PVM_RSH="/usr/bin/rsh"
declare -x PWD="/tmp"
declare -x QTDIR="/usr/lib/qt-3.1"
declare -x REMOTEHOST="192.168.20.148"
declare -x SHELL="/bin/bash"
declare -x SHLVL="1"
declare -x SSH_ASKPASS="/usr/libexec/openssh/gnome-ssh-askpass"
declare -x TERM="xterm"
declare -x USER="root"
declare -x USERNAME="root"
declare -x XPVM_ROOT="/usr/share/pvm3/xpvm"

```

3.3.11 false 命令：返回值为失败

false 命令什么事情也不做，直接返回一个状态码，且状态码表示失败，即值为“1”。例如，执行 false 命令，其返回值为“1”，如下所示。

```

# ls
*#          bubble      bubble.c      install.log      kk2  showVar
anaconda-ks.cfg bubble.bb  bubble.c.gcov install.log.syslog kk3  temp.log
b           bubble.bbg bubble.da      kk               kk4  textbox.txt
# echo $?
0
//上次命令成功执行，返回值为0
# false
//设置状态码值为1
# echo $?
//返回上次命令的执行结果
1
//值为1，表示失败

```

3.3.12 fbset 命令：显示或设置帧缓存

fbset 命令是一个系统工具，用来显示或更改帧缓存设备的设置。帧缓存设备提供一个相对简单的接口，用来实现不同种类的图像显示。帧缓存设备通过/dev 目录下指定的设备节点来访问，fbset 使用自己的视频模式数据库（通常为/etc/fb.modes），可以在该数据库中定义视频模式。该命令的语法格式如下所示。

```

fbset [-h, --help][ --test][ -s, --show][ -i, --info][ -v, --verbose][ -V,
--version][ -x, --xfree86][ -a, --all]
      [ -fb <device>][ -db <file>][ -xres <value>][ -yres <value>][ -vxres
<value>][ -vyres <value>][ -depth <value>]
      [ -nonstd <value>][ -g, --geometry][ -match][ -pixclock <value>][ -left
<value>][ -right <value>][ -upper <value>]
      [ -lower<value>][ -hslen <value>][ -vslen <value>][ -t, --timings ...][ -accel
<value>][ -hsync <value>]
      [ -vsync <value>][ -vsync <value>][ -csync <value>][ -gsync <value>][ -extsync
<value>][ -bcast <value>]
      [ -laced<value>][ -double<value>][ -rgba<r,g,b,a>][ -grayscale<value>][ -move
<direction>][ -step <value>]

```



其中各选项的具体说明如表 3-79 所示。

表 3-79 fbset 命令的选项说明

选 项	说 明
-h, --help	显示帮助信息
--test	仅对模式是否正确进行测试，不进行更改
-s, --show	显示显卡模式设置
-i, --info	显示所有帧缓存信息
-v, --verbose	冗余模式
-V, --version	显示版本信息
-x, --xfree86	XFree86 兼容模式
-a, --all	修改该设备上的所有虚拟终端
-fb <device>	指定帧缓存设备，默认为/dev/fb0
-db <file>	指定视频模式数据库文件，默认为/etc/fb.modes
-xres <value>	水平分辨率（单位为像素）
-yres <value>	垂直分辨率（单位为像素）
-vxres <value>	虚拟水平分辨率（单位为像素）
-vyres <value>	虚拟垂直分辨率（单位为像素）
-depth <value>	颜色深度（单位为位/像素）
-nonstd <value>	选择非标准视频模式
-g, --geometry ...	一次性设置所有的几何参数
-match	通过虚拟分辨率设置虚拟垂直分辨率
-pixclock <value>	像素时钟（单位为皮秒）
-left <value>	左边距（单位为像素）
-right <value>	右边距（单位为像素）
-upper <value>	上边距（单位为像素）
-lower <value>	下边距（单位为像素）
-hslen <value>	水平同步宽度（单位为像素）
-vslen <value>	垂直同步宽度（单位为像素）
-t, --timings ...	一次性设置所有的定时参数
-accel <value>	硬件文件加速设置（值为 false 或 true）
-hsync <value>	水平同步极设置（值为 low 或 high）
-vsync <value>	垂直同步极设置（值为 low 或 high）
-csync <value>	复合同步极设置（值为 low 或 high）
-gsync <value>	绿色同步（值为 false 或 true）
-extsync <value>	扩展同步（值为 false 或 true）
-bcast <value>	允许广播（值为 false 或 true）

续表

选 项	说 明
-laced <value>	允许隔行扫描（值为 false 或 true）
-double <value>	允许双倍扫描（值为 false 或 true）
-rgba <r,g,b,a>	推荐的颜色深度
-grayscale <value>	允许灰阶（值为 false 或 true）
-move <direction>	移动可见部分（上、下、左、右）
-step <value>	步长设置（单位为像素或像素行，默认水平为 8，垂直为 2）

例如，设置画面的分辨率为 800×600、桌面的分辨率为 1024×768，命令如下所示。

```
# fbset -g 800 600 1024 768 //设置分辨率
```

例如，启动硬件文本加速，命令如下所示。

```
# fbset -accel true //启动硬件文本加速
```

例如，启动广播功能，命令如下所示。

```
# fbset -bcast true //启动广播功能
```

例如，对默认模式进行测试，命令如下所示：

```
# fbset -test //进行测试
open /dev/fb0: No such device
```

3.3.13 hostid 命令：显示出当前主机的数字标识

hostid 命令以十六进制格式显示当前主机的数字化标识。该命令的语法格式如下所示。

```
hostid [--help][--version]
```

其中各选项的具体说明如表 3-80 所示。

表 3-80 hostid 命令的选项说明

选 项	说 明
--help	显示 hostid 命令的帮助信息
--version	显示 hostid 命令的版本信息

例如，显示当前主机的数字标识，命令如下所示。

```
# hostid //显示主机标识
7f0100
#
```

3.3.14 insmod 命令：载入模块

insmod 命令用来向内核中添加模块。如果 insmod 命令在使用时没有指定搜索的路径，则其会在默认的公共目录中进行搜索。如果指定了环境变量 MODPATH，默认搜索目录将被重载。如果模块配置文件/etc/modules.conf 存在，则 MODPATH 中的指定将被重载。insmod 命令的语法格式如下所示。

```
insmod [-f, --force][ -k, --autoclean][ -m, --map][ -p, --prob][ -s, --syslog][ -v,
--verbose][ -V, --version] [ -h, --help]
[ -o module_name, --name=module_name][ -L, --lock][ -O blob_name,
--blob=blob_name] [ -q, --quiet]
[ -N, --numeric-only]
```



其中各选项的具体说明如表 3-81 所示。

表 3-81 insmod 命令的选项说明

选 项	说 明
-f, --force	强制加载模块
-k, --autoclean	模块的自动卸载设置
-m, --map	显示模块加载信息
-p, --probe	检测模块是否被成功加载, 包括定位模块路径是否存在目标文件, 检查版本号, 解析标识
-s, --syslog	将所有信息进行日志, 而不是从终端输出信息
-v, --verbose	运行时, 显示详细的处理信息
-V, --version	显示版本信息
-o module_name, --name=module_name	明确地指定模块名称, 而不是从源文件的主文件名获得名称
-L, --lock	通过使用 flock(2)阻止同时加载相同的模块
-O blob_name, --blob=blob_name	在 blob_name 中保存二进制对象。该文件中保存了加载模块的准备信息
-h, --help	显示帮助信息
-q, --quiet	静默模式, 不显示未解析的内容, 不对版本是否匹配进行提示
-N, --numeric-only	仅对模块版本号和内核版本号的数字部分进行检测。对于 2.5 以后的内核版本, 该选项是默认选项, 之前版本为可选



大部分用户可能会使用 modprobe 命令添加新的模块。与 insmod 命令相比, modprobe 命令使用起来更方便一些。

例如, 加载模块 usb-uhci, 命令如下所示。

```
# lsmod //显示已经安装的模块
Module      Size  Used by    Not tainted
nls_iso8859-1 3516  0 (autoclean)
ide-cd       35708 0 (autoclean)
cdrom        33728  0 (autoclean) [ide-cd]
parport_pc   19076  1 (autoclean)
mousedev     5492   0
input        5856   0 [keybdev mousedev hid]
sd_mod       13452   6
scsi_mod     107128  2 [BusLogic sd_mod]
... ..
# insmod -v usb-uhci //使用 insmod 命令安装模块“usb-uhci”,
xftw starting at /lib/modules/boot lstat on /lib/modules/boot failed
xftw starting at /lib/modules/2.4.20-8
xftw_readdir /lib/modules/2.4.20-8
pruned build
pruned modules.dep
pruned modules.generic_string
pruned modules.ieee1394map
pruned modules.isapnpmap
... ..
```



```
# lsmod                                     //显示已经安装的模块
Module      Size Used by    Not tainted
nls_iso8859-1 3516  0  (autoclean)
ide-cd       35708  0  (autoclean)
cdrom        33728  0  (autoclean) [ide-cd]
parport_pc   19076  1  (autoclean)
mousedev     5492   0
input        5856   0  [keybdev mousedev hid]
usb-uhci     26348  0  (unused)           //已加载
sd_mod       13452   6
scsi_mod     107128  2  [BusLogic sd_mod]
... ..
```

3.3.15 ldconfig 命令：配置动态链接绑定

ldconfig 命令用于创建与最近共享库之间的链接和缓存。最近共享库包括命令行中的指定或/etc/ld.so.conf 文件中的定义，以及一些可信目录（如/usr/lib 或/lib）。ldconfig 命令的语法格式如下所示。

```
ldconfig [-c, --format=FORMAT] [-C CACHE] [-f CONF] [-l] [-n] [-N] [-p, --print-cache] [-r ROOT] [-v, --verbose]
          [-X] [-?, --help] [--usage] [-V, --version]
```

其中各选项的具体说明如表 3-82 所示。

表 3-82 ldconfig 命令选项说明

选 项	说 明
-c, --format=FORMAT	设定格式：新、旧、压缩（默认）
-C CACHE	使用指定的 CACHE 作为缓存文件
-f CONF	使用指定的 CONF 作为配置文件
-l	根据用户指定来链接库文件
-n	只处理在命令行中指定的目录，不建立缓存
-N	不建立缓存
-p, --print-cache	打印缓冲区中的目录列表和函数库
-r ROOT	更改 root 目录
-v, --verbose	冗余模式，显示详细信息
-X	不产生链接
-, --help	显示帮助列表
--usage	显示用法摘要
-V, --version	显示版本信息



由于 ldconfig 命令可能需要对一些 root 权限的文件进行写操作，所以，通常在 root 用户下运行 ldconfig 命令。

例如，建立对于共享库的正确链接，更新缓冲区，并显示所有相关信息，命令如下。



```
# ldconfig -v
/lib:
    libcrypto.so.2 -> libcrypto.so.0.9.6b
    libssl.so.2 -> libssl.so.0.9.6b
    libnss_db.so.1 -> libnss_db.so.1.0.0
    libnss_db.so.2 -> libnss_db.so.2.0.0
    libpwdb.so.0 -> libpwdb.so.0.62
    libdb-3.3.so -> libdb-3.3.so
    libdb-3.2.so -> libdb-3.2.so
    libdb.so.3 -> libdb2.so.3
    libdb-3.1.so -> libdb-3.1.so
    libcap.so.1 -> libcap.so.1.10
    libnss_wins.so -> libnss_wins.so.2
    libiw.so.25 -> libiw.so.25
    libnss_winbind.so -> libnss_winbind.so.2
    libnss_ldap.so.2 -> libnss_ldap-2.3.1.so
    libpam_misc.so.0 -> libpam_misc.so.0.75
    libpam.so.0 -> libpam.so.0.75
    libproc.so.2.0.11 -> libproc.so.2.0.11
```

例如，只针对指定的目录/lib 进行更新，并显示所有相关详细信息，命令如下所示。

```
# ldconfig -n /lib -v
/lib:                                     //对/lib 目录中的库文件进行更新
    libcrypto.so.2 -> libcrypto.so.0.9.6b
    libssl.so.2 -> libssl.so.0.9.6b
    libnss_db.so.1 -> libnss_db.so.1.0.0
    libnss_db.so.2 -> libnss_db.so.2.0.0
    libpwdb.so.0 -> libpwdb.so.0.62
    libdb-3.3.so -> libdb-3.3.so
    libdb-3.2.so -> libdb-3.2.so
    libdb.so.3 -> libdb2.so.3
    libdb-3.1.so -> libdb-3.1.so
    libcap.so.1 -> libcap.so.1.10        //用于捕获网络包的库文件
    libnss_wins.so -> libnss_wins.so.2
    libiw.so.25 -> libiw.so.25
    libnss_winbind.so -> libnss_winbind.so.2
    libnss_ldap.so.2 -> libnss_ldap-2.3.1.so
    libpam_misc.so.0 -> libpam_misc.so.0.75
    libpam.so.0 -> libpam.so.0.75
    libproc.so.2.0.11 -> libproc.so.2.0.11
    libpamc.so.0 -> libpamc.so.0.75
    liblvm-10.so.1 -> liblvm-10.so.1.0
    libtermcap.so.2 -> libtermcap.so.2.0.8
    libpcrc.so.0 -> libpcrc.so.0.0.1
    libgcc_s.so.1 -> libgcc_s-3.2.2-20030225.so.1
    libcrypto.so.4 -> libcrypto.so.0.9.7a
    libssl.so.4 -> libssl.so.0.9.7a     //SSL 库文件
    libacl.so.1 -> libacl.so.1.0.3
    libuuid.so.1 -> libuuid.so.1.2
    libattr.so.1 -> libattr.so.1.0.1
    libss.so.2 -> libss.so.2.0
    libext2fs.so.2 -> libext2fs.so.2.4
    libcom_err.so.2 -> libcom_err.so.2.0
    libdb-4.0.so -> libdb-4.0.so
    libutil.so.1 -> libutil-2.3.2.so
```



```
libthread_db.so.1 -> libthread_db-1.0.so      //用于线程处理的库文件
librt.so.1 -> librt-2.3.2.so
libresolv.so.2 -> libresolv-2.3.2.so
libpthread.so.0 -> libpthread-0.10.so
libnss_nisplus.so.2 -> libnss_nisplus-2.3.2.so
libnss_nis.so.2 -> libnss_nis-2.3.2.so
libnss_hesiod.so.2 -> libnss_hesiod-2.3.2.so
libnss_files.so.2 -> libnss_files-2.3.2.so
libnss_dns.so.2 -> libnss_dns-2.3.2.so      //DNS 库文件
libnss_compat.so.2 -> libnss_compat-2.3.2.so
libnsl.so.1 -> libnsl-2.3.2.so
libm.so.6 -> libm-2.3.2.so
libdl.so.2 -> libdl-2.3.2.so
libcrypt.so.1 -> libcrypt-2.3.2.so
libc.so.6 -> libc-2.3.2.so
libanl.so.1 -> libanl-2.3.2.so
libSegFault.so -> libSegFault.so
libBrokenLocale.so.1 -> libBrokenLocale-2.3.2.so
libe2p.so.2 -> libe2p.so.2.3
ld-linux.so.2 -> ld-2.3.2.so
/lib/i686: (hwcaps: 0x8000000000000000)      //对/lib/i686 目录中的库文件进行更新
    libm.so.6 -> libm-2.3.2.so
    libc.so.6 -> libc-2.3.2.so
    libpthread.so.0 -> libpthread-0.10.so
/lib/tls: (hwcaps: 0x8000000000000000)      //对/lib/tls 目录中的库文件进行更新
    libthread_db.so.1 -> libthread_db-1.0.so
    libm.so.6 -> libm-2.3.2.so
    libc.so.6 -> libc-2.3.2.so
    libpthread.so.0 -> libpthread-0.29.so
```

3.3.16 ldd 命令：显示共享库文件的相互依赖关系

ldd 命令可以显示命令行指定的程序或库文件所使用的共享库，以及其相互依赖关系。该命令的语法格式如下所示。

```
ldd [-v,--verbose] [-d, --data-relocs] [-r,--function-relocs] [--version] [--help]
```

其中各选项的具体说明如表 3-83 所示。

表 3-83 ldd 命令的选项说明

选 项	说 明
-v, --verbose	冗余模式，显示所有信息
-d, --data-relocs	执行数据对象重定位并报告任何缺失对象
-r,--function-relocs	对于数据对象和函数都实行重定位，并报告任何缺失对象
--version	显示 ldd 的版本信息
--help	显示帮助信息

例如，显示 “/bin/ls” 命令所使用的共享函数库，命令如下所示。

```
# ldd /bin/ls
libtermcap.so.2 => /lib/libtermcap.so.2 (0x40033000)
libc.so.6 => /lib/tls/libc.so.6 (0x42000000)
/lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
```



例如，以冗余模式显示命令“/bin/ls”所使用的共享库，命令如下所示。

```
# ldd -v /bin/ls
libtermcap.so.2 => /lib/libtermcap.so.2 (0x40033000)
libc.so.6 => /lib/tls/libc.so.6 (0x42000000)
/lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)

Version information:                               //版本信息
/bin/ls:
  libc.so.6 (GLIBC_2.2.3) => /lib/tls/libc.so.6
  libc.so.6 (GLIBC_2.1.3) => /lib/tls/libc.so.6
  libc.so.6 (GLIBC_2.1) => /lib/tls/libc.so.6
  libc.so.6 (GLIBC_2.3) => /lib/tls/libc.so.6
  libc.so.6 (GLIBC_2.2) => /lib/tls/libc.so.6
  libc.so.6 (GLIBC_2.0) => /lib/tls/libc.so.6
/lib/libtermcap.so.2:
  libc.so.6 (GLIBC_2.1.3) => /lib/tls/libc.so.6
  libc.so.6 (GLIBC_2.1) => /lib/tls/libc.so.6
  libc.so.6 (GLIBC_2.0) => /lib/tls/libc.so.6
/lib/tls/libc.so.6:
  ld-linux.so.2 (GLIBC_2.3) => /lib/ld-linux.so.2
  ld-linux.so.2 (GLIBC_2.1) => /lib/ld-linux.so.2
  ld-linux.so.2 (GLIBC_2.0) => /lib/ld-linux.so.2
  ld-linux.so.2 (GLIBC_PRIVATE) => /lib/ld-linux.so.2
```

3.3.17 lilo 命令：安装启动装载程序

lilo 命令用于安装一个启动装载程序，在下次启动时会运行，单独执行会根据/etc 目录下的 lilo.conf 配置文件安装 lilo。该命令的语法格式如下所示。

```
lilo [-b bootdev] [-c] [-d dsec] [-D label] [-i bootsector] [-f file] [-l] [-L] [-m
mapfile] [-P fix] [-P ignore] [-s file]
      [-S file] [-v] [-C config-file]
```

其中各选项的具体说明如表 3-84 所示。

表 3-84 lilo 命令选项说明

选 项	说 明
-b bootdev	指定安装 lilio 的设备
-c	压缩模式，将系统启动时所需要的文件放在连续的磁区里，可以增加读取效率
-d dsec	设置开机等待的时间。预设为 5s
-D label	采用指定的标识区分不同的内核选项
-i bootsector	将指定文件作为新的启动扇区使用，默认为/boot 目录里的 boot.b 文件
-f file	指定磁盘几何参数文件，默认为/etc 目录下的 disktab 文件
-l	采用线性扇区地址，而不是扇区/磁头/柱面方式
-L	采用 32 位逻辑地址，而不是柱面/磁头/扇区地址方式，允许访问大于 8.4GB 的分区
-m mapfile	通过参数指定映射文件，默认为/boot 下的 map
-P fix	修复破坏的分区表。例如分区的线性扇区地址表示，无法与柱面/磁头/扇区地址表示方式相对应
-P ignore	忽略破坏的分区表

续表

选 项	说 明
-s file	当 lilo 重写引导扇区时，指定备份文件。默认为/boot 目录里的文件 boot.xxxx，其中 xxxx 为数字代码
-S file	强制进行备份
-v	设置冗余模式，多个该参数连用可以提高冗余模式级别
-C config-file	指定 lilo 的配置文件，默认的文件为/etc/lilo.conf

例如，安装 lilo 到第一个 SCSI 硬盘的第 2 个主分区，采用 2 级冗余模式，命令如下所示。

```
# lilo -b /dev/sda2 -v -v
```

例如，指定安装 lilo 的配置文件为 lilo.conf，备份文件/boot/boot.backup，命令如下所示。

```
# lilo -C /etc/lilo.conf -s /boot/boot.backup
```

3.3.18 losetup 命令：设置和控制循环设备

losetup 命令用来设置和控制循环设备，可以建立循环设备与常规文件或块设备的连接。该命令的语法格式如下所示。

```
losetup [ -e encryption ] [ -o offset ] [ -d ][loop_device]
```

其中各选项的具体说明如表 3-85 所示。

表 3-85 losetup 命令的选项说明

选 项	说 明
-e encryption	设置加密方式，包括 NONE（不加密）、XOR（简单的 XOR 加密）、DES（DES 加密，需要 DES 模块已被加载到内核）
-d	取消循环设备与常规文件或块设备的连接
-o offset	数据在写入指定文件或设备之前的偏移量
loop_device	循环设备



如果不为循环设备指定参数，losetup 命令可以显示循环设备的状态。

例如，显示设备/dev/loop0 的状态，命令如下所示。

```
# losetup /dev/loop0 //显示设备/dev/loop0 的状态
loop: can't get info on device /dev/loop0: 没有那个设备或地址
```

3.3.19 mkraid 命令：初始化或升级 raid

mkraid 可以将一组块设备设置为一个 raid 设备阵列。该命令适合所有的 raid 类型，包括 RAID1、RAID4、RAID5、线性和 RAID0。该命令的语法格式如下所示。

```
mkraid [-c, --configfile filename] [-f, --force] [-h, --help] [-o, --upgrade] [-V, --version] </dev/md?>+
```

其中各选项的具体说明如表 3-86 所示。



表 3-86 mkraid 命令的选项说明

选 项	说 明
-c, --configfile filename	以指定文件作为配置文件，默认为/etc/raidtab
-f, --force	强制初始化设备，不管设备原有的数据
-h, --help	显示帮助信息
-o, --upgrade	将旧的 raid 阵列升级为当前内核 raid 版本
-V, --version	显示版本信息
</dev/md?>+	组成 raid 的设备，彼此之间以“+”分隔



初始化 raid 设备阵列将毁坏所有原数据。

例如，将设备/dev/sda1 和/dev/fd0 组成 raid 磁盘阵列，命令如下所示。

```
# mkraid -f /dev/sda1+/dev/fd0
```

3.3.20 modprobe 命令：自动处理可载入模块

modprobe 命令会根据模块内部的依赖关系自动安装所有依赖的模块。如果安装过程中发生错误，将卸载所有相关模块，而这些操作都是由模块自己声称的依赖关系决定的。该命令的语法格式如下所示。

```
modprobe [-a, --all] [-c, --showconfig] [-d, --debug] [-h, --help] [-k, --autoclean] [-l, --list] [-n, --show] [-q, --quiet]
          [-r, --remove] [-s, --syslog] [-t, --type moduletype] [-v, --verbose] [-V, --version] [-C, --config configfile]
```

其中各选项的具体说明如表 3-87 所示。

表 3-87 modprobe 命令的选项说明

选 项	说 明
-a, --all	加载所有匹配的模块
-c, --showconfig	显示当前配置
-d, --debug	显示调试信息
-h, --help	显示帮助信息
-k, --autoclean	对加载的模块设置自动清除
-l, --list	列出所有匹配的模块
-n, --show	仅演示操作，不实际应用
-q, --quiet	静默模式，不输出错误信息
-r, --remove	将卸载指定的模块，其依赖的模块也将一起被卸载
-s, --syslog	使用日志，而不是从标准错误设备输出报告

续表

选 项	说 明
-t, --type moduletype	仅查找指定类型的模块
-v, --verbose	冗余模式，显示所有的命令
-V, --version	显示版本信息
-C, --config configfile	使用指定的配置文件，而不是默认的/etc/modules.conf



modprobe 命令并不会对模块本身产生任何影响。

例如，安装网卡 8139too，命令如下所示。

```
# modprobe -v 8139too
/sbin/insmod /lib/modules/2.4.20-8/kernel/drivers/net/8139too.o
Using /lib/modules/2.4.20-8/kernel/drivers/net/8139too.o
Symbol version prefix ''
... ..
```

例如，显示当前的默认配置，命令如下所示。

```
# modprobe -c
# Generated by modprobe -c (2.4.22)
path[boot]=/lib/modules/boot
path[toplevel]=/lib/modules/2.4.20-8
path[toplevel]=/lib/modules/2.4
path[kernel]=/lib/modules/kernel
path[fs]=/lib/modules/fs
path[net]=/lib/modules/net
path[scsi]=/lib/modules/scsi
path[block]=/lib/modules/block
path[cdrom]=/lib/modules/cdrom
path[ipv4]=/lib/modules/ipv4
```

3.3.21 mouseconfig 命令：设置鼠标相关参数

mouseconfig 是 Red Hat Linux 下的鼠标配置工具，与命令 redhat-config-mouse 相同。该命令的语法格式如下所示。

```
mouseconfig [--help] [--noui] [--text] [--emulthree] [--device <dev>] [<mousetype>]
```

其中各选项的具体说明如表 3-88 所示。

表 3-88 mouseconfig 命令的选项说明

选 项	说 明
--help	显示帮助信息
--noui	在命令行模式中运行
--text	在文本界面模式中运行
--emulthree	模拟三键鼠标
--device <dev>	指定鼠标所在的端口。 <dev> 典型设备是 ttyS[0-3]，位于 PS/2 端口的 PS/2 鼠标则为 psaux
<mousetype>	指定鼠标类型，类型可以是表 3-89 所示的类型之一



表 3-89 mouseconfig 命令支持的鼠标类型

alpsps/2	asciips/2	ascii	atibm	genericps/2
genericusb	generic	generic3ps/2	generic3usb	generic3
genericwheels/2	genericwheelusb	geniusnmps/2	geniusnm	geniusprops/2
geniusscrollps/2	geniusscrollps/2+	thinkingps/2	thinking	logibm
logitech	logitechcc	- logimmusb	logimman+ps/2	logimman+
logimmanps/2	logimman	mmhittab	mmseries	msbm
microsoft	msintellips/2	msintelliusb	msintelli	msnew
mousesystems	none	sun		

例如，在图形界面方式下配置鼠标，命令如下所示。

```
# mouseconfig --text
```

执行该命令后，出现图 3-18 所示界面。

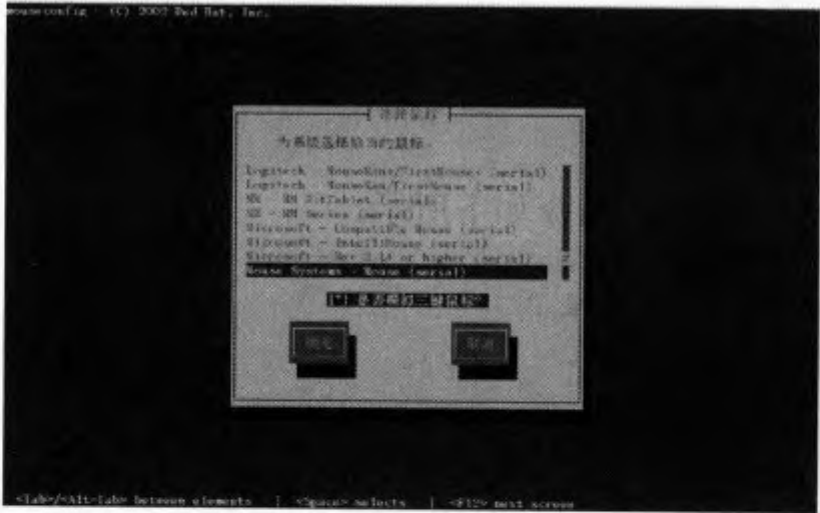


图 3-18 以图形方式运行 mouseconfig

例如，以命令形式指定鼠标类型为 genericusb，如下所示。

```
# mouseconfig --noui genericusb
```

关闭控制台鼠标服务： [确定]
启动控制台鼠标服务： [确定]

例如，指定鼠标所在的端口为 PS/2，鼠标为 psaux，命令如下所示。

```
#mouseconfig --device psaux
```

执行该命令后，出现图 3-19 所示界面。

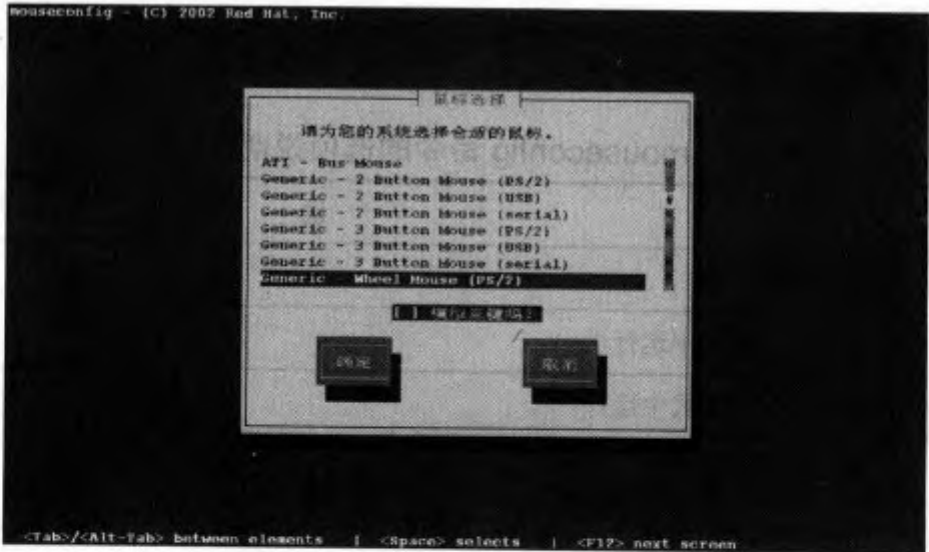


图 3-19 指定端口为 PS/2

3.3.22 resize 命令：设置终端视窗的大小

resize 命令用来设置虚拟终端的屏幕显示大小。该命令的语法格式如下所示。

```
resize [-u][-c ] [ -s [ row col ] ]
```

其中各选项的具体说明如表 3-90 所示。

表 3-90 resize 命令的选项说明

选 项	说 明
-c	采用 C shell 命令
-u	采用 Bourne Shell 命令
-s [rows columns]	指定屏幕大小，rows 表示行，columns 表示列

例如，使用 C shell 表示当前终端窗口大小，命令如下所示。

```
# resize -c //使用 C shell
set noglob;
setenv COLUMNS '80';
setenv LINES '21';
unset noglob;
```

例如，使用 Bourne Shell 表示当前终端窗口大小，命令如下所示。

```
# resize -u //使用 Bourne shell
COLUMNS=80;
LINES=21;
export COLUMNS LINES;
```

例如，设置虚拟终端的大小，高为 30 列，长为 100 个字符，命令如下所示。

```
# resize -s 30 100 //设置屏幕大小
COLUMNS=100;
LINES=30;
export COLUMNS LINES;
```

3.3.23 rpm 命令：管理 RPM 包

rpm 命令是一个功能强大的包管理器，它可以用来建立、安装、请求、确认、更新和卸载软件包。该命令的语法格式如下所示。

```
rpm[-c,--configfiles][-d,--docfiles][--dump][-l,--list][ --queryformat=QUERYFORMAT]
[-s,--state][-a,--all]
    [-f,--file][-g,--group][-p,--package][--specfile][--whatrequires][--whatprovides]
[--nomd5][ --nofiles]
    [--nodeps][--noscript][-a,--all][-f,--file][-g,--group][-p,--package][--specfile]
[ --addsign] [ --filecolor]
    [-K,--checksig][--import][--resign][--nodigest][--nosignature][--initdb]
[--rebuilddb][--aid][--allfiles]
    [--allmatches][--badreloc][-e,--erase=<package>+][--excludedocs][--excludepath
=<path>][--force]
    [-F,--freshen=<packagefile>+][-h,--hash][--ignorearch][--ignoreeos][--ignoresize]
[ -i, --install][ --justdb]
    [--nomd5][--noorder][--nosuggest][--noscripts][--notriggers][--oldpackage]
[--percent][ --prefix=<dir>]
    [--relocate=<old>=<new>][--repackage][--replacefiles][--replacepkgs][--test]
[-?,--help][ --usage]
```



```
[-U,--upgrade=<packagefile>+][-D,--define='MACROEXPR'][-E,--eval='EXPR']
[ --macros=<FILE:...>]
    [--nodigest][--nosignature][--rcfile=<FILE:...>][-r,--root=ROOT][--querytags]
[ --showrc][ --quiet]
    [-v,--verbose][--version][--scripts][--setperms][--setugids][--with=<option>]
[--without=<option>]
    [--conflicts][--obsoletes][--provides][--requires][--info][--changelog]
[--triggers][--last][--filerequire]
    [--filesbypkg][--fileclass]
```

其中各选项的具体说明如表 3-91 所示。

表 3-91 rpm 命令的选项说明

选 项	说 明
查询选项	
-c, --configfiles	列出所有配置文件
-d, --docfiles	列出所有文档文件
--dump	输出基本文件信息
-l, --list	列出包中的文件
--queryformat=QUERYFORMAT	使用指定的查询格式
-s, --state	显示列出文件的状态
-a, --all	查询/校验所有的包
-f, --file	查询/校验属于指定文件的包
-g, --group	查询/校验指定用户组的包
-p, --package	查询/校验包文件
--specfile	查询指定的文件
--whatrequires	查询/校验需要依赖关系的包
--whatprovides	查询/校验提供依赖关系的包
校验选项	
--nomd5	不对文件进行 MD5 校验
--nofiles	不对包中文件进行校验
--nodeps	不校验包的依赖性
--noscript	不执行校验脚本
-a, --all	查询/校验所有的包
-f, --file	查询/校验属于指定文件的包
-g, --group	查询/校验指定用户组的包
-p, --package	查询/校验包文件
--specfile	查询指定文件
--whatrequires	查询/校验需要依赖关系的包
--whatprovides	查询/校验提供依赖关系的包
签名选项	

续表

选 项	说 明
--addsign	对包进行签名（相当于--resign）
-K, --checksig	校验包签名
--import	引入公共密钥
--resign	对包进行签名（相当于--addsign）
--nodigest	不对包的摘要进行校验
--nosignature	不对包的签名进行校验
Database options:数据库选项	
--initdb	初始化数据库
--rebuilddb	从安装的包头中重建数据库反向列表
安装/升级/删除选项	
--aid	添加处理的包
--allfiles	安装所有的文件，甚至包括配置文件
--allmatches	删除所有匹配<package>的包（如果<package>指定了多个包，则产生错误）
--badreloc	在非重新定位包中重新定位文件
-e, --erase=<package>+	删除（卸载）包
--excludedocs	不安装文档资料
--excludepath=<path>	跳过指定路径的文件
--force	强制对包、文件进行替换
-F, --freshen=<packagefile>+	如果已安装，则进行升级
-h, --hash	在安装过程中显示哈希标识
--ignorearch	不对包的架构进行验证
--ignoreos	不校验包的操作系统
--ignoresize	在安装前不检验磁盘空间
-i, --install	安装包
--justdb	升级数据库但不修改文件系统
--nodeps	不对包的依赖性进行校验
--nomd5	不对文件进行 MD5 校验
--noorder	不依据依赖关系调整包的安装顺序
--nosuggest	不对丢失的依赖提出建议
--noscripts	不执行包中脚本
--notriggers	不执行由此包引发的脚本
--oldpackage	升级到包的一个旧版本（升级时使用选项“--force”默认具有此功能）
--percent	显示包安装进度
--prefix=<dir>	如果需要重新定位，重新定位包到目录<dir>中



续表

选 项	说 明
--relocate=<old>=<new>	重新定位文件路径，从<old>到<new>
--repackage	通过重新打包，保存删除的包文件
--replacefiles	即使包替换了安装文件，依然进行安装
--replacepkgs	如果包已经存在，重新安装
--test	不进行安装，仅进行测试
-U, --upgrade=<packagefile>+	升级包
通用选项	
-D, --define='MACRO EXPR'	定义宏
-E, --eval='EXPR'	显示宏扩展
--macros=<FILE:...>	从指定文件读取宏
--nodigest	不对包的摘要进行校验
--nosignature	不校验包的签名
--rcfile=<FILE:...>	读取指定的文件
-r, --root=ROOT	使用根目录（"/"）作为顶级目录
--querytags	显示已知的查询标签
--showrc	显示配置内容
--quiet	静默模式
-v, --verbose	冗余模式，输出详细信息
--version	显示当前的 rpm 版本
--scripts	列出包中的安装/删除脚本
--setperms	设置包中的文件权限
--setugids	设置包中文件的用户/用户组
--conflicts	列出包冲突
--obsoletes	列出安装过程中删除的过时的包
--provides	列出该包提供的容量
--requires	列出包需要的容量
--info	列出包的描述信息
--changelog	列出该包发生变化的日志
--triggers	列出该包的触发脚本
--last	依据安装时间、最近优先顺序列出包
--filesbypkg	列出每一包中的所有文件
--fileclass	列出文件名，且具有类别
--filecolor	列出文件名，且具有颜色
--filerequire	列出文件名，且包含需求

续表

选 项	说 明
--with=<option>	使用配置选项
--without=<option>	关闭配置选项
-, --help	显示帮助信息
--usage	显示用法摘要

例如，安装软件 RealPlay10 的 rpm 软件包 RealPlayer10GOLD.rpm，安装过程中显示安装进度，命令如下所示。

```
# rpm -hiv RealPlayer10GOLD.rpm //安装 RPM 包
Preparing... ##### [100%]
1:RealPlayer ##### [100%]
```

安装完成之后，系统就可以使用 RealPlayer 软件了。

例如，查询软件 RealPlay 的安装信息，命令如下所示。

```
# rpm -qi RealPlayer //显示软件的安装信息
Name      : RealPlayer           Relocations: /usr/local
Version   : 10.0.8.805          Vendor: RealNetworks, Inc
Release   : 20060718            Build Date: 2006 07 月 19 日 12:17:03
Install Date: 2007 05 月 28 日 20:38:03      Build Host: marlin
Group:Applications/Multimedia  Source RPM: RealPlayer-10.0.8.805-20060718.src.rpm
Size      : 15265081             License: RPSL, EULA
Signature : (none)
Packager  : RealNetworks, Inc
URL       : http://www.real.com/
Summary   : RealPlayer
Description:
RealPlayer is a media player with the aim of providing solid media
playback locally and via streaming. It plays RealAudio, RealVideo, MP3,
3GPP Video, Flash, SMIL 2.0, JPEG, GIF, PNG, RealPix and RealText and
more. The RealPlayer 10 for Linux builds on top of the popular
open-source Helix Player.
```

例如，卸载软件 RealPlay，命令如下所示。

```
# rpm -ev RealPlayer //卸载软件 RealPlayer
```

如果要连续卸载多个相关软件包，可以通过在命令行写一段简单的代码来完成。例如，删除系统中与软件 php 相关的所有已安装的软件包，可以使用下面命令。

```
#for i in `rpm -qa | grep -i php`
>do rpm -e $i --nodeps
>done
```

执行上面命令后，即对多个与 php 相关的软件包进行了统一删除。

3.3.24 setserial 命令：设置或显示串口的信息

setserial 命令可以显示当前系统串口的设置，也可以设置当前串口的参数。该命令的语法格式如下所示。

```
setserial [-a][-b][-G][-v][-V] [device]
```

其中各选项的具体说明如表 3-92 所示。



表 3-92 setserial 命令的选项说明

选 项	说 明
-a	显示详细的信息
-b	当显示串口设备配置信息时，同时显示设备配置信息的统计结果
-G	显示能够给命令 setserial 进行反馈的串口配置信息
-v	运行时显示详细的处理信息
-V	显示版本信息
-q	静默模式
device	串口设备

例如，显示串口/dev/ttyS1 和/dev/ttyS2 的配置信息，命令如下所示。

```
# setserial -v /dev/ttyS1
/dev/ttyS1, UART: 16550A, Port: 0x02f8, IRQ: 3
# setserial -v /dev/ttyS2
/dev/ttyS2, UART: unknown, Port: 0x03e8, IRQ: 4
```

例如，显示串口/dev/ttyS1 的详细配置信息，命令如下所示。

```
# setserial -a /dev/ttyS1
/dev/ttyS1, Line 1, UART: 16550A, Port: 0x02f8, IRQ: 3
    Baud_base: 115200, close_delay: 50, divisor: 0
    closing_wait: 3000
    Flags: spd_normal skip_test
```

3.3.25 setup 命令：设置公用程序

setup 命令用来设置认证、防火墙、鼠标、网络配置、打印机配置、系统服务、时区配置等公用程序。

例如，使用 setup 命令进行网络设置，如下所示。

```
# setup
```

运行命令 setup 后，将出现图 3-20 所示的界面。

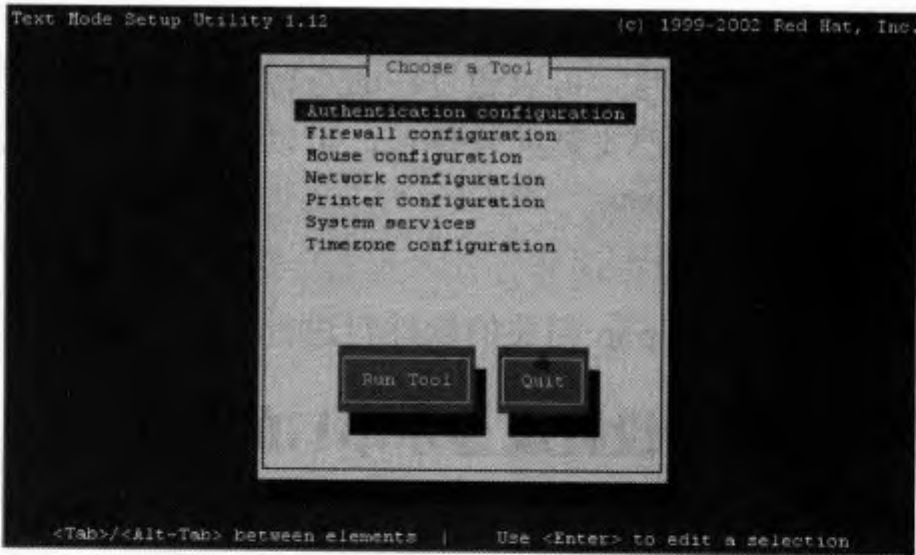


图 3-20 配置工具选择菜单

在图 3-19 中选择【Network configuration】，直接回车或者按 Tab 键选择【Run Tool】再

回车，将会出现图 3-21 所示界面。

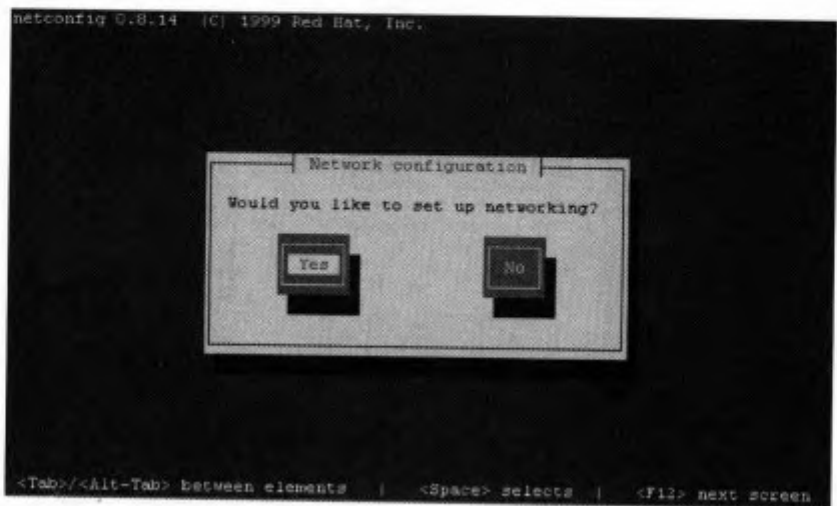


图 3-21 是否进行网络配置

图 3-21 所示界面中提示用户将要进行网络配置，单击【Yes】按钮，进入下一步，出现图 3-22 所示的网络配置界面。

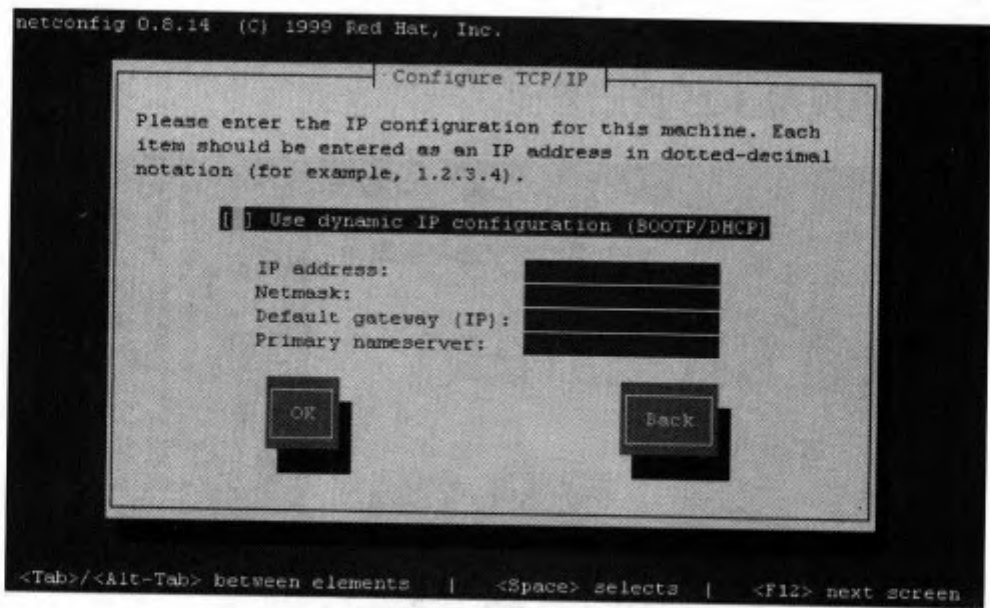


图 3-22 网络配置界面

在图 3-22 所示界面中进行网络的相关配置，如图 3-23 所示。

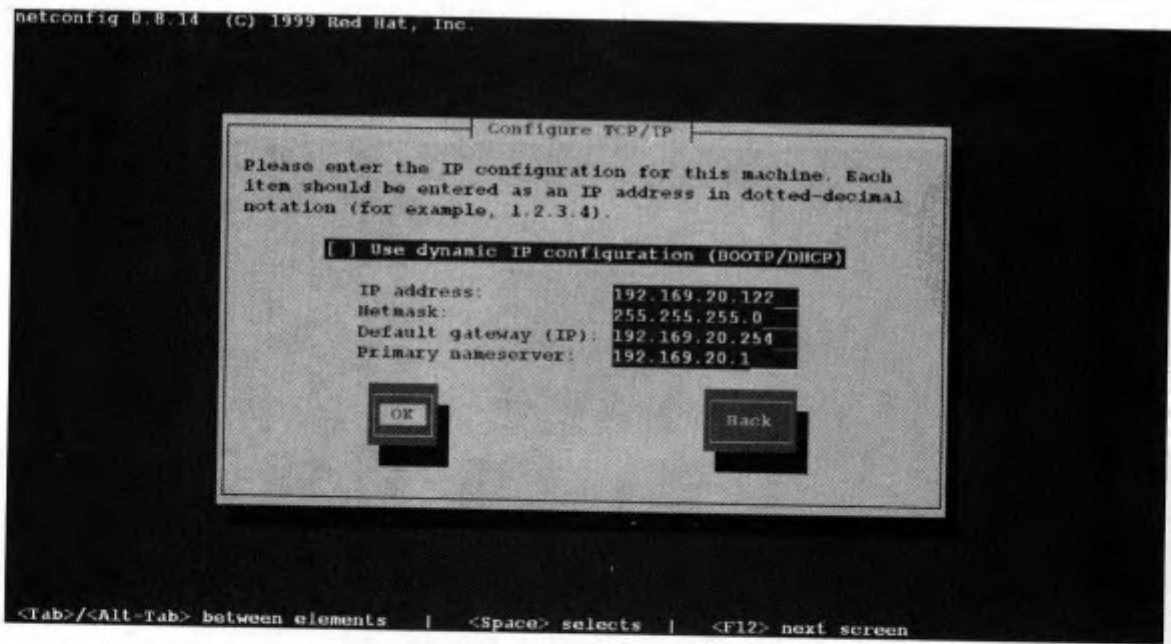


图 3-23 对网络进行配置



单击【OK】按钮，按下回车键完成配置返回初始的配置界面，如图 3-24 所示。

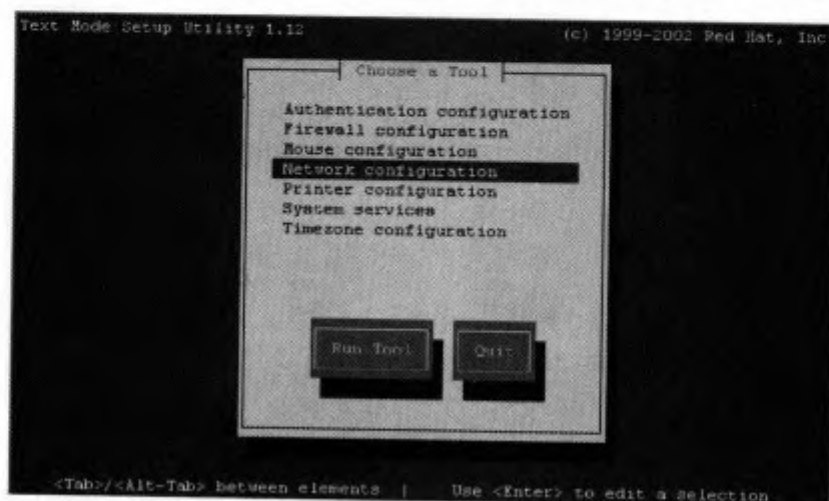


图 3-24 返回初始配置界面

此时，网络配置已经完成，若不再进行其他的配置，则单击【Quit】按钮，退出 setup 设置。

例如，使用 setup 命令进行系统服务配置，首先执行如下所示命令

```
# setup //设置
```

运行命令 setup 后，出现图 3-25 所示界面。

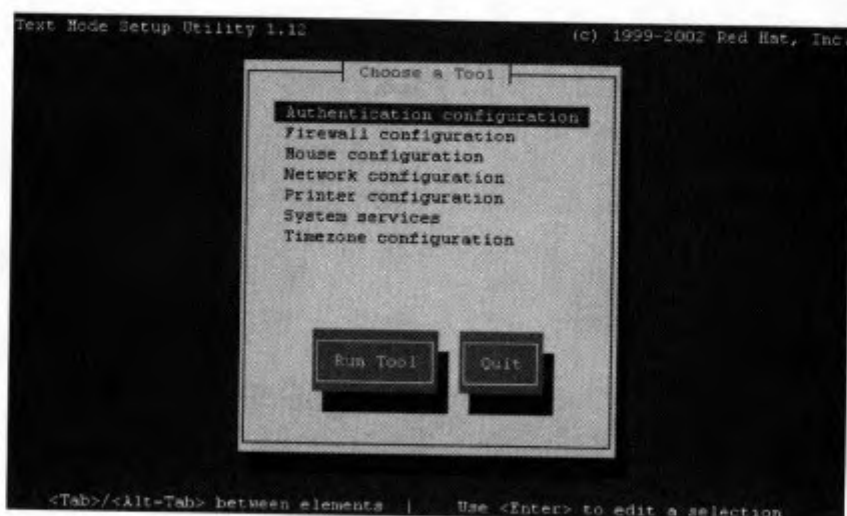


图 3-25 配置工具主菜单

选择【System services】选项，直接按回车键或按 Tab 键选择【Run Tool】再按回车键，出现图 3-26 所示的系统服务配置界面。

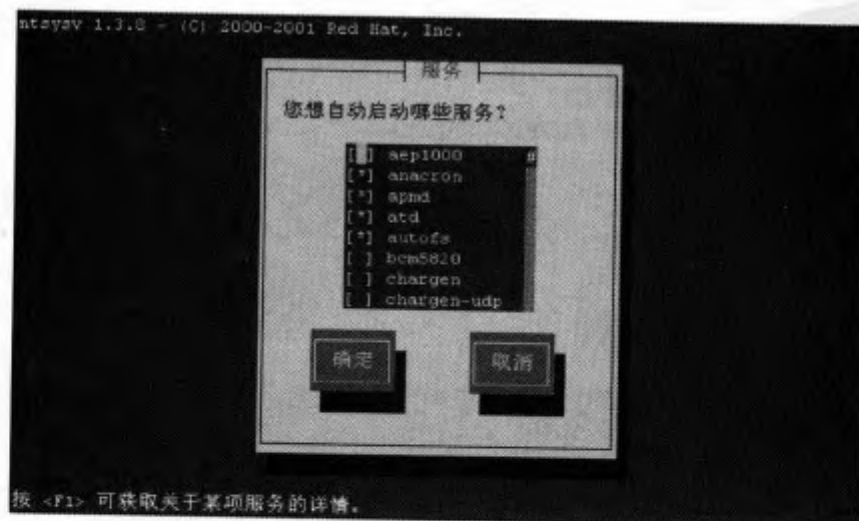


图 3-26 选择启动的服务

选择要启用的服务，前面的“[]”内有“*”字符的选项代表该服务已启动。将光标移到相应的服务选项，反复按空格键可以完成对该服务的选中或取消选中。按 F1 键可以显示服务的具体帮助信息，如图 3-27 所示。

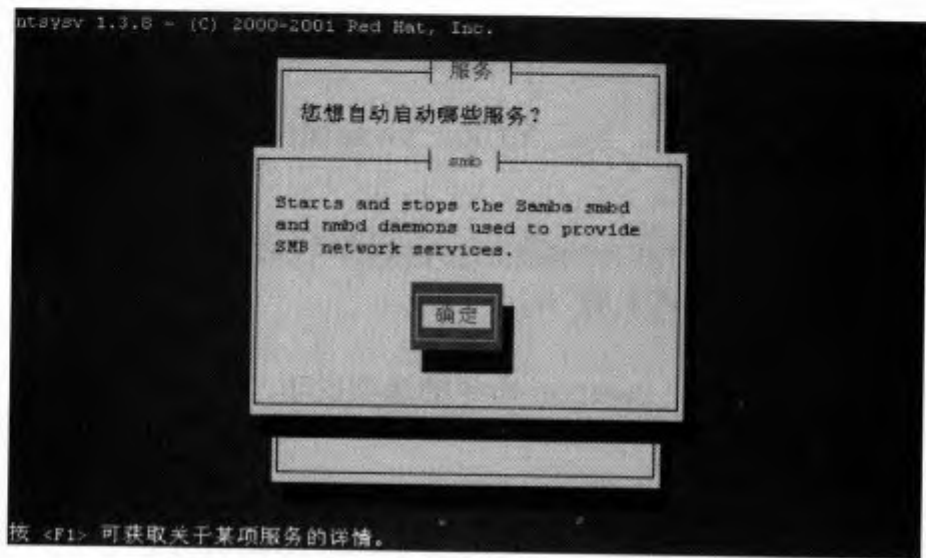


图 3-27 smb 服务的帮助信息

选择完成之后，连续按【确定】按钮，然后退出，即可完成配置。



依照上述方法，可以类似地完成其他配置，如认证配置、防火墙配置、鼠标配置、打印机配置、时区配置等。

3.3.26 swapoff 命令：卸载交换区（swap area）

swapoff 命令用来关闭交换分区。与之相反，swapon 命令用来打开交换分区。swapoff 命令的语法格式如下所示。

```
swapoff [-h] [-V] [specialfile]
```

其中各选项的具体说明如表 3-93 所示。

表 3-93 swapoff 命令的选项说明

选 项	说 明
-a	卸载所有已知的交换分区（根据/proc/swaps 或/etc/fstab 中的记录）
specialfile	指定的文件系统
-h	帮助信息
-V	版本信息

例如，使用 fdisk 命令显示系统当前的分区信息，如下所示。

```
# fdisk -l //显示分区信息
Disk /dev/sda: 8589 MB, 8589934592 bytes
255 heads, 63 sectors/track, 1044 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

Device      Boot  Start      End  Blocks  Id System
/dev/sda1   *        1      13    104391  83  Linux
```



```
/dev/sda2      14      979  7759395  83  Linux
/dev/sda3      980     1044   522112+ 82  Linux swap
```

可以看到，/dev/sda3 为交换分区。使用 swapoff 命令卸载交换分区，如下所示。

```
# swapoff /dev/sda3                                //卸载交换分区
```

3.3.27 swapon 命令：加载交换区 (swap area)

swapon 命令用来启动交换分区。与之相反，swapoff 命令用来关闭交换分区。swapon 命令的语法格式如下所示。

```
swapon [-a ][-v] [-e][ -p priority] [ specialfile]
```

其中各选项的具体说明如表 3-94 所示。

表 3-94 swapon 命令的选项说明

选 项	说 明
-a	加载所有在/etc/fstab 中标识为交换分区的分区，但跳过已经加载的分区
-c	当使用-a 时，让 swapon 跳过不存在的设备
-s	显示交换分区使用的信息
-h	显示帮助信息
-V	显示版本信息
-p priority	定义优先级，优先级介于 0~32767 之间

例如，使用 fdisk 命令显示系统当前的分区信息，如下所示。

```
# fdisk -l                                           //显示分区信息
Disk /dev/sda: 8589 MB, 8589934592 bytes
255 heads, 63 sectors/track, 1044 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
```

```
Device      Boot      Start         End      Blocks   Id  System
/dev/sda1   *           1          13       104391   83  Linux
/dev/sda2             14          979     7759395   83  Linux
/dev/sda3             980         1044     522112+   82  Linux swap
```

可以看到，/dev/sda3 为交换分区。使用 swapon 命令加载交换分区/dev/sda3，如下所示。

```
# swapon /dev/sda3                                //加载交换分区
```

例如，显示交换分区/dev/sda3 的信息，命令如下所示。

```
# swapon -s /dev/sda3                             //显示交换分区信息
Filename                                     Type      Size      Used      Priority
/dev/sda3                                   partition 522104      0         -2
```

3.3.28 set 命令：设置 Shell 执行方式

set 命令可以用来设置 Shell 的执行方式，也可以用来显示和设置环境变量。该命令的语法格式如下所示。

```
set [-a][-b][-e][-f][-h][-k][-m][-n][-p][-t][-u][-v][-x] [P][ -H Shell]
```

其中各选项的具体说明如表 3-95 所示。

表 3-95 set 命令的选项设置

选 项	说 明
-a	显示修改的变量
-b	显示程序执行情况
-x	在执行时显示命令名及参数
-e	出现错误时退出 Shell
-f	禁止使用通配符
-h	记录被搜索命令的位置
-k	环境变量设置
-l	for 循环设置
-m	monitor 模式
-n	测试模式
-p	优先模式
-P	用符号链接指向的实际文件代替符号链接
-t	命令执行完毕，就退出 Shell
-u	不执行未定义的变量
-v	运行时，显示详细的调试信息
-H Shell	执行历史命令
+	取消相应选项设置



Shell 定义了 3 种通配符。其中 “*” 匹配任意长度的任意字符，“?” 匹配任意单个字符，“[]” 匹配括号中的任意一个字符。

例如，显示当前的环境变量设置，命令如下所示。

```
# set
BASH=/bin/bash
BASH_ENV=/root/.bashrc
BASH_VERSINFO=([0]="2"[1]="05b"[2]="0"[3]="1"[4]="release"[5]="i386-redhat-linux-gnu")
BASH_VERSION='2.05b.0(1)-release'
COLORS=/etc/DIR_COLORS.xterm
COLUMNS=80
DIRSTACK=()
EUID=0
REMOTEHOST=192.168.20.148
SHELL=/bin/bash
SHELLOPTS=braceexpand:emacs:hashall:histexpand:history:interactive-comments:monitor
SHLVL=1
... ..
```

例如，通过命令 “set SHELL “/bin/sh” ”，设置环境变量 SHELL 的值为 “/bin/sh”，执行如下所示命令。

```
# set SHELL "/bin/sh" //设置环境变量
```



例如，下面命令中，使用“-f”取消通配符设置，使用“+f”添加了通配符设置。

```
# ll                                     //显示文件列表
总用量 24
-rw-r--r--  1 root    root      17  5月 29 09:45 a001
-rw-r--r--  1 root    root      17  5月 29 09:45 a002
-rw-r--r--  1 root    root      17  5月 29 09:45 a003
-rw-r--r--  1 root    root      17  5月 29 09:45 a004
-rw-r--r--  1 root    root      17  5月 29 09:46 a005
-rw-r--r--  1 root    root      17  5月 29 09:46 a00
# set -f                                 //取消通配符设置
# rm a000?                               //删除文件
rm: 失败: 没有那个文件或目录           //由于取消了通配符设置，无法解析 a000?
# set +f                                 //设置通配符
# rm a000?                               //删除文件
# ll
总用量 4
-rw-r--r--  1 root    root      17  5月 29 09:46 a00
```

3.3.29 telinit 命令：设置系统的运行等级

telinit 命令实质是/sbin/init 命令的链接，可以通知 init 命令设置当前系统的运行等级或执行相应的操作。该命令的语法格式如下所示。

```
telinit [0, 1, 2, 3, 4, 5, 6][S,s][Q,q][U,u]
```

其中各选项的具体说明如表 3-96 所示。

表 3-96 telinit 命令的选项说明

选 项	说 明
0~6	通知 init 命令切换系统的运行等级
Q,q	通知 init 命令重新检测/etc/inittab
S,s	通知 init 命令进入单用户模式
U,u	通知 init 命令重新执行

例如，首先使用 runlevel 命令显示当前系统的运行等级，然后使用 telinit 命令将当前的运行等级切换到 2，命令如下所示。

```
# runlevel                               //显示当前的执行级别
N 5
# telinit 2                             //设置执行级别
# runlevel                               //显示执行等级
5 2
```

3.3.30 true 命令：返回值为成功

true 命令什么事情也不做，直接返回一个状态码，且状态码表示成功，值为“0”。例如，执行 true 命令，其返回值为“0”，如下所示。

```
# ls kdjf                               //ls 命令参数错误
ls: kdjf: 没有那个文件或目录
# echo $?                               //返回上次命令的执行结果
1                                       //值为 1，表示命令没有正确执行
# true                                  //设置状态码值为 0
```



```
# echo $? //返回上次命令的执行结果
0 //值为 0，表示成功
```

3.3.31 ulimit 命令：控制 Shell 及进程的资源

ulimit 命令用来控制 Shell 以及 Shell 下各进程的系统资源。该命令的语法格式如下所示。

```
ulimit [-S][-H][-a][-c][-d][-f][-l][-m][-n][-p][-s][-t][-u][-v] [limit]
```

其中各选项的具体说明如表 3-97 所示。

表 3-97 ulimit 命令的选项说明

选 项	说 明
-S	定义软限制，当达到软限制时，开始报警，但仍可继续增加，直到达到硬限制
-H	定义硬限制，当达到硬限制时，无法继续添加
-a	显示当前所有资源的设置
-c	系统 core 文件大小的最大值
-d	进程数据段大小的最大值
-f	由 Shell 创建的文件大小的最大值
-l	内存锁定的最大值
-m	常驻内存大小的最大值
-n	打开文件数的最大值（绝大多数操作系统不允许设置该值）
-p	管道的大小限制在 512 字节块
-s	堆栈大小的最大值
-t	每秒 CPU 时间的最大值
-u	单用户模式下允许创建进程的最大数
-v	可由 Shell 使用的最大虚拟内存数
limit	指定相关资源的限制



资源限制一般可以是指定的数值，也可以使用 hard（硬限制）、soft（软限制）和 unlimited（无限制）。

例如，显示系统资源配置，命令如下所示。

```
# ulimit -a //显示系统资源设置
core file size (blocks, -c) 0
data seg size (kbytes, -d) unlimited
file size (blocks, -f) unlimited
max locked memory (kbytes, -l) unlimited
max memory size (kbytes, -m) unlimited
open files (-n) 1024
pipe size (512 bytes, -p) 8
stack size (kbytes, -s) 8192
cpu time (seconds, -t) unlimited
max user processes (-u) 2048
virtual memory (kbytes, -v) unlimited
```

例如，设置单用户允许创建进程数目的上限为 500，命令如下所示。



```
# ulimit -u 500 //设置单用户创建进程上限为 500
# ulimit -a //显示系统资源配置
core file size (blocks, -c) 0
data seg size (kbytes, -d) unlimited
file size (blocks, -f) unlimited
max locked memory (kbytes, -l) unlimited
max memory size (kbytes, -m) unlimited
open files (-n) 1024
pipe size (512 bytes, -p) 8
stack size (kbytes, -s) 8192
cpu time (seconds, -t) unlimited
max user processes (-u) 500 //最大值为 500
virtual memory (kbytes, -v) unlimited
```

例如，设置同一时间允许打开的文件数目的最大值为 1000，命令如下所示。

```
# ulimit -n 1000 //设置同一时间可打开的文件数目
[root@localhost root]# ulimit -a //显示系统资源配置
core file size (blocks, -c) 0
data seg size (kbytes, -d) unlimited
file size (blocks, -f) unlimited
max locked memory (kbytes, -l) unlimited
max memory size (kbytes, -m) unlimited
open files (-n) 1000 //最大值为 1000
pipe size (512 bytes, -p) 8
stack size (kbytes, -s) 8192
cpu time (seconds, -t) unlimited
max user processes (-u) 500
virtual memory (kbytes, -v) unlimited
```

例如，设置由 Shell 使用的最大虚拟内存为 1M，命令如下所示。

```
# ulimit -v 1024 //设置最大虚拟内存数
# ulimit -a //显示系统资源配置
core file size (blocks, -c) 0
data seg size (kbytes, -d) unlimited
file size (blocks, -f) unlimited
max locked memory (kbytes, -l) unlimited
max memory size (kbytes, -m) unlimited
open files (-n) 1000
pipe size (512 bytes, -p) 8
stack size (kbytes, -s) 8192
cpu time (seconds, -t) unlimited
max user processes (-u) 500
virtual memory (kbytes, -v) 1024 //最大值为 1024
```

3.3.32 unalias 命令：取消指定的别名

unalias 命令用来删除指定的别名，其语法格式如下所示。

```
unalias [-a] [name ...]
```

其中各选项的具体说明如表 3-98 所示。

表 3-98 unalias 命令的选项说明

选 项	说 明
-a	删除所有别名
name	取消的别名
--help	显示 unalias 命令的帮助信息

例如，给命令 `tcpdump` 命令设置别名“`td`”，然后使用 `unalias` 命令删除该别名，如下所示。

```
# alias td=tcpcdump //设置别名
# alias td //显示别名信息
alias td='tcpcdump'
# unalias td //删除别名
# alias td //显示别名
-bash: alias: td: not found //已删除[
```

3.3.33 unset 命令：删除变量或函数

`unset` 命令用来删除指定的变量或函数。该命令的语法格式如下所示。

```
unset [-f] [-v] [name ...]
```

其中各选项的具体说明如表 3-99 所示。

表 3-99 unset 命令的选项说明

选 项	说 明
-f	name 为 Shell 函数，删除 Shell 函数
-v	name 为 Shell 变量，删除 Shell 变量，无法删除只读变量
--help	显示帮助信息

例如，删除环境变量 `PWD`，命令如下所示。

```
# set //显示环境变量
BASH=/bin/bash
BASH_ARGC=()
BASH_ARGV=()
BASH_LINENO=()
... ..
PWD=/root
... ..
# unset PWD //删除环境变量
# set //显示环境变量
BASH=/bin/bash
BASH_ARGC=()
BASH_ARGV=()
BASH_LINENO=()
... ..
_=PWD //环境变量 PWD 已删除
```

3.3.34 up2date 命令：升级 Red Hat Linux 系统

`up2date` 命令用来升级 Red Hat Linux 系统，使系统保持最新状态。该命令的语法格式如下所示。

```
up2date [--configure] [-d,--download] [-i,--install] [--src] --nosrc [--dbpath=<dir>] [-f,
--force] [ --justdb] [ -l, --list]
[ --dry-run] [-?,--usage] [--nosig] [--nox] [-p,--packages] [--hardware]
[ --register] [ --showall] [ --version]
[ --show-orphans] [--solvedeps=<deps>] [--tmpdir=<temp-dir>] [--undo] [ -u,
--update] [--needsUp2date]
[ --show-channels] [--list-rollbacks] [--installall] [--channel=<channel>]
[--nodownload] [ -v, --verbose]
```



```
[--proxyUser=<proxyuserid>] [--proxyPassword=<proxypassword>] [--serverUrl=<url>] [-h, --help]
[--show-available] [--proxy=<proxyurl>] [--show-package-dialog]
```

其中各选项的具体说明如表 3-100 所示。

表 3-100 up2date 命令的选项说明

选 项	说 明
--configure	配置更新代理选项
-d, --download	仅下载更新包，不进行安装，即使配置文件中设定进行安装
-i, --install	安装更新，即使配置文件中设定仅进行下载
--src	下载二进制 rpm 升级包的同时，下载源码包
--nosrc	不下载源码包，即使配置文件中设定下载
--dbpath=<dir>	指定一个 rpm 替换数据库的路径
-f, --force	强制安装
--justdb	仅将包添加到数据库，不安装到系统中
-l, --list	列出可用的包
--dry-run	执行除下载和安装之外的所有操作，例如显示依赖关系
-, --usage	使用用法的简洁描述
--nosig	不使用 GPG 检测包签名
--nox	不使用 X
-p, --packages	只升级与当前系统相关的软件包
--hardware	升级与当前系统相关的硬件描述
--register	注册系统（如果需要，再次注册）
--showall	列出可供下载的所有包
--show-available	列出当前没有安装的所有可用的包
--show-orphans	列出没有依据系统定制要求安装的包
--solvedeps=<deps>	依据依赖关系，搜索、下载并进行安装
--tmpdir=<temp-dir>	指定临时文件存放位置
--undo	恢复最近一次包的升级操作
-u, --update	升级系统以及所有相关包
--version	显示版本信息
-v, --verbose	冗余模式，显示详细信息
--show-channels	显示通道名称以及相应的包名称
--list-rollbacks	显示可回溯的 rpm
--installall	安装所有可用的包，与--channel 联合使用
--channel=<channel>	指定使用哪一个通道
--nodownload	不下载，只测试
--proxy=<proxyurl>	指定使用的 http 代理

续表

选 项	说 明
--proxyUser=<proxy userid>	指定一个用户名，作为可被认证的 http 代理
--proxyPassword=<proxy password>	指定认证 http 代理使用的密码
--serverUrl=<url>	指定服务器的 URL
--needsUp2date	设定首先升级 up2date 软件包本身
--show-package-dialog	以图形方式显示包安装对话框
-h, --help	显示帮助信息

例如，更新系统，命令如下所示。

```
# up2date //更新系统
0. debug No
1. useRhn No
2. rhnuuid 3df31014-915f-11db-8781-af09d047da8a
3. isatty Yes
4. showAvailablePacka No
5. useNoSSLForPackage No
6. storageDir /var/spool/up2date
7. pkgSkipList ['kernel*']
8. retrieveOnly No
9. noSSLServerURL http://xmlrpc.rhn.redhat.com/XMLRPC
10. networkRetries 5
11. pkgsToInstallNotUp ['kernel', 'kernel-modules', 'kernel-devel']
12. enableProxy No
13. noBootLoader No
14. proxyPassword
15. updateUp2date Yes
16. keepAfterInstall No
17. useGPG Yes
18. headerCacheSize 40
19. forceInstall No
20. systemIdPath /etc/sysconfig/rhn/systemid
21. retrieveSource No
22. enableRollbacks No
23. gpgKeyRing /etc/sysconfig/rhn/up2date-keyring.gpg
24. adminAddress ['root@localhost']
25. serverURL https://xmlrpc.rhn.redhat.com/XMLRPC
26. fileSkipList []
27. versionOverride
28. sslCACert /usr/share/rhn/RHNS-CA-CERT
29. noReplaceConfig Yes
30. enableProxyAuth No
31. disallowConfChange ['noReboot', 'sslCACert', 'useNoSSLForPackages', 'noSSLSe
32. headerFetchCount 10
33. proxyUser
34. removeSkipList ['kernel*']
35. httpProxy
36. noReboot No
```

Enter number of item to edit <return to exit, q to quit without saving>: //从以上列表中选择



可以根据提示，一步步完成系统的更新。
例如，列出所有可用的升级包，命令如下所示。

```
# up2date -l
```

3.4 自动作业处理

在 Red Hat Linux 9 系统中，可以使用“自动作业程序”来设置系统在某个时间执行特定的命令和进程。自动作业程序可以帮助系统管理员自动地执行数据备份、病毒扫描、检查邮箱，以及删除不必要的文件等工作。通过定期执行设定的任务，系统可以实现自动更新和维护。实现自动作业处理的命令包括 `crontab`、`anacron`、`at` 和 `batch` 等。

3.4.1 crontab 命令：设置定时执行

`crond` 是 Linux 中一个重要的 Daemon（守护进程），启动后该程序会常驻内存并定期启动设定的程序。

检查系统中是否已安装 `crond` 服务程序，可以使用下面的命令。

```
# rpm -qa crontabs
crontabs-1.10-5 //已安装
```

启动 `crond` 服务程序，可以使用下面的命令。

```
# /sbin/service crond start
启动 crond: [ 确定 ]
```

停止 `crond` 服务程序，可以使用下面的命令。

```
# /sbin/service crond stop
停止 crond: [ 确定 ]
```

重新启动 `crond` 服务程序，可以使用下面的命令。

```
# /sbin/service crond restart
停止 crond: [ 确定 ]
启动 crond: [ 确定 ]
```

如果 `crond` 服务程序已经启动，则会显示如下错误信息。

```
# /sbin/service crond start
启动 crond: [ 失败 ]
```

`crontab` 命令可以添加、删除、列表显示用于驱动 `crond` 服务的各种任务，该命令的语法格式如下所示。

```
crontab [-u user][ -e ][ -l ][ -r ]
```

其中各选项的具体说明如表 3-101 所示。

表 3-101 crontab 命令的选项说明

选 项	说 明
-e	编辑用户的 <code>crontab</code> 文件
-l	列出用户在 <code>crontab</code> 中设定的任务
-r	删除用户在 <code>crontab</code> 中设定的任务
-i	交互式模式，在删除用户设定的任务前进行提示
-u user	指定用户的 <code>crontab</code> 文件被启动



每一个用户都可以拥有自己的 crontab 文件。

可以通过编辑/etc/crontab 文件，或者直接使用“crontab -e”命令设置 crond。若使用的是“crontab -e”命令进行编辑，则在编写完毕后，系统默认的存储位置是/tmp。可以通过使用“crontab /etc/crontab”重新指定到/etc/crontab。下面是一个 crontab 文件的内容示例。

```
# vi /etc/crontab
SHELL=/bin/bash
PATH=/sbin:/bin:/usr/sbin:/usr/bin
MAILTO=root
HOME=/
# run-parts          //下面部分是Cron定期执行的程序
01 * * * * root run-parts /etc/cron.hourly
02 4 * * * root run-parts /etc/cron.daily
22 4 * * 0 root run-parts /etc/cron.weekly
42 4 1 * * root run-parts /etc/cron.monthly
```

其中共包括 9 行记录：第 1 行表示执行任务时使用的 Shell，第 2 行表示执行任务时系统的搜索路径，第 3 行表示将执行结果 mail 给 root，第 4 行指定主目录为根，第 5 行是注释，第 6 行表示每小时的第 1 分钟以 root 身份执行/etc/cron.hourly 中的所有执行文件，第 7 行表示每天的 4 点 02 分以 root 身份执行/etc/cron.daily 中的所有执行文件，第 8 行表示每周日的 4 点 22 分以 root 身份执行/etc/cron.weekly 中的所有执行文件，第 9 行表示每月 1 日的 4 点 42 分以 root 身份执行/etc/cron.monthly 中的所有执行文件。

可以看到，/etc/crontab 文件中的任务记录有专门的格式，具体格式设定如下。

[分钟] [小时] [日期] [月份] [星期] [用户] [命令]

其中包括 7 个字段，各字段之间要以空格或 Tab 键隔开。每一行必须以回车结束，如果不加回车，crond 服务程序将忽略该行，不执行该行设定的任务。其中各字段说明见表 3-102。

表 3-102 crontab 文件各字段说明

字 段	说 明	示 例
分钟	每小时的第几分钟执行	取值为 0~59
小时	每天的第几小时执行	取值为 0~23
日期	每月的第几天执行	取值为 0~31
月份	每年的第几个月执行	取值为 0~12 或英文缩写，如 May、Feb、Nov 等
星期	每周的第几天执行	取值为 0~6 或英文缩写，如 Sun、Mon、Tue 等
用户	执行该命令的用户身份	root 等
命令	定期执行的命令	如显示时间：date

在时间域中，可以使用“-”符号代表一段时间，例如在小时域中输入“6-12”，表示每小时的第 6、7、8、9、10、11、12 分钟；可以使用“*”表示全部时间，例如在日期字段输入“*”，则表示每一个月的每一天都执行该命令；使用“,”表示特定的时间，例如在月份字段输入“3, 5, 12”，则表示每一年的 3 月、5 月和 12 月；使用“/”表示“每隔”，例如在



分钟字段输入“*/5”，则表示每隔 5 分钟。

修改 crontab 文件之后，不需要重新启动 crond 服务程序。crond 服务程序会自动根据 crontab 文件的内容刷新任务列表。

例如，使用“crontab -e”添加新的任务，要求每隔 5 分钟将系统时间写入 /myshare/cron_test 文件中，编辑命令行为：

```
* /5 * * * * date >> /myshare/cron_test
```

查看/myshare/cron_test 文件，可以看到时间已经自动写入该文件如下所示。

```
# cat /myshare/cron_test
Wed Aug 8 07:08:02 CST 2007
Wed Aug 8 07:09:01 CST 2007
```

例如，在每月的 3 日 23:30 自动删除/var/log/httpd 目录下的所有文件，命令如下所示。

```
30 23 3 * * root rm -f /var/log/httpd/*
```

例如，每隔 5 分钟查询一次系统当前运行的进程，并将查询结果保存到/myshare/cron_test 文件中，命令行如下所示。

```
* /5 * * * * ps -aux > /myshare/cron_test
```

例如，每周一的 1:00、3:00、8:00 各查询一次根目录结构，并将查询结果保存到 /myshare/cron_test 文件中，命令行如下所示。

```
0 1,3,8 * * Mon ls -al / > /myshare/cron_test
```

不带“-u”选项使用 crontab 命令时，系统默认为当前用户创建任务；如果为其他用户设定任务，只需在“-u”后指定用户。例如，编辑用户 teacher1 的任务，可以使用如下命令。

```
# crontab -e -u teacher1
```

使用“-l”选项可以列出所有任务列表。例如，列出 root 用户的所有任务列表，命令如下。

```
# crontab -l -u root
SHELL=/bin/bash
PATH=/sbin:/bin:/usr/sbin:/usr/bin
MAILTO=root
HOME=/
# run-parts
01 * * * * root run-parts /etc/cron.hourly
02 4 * * * root run-parts /etc/cron.daily
22 4 * * 0 root run-parts /etc/cron.weekly
*/1 * * * * date >> /myshare/cron_test
42 4 1 * * root run-parts /etc/cron.monthly
```

为了增强 crond 服务程序的安全性，系统使用/etc/cron.allow 和/etc/cron.deny 文件对 crontab 文件的存取进行管理。如果/etc/cron.allow 文件存在，则只有该文件列出的用户可以使用 crond。如果/etc/cron.deny 文件存在，则在该文件中列出的用户不可以使用 crond。对于 root 用户，无论是否包含在这两个文件中，都有使用 crond 服务的权力。

crond 服务程序经常被用来帮助系统管理员定期清除垃圾文件。例如，Linux 中的 core 文件经常占用大量磁盘空间，可以编辑 crontab 文件，以实现每周一早上 2:00 自动删除所有一周以来没有访问过的 core 文件，命令如下所示。

```
0 2 * * Mon find / -name core -atime +7 -exec rm -f { } \;
```



上面命令中“{”与“\”之间要有空格。

3.4.2 anacron 命令：任务调度守护进程

anacron 命令与 crond 类似，属于任务调度工具，也是 Linux 中一个重要的守护进程。anacron 设定的任务如果在指定时间没有成功执行，anacron 会间隔一段时间后再次执行该任务；而 crontab 设定的任务如果在指定时间没能完成，则调度工作就会失败。因而 anacron 设定的任务执行机率要比 crontab 高。

使用 anacron 前，应检查系统中是否已安装 anacron 服务程序，可以使用下面的命令。

```
# rpm -qa anacron
anacron-2.3-25 //已安装
```

启动 anacron 服务程序，可以使用下面的命令。

```
# /sbin/service anacron start
启动 anacron: [ 确定 ]
```

若要停止 anacron 服务程序，可以使用下面的命令。

```
# /sbin/service anacron stop
停止 anacron: [ 确定 ]
```

若要重新启动 anacron 服务程序，可以执行下面命令。

```
# /sbin/service anacron restart
停止 anacron: [ 确定 ]
启动 anacron: [ 确定 ]
```

设置 anacron 服务程序可以通过编辑/etc/anacrontab 文件实现。文件中除了注释和环境变量外，每一行表示一个任务，基本要求与/etc/crontab 文件类似，设定格式如下。

```
[时间间隔] [等待时间] [任务标识] [命令]
```

其中各项说明如下。

- [时间间隔]：指执行任务的时间间隔，以日为单位进行计算。
- [等待时间]：指时间间隔到期后，由于主机没有正常开机等原因导致任务没有顺利执行，则等待一段时间后尝试再次执行，以秒为单位进行计算。
- [任务标识]：记录此任务相关的说明。
- [命令]：设定执行的程序。

下面是一个/etc/anacrontab 文件内容示例。

```
#cat anacrontab //以下是 anacrontab 文件内容
# /etc/anacrontab: configuration file for anacron
# See anacron(8) and anacrontab(5) for details.
SHELL=/bin/sh
PATH=/sbin:/bin:/usr/sbin:/usr/bin
MAILTO=root
1 65 cron.daily run-parts /etc/cron.daily
7 70 cron.weekly run-parts /etc/cron.weekly
30 75 cron.monthly run-parts /etc/cron.monthly
```

其中共 8 行，第 1、2 行是注释，表明该文件是 anacron 的配置文件，详细使用说明可以查看 anacron (8) 或 anacrontab (5)；第 3 行表示默认的 Shell；第 4 行是执行该任务时，默认搜索的路径；第 5 行表示将执行结果以电子邮件方式通知 root 用户；第 6 行表示每隔 1 天执行/etc/cron.daily 目录下的程序，如果未能按时执行，则间隔 65 分钟后继续尝试；第 7 行表示每隔 7 天执行/etc/cron.weekly 目录下的程序，如果未能按时执行，则间隔 70 分钟后继续尝试；第 8 行表示每隔 30 天执行/etc/cron.monthly 目录下的程序，如果未能按时执行，则间隔



75 分钟后继续尝试。

3.4.3 at 命令：系统将执行的任务排队

at 命令也是一种任务管理工具，允许用户将一个或多个任务放到队列中，按时执行。与 crond 不同的是，at 设置的任务只在某个时刻执行，并且只执行一次。如果要使用 at 命令调度任务，必须首先启动 atd 守护进程，命令如下。

```
#/usr/sbin/atd
```

启动 atd 守护进程后，就可使用 at 命令设置任务。at 命令的语法格式如下所示。

```
at [-c ] [-V] [-q queue] [-f file] [-m][-l][-d][-v] TIME
```

其中各选项的具体说明如表 3-103 所示。

表 3-103 at 命令的选项说明

选 项	说 明
-m	当指定的任务被完成后，将给用户发送邮件，即使没有标准输出
-l	等同于 atq 命令
-d	等同于 atrm 命令
-v	显示任务将被执行的时间
-c	输出任务的内容到标准输出
-V	显示版本信息
-q queue	使用指定的队列
-f file	从指定文件读入任务而不是从标准输入设备读入
TIME	指定任务执行的时间

例如，设置 21 小时 55 分将根目录下所有文件的列表保存到/myshare/at_test 文件里，命令行如下所示。

```
# at 21:55
at> ls -al / > /myshare/at_test
at> <EOT> //Ctrl+d退出
job 2 at 2007-08-09 21:55
```

在键入“at 21:55”并按回车键后，at 命令会显示一个专用的提示符“at>”，输入需要执行的命令之后按 Ctrl+d 组合键退出 at，at 会提示任务执行的时间及编号。用户在“at>”提示符下可以输入多个命令，直到按下 Ctrl+d 组合键。

时间的设定可以采用多种格式，如 8:50 p.m May07、11:00 pm、+10 days、tomorrow、now、noon、midnight 等。例如，设置任务，要求 5 分钟后向/myshare/at_test 文件中存入单词“hello”，命令行如下。

```
# at now +5 min
at> echo "hello" >> /myshare/at_text
at> <EOT>
job 4 at 2007-08-09 09:59
# date //系统当前时间为 09:59
2007 年 08 月 09 日 星期四 09:59:17 CST
# cat /myshare/at_text
hello //任务已执行完毕
```


例如，设定明天下午 3 点执行指定的任务（在屏幕显示“break for a minutes”），命令如下所示。

```
# at 3pm tomorrow          //设定任务明天下午 3 点执行
at> echo break for a minutes      at> <EOT>
job 3 at 2008-05-06 15:00
```

如此，明天下午 3 点，系统将打印信息提醒休息一会。例如，设置三天后早上 8 点执行 date 命令，如下所示。

```
# at 8am+3days              //设定任务三天后早上 8 点执行
at> /bin/date                //执行 date 命令
at> <EOT>
job 4 at 2008-05-08 08:00
```

例如，查询已经设置的任务，命令如下所示：

```
# at -l                      //查询已经设置的任务
3      2008-05-06 17:00 a root
4      2008-05-08 08:00 a root
```

例如，显示已经设置的任务 6 的内容，命令如下所示。

```
# at -c 6                    //显示已经设置的任务 6 的内容
#!/bin/sh
i=1
while [ $i -le 50 ]
do
    userdel -r stud${i}
    i=$((i+1))
done
```

所有已执行的任务都被存储在/var/spool/at 目录中，可以从该目录中查看设置的任务，如下所示。

```
# cd /var/spool/at
# ls
a00001012dc8f4 a00002012dc903 spool
# cat a00001012dc8f4
#!/bin/sh
# atrun uid=0 gid=0
# mail      root 0
... ..      //环境变量
umask 22
... ..
du -a > /myshare/at_text          //用户设置的任务
```

不是所有用户都有使用 at 程序的权限。at 的权限设置是通过/etc/at.allow 和/etc/at.deny 两个文件来完成的。如果 at.allow 文件存在，系统会只允许在该文件中列出的用户使用 at 程序，其他用户无权使用；如果只需要拒绝很少一部分用户，而允许其他用户，可以使用 at.deny 文件，所有在该文件中被列出的用户将无权使用 at；如果 at.allow 和 at.deny 文件都不存在，则除了 root 用户，其他用户均无权使用 at。



在 at.allow 和 at.deny 文件中，每行只能输入一个用户名。



3.4.4 atd 命令：执行已经排队的任务

atd 命令执行已被排队的任务。该命令的语法格式如下所示。

```
atd [-l load_avg] [-b batch_interval] [-d] [-s]
```

其中各选项的具体说明如表 3-104 所示。

表 3-104 atd 命令的选项说明

选 项	说 明
-d	输出调试信息，将错误信息输出到标准错误输出设备
-s	只执行已排队的任务一次
-l	指定一个限制的负载参数，超过这个参数任务将不被执行
-b	指定两个批处理任务之间最小的时间间隔，默认是 60 秒

例如，以输出调试信息的方式运行队列中的任务，命令如下所示。

```
# atd -d
```

3.4.5 atq 命令：检查排队的任务

atq 命令列出已排队的任务。如果用户是超级用户，那么所有任务将被列出。标准输出的格式为：任务号、天数、小时数、工作类型。atq 命令的语法格式如下所示。

```
atq [-V] [-h] [-q queue]
```

其中各选项的具体说明如表 3-105 所示。

表 3-105 atq 命令的选项说明

选 项	说 明
-V	显示 atq 命令的版本信息
-h	显示 atq 命令的帮助信息
-q queue	指定队列

例如，使用 atq 命令查询当前已经设置的任务，命令行如下。

```
# atq //显示已经设置的队列任务
1      2008-08-09 21:40 a root
2      2008-08-09 21:55 a root
3      2008-08-09 22:57 a root
```

3.4.6 atrm 命令：删除已经排队的任务

atrm 命令用来删除已经排队的任务。删除时，任务将以任务号为标识，该命令的语法格式如下所示。

```
atrm [-V] [job...]
```

其中各选项的具体说明如表 3-106 所示。

表 3-106 atrm 命令的选项说明

选 项	说 明
-V	显示版本信息
-h	显示帮助信息
job	指定删除的任务，任务以编号标定

例如，删除当前任务队列中的第 3 个任务，任务编号可以通过 atq 命令查询，命令如下所示。

```
# atq
1      2007-08-09 21:40 a root
2      2007-08-09 21:55 a root
3      2007-08-09 22:57 a root
# atrm 3
# atq
1      2007-08-09 21:40 a root
2      2007-08-09 21:55 a root
```

3.4.7 atrun 命令：执行已排队的任务

atrun 命令运行 at 命令设置的任务，其实质是一个 Shell 脚本，调用使用 “-s” 选项的 atd 命令。该命令向后兼容旧版本。atrun 命令的语法格式如下所示。

```
atrun [-l load_avg] [-d]
```

其中各选项的具体说明如表 3-107 所示。

表 3-107 atrun 命令的选项说明

选 项	说 明
-l load_avg	设置负载参数，超过将不执行
-d	调试模式，向标准错误输出设备输出错误信息
-V	显示版本信息
-h	显示帮助信息

例如，执行已排队的任务，命令如下所示。

```
# atrun //执行已排队的任务
```

3.4.8 batch 命令：批处理

batch 命令用来实现批处理，即一次连接执行多个命令。该命令的语法格式如下所示。

```
batch [-V] [-q queue] [-f file] [-mv] [TIME]
```

其中各选项的具体说明如表 3-108 所示。

表 3-108 batch 命令的选项说明

选 项	说 明
-m	当任务执行完毕，即使任务没有输出，也向用户发送电子邮件
-f file	从指定的文件读取任务，而不是从标准输入设备
-q queue	指定队列
-V	显示版本信息
TIME	指定批处理任务开始执行的时间

例如，建立批处理文件，删除当前目录下 1.doc 和 2.doc 文件，命令如下所示。

```
#ls
1.doc 2.doc 3.doc 4.doc
# batch //批处理
```



```
at> rm 1.doc //删除文件
at> rm 2.doc //删除文件
at> <EOT> //结束输入
job 10 at 2008-06-26 14:54
# ls
3.doc 4.doc
```

实际上，batch 命令使用的仍然是 atd 守护进程，与 at 命令格式、用法也完全相同。不同之处在于，由 batch 设置的任务会在系统平均负载低于一个特定值（默认为 0.8）时才会运行，这可以有效防止调度的任务占用过多的处理器时间。例如，利用命令 du 对磁盘空间的使用情况进行统计通常是一项极耗费系统资源的工作，可以使用 batch 命令添加一个新任务，使该工作在平均负载比较低时才被执行，并将结果保存到/tmp/diskspace 文件中，命令如下。

```
#batch
at> du -h / > /tmp/diskspace
at><Ctrl+d>
job 6 at 2008-08-09 19:12
```

3.5 系统日志

日志是实现 Linux 系统安全的重要手段，利用日志可以审计和监测系统出现的错误，侦察和追踪入侵，协助系统的恢复和排障。Red Hat Linux 9 具有自动记录日志功能，通常使用 syslog 和 klog 来完成。syslog 记录常规系统日志，而 klog 针对内核活动进行记录。

日志会按类别记录在日志文件中，一般位于/var/log 目录下。其中，/var/log/dmesg 文件记录了与启动 Linux 相关的基本引导信息，/var/log/messages 文件记录了系统引导之后的附加信息，/var/log/boot.log 文件记录了与启动和停止守护进程相关的信息，/var/log/wtmp 文件帮助监视系统登录情况。root 用户可以使用文本编辑器（如 gedit 或 vi）查看这些日志文件，也可以使用 Red Hat Linux 9 的【系统工具】|【系统日志】间接查看日志文件。

3.5.1 syslogd 命令：启动 syslog 日志进程

几乎所有的类 UNIX 系统都采用 syslog 进行系统日志记录。syslog 有两个重要的文件：/etc/syslogd（守护进程）和/etc/syslog.conf 配置文件。启动 syslog 日志守护进程，使用如下命令。

```
#!/sbin/syslogd
```

如果希望守护进程可以接收来自网络的 syslog 信息，可以使用“-r”选项，命令如下所示。

```
#!/sbin/syslogd -r
```

通常在系统启动时，会自动启动 syslog 守护进程，重复运行“/etc/syslogd”命令，系统会提示已运行，如下所示。

```
# syslogd
syslogd: Already running.
```

在重新修改配置文件后，需要重启 syslog 进程才能使新的配置生效，命令如下所示。

```
#killall -HUP syslogd
```


其中“-HUP”使 syslog 关闭所有日志文件，重读/etc/syslog.conf 配置文件后重新开始记录日志。

3.5.2 syslog.conf: 系统日志配置文件

syslog 的执行内容由/etc/syslog.conf 文件设定，syslog.conf 的语法格式如下。

[消息设备.消息级别] [动作]

其中，[消息来源.消息级别]和[动作]之间必须用 Tab 分隔。消息来源指发出消息的设备或程序，表 3-109 列出了常用的消息来源。

表 3-109 syslog 常用消息来源

消 息 来 源	说 明
kern	内核
uucp	UUCP 程序
user	用户程序
news	Usenet 系统消息
mail	邮件系统
daemon	守护进程
auth	与安全认证及权限修改相关的命令
syslog	syslog 自身产生的消息
cron	crond 程序产生的消息
mark	时间戳
authpriv	私有的授权信息
lpr	打印机
local 0-7	本地消息

消息级别指消息的紧急程度。如果在一行上出现多个[消息设备.消息级别]，各项之间需用分号分隔。例如，“kern.emerg”表示来自内核的紧急信息，“mail.*; daemon.noctice”表示所有与邮件相关的信息和来自守护进程的警告信息。常用的消息级别如表 3-110 所示，其紧急程度由上到下逐级递减。

表 3-110 syslog 常用消息级别

消 息 级 别	说 明
emerg	最高的紧急等级，指极度恐慌，与 panic 同意
alert	紧急状态
crit	临界状态
err	出现错误
warning	警告
notice	出现了不正常现象，可能需要检查
info	一般性消息
debug	调试信息



紧急程度遵循向上匹配原则。例如，一个“err”选项表示所有大于等于“err”等级的消息都将被处理，即所处理的消息包括“err”、“crit”、“alert”、“emerg”；而如果被标志为“debug”，则所有消息都要被处理。如果只希望匹配某个确定的紧急程度，而不是向上匹配，需要使用等号进行设定。例如，“kern.=alert”表示只对内核产生的紧急信息（alert）进行处理。

syslog.conf 文件的配置行也支持通配符“*”和“none”，其中“*”表示匹配全部，“none”表示忽略全部。例如，“daemon.*”表示守护进程产生的所有信息，而“kern.none”将忽略内核的所有信息。

[动作]用于设定 syslog 如何处理对应的信息。可以设定将信息写入文件或显示在终端上，或直接发送给指定用户，也可以发送给指定的另一台主机。syslog 可用动作如表 3-111 所示。

表 3-111 syslog 可用动作及说明

动 作	说 明
@主机名	转发给另一台主机上的 syslog 程序
@IP 地址	转发给另一个 IP 地址的主机
*	转发到所有用户的终端上
/dev/console	转发到本地主机的终端上
程序	通过命名管道转发给程序
文件名	将信息写入指定的文件（文件必须使用绝对路径）
用户列表	将信息发给用户列表中的所有用户，用户名之间用逗号分隔

文件/etc/syslog.conf 的默认配置如下所示。

```
# Log all kernel messages to the console.
# Logging much else clutters up the screen.
#kern.* /dev/console //将内核的所有信息发送到本地主机终端上
# Log anything (except mail) of level info or higher.
# Don't log private authentication messages! // 不对私人认证信息进行日志记录
*.info;mail.none;news.none;authpriv.none;cron.none /var/log/messages
# The authpriv file has restricted access.
authpriv.* /var/log/secure
# Log all the mail messages in one place.
mail.* /var/log/maillog
# Log cron stuff
cron.* /var/log/cron
# Everybody gets emergency messages
*.emerg * //将所有的 emerge 信息发送给所有登录的用户
# Save news errors of level crit and higher in a special file.
uucp,news.crit /var/log/spooler
# Save boot messages also to boot.log
local7.* /var/log/boot.log//将启动信息同时记录到 boot.log 文件中
#
# INN
#
news.=crit /var/log/news/news.crit //将 Usenet 发出的 crit 信息记录到/var/
log/news/news.crit 文件中
news.=err /var/log/news/news.err //将 Usenet 发出的 err 信息记录到/var/
log/news/news.err
```



```
news.notice    /var/log/news/news.notice    //将 Usenet 发出的 notice、warning、err、
crit、alert、emerg     //信息记录到/var/log/news/news.notice
```

例如，将一般性的消息保存在/var/log/messages 文件中，但不包括邮件、新闻组、本地安全认证、守护进程、cron 程序所产生的信息，配置行如下。

```
*.info;mail.none;news.none;authpriv.none;daemon.none;cron.none
/var/log/messages
```

例如，发生内核恐慌 (emerg)，则将消息发送给所有登录的用户，配置行如下。

```
kern.emerg     *
```

例如，将所有与电子邮件相关的信息保存到/var/log/maillog 文件中，配置行如下。

```
mail.*         /var/log/maillog
```

例如，将与打印机相关的信息发送到 student.bit.edu.cn 主机上记录下来，配置行如下。

```
lpr.*          @student.bit.edu.cn
```

3.5.3 logger 命令：测试 syslog.conf

logger 命令可以模拟各类消息，从而可以对 syslog.conf 文件的配置进行测试。用户按着配置文件 syslog.conf 中的设置，设定 logger 发出指定类型的消息，从而可以检测配置是否正确。logger 命令格式如下。

```
logger -p [消息] [消息内容]
```

例如，测试 syslog.conf 中的 “*.emerg *”，可以执行如下命令。

```
# logger -p kern.emerg "Just test"
Message from syslogd@localhost at Fri Aug 10 13:34:46 2007 ...
localhost root: Just test
```

例如，测试 syslog.conf 中的 “mail.* /var/log/maillog”，可以执行如下命令。

```
# logger -p mail.info "Just mail test"
# grep test /var/log/maillog
Aug 10 13:38:35 localhost root: Just mail test
```

3.5.4 使用 echo 命令清空日志

随着系统运行时间越来越长，日志文件也会变得越来越大。如果创建日志文件的服务正在运行，必须停止服务，才能正常删除日志，否则会产生不可预测的后果。但可以利用 echo 命令清空日志，而不必停止日志服务。echo 的命令格式为：

```
#echo > 日志文件
```

例如，要清空保存在/usr/local/apache/logs/error_log 中的 Apache 服务器日志，可以运行如下命令。

```
#echo > /user/local/apache/logs/error_log
```

LINUX

第4章

用户和用户组管理

Linux 和其他类 UNIX 系统一样，是一个多用户、多任务操作系统。多用户特性允许多人在 Linux 中创建独立的账户来确保用户个人数据的安全性。而多任务机制允许多个用户同时登录，使用系统的软硬件资源。

用户和用户组管理是 Linux 系统管理的基础，也是实现 Linux 系统安全的重要手段。良好的用户和用户组管理可以为进一步扩展 Linux 系统应用提供强有力的支持。本章将分别从普通用户、root 用户以及用户组的角度对系统设置与管理的最常用命令进行介绍，并对用户管理常见问题进行分析。

在 Linux 操作系统中，每一个用户都有一个惟一的身份标识，称为用户 ID (UID)。每一个用户至少需要属于一个用户组。用户组是由系统管理员创建，由多个用户组成的用户群体。每一个用户组也有一个惟一的身份标识，称为用户组 ID。不同的用户和用户组对系统拥有不同的权限。对文件或目录的访问以及对程序的执行都需要调用者拥有相符合的身份，同样，一个正被执行的程序也相应地继承了调用者的所有权限。

Linux 用户被划分为两类：一类是根用户 (root 用户) 也被称为超级用户，一类是普通用户。根用户是系统的所有者，对系统拥有最高的权力，可以对所有文件、目录进行访问，执行系统中的所有程序而不管文件、目录和程序的所有者同意与否。普通用户的权限由系统管理员创建时赋予。通常普通用户只能管理自己的属主文件或组内共享及完全共享的文件。根用户与 Windows 系统中的 Administrator 地位相当，根用户在 Linux 系统中是惟一的，且不允许重新命名。



4.1 普通用户的管理

用户管理的基本任务包括添加新用户、删除用户、修改用户属性以及对现有用户的访问权限进行设置。与此密切相关的文件包括/etc/passwd、/etc/shadow 及/home 目录。虽然 Red Hat Linux 9 提供了图形化工具可以完成这些任务,但大多数管理员更习惯从命令行界面中执行管理操作。

4.1.1 解析/etc/passwd 文件

/etc/passwd 文件存储着用户的相关信息,包括用户名、密码、主目录位置等。根用户对该文件有读和写的权限,普通用户只有读权限。Linux 2.0 以上版本为了增强系统的安全性,采用了用户基本信息与密码分开存储的方法,密码已不存放在/etc/passwd 文件中,而是转存到了同目录下的/etc/shadow 文件中,原来存放密码的位置用“x”标识。/etc/passwd 存储的信息格式如下所示。

Username:encrypted password:UID:GID:full name:home directory:login shell

共分 7 个字段,各字段之间用“:”分隔。利用 cat 命令查看/etc/passwd 文件内容,如下所示。

```
#cat /etc/passwd
root:x:0:0:root:/root:/bin/bash           //根用户的设置项,UID 为 0,属于 root 组,登录 Shell 为/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin          //该用户在/bin下有许多命令,且主目录为/bin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin    //该用户可以控件一些打印功能,包括移动或清除 lp 日志、打印假脱 //机文件等工作。lp 的主目录是/var/spool/lpd
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin//该用户可以管理电子邮件
                                              //Mail 组有使用/var/spool/mail 文件的权限
news:x:9:13:news:/etc/news:               //该用户可以对 Internet 新闻服务进行管理
uucp:x:10:14:uucp:/var/spool/uucp:/sbin/nologin
operator:x:11:0:operator:/root:/sbin/nologin
games:x:12:100:games:/usr/games:/sbin/nologin
gopher:x:13:30:gopher:/var/gopher:/sbin/nologin
ftp:x:14:50:FTP User:/var/ftp:/sbin/nologin //该用户管理匿名 ftp 服务
nobody:x:99:99:Nobody:/:/sbin/nologin
rpm:x:37:37:/:/var/lib/rpm:/sbin/nologin
dbus:x:81:81:System message bus:/:/sbin/nologin
avahi:x:70:70:Avahi daemon:/:/sbin/nologin
```



```

mailnull:x:47:47::/var/spool/mqueue:/sbin/nologin
smmsp:x:51:51::/var/spool/mqueue:/sbin/nologin
nscd:x:28:28:NSCD Daemon:/:/sbin/nologin
vcsa:x:69:69:virtual console memory owner:/dev:/sbin/nologin
haldaemon:x:68:68:HAL daemon:/:/sbin/nologin
rpc:x:32:32:Portmapper RPC user:/:/sbin/nologin
rpcuser:x:29:29:RPC Service User:/var/lib/nfs:/sbin/nologin
nfsnobody:x:65534:65534:Anonymous NFS User:/var/lib/nfs:/sbin/nologin
sshd:x:74:74:Privilege-separated SSH:/var/empty/sshd:/sbin/nologin
pcap:x:77:77::/var/arpwatch:/sbin/nologin
ntp:x:38:38::/etc/ntp:/sbin/nologin
gdm:x:42:42::/var/gdm:/sbin/nologin
apache:x:48:48:Apache:/var/www:/sbin/nologin           //该用户管理 http 服务
distcache:x:94:94:Distcache:/:/sbin/nologin
postgres:x:26:26:PostgreSQL Server:/var/lib/pgsql:/bin/bash
mysql:x:27:27:MySQL Server:/var/lib/mysql:/bin/bash     //该用户管理 mysql 数据库
dovecot:x:97:97:dovecot:/usr/libexec/dovecot:/sbin/nologin
webalizer:x:67:67:Webalizer:/var/www/usage:/sbin/nologin
squid:x:23:23::/var/spool/squid:/sbin/nologin
named:x:25:25:Named:/var/named:/sbin/nologin
xfs:x:43:43:X Font Server:/etc/X11/fs:/sbin/nologin
sabayon:x:86:86:Sabayon user:/home/sabayon:/sbin/nologin
yang:x:500:500:yang:/home/yang:/bin/bash               //系统安装时创建的普通用户
teacher:x:501:501::/home/teacher:/bin/bash             //管理员自行添加的用户

```



可以使用 `vipw` 命令直接编辑 `/etc/passwd` 文件。`vipw` 命令在功能上相当于“`vi /etc/passwd`”命令，但比直接使用 `vi` 命令更安全。在用 `vipw` 命令编辑 `passwd` 文件时将自动对该文件加锁，编辑结束自动解锁，从而保证了数据的一致性。

其中列出了所有的用户信息，每个用户占用一行，各字段含义如下。

(1) 用户名：用户名是用户在系统中的标识，通常长度不超过 8 个字符，由字母、数字、下划线或句点组成。

(2) 密码：该字段存放加密后的用户密码。由于现在的系统大多采用 `shadow` 技术，该字段通常只存放一个特殊的字符“`x`”，真正的密码已被转移到 `/etc/shadow` 文件中。

如果该字段的第一个字母是“`#`”（如下例所示），表示该用户已被停用，即系统暂时不再允许该用户登录，但该用户的用户信息、相应的主目录及属主文件仍保存在系统中，并没有被系统删除。

```

yang:x:500:500:yang:/home/yang:/bin/bash
# teacher:x:501:501::/home/teacher:/bin/bash

```

(3) 用户标识号 (UID)：UID 是用户在系统中的惟一标识号，必须是整数，通常和用户名一一对应。当有多个用户名对应同一个 UID 时，系统会把它们视为同一用户。

UID 标识号的取值范围是 0~65535。0~499 一般由系统自己保留，其中“0”由根用户占用，新增用户的 UID 和 GID 要大于等于 500。

(4) 用户组标识号 (GID)：该字段记录用户所属的用户组。关于用户组的具体定义，可以查看/etc/group 文件。

(5) 个人信息描述：该字段记录用户的真实姓名、电话、地址、邮编等个人信息。各项之间用“,”分隔，该字段内容可以为空。

(6) 登录目录：该目录是用户登录系统后的默认目录，通常就是用户的主目录，一般在/home 下。根用户登录系统后默认的登录目录是/root。

(7) 登录 Shell：用户以文本方式登录系统后需要启动一个 Shell 进程。Shell 是用户和 Linux 内核之间的接口程序，负责将用户的操作传递给内核，所以 Shell 也被称为命令解释器。在 Linux 系统中有多种 Shell 可以使用，各 Shell 之间略有差别，常用的包括 sh(Bourne Shell)、csh (C Shell)、ksh (Korn Shell)、tcsh (TENEX/TOPS-20 type C Shell)、bash (Bourne Again Shell)等。其中，C shell 可以提供方便的用户界面设计，其语法与 C 语言很相似，而 Korn Shell 兼有 C shell 和 Bourne Shell 的优点。

无论是普通用户还是根用户，登录系统后都会进入该字段指定的命令解释器状态下，用户输入的每一个命令都将被这个命令解释器翻译执行。

该字段也可以被指定为一个特定的程序，此时用户登录后只能执行该程序。程序执行结束，用户就自动退出了系统。

4.1.2 解析/etc/shadow 文件

由于普通用户可以读取/etc/passwd 文件，因此将密码直接保存在该文件中是极不安全的，很可能会被别有用心的人获取并破译。目前的操作系统在密码保护方面大多采用了 Shadow Passwords 及 MD5 口令保护功能。Shadow Passwords 技术，即影子密码，是将加密的口令放在了另一个文件/etc/shadow 中，并且对/etc/shadow 文件设置严格的权限，只有根用户可以读取该文件。/etc/shadow 文件又称为影子文件，其存储的信息格式如下。

```
Username:Encrypted password:Number of days:Minimum password life:Maximum password life:Warning period:Disable account:Account expiration:Reserved
```

其中共分 9 个字段，各字段之间用“:”分隔。利用 cat 命令查看/etc/shadow 文件内容，如下所示。

```
#cat /etc/shadow
root:$1$f9.s.ENV$9PcgRNEuspQsWgG8vX.8V/:13698:0:99999:7::: //根用户已设置密码,账户
99999 天后失效,

//失效前 7 天系统会给出警告
bin:!:13698:0:99999:7:::
daemon:!:13698:0:99999:7:::
adm:!:13698:0:99999:7:::
lp:!:13698:0:99999:7:::
sync:!:13698:0:99999:7:::
shutdown:!:13698:0:99999:7:::
```



```
halt:!:13698:0:99999:7:::
mail:!:13698:0:99999:7:::
news:!:13698:0:99999:7:::
uucp:!:13698:0:99999:7:::
operator:!:13698:0:99999:7:::
games:!:13698:0:99999:7:::
gopher:!:13698:0:99999:7:::
ftp:!:13698:0:99999:7::: //ftp 用户，目前已禁止该用户登录
nobody:!:13698:0:99999:7:::
rpm:!:13698:0:99999:7:::
dbus:!:13698:0:99999:7:::
avahi:!:13698:0:99999:7:::
mailnull:!:13698:0:99999:7:::
smmsp:!:13698:0:99999:7:::
nscd:!:13698:0:99999:7:::
vcsa:!:13698:0:99999:7:::
haldaemon:!:13698:0:99999:7:::
rpc:!:13698:0:99999:7:::
rpcuser:!:13698:0:99999:7:::
nfsnobody:!:13698:0:99999:7:::
sshd:!:13698:0:99999:7:::
pcap:!:13698:0:99999:7:::
ntp:!:13698:0:99999:7:::
gdm:!:13698:0:99999:7:::
apache:!:13698:0:99999:7::: //http 用户，未启用，暂无法登录
distcache:!:13698:0:99999:7:::
postgres:!:13698:0:99999:7:::
mysql:!:13698:0:99999:7::: //mysql 用户，未启用，暂无法登录
dovecot:!:13698:0:99999:7:::
webalizer:!:13698:0:99999:7:::
squid:!:13698:0:99999:7:::
named:!:13698:0:99999:7:::
xfs:!:13698:0:99999:7:::
sabayon:!:13698:0:99999:7:::
yang:$1$9y0JsFWl$fW4uZTnP6r7hLykdf7jQ7l:13698:0:99999:7:::
teacher:!:13698:0:99999:7::: //设置用户 teacher 密码为空
```

上面列出了所有有效用户的密码信息，每个用户占用一行，一行分 9 个字段，分别表示用户名、密码、从 1970 年到上次修改密码的天数、密码必须连续使用的天数、密码有效期、密码失效前告警的天数、从密码过期到彻底停用的天数、账号失效日期，最后一段作为保留字段，详情见表 4-1。从/etc/shadow 文件和/etc/passwd 文件的对比中可以看出，其每行记录所记载的用户信息是一一对应的。

表 4-1 文件/etc/shadow 的格式说明

序 号	字 段	描 述
1	Username	用户的登录名
2	Encrypted password	已加密的密码
3	Number of days	从 1970 年 1 月 1 日到上次修改密码的天数
4	Minimum password life	至少在设定的天数内，密码是不能被修改的
5	Maximum password life	在设定的天数之后必须重新设置密码
6	Warning period	在密码失效前，提前提醒用户密码即将失效的天数
7	Disable account	设定密码过期之后，如果该账号仍没有被使用，则停用该账号的天数
8	Account expiration	设定账号失效的时间。如果到这个时间还没有使用该账号，用户将不能以该账号身份登录。时间的格式可以是 YYYY-MM-DD，也可以是距 1970 年 1 月 1 日的天数
9	Reserved	系统保留



MD5 (Message Digest v5) 是密码学中的经典算法之一，经常应该于数字签名技术。MD5 可以接受任意长度的字符串，并根据输入的字符串产生一组 128 位的信息摘要。从理论上讲，两组不同的字符串输入是不可能得到相同的输出，所以 MD5 可以有效保护用户的口令安全。

4.1.3 pwconv 命令：创建影子文件

pwconv 命令将普通密码文件/etc/passwd 中的密文转换为“x”，同时创建影子文件/etc/shadow 保存密文，即密文保存在影子文件中，而原密码文件/etc/passwd 不再包含密文。该命令可以使系统下用户的密码得到更安全的保护。

例如，使用 pwconv 命令创建影子文件，如下所示。

```
# cat /etc/passwd //显示当前的密码文件
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
... ..
desktop:!!:80:80:desktop:/var/lib/menu/kde:/sbin/nologin
radvd:!!:75:75:radvd user:/:/sbin/nologin
yang:$1$u6oQ8taD$nYGI0CBk9ourDkiLbBFBt1:500:500:./home/yang:/bin/bash //用户 yang 的密文直接保存

//在/etc/passwd 文件中
#ls /etc/shadow
ls: /etc/shadow: 没有那个文件或目录 //不存在/etc/shadow 文件
# pwconv //创建影子文件
# cat /etc/passwd //显示当前的密码文件
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
... ..
```



```

desktop:!!:80:80:desktop:/var/lib/menu/kde:/sbin/nologin
radvd:!!:75:75:radvd user:/:/sbin/nologin
yang:x:500:500:~/home/yang:/bin/bash //密文已由“x”代替
#cat /etc/shadow //文件/etc/shadow 已创建，内容如下
root:$1$Q4C9mmGk$e.4YPTON9lixkP4ZVdzE2.:14092:0:99999:7:::
bin:*:14092:0:99999:7:::
daemon:*:14092:0:99999:7:::
adm:*:14092:0:99999:7:::
lp:*:14092:0:99999:7:::
... ..
desktop:!!:14092:0:99999:7:::
radvd:!!:14092:0:99999:7:::
yang:$1$u6oQ8taD$nYGI0CBk9ourDkiLbBFBt1:14092:0:99999:7::: //用户 yang 的密文

```

通过命令 `pwconv`，可以创建用户的影子文件，使密码得到更好的保护，不容易被别的用户得到。

4.1.4 pwunconv 命令：从影子文件恢复密文

`pwunconv` 命令与 `pwconv` 命令相反，用来从用户的影子文件恢复密码文件 `/etc/passwd` 中的密文，同时删除影子文件 `/etc/shadow`。

例如，从影子文件中恢复密文，执行过程如下所示。

```

# cat /etc/passwd //显示当前的密码文件
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
... ..
desktop:!!:80:80:desktop:/var/lib/menu/kde:/sbin/nologin
radvd:!!:75:75:radvd user:/:/sbin/nologin
yang:x:500:500:~/home/yang:/bin/bash //密文已由“x”代替
#cat /etc/shadow //显示/etc/shadow 文件
root:$1$Q4C9mmGk$e.4YPTON9lixkP4ZVdzE2.:14092:0:99999:7:::
bin:*:14092:0:99999:7:::
daemon:*:14092:0:99999:7:::
adm:*:14092:0:99999:7:::
lp:*:14092:0:99999:7:::
... ..
desktop:!!:14092:0:99999:7:::
radvd:!!:14092:0:99999:7:::
yang:$1$u6oQ8taD$nYGI0CBk9ourDkiLbBFBt1:14092:0:99999:7::: //用户 yang 的密文
# pwunconv //由影子文件创建密码文件
# cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
... ..
desktop:!!:80:80:desktop:/var/lib/menu/kde:/sbin/nologin
radvd:!!:75:75:radvd user:/:/sbin/nologin
yang:$1$u6oQ8taD$nYGI0CBk9ourDkiLbBFBt1:500:500:~/home/yang:/bin/bash //用户 yang 的密

```


文直接保存

```
# ls /etc/shadow
```

```
ls: /etc/shadow: 没有那个文件或目录
```

//在/etc/passwd 文件中

//影子文件/etc/shadow 已被删除

4.1.5 useradd 命令：添加新用户

系统中，一个合法的用户应该具有用户名、真实姓名、密码、登录环境等用户信息。与此相对应，添加一个新用户通常需要完成以下几项操作。

(1) 设置用户名称及密码。

(2) 设置用户的 UID。系统在/etc/passwd 文件中查找目前使用的大于 500 的 UID 的最大编号，将其加 1 后赋予当前的新用户；若目前还没有大于 500 的编号，则将 500 赋予该用户。

(3) 添加该新用户的用户组。每一个用户都会属于一个或多个用户组。系统在添加新用户时默认添加的用户组名与新用户名相同，同时会赋予该用户组一个 GID，GID 的编号通常与 UID 的编号相同。

(4) 创建以新用户的用户名为名称的主目录。在大多数系统中，用户的主目录都被创建在同一个特定目录下，如/home。各用户对自己的主目录有完全的读、写、执行权限，其他用户只能依据该目录的权限进行访问。

(5) 设定用户的 Shell 环境，默认是/bin/bash。

(6) 设定用户的失效时间，默认是 99999 天后。

(7) 设定失效前发出警告的天数，默认是失效前 7 天。

在 Red Hat Linux 9 的安装过程中，系统会自动创建若干默认的标准用户（Standard Users），其中除了 root 代表系统管理者之外，其余账号都是系统账号。系统账号是应用程序在运行过程中所具有的权限。有关标准用户的详细说明如表 4-2 所示。

表 4-2 Linux 系统标准用户说明

用 户 名	用户 ID	用户组 ID	用户所在组	用户主目录	使用的 Shell
root	0	0	root	/root	/bin/bash
bin	1	1	bin	/bin	/sbin/nologin
daemon	2	2	daemon	/sbin	/sbin/nologin
adm	3	4	adm	/var/adm	/sbin/nologin
lp	4	7	lp	/var/spool/lpd	/sbin/nologin
sync	5	0	sync	/sbin	/bin/sync
shutdown	6	0	shutdown	/sbin	/sbin/shutdown
halt	7	0	halt	/sbin	/sbin/halt
mail	8	12	mail	/var/spool/mail	/sbin/nologin
news	9	13	news	/etc/news	
uucp	10	14	uucp	/var/spool/uucp	/sbin/nologin
operator	11	0	operator	/root	/sbin/nologin
games	12	100	games	/usr/games	/sbin/nologin
gopher	13	30	gopher	/var/gopher	/sbin/nologin
ftp	14	50	FTP User	/var/ftp	/sbin/nologin



续表

用 户 名	用户 ID	用户组 ID	用户所在组	用户主目录	使用的 Shell
nobody	99	99	Nobody	/	/sbin/nologin
rpm	37	37		/var/lib/rpm	/sbin/nologin
dbus	81	81	System message bus	/	/sbin/nologin
avahi	70	70	Avahi daemon	/	/sbin/nologin
mailnull	47	47		/var/spool/mqueue	/sbin/nologin
smmsp	51	51		/var/spool/mqueue	/sbin/nologin
nscd	28	28	NSCD Daemon	/	/sbin/nologin
vcsa	69	69	virtual console memory owner	/dev	/sbin/nologin
haldaemon	68	68	HAL daemon	/	/sbin/nologin
rpc	32	32	Portmapper RPC user	/	/sbin/nologin
rpcuser	29	29	RPC Service User	/var/lib/nfs	/sbin/nologin
nfsnobody	65534	65534	Anonymous NFS User	/var/lib/nfs	/sbin/nologin
sshd	74	74	Privilege-separated SSH	/var/empty/sshd	/sbin/nologin
pcap	77	77		/var/arpwatch	/sbin/nologin
ntp	38	38		/etc/ntp	/sbin/nologin
gdm	42	42		/var/gdm	/sbin/nologin
apache	48	48	Apache	/var/www	/sbin/nologin
distcache	94	94	Distcache	/	/sbin/nologin
postgres	26	26	PostgreSQL Server	/var/lib/pgsql	/bin/bash
mysql	27	27	MySQL Server	/var/lib/mysql	/bin/bash
dovecot	97	97	dovecot	/usr/libexec/dovecot	/sbin/nologin
webalizer	67	67	Webalizer	/var/www/usage	/sbin/nologin
squid	23	23		/var/spool/squid	/sbin/nologin
named	25	25	Named	/var/named	/sbin/nologin
xfp	43	43	X Font Server	/etc/X11/fs	/sbin/nologin
sabayon	86	86	Sabayon user	/home/sabayon	/sbin/nologin

管理员可以使用 `useradd` 或 `adduser` 命令来添加一个新的用户。在 Red Hat Linux 9 中，通过查看 `adduser` 和 `useradd` 这两个命令的文件信息，可以看出其功能是完全相同的。

```
#ls -l /usr/sbin/useradd /usr/sbin/adduser
lrwxrwxrwx 1 root root 7 07-12 07:05 /usr/sbin/adduser->useradd //链接文件
-rwxr-x--- 1 root root 74512 01-17 03:50 /usr/sbin/useradd //普通文件
```

可以看出，`adduser` 命令只是 `useradd` 命令的一个链接文件，如此设计只是为了方便用户的使用。`useradd` 程序通常在 `/usr/sbin` 目录中，命令格式如下。

```
useradd [-c comment] [-d home_dir] [-e expire_date] [-f inactive_time]
        [-g initial_group] [-G group[,...]]
        [-m [-k skeleton_dir] | -M] [-s shell]
        [-u uid [-o]] [-n] [-r] login
useradd -D [-g default_group] [-b default_home]
        [-f default_inactive] [-e default_expire_date]
        [-s default_shell]
```


不带“-D”选项时，`useradd` 命令用来指定新账户的设定值，如果没有指定该设定值则使用系统的默认值。`useradd` 可使用的选项说明如下。

- `-c comment`: 用户的注释说明。
- `-d home_dir`: 用户每次登录系统时所使用的登录目录，可以用来取代默认的 `/home/username` 主目录。
- `-e expire_date`: 账号失效日期。日期的指定格式为 `MM/DD/YY`。
- `-f inactive_days`: 设定从账号过期到永久停用的天数。当值为 0 时，账号到期后会立即被停用；而当值为 -1 时，账号不会被停用。系统默认值为 -1。
- `-g initial_group`: 用户默认的用户组或默认的组 ID。该用户组或组 ID 必须是已经存在的，其默认组 ID 值为 100，即属于 `users` 组。
- `-G group,[...]`: 设定该用户为若干用户组的成员。每个用户组之间用“,”分隔，且不可以夹杂空格。组名与“-g”选项的限制相同，且“-g”的设定值为用户的第一用户组。
- `-m`: 用户目录如不存在则自动建立。若使用“-k”选项，则目录“`skeleton_dir`”内的文档会被复制至此用户目录中，同时 `/etc/skel` 目录下的文档也会被复制过去。任何在 `skeleton_dir` 或 `/etc/skel` 中的目录也同样会在该用户目录下一一被建立。“-k”和“-m”的默认值是不建立目录及不复制任何文档。
- `-M`: 不建立用户主目录，使用 `/etc/login.defs` 系统文件对用户进行设定。
- `-n`: 系统默认用户组名称与用户名称相同。打开此选项将取消此默认设定。
- `-r`: 此选项用来建立系统账号。系统账号的 UID 是比定义在 `/etc/login.defs` 中的 `UID_MIN` 小的值，`UID_MIN` 的默认值是 500。



“`useradd -r`”命令所建立的账号不会创建用户主目录，也不会依据 `/etc/login.defs` 对用户进行设置。如果想创建用户主目录，需额外指定 `-m` 参数。

- `-shell default_shell`: 指定用户的登录 Shell，系统默认为 `/bin/bash`。
- `-u uid`: 用户的 UID 值。该数值在系统中必须惟一，且数字不可为负值。0~499 传统上预留给系统账号使用。

若带选项“-D”且配合其他选项，`useradd` 可以对系统的默认值进行重新设定。如不带任何其他选项，则显示当前的默认值，如下所示。

```
#useradd -D
GROUP=100
HOME=/home
INACTIVE=-1
EXPIRE=
SHELL=/bin/bash
SKEL=/etc/skel
CREATE_MAIL_SPOOL=no
```

例如，修改 `useradd` 命令的默认值，命令行如下。

```
//修改 useradd 命令使用的 shell 的默认值为“/bin/csh”
#useradd -D -s /bin/csh
```



```
#useradd -D //显示默认值
GROUP=100
HOME=/home
INACTIVE=-1
EXPIRE=
SHELL=/bin/csh //默认 Shell 已改为/bin/csh
SKEL=/etc/skel
CREATE_MAIL_SPOOL=no
```

例如，添加一新用户 student2，UID 为 550，用户组 ID 为 100（users 用户组的标识符是 100），用户目录为 /home/student2，用户的默认 Shell 为 /bin/bash，账号的失效日期为 2007 年 10 月 30 日，其命令行如下所示。

```
#useradd student2 -u 550 -d /home/student -s /bin/bash -e 10/30/07 -g 100
//新添加用户的用户信息存储在/etc/passwd 和/etc/shadow 文件尾行，使用 tail 命令可以查看指定的行
#tail -1 /etc/passwd //显示/etc/passwd 文件最后一行内容
student2:x:550:100::/home/student2:/bin/bash
#tail -1 /etc/shadow //显示/etc/shadow 文件最后一行内容
student2:!:13707:099999:7: :13816:
```

如果新添加的用户名已经存在，那么执行 useradd 命令后，系统会提示用户已存在，如下所示。

```
#useradd student2
useradd: user student2 exists
```

4.1.6 usermod 命令：修改用户的账号

修改用户的账号包括更改用户的用户名、密码、主目录、所属用户组和登录 Shell 等信息。修改用户的信息可以使用 usermod 命令，其命令格式如下所示。

```
usermod [-c comment] [-d home_dir [-m]] [-e expire_date] [-f inactive_time] [-g
initial_group] [-G group,...]]
        [-l login_name] [-s shell] [-u uid [-o]] login
```

usermod 命令会参照命令行上指定的选项对用户账号进行修改。usermod 可用的选项如下所示。

- -c comment: 更新用户的注释信息。
- -d home_dir: 更新用户的登录目录。如果指定了“-m”选项，则旧目录中的内容会被复制到新的目录中。如果新目录不存在则自动创建。
- -e expire_date: 更新用户账号停用日期。其日期格式为 MM/DD/YY。
- -f inactive_days: 设定账号失效到永久停用的天数。当值为 0 时，账号到期后立刻被停用；而当值为-1 时，则关闭此功能。默认值为-1。
- -g initial_group: 更新用户起始登录的用户组，即第一用户组。用户组名必须是已经存在的，默认值为 users 组。
- -G group, [...]: 更新用户所属的用户组。通常一个用户可以属于多个用户组，成为多个用户组的成员。每个用户组名之间必须用“,”分隔，如果用户当前所在的用户组不在此例中，则会从该当前用户组中删除此用户。
- -l login_name: 变更用户登录时的名称为 login_name。
- -s shell: 指定用户新的登录 Shell。如果此项留白，系统将选用默认的 Shell。
- -u uid: 更新用户的 UID 值。该值被修改后，用户目录树下所有文件、目录的用户

UID 值会自动改变，但放在用户主目录外的文件、目录的 UID 值则需要用户手动更新。

例如，将用户的登录目录改为/home/student2，用户的登录 Shell 改为 ksh，用户所在的组改为 users 和 student，命令如下所示。

```
#usermod -d /home/student2 -s /bin/ksh -g users student
```

4.1.7 userdel 命令：删除用户

当不允许用户再次登录本系统时，可以将该用户从系统中删除。如果准备删除的用户已经登录，则必须退出系统才能删除。与添加用户的操作相反，删除用户时系统需要删除/etc/passwd、/etc/shadow、/etc/group 文件中的对应条目，还需删除用户的主目录及所属文件。userdel 命令的语法格式如下。

```
userdel [-r]
```

如果不带选项“-r”，则只删除用户在系统中的账户信息，用户的主目录及相关文件依然保留在系统中。使用选项“-r”，则可将用户主目录下的文档全部删除，同时该用户放在其他位置的文档也会被一一找出并删除。例如：

```
userdel -r student
```

执行该命令后，系统将删除 student 账号，同时删除 student 的主目录、邮件及相关属主文档。删除用户前，应检查系统是否还有该用户的相关进程正在执行。如果存在该用户的进程，则需等待其执行完毕或直接终止该用户进程。例如，可以使用 ps 或 top 命令对进程进行检查，使用 kill 命令终止该用户进程，如下所示。

```
#ps -aux | grep "student"
student 4001 0.0 0.5 4712 666 ? S 12:15 0:00 studentproc
#kill 4001
```

用 crontab 命令查看是否还有该用户设定的定时任务，如果有则进行删除，如下所示。

```
#crontab -u student -r
```

4.1.8 passwd 命令：修改用户密码

密码管理是用户管理的一项重要内容。用户在刚创建账号时如果没有设定密码，则该账号是被系统锁定的，用户无法用该账号登录。只有为其指定密码（即使密码为空）才能激活账号。

指定和修改用户密码的命令是 passwd。根用户不仅可以修改自己的密码，还可以修改其他用户的密码。普通用户则只能修改自己的密码。passwd 命令的语法格式如下所示。

```
passwd [-k] [-l] [-u] [-f] [-d] [-n mindays] [-x maxdays] [-w warndays]
        [-i inactivedays] [-S] [--stdin] [username]
```

其中各项说明如下。

- -k：表示只有密码过期才需要用户重新设定密码。
- -l：通过在用户的密码字段前加前缀“!”，对用户进行锁定。锁定的用户无法登录系统，该命令只有根用户有权使用。例如，锁定 student 用户，使其不能登录的命令如下所示。

```
#passwd -l student
```

- --stdin：表示从标准输入重新读入密码，该标准输入也可为一管道。



- **-u**: 该选项与“-l”相反，是对锁定的用户进行解锁操作。该操作会删除密码字段前的“!”，使用户可以重新登录系统。对于口令为空的用户，系统原则上是不允许解锁的，需配合使用“-f”选项，才能强制解锁。
- **-d**: 快速删除用户的密码。该命令只对根用户有效。例如，删除 student 用户的密码可以执行下面的命令。

```
#passwd -d student
```

- **-n mindays**: 设定最短的密码有效期。
- **-x maxdays**: 设定最长的密码有效期。
- **-w**: 设定密码过期前，提前发出警报的天数。
- **-i inactivedays**: 设定密码过期到账号停用的天数。
- **-S**: 显示指定用户当前密码状态。

其中 **username** 的默认值为当前用户，即用户名为空，则修改当前用户的密码。设定新密码需输入旧密码进行验证，如果验证不正确则不允许修改。新密码需连续输入两次，如果两次的输入一致，则新密码生效，如下所示。

```
$passwd
Changing password for user student
Changing password for student
(current) UNIX password:
passwd: authentication token manipulation error //不能提供合法的旧密码，验证失败
$passwd
Changing password for user student
Changing password for student
(current) UNIX password:
New password:
Retype new password:
passwd: all authentication tokens updated successfully. //两次输入一致，密码生效
```

如果是根用户修改普通用户的密码，则无需知道该普通用户的原始密码，如下所示。

```
#passwd student
Changing password for user student
New UNIX password:
Retype new UNIX password:
passwd: all authentication tokens updated successfully
```

密码应是字母、数字及符号的组合，不应小于 5 个字符长，密码过短或过于简单系统会提示其是弱口令。如果用户执意要将弱口令设为密码只需继续输入，如下所示。

```
#passwd student
Changing password for user student
New UNIX password: //仅输入 3 位密码
BAD PASSWORD: it is WAY too short //密码过短
Retype new UNIX password:
passwd: all authentication tokens updated successfully //虽然密码过短但已生效
#passwd student
Changing password for user student
New UNIX password:
BAD PASSWORD: it is too simplistic/systematic //密码过于简单
Retype new UNIX password:
passwd: all authentication tokens updated successfully //虽然密码过于简单但已生效
```


4.1.9 newusers 命令：批量创建用户

管理员有时需要一次性创建大量用户账号，例如新学期开学或成立新的部门时。如果仍采用 `useradd` 命令逐一创建，不仅浪费时间而且在录入期间也很可能产生错误。在此情况下通常利用脚本程序完成批量用户的添加和修改。

例如编写 Shell 程序实现系统自动创建 20 个用户，用户名为 `student1~student20`，用户组为 `users`。程序中主要用到变量赋值语句、命令替换语句、循环语句以及流过滤语句 `awk`。用 `tail` 命令先查看一下 `/etc/passwd` 和 `/etc/shadow` 文件格式，如下所示。

```
#tail -1 /etc/passwd
teacher:x:500:500:teacher:/home/teacher:/bin/bash
#tail -1 /etc/shadow
teacher: :13707:0:99999:7:::
```

根据上述情况编写脚本程序如下。

```
#!/bin/sh
i=1
//提取最大的用户 ID 号
awk 'BEGIN { FS= ":" ; } { print $3 } ' /etc/passwd > uid_list//将 passwd 文件中的第三
列暂存到 uid_list 文件中
temp=' tail -1 uid_list ' //该处使用后引号，提取 uid_list 文件中的最后一行
while [ $i -le 20 ]
do
//创建新用户的主目录
mkdir /home/student${i}
temp=$((i+1))
//在/etc/passwd 和/etc/shadow 文件中添加新的用户信息
echo "student${i}:x:${temp}:100:student${i}:/home/student${i}:/bin/bash">>/etc/passwd
echo "student${i}: :13707:0:099999:7:::">>/etc/shadow
i=$((i+1))
done
```

对于不熟悉程序编写的用户，Red Hat Linux 9 提供了创建大量用户账号的工具——`newusers`。`newusers` 命令能够用一个含有用户名和密码的用户信息文件来生成和修改大量的账号。这个用户信息文件必须具有与 `/etc/passwd` 文件相同的格式，且每个账号的用户名和用户 ID 必须不同。密码字段可以为空或输入“x”。建立上例中所需的用户信息文件，如下所示。

```
#vi /new_account
student1:x:501:100: :/home/student1:/bin/bash
student2:x:501:100: :/home/student2:/bin/bash
student3:x:501:100: :/home/student3:/bin/bash
student4:x:501:100: :/home/student4:/bin/bash
... ..
```

将所建立的用户信息文件 `/new_account` 输出给 `newusers` 命令，可以完成新用户的批量自动建立，命令行如下所示。

```
#newusers < /new_account
```

查看 `/etc/passwd` 文件和 `/home` 目录，可能看到新用户已经被添加，并且相应的主目录也已经被创建，如下所示。

```
#tail -20 /etc/passwd
student1:x:501:100: :/home/student1:/bin/bash
```



```
student2:x:501:100: :/home/student2:/bin/bash
student3:x:501:100: :/home/student3:/bin/bash
student4:x:501:100: :/home/student4:/bin/bash
...
#ls /home
student1 student2 student3 student4 student5 student6 student7 student8 student9
student10
student11 student12 student13 student14 student15 student16 student17 student18
student19 student20
teacher yang
```

4.1.10 chpasswd 命令：批量创建用户密码

创建批量用户的密码也可采用与 newusers 命令类似的方法。首先建立用户的密码文件，如下所示。

```
#vi /password_account
student1:ijl335u
student2:4jslfjkl
student3:ijl335u
student4:4jslfjkl
...
```

将编辑好的密码文件传输给 chpasswd 命令，完成用户密码的批量设置，如下所示。

```
#chpasswd < /password_account
```

完成以上步骤后，就可以使用这些账号来登录系统。

4.1.11 chage 命令：查看账户密码的有效期

账户到期或密码失效都会使用户无法登录系统，可以使用 chage 命令查看账户密码的有效期。命令 chage 的语法格式如下所示。

```
chage [-d, --lastday] [-E, --expiredate] [-h, --help] [-I, --inactive] [-l, --list] [-m, --mindays] [-M, --maxdays]
[-W, --warndays]
```

其中各选项的具体说明如表 4-3 所示。

表 4-3 chage 选项定义

选 项	定 义	说 明
-d, --lastday	最近日期	将最近一次密码设置时间设为“最近日期”
-E, --expiredate	过期日期	设置帐户的过期时间
-h, --help	选项帮助	显示帮助信息并退出
-I, --inactive	密码失效天数	设置从密码过期到账户登录禁用的天数
-l, --list	当前设置	显示账户时效信息
-m, --mindays	最小天数	设置两次改变密码之间相距的最小天数
-M, --maxdays	最大天数	设置两次改变密码之间相距的最大天数
-W, --warndays	警告天数	设置密码过期之前，向用户发出“密码将过期”警告的天数

例如，使用“-l”选项显示 student1 用户的密码时效设置如下。

```
#chage -l student1
最近一次密码修改时间          : 7月 21, 2007
```


密码过期时间 : 7月 23, 2007
密码失效时间 : 从不
账户过期时间 : 7月 29, 2007
两次改变密码之间相距的最小天数 : 0
两次改变密码之间相距的最大天数 : 99999
在密码过期之前警告的天数 : 7

当 student1 用户 2007 年 7 月 23 日之后登录系统，系统会强行要求用户修改密码。如果用户未能在 2007 年 7 月 29 日之前修改密码，则会因为账号失效而导致无法登录系统。

4.2 根用户的管理

在 Red Hat Linux 9 系统中，首要的管理员用户被称为根用户。该用户对系统拥有完全的控制权，可以对系统做任何的设置和更改。因其权力远远大于普通用户，所以该用户一般也被称为 Super User，即超级用户。由于根用户对系统的使用不会受到任何限制和约束，非法用户以 root 身份登录系统或者经验不足的 Linux 用户使用 root 账号所做的误操作，都有可能对系统造成严重的后果。因此根用户的账号安全显得格外重要。在 Red Hat Linux 9 系统安装过程中，除创建根用户外，还会创建系统默认普通用户。在不是绝对必要的情况下，最好不要用 root 账号登录系统。

4.2.1 su 命令：变更用户身份

当用户以普通账号登录系统后，由于缺少管理权限无法对系统进行重新设置，可利用 su 命令临时切换为 root 身份。su 命令用于切换系统的用户帐号，不输入用户名时，默认的用户为 root 用户。该命令的语法格式如下所示。

```
su[-l,--login][-c,--command=COMMAND][-f,--fast][-m,--preserve-environment][-p][-s,--shell=SHELL]
```

其中各选项的具体说明如表 4-4 所示。

表 4-4 su 命令的选项说明

选 项	说 明
-l, --login	使用登录 Shell
-c, --command=COMMAND	为 Shell 传递一个单一命令
-f, --fast	为 Shell 传递选项-f（适用于 csh 或 tcsh）
-m, --preserve-environment	不重置环境变量
-p	与 -m 选项相同
-s, --shell=SHELL	如果/etc/shells 允许，运行 Shell
--help	显示帮助信息
--version	显示版本信息

例如，当前目录为 teacher，利用 su 命令临时切换为 root 身份，如下所示。

```
$whoami //显示当前用户的用户名  
teacher
```



```
$su //切换为根用户
Password:
#whoami //显示当前用户的用户名
root
```

完成相关的系统设置之后，利用 `exit` 命令可以返回普通用户身份。如果要切换到其他用户，可以在 `su` 命令后加上其他用户的账号名，并输入该账号的密码，如下所示。

```
$whoami
teacher
$su //切换到根用户
Password:
#whoami
root
#exit //返回到用户 teacher
$whoami
teacher
$su student //切换到 student 用户
Password:
$whoami
Student
```

4.2.2 以单用户模式解决 root 密码丢失问题

由于对系统的许多配置必须由 `root` 来完成，`root` 密码丢失会导致系统配置无法进行，从而使系统失去系统管理员的控制。

Linux 系统可以运行在多种模式下，其中单用户模式不需要用户输入密码即可进入。丢失 `root` 密码的用户可以以单用户模式进入系统，对密码进行重新设置。操作步骤如下。

(1) 当系统启动时，按回车键进入 GRUB 界面，如图 4-1 所示。若安装了多个系统，菜单中会显示多个系统引导选项。用上下光标键选中要启动的系统，按回车键直接启动系统。按“e”键可以对启动命令进行编辑，按“a”键可以修改内核的启动参数，按“c”键直接进入 GRUB 命令行。此处，选择【Red Hat Linux (2.4.20-8)】，并按“e”键进入命令菜单编辑状态。

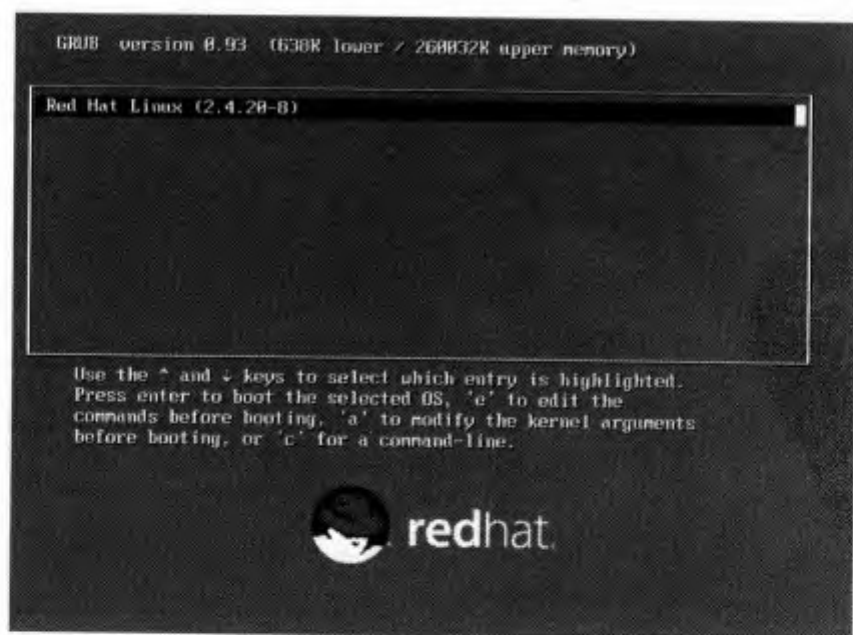


图 4-1 系统选项菜单

(2) 图 4-2 所示界面中列出了目前系统启动时可用的命令选项。其中，按“e”键可以对

命令行进行编辑，按“b”键启动该命令行，按“c”键进入 GRUB 命令行，按“o”键在选定行下添加一新行，按“O”键在选定行前添加一新行，按“d”键对指定行进行删除，按 Esc 键返回主菜单。此处，选择【kernel /vmlinuz-2.4.20-8 ro root=LABEL=/】，并按“e”键，进入命令行编辑菜单。

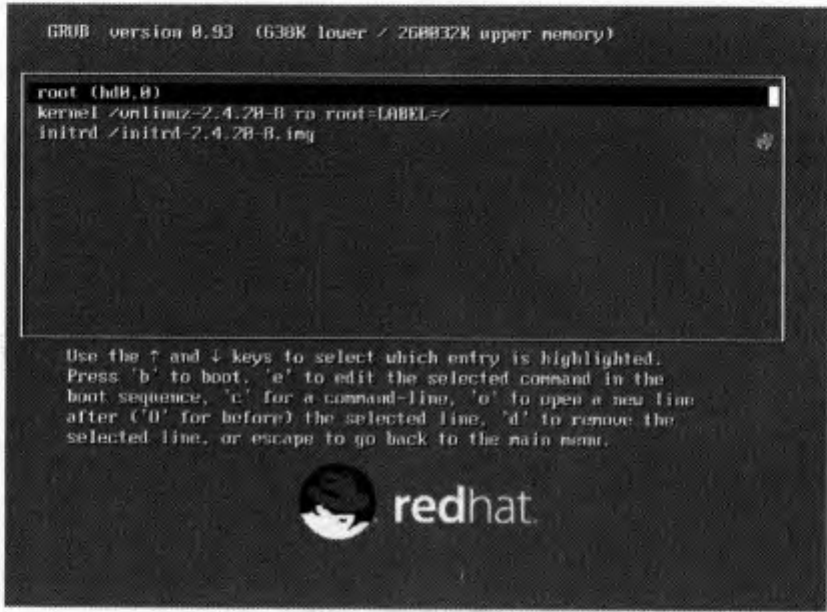


图 4-2 命令行选项菜单

(3) 在命令编辑菜单中输入“single”，设定系统以单用户模式启动，如图 4-3 所示。返回上一级菜单并按“b”键启动系统。

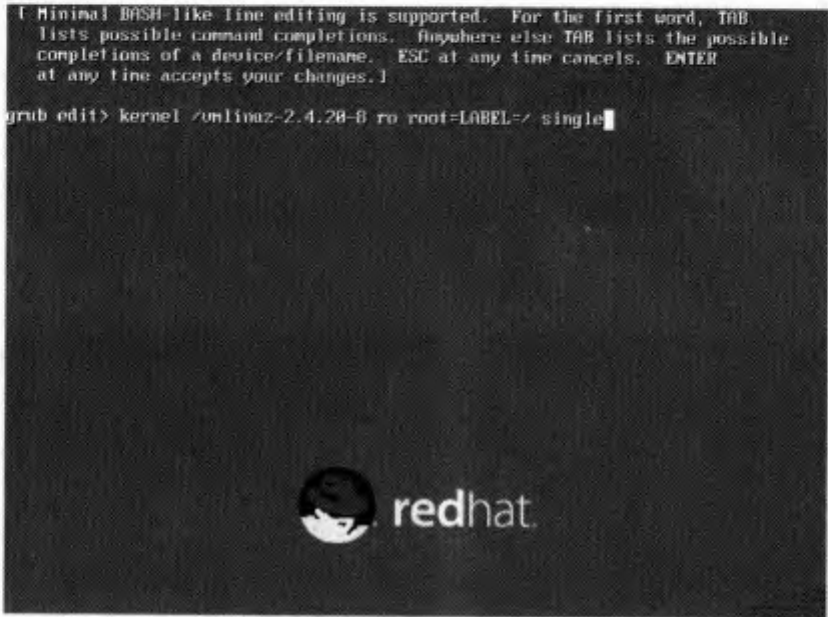


图 4-3 添加参数“single”

(4) 系统启动成功后，将自动进入单用户模式。由于 Linux 密码文件存放在/etc/shadow 文件中，所以可以通过对该文件的修改重新设定 root 密码。

(5) 用 vi 打开/etc/passwd 文件，在 vi 的命令模式下按“x”键逐一删除 root 行中密码字段的内容，并存盘退出，如下所示。

```
#vi /etc/passwd
root::13698:0:99999:7:::           //删除密码字段的字符串
bin:*:13698:0:99999:7:::
daemon:*:13698:0:99999:7:::
... ..
```



(6) 重新启动系统，输入用户名 root（密码为空），可直接登录系统。

4.2.3 以救援模式解决 root 密码丢失问题

Linux 登录密码存放在/etc/shadow 文件中，通过救援模式直接对该文件进行修改可以删除 root 密码。步骤如下：

(1) 用光盘引导系统，并在启动菜单中按“F5”键选择 Linux Rescue 模式，如图 4-4 所示。

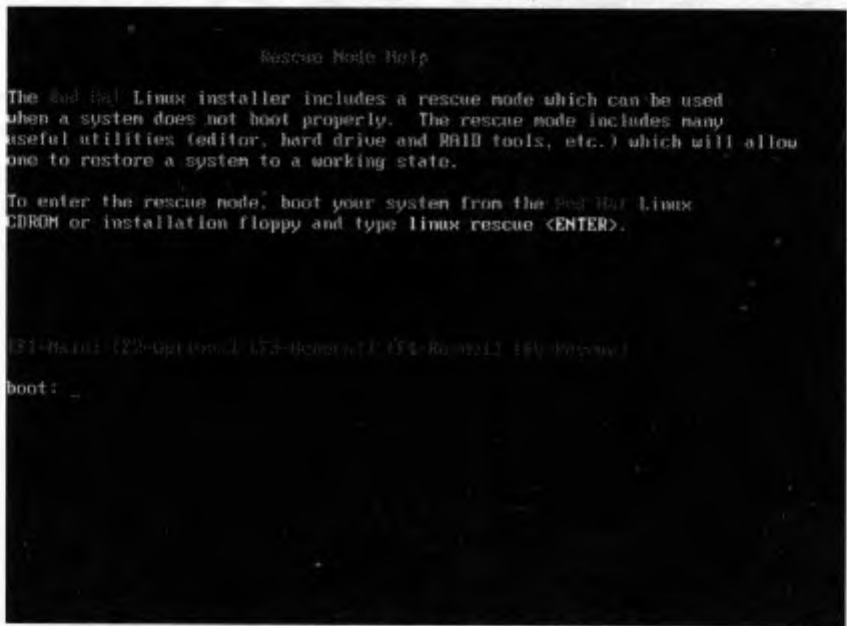


图 4-4 Linux 文件系统被挂载到/mnt/sysimage 目录下

(2) 在图 4-4 中的提示符下输入“linux rescue”并按回车键，出现图 4-5 所示的语言选择界面，按 UP/DOWN 键选择【Chinese (Simplified)】（即简体中文），按回车键进入键盘选项界面，如图 4-6 所示。

(3) 在图 4-6 所示界面中，选择键盘类型，默认为 US，即美式键盘，按【确定】按钮进入网络设置界面，如图 4-7 所示。

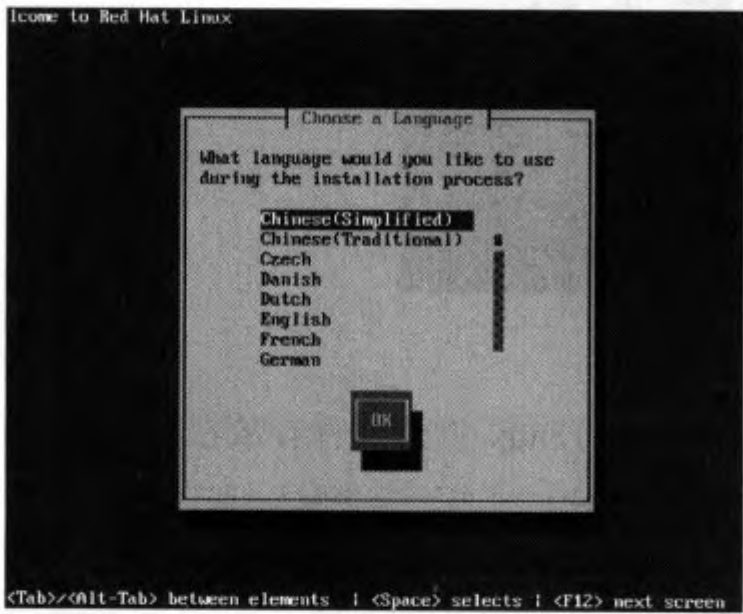


图 4-5 语言选择界面

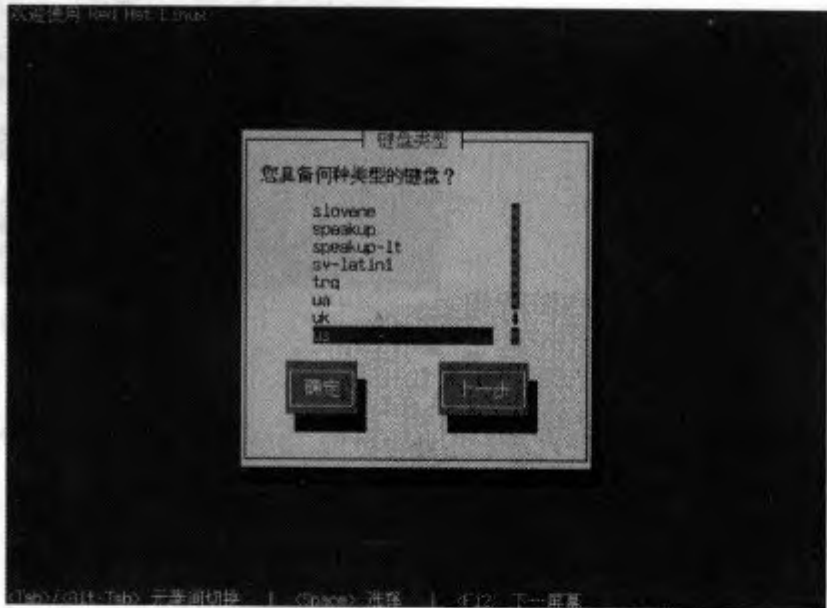


图 4-6 选择键盘

(4) 由于本例仅删除 root 用户的密码，无需网络的支持，因此选择【否】并按回车键，出现图 4-8 所示界面。

(5) 在图 4-8 所示的救援方式选择界面选择【继续】，救援（Linux Rescue）模式会自动搜索存在的 Linux 文件系统，找到后将其自动挂载到/mnt/sysimage 目录下。系统出现如图 4-10 所示提示对话框。



图 4-7 选择网络设置

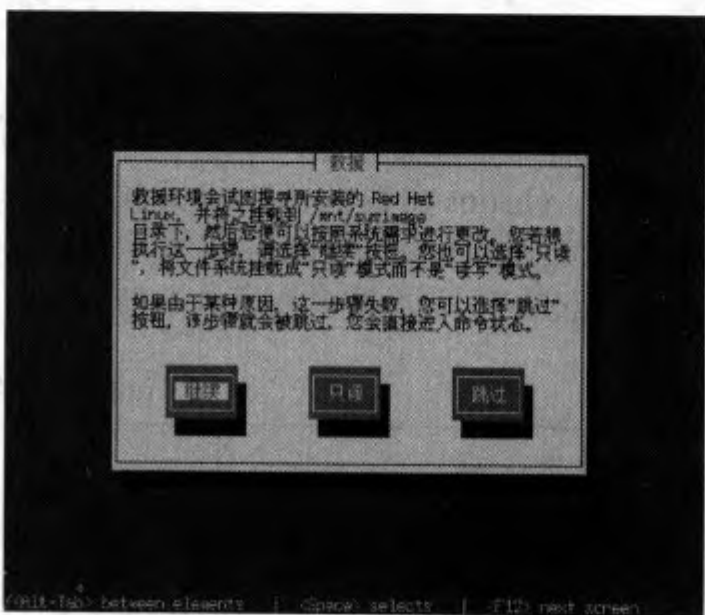


图 4-8 选择救援方式

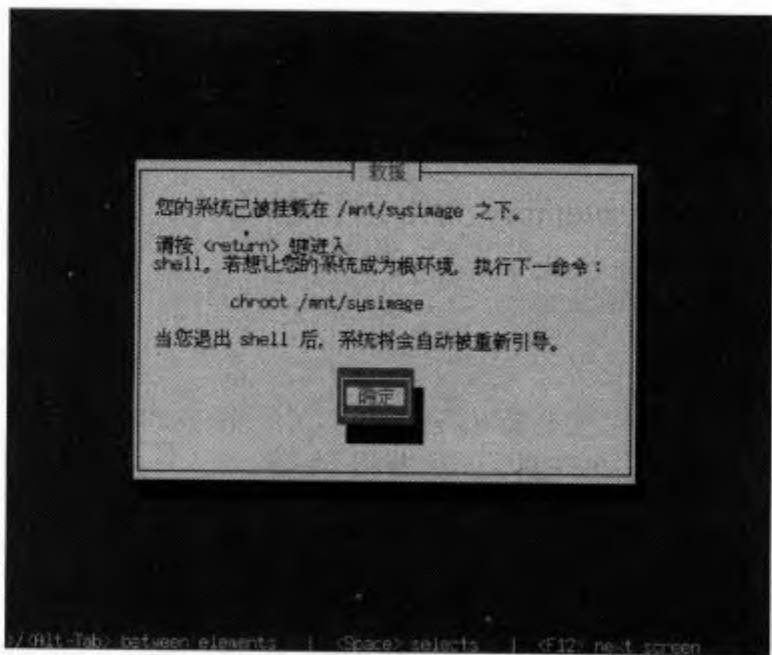


图 4-9 救援模式提示框

(6) 依据提示，原系统被自动加载到/mnt/sysimage 目录下（如果希望原系统的根目录成为当前系统的根目录，需要使用 chroot 命令更改根设置。有关 chroot 命令请参见下一小节）。用 vi 打开/mnt/sysimage/etc/shadow 文件，在命令模式下按“x”键逐一删除 root 行中密码字段的内容，并存盘退出。

```
#vi /etc/passwd
root::14092:0:99999:7:::           //删除密码字段的字符串
bin:*:14092:0:99999:7:::
daemon:*:14092:0:99999:7:::
adm:*:14092:0:99999:7:::
lp:*:14092:0:99999:7:::
sync:*:14092:0:99999:7:::
shutdown:*:14092:0:99999:7:::
... ..
```



(7) 重新启动系统并用 root 登录，此时 root 密码为空。进入系统后，利用 passwd 命令重新修改根用户密码。

4.2.4 chroot 命令：改变根目录

chroot 用于对进程建立新的虚拟根目录 (“/”), 通常是为了保护真正的根目录。以前的 bind 不是很安全，所以用了 chroot，通过改变根目录，能增加系统的安全性，限制登录用户所做的事情。chroot 命令的语法格式如下所示。

```
chroot NEWROOT [COMMAND...] [--help] [--version]
```

其中各选项的具体说明如表 4-5 所示。

表 4-5 chroot 命令的选项说明

选 项	说 明
NEWROOT	指定新的虚拟根目录
COMMAND	执行的命令
--help	显示 chroot 命令的帮助信息
--version	显示 chroot 命令的版本信息

例如，创建一个空目录/tmp/empty，然后创建 ls 命令的链接文件并将其保存在/tmp/empty 目录中。使用 chroot 命令，以/tmp/empty 为根运行 ls 命令，如下所示。

```
$ chroot /tmp/empty /ls -Rl /
/:
total 1023
-rwxr-xr-x  1 0      0      1041745 Aug 16 11:17 ls
```

可以看到，该命令显示的路径已被标识为根目录。

4.3 用户组的管理

用户组的管理主要包括用户组的添加、修改、删除等操作，与系统中的/etc/group 文件和 /etc/gshadow 文件密切相关，涉及 groupadd、groupmod、groupdel 等命令。

4.3.1 解析/etc/group 与/etc/gshadow 文件

用户组文件位于/etc/group，用于存放用户组的账号信息。用户组的添加、删除和修改实际上就是对该文件的更新。该文件的内容任何用户都可以读取，但只有根用户可以修改，如下所示。

```
#ls -l /etc/group
-rw-r--r-- 1 root root 735 Jul 20 07:10 /etc/group
```

/etc/group 文件中每一行对应一个用户组，用 “:” 分隔成 4 个字段，各字段的说明如表 4-6 所示。

表 4-6 /etc/group 文件各字段说明

字 段	说 明
Group_name	用户组的名称
Encrypted_password	用户组的密码，由于安全原因，相应内容已转到 gshadow 文件中，在此仅用“x”占位
GID	用户组标识号，该数字在系统中必须惟一，且不能为负，0~499 为系统预留
User_list	组成员列表

文件/etc/group 内容如下。

```
#cat /etc/group
root:x:0:root //root 用户组，组成员只有 root，GID 为 0
bin:x:1:root,bin,daemon
daemon:x:2:root,bin,daemon //守护进程用户组，组成员包括 root、bin、daemon，GID 为 2
sys:x:3:root,bin,adm
adm:x:4:root,adm,daemon
tty:x:5:
disk:x:6:root
lp:x:7:daemon,lp mem:x:8:
kmem:x:9:
wheel:x:10:root
mail:x:12:mail
news:x:13:news
uucp:x:14:uucp
man:x:15:
games:x:20: //安装 x-windows 服务需要该组
gopher:x:30:
dip:x:40:
ftp:x:50: //ftp 用户组，GID 为 50
lock:x:54:
nobody:x:99:
users:x:100:student3,student4 //普通 users 用户组,GID 为 100,组成员包括 student3、student4,
创建普通用户时//若不指定组则默认为 users 组
rpm:x:37:
dbus:x:81:
utmp:x:22:
utempter:x:35:
avahi:x:70:
mailnull:x:47:
smmisp:x:51:
nscd:x:28:
floppy:x:19:
vcsa:x:69:
```



```
haldaemon:x:68:
rpc:x:32:
rpcuser:x:29:           //远程过程调用用户组
nfsnobody:x:65534:
sshd:x:74:
pcap:x:77:
ntp:x:38:
slocate:x:21:
gdm:x:42:
apache:x:48:
distcache:x:94:
postgres:x:26:          //安装 postgres 数据库服务器时使用
mysql:x:27:              //安装 mysql 服务器时使用
dovecot:x:97:
webalizer:x:67:
squid:x:23:              //安装代理服务器 squid 时使用
named:x:25:              //安装 DNS 服务器时使用
xfs:x:43:
sabayon:x:86:
screen:x:84:
teacher:x:500:teacher,director //teacher 用户的 GID 为 500,组成员包括 teacher、director
```



与 vipw 命令类似,可以使用 vigr 命令直接编辑 group 文件。vigr 命令在功能上相当于“vi /etc/group”命令,但比直接使用 vi 命令更安全。在用 vigr 命令打开 group 文件时,系统会自动对 group 文件加锁,编辑结束自动解锁,从而保证了数据的一致性。

在 Red Hat Linux 9 中,用户组密码的保护机制与用户密码的保护机制一样,采用了 Shadow Password 技术。加密后的用户组密码信息保存在/etc/gshadow 文件中。gshadow 文件只有 root 用户可以读取,文件中每行定义一个用户组的信息,行中各字段用“:”分隔。/etc/gshadow 文件内容如下。

```
#cat /etc/gshadow
root:::root           //root 用户组,组成员只有 root,组密码为空
bin:::root,bin,daemon
daemon:::root,bin,daemon
sys:::root,bin,adm
adm:::root,adm,daemon
tty:::
disk:::root
lp:::daemon,lp
mem:::
kmem:::
wheel:::root
mail:::mail
```



```
news:::news
uucp:::uucp           //UUCP 用户组
man:::
games:::
gopher:::
dip:::
ftp:::                //ftp 用户组, 组成员为空, 密码为空
lock:::
nobody:::
users:::student3,student4 //普通用户组, 组密码为空
rpm:x::
dbus:x::
utmp:x::
utempter:x::
avahi:x::
mailnull:x::
smmisp:x::
nscd:x::
floppy:x::
vcsa:x::
haldaemon:x::
rpc:x::
rpcuser:x::           //RPC 用户组
nfsnobody:x::
sshd:x::
pcap:x::
ntp:x::
slocate:x::
gdm:x::
apache:x::            //apache 服务的密码设置项
distcache:x::
postgres:x::
mysql:x::             //mysql 数据库的密码设置项
dovecot:x::
webalizer:x::
squid:x::
named:x::             //域名服务器的密码设置项
xfs:x::
sabayon:x::
screen:x::
teacher:::
student:::            //用户组 student 已停用
```

从 gshadow 文件内容可以看出，每行共分 4 个字段，各字段的内容定义如表 4-7 所示。



表 4-7 gshadow 文件各字段说明

字 段	说 明
Group_name	用户组的名称
Encrypted_password	用户组的密码，该字段用于保存加密后的口令
Group administrators	用户组的管理员账号，管理员可以对该组进行增、删、改等操作
User_list	组成员列表，列表中多个用户间用“,”分隔

4.3.2 groupadd 命令：添加新用户组

在 Linux 系统中，每个账号都会属于一个用户组。账号的管理应以“组”为单位进行，即先把希望具有相同权限的用户分到同一个用户组，然后再对该用户组的权限进行指定，以此来对用户进行统一管理。

在 Red Hat Linux 9 安装过程中，系统除了要自动创建默认的标准账号外，也会自动创建默认的标准用户组（Standard Groups）账号。除了 root 组是用来组织管理者之外，其他的账号都是提供给应用程序在执行时使用的。标准用户组的说明如表 4-8 所示。

表 4-8 Linux 标准用户组说明

用户组的名称	用户组的标识号	用户组成员列表 (列表中多个用户间用“,”分隔)
root	0	root
bin	1	root,bin,daemon
daemon	2	root,bin,daemon
sys	3	root,bin,adm
adm	4	root,adm,daemon
tty	5	
disk	6	root
lp	7	daemon,lp
mem	8	
kmem	9	
wheel	10	root
mail	12	mail
news	13	news
uucp	14	uucp
man	15	
games	20	
gopher	30	
dip	40	
ftp	50	
lock	54	

续表

用户组的名称	用户组的标识号	用户组成员列表 (列表中多个用户间用“,”分隔)
nobody	99	
users	100	
rpm	37	
dbus	81	
utmp	22	
utempter	35	
avahi	70	
mailnull	47	
smmsp	51	
nscd	28	
floppy	19	
vcsa	69	
haldaemon	68	
rpc	32	
rpcuser	29	
nfsnobody	65534	
sshd	74	
pcap	77	
ntp	38	
slocate	21	
gdm	42	
apache	48	
distcache	94	
postgres	26	
mysql	27	
dovecot	97	
webalizer	67	
squid	23	
named	25	
xfx	43	
sabayon	86	
screen	84	

可以使用 `groupadd` 命令添加用户组，其命令格式如下。

```
groupadd [-g GID [-o]] [-r] [-f] [-K KEY=VALUE] groupname
```

`groupadd` 命令会参照命令行上指定的选项对用户组进行设定。`groupadd` 可用的选项如下所示。



- **-g GID**: 组的 GID, 除非使用了“-o”选项, 否则该值在系统中必须惟一, 且不能为负。该值应大于 499 且大于系统中已存在的任何组的 GID 值。其中 0~499 是系统预留。
- **-r**: 创建小于 500 的系统组。若不指定“-g”选项, 则按递减顺序从小于 500 的可用值中挑选。
- **-f**: 如果所定义的组已经存在, 则退出并显示成功信息。如果同时指定了“-g”和“-f”选项, 而“-g”所指定的组已经存在, 则忽略“-g”的值, 重新指定新的值。
- **-o**: 允许指定不惟一的 GID, 即用新的标识号取代原用户组标识号。
- **-K KEY=VALUE**: 重载/etc/login.defs 中的默认值。例如用 GID_MIN 对用户组标识号最小值进行设定, 用 GID_MAX 对用户组标识号最大值进行设定。
- **groupname**: 指定用户组名称。

例如, 添加一个新的用户组 student, GID 为 502, 命令行如下所示。

```
#groupadd -g 502 student
#tail -1 /etc/group
student:x:502:
```

在 Red Hat Linux 9 中, 可以使用 **useradd** 或 **adduser** 命令添加用户, 但不存在添加用户组的 **addgroup** 命令。如果需要, 可自行创建一个名为 **addgroup** 的链接命令, 链接到 **groupadd** 命令, 如下所示。

```
#ln /usr/sbin/groupadd /usr/sbin/addgroup
```

4.3.3 groupmod 命令: 修改用户组属性

修改用户组属性的命令是 **groupmod**, 其语法格式如下所示。

```
groupmod [-g GID [-o]] [-n new groupname] groupname
```

groupmod 命令会参照命令行中指定的选项对用户组属性进行修改。下列为 **groupmod** 可用的选项及其说明。

- **-g GID**: 为用户组指定新的 GID, 除非使用了“-o”选项, 否则该值在系统中必须惟一, 且不能为负。但该选项并不能自动更新文件的 GID, 必须用户手动修改。
- **-n new groupname**: 更改用户组的名称。
- **groupname**: 用户组名称。

例如, 修改 teacher 用户组的组标识号为 503, 命令如下所示。

```
#groupmod -g 503 teacher
```

例如, 将 teacher 用户组的组标识号改为 550, 用户组名称改为 director, 命令如下所示。

```
#groupmod -g 550 -n director teacher
```

4.3.4 groupdel 命令: 删除用户组

在创建用户账号时, 系统会自动创建该账号所属的用户组。但在删除用户账号时, 所创建的用户组不会自动被删除。删除用户组可以使用 **groupdel** 命令完成, 其语法格式如下所示。

```
groupdel [groupname]
```

例如, 删除 teacher 用户组可以使用如下命令。

```
#groupdel teacher
```

如果希望删除的用户组中仍有用户登录系统, 则无法删除该用户组。必须等该用户组的

所有用户退出系统才能正常删除，如下所示。

```
$whoami
teacher
$groupdel teacher
groupdel: cannot remove user's primary group.
```

4.3.5 newgrp 命令：切换所属用户组

一个用户同时可以属于多个用户组。但用户登录系统后，默认只属于一个用户组，可以使用 newgrp 命令使用户在多个用户组之间进行切换，其语法格式如下所示。

```
newgrp [-] [groupname]
```

选项 “-” 用于重新加载用户工作环境。如果不带 “-” 选项，则在切换用户组时，用户的工作环境（包括当前工作目录等）不会改变。

4.4 用户和用户组管理的安全防范措施

用户和用户组管理是 Linux 系统管理中极为重要的环节，管理不当就可能出现账号锁定、无法登录等诸多问题，从而留下大量的安全隐患。

系统管理员可以通过 who 和 last 等命令来查询当前系统登录用户和用户历史登录情况，对于可疑用户和非法登录用户应及时采取措施。

4.4.1 ac 命令：统计用户连接时间

ac 命令用于显示用户连接时间的统计信息。该命令的语法格式如下所示。

```
ac[-d,--daily-totals][-y,--print-year][-p,--individual-totals][people][-f,--filename]
[-h,--help]
[-a,--all-days][--complain][--reboots][--supplants][--timewarps][--compatibility]
[-V,--version]
[--tw-leniency num][--tw-suspicious num][-z,--print-zeros][--debug]
```

其中各选项的具体说明如表 4-9 所示。

表 4-9 ac 命令的选项说明

选 项	说 明
-d, --daily-totals	按每一天进行统计，而不是仅显示一个对所有天数的总计结果，显示格式如下所示： Jul 3 total 1.17 Jul 4 total 2.10 Jul 5 total 8.23 Jul 6 total 2.10 Jul 7 total 0.30
-y, --print-year	显示日期时，显示年份
-p, --individual-totals	按用户进行统计，显示格式如下所示： bob 8.06 goff 0.60 maley 7.37 root 0.12 total 16.15



续表

选 项	说 明
people	指定进行统计的用户
-f, --file filename	从指定的文件读取信息，而不是从默认的 wtmp 文件
-a, --all-days	显示每天的统计信息，即使期间存在没有用户登录的情况
--complain	当 wtmp 文件中出现丢失记录等问题时，显示错误信息
--reboots	对重启时间进行统计
--supplants	记录用户本次登录到下次登录的时间
--timewarps	计算时间偏差
--compatibility	等同于--timewarps --reboots --supplants
--tw-leniency num	设置时间偏差容忍值，默认为 60 秒
--tw-suspicious num	设置时间偏差近似值，单位为秒
-z, --print-zeros	对于统计结果为 0 的类别，依然显示其值。默认为压缩显示，即不显示值为 0 的项
--debug	冗余模式，显示错误结果
-V, --version	显示版本信息
-h, --help	显示帮助信息

例如，使用不带任何选项的 ac 命令可以查看系统总的连接时间，默认的时间单位是小时，命令如下所示。

```
# ac
total      240.03
```

例如，使用选项 “-p” 可以按用户进行统计，列出所有用户的连接时间，命令如下所示。

```
# ac -p
root      200.02
student   40.01
total     240.03
```

例如，使用选项 “-d”，按天对连接进行汇总，如下所示。

```
# ac -d
Aug 2 total      16.23
Aug 3 total       0.70
Aug 4 total       9.36
Aug 5 total      30.90
Aug 7 total       1.28
Aug 8 total      61.47
Today total     120.09
```

4.4.2 last 命令：查询最近的使用者登录时间

last 命令查找/var/log/wtmp 文件，并显示在此文件创建后的所有登录用户。可以指定时间，来显示特定时间用户的登录情况。该命令的语法格式如下所示。

```
last [-R] [-num] [ -n num ] [-a][-d][-i][-o][-x] [ -f file ] [ -t YYYYMMDDHHMMSS ]
```

其中各选项的具体说明如表 4-10 所示。

表 4-10 last 命令的选项说明

选 项	说 明
-num	指定显示的行数
-n num	指定显示的行数，与“-”相同
-t YYYYMMDDHHMMSS	显示指定时间的登录状态。例如查找某一指定时间谁登录系统
-f file	指定搜索的文件，默认为/var/log/wtmp
-R	压缩显示用户名
-a	显示用户名称
-d	对于非本地用户，Linux 不仅保存远端用户的用户名，而且保存 IP 地址。该选项将 IP 地址转换为主机名
-i	该选项与-d 类似，但仅显示 IP 地址
-o	读取旧类型的 wtmp 文件
-x	显示系统关机及系统运行等级变化

例如，显示最近用户的登录信息，命令如下所示。

```
# last
root pts/2 192.168.20.148 Sat Aug 2 05:52 still logged in
root pts/0 192.168.20.148 Sat Aug 2 05:15 still logged in
root pts/1 :0.0 Sat Aug 2 05:14 still logged in
root pts/0 192.168.20.148 Sat Aug 2 05:11 - 05:15 (00:03)
root :0 Sat Aug 2 05:11 still logged in
reboot system boot 2.4.20-8 Sat Aug 2 04:45 (02:38)
root :0 Sat Aug 2 04:06 - crash (00:38)
reboot system boot 2.4.20-8 Sat Aug 2 03:51 (03:32)
reboot system boot 2.4.20-8 Sat Aug 2 03:45 (00:05)
reboot system boot 2.4.20-8 Sat Aug 2 03:37 (00:13)
reboot system boot 2.4.20-8 Sat Aug 2 03:27 (00:23)
root pts/1 192.168.20.148 Sat Aug 2 02:03 - down (01:22)
root pts/0 192.168.20.148 Fri Aug 1 23:58 - down (03:27)
root :0 Fri Aug 1 23:54 - down (03:31)
reboot system boot 2.4.20-8 Fri Aug 1 23:53 (03:32)
root pts/1 192.168.20.148 Fri Aug 1 19:38 - down (04:14)
root pts/0 192.168.20.148 Fri Aug 1 19:26 - down (04:26)
root :0 Fri Aug 1 19:26 - down (04:26)
reboot system boot 2.4.20-8 Fri Aug 1 19:24 (04:28)
root pts/1 192.168.20.148 Fri Aug 1 18:53 - 19:23 (00:29)
root pts/0 192.168.20.148 Fri Aug 1 18:17 - crash (01:07)
reboot system boot 2.4.20-8 Fri Aug 1 18:12 (05:40)
root tty1 Fri Aug 1 18:10 - down (00:01)
root pts/1 192.168.20.148 Fri Aug 1 07:14 - down (10:56)
root pts/0 192.168.20.148 Fri Aug 1 07:14 - down (10:57)
```

wtmp begins Fri Aug 1 07:10:53 2008

例如，仅显示 10 行记录，命令如下所示。

```
# last -n 10
root pts/2 192.168.20.148 Sat Aug 2 05:52 still logged in
root pts/0 192.168.20.148 Sat Aug 2 05:15 still logged in
root pts/1 :0.0 Sat Aug 2 05:14 still logged in
```



```
root pts/0 192.168.20.148 Sat Aug 2 05:11 - 05:15 (00:03)
root :0 Sat Aug 2 05:11 still logged in
reboot system boot 2.4.20-8 Sat Aug 2 04:45 (02:32)
root :0 Sat Aug 2 04:06 - crash (00:38)
reboot system boot 2.4.20-8 Sat Aug 2 03:51 (03:26)
reboot system boot 2.4.20-8 Sat Aug 2 03:45 (00:05)
reboot system boot 2.4.20-8 Sat Aug 2 03:37 (00:13)
```

```
wtmp begins Fri Aug 1 07:10:53 2008
```

例如，查询最近的使用者登录情况，同时显示系统关机及系统运行等级变化的情况，命令如下所示。

```
# last -x
root pts/2 192.168.20.148 Sat Aug 2 05:52 still logged in
root pts/0 192.168.20.148 Sat Aug 2 05:15 still logged in
root pts/1 :0.0 Sat Aug 2 05:14 still logged in
root pts/0 192.168.20.148 Sat Aug 2 05:11 - 05:15 (00:03)
root :0 Sat Aug 2 05:11 still logged in
runlevel (to lvl 5) 2.4.20-8 Sat Aug 2 04:45 - 07:33 (02:48) //等级 5
runlevel (to lvl 6) 2.4.20-8 Sat Aug 2 03:50 - 03:51 (00:00) //等级 6
reboot system boot 2.4.20-8 Sat Aug 2 03:45 (00:05) //重启
runlevel (to lvl 5) 2.4.20-8 Sat Aug 2 03:37 - 03:50 (00:13) //等级 5
reboot system boot 2.4.20-8 Sat Aug 2 03:37 (00:13) //重启
runlevel (to lvl 5) 2.4.20-8 Sat Aug 2 03:27 - 03:37 (00:10) //等级 5
reboot system boot 2.4.20-8 Sat Aug 2 03:27 (00:23) //重启
shutdown system down 2.4.20-8 Sat Aug 2 03:26 - 03:50 (00:24) //关机
runlevel (to lvl 6) 2.4.20-8 Sat Aug 2 03:26 - 03:26 (00:00) //等级 6
... ..
```

4.4.3 lastb 命令：显示最近登录系统失败的用户信息

与 last 命令类似，lastb 命令查找显示/var/log/btmp 文件的内容，此文件包含所有登录系统失败的用户的的信息。该命令的语法格式如下所示。

```
lastb [-R] [-num] [ -n num ] [ -f file ] [ -t YYYYMMDDHHMMSS ] [-a][-d][-i][-o][-x]
```

其中各选项的具体说明如表 4-11 所示。

表 4-11 lastb 命令的选项说明

选 项	说 明
-num	指定显示的行数
-n num	指定显示的行数
-t YYYYMMDDHHMMSS	显示指定时间的登录状态。例如查找某一指定时间谁登录系统
-f file	指定搜索的文件，默认为/var/log/wtmp
-R	压缩显示用户名
-a	显示用户名称
-d	对于非本地用户，Linux 不仅保存远端用户的用户名，而且保存 IP 地址。该选项将 IP 地址转换为主机名
-i	该选项与-d 类似，但仅显示 IP 地址
-o	读取旧类型的 wtmp 文件
-x	显示系统关机及系统运行等级变化

例如，显示最近登录失败用户的相关信息，命令如下所示。

```
# lastb
```

4.4.4 who 命令：查询当前系统中登录的用户

who 命令用来显示当前登录系统的用户的信息。该命令的语法格式如下所示。

```
who[-a,--all][-b,--boot][-d,--dead][-H,--heading][-i,--idle][--login][-l,--lookup][
-m][-p,--process][--help]
[-q,--count][-r,--runlevel][-s,--short][-t,--time][-T,-w,--mesg][-u,--users][--mess
age][--writable]
[ --version]
```

其中各选项的具体说明如表 4-12 所示。

表 4-12 who 命令的选项说明

选 项	说 明
-a, --all	与 -b -d --login -p -r -t -T -u 相同
-b, --boot	上次系统启动时间
-d, --dead	显示僵尸进程
-H, --heading	显示列标题
-i, --idle	添加空闲时间，以“小时：分钟”表示
--login	显示系统登录进程
-l, --lookup	尝试通过 DNS 解析主机名
-m	显示使用当前标准输入设备的主机名和用户名
-p, --process	显示由 init 启动的活动进程
-q, --count	显示所有登录的用户名以及登录用户的数目
-r, --runlevel	显示当前的运行等级
-s, --short	显示用户名、连线位置、时间，该项为默认
-t, --time	显示上次系统时钟变化
-T, -w, --mesg	显示用户信息状态，以“+”、“-”、和“?”表示
-u, --users	显示登录用户列表
--message	与 -T 相同
--writable	与 -T 相同
--help	显示帮助信息
--version	显示版本信息

例如，显示当前登录系统的用户，命令如下所示。

```
# who //显示当前登录系统的用户
root    tty1      May 31 15:15
root    tty2      May 31 15:15
root    pts/1     May 31 15:07 (192.168.20.3)
root    :0        May 31 15:15
#
```

例如，在显示登录系统的用户信息时，显示标题栏，命令如下所示。



```
# who -H //显示标题栏
NAME      LINE      TIME      COMMENT
root      tty1       May 31 15:15
root      tty2       May 31 15:15
root      pts/1     May 31 15:07 (192.168.20.3)
root      :0        May 31 15:15
#
```

例如，显示空闲时间字段，命令如下所示。

```
# who -H -i
NAME      LINE      TIME      IDLE      PID COMMENT
root      :0        Aug 2 05:11  ?        2687
root      pts/1     Aug 2 05:14 00:53    2842 (:0.0)
root      pts/0     Aug 2 05:15  .        2891 (192.168.20.148)
root      pts/2     Aug 2 05:52 00:15    3162 (192.168.20.148)
#
```

例如，显示用户状态信息，命令如下所示。

```
# who -T -H
NAME      LINE      TIME      COMMENT
root      + tty1    May 31 15:15
root      + tty2    May 31 15:16
root      + pts/1   May 31 15:07 (192.168.20.7)
root      ? :0      May 31 15:15
#
```

例如，只显示使用当前标准输入设备的用户，命令如下所示。

```
# who -H -m
NAME      LINE      TIME      COMMENT
root      pts/0     Aug 2 05:15 (192.168.20.148)
#
```

例如，显示所有登录的用户名以及登录用户的数目，命令如下所示。

```
# who -q
root root root root
# users=4
```

例如，显示由 init 进程创建的活动进程，命令如下所示。

```
# who -H -p
NAME      LINE      TIME      PID COMMENT
          Aug 2 04:45    2557 id=1
          Aug 2 04:45    2558 id=2
          Aug 2 04:45    2559 id=3
          Aug 2 04:45    2560 id=4
          Aug 2 04:45    2561 id=5
          Aug 2 04:45    2562 id=6
          Aug 2 04:45    2563 id=x
```

4.4.5 w 命令：查看登录用户及其进行的操作

在多用户环境下，每个用户都可以登录到系统，执行不同的程序。利用 w 命令可以查看当前有哪些用户已经登录，以及正在进行什么操作。w 命令的语法格式如下所示。

```
w [-h][-u][-s][-f] [-V] [user]
```

其中各选项的具体说明如表 4-13 所示。

表 4-13 w 命令的选项说明

选 项	说 明
-h	不显示标题
-l	显示详细信息
-s	显示简洁信息，不显示登录时间，JCPU 时间和 PCUP 时间
-f	不显示 from 字段（针对远端用户），即登录位置。默认显示
-u	在计算当前进程及 CPU 时间时，忽略用户名
-V	显示版本信息
user	指定查询的用户

例如，显示当前登录用户及其所进行的操作，命令如下所示。

```
# w
08:19:30 up 3:34, 5 users, load average: 1.13, 0.34, 0.22
USER      TTY      FROM            LOGIN@   IDLE   JCPU   PCPU   WHAT
root      :0        -                5:11am   ?      0.00s  0.38s  gnome-session
root      pts/1     :0.0             5:14am   3:04m  0.04s  0.04s  bash
root      pts/0     192.168.20.148   5:15am   0.00s  0.26s  0.01s  w
root      pts/2     192.168.20.148   5:52am   1:08m  0.08s  0.01s  man last
yang      pts/3     192.168.20.148   7:58am   21:08  0.48s  0.46s  vim
```

显示系统当前时间为 08:19:30，即 w 命令被执行的时间；“up 3:34”表示该系统已经运行了 3 小时 34 分钟；“5 users”表示当前共有 5 位用户登录此系统；“1.13, 0.34, 0.22”表示系统在前 1 分钟、5 分钟、15 分钟的平均负载，其值越小表示系统负载越低，性能越佳。从第二行开始每行共 8 个字段，用来显示用户正在进行的操作、占用的系统资源等情况，详细字段说明如表 4-14 所示。

表 4-14 w 命令显示输出中各字段说明

字 段	说 明
USER	登录系统的用户名
TTY	用户登录的终端
FROM	显示用户从何处登录，如果是本地登录，此字段为“-”，如果是远程登录，则显示远程主机的主机名或 IP 地址，其中“:0.0”表示该用户是从 X-Window 以命令行模式登录
LOGIN@	显示用户登录系统时的时间
IDLE	显示用户空闲时间
JCPU	显示与该终端相关的所有进程所消耗的 CPU 时间
PCPU	表示 CPU 执行程序所消耗的时间
WHAT	显示用户正在执行的程序的名称，如果正在执行命令行模式下的命令，则显示用户环境名称

例如，通过命令“w -f”显示登录用户信息，但不显示登录位置（即 from 字段），命令如下所示。

```
# w -f
08:07:27 up 3:22, 5 users, load average: 0.00, 0.00, 0.00
USER      TTY      LOGIN@  IDLE   JCPU   PCPU   WHAT
root      :0        5:11am   ?      0.00s  0.38s  gnome-session
```



```
root pts/1 5:14am 2:52m 0.04s 0.04s bash
root pts/0 5:15am 0.00s 0.25s 0.01s w -f
root pts/2 5:52am 55:57 0.08s 0.01s man last
yang pts/3 7:58am 9:05 0.48s 0.46s vim
```

例如，以精简模式显示登录用户信息，命令如下所示。

```
# w -s
08:11:59 up 3:27, 5 users, load average: 0.00, 0.00, 0.00
USER TTY FROM IDLE WHAT
root :0 - ? gnome-session
root pts/1 :0.0 2:57m bash
root pts/0 192.168.20.148 0.00s w -s
root pts/2 192.168.20.148 1:00m man last
yang pts/3 192.168.20.148 13:37 vim
```

例如，在显示登录用户信息时，不显示标题栏，命令如下所示。

```
# w -h //显示登录用户但不显示标题栏
root :0 - 5:11am ? 0.00s 0.38s gnome-session
root pts/1 :0.0 5:14am 2:58m 0.04s 0.04s bash
root pts/0 192.168.20.148 5:15am 0.00s 0.25s 0.00s w -h
root pts/2 192.168.20.148 5:52am 1:01m 0.08s 0.01s man last
yang pts/3 192.168.20.148 7:58am 14:58 0.48s 0.46s vim
```

例如，只查询指定用户 root 的信息，可以在命令 w 后指定用户名 root，命令如下所示。

```
# w root
08:18:42 up 3:33, 5 users, load average: 0.00, 0.00, 0.00
USER TTY FROM LOGIN@ IDLE JCPU PCPU WHAT
root :0 - 5:11am ? 0.00s 0.38s gnome-session
root pts/1 :0.0 5:14am 3:04m 0.04s 0.04s bash
root pts/0 192.168.20.148 5:15am 0.00s 0.26s 0.01s w root
root pts/2 192.168.20.148 5:52am 1:07m 0.08s 0.01s man last
```

4.4.6 使用 userdel 和 groupdel 命令删除不需要的用户和用户组

从前几节的学习中，可以看到 Linux 系统提供了一些预置账号，而用户可能根本不需要这些账号。如果用户确实不需要这些账号，可以将其删掉，因为这些账号名是公开的，很容易受到攻击。用下面的命令删除一些不必要的用户。

```
#userdel adm //一些用不到的用户可以被删除
#userdel lp
#userdel sync
#userdel shutdown
#userdel halt
#userdel news
#userdel uucp
#userdel operator
#userdel gopher
#userdel ftp //如果安装了匿名 ftp 服务，请不要删除
//一些用不到的用户组也可以删除
#groupdel adm
#groupdel lp
#groupdel news
#groupdel uucp
#groupdel dip
#groupdel pppusers
#groupdel popusers //如果安装了 POP 服务器，请不要删除
#groupdel slipusers
```


4.5 其他用户和用户组管理的常用命令

Linux 还提供了一些增强用户及用户组管理的命令，如 `finger`、`groups`、`id`、`logname`、`newgrp`、`rwho`、`write` 等。通过这些命令，管理员可以方便地掌握用户和用户组的相关信息。

4.5.1 finger 命令：查找并显示用户信息

`finger` 命令用于显示系统用户的相关信息，其语法格式如下所示。

```
finger [-l][-m][-s][-p] [user] [user@host]
```

其中各选项的具体说明如表 4-15 所示。

表 4-15 finger 命令的选项说明

选 项	说 明
-l	以多行格式显示用户的登录名、实名、终端名、写状态(如果禁止写则在终端名后有符号“*”)、空闲时间、登录时间、办公地点、办公电话、用户主目录、家庭电话、登录 Shell、电子邮件、用户主目录以及.plan、.project、.pgpkey、.forward 等文件中的内容
-m	不匹配真实用户名
-s	列出用户的登录名、实名、终端名、写状态(如果禁止写则在终端名后有符号“*”)、空闲时间、登录时间、办公地点、办公电话
-p	列出用户的帐号信息，与-l 基本相同，但不显示.plan、.project、.pgpkey 文件中的内容
user	指定查找的用户名
user@host	指定查找的“用户名@主机名”

例如，列出当前登录用户的相关信息，命令如下所示。

```
# finger -l //显示用户信息
Login: root Name: root
Directory: /root Shell: /bin/bash
On since Sat Aug 2 05:11 (CST) on :0 (messages off)
On since Sat Aug 2 05:14 (CST) on pts/1 from :0.0
5 hours 37 minutes idle
On since Sat Aug 2 05:15 (CST) on pts/0 from 192.168.20.148
On since Sat Aug 2 05:52 (CST) on pts/2 from 192.168.20.148
3 hours 40 minutes idle
New mail received Sat Aug 2 10:50 2008 (CST)
Unread since Fri Dec 21 15:39 2007 (CST)
No Plan.

Login: yang Name: (null)
Directory: /home/yang Shell: /bin/bash
On since Sat Aug 2 07:58 (CST) on pts/3 from 192.168.20.148
19 minutes 7 seconds idle
No mail.
No Plan.
```

例如，显示指定用户 yang 的用户信息，命令如下所示。

```
# finger yang //显示用户 yang 的用户信息
Login: yang Name: (null)
```



```
Directory: /home/yang          Shell: /bin/bash
On since Sat Aug 2 07:58 (CST) on pts/3 from 192.168.20.148
 21 minutes 2 seconds idle
No mail.
No Plan.
```

4.5.2 groups 命令：显示指定用户所属的用户组

用于显示指定用户所属的用户组，如果未指定用户则显示当前用户所属的组。该命令的语法格式如下所示。

```
groups [--help][--version] [USERNAME]
```

其中各选项的具体说明如表 4-16 所示。

表 4-16 groups 命令的选项说明

选 项	说 明
--help	显示命令 groups 的帮助信息
--version	显示命令 groups 的版本信息

例如，显示当前用户 root 所属的用户组，命令如下所示。

```
#groups
root bin daemon sys adm disk wheel
```

可以看到，root 用户所属的用户组除了 root 外，还包括 bin、daemon、sys、adm、disk 和 wheel。这与/etc/group 文件中所标记的内容一致，如下所示。

```
# cat /etc/group | grep root
root:x:0:root
bin:x:1:root,bin,daemon
daemon:x:2:root,bin,daemon
sys:x:3:root,bin,adm
adm:x:4:root,adm,daemon
disk:x:6:root
wheel:x:10:root
```

例如，显示当前登录的用户 yang 所属的用户组，命令如下所示。

```
[yang@localhost yang]$ groups
yang
```

可以看到，用户 yang 仅属于用户组 yang。

4.5.3 id 命令：显示用户的 UID、GID

id 命令用来显示指定用户或当前用户的用户 ID 或组 ID。该命令的语法格式如下所示。

```
id [-a][ -g, --group][ -G, --groups][ -n, --name][ -r, --real][ -u,
--user][ --help][ --version]
```

其中各选项的具体说明如表 4-17 所示。

表 4-17 id 命令的选项说明

选 项	说 明
-a	为与其他版本兼容，忽略提示
-g, --group	仅显示有效组 ID
-G, --groups	显示所有的用户组 ID

续表

选 项	说 明
-n, --name	显示名称而不是数字标识
-r, --real	显示真实用户 ID 而不是有效用户 ID
-u, --user	仅显示有效的用户 ID
--help	显示此帮助信息并离开
--version	显示版本信息并离开

例如，显示当前用户的 UID、GID 和用户所属的组列表，命令如下所示。

```
# id //显示当前用户信息
uid=0(root) gid=0(root) groups=0(root),1(bin),2(daemon),3(sys),4(adm),6(disk),10(wheel)
显示 root 用户的用户信息，命令如下所示。
```

```
# id root
uid=0(root) gid=0(root) groups=0(root),1(bin),2(daemon),3(sys),4(adm),6(disk),10(wheel)
例如，显示当前用户的用户组 GID，命令如下所示。
```

```
# id -g //显示用户群组 ID
0
```

例如，显示所有的用户组 GID，命令如下所示。

```
# id -G //显示所有群组 ID
0 1 2 3 4 6 10
```

例如，显示指定用户 yang 的用户信息，命令如下所示。

```
# id yang //显示用户 yang 的信息
uid=500(yang) gid=500(yang) groups=500(yang)
```

例如，显示所有的群组名称，命令如下所示。

```
# id -G -n
root bin daemon sys adm disk wheel
```

4.5.4 logname 命令：显示登录用户的用户名

logname 命令用来显示登录用户的用户名。该命令的语法格式如下所示。

```
logname [--help][--version]
其中各选项的具体说明如表 4-18 所示。
```

表 4-18 logname 命令的选项说明

选 项	说 明
--help	显示帮助信息
--version	显示 logname 命令的版本信息

例如，显示登录用户的用户名，命令如下所示。

```
# logname //显示登录用户的用户名
root
```

4.5.5 newgrp 命令：修改用户的用户组

newgrp 命令用于修改当前用户的用户组。一个用户可以同时属于多个用户组，其中有一个主用户组，该主用户组通常在创建用户时指定或由系统默认设定。Newgrp 命令的语法格式



如下所示。

```
newgrp [ groupname ]
其中 groupname 指定修改后的用户组名称。例如，下面的命令实现了用户组的修改。
#groups //显示当前用户 root 所属的用户组
root bin daemon sys adm disk wheel
#id root //显示根用户的组信息，当前用户的主用户组是 root
uid=0(root) gid=0(root) groups=0(root),1(bin),2(daemon),3(sys),4(adm),6(disk),10(wheel)
#newgrp bin // 改变当前用户的主用户组为 bin
#id
uid=0(root) gid=1(bin) groups=0(root),1(bin),2(daemon),3(sys),4(adm),6(disk),10(wheel)
#newgrp //不指定转换的用户组时，系统默认转换为登录时的用户组
#id
uid=0(root) gid=0(root) groups=0(root),1(bin),2(daemon),3(sys),4(adm),6(disk),10(wheel)
//用户的主用户组已变更为 root
```

4.5.6 rwho 命令：查看登录本地的用户信息

rwho 命令与 who 命令类似，只是 rwho 可以显示局域网中多台主机的用户登录信息。如果 rwho 命令在 11 分钟内没有收到主机的反馈信息，则会认为该主机已关闭，rwho 命令也不会报告已知的登录用户。该命令的语法格式如下所示。

```
rwho [-a]
如果用户 1 分钟没有进行任何操作，显示状态为空闲；如果用户 1 小时没有操作，忽略该用户（除非使用选项“-a”）。
例如，显示本地局域网中的所有用户登录信息，如下所示。
# rwho -a //显示局域网内所有用户
```

4.5.7 users 命令：显示当前登录的用户

users 命令用来显示当前登录的用户，该命令的语法格式如下所示。

```
users [--help][--version] [ FILE ]
其中各选项的具体说明如表 4-19 所示。
```

表 4-19 users 命令的选项说明

选 项	说 明
--help	显示 users 命令的帮助信息
--version	显示 users 命令版本信息
FILE	依据指定的文件进行输出。如果没有指定，默认为/var/run/utmp

例如，显示所有登录的用户，命令如下所示。

```
# users //显示所有登录的用户
root root root root yang
```

4.5.8 write 命令：将信息实时传递给登录的用户或终端

使用 write 命令，可以将信息实时传递给登录的用户或终端。通过在命令行指定用户或终端可以控制信息发往何处。

例如，向登录用户 yang 发送即将关机的信息，命令如下所示。


```
# who
root      :0          Aug  2 05:11
root      pts/1        Aug  2 05:14 (:0.0)
root      pts/0        Aug  2 05:15 (192.168.20.148)
root      pts/2        Aug  2 05:52 (192.168.20.148)
yang      pts/3        Aug  2 07:58 (192.168.20.148)
#write yang
System will shutdown soon!
EOF
```

用户 yang 会收到如下的信息。

```
Message from root@localhost.localdomain on pts/0 at 11:10 ...
System will shutdown soon!
EOF
```

4.5.9 whoami 命令：显示当前用户的信息

whoami 用于显示当前用户的名称，该命令的语法格式如下所示。

```
whoami [--help][--version]
```

其中各选项的具体说明如表 4-20 所示。

表 4-20 whoami 命令的选项说明

选 项	说 明
--help	显示命令 whoami 的帮助信息
--version	显示命令 whoami 的版本信息

例如，显示当前登录用户的用户名称，命令如下所示。

```
# whoami
Root
```

4.6 用户及用户组管理命令的综合应用

用户和用户组管理是 Linux 系统管理中极为重要的环节，管理不当就可能出现账号锁定、无法登录等诸多问题。综合使用多个命令可以较好地解决上述问题。

4.6.1 解决/etc/nologin 文件引起的普通用户无法登录问题

/etc/nologin 文件给系统管理员提供了一种在 Linux 系统维护期间禁止用户登录的方式。为了提高系统的安全性和防止数据的不同步现象，系统管理员在对系统进行维护时，通常建立 0 字节的/etc/nologin 文件，如果存在/etc/nologin 文件，非 root 用户的登录尝试会失败。建立/etc/nologin 文件的命令行如下。

```
touch /etc/nologin
```

这种保护机制也被应用到系统的关机、重启等环节中。系统在关闭时通常要自动创建 nologin 文件，禁止所有用户登录，然后处理相关程序的退出停止工作，并在即将关闭系统前删除/etc/nologin 文件。如果系统在还未正常删除/etc/nologin 文件前出现突然断电停机等情况，



就会出现因 `nologin` 文件未删除而引起的普通用户无法登录现象。此时只需根用户登录系统并删除 `nologin` 文件，问题即可解决。建议用户在使用系统过程中，不要采用直接关闭电源等不正常手段关闭系统。不正常关闭系统会对系统造成严重的损害。



由于 SSH 不查看 `/etc/nologin` 文件，所以即使 `/etc/nologin` 文件存在，使用 SSH 仍可登录系统。

4.6.2 创建用户共享目录

Red Hat Linux 9 可以利用用户组来组织用户，进而加强系统的安全性。在 `/etc/passwd` 配置文件中，每个用户的 UID 通常与其 GID 是一致的。但有些时候有些用户可能属于一个公共部门，也可能正在从事一个公共项目。通过创建一个共同的组和一个共享目录，可以使这个组的所有成员自由读取共享目录中所输入的文件。

例如，项目组成员 `student1`、`student2`、`student3` 要创建一个公共目录，用于存放项目开发中的文件。可以将这些用户组织到一个公共组并创建一个共享目录。通过设置 SGID 位，允许该组中的任何用户把文件复制到共享目录中，并允许该组其他成员可以读取共享目录中的所有文件。具体操作步骤如下。

(1) 使用 `groupadd` 命令创建一个公共组。

```
#groupadd student
```

(2) 使用 `useradd` 命令创建 `student1`、`student2` 和 `student3` 账号，并给每个用户分配一个密码。

```
#useradd student1 -g student           //添加 student1 用户，并指定为 student 组
#passwd student1
Changing password for user student1
New UNIX password:
Retype new UNIX password:
passwd: all authentication tokens updated successfully
#useradd student2 -g student           //添加 student2 用户，并指定为 student 组
#passwd student2
Changing password for user student2
New UNIX password:
Retype new UNIX password:
passwd: all authentication tokens updated successfully
#useradd student3 -g student           //添加 student3 用户，并指定为 student 组
#passwd student3
Changing password for user student3
New UNIX password:
Retype new UNIX password:
passwd: all authentication tokens updated successfully
```

(3) 创建一个名为 `/home/student` 的新共享目录。对于拥有这个目录的用户和组，使用 `chmod 770 /home/student` 命令，给这个目录分配读、写和执行权限 (rwx)，命令如下所示。

```
#mkdir /home/student
#chmod 770 /home/student
```

(4) 在共享目录上配置 SGID 位，使得拥有这个目录的组中的所有用户都能够拥有所有

权级权限。

```
#chmod g+s /home/student
```

(5) 使用 `chgrp` 命令为这个目录设置所有权。

```
#chgrp student /home/student
```

4.6.3 对/etc/shadow 文件的编辑导致用户密码丢失

当用户尝试修改密码时，如果有人正在编辑/etc/shadow 文件，有可能导致该用户修改的密码失效。例如，系统管理员正在使用编辑器编辑/etc/shadow 文件时，用户 teacher 同时修改了用户密码。用户 teacher 在 /etc/shadow 的记录行未修改前如下所示。

```
teacher:$1$v4pZr72Z$wBrJyrolSI622P4nI.UJE.:13698:0:99999:7:::
```

使用 `passwd` 命令修改密码后，teacher 用户在/etc/shadow 文件的记录行变为如下所示。

```
teacher: $1$9y0JsFWl$FW4uZTnP6r7hLykdf7jQ71.:13698:0:99999:7:::
```

用户 teacher 在系统管理员完成对/etc/shadow 文件的编辑、存盘退出前成功地修改了密码，此时系统管理员保存文件时系统会提示如下信息。

```
WARNING: The file has been changed since reading it !!!  
Do you really want to write to it(y/n)?
```

如果系统管理员此时回答 y，用户 teacher 就会丢失他新设定的密码。用户 teacher 的 /etc/shadow 行就会恢复成原来的值，如下所示。

```
teacher:$1$v4pZr72Z$wBrJyrolSI622P4nI.UJE.:13698:0:99999:7:::
```

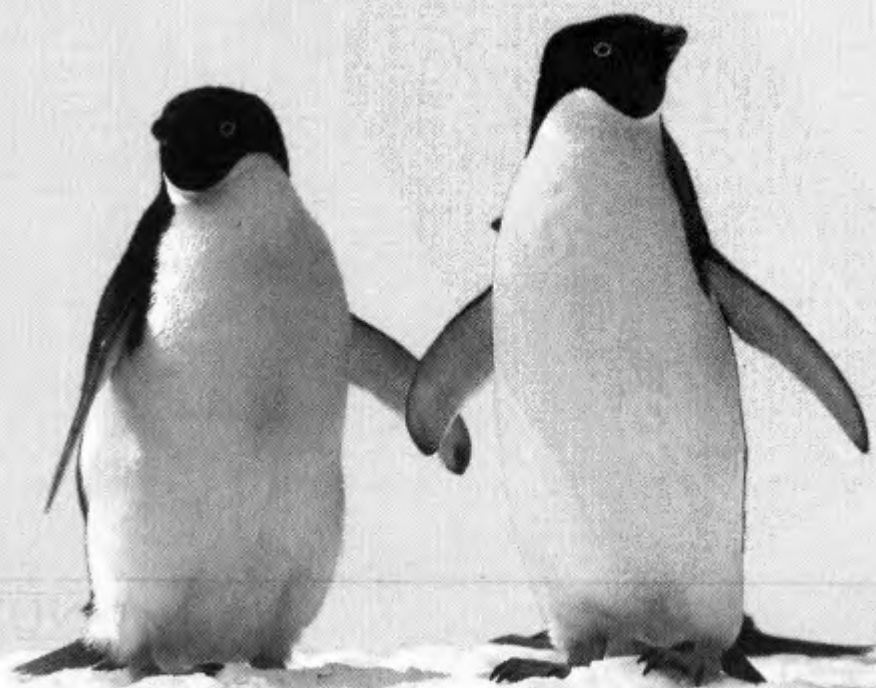
所以，修改/etc/passwd 或/etc/shadow 时一定要谨慎，应当尽量使用命令来修改，而不是编辑该文件。

UNIX

第5章

磁盘管理

系统管理员的一项重要工作就是管理磁盘。磁盘作为存储数据的重要载体，在如今日渐庞大的软件资源面前显得尤为重要。随着硬件成本的逐年下降，磁盘容量越来越大，但同时磁盘管理难度也越来越高。良好的磁盘管理可以进一步节省存储空间，提高系统效能，节约成本。磁盘管理通常涉及磁盘的分区、格式化、空间管理等内容。



5.1 Linux 磁盘分区工具

磁盘分区工具是系统管理员必需掌握和经常使用的软件，对磁盘进行分区是安装操作系统前的必经步骤。常用的磁盘管理工具包括 fdisk、cfdisk、parted、sfdisk 等。

5.1.1 Linux 磁盘分区简介

由于硬盘容量动辄几百 GB，为了方便管理，通常将硬盘分成若干分区。对于普通用户而言，每个分区都可以视为独立的磁盘来使用。硬盘的分区方案被记录到“磁盘分区表”中。通常磁盘分区表由 4 个部分组成，每个部分定义一个分区的信息。因此，原始概念中一个硬盘最多只能建立 4 个分区，称为“主分区”。当系统中存在多个主分区时，必须指定一个主分区为“活动分区”。活动分区上的操作系统在系统引导时将被自动引导。

由于硬盘容量越来越大，仅仅 4 个分区已无法满足用户的需要，为此引入了“扩展分区”的概念。扩展分区由扩展磁盘分区表维护，可以在扩展分区内划分若干更小的称为“逻辑分区”的分区。逻辑分区没有数量的限制，如果删除了扩展分区，其下的所有逻辑分区都将被删除。扩展分区的必须建立在主分区上，即扩展分区必须属于 4 个主分区之一。主分区、扩展分区与逻辑分区的关系如图 5-1 所示。

在 Red Hat Linux 9 安装到硬盘上以前，必须建立硬盘分区。理论上，在硬盘空间足够时，可以建立任意数量的分区（挂载点），但除非在极为特殊的条件下，否则建议至少建立以下三个分区。

- swap 分区。

swap 分区的系统标识为 82。swap 分区实际上是虚拟内存的一部分。所谓虚拟内存（Virtual Memory）技术是指在物理内存无法提供足够的处理空间时，多余的数据就会暂时被写入硬盘指定的分区中，待物理内存可处理时再由该分区中将数据移入。虚拟内存技术突破了物理内存的限制，使程序可以使用大于实际物理内存的空间。一般情况下，swap 分区应该大于或等于物理内存，不应小于 32 MB。建议物理内存存在 2GB 以下时，将 swap 分区设为物理内存的 2~2.5 倍。如果物理内存存在 2GB 以上，一般将 swap 分区设为与物理内存相同即可。

- boot 分区。

boot 分区中存放着操作系统的内核。建议使用 32MB 以上的空间。

- 根分区 (/)。

根分区是整个操作系统的根目录，几乎所有的文件都位于此目录之下，因此它的容量越

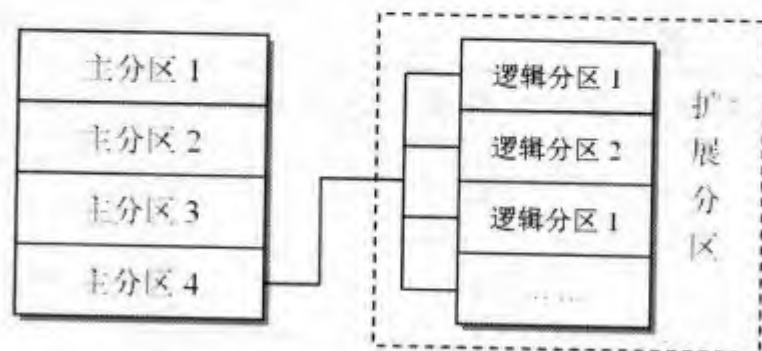


图 5-1 主分区、扩展分区与逻辑分区



大越好，建议将硬盘中剩余的空间都提供给根分区使用。

Linux 还提供了多种分区创建方案，见表 5-1。

表 5-1 Linux 常用的分区方案

分区方案	说明
根分区、swap 分区	适用于磁盘空间有限的计算机
根分区、/boot 分区、swap 分区	较大磁盘空间的典型配置，也是 Red Hat Linux 9 的默认配置
根分区、/boot 分区、/var 分区、swap 分区	可以避免日志文件大小失控
根分区、/boot 分区、/home 分区、swap 分区	对于一台为许多用户提供服务的计算机，可以帮助控制用户占用的空间大小



在 Red Hat Linux 9 安装过程中，系统默认把根 (/) 和 /boot 目录安装到独立的分区上。

Linux 支持的分区类型众多，每一种分区类型对应一个系统标识号。系统标识号由两位 16 进制数表示，包含了操作系统类型、文件系统类型、分区容量等信息。常用分区类型与系统标识如表 5-2 所示。

表 5-2 常用分区类型与系统标识

系统标识	分区类型	系统标识	分区类型	系统标识	分区类型	系统标识	分区类型
0	Empty	1e	Hidden Win95 FAT16	80	Old Minix	be	Solaris boot
1	FAT12	24	NEC DOS	81	Minix / old Linux	bf	Solaris
2	XENIX root	39	Plan 9	82	Linux swap/ So	c1	DRDOS/sec (FAT-12)
3	XENIX usr	3c	PartitionMagic	83	Linux	c4	DRDOS/sec (FAT-16)
4	FAT16 <32M	40	Venix 80286	84	OS/2 hidden C:	c6	DRDOS/sec (FAT-16)
5	Extended	41	PPC PReP Boot	85	Linux extended	c7	Syrinx
6	FAT16	42	SFS	86	NTFS volume set	da	Non-FS data
7	HPFS/NTFS	4d	QNX4.x	87	NTFS volume set	db	CPM/CTOS.
8	AIX	4e	QNX4.x 2nd part	88	Linux plaintext	de	Dell Utility
9	AIX bootable	4f	QNX4.x 3rd part	8e	Linux LVM	df	BootIt
a	OS/2 Boot Manager	50	OnTrack DM	93	Amoeba	e1	DOS access

续表

系统标识	分区类型	系统标识	分区类型	系统标识	分区类型	系统标识	分区类型
b	Win95 FAT32	51	OnTrack DM6 Aux	94	Amoeba BBT	e3	DOS R/O
c	Win95 FAT32 (LBA)	52	CP/M	9f	BSD/OS	e4	SpeedStor
e	Win95 FAT16 (LBA)	53	OnTrack DM6 Aux	a0	IBM Thinkpad hi	eb	BeOS fs
f	W95 Ext'd (LBA)	54	OnTrackDM6	a5	FreeBSD	ee	EFI GPT
10	OPUS	55	EZ-Drive	a6	OpenBSD	ef	EFI (FAT-12/16/)
11	Hidden FAT12	56	Golden Bow	a7	NeXTSTEP	f0	Linux/PA-RISC b
12	Compaq diagnostics	5c	Priam Edisk	a8	Darwin UFS	f1	SpeedStor
14	Hidden FAT16 <32M	61	SpeedStor	a9	NetBSD	f4	SpeedStor
16	Hidden FAT16	63	GNU HURD or Sys	ab	Darwin boot	f2	DOS secondary
17	Hidden HPFS/NTF	64	Novell Netware	b7	BSDI fs	fd	Linux raid auto
18	AST SmartSleep	65	Novell Netware	b8	BSDI swap	fe	LANstep
1b	Hidden Win95 FAT3	70	DiskSecure Multi-Boot	bb	Boot Wizard hidden	ff	BBT

5.1.2 fdisk 命令：磁盘分区工具

fdisk 是一种功能强大的磁盘分区工具，它不仅适合于 Linux 系统，在 Windows 及 MS-DOS 中也有广泛应用。fdisk 命令的语法格式如下所示。

```
fdisk [-u] [-v] [-b sectorsize] [-C cyls] [-H heads] [-S sects] [-l] [-u] [-s partition] [device]
```

其中常用的设备名包括下面几种。

- /dev/hda: 第一块 IDE 硬盘。
- /dev/hdb: 第二块 IDE 硬盘。
- /dev/sda: 第一块 SCSI 硬盘。
- /dev/sdb: 第二块 SCSI 硬盘。

设备名的命名规则是：对于 IDE 设备，设备名为/dev/hd[a-h]；对于 SCSI 设备，设备名为/dev/sd[a-p]；对于 ESDI 设备，使用/dev/ed[a-d]；XT 设备使用/dev/xd[ab]。其中的设备是指整个物理磁盘设备。fdisk 命令的常用选项及说明如表 5-3 所示。



表 5-3 fdisk 常用选项及说明

选 项	说 明
-b sectorsize	定义磁盘扇区的大小，有效值包括 512、1024 和 2048。磁盘扇区大小是系统内核已知的参数，设置该选项仅仅是对以前内核版本的支持，以及满足重载内核的需要
-C cyls	定义磁盘柱面数
-H heads	定义磁盘分区的磁头数，该值专指分区中使用的磁头数目。有效值为 255 或 16
-S sects	定义磁盘分区中每磁道包含的扇区数目，有效值为 63
-l	列出指定设备的分区表并退出。如果没有指定设备，则使用/proc/partitions 文件提供的信息
-u	在列出分区表时，以扇区为单位显示，而不是以柱面为单位
-s partition	在标准输出设备中列出指定分区的大小，默认以 block 为单位
-v	显示版本信息
device	指定的设备

例如，使用选项“-l”显示当前设备的分区表并退出，命令行如下。

```
# fdisk -l
Disk /dev/sda: 8589 MB, 8589934592 bytes
255 heads, 63 sectors/track, 1044 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Device      Boot      Start         End      Blocks   Id  System
/dev/sda1   *           1          13       104391    83   Linux
/dev/sda2           14        1044     8281507+   8e   Linux LVM
```

使用选项“-s”可以显示指定分区的大小。在上例中分别显示/dev/sda1 和/dev/sda2 分区大小，命令行如下。

```
# fdisk -s /dev/sda1
104391
# fdisk -s /dev/sda2
8281507
```

如果没有指定具体的分区，而仅指定设备名，则显示该设备中所有分区大小的总和，命令如下所示。

```
# fdisk -s /dev/sda
8388608
```

使用选项“-v”可以显示 fdisk 的版本号，命令行如下。

```
# fdisk -v
fdisk (util-linux 2.13-pre7)
```

5.1.3 fdisk 命令交互模式

使用命令“fdisk 设备名”，系统会进入分区管理交互模式。用户可以通过输入 fdisk 内部命令，执行相应的磁盘管理操作，fdisk 的内部命令如表 5-4 所示。

表 5-4 fdisk 内部命令

内 部 命 令	说 明
a	设置或取消可引导标志，硬盘驱动器应至少有一个主分区被设为可引导分区
d	删除一个分区
l	列举出已知的分区类型，共有 100 多个类型，其中 83 是 Linux 分区，82 是 Linux swap 分区
m	显示可使用的 fdisk 命令

续表

内部命令	说明
n	创建一个新分区
p	显示当前的分区表
q	退出 fdisk, 但不自动保存修改
t	更改分区的 ID
v	对当前的分区表进行验证
w	保存修改结果并退出 fdisk
b	编辑 BSD 磁盘标签
c	设置 DOS 兼容标志
o	创建一个新的 DOS 分区表
s	创建一个空的 Sun 磁盘标签
u	改变显示单位

例如, 对/dev/sda 设备使用 fdisk 命令, 如下所示。

```
# fdisk /dev/sda
The number of cylinders for this disk is set to 1044.
There is nothing wrong with that, but this is larger than 1024,
and could in certain setups cause problems with:
1) software that runs at boot time (e.g., old versions of LILO)
2) booting and partitioning software from other OSs
   (e.g., DOS FDISK, OS/2 FDISK)
Command (m for help):
```

fdisk 命令进入交互模式后, 首先提示当前的磁盘柱面数为 1044。由于当前磁盘的柱面数大于 1024, 可能会与一些软件产生冲突, 包括在系统启动时运行的软件 (例如旧版的 LILO 程序) 以及其他操作系统的启动和分区软件 (例如 MS-DOS 或 OS/2 下的 fdisk 命令)。在 “Command(m for help):” 提示符下键入 “m”, fdisk 将显示所有内部命令列表, 如下所示。

```
Command (m for help): m
Command action
a  toggle a bootable flag
b  edit bsd disklabel
c  toggle the dos compatibility flag
d  delete a partition
l  list known partition types
m  print this menu
n  add a new partition
o  create a new empty DOS partition table
p  print the partition table
q  quit without saving changes
s  create a new empty Sun disklabel
t  change a partition's system id
u  change display/entry units
v  verify the partition table
w  write table to disk and exit
x  extra functionality (experts only)
```

输入命令 “p”, 列出当前/dev/sda 设备上的分区信息, 如下所示。



```
Command (m for help):p

Disk /dev/sda: 8589 MB, 8589934592 bytes
255 heads, 63 sectors/track, 1044 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

Device      Boot      Start        End      Blocks    Id System
/dev/sda1   *           1          13       104391    83  Linux
/dev/sda2                14       1044    8281507+   8e  Linux LVM
```

可以看到，当前/dev/sda 设备容量为 8 589MB，包括 255 个磁头，每磁道 63 扇区，磁盘柱面总共 1044。列表以柱面为单位进行显示，其中 1 柱面等于 8 225 280 字节。如果用户希望以扇区为单位进行显示，可以在执行 fdisk 命令时加入 “-u” 选项，命令行如下。

```
# fdisk /dev/sda
The number of cylinders for this disk is set to 1044.
There is nothing wrong with that, but this is larger than 1024,
and could in certain setups cause problems with:
 1) software that runs at boot time (e.g., old versions of LILO)
 2) booting and partitioning software from other OSs
   (e.g., DOS FDISK, OS/2 FDISK)
Command (m for help): p           //输入 p 指令

Disk /dev/sda: 8589 MB, 8589934592 bytes
255 heads, 63 sectors/track, 1044 cylinders, total 16777216 sectors
Units = sectors of 1 * 512 = 512 bytes

Device      Boot      Start        End      Blocks    Id System
/dev/sda1   *          63       208844       104391    83  Linux
/dev/sda2                208845    16771859    8281507+   8e  Linux LVM
```

可以看到，显示内容已经改为以扇区为单位，其中 1 扇区等于 512 字节。用户也可以在进入 fdisk 交互模式后执行内部命令 “u”，完成单位切换，如下所示。

```
Command (m for help): u
Changing display/entry units to cylinders           //将显示单位改为柱面

Command (m for help): u
Changing display/entry units to sectors             //将显示单位改为扇区
```

使用内部命令 “d” 可以删除已经存在的分区，如下所示。

```
Command (m for help): p

Disk /dev/sda: 8589 MB, 8589934592 bytes
255 heads, 63 sectors/track, 1044 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

Device      Boot      Start        End      Blocks    Id System
/dev/sda1   *           1          13       104391    83  Linux
/dev/sda2                14       1044    8281507+   8e  Linux LVM

Command (m for help): d
Partition number (1-4): 1 //输入要删除的分区号，其中 1 指/dev/sda1 分区，2 指/dev/sda2 分区
Command (m for help): p

Disk /dev/sda: 8589 MB, 8589934592 bytes
255 heads, 63 sectors/track, 1044 cylinders, total 16777216 sectors
```


Units = sectors of 1 * 512 = 512 bytes

Device	Boot	Start	End	Blocks	Id	System
/dev/sda2		208845	16771859	8281507+	8e	Linux LVM//dev/sda1 分区已被删除



如果出现误删除，只需用“q”命令退出 fdisk，则本次所做的所有操作均不会生效，用户只需重新进入 fdisk 并重新进行配置。如果使用“w”命令退出 fdisk，则 fdisk 会在退出前进行写入操作，被删除的分区将无法恢复。另外，如果删除了扩展分区，则扩展分区下的逻辑分区都会被删除。

使用内部命令“n”可以创建一个新分区，如下所示。

```
Command (m for help):p           //显示当前磁盘分区

Disk /dev/sda: 8589 MB, 8589934592 bytes
255 heads, 63 sectors/track, 1044 cylinders, total 16777216 sectors
Units = sectors of 1 * 512 = 512 bytes

   Device Boot      Start         End      Blocks   Id  System
/dev/sda2             208845     16771859     8281507+   8e  Linux LVM

Command (m for help): n           //新建一个分区
Command action                 //内部命令“e”表示创建扩展分区， 内部命令“p”表示创建主分区
  e   extended
  p   primary partition (1-4)
p                                   //创建主分区
Partition number (1-4): 1         //指定分区号为 1
First sector (63-16777215, default 63)://输入扇区开始位置，默认为 63
Using default value 63
Last sector or +size or +sizeM or +sizeK (63-208844, default 208844): //输入扇区结束位置，默认为 208844
Using default value 208844

Command (m for help): p           //显示当前磁盘分区

Disk /dev/sda: 8589 MB, 8589934592 bytes
255 heads, 63 sectors/track, 1044 cylinders, total 16777216 sectors
Units = sectors of 1 * 512 = 512 bytes

   Device Boot      Start         End      Blocks   Id  System
/dev/sda1             63           208844      104391    83  Linux
/dev/sda2             208845     16771859     8281507+   8e  Linux LVM
```

对于新建的分区，系统默认的分区类型为 83，即 Linux 分区。如果希望设置其他的分区类型，可以使用内部命令“-t”，如下所示。

```
Command (m for help): t           //改变系统标识码
Partition number (1-4): 1         //操作的分区
Hex code (type L to list codes): 1 //列出所有可用的标识码
 0 Empty                1e Hidden W95 FAT1  80 Old Minix          be Solaris boot
 1 FAT12                 24 NEC DOS          81 Minix / old Linux  bf Solaris
 2 XENIX root            39 Plan 9           82 Linux swap / So    c1 DRDOS/sec (FAT-12)
 3 XENIX usr             3c PartitionMagic   83 Linux              c4 DRDOS/sec
(FAT-16)
 4 FAT16 <32M           40 Venix 80286       84 OS/2 hidden C:     c6 DRDOS/sec (FAT-16)
 5 Extended             41 PPC PReP Boot     85 Linux extended     c7 Syrix
```




```
6 FAT16          42 SFS          86 NTFS volume set  da Non-FS data
7 HPFS/NTFS      4d QNX4.x        87 NTFS volume set  db CP/M / CTOS /
8 AIX            4e QNX4.x 2nd part  88 Linux plaintext  de Dell Utility
9 AIX bootable   4f QNX4.x 3rd part  8e Linux LVM       df BootIt
a OS/2 Boot Manag 50 OnTrack DM      93 Amoeba          e1 DOS access
b W95 FAT32       51 OnTrack DM6 Aux  94 Amoeba BBT       e3 DOS R/O
c W95 FAT32 (LBA) 52 CP/M          9f BSD/OS          e4 SpeedStor
e W95 FAT16 (LBA) 53 OnTrack DM6 Aux a0 IBM Thinkpad hi eb BeOS fs
f W95 Ext'd (LBA) 54 OnTrackDM6    a5 FreeBSD         ee EFI GPT
10 OPUS          55 EZ-Drive        a6 OpenBSD         ef EFI (FAT-12/16/
11 Hidden FAT12   56 Golden Bow     a7 NeXTSTEP        f0 Linux/PA-RISC b
12 Compaq diagnost 5c Priam Edisk    a8 Darwin UFS      f1 SpeedStor
14 Hidden FAT16 <3 61 SpeedStor    a9 NetBSD          f4 SpeedStor
16 Hidden FAT16   63 GNU HURD or Sys ab Darwin boot     f2 DOS secondary
17 Hidden HPFS/NTF 64 Novell Netware b7 BSDI fs         fd Linux raid auto
18 AST SmartSleep 65 Novell Netware b8 BSDI swap       fe LANstep
1b Hidden W95 FAT3 70 DiskSecure Mult bb Boot Wizard hid ff BBT
1c Hidden W95 FAT3 75 PC/IX

Hex code (type L to list codes): 8e //输入 Linux 分区 16 进制标识码 8e
Command (m for help): p           //显示当前磁盘分区
```

```
Disk /dev/sda: 8589 MB, 8589934592 bytes
255 heads, 63 sectors/track, 1044 cylinders, total 16777216 sectors
Units = sectors of 1 * 512 = 512 bytes
```

Device	Boot	Start	End	Blocks	Id	System
/dev/sda1		63	208844	104391	8e	Linux LVM//系统标识码已改为 8e
/dev/sda2		208845	16771859	8281507+	8e	Linux LVM



在创建逻辑分区前必须创建扩展分区。

5.1.4 使用 fdisk 划分磁盘分区实例

例如，对当前硬盘 (/dev/sda, 容量 8.5GB) 划分磁盘分区：其中主分区 2 个，一个 100MB，一个 5GB，其余空间分配给扩展分区；扩展分区被划分为两个逻辑分区，一个 2GB，一个 1GB。所有分区默认类型均为 83，命令执行过程如下。

```
# fdisk /dev/sda
The number of cylinders for this disk is set to 1044.
There is nothing wrong with that, but this is larger than 1024,
and could in certain setups cause problems with:
1) software that runs at boot time (e.g., old versions of LILO)
2) booting and partitioning software from other OSs
   (e.g., DOS FDISK, OS/2 FDISK)
```

```
Command (m for help): p //显示当前设备的分区表
```

```
Disk /dev/sda: 8589 MB, 8589934592 bytes
255 heads, 63 sectors/track, 1044 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
```


Device	Boot	Start	End	Blocks	Id	System	//当前/dev/sda 设备
--------	------	-------	-----	--------	----	--------	-----------------

分区表为空

创建 100MB 的主分区，可以用来作为/boot 分区，如下所示。

```
Command (m for help): n           //创建分区
Command action                     //选择创建分区类型
  e   extended
  p   primary partition (1-4)
p                                   //创建主分区
Partition number (1-4): 1          //输入分区号
First cylinder (1-1044, default 1): //输入起始柱面
Using default value 1
Last cylinder or +size or +sizeM or +sizeK (1-1044, default 1044): +100M //此处可以
输入柱面号也可以直接输入空间大小，输入 "+100M"
Command (m for help): p           //显示创建结果

Disk /dev/sda: 8589 MB, 8589934592 bytes
255 heads, 63 sectors/track, 1044 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
```

Device	Boot	Start	End	Blocks	Id	System
/dev/sda1		1	13	104422	83	Linux //已创建 100M 主分区，类型 83

然后创建 5GB 的主分区，可以作为根分区使用，如下所示。

```
Command (m for help): n
Command action
  e   extended
  p   primary partition (1-4)
p
Partition number (1-4): 2
First cylinder (14-1044, default 14):
Using default value 14
Last cylinder or +size or +sizeM or +sizeK (14-1044, default 1044): +5000M // 输入
该分区大小
```

```
Command (m for help): p
```

```
Disk /dev/sda: 8589 MB, 8589934592 bytes
255 heads, 63 sectors/track, 1044 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
```

Device	Boot	Start	End	Blocks	Id	System
/dev/sda1		1	13	104422	83	Linux
/dev/sda2		14	622	4891792+	83	Linux //已创建 5GB 主分区，类型为 83

至此，主分区创建完毕。接下来创建扩展分区，如下所示。

```
Command (m for help): n
Command action
  e   extended
  p   primary partition (1-4)
e                                   //创建扩展分区
Partition number (1-4): 3          //输入分区号
First cylinder (623-1044, default 623):
Using default value 623
Last cylinder or +size or +sizeM or +sizeK (623-1044, default 1044): //将默认的所有
剩余空间分配给扩展分区
```




Using default value 1044

Command (m for help): p

Disk /dev/sda: 8589 MB, 8589934592 bytes
255 heads, 63 sectors/track, 1044 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

Device	Boot	Start	End	Blocks	Id	System
/dev/sda1		1	13	104422	83	Linux
/dev/sda2		14	622	4891792+	83	Linux
/dev/sda3		623	1044	3389715	5	Extended //已创建扩展分区, 类型为 5

扩展分区创建完成后, 可以在扩展分区上创建逻辑分区。首先创建 2GB 的逻辑分区, 如下所示。

Command (m for help): n

Command action

l logical (5 or over)

p primary partition (1-4)

l

//输入“l”, 创建逻辑分区

First cylinder (623-1044, default 623):

Using default value 623

Last cylinder or +size or +sizeM or +sizeK (623-1044, default 1044): +2000M//创建 2GB 的逻辑分区

Command (m for help): p

Disk /dev/sda: 8589 MB, 8589934592 bytes
255 heads, 63 sectors/track, 1044 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

Device	Boot	Start	End	Blocks	Id	System
/dev/sda1		1	13	104422	83	Linux
/dev/sda2		14	622	4891792+	83	Linux
/dev/sda3		623	1044	3389715	5	Extended
/dev/sda5		623	866	1959929+	83	Linux //已创建 2GB 逻辑分区, 类型为 83

然后将剩余空间 (约 1GB) 分配给最后一个逻辑分区, 如下所示。

Command (m for help): n

Command action

l logical (5 or over)

p primary partition (1-4)

l

//创建逻辑分区

First cylinder (867-1044, default 867):

Using default value 867

Last cylinder or +size or +sizeM or +sizeK (867-1044, default 1044): //将扩展分区剩余空间分配给最后一个分区

Using default value 1044

Command (m for help): p

//显示创建的分区表

Disk /dev/sda: 8589 MB, 8589934592 bytes
255 heads, 63 sectors/track, 1044 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

Device	Boot	Start	End	Blocks	Id	System
--------	------	-------	-----	--------	----	--------


```
/dev/sda1      1      13      104422      83  Linux
/dev/sda2      14     622     4891792+   83  Linux
/dev/sda3     623    1044     3389715     5  Extended
/dev/sda5     623     866     1959929+   83  Linux
/dev/sda6     867    1044     1429784+   83  Linux
```

分区表编辑完成，按“w”键，执行写入操作并退出操作，如下所示。

```
Command (m for help): w
The partition table has been altered!
```

如果放弃本次所做的修改，按“q”键，直接退出 fdisk 交互模式，如下所示。

```
Command (m for help): q
```

5.1.5 parted 命令：磁盘分区工具

parted 是 Linux 下的磁盘分区与分区调整工具，可以对 Linux 下的分区进行创建、删除或重新划分大小。使用 parted 命令要求分区所在设备不能正在被使用（即分区不能被挂载，并且交换空间不能被启用），可以使用 umount 命令来卸载分区，使用 swapoff 命令来关闭硬盘驱动器上的交换空间。parted 命令的语法格式如下所示。

```
parted [-h][-i][-s][-v] [device]
```

其中各选项的具体说明如表 5-5 所示。

表 5-5 parted 命令的选项说明

选 项	说 明
-h	显示帮助信息
-i	交互模式
-s	脚本模式
-v	显示 parted 命令的版本信息
device	指定分区设备

例如，使用 parted 命令对/dev/sda 进行分区操作，如下所示。

```
# parted /dev/sda
GNU Parted 1.6.3
Copyright (C) 1998, 1999, 2000, 2001, 2002 Free Software Foundation, Inc.
This program is free software, covered by the GNU General Public License.

This program is distributed in the hope that it will be useful, but WITHOUT ANY
WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A
PARTICULAR PURPOSE. See the GNU General Public License for more details.

Using /dev/sda
Information: The operating system thinks the geometry on /dev/sda is 1044/255/63.
Therefore, cylinder 1024 ends at 8032.499M.
(parted)
```

5.1.6 parted 命令交互模式

在启动 parted 命令后，系统会进入交互模式，用户可以通过 parted 的内部命令完成对分区的管理。常用的 parted 内部命令如表 5-6 所示。



表 5-6 parted 的内部命令及说明

内 部 命 令	说 明
check NUMBER	对文件系统进行检测
cp [FROM-DEVICE] FROM-NUMBER TO-NUMBER	把文件系统复制到另一个分区
help [COMMAND]	显示可用的帮助命令或指定命令的帮助信息
mklabel	创建一个新的磁盘标签
mkfs NUMBER FS-TYPE	在分区上创建 FS-TYPE 类型的文件系统
mkpart PART-TYPE [FS-TYPE] START END	创建一个分区
mkpartfs PART-TYPE FS-TYPE START END	创建分区和指定的操作系统
move NUMBER START END	移动分区
name NUMBER NAME	以分区号命名
print [free NUMBER all]	显示分区表
quit	退出
rescue START END	修复丢失的分区
resize NUMBER START END	重新划分分区
rm NUMBER	删除分区
select DEVICE	选择需要配置的设备
set NUMBER FLAG STATE	改变分区的 FLAG 标志
toggle [NUMBER [FLAG]]	切换分区的 FLAG 标志



其中,NUMBER 指 Linux 分区号,主分区号为 1~4,逻辑分区号大于等于 5.START 和 END 用于磁盘定位,可以采用如 4GB 或 10%的方式来表示。如果该值为负数,表示从磁盘的末端开始计算,例如,“-1s”表示从磁盘末端开始的第一个扇区,即磁盘的最后一个扇区开始计算。

例如,启动了 parted 后,键入以下命令查看分区表。

```
(parted) print
Disk geometry for /dev/sda: 0.000-8192.000 megabytes
Disk label type: msdos
Minor      Start      End      Type      Filesystem  Flags
1          0.031     101.975  primary   ext3        boot
2          101.975   7679.509 primary   ext3
3          7679.509  8189.384 primary   linux-swap
(parted)
```

其中,第一行显示了磁盘的类型、磁盘容量,第二行显示磁盘的标识名,其余各行输出显示磁盘分区表。在分区表中,Minor 表示分区号,例如,分区号码为 1 的分区和/dev/sda1 相对应。Start (开始) 和 End (结束) 值以 MB 为单位。Type (类型) 取值为主分区、扩展分区或逻辑分区。Filesystem (文件系统) 是文件系统的类型,可以是 ext2、ext3、FAT、hfs、jfs、linux-swap、ntfs、reiserfs、hp-ufs、sun-ufs、xfs 等。Flags (标志) 列表示分区被设置的标志,可用的标志有 boot、root、swap、hidden、raid、lvm 和 lba。

如果要选择不同的设备,不需要重新启动 parted,在 parted 交互模式下使用 select 命令

即可。例如，要查看 `/dev/sdb`，只需输入下面命令，就可以查看或配置 `/dev/sdb` 的分区表。

```
(parted)select /dev/sdb
Using /dev/sdb
```

如果要在 `parted` 下删除分区 `/dev/sdb2`，首先需要查看当前的分区表来判定要删除的分区码，如下所示。

```
(parted) print
Disk geometry for /dev/sda: 0.000-8192.000 megabytes
Disk label type: msdos
Minor      Start      End      Type      Filesystem  Flags
1          0.031     101.975  primary   ext3        boot
2          101.975   6679.509 primary   ext3
3          6679.509  7679.509 primary   linux-swap
4          7679.509  8189.384 primary   ext3
```

使用内部命令 `rm` 来删除分区号为 4 的磁盘分区，如下所示。

```
(parted) rm 4
```

与 `fdisk` 不同，`parted` 不需再执行写入操作，只需按 `Enter` 键，改变就会立即发生。因此，在执行前请仔细检查。删除了分区后，可以使用 `print` 命令来确认分区在分区表中是否已被删除，如下所示。

```
(parted) print
Disk geometry for /dev/sda: 0.000-8192.000 megabytes
Disk label type: msdos
Minor      Start      End      Type      Filesystem  Flags
1          0.031     101.975  primary   ext3        boot
2          101.975   6679.509 primary   ext3
3          6679.509  7679.509 primary   linux-swap
```

也可以使用下面的命令来确认内核是否已经知道该分区已被删除。

```
# cat /proc/partitions
```

退出 `parted` 程序后，如果需要，还应对 `/etc/fstab` 文件进行更新，删除已删除分区的对应行。



正在被使用的设备上的分区是无法被删除的。

可以使用内部命令 `resize` 重新划分分区大小。分区类型必须是下列之一：`FS-TYPEs`、`ext3`、`ext2`、`fat32`、`fat16`、`hfsx`、`hfs+`、`hfs` 和 `linux-swap`。要重划大小的分区上已用的空间必须大于新建的大小。使用内部命令 `resize` 前可以使用 `print` 命令查看分区号，然后在 `resize` 命令后跟随分区号以及以 `MB` 为单位的起始点和终止点。例如：

```
(parted) resize 2 107 2048
```

分区被重新划分大小后，使用 `print` 命令来确认分区已被正确地重新划分了大小，并且具备正确的分区类型和文件系统类型。也可以在重新引导系统后，使用 `df` 命令来查看分区是否已被挂载，以及分区的大小、类型。



正在被使用的设备上不允许使用 `parted` 命令重新划分分区的大小。



5.1.7 sfdisk 命令：Linux 分区工具

sfdisk 是 Linux 分区工具，主要有 4 个功能：显示分区大小，显示设备上的分区，检测设备上的分区和对设备进行重新分区。该命令的语法格式如下所示。

```
sfdisk[-s,-show-size][-c,--id][-l,--list][-d,--dump][-i,--increment][-uS,-uB,-uC,-uM][-T,--list-types]
        [-D,--DOS][-R,--re-read][-N#][-n][-Ofile][-Ifile][-v,--version][-?,--help]
[-g,--show-geometry]
        [-x,--show-extended][-L,--Linux][-q,--quiet][-C#,--cylinders#][H#,--heads#]
[-f,--force]
        [-S#,--sectors#]
```

其中各选项的具体说明如表 5-7 所示。

表 5-7 sfdisk 命令的选项说明

选 项	说 明
-s , -show-size	列出分区大小
-c , --id	显示或改变分区 ID
-l , --list	列出每一设备的分区信息
-d, --dump	与-l 相同，但显示格式更适于以后的输入
-i, --increment	柱面数，从 1 而不是从 0 开始计算
-uS, -uB, -uC, -uM	单位为：扇区/块/柱面/MB
-T , --list-types	列出已知的分区类型
-D , --DOS	DOS 兼容模式，浪费较少的磁盘空间
-R , --re-read	通知内核重读分区表
-N#	使用分区号更改分区
-n	不写入磁盘
-O file	将被重写的扇区写入文件
-I file	恢复扇区
-v , --version	显示版本号
-? , --help	显示帮助信息
-g , --show-geometry	显示磁盘的几何信息
-x , --show-extended	列出扩展分区
-L , --Linux	忽略与 Linux 不兼容的警告信息
-q , --quiet	静默模式，不显示警告信息
-C# , --cylinders #	设置柱面
-H# , --heads #	设置磁头
-S# , --sectors #	设置扇区
-f , --force	强制执行

例如，显示分区信息，命令如下所示。


```
# sfdisk -l //显示分区信息
Disk /dev/sda: 1044 cylinders, 255 heads, 63 sectors/track
Units = cylinders of 8225280 bytes, blocks of 1024 bytes, counting from 0

Device   Boot   Start    End  #cyls   #blocks  Id System
/dev/sda1 *      0+      12     13-    104391   83  Linux
/dev/sda2          13     978    966    7759395   83  Linux
/dev/sda3        979    1043     65     522112+   82  Linux swap
/dev/sda4          0      -      0         0    0  Empty
```

例如，显示磁盘/sda 以及分区/sda1、/sda2 和/sda3 的大小，命令如下所示。

```
# sfdisk -s /dev/sda //显示/sda 磁盘大小
8388608
# sfdisk -s /dev/sda1 //显示/sda1 分区大小
104391
# sfdisk -s /dev/sda2 //显示/sda2 分区大小
7759395
# sfdisk -s /dev/sda3 //显示/sda3 分区大小
522112
```

5.2 常用的磁盘管理命令

磁盘管理涉及磁盘容量统计、坏块检测、硬盘参数设定、磁盘数据同步等操作，常用命令包括 badblocks、du、df、fdformat 以及 sync 等。综合应用这些命令可以帮助管理员及时了解磁盘的使用情况。

5.2.1 badblocks 命令：检查设备坏块

badblocks 命令可以用来检查磁盘或磁盘的一个分区，检测其中是否有坏道，以进行相应的修复。badblocks 是 e2fsprogs 软件包的一部分，该命令的语法格式如下所示。

```
badblocks [-b block_size] [-i input_file] [-o output_file] [-s][-v][-w][-n][-f] [-c
blocks_at_once] [-p num_passes] device [ last-block ] [ start-block ]
```

其中各选项的具体说明如表 5-8 所示。

表 5-8 badblocks 命令的选项说明

选 项	说 明
-s	显示检查进度
-v	运行命令报告详细处理信息
-w	进行写入检测
-b block-size	指定块的大小，单位为字节数
-c number of blocks	指定同时被检测的块数，默认为 16。增加该值将提高检测效率，但会占用更多内存
-f	通常为了保护文件系统，badblocks 不会对正挂载的文件系统进行读写等检测操作，该选项强制进行检测
-i input_file	从指定文件中读入坏块列表，对于已经列出的坏块，不再进行检测
-o output_file	将检测出的坏块写入指定的文件，如果不指定文件，默认从标准检出设备显示



续表

选 项	说 明
-p num_passes	指定连续扫描的次数。默认为 0，即仅进行一次扫描
-n	使用非破坏读写模式
-o<文件>	将检测结果写入到指定文件
-s	通过输出检测的块号，显示扫描的进度
-v	冗余模式
-w	使用写入模式进行检测
device	指定检测的设备
last-block	指定检测的最后一个块，如果未指定则默认为设备的最后一个块
start-block	指定检测的第一个块，如果未指定则默认为设备的第一个块

例如，检测软磁盘/dev/fd0 的坏道，命令如下所示。

```
# badblocks /dev/fd0 //检测磁盘的坏道
```

5.2.2 du 命令：统计文件和目录占用的磁盘空间

可以使用 du 命令统计文件和目录占用的磁盘空间情况，其语法格式如下所示。

```
du [-s][--a][--c][--k][--b][--l][--x][--m][--h]
```

如果没有指定文件或目录名，则默认对当前目录进行统计。命令 du 的选项说明如表 5-9 所示。

表 5-9 du 命令选项说明

选 项	说 明
-s	仅显示文件或目录占用的块数，默认 1 块等于 1024 字节
-a	显示所有文件及其子目录占用的数据块数
-c	在显示结果最后添加一个总计（系统默认设置）
-k	以 1024 字节即 1KB 为单位进行统计（系统默认设置）
-b	以字节为单位进行统计
-l	对于硬链接文件，系统会计算多次
-x	对不同文件系统上的同名目录不进行统计
-m	以 1024K 字节即 1MB 为单位进行统计
-h	以人们习惯的方式显示输出结果

例如，使用选项“-a”对用户 kitty 主目录进行统计（包括其中的子目录），命令行如下。

```
# du -a ~kitty
4 /home/stud1/test.2
4 /home/stud1/test.4
4 /home/stud1/test.3
4 /home/stud1/pic/hen
4 /home/stud1/pic/cock
16 /home/stud1/pic
4 /home/stud1/homework.1
```



```
4      /home/stud1/test.1
```

```
44     /home/stud1
```

例如，使用选项“-s”对指定的 kitty 用户进行统计，命令行如下。

```
# du -s ~ kitty
```

```
44     /home/stud1
```

如果以字节为单位进行统计，命令行如下。

```
# du -b ~ kitty
```

```
4096   /home/stud1/pic
```

```
8192   /home/stud1
```

如果以 1024 字节即 1KB 为单位进行统计，命令行如下。

```
# du -k ~ kitty
```

```
16     /home/stud1/pic
```

```
44     /home/stud1
```

如果以 1024K 字节即 1MB 为单位进行统计，命令行如下。

```
# du -m ~ kitty
```

```
1      /home/stud1/pic
```

```
1      /home/stud1
```

如果希望给显示结果加上总计，应使用选项“-c”，命令行如下。

```
# du -c ~ kitty
```

```
16     /home/stud1/pic
```

```
44     /home/stud1
```

```
44     总计
```

为了使输出结果更加友好（按 KB、MB 或 GB），可以使用“-h”选项，如下所示。

```
#du -h ~kitty
```

```
16k    ./pic
```

```
76k    .
```

命令 du 配合命令 sort，可以按占用空间的大小顺序对磁盘空间的使用情况进行统计。例如，使用下面的命令对当前目录进行统计，并将统计结果保存到/home/disk_used 文件中。

```
#du -a | sort -n > /home/disk_used &
```

其中“du -a”用于统计当前目录下所有文件和目录占用的磁盘空间，“sort -n”命令对“du -a”的统计结果按从小到大的顺序进行排序，然后存储到/home/disk_used 文件中。由于这种对磁盘空间使用情况的统计一般会占用相当长的时间，可使用“&”将该命令移到后台运行。命令执行完毕，生成的/home/disk_used 文件内容如下。

```
# cat /home/disk_used
```

```
4      ./pic/cock
```

```
4      ./pic/hen
```

```
4      ./test.3
```

```
8      ./homework.1
```

```
8      ./test.2
```

```
8      ./test.4
```

```
16     ./pic
```

```
24     ./test.1
```

```
76     .
```

5.2.3 df 命令：显示文件系统磁盘空间使用情况

df 命令显示指定磁盘文件系统的可用空间。如果没有文件系统被指定，则所有当前挂载的文件系统的可用空间将被显示。默认情况下，磁盘空间将以 1KB 为单位进行显示，除非环



境变量 POSIXLY_CORRECT 被指定。如果指定 POSIXLY_CORRECT，将以 512 字节为单位进行显示。该命令的语法格式如下所示。

```
df[-a,--all][-B,--block-size=SIZE][-h,--human-readable][-H,--si][-i,--inodes][-k][-l,--local][--no-sync]
    [--sync][-P,--portability][-T,--print-type][-t,--type=TYPE][-x,--exclude-type=TYPE][--help][--version]
```

其中各选项的具体说明如表 5-10 所示。

表 5-10 df 命令的选项说明

选 项	说 明
-a, --all	全部文件系统的列表，包括包含 0 个块的文件系统
-B, --block-size=SIZE	指定块大小
-h, --human-readable	以便于用户读取的方式显示信息，例如 1K、 234M 、 2G
-H, --si	等于-h，但是计算时，1K=1000，而不是 1K=1024
-i, --inodes	显示 inode 信息
-k	指定块大小，与--block-size=SIZE 相同
-l, --local	只显示本地文件系统
--no-sync	忽略 sync 命令
--sync	先执行 sync 命令
-P, --portability	显示格式为 POSIX
-T, --print-type	显示文件系统类型
-t, --type=TYPE	只显示指定文件系统的磁盘信息
-x, --exclude-type=TYPE	不显示指定文件系统的磁盘信息
--help	显示帮助信息
--version	显示版本信息

例如，通过命令 df，显示当前磁盘的使用情况，如下所示。

```
# df //显示当前的磁盘使用情况
文件系统      1K-块      已用      可用      已用%      挂载点
/dev/sda2      7637400    6219652    1029780    86%        /
/dev/sda1      101089     9426       86444     10%        /boot
none           127632     0          127632     0%         /dev/shm
```

例如，以 inode 模式显示磁盘的使用情况。显示时，列出的是文件系统的全部、已用、可用的 inode 数量，命令如下所示。

```
# df -i
文件系统      Inode (I)  已用 (I)  可用 (I)  已用%      挂载点
/dev/sda2      971520    309395    662125    32%        /
/dev/sda1      26104     41        26063     1%         /boot
none           31908     1         31907     1%         /dev/shm
```

例如，以用户容易识别的方式显示磁盘信息，命令如下所示。

```
# df -h
文件系统      容量  已用  可用  已用%      挂载点
/dev/sda2      7.3G  6.0G  1006M    86%        /
/dev/sda1      99M   9.3M   85M    10%        /boot
none           125M   0    125M     0%         /dev/shm
```


例如，显示当前磁盘的使用信息，并且通过-t 选项使得只显示 ext3 类型磁盘的信息，命令如下所示。

```
#df -t ext3 //显示磁盘信息
文件系统      1K-块      已用      可用  已用%  挂载点
/dev/sda2      7637400    6219656    1029776  86%    /
/dev/sda1      101089     9426      86444   10%    /boot
```

例如，显示磁盘的信息，并且通过-x 选项，不显示指定磁盘类型的磁盘使用信息，命令如下所示。

```
# df -x ext3
文件系统      1K-块      已用      可用  已用%  挂载点
none          127632      0        127632  0%    /dev/shm
```

5.2.4 fdformat 命令：格式化磁盘

命令 fdformat 可以实现对软盘的格式化，该格式化属于低级格式化。fdformat 命令的语法格式如下所示。

```
fdformat [ -n ] device
```

选项“-n”表示格式化完成后，不进行验证。device 是指定操作的软盘设备。软盘设备可以是下列之一，其中主设备号均为 2，从设备号如括号中所示。

```
/dev/fd0      (minor=0)
/dev/fd0d360  (minor = 4)
/dev/fd0h1200 (minor = 8)
/dev/fd0D360  (minor = 12)
/dev/fd0H360  (minor = 12)
/dev/fd0D720  (minor = 16)
/dev/fd0H720  (minor = 16)
/dev/fd0h360  (minor = 20)
/dev/fd0h720  (minor = 24)
/dev/fd0H1440 (minor = 28)
/dev/fd1d360  (minor = 5)
/dev/fd1h1200 (minor = 9)
/dev/fd1D360  (minor = 13)
/dev/fd1H360  (minor = 13)
/dev/fd1D720  (minor = 17)
/dev/fd1H720  (minor = 17)
/dev/fd1h360  (minor = 21)
... ..
```



除了上面定义的软盘设备文件外，用户还可以根据需要使用 mknod 命令自行创建。

例如，格式化软盘设备/fd0，命令如下所示。

```
#fdformat /fd0
```

5.2.5 hdparm 命令：显示和设定硬盘参数

hdparm 命令提供了一个命令行接口来显示设置硬盘参数。一些选项参数可能只能在最新的内核下工作，因此，最好在最新的内核下编译 haparm 命令。该命令的语法格式如下所示。



hdparm [flags] [device]

其中各选项的具体说明如表 5-11 所示。

表 5-11 hdparm 命令的选项说明

选 项	说 明
-a	预先显示/设置 fs
-A	设备驱动预读标志
-b	显示/设置总线状态 (0 表示关闭, 1 表示开启, 2 表示三态)
-B	设置高级电源管理设备 (1~255)
-c	显示/设置 IDE 32 位输入/输出设备
-C	检查 IDE 电源模式状态
-d	显示/设置 DMA 标志
-D	启动/关闭驱动错误管理
-E	设置 CDROM 驱动器速度
-f	将缓存数据写回并退出
-g	显示驱动器的几何参数
-h	显示简洁帮助信息
-i	显示驱动标识
-I	从驱动器直接读取当前信息
-Istdin	与 -I 类似, 但将 /proc/ide/*/hd?/ 识别为输入设备
-L	设置驱动门锁 (仅适用于可移动设备)
-M	显示/设置声音管理
-m	显示/设置扇区数
-n	显示/设置 忽略写错误标志
-p	在 IDE 接口芯片上设置 PIO 模式
-P	设置驱动器预存取数
-q	静默模式
-Q	显示/设置 DMA 队列长度
-r	显示/设置只读标志
-R	注册 IDE 接口
-S	设置等待时间
-t	执行设备读取定时
-T	执行缓存读取定时
-u	显示/设置 unmaskirq 标记
-U	取消注册的 IDE 接口
-V	显示版本号
-w	执行设备重启

续表

选 项	说 明
-W	设置设备写缓存标志
-x	热交换三态设备
-X	设置 IDE xfer 模式
-y	将 IDE 驱动设置为待机状态
-Y	将 IDE 设备设置为休眠
-Z	关闭西杰硬盘自动节能模式
-z	重读分区表

例如，显示/dev/hdc 的电源管理模式，命令如下所示。

```
# hdparm -C /dev/hdc //显示电源管理模式
/dev/hdc:
IO_support = 0 (default 16-bit)
```

例如，开启/关闭 DMA 模式，命令如下所示。

```
# hdparm -d 1 /dev/hdc //开启 DMA 模式
/dev/hdc:
setting using_dma to 1 (on)
using_dma = 1 (on)
# hdparm -d 0 /dev/hdc //关闭 DMA 模式
/dev/hdc:
setting using_dma to 0 (off)
using_dma = 0 (off)
```

5.2.6 mkswap 命令：设置交换分区（swap area）

mkswap 命令用于在一个文件或设备上建立交换分区。在建立完之后，要使用 swapon 命令启动这个交换区。mkswap 命令的语法格式如下所示。

```
mkswap [-c] [-vN] [-f] [-p PSZ] device [size]
```

其中各选项的具体说明如表 5-12 所示。

表 5-12 mkswap 命令的选项说明

选 项	说 明
device	指定作为交换分区的设备或文件
size	指定了交换区的大小，这个选项是为了后向兼容而设置的。一般没有使用的必要，通常都将整个文件或设备作为交换区
-c	在创建交换分区前检查坏块
-f	强制执行命令，不管命令是否合理
-p PSZ	指定使用的页大小
-v0	建立一个旧式的交换区
-v1	建立一个新式的交换区

例如，将/dev/hda4 指定为一个新式的交换分区，在创建前检测该设备是否有坏块，命令如下所示。

```
# mkswap -c -v1 /dev/hda4
```



5.2.7 mkisofs 命令：建立 ISO9660/JOLIET/HFS 文件系统

mkisofs 命令用于建立 ISO9660/JOLIET/HFS 文件系统，通常生成 iso9660 镜像文件。该命令的语法格式如下所示。

```
mkisofs [ options ] [ -o filename ] pathspec [pathspec ...]
```

其中各选项的具体说明如表 5-13 所示。

表 5-13 mkisofs 命令的选项说明

选 项	说 明
-biblio FILE	指定 bibliographic 的文件名
-b eltorito_boot_image	指定启动镜像的路径和文件名
-x path	指定不被写入 CDROM 的路径
-v	冗余模式，制作时输出过程信息
-J	除 iso9660 文件名外，产生 Joliet 目录记录
-R	使用 Rock Ridge 协议产生 SUSP 和 RR 记录，以进一步描述 iso9660 文件系统
-o filename	指定 iso9660 文件系统镜像的文件名
-quiet	静默模式
-V volid	指定卷标
-sysid ID	指定系统 ID

例如，将文件 bubble 制作为 iso9660 镜像文件，命令如下所示。

```
# mkisofs -o myiso bubble
Total translation table size: 0
Total rockridge attributes bytes: 0
Total directory bytes: 0
Path table size(bytes): 10
Max brk space used 3024
48 extents written (0 Mb)
```

5.2.8 mkinitrd 命令：创建 ramdisk 的镜像文件

mkinitrd 命令为预加载模块创建初始 ramdisk 镜像文件。该命令的语法格式如下所示。

```
mkinitrd[--version] [-v] [-f] [--preload=module] [--omit-scsi-modules] [--omit-raid-modules]
[--omit-lvm-modules]
        [--with=module] [--image-version] [--fstab=fstab] [--nocompress] [--builtin=module]
[--nopivot]
image kernel-version
```

其中各选项的具体说明如表 5-14 所示。

表 5-14 mkinitrd 命令的选项

选 项	说 明
image	指定镜像文件名称和路径
kernel-version	指定所依据的内核版本
--builtin=modul	认为指定模块已经装入内核，忽略错误
-f	允许覆盖已存在的镜像文件

续表

选 项	说 明
--image-version	在创建镜像文件创建时，内核版本号将附加到建立的镜像文件的目录前
--fstable=<文件系统列表>	使用列表自动探测根设备所建立的文件系统类型
--nocompress	不压缩生成的镜像文件
--nopivot	不使用 pivot_root 系统调用作为映像的一部分
--omit-lvm-modules	不载入任何 lvm 模块
--omit-scsi-modules	不载入任何 scsi 模块
--omit-raid-modules	不载入任何 raid 模块，忽略/etc/fstab 和/etc/raidtab 文件中的指定
--with=module	将指定的模块载入镜像中
-v	冗余模式，在创建镜像过程中显示详细信息
--version	显示版本信息

例如，建立一个镜像文件/boot/initrd-2.2.5-15.img，内核版本号为 2.2.5-15，命令如下所示。

```
# mkinitrd /boot/initrd-2.2.5-15.img 2.2.5-15
```

5.2.9 mt 命令：磁带驱动操作

mt 命令用来进行磁带驱动的操作，该命令的语法格式如下所示。

```
mt [-v,--version] [-h] [-fdevice] [fsf] [Fsfm] [bsf] [bsfm] [asf] [fsr] [bsr] [fss] [bss] [eod,s  
eod] [ rewind]  
[offline,rewoffl,eject] [retention] [weof,eof] [west] [erase] [status] [seek] [tell]  
[setpartition] [ -f device]
```

其中各选项的具体说明如表 5-15 所示。

表 5-15 mt 命令的选项说明

选 项	说 明
fsf	将磁带定位在指定文件的下一个文件所在的第一个块上
Fsfm	将磁带定位在指定文件的前一个文件所在的最后一个块上
bsf	同上，使磁带定位在指定文件的前一个文件所在的最后一个块上
bsfm	使磁带定位在指定文件的下一个文件所在的第一个块上
asf	磁带定位在指定文件的开始
fsr	前进指定数目的记录
bsr	后退指定数目的记录
fss	SCSI 磁带前进指定数目的设置标记（setmark）
bss	SCSI 磁带后退指定数目的设置标记
eod,seod	到可用数据的末端
rewind	回卷磁带
offline,rewoffl,eject	回卷磁带，如果可能并卸载磁带



续表

选 项	说 明
retention	回卷磁带，然后卷磁带到末端，然后再次回卷
weof,eof	在当前位置写入指定的 EOF 字符
west	在当前位置写入指定的设置标记 (setmark)
erase	擦除磁带
status	打印磁带的当前状态信息
seek	在磁带上搜寻指定的块
tell	判断当前的块
setpartition	切换到指定的分区
-h	列出帮助信息
-v,--version	列出版本信息
-f device	指定要操作的设备



mt 命令有个类似的命令 rmt。rmt 命令是远端磁带传输协议模块，可以通过进程间通信，远程控制磁带机进行备份、恢复备份操作。

5.2.10 mzip 命令：弹出 zip/jaz 设备中的磁盘

mzip 命令用于改变保护模式或者弹出在 Zip/Jaz 驱动器上的磁盘。该命令的语法格式如下所示。

```
mzip [-e] [-p] [-q] [-r] [-w] [-x]
```

其中各选项的具本说明如表 5-16 所示。

表 5-16 mzip 命令的选项说明

选 项	说 明
-e	弹出磁盘
-f	与-e 共同使用，强制弹出磁盘
-r	磁盘写保护
-w	去除写保护
-p	密码写保护
-x	密码保护
-u	弹出磁盘以前，暂时解除磁盘的保护状态。当弹出磁盘时，磁盘可写并转换回之前状态
-q	查询状态

5.2.11 sync 命令：将缓冲区内的数据写入磁盘

sync 命令将缓冲区中的数据写入磁盘。该命令的语法格式如下所示。

sync [--help][--version]

其中各选项的具体说明如表 5-17 所示。

表 5-17 sync 命令的选项说明

选 项	说 明
--help	显示 sync 命令的帮助信息
--version	显示 sync 命令版本信息

例如，在卸载移动硬盘之前，将内存数据写入磁盘，命令如下所示。

```
# sync //将内存数据写入磁盘
```

5.2.12 rsync 命令：远程同步

rsync 是 remote synchronize 的缩写，通常用于在系统之间创建文件和目录的同步副本。使用 rsync 可以在本地磁盘和远程系统之间备份文件和目录。由于其只将修改过的文件和目录从源位置复制到目的位置，因而可以有效节省带宽。该命令的语法格式如下所示。

```
rsync [-a, --archive] [-v, --verbose] [-q, --quiet] [-c, --checksum] [-r, --recursive]
[-h, --help] [src] [dest]
[-R, --relative] [--timeout=TIME] [-z, --compress] [-I, --ignore-times]
```

其中各选项的具体说明如表 5-18 所示。

表 5-18 rsync 命令的选项说明

选 项	说 明
-a, --archive	归档模式
-v, --verbose	verbose 模型
-q, --quiet	静默模式，减少冗余
-c, --checksum	执行校验
-r, --recursive	递归包含目录及子目录
-R, --relative	使用相对路径
--timeout=TIME	设定 IO 超时时间
-z, --compress	压缩文件数据
-I, --ignore-times	忽略时间约束
-h, --help	显示帮助信息
src	指定源
dest	指定目标



rsync 的选项很多，可以使用 man 命令参考 rsync 联机帮助。

例如，将一台名为 studnet.example.com 主机上的/home/share 目录中的所有 doc 文件备份到主机 teacher.example.com 的/usr/local/share 目录下，应在 student 主机上执行下面的命令。



```
#rsync -av -e ssh /home/share/*.doc root@teacher.example.com:/usr/local/share
root@teacher.example.com's password:
building file list ...done
file1.doc
file2.doc
file3.doc
write 23432423 bytes read 242 bytes 2534444.56 bytes/sec
total size is 23432165 speedup is 1.00
```

可以看到,rsync 命令通过 SSH 与 teacher.example.com 主机建立了连接,并将/home/share 目录下的所有文件备份到了 teacher 主机的/usr/local/share 目录下。

如果 studnet.example.com 主机中的/home/share 目录自上次运行 rsync 命令以来没有发生变化,则再次运行 rsync 命令,显示如下。

```
#rsync -av -e ssh /home/share/*.doc root@teacher.example.com:/usr/local/share
root@teacher.example.com's password:
building file list ...done
file1.doc
file2.doc
file3.doc
write 274 bytes read 23 bytes 86.34 bytes/sec
total size is 23432165 speedup is 55778.89
```

可以看到,由于 studnet.example.com 主机上的/home/share 与 teacher.example.com 主机的/usr/local/share 已经同步,再次使用 rsync 命令,并没有复制任何文件。

如果向 studnet.example.com 主机上的/home/share 目录中添加文件 file4.doc,再次运行 rsync 命令,显示如下。

```
#cp file4.doc /home/share
#rsync -av -e ssh /home/share/*.doc root@teacher.example.com:/usr/local/share
root@teacher.example.com's password:
building file list ...done
file4.doc
write 2374 bytes read 29 bytes 23486.34 bytes/sec
total size is 23434994 speedup is 78.89
```

可以看到,rsync 仅对新添加的文件 file4.doc 进行了复制。

5.3 磁盘配额管理

Linux 是一个多用户、多任务操作系统,支持多客户端、多用户的使用。经验表明,在多用户系统上,加入的磁盘空间越多,用户使用的也越多,磁盘空间的浪费也越多,同时系统的可靠性也会大幅降低。保证系统有足够磁盘空间的最好方法就是有效地限制用户的使用量。

在 Linux 中,可以使用磁盘配额技术。磁盘配额可以为每一个用户或用户组,甚至一个文件系统设定磁盘使用额度。通过磁盘配额,当用户使用的磁盘空间过多或分区占用过大时,系统管理员都会收到警告,从而及时采取措施。

在 Red Hat Linux 9 中,系统自带了磁盘配额软件,默认情况下系统会自动安装,因此可以直接使用磁盘配额系统。

5.3.1 磁盘配额的系统配置

磁盘配额的系统配置主要包括添加参数、重载分区和生成配额文件 3 个步骤。

(1) 添加 “usrquota” 和 “grpquota” 参数。

首先，需要确定要使用磁盘配额的分区。例如，/home 分区由于其目录下包含了所有用户的主目录，且用户利用 FTP 登录主机时也通常使用该目录作为起始目录，因此最好利用磁盘配额对其空间的使用进行限制。

在确定需要使用磁盘配额的分区后，需要在/etc/fstab 文件中对该分区进行标注。本例中打开/etc/fstab 文件，在/home 分区记录项添加 “usrquota” 和 “grpquota” 参数，如下所示。

```
# cat /etc/fstab
LABEL=/                /                ext3    defaults        1 1
LABEL=/boot            /boot            ext3    defaults        1 2
LABEL=/home            /home            ext3    defaults,usrquota,grpquota 1 2
none                   /dev/pts         devpts  gid=5,mode=620   0 0
none                   /proc            proc    defaults        0 0
none                   /dev/shm         tmpfs   defaults        0 0
/dev/sda3              swap             swap    defaults        0 0
/dev/cdrom              /mnt/cdrom       udf,iso9660 noauto,owner,kudzu,ro 0 0
/dev/fd0                /mnt/floppy      auto    noauto,owner,kudzu 0 0
```

其中，参数 “usrquota” 表示要在该文件系统中限制用户的使用空间，而参数 “grpquota” 表示要在该文件系统中限制用户组的使用空间。

(2) 重新挂载配额分区。

为使修改生效需要重新启动计算机，也可以使用 mount 命令对该分区进行重载，执行如下所示命令。

```
#mount -O remount /home
```

重载过程是先卸载分区，然后再重新挂载。由于无法卸载根分区，所以不能对根分区使用该命令。

(3) 生成 aquota.group 和 aquota.user 磁盘配额文件。

为了使系统能够按着磁盘配额进行工作，必须建立磁盘配额文件 aquota.group 和 aquota.user。

5.3.2 edquota 命令：对用户或用户组设置磁盘配额

在创建了用户和用户组磁盘配额文件 aquota.group 和 aquota.user 后，由于 aquota.group 和 aquota.user 文件结构复杂，无法直接在编辑器中打开，必须通过 edquota 命令才能正常编辑。edquota 命令的语法格式如下所示。

```
edquota [-p protoname ][ -F format-name] [ -u ][ -g ] [ -r ] [ -t ]
```

其中各选项的具本说明如表 5-19 所示。

表 5-19 edquota 命令参数及说明

选 项	说 明
-g	设置用户组的磁盘配额
-u	设置用户的磁盘配额，默认值
-p protoname	对磁盘配额设置进行复制
-t	对文件系统设置软时间限制
-F format-name	按指定格式编辑配额



edquota 命令进入的是 vi 编辑状态 (vi 的使用方法参见第 10 章), 可以编辑用户和用户组的磁盘空间限制。图 5-2 列出了 edquota 命令的编辑格式。

Filesystem	blocks	soft	hard	inodes	soft	hard
/dev/sda1	20	0	0	9	0	0
设置 quota 分区	已有磁盘空间	磁盘空间配额		已用 i 节点数	i 节点配额	

图 5-2 edquota 命令编辑格式

其中, “Filesystem” 字段指出目前使用磁盘配额的文件系统为 /dev/sda1, “block” 字段表示用户已经使用的空间为 20 个 blocks, 其中 1 个 block 是 1024 字节。“block” 字段后的 “soft” 和 “hard” 字段, 表示磁盘空间配额的软限额和硬限额, 值为 “0” 表示没有限制。“inodes” 表示已使用的 i 节点数, 即文件和目录数。其后的 “soft” 和 “hard” 字段分别表示 i 节点数的软限额和硬限额, 值为 “0” 表示没有限制。

所谓软限额是指用户达到此限制时, 系统会发出警告信息, 但是用户仍然可以继续使用。不过一旦达到硬限额时, 用户就无法再写入了。

例如, 为用户 student 设置磁盘限额, 其磁盘空间软限额为 10MB, 硬限额是 50MB, i 节点的软限额为 600, 硬限额为 800, 使用 edquota 命令编辑 student 用户配额如下。

```
#edquota -u student
Disk quotas for user student (uid 605):
Filesystem    blocks    soft    hard    inodes    soft    hard
/dev/sda1      20      10000   50000     8       600    800
```

其中, blocks 为 20 表示用户 student 已经使用了 20 块 (共 $20 \times 1KB = 20KB$) 的磁盘空间。当用户使用的磁盘空间达到软限额 10MB 时, 系统会提示用户已达到软限额, 但用户仍可以继续写入。当达到硬限额 50MB 时, 系统将禁止 student 向磁盘写入任何数据。inode 为 8 表示用户 student 已经建立了 8 个文件, 当用户建立的文件数达到 600 时, 系统会给出警告, 提示用户已达到软限额。当文件数达到 800 将不再允许建立文件或目录。

例如, 设置用户 teacher 使用的磁盘软限额为 500MB, 硬限额为 800MB, 规定建立文件/目录的软限额为 128, 硬限额为 512, 使用 edquota 命令如下。

```
#edquota -u teacher
Disk quotas for user teacher (uid 601):
Filesystem    blocks    soft    hard    inodes    soft    hard
/dev/sda1      0      500000  800000     0       128    512
```

如果有很多用户需要设置相同的配额, 可以使用 edquota 命令的 “-p” 选项。例如, 为用户 teacher1、teacher2、teacher3 建立与上例中用户 teacher 一样的磁盘配额方案, 命令如下所示。

```
#edquota -p teacher -u teacher1 teacher2 teacher3
```

除了可以限制用户使用磁盘空间大小和 i 节点数目外, 还可以设置 “grace period”, 即宽限期。所谓宽限期是指当用户使用的磁盘空间或 i 节点数已达到软限额, 但还未达到硬限额情况下, 允许用户继续使用的天数。一旦达到硬限额或超出设置的宽限期, 用户将无法继续使用, 直到将磁盘的使用量或 i 节点数降到软限额以下。使用 “edquota -t” 命令可以设置宽限期, 如下所示。


```
#edquota -t
Grace period before enforcing soft limits for users:
Time units may be: days, hours, minutes, or seconds
  Filesystem      Block grace period   Inode grace period
  /dev/sdal        7days                7days
```

系统默认的宽限期为 7 天，可以以日、小时、分钟和秒为单位进行修改。在此设置的宽限期对所有用户和用户组均有效。

5.3.3 quota 命令：显示磁盘已使用的空间与配额限制

在成功建立 aquota.group 和 aquota.user 文件后，可以使用 quota 命令显示磁盘已使用的空间与限制。该命令的语法格式如下所示。

```
quota [-g][ -q][ -u][ -v][ --version] [ -F format-name ]
```

其中各选项的具本说明如表 5-20 所示。

表 5-20 quota 命令的选项说明

选 项	说 明
-g	显示用户组的磁盘配额信息
-q	以简略格式进行显示，仅显示使用配额的文件系统
-u	列出磁盘空间配额
-v	显示文件系统的磁盘配额
-F format-name	按指定格式显示配额信息
--version	显示版本信息



直接运行命令“quota 用户名”，可以显示分给指定用户的磁盘配额。

例如，显示用户 student 的磁盘配额，命令如下所示。

```
# quota student
Disk quotas for user tanli (uid 605): none      //没有对 student 用户进行磁盘配额设置
```

例如，查看用户 kitty 使用的磁盘空间情况，命令如下所示。

```
#quota -u kitty
Disk quotas for user kitty (uid 608):
  Filesystem      blocks      soft      hard      inodes      soft      hard
  /dev/sdal        10          0         0         3           0         0
```

可以看到，用户 kitty 使用了 10KB 磁盘空间，建立了 3 个文件或目录。

如果系统没有设置磁盘配额，则无法使用 quota 命令来查看磁盘空间的使用情况，系统会提示如下信息。

```
#quota -u kitty
Disk quotas for user kitty (uid 608):none
```

5.3.4 quotacheck 命令：检查磁盘的使用空间与限制

使用 quotacheck 命令可以完成配额文件的自动创建。quotacheck 命令还具有检查文件系



统，建立硬盘使用率列表，以及检查每个文件系统的空间配额等功能。quotacheck 命令的语法格式如下所示。

```
quotacheck [ -g ][ -u ][ -c ] [ -v ][ -d ][ -a ][ -f ] -M [ -m ][ -i ][ -R ] [ -F quota-format ]
```

其中各选项的具体说明如表 5-21 所示。

表 5-21 quotacheck 主要参数及说明

选 项	说 明
-a	扫描在/etc/mstab 文件中所有挂载的非 NFS 文件系统
-d	启用调试模式，会产生大量的调试信息，有详细的输出结果，但扫描速度比较慢
-u	计算每个用户占用的目录和文件数目，并创建 aquota.user 文件
-g	计算每个用户组占用的目录和文件数目，并创建 aquota.group 文件
-c	忽略现存的配额文件，重新扫描并建立新配额文件
-v	显示命令执行过程
-b	强制 quotacheck 在建立新配额文件前对旧配额文件进行备份
-F quota-format	按指定的格式检测磁盘配额
-f	强制检测文件系统。由于创建的 quota 文件可能存在不同步问题，通常不建议使用该选项
-M	强制以读/写模式强制执行文件系统检测
-m	不尝试以只读模式重新加载文件系统
-i	交互模式。非交互模式下，只有发现错误 quotacheck 才进行显示
-R	与选项-a 同时使用时，除了根文件系统外的所有文件系统都将被检测

文件 /etc/mstab 与/etc/fstab 内容类似，/etc/fstab 表示系统开机时默认加载的分区，而 /etc/mstab 表示目前系统加载中的分区。例如，扫描/etc/mstab 文件，建立 aquota.group 和 aquota.user 磁盘配额文件，命令如下所示。

```
quotacheck -avgu
```

执行 quotacheck 命令后，可以看到在/home 文件系统中新建了 aquota.group 和 aquota.user 文件，如下所示。

```
#ls -al
drwxr-xr-x 13 root    root    4096 08-31 07:59 .
drwxr-xr-x 24 root    root    4096 08-31 13:42 ..
-rw----- 1 root    root    6144 08-31 13:38 aquota.group
-rw----- 1 root    root    6144 08-31 13:38 aquota.user
drwx----- 2 stud1   users   4096 2007-07-14 stud1
drwx----- 2 stud2   users   4096 2007-07-14 stud2
... ..
```

对于有些版本的 Linux，在对/etc/fstab 文件设置配额参数后，需要修改/etc/rc.d/rc.local 脚本文件，才能使系统每次启动时都能够自动创建配额文件并自动启用磁盘配额程序。例如，修改/etc/rc.d/rc.local 脚本文件如下。

```
... ..
#Check quota and then turn quota on
if [ -x /sbin/quotacheck ]; then
echo " Checking quotas. This may take some time..."
/sbin/quotacheck -avug
```



```
echo "Done"
fi
if [ -x /sbin/quotachk ] ; then
echo "Enabling disk quota.."
/sbin/quotachk -avug
echo "Done"
fi
... ..
```

其中，命令“/sbin/quotachk -avug”用于扫描/etc/fstab 中设置磁盘配额的分区，以及在该分区下建立配额文件 aquota.group 和 aquota.user。



对于 Red Hat Linux 9，不需要这个步骤，因为在 Red Hat Linux 9 的/etc/rc.d/rc.sysinit 文件中已经设置启用 quota。

5.3.5 quotaoff 命令：关闭磁盘空间限制

在设置好磁盘配额之后，可以使用 quotaoff 命令关闭磁盘空间配额。该命令的语法格式如下所示。

quotaoff [-g][-u][-v][p] [-a] [-F quotaformat] [-x state][--help][--version]

其中各选项的具本说明如表 5-22 所示。

表 5-22 quotaoff 命令的选项说明

选 项	说 明
-a	关闭所有在/etc/fstab 列出的磁盘配额
-g	关闭指定用户组磁盘配额限制
-p	列出状态
-u	用户配额限制
-v	运行时显示详细的处理信息
-F format-name	按指定格式显示磁盘配额
-x delete	释放保存配额信息的磁盘空间
--help	帮助信息
--version	版本信息

例如，关闭所有配额限制，命令如下所示。

```
#quotaoff -a //关闭配额限制
```

例如，使用命令 quotaoff 关闭/home 的磁盘配额，命令如下所示。

```
# quotaoff /home
/dev/sda1 [/home]: group quotas turned off
/dev/sda1 [/home]: user quotas turned off
```

也可以不指定操作的分区，使用 “-aguv” 选项设定自动搜索，命令如下所示。

```
# quotaoff -aguv
/dev/sda1 [/home]: group quotas turned off
/dev/sda1 [/home]: user quotas turned off
/dev/sda1 [/www]: group quotas turned off
/dev/sda1 [/www]: user quotas turned off
```



5.3.6 quotaon 命令：启动磁盘空间配额限制

在设置好磁盘配额之后，可以使用 quotaon 命令启动磁盘空间配额的限制。该命令的语法格式如下所示。

```
quotaon [-v][-u][-g][-p ] [ -F format-name ] [--help][--version]
```

其中各选项的具本说明如表 5-23 所示。

表 5-23 quotaon 命令的选项说明

选 项	说 明
-a	启动在/etc/fstab 列出的所有磁盘配额限制
-g	组磁盘配额限制
-p	列出状态
-u	用户配额限制
-v	运行时显示详细的处理信息
-F format-name	按指定的格式进行显示
--help	显示帮助信息
--version	显示版本信息

例如，启动所有的磁盘配额限制，命令如下所示。

```
#quotaon -a //开启磁盘限制
```

例如，启动/home 磁盘空间配额，命令行如下所示。

```
# quotaon /home
/dev/sda1 [/home]: group quotas turned on
/dev/sda1 [/home]: user quotas turned on
```

也可以不指定操作的分区，使用 “-aguv” 选项设定自动搜索，如下所示。

```
# quotaon -aguv
/dev/sda1 [/home]: group quotas turned on
/dev/sda1 [/home]: user quotas turned on
/dev/sda1 [/www]: group quotas turned on
/dev/sda1 [/www]: user quotas turned on
```

5.3.7 quotastats 命令：显示磁盘空间的限制

quotastats 命令用来显示系统的磁盘空间限制的当前状态。

例如，显示磁盘配额限制状态，命令如下所示。

```
# quotastats //显示磁盘配额限制状态
Kernel quota version: 6.5.1
Number of dquot lookups: 0
Number of dquot drops: 0
Number of dquot reads: 0
Number of dquot writes: 0
Number of quotafile syncs: 19
Number of dquot cache hits: 0
Number of allocated dquotes: 0
Number of free dquotes: 0
Number of in use dquot entries (user/group): 0
```


5.3.8 repquota 命令：对文件系统的磁盘配额情况进行统计

执行 repquota 命令，可以显示磁盘空间的限制，以及磁盘的使用情况。该命令的语法格式如下所示。

```
repquota [-v][-u][-g][-s] [-t][-n][-a] [-F quotaformat]
```

其中各选项的具体说明如表 5-24 所示。

表 5-24 repquota 命令的选项说明

选 项	说 明
-a	显示在/etc/mtab 中列出的具有可读/写配额的文件系统的情况
-g	显示用户组的配额使用情况
-n	以数字的形式显示 UID/GID，这可以加快访问速度
-s	以更适合的方式显示已用空间、i 节点数和配额限制
-t	当用户名/用户组名的长度大于 9 时，进行截取
-u	显示用户的磁盘配额使用情况
-v	显示所有文件系统的磁盘配额使用情况，即使该配额没被使用
-F format-name	以指定的格式进行显示

例如，显示/etc/mtab 中定义的具有可读写配额的文件系统的磁盘使用情况，命令如下所示。

```
# repquota -a //显示文件系统的磁盘使用情况
```

5.4 逻辑卷管理（LVM）

LVM 为磁盘空间的调整带来了极大的方便，以往在安装 Red Hat Linux 之前，一项重要的工作就是对磁盘空间的划分进行决策，因为一旦划分好就很难重新进行调整。LVM 允许在多个不同的文件系统之间重新分配大块的磁盘空间。LVM 可以包含多个硬盘分区，如果一个卷耗尽了空间，可以简单地附加上一个新物理卷（如一个硬盘分区）来扩展它的容量；而不必重新调整分区大小或创建一个新分区。与 LVM 相关的定义如表 5-25 所示。

表 5-25 LVM 相关定义

定 义	说 明
物理卷（PV）	物理卷（Physical volume，PV）相当于硬盘驱动器的一个标准主分区或逻辑分区
物理盘区（PE）	物理盘区（Physical extent，PE）就是一块磁盘空间。物理卷被划分成多个同等大小的 PE
逻辑卷（LV）	逻辑卷（Logical volume，LV）是 LE 的集合
逻辑盘区（LE）	在 LVM 系统中，一个逻辑盘区（Logical extent，PE）和一个 PE 相对应，大小相同
卷组（VG）	卷组（Volume group，VG）是 LV 的集合



5.4.1 pvdisplay 命令：查看组成 LVM 卷的物理卷

可以使用 `pvdisplay` 命令查看组成 LVM 卷的物理卷 (PV)，如下所示。

```
# pvdisplay
--- Physical volume ---
PV Name                /dev/sda2
VG Name                VolGroup00
PV Size                7.90 GB / not usable 23.41 MB    //容量 7.9GB, 剩余 23.41MB
Allocatable            yes (but full)
PE Size (KByte)        32768
Total PE               252
Free PE                0
Allocated PE           252
PV UUID                4BrhEz-mAgc-90Qm-2LBj-i8Rg-H3pI-FE3k02
```

5.4.2 pvcreate 命令：创建物理卷

可以使用 `pvcreate` 命令创建物理卷。例如，在二级 SCSI 控制器的从属驱动器上创建一个新的物理卷，运行下面的命令。

```
#pvcreate /dev/sdd
```

也可以把 PV 安装到分区上。如果已经创建了分区（例如通过 `fdisk` 创建了 Linux 分区），只需将分区的系统 ID 改为“8e”，如下所示。

```
#fdisk /sddl
Comand (m for more):t
Partition number:4
Hex code (type L to list codes):8e
```

然后通过 `pvcreate` 命令创建，如下所示。

```
#pvcreate /dev/sddl
```

5.4.3 vgcreate 命令：创建卷组

可以通过 `vgcreate` 命令创建一个卷组。卷组是由一个或多个硬盘驱动器上的多个物理卷 (PV) 所构成的一个集合。可以把现有的 PV 创建到一个 VG 上，当添加更多的 PV 时，可以直接添加到 VG 上。例如，创建一个名为 `myvolume` 的 VG，命令如下所示。

```
#vgcreate myvolume /dev/sddl /dev/sdc2
```

5.4.4 vgextend 命令：在卷组里添加一个新的物理卷

如果有多余的 PE，可以使用 `vgextend` 命令在卷组里添加一个新的物理卷。例如，使用下面的命令对 `/dev/myvolume/mylogical` 的容量进行添加。

```
#lvextend -L 2000M /dev/myvolume/mylogical
```

5.4.5 lvcreate 命令：创建一个逻辑卷

使用 `lvcreate` 命令可以创建一个逻辑卷。例如，一个 PE 的大小为 32MB，使用 `lvcreate` 命令来创建一个所需的 LV。需要创建的 LV 名为 `mylocigcal`，容量为 $1.6\text{GB} = 32\text{MB} \times 50$ ，执行如下所示命令。

```
#lvcreate -l 50 myvolume -n mylogical
```


这样就创建了一个/dev/myvolume/mylogical 的新设备。可以像对待其他硬盘驱动器一样对其进行格式化和挂载。例如,把 mylogical 格式化为 ext3 文件系统,并安装到/mnt/mynewdisk 目录上,命令如下所示。

```
#mkfs -j /dev/myvolume/mylogical
#mount -t ext3 /dev/myvolume/mylogical /mnt/mynewdisk
```

5.5 磁盘管理命令的综合应用

由于磁盘应用越来越广泛,面向磁盘的各种技术不断革新,作为系统管理员,需要诊断和排除的与磁盘相关的各种问题也越来越复杂。系统管理员不仅需要掌握大量的工具,还需要熟悉各种解决问题的方法。

5.5.1 添加新磁盘

由于硬盘空间耗尽而需要添加新硬盘是系统管理员经常遇到的问题。在 Linux 中添加一块新硬盘通常涉及以下几个步骤。

(1) 硬盘的安装。

首先关闭计算机,然后根据硬盘的类型(SATA、EIDE 或 SCSI),按照硬盘供应商的推荐值,对硬盘及 BIOS 进行设置,将硬盘安装到系统中。

重新启动计算机,并进一步确认新硬盘是否已被 Linux 系统识别。可以通过 dmesg 命令进行核实。例如,假设新硬盘是安装到计算机第 2 个 IDE 接口的第 1 块硬盘,则新硬盘的设备名应为“hdc”,以 root 身份在 Shell 下运行下面命令。

```
#dmesg | grep hdc
```

如果输出结果中有 hdc 相关内容的描述,则说明新硬盘已被 Linux 系统识别,否则还需做进一步检查。

(2) 在新硬盘上创建分区。

使用 fdisk 或 parted 命令可以在一块硬盘上创建一个或多个 Linux 分区,并按需要为其分配不同的文件系统类型和不同的挂载点。最简单的创建分区的方法是把整个硬盘作为一个单独的 Linux 分区。例如在上例中使用 fdisk 命令,对 hdc 创建 Linux 分区,如下所示。

```
#fdisk /dev/hdc
```

按 5.1 节所讲的方法,使用 fdisk 命令将整个硬盘划分为一个分区/dev/hdc1。

(3) 创建文件系统。

在 Linux 中创建文件系统的命令是 mkfs。例如,将/dev/hdc1 格式化为 ext3 文件系统,执行命令如下所示。

```
#mkfs -t ext3 /dev/hdc1
```

(4) 挂载文件系统。

文件系统创建后,需要建立分区挂载点。例如,将/dev/hdc1 挂载到/mnt/newdisk 目录下,需要在/mnt 目录下创建空目录 newdisk,然后使用 mount 命令进行挂载,命令如下所示。

```
#mkdir /mnt/newdisk
#mount /dev/hdc1 /mnt/newdisk
```



如果希望每次系统启动时自动挂载该硬盘上的文件系统，需要编辑/etc/fstab 文件，添加类似下面的一行命令。

```
/dev/hdc1 /mnt/newdisk ext3 default 1 1
```

5.5.2 无法使用全部硬盘空间

如果用户创建了一个 swap 分区和一个根分区 (/)，并且在分配完 swap 空间后，将所有剩余硬盘空间分配给了根分区，但最终却发现根分区并没有获得剩余的全部空间。此问题可能是由于硬盘容量大于 1024 个柱面引起。对于大于 1024 柱面的硬盘，必须创建一个独立的 /boot 分区，才能使根分区使用硬盘上的所有剩余空间。

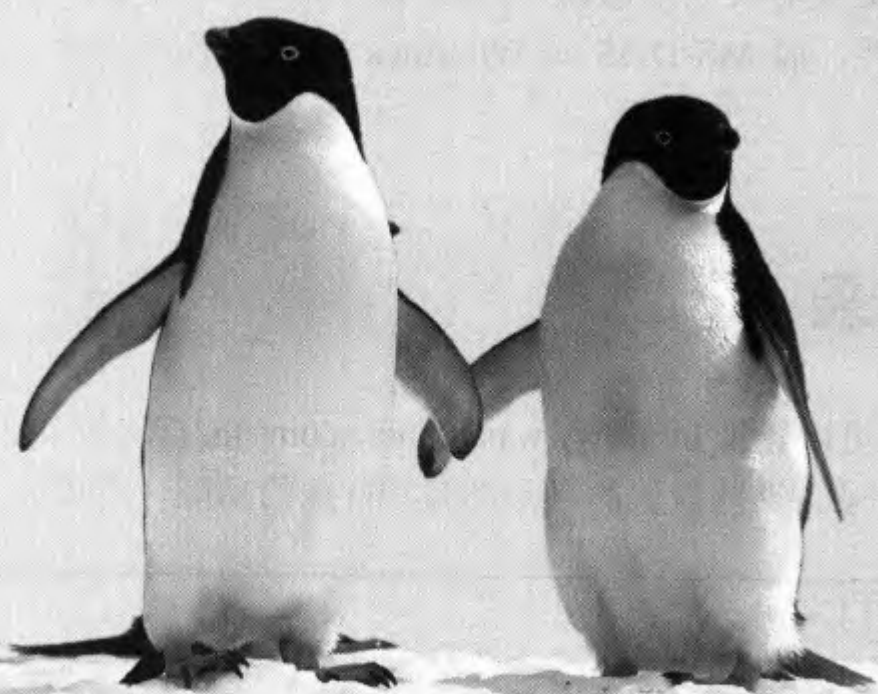


LINUX

第6章

文件和目录管理

文件和目录管理是使用操作系统过程中经常涉及的基本工作。本章将系统地介绍 Linux 文件系统的组织结构、文件系统的权限管理，并对 Red Hat Linux 9 下常用的文件和目录管理命令进行详细的介绍。





6.1 文件系统的概念

文件系统 (File System) 是操作系统用来存储和管理文件的方法。从系统角度来看, 文件系统对文件存储空间进行组织、分配, 并对文件的存储进行保护和检查。从用户角度来看, 文件系统可以帮助用户建立文件, 并对文件的读、写和删除操作提供保护和控制。

6.2 文件系统的组织方式

不同的操作系统对文件的组织方式不同, 其所支持的文件系统数量和种类也不一定相同。Linux 文件系统的组织方式称做文件系统分层标准 (Filesystem Hierarchy Standard, 简称 FHS), 即采用层次式的树状目录结构。此结构的最上层是根目录 “/” (斜杠), 然后在此根目录下创建其他目录和子目录, 如图 6-1 所示。

Linux 与 MS-DOS 及 Windows 一样, 采用“路径”来表示文件或目录在文件系统中所处的层次。路径由以 “/” 为分隔符的多个目录名字符串组成, 分为绝对路径和相对路径。

所谓绝对路径是指由根目录 “/” 为起点来表示系统中某个文件或目录的位置的方法。例如, 如果用绝对路径表示图 6.1 中第 4 层目录中的 linux 目录, 应为 “/usr/src/linux”。

相对路径则是以当前目录为起点, 表示系统中某个文件或目录在文件系统中的位置的方法。若当前工作目录是 “/usr”, 则用相对路径表示图 6.1 中第 4 层目录中的 linux 目录, 应为 “src/linux” 或 “./src/linux”, 其中 “./” 表示当前目录, 通常可以省略。

Linux 文件名称最长允许 256 个字符, 可以包括数字、字符以及 “.”、“-”、“_” 等符号。Linux 文件名不像 MS-DOS 或 Windows 由主文件名和扩展文件名两部分组成, Linux 文件名没有扩展名的概念。Linux 环境下, 文件名大小写敏感 (Case Sensitive), 例如, test.txt 与 Test.txt 会被识别成两个不同的文件, 而 MS-DOS 或 Windows 平台是不区分大小写的。

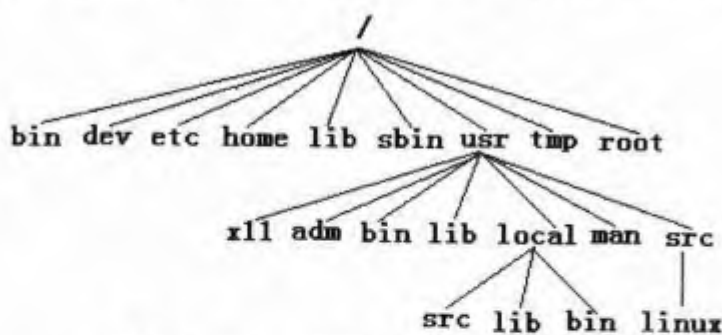


图 6-1 Linux 文件系统目录层次结构

6.3 系统的默认安装目录

依据 FHS 的要求 (关于 FHS 的详细信息可以登录 <http://www.pahtname.com/fhs/> 查询), Linux 系统在安装过程中会创建一些默认的目录。这些默认的目录都有其特殊的功能, 不可

随便将其更名，以免造成系统的错误。表 6-1 列出了这些目录及其功能说明。

表 6-1 Linux 默认目录及其功能说明

目 录 名 称	说 明
/	Linux 文件系统的最上层根目录，其他所有目录均是该目录的子目录
/bin	Binary 的缩写，存放用户的可执行程序，如 cp 和 mv 等，也存放 Shell，如 bash 和 csh。不应把该目录放到一个单独的分区中，否则 Linux Rescue 模式无法使用这些命令
/boot	操作系统启动时所需的文件，包括 vmlinuz 和 initrd.img，这些文件若是被损坏常会导致系统无法正常启动，因此最好不要做任意改动
/dev	设备文件目录，如/dev/sda 表示第一块 SCSI 设备，/dev/hda 表示第一块 IDE 设备
/etc	有关系统设置与管理的文件，包括密码、守护程序以及 X-Window 相关的配置。可以通过编辑器（如 vi、gedit 等）打开并编辑相关的配置文件
/etc/X11	X-Window System 的配置目录
/home	普通用户的主目录或 FTP 站点目录，一般存放在/home 目录下
/lib	存放共享函数库（library）
/mnt	文件系统挂载点（Mount），例如光盘的挂载点可以是/mnt/cdrom，软盘的挂载点可以是/mnt/floppy，Zip 驱动器为/mnt/zip
/opt	该目录通常提供给第三方较大型的应用程序使用，如 Sun Staroffice、Corel WordPerfect，这可避免将文件分散至整个文件系统
/proc	保存目前系统内核与程序执行的相关信息，与利用 ps 命令看到的内容相同。例如 /proc/interrupts 文件保存了当前分配的中断请求端口号，/proc/cpuinfo 保存了当前处理器信息
/root	根用户的主目录
/sbin	System Binary 的缩写。此目录存放的是系统启动时所需执行的系统程序
/tmp	temporary 的缩写，用来存放临时文件的目录
/usr	存放用户使用的系统命令和应用程序
/usr/bin	存放用户可执行的程序，例如 OpenOffice 的可执行程序
/usr/doc	存放各种文档的目录
/usr/include	存放 C 言语用到的头文件
/usr/include/X11	存放 X-Window 程序使用的头文件
/usr/info	存放 GNU 文档的目录
/usr/lib	函数库
/usr/lib/X11	X-Window 的函数库
/usr/local	提供自动安装的应用程序位置
/usr/man	存放在线手册的目录
/usr/sbin	存放用户经常使用的程序
/usr/src	保存程序的源文件的目录，一般系统内核源码存放在/usr/src/linux 目录下
/usr/X11R6/bin	存放 X-Window 的可执行程序
/var	Variable 的缩写，存放包括日志、邮件等经常变化的文件。由于/var 目录的大小经常变动，为了防止失去控制而侵占其他目录所需要的空间，建议将/var 安装到一个独立的分区上



Linux 文件系统的组织与 Windows 操作系统不同。对于在 Linux 下使用的设备，不需要像 Windows 那样创建驱动器盘符，Linux 会将本地磁盘、网络文件系统、CD-ROM、U 盘等所有设备识别为设备文件，并将其嵌入到 Linux 文件系统来进行管理。一个设备文件不占用文件系统的任何空间，仅仅是访问某个设备驱动程序的入口。Linux 系统中有两类特殊文件：面向字符的特殊文件和面向块（block）的特殊文件。前者允许 I/O 操作以字符的形式进行，而后者通过内存缓冲区来使数据的读写操作以数据块的方式实现。当对设备文件进行 I/O 操作时，该操作会被转给相应的设备驱动程序。一个设备文件是用主设备号（指出设备类型）和从设备号（指出是该类型中的第几个设备）来表示，可以通过 `mknod` 命令进行创建。软盘、光盘、硬盘在 Linux 系统中的表示方法见表 6-2。

表 6-2 典型设备文件在 Linux 系统中的表示方法

设备名	Linux 系统中的表示方法
第一个 IDE 接口的 Master 硬盘	/dev/hda
第一个 IDE 接口的 Slave 硬盘	/dev/hdb
第二个 IDE 接口的 Master 硬盘	/dev/hdc
第二个 IDE 接口的 Slave 硬盘	/dev/hdd
第一个 SCSI 接口的 Master 硬盘	/dev/sda
第一个 SCSI 接口的 Slave 硬盘	/dev/sdb
第二个 SCSI 接口的 Master 硬盘	/dev/sdc
第二个 SCSI 接口的 Slave 硬盘	/dev/sdd
光驱	/dev/cdrom
第一个软盘	/dev/fd0

6.4 文件系统的类型

Linux 是一种兼容性很高的操作系统，除了能够挂载各种类型的设备，还可以挂载其他各种文件系统。文件 `/proc/filesystem` 文件中列出了系统当前可用的文件系统类型，其中不仅包括 Unix 支持的各种文件系统类型，也包括 Windows 9x/NT/2000/XP 文件系统。对于普通用户而言，这些功能最切实的意义是允许用户使用软盘、U 盘、CD-ROM 等。

为了查看系统当前可用的文件系统类型，可以使用“`cat`”命令，如下所示。

```
#cat /proc/filesystems
```

Linux 所支持的文件系统类型包括如下几种。

- `adfs`: acron 磁盘文件系统，是在 RiscOS 操作系统中使用的标准文件系统。
- `befs`: BeOS 操作系统使用的文件系统。
- `cifs`: 通用 Internet 文件系统（Commnn Intemet File System，简称为 CIFS），用于访问符合 SNIA CIFS 标准的服务器。CIFS 对 SMB 协议进行改进和标准化（SMB 协议可用于在 Linux 和 Windows 之间共享文件），是一种虚拟文件系统。

- **ext:** ext 文件系统的第一个版本, 现在已经很少使用。
- **ext2:** ext2 是专门为 Linux 系统设计的文件系统, 在 Red Hat Linux 7.2 版本之前是 Linux 的缺省文件系统类型, 具有速度快和 CPU 占用率低等特点。它既可以用于标准的块设备, 也可以应用到移动存储介质上。ext2 不包括日志功能。
- **ext3:** ext3 文件系统是 Linux 中最常用的文件系统, 是 Red Hat Linux 7.2 的新特性, 也被称为第 3 次扩展 (Third Extended) 的文件系统。现在, ext3 已经成为许多 Linux 系统的默认文件系统类型 (包括 Fedora 和 RHEL)。在 ext2 和 ext3 之间可以方便地进行转换, 转换前不需重新格式化文件系统。与 ext2 文件系统相比, ext3 包含了日志功能。日志功能维护了最近更改的源数据 (源数据是指和文件有关的信息, 包括权限、所有者、创建时间、访问时间等) 的记录, 如果源数据由于非法关机等原因遭到破坏, 文件系统将不能正常工作。通过 ext3 的日志系统, 可以对源数据进行适当的恢复。此外, ext3 的日志功能可使硬盘读写头的移动达到最佳化。
- **is09660:** 从 High Sierra (CD-ROM 使用的最初标准) 发展而来的文件系统, 是 CD-ROM 的标准文件系统。
- **kafs:** AFS 客户端文件系统, 用于分布式计算环境, 可与 Linux、Windows 和 Macintosh 客户端共享文件。
- **minix:** minix 文件系统类型, 最初用于 Unix 的 minix 版本, 只支持长度为 30 个字符以下的文件名。
- **msdos:** MS-DOS 文件系统。MS-DOS、Windows 和 OS/2 使用该文件系统, 它不支持长文件名, 主要用于挂载 Microsoft 操作系统生成的软盘。
- **vfat:** Microsoft 扩展 FAT (VFAT) 文件系统, 支持长文件名, 可被 Windows 9x/2000/xp 使用。
- **umsdos:** 扩展的 MS-DOS 文件系统, 不仅支持长文件名, 还保持了对 UID/GID、POSIX 权限和特殊文件 (如管道、设备) 的兼容。
- **proc:** proc 是一个基于内存的伪文件系统, 不占用外存空间, 只是以文件的方式为访问 Linux 内核数据提供接口。由于 proc 文件系统是虚拟的, 因此无需挂载。用户和应用程序可以通过 /proc 得到系统的运行信息, 并可以改变内核的某些参数。许多应用程序和工具依靠 proc 来访问 Linux 内核信息。
- **reiser:** reiserfs 日志文件系统。
- **swap:** 用于交换 (swap) 分区。交换分区是系统虚拟内存的一部分, 用于在当前内存不足时暂时保存数据。数据被交换到交换分区, 再次需要时再被调回内存。
- **nfs:** 网络文件系统 (Network File System, 简称为 NFS) 类型。
- **hpfs:** 该文件系统用于只读挂载 OS/2 HPFS 文件系统。
- **ncpfs:** Novell Netware 文件系统, 可以通过网络挂载。
- **affs:** Amiga 计算机使用的文件系统。
- **ufs:** Sun Microsystems 操作系统 (即 Solaris 和 SunOS)。
- **xfs:** 一种在高性能环境中很有用的日志文件系统, 支持完整的 64 位寻址。
- **fs:** JFS 主要适合于企业系统, 是为大文件系统和高性能环境而设计的。



- xiafs: 与 minux 文件系统相比, 这种文件系统支持长文件名和更大的 i 节点。
- coherent: System V 使用的文件系统类型。
- smb: 支持 SMB 协议的网络文件系统, 可用于实现 Linux 与 Windows 系统的文件共享。

6.5 文件系统的组成

在 Red Hat Linux 9 中, 系统默认安装的是 ext3 文件系统。ext3 文件系统将磁盘分为 4 个部分, 如图 6-2 所示。块 0 称为引导块, 包含系统启动程序的磁盘区块; 块 1 称为超级块, 主要用来记录文件系统的配置方式, 其中包括 i 节点数量、磁盘区块数量、未使用的磁盘区块以及 i 节点表、空闲块表在磁盘中的存放位置。由于超级块保存了极为重要的文件信息, 因此系统保存超级块冗余。系统在使用 fsck 等命令修复处于严重瘫痪状态的文件系统时, 实际上就是对超级块进行恢复操作。从块 2 开始是 i 节点表, i 节点 (i-node 是 Index-Node 的缩写) 表中记录的信息很多, 包括文件大小、用户 UID、用户组 GID、文件存取模式 (包括读、写或执行)、链接数目 (文件每创建一个链接, 链接计数加 1, 每删除一个链接, 文件计数减 1)、文件最后修改时间、磁盘区块地址、间接区块等。i 节点表之后的数据存储块用于存放文件内容。

引导块	超级块	i-node	数据存储块
-----	-----	--------	-------

图 6-2 exte3 文件系统磁盘划分

文件有逻辑结构和物理结构两种不同的组织方式。逻辑结构是面向用户的, 是用户可以看到的表示文件内容的字符流, 例如使用编辑命令 vi 或显示命令 cat 所看到的文件内容。物理结构是文件在磁盘上的存储组织方式, 涉及到具体的存放磁盘区块。用户所看到的文件内容是连续的, 但实际上文件可能并不是以连续的方式存放在磁盘上的。事实上, 大于 1 块 (block) 的文件将被分散地存放在磁盘上。然而当用户存取某文件时, Linux 文件系统会以正确的顺序读取此文件全部块, 并将该文件的逻辑结构提供给用户。设计 i 节点表实际上就是为了帮助实现这种物理结构到逻辑结构的转换。

i 节点是一个 64 字节长的表, 表中包含了文件的相关信息和磁盘地址表。磁盘地址表中有 13 个块号。前 10 个块号是文件前 10 块的存放地址, 文件将根据块号在磁盘地址表中出现的顺序依次读取相应的块。当文件长度大于 10 个块, 将会用到后面的 3 块所给出的间接地址块。

磁盘地址表中的第 11 块给出了扩展块的位置, 这个块号指出的块中含有 256 个块号, 加上前 10 块, 文件的长度为 266 块 (272384 字节)。如果文件大于 266 块, 再启用磁盘地址表的第 12 块所定义的扩展块。当文件大于 $266+256\times 256$ 块时, 再启用第 13 个扩展块。这时文件所占的块数为 $266+256\times 256+256\times 256\times 256$ 。

文件系统采用了一对一映射的方法来实现文件名到 i 节点的转换。目录实际上是一个含有目录表的文件, 对于目录中的每个文件, 在目录表中都会有一个入口项, 入口项中含有文

件名和与文件相应的 i 节点号。当用户键入“more ABC.txt”时，文件系统就在当前目录表中查找名为“ABC.txt”的表项，由此得到与文件 ABC.txt 相应的 i 节点号，然后开始读取含有该文件内容的数据块。

可以通过 df 命令查看系统中节点的大小与数量，如下所示。

```
# df -i
文件系统              Inode (I)   已用   (I)可用   (I)已用% 挂载点
/dev/sda2              1918208  157550  1760658    9%      /
/dev/sda1               26104     35    26069    1%     /boot
tmpfs                  31737      1    31736    1%     /dev/shm
```

其中，/sda1 文件系统的挂载点是/boot，总共有 26104 个 i-node，已经使用 35 个，剩余 26 069 个，已用 i-node 占总数的 1%。

若要查看文件的 i-node 编号，可以使用命令 ls，如下所示。

```
# ls -li
总计 162
650241 drwxr-xr-x  2 root root  4096 07-20 03:03 bin
      2 drwxr-xr-x  4 root root  1024 07-15 19:59 boot
      657 drwxr-xr-x 14 root root  3940 07-25 05:45 dev
292609 drwxr-xr-x 103 root root 12288 07-25 05:44 etc
... ..
```

可以看到，目录也是一种特殊的文件，其中/bin 目录的 i-node 号为 650241。

6.6 文件的类型

在 Linux 中，文件的类型是通过文件权限的首位定义的，可以分为以下几类。

1. 普通文件

普通文件包括源程序文件、脚本文件、可执行程序文件以及各种数据文件。普通文件的文件类型标识位为“-”，使用 ls 命令可以查看文件的类型，如下所示。

```
#ls -l suple
-rw-r--r--  1 root root  540 07-23 08:31 suple
```

2. 目录文件

目录实际上是一种特殊的文件。目录下可以包含文件和子目录。目录文件的类型标识位为“d”，如下所示。

```
#ls -l tmp
drwxrwxrwt 21 root root  4096 07-25 22:49 tmp
```

3. 套接字文件

套接字(socket)是用来进行网络之间通信的常用方法之一。Linux 文件系统可以通过套接字文件实现网络通信。套接字的文件类型标识位为“s”，如下所示。

```
#ls -l x1
srwx----- 2 root root  0 07-25 17:49 x1
```

4. 命名管道

文件系统通过命名管道文件可以实现进程间的通信。命名管道的文件类型标识位为“p”，如下所示。



```
#ls -l pl
prwx----- 2 root root 0 07-25 17:59 pl
```

5. 设备文件

Linux 系统将设备识别为特殊的文件进行处理。设备文件可以分为两类：字符设备和块设备。字符设备的文件类型标识位为“c”，打印机、键盘等都属于字符设备。磁盘、磁带等都属于块设备，块设备的文件类型标识位为“b”。系统的/dev 目录下存放了大量的设备文件，例如字符终端 tty1 的设备文件为/dev/tty1。使用 ls 命令可以看到字符设备的类型标识位为“c”，块设备的文件类型标识位为“b”，如下所示。

```
#ls -l /dev/ttyl
crw----- 1 root root 4, 1 07-25 05:44 ttyl
#ls -l /dev/sdal
brw-r----- 1 root disk 8, 1 07-25 05:41 sdal
```

6. 链接文件

为了方便使用、管理和节省磁盘空间，Linux 允许一个物理文件有一个以上的逻辑名，即可以为一个文件创建一个链接文件，用来表示该文件的另一个名字。链接中的不同文件可为之指定不同的访问权限，从而实现既可共享，又可安全控制的目的。

Linux 文件系统中有两类链接文件：一类叫做硬链接，一类叫做符号链接。使用 ls 命令可以看到符号链接文件的文件类型标识位为“l”，如下所示。

```
# ls -il fuser
390162 lrwxrwxrwx 1 root root 7 07-26 01:18 fuser -> ftpuser
```

6.7 创建 Linux 文件系统

在对磁盘空间进行分区处理之后，需要创建文件系统才能真正使用磁盘。创建文件系统类似于 Windows 下的格式化操作。在 Linux 下，可以使用 mkfs 命令完成文件系统的创建工作，还可以使用 mkbootdisk 命令创建启动盘。

6.7.1 mkfs 命令：创建文件系统

在创建文件系统之前，需要对磁盘空间进行分区处理。Linux 提供了多种分区创建方案，见表 6-3。

表 6-3 Linux 可能的分区方案

分区方案	说明
根分区、swap 分区	适用于磁盘空间有限的计算机
根分区、/boot 分区、swap 分区	较大磁盘空间的典型配置，也是 Red Hat Linux 9 的默认配置
根分区、/boot 分区、/var 分区、swap 分区	可以避免日志文件大小失控
根分区、/boot 分区、/home 分区、swap 分区	对于一台为许多用户提供服务的计算机，可以帮助控制用户占用的空间量

在安装 Red Hat Linux 9 过程中，系统默认把根 (/) 和/boot 目录安装到独立的分区上。

fdisk 是一种功能强大的磁盘分区工具，其详细使用方法请参见第 5 章。

在利用 fdisk 完成磁盘分区后，需要使用 mkfs 命令创建文件系统。mkfs 命令的语法格式如下所示：

```
mkfs [ -V ] [ -t fstype ] [-l filename ] [-c] [filesystem] [ device]
```

其中各选项的具体说明如表 6-4 所示。

表 6-4 mkfs 命令的选项说明

选 项	说 明
-t fstype	文件系统的类型。如果没有指定，默认为 ext2
-c	在建立文件系统之前检查设备是否有物理坏块
-l filename	从文件中读出坏块列表
-V	冗余模式
filesystem	指定创建的文件系统类型
device	指定创建文件系统的设备

例如，对第一个 IDE 接口的主硬盘的第 4 个分区创建 ext3 文件系统，命令如下所示。

```
#mkfs -t ext3 /dev/hda4
```

例如，使用 mkfs 命令在软盘上创建文件系统命令如下所示。

```
#mkfs -t ext2 /dev/fd0
```

必须对所创建的磁盘分区进行挂载才能正常使用，如下所示。

```
#mkdir /mnt/disk4 //创建挂载点
#mount /dev/hda4 /mnt/disk4
#mkdir /mnt/floppy //创建挂载点
#mount /dev/fd0 /mnt/floppy
```

如果希望系统启动时该分区能够自动加载，需要编辑/etc/fstab 文件，加入如下的记录项。

```
hda4 /mnt/disk4 ext3 defaults 1 2
```

6.7.2 mkbootdisk 命令：建立目前系统的启动盘

mkbootdisk 命令可为当前正在运行的系统创建一个启动盘。其制作的启动盘完全自包含，而且包含了一个初始化 RAM 盘的镜像。这个镜像载入了所有系统必须的 SCSI 模块。创建的启动盘会根据/etc/fstab 的建议，搜索根文件系统。惟一需要的参数是要写入启动盘的内核版本号。mkbootdisk 命令的语法格式如下所示。

```
mkbootdisk [--version] [--noprompt] [--verbose] [--device devicefile] [--size size]
[--kernelargs <args>] [--iso] kernel
```

其中各选项的具体说明如表 6-5 所示。

表 6-5 mkbootdisk 命令的选项说明

选 项	说 明
kernel	制作的启动盘的内核版本
--device devicefile	指定创建启动盘的默认设备，默认为/dev/fd0
--noprompt	提示用户插入软盘并确认



续表

选 项	说 明
--verbose	打印程序执行信息
--kernelargs <args>	附加指定的参数到内核命令行参数
--size size	指定镜像文件的大小。默认为标准的 1.44MB
--version	显示版本信息

例如，在/dev/fd0 制作一张启动盘，内核版本号为 2.4.20-8，命令如下所示。

```
# mkbootdisk --device /dev/fd0 2.4.20-8
Insert a disk in /dev/fd0. Any information on the disk will be lost.
Press <Enter> to continue or ^C to abort: //按回车键
读入了 20+0 个块
输出了 20+0 个块
```

将当前目录切换到软盘，通过 ls 命令，可以看到软件中已经创建了启动文件，如下所示。

```
#mount /dev/fd0 /mnt
#cd /mnt
# ls
boot.msg initrd.img ldlinux.sys syslinux.cfg vmlinuz
```

6.8 文件和目录的权限

Linux 是一个多用户操作系统，权限管理是实现 Linux 系统安全的主要途径。通过权限设置可以有效保护系统和用户的数据安全。

6.8.1 权限的常规表示

在 Linux 中，每一个文件和目录都具有相应的权限，如表 6-6 所示。

表 6-6 文件和目录的权限

文 件	目 录
无权限 (-)	无权限 (-)
读 (r) :允许读文件的内容	读 (r) :允许查看目录中有哪些文件和目录
写 (w) :允许向文件中写入数据	写 (w) :允许在目录下创建（或删除）文件、目录
执行 (x) :允许将文件作为程序执行	执行 (x) :允许访问目录（用 cd 命令进入该目录，并查看目录中可读文件的内容）

权限共分为 5 类，其表示方法与含义见表 6-7。

表 6-7 权限的表示法

符 号 表 示	八进制表示	含 义
-	0	没有权限
r	4	read 的缩写，拥有读权限
w	2	write 的缩写，拥有写权限
x	1	execute 的缩写，拥有执行权限
s、S、t、T		特殊权限

权限的作用范围可以分为 4 类，如表 6-8 所示。

表 6-8 权限的作用范围

符号表示	含 义
u	user 的缩写，文件所有者（文件的创建者）
g	group 的缩写，同组用户（与文件所有者同组的用户）
o	other 的缩写，其他用户（系统中除所有者、同组用户以外的用户）
a	all 的缩写，全部的用户，包括所有者、同组用户及其他用户

在 Linux 中，每个文件和目录都与以下 3 个实体相关。

- 属主：文件或目录的所有者。通常情况下就是该文件或目录的创建者，对应表 6-8 中的“u”。
- 用户组：该文件或目录所在的用户组，对应表 6-8 中的“g”。
- 其他用户：其他所有可能对该文件或目录进行操作的用户，对应表 6-8 中的“o”。

相应地，文件或目录的权限由以上 3 部分及文件类型，共 10 个字符位、4 部分组成，如表 6-9 所示。

表 6-9 文件和目录的权限字段

位	1	2	3	4	5	6	7	8	9	10
值	-	r 或-	w 或-	x 或-	r 或-	w 或-	x 或-	r 或-	w 或-	r 或-
说 明	文件类型	属主的权限			组权限			其他用户的权限		

其中，2、5、8 位表示读权限，若要赋予读取的权限，可以将这 3 位对应值设为“r”，若不允许读取则输入“-”；3、6、9 位表示写入权限，若要赋予写权限，可以将这 3 位对应值设为“w”，若不允许写入则设为“-”；4、7、10 位表示可执行权限，若要赋予可执行权限，可以将这 3 位对应值设为“x”，若不允许执行则设为“-”。例如，设定普通文件 ABC.exe 的属主权限为读、写、执行，组权限为读、写，其他用户的权限为读，则文件 ABC.exe 的权限字段为：-rwxrx-r--。

权限除了字符表示法，还有数字表示法。数字表示法是指将读（r）、写入（w）和执行（x）分别以二进制对应位置“1”或置“0”来表示是否有读、写或执行的权限。权限的字符表示法与数字表示法的对应关系见表 6-10。

表 6-10 权限的字符、二进制、八进制表示法

权 限	二 进 制	八 进 制	权 限	二 进 制	八 进 制
---	000	0	--x	001	1
-w-	010	2	-wx	011	3
r--	100	4	r-x	101	5
rw-	110	6	rw x	111	7

表 6-11 列出了几个存取权限的范例。



表 6-11 存取权限的范例

字符表示	二进制表示	八进制表示
rwXrwxrwx	111111111	777
rw-r-----	110100000	640
rw-r--rw-	110100110	646
rw-r--r--	110100100	644

可以使用带选项“-l”的ls命令来查看文件或目录的权限。“ls -l”会以长格式显示文件的信息，包括文件的权限、链接数、创建日期、时间等。例如，以长格式显示文件p.c的详细信息，如图6-3所示。



图 6-3 文件 p.c 的详细信息

6.8.2 chmod 命令：权限设置与修改

命令 chmod 用来为文件或目录赋予权限，其语法格式如下所示。

```
chmod [-c, --changes] [-f, --silent, --quiet] [-v, --verbose] [--reference=file] [-R, --recursive] [-help] [-version]
```

其中各选项的具体说明如表 6-12 所示。

表 6-12 chmod 命令的选项说明

选 项	说 明
-c, --changes	与 verbose 相反，只在有更改时才显示结果
-f, --silent, --quiet	去除大部分的错误信息，静默模式
-v, --verbose	显示全部信息，冗余模式
--reference=file	不再使用自行指定权限的模式进行权限赋值，而使用参考文件 file 的权限
-R, --recursive	以递归方式更改所有的文件及子目录
-help	显示帮助信息并退出
-version	显示版本信息并退出

权限可以采用八进制模式，即以八进制数字表示权限，同表 6-11。例如，预设置普通文件 ABC.exe 的属主权限为读、写、执行，组权限为读、写，其他用户的权限为读，文件 ABC.exe 的权限为 rwxrx-r--，转化为八进制可表示为 764，使用命令如下。

```
#chmod 764 ABC.exe
#ls -l ABC.exe
-rwxrx-r-- 1 root root 224 07-21 23:10 ABC.exe
```

例如，把/home/teacher 目录的权限设为：属主可以读、写、执行，用户组也可以读、写、

执行，其他用户只能读，则权限应设置为 `rw-rw-r--`，转化为数字表示应为 `774`，命令如下所示。

```
#chmod 774 /home/teacher
#ls -l
drwxrwxr-- 2 root root 224 07-21 23:30 teacher
... ..
```

上例中的命令只能修改 `/home/teacher` 目录的权限，如果需要将 `/home/teacher` 目录及其目录中的文件和子目录的权限一并进行修改，则需使用递归选项 `-R`。例如，将 `/home/teacher` 中的所有文件及子目录一并修改为 `rw-rw-rwx`，命令如下所示。

```
# chmod -R 777 teacher1
# ls -l teacher1
drwxrwxrwx 2 root root 4096 07-26 15:58 homework
-rwxrwxrwx 1 root root 0 07-26 15:58 pp.c
#cd ..
# ls -l
drwxrwxrwx 4 teacher1 teacher1 4096 07-26 15:58 teacher1
... ..
```

可以看到，目录 `teacher`、目录 `teacher` 下的子目录 `homework` 以及目录 `teacher` 下的文件 `pp.c` 具有相同的权限。

`chmod` 的权限模式既可以用八进制数字的方式表示，也可以使用字符方式，配合运算符 `+`，`-`，`=`，可以实现增加、减少权限或指定权限。修改权限可选用的字符选项见表 6-13。

表 6-13 权限的字符选项

用户对象表	修 改 操 作	权 限 表 示
u 文件主 g 同组用户 o 其他用户 a 所有用户	+ 赋予权限 - 拒绝权限 = 设置权限	r 读权限 w 写权限 x 执行权限 - 无权限 s, S 设置 set-UID、set-GID t, T 粘滞位

例如，给文件 `ABC.exe` 的所有者增加执行权限，命令行如下。

```
# ls -l
-rw-r--r-- 1 root root 0 07-26 08:16 ABC.exe
# chmod u+x ABC.exe
# ls -l
-rwxr--r-- 1 root root 0 07-26 08:16 ABC.exe
```

例如，去掉文件的同组和其他用户的读权限，可以使用如下命令。

```
# ls -l
-rwxr--r-- 1 root root 0 07-26 08:16 ABC.exe
# chmod g-r,o-r ABC.exe
# ls -l
-rwx----- 1 root root 0 07-26 08:16 ABC.exe
```



设置的各权限之间用 `“,”` 分隔，且 `“,”` 前后不可有空格，否则无法执行命令。

例如，重新设定文件 `ABC.exe` 的其他用户的权限为读取，可以使用如下命令。



```
# ls -l
-rwx----- 1 root root 0 07-26 08:16 ABC.exe
# chmod o=r ABC.exe
# ls -l
-rwx---r-- 1 root root 0 07-26 08:16 ABC.exe
```

例如，重新设定文件 ABC.exe 的其他用户的权限为读取、写入和执行，可以使用如下命令。

```
# ls -l
-rwx---r-- 1 root root 0 07-26 08:16 ABC.exe
# chmod o=rwx ABC.exe
# ls -l
-rwx---rwx 1 root root 0 07-26 08:16 ABC.exe
```

例如，为所有用户（包括属主、组、其他）赋予写权限，可以使用如下命令。

```
# ls -l
-r-x---r-- 1 root root 0 07-26 08:16 ABC.exe
# chmod a+w ABC.exe
# ls -l
-rwx-w-rwx 1 root root 0 07-26 08:16 ABC.exe
```

利用 chmod 的[reference=file]选项可以对文件的权限进行复制，将“reference（参考）”文件的权限直接进行拷贝，如下所示。

```
# ls -l
-rwx---r-- 1 root root 0 07-26 08:16 ABC.exe
# chmod -reference=ABC.exe copy. x
# ls -l
-rwx---rwx 1 root root 0 07-26 08:19 ABC.exe
-rwx---rwx 1 root root 0 07-26 08:19 copy. x
... ..
```

可以看到，copy.x 文件与 ABC.exe 文件拥有了相同的权限。

6.8.3 使用 chomd 命令设置特殊权限

除了读、写、执行权限之外，Red Hat Linux 9 文件系统中还有类特殊权限，包括 set-UID、set-GID 和粘滞位（sticky）。

从表 6-13 可以看到，使用 chmod 命令可以对特殊权限进行设置。如果要加上特殊权限，chmod 命令必须使用 4 位八进制数字格式。其中第 1 位表示特殊权限，set-UDI 对应值为“4”，set-GDI 对应值为“2”，粘滞位对应值为“1”。其余三位表示普通权限（包括属主、用户组及其他用户的权限）。

如果文件的权限被设置为 set-UID，则表示如果该文件是可执行的，则运行该文件的用户将在程序运行期间拥有与该文件的所有者相同的权限。例如，假设有下面的 Shell 脚本程序被存储到 test.exe 文件中。

```
#cat test.exe
#!/bin/bash
echo $PATH
ps -aux >>/home/process
#
```

使用 chown 命令设置 test 脚本的属主，使用 chmod 命令设定脚本的权限，如下所示。

```
chown root:root test.exe
chmod 4777 test.exe
```



```
# ls -l test.exe
-rwsrwxrwx 1 root root 0 07-26 08:19 test.exe
```

此时 test.exe 的所有者是 root，由于设置了 set-UID 位，任何运行该文件的用户都会临时具有 root 身份。无疑这是非常危险的，应慎重使用 set-UID 权限。

与 set-UID 类似，如果文件的权限被设置为 set-GID，则表示如果该文件是可执行的，则运行该文件的用户会在运行期间拥有与该文件的用户组相同的权限。例如，可以对文件 test1.exe 设定 set-GID，如下所示。

```
chown root:nobody test1.exe
chmod 2775 test1.exe
# ls -l test1.exe
-rwxrwsr-x 1 root root 0 07-26 09:19 test1.exe
```

如果文件被设置了粘滞位 (sticky-bit)，则表示该文件如果是可执行的，一旦被装载进内存就一直驻留内存。例如，对文件 test2.exe 设定粘滞位，命令如下。

```
chmod 1775 test2.exe
# ls -l test2.exe
-rwxrwxr-t 1 root root 0 07-26 10:19 test1.exe
```

只需将特殊权限所对应的值相加，就可一次完成文件的特殊权限设置。例如，对文件 test3.exe 设置 set-UID、set-GID 及粘滞位，各项值累加为 $4+2+1=7$ ，则命令如下所示。

```
chmod 7775 test3.exe
# ls -l test2.exe
-rwsrwsr-t 1 root root 0 07-23 13:19 test3.exe
... ..
```

6.8.4 umask 命令：设置文件或目录的默认权限

每一个新创建的文件或目录系统都会被自动赋予一个默认的权限。可以使用 umask 命令设置文件或目录的默认权限。umask 命令的语法格式如下所示。

```
umask [mask]
```

其中，[mask] 可以是由 4 个 8 进制数字组成的权限掩码，直接使用 umask 命令可以显示系统默认的权限掩码，如下所示。

```
#umask
0022
```

通常新建文件的默认权限值为 0666，新建目录的默认权限值为 0777。与当前的权限掩码为 0022 相减，每个新增加的文件的最终权限值为 $0666-0022=0644$ ，而新建目录的最终权限值为 $0777-0022=0755$ 。

例如，新建文件 test，新建设目录 T，通过 ls 命令可以看到生成的最终权限，如下所示。

```
#umask
0022
# touch test
# ls -l test
-rw-r--r-- 1 root root 0 07-26 09:06 test //test 的权限为 rw-r--r--，即 644
# mkdir T
# ls
T test
# ls -l
```



```
drwxr-xr-x 2 root root 4096 07-26 09:07 T //T 的权限为 rwxr-xr-x, 即 755
-rw-r--r-- 1 root root 0 07-26 09:06 test
```

可以使用 `umask` 命令重新设置权限掩码。例如, 将系统默认的权限掩码设为 0002, 则新建文件的最终权限为 $0666 - 0002 = 0664$ (即 `rw-rw-r--`), 新建目录的最终权限为 $0777 - 0002 = 0775$ (即 `rw-rw-r--`), 如下所示。

```
# umask 0002
# touch test3
# ls -l test3
-rw-rw-r-- 1 root root 0 07-26 09:40 test3
# mkdir new
# ls -l
drwxrwxr-x 2 root root 4096 07-26 09:42 new
-rw-rw-r-- 1 root root 0 07-26 09:40 test3
```

使用 `umask` 命令也可以直接通过表 6-13 的权限参数直接设置新建文件或目录的默认权限。例如, 将默认权限改为所有者读、写、执行, 同组用户读, 其他用户为读、执行, 可以使用如下命令。

```
# umask u=rwx,g=r,o=rw
# umask
0031
# touch p
# ls -l p
-rw-r--rw- 1 root root 0 07-26 09:14 p
# mkdir M
# ls -l
drwxr--rw- 2 root root 4096 07-26 09:15 M
-rw-r--rw- 1 root root 0 07-26 09:14 p
```

从本例可以看出, “`umask u=rwx,g=r,o=rw`” 与 “`umask 0031`” 作用相同, 但文件的执行权限不可由 `umask` 命令的 “x” 选项指定。

6.8.5 setfacl 命令: 设置访问控制列表 ACL

基于用户和用户组的权限机制奠定了 Linux 系统安全的基础, 但在十几年的使用中它也暴露出一些不足, 例如权限只能基于用户或用户组进行设定, 无法为用户组中的个别几个用户设定不同的权限。为了增加文件或目录权限管理的灵活性, 从 Red Hat Linux 9 开始, 访问控制列表 (ACL) 被引入到系统中。通过 ACL, 可以根据需要对用户的权限进行定制, 它支持标准的 ext3 文件系统、NTFS 文件系统以及 SMB 文件系统。

启动 ACL 之前, 首先应对需要进行访问控制的分区或目录进行挂载, 具体可以参照 `mount` 命令, 格式如下。

```
mount -o acl <device> <mount point>
```

例如, 在 `/myfile` 目录上挂载带有 ACL 支持的 `/dev/sda1` 分区, 命令如下所示。

```
#mount -o acl /dev/sda1 /myfile
```

在分区中成功启动 ACL 后, 可以使用 `setfacl` 命令来添加、修改或删除访问权限。`setfacl` 命令的语法格式如下所示。

```
setfacl [--set , --set-file][ -x, --remove][ -m, --modify][ -d, --default][ -k,
--remove-default][ --test] [ --version]
[ -R, --recursive][ -b, --remove-all][ --help] [ acl_spec ]
```


其中各选项的具体说明如表 6-14 所示。

表 6-14 setfacl 命令的选项说明

选 项	说 明
--set, --set-file	设置 ACL 文件或目录
-x, --remove	删除 ALC 项
-m, --modify	修改 ACL 项
-d, --default	设置默认的 ACL
-k, --remove-default	删除默认的 ACL，默认 ACL 包括所有者、用户组、其他
--test	仅进行测试
-R, --recursive	递归地将操作应用于所有文件及目录
-b, --remove-all	删除扩展 ACL
--help	显示帮助信息
--version	显示 setfacl 命令的版本信息
acl_spec	指定的 ACL 项

setfacl 命令识别如表 6-15 所示的 ACL 格式。

表 6-15 ACL 格式说明

ACL 格式	说 明
[d[efault]:] [u[ser]:]uid [:perms]	设置用户的权限，4 个字段分别表示默认设置、用户名、用户 ID、权限
[d[efault]:] g[roup]:gid [:perms]	设置用户组的权限，4 个字段分别表示默认设置、用户组名、用户组 ID、权限
[d[efault]:] m[ask][:] [:perms]	设置权限掩码，3 个字段分别表示默认设置、掩码、权限
[d[efault]:] o[ther][:] [:perms]	设置其他用户的权限，4 个字段分别表示默认设置、其他用户名、其他用户名 ID、权限

可以使用“-m”选项添加用户或用户组的权限。例如，对用户 teacher1、teacher2 设置对 /homework 目录的读、写和执行权限，可以运行下面的命令。

```
#setfacl -m u:teacher1:rwX /home/homework
#setfacl -m u:teacher2:rwX /home/homework
```

对用户 student1、student2 设置对 /homework 目录的读、写权限，可以运行下面的命令。

```
#setfacl -m u:student1:rw /home/homework
#setfacl -m u: student2:rw /home/homework
```

例如，对用户组 director 设置对 /home/director 目录的读权限，命令如下所示。

```
#setfacl -m u: director:r /home/director
```

要对权限进行修改，与添加 ACL 相同，可以使用带“-m”选项的 setfacl 命令。例如，修改上例中的 director 用户组的权限为读、写、执行，命令如下所示。

```
#setfacl -m u: director:rwX /home/director
```

可以使用“-x”选项来删除一个用户或用户组的权限。例如，删除上例中 director 用户组的权限，可以使用如下命令。

```
#setfac l -x u: director /home/director
```

可以使用“-d”选项对一个文件或目录设置默认的 ACL。例如，对 /home/ftp 目录的用户



组设定默认的权限为读、写入，命令如下。

```
#setfacl -d g:rw /home/ftp
```



若组中用户重新指定了权限，则默认组权限将被覆盖，即具体指定的 ACL 权限优先于默认 ACL 权限。

6.9 修复受损的文件系统

非正常关机经常会导致文件系统出现错误。在 Windows 下通常使用 `chkdisk` 命令进行修复，在 Linux 下可以使用 `fsck`、`e2fsck`（专门针对 `ext2` 文件系统的修复命令）等命令完成类似的工作。

6.9.1 fsck 命令：检测文件系统并尝试修复错误

`fsck` 命令用于检测并修复出错的文件系统，操作对象可以是设备名（如 `“/dev/hdc1”`、`“/dev/sda2”`），也可以是挂载点（如 `“/”`、`“/usr”`、`“/home”`）、文件系统的标签（如 `“LABEL=root”`），还可以是 UUID（例如 `“UUID=8868abf6-88c5-4a83-98b8-bfc24057f7bd”`）。该命令的语法格式如下所示。

```
fsck [-s][-A][-C][-V][-R][-T][-N][-P] [-t fstype ][filesys]
```

其中各选项的具体说明如表 6-16 所示。

表 6-16 fsck 命令的选项说明

选 项	说 明
-a	非交互式模式，不询问用户如何操作即自动进行修复
-r	交互模式，在修复之前进行询问
-s	串行化 fsck 检查操作
-t fslist	指定检测的文件系统
-A	根据 <code>/etc/fstab</code> ，检测所有的文件系统
-v	运行命令时，显示详细的处理信息
-c	显示检测进度
-N	不执行，仅显示做些什么
-P	如果设置 <code>-A</code> 选项，在检测其他文件系统的同时，并行检测 <code>root</code> 文件系统
-R	如果设置 <code>-A</code> 选项，在检测其他文件系统的同时，跳过对 <code>root</code> 文件系统的检测
-f	强制进行检查
-T	不显示标题
-V	冗余模式
filesys	指定修复的文件系统

文件系统受损后，经常提示如下信息。

```
... ..
Giver root password for maintenance
(or type Control-D for nomal setup):
```

在提示符下输入 root 密码，然后按 Enter 键进入修复模式。通过命令 fsck 检查文件系统，如果有错误，将尝试修复，如下所示。

```
# fsck //检查文件系统
fsck 1.32 (09-Nov-2002)
/: recovering journal
Clearing orphaned inode 229425 (uid=0, gid=0, mode=0100644, size=2472)
Clearing orphaned inode 213207 (uid=0, gid=0, mode=0140600, size=0)
Clearing orphaned inode 213206 (uid=0, gid=0, mode=0140600, size=0)
Clearing orphaned inode 213205 (uid=0, gid=0, mode=0140600, size=0)
Clearing orphaned inode 213200 (uid=27, gid=27, mode=0100600, size=0)
Clearing orphaned inode 213199 (uid=27, gid=27, mode=0100600, size=0)
Clearing orphaned inode 213187 (uid=27, gid=27, mode=0100600, size=0)
Clearing orphaned inode 213186 (uid=27, gid=27, mode=0100600, size=0)
Clearing orphaned inode 213185 (uid=27, gid=27, mode=0100600, size=0)
/: clean, 309388/971520 files, 1585359/1939848 blocks
/dev/sda1 is mounted.
/boot: recovering journal
/boot: clean, 41/26104 files, 12728/104391 blocks
```

如果修复一个没有卸载的文件系统，系统将提示如下信息。

```
# fsck /dev/sda1
fsck 1.39 (29-May-2006)
/dev/sda1 is mounted.
WARNING!!! Running fsck on a mounted filesystem may cause
SEVERE filesystem damage.
Do you really want to continue (y/n)? no
check aborted.
```

上面的提示表明在一个正被挂载的文件系统上使用 fsck 命令，可能会造成文件系统被严重损坏。系统会询问：如果继续使用 fsck 命令按“y”，否则按“n”。

在建立服务器时，最好建立两个以上的 Linux 主文件系统。在一个文件系统工作时，另一个文件系统做系统备份。当一个文件系统需要修复时，可以用正常的文件系统引导机器，然后卸载受损的文件系统，并用 fsck 命令进行修复。

如果服务器上只安装了一个文件系统，可以使用支持 fsck 命令的光盘或软盘启动系统，然后用 fsck 命令对该受损文件系统进行检查和修复。

6.9.2 e2fsck 命令：检查 ex2 文件系统

e2fsck 命令是 Linux 操作系统中非常有用的工具，其主要功能是检查文件系统的正确性，对受损的文件系统进行修复。由于系统长期运行，很可能会因为网络攻击或非正常关机而导致文件系统受损，造成诸如文件系统的 i 节点表和磁盘的内容不一致等问题。e2fsck 是专门为 Linux 操作系统的 ext2 文件系统设计的检查和修复工具，其命令的语法格式如下。

```
e2fsck[-p][-a][-c][-n][-y][-r][-d][-f][-k][-v][-s][-t][-D][-F][-S][-V][-l][-bsuperb
lock][-Bblocksize]
[-Lbad_blocks_file][-Cfd][-jexternal-journal][-Eextended_options][filesystem]
```

其主要选项及其说明如表 6-17 所示。



表 6-17 e2fsck 命令选项及说明

选 项	说 明
-a	不询问便自动进行修复，与-p 选项作用相同。保留该选项仅为了向后兼容，建议尽可能使用-p 选项
-b superblock	使用指定的超级块（superblock），而不使用预设的超级块
-B blocksize	指定搜索的超级块的大小
-c	调用 badblocks 程序对磁盘做只读性扫描，查找存在的坏块。如果该选项被定义了两次，则使用非破坏性的读/写方法对磁盘坏块进行扫描
-C fd	将扫描过程信息记录在指定的文件中，使整个检测过程受到监控
-d	显示调试信息
-D	优化文件系统目录。如果文件系统支持目录索引，则重新建立索引；对于传统的线性目录或较小的目录，使用排序和压缩技术
-E extended_options	设置 e2fsck 扩展选项，选项之间用逗号分隔
-f	即使文件系统很简单也强制检测
-F	在开始检测前将文件系统的设备缓冲区清空
-j external-journal	设置外部日志的路径
-k	与-c 选项联合使用，对已在坏块列表中列出的坏块不做修改，通过使用 badblocks 程序检测出的新坏块将被添加到坏块列表中
-l	将文件中指定的区块添加到坏块列表中
-L bad_blocks_file	与-l 命令相似，只是在将指定区块添加到坏块列表前，清空坏块列表
-n	以只读方式打开文件系统，并且假定所有交互的答案均为“no”
-p	不进行询问，自动修复文件系统
-r	该选项并无实际意义，只是为了向前兼容
-s	如果文件系统已是标准的字节顺序，则不进行任何操作
-S	忽略当前的字节顺序，直接进行交换
-t	显示 e2fsck 的时间统计
-v	冗余模式
-V	显示版本信息
-y	在交互模式下，假定所有的答案均为“yes”
filesystem	指定修复的文件系统

与 fsck 命令类似，在使用 e2fsck 命令修复文件系统时，必须将要修复的文件系统卸载，否则是极不安全的，甚至会造成文件系统的崩溃。因此在安装 Linux 文件系统时，建议至少创建两个以上的主文件系统，这样在一个文件系统受损时，可以通过另一个文件系统对其进行修复。

例如，假定计算机上安装了两个主文件系统——hda1 和 hda2，当 hda1 文件系统出现故障时，可以使用 hda2 引导系统，然后卸载 hda1 并利用 e2fsck 修复，命令行如下。

```
#umount /dev/hda1
#e2fsck -av /dev/hda2
```

修复完成后，重新启动计算机，即可使用 hda1 的文件系统重新引导系统。

6.10 mount 与 umount 命令

软盘、光盘、USB 存储器（包括 U 盘、移动硬盘等）是 Linux 系统重要的外部设备。在 Linux 中，外部设备是被当作文件来使用的，Red Hat Linux 9 系统一般会自动对其进行识别并挂载。当自动识别失败或需要用户手工设定挂载参数时，可以使用 mount 命令来协助完成。

6.10.1 mount 命令格式

mount 命令实际上是用于挂载指定的文件系统，其语法格式如下所示。

```
mount [-r][ -a][ -A][ -h][ -t][ device][ dir]
```

其主要选项及其说明如表 6-18 所示。

表 6-18 mount 命令选项及说明

选 项	说 明
-r	表示挂载的文件系统为只读
-a	表示挂载/etc/fstab 文件中列出的所有文件系统
-A	卸载当前已被挂载的所有文件系统
-h	显示帮助信息
-t	挂载的文件系统类型
device	指定挂载的设备
dir	指定挂载点，即指定设备被挂载的位置

其中，[device]指需要挂载的设备名称，[dir]是一个已存在的空目录，被挂载设备的根路径将指定在该目录位置。挂载点可以指定目录树中的任何位置，但必须是空目录。选项“-t”指定了挂载的文件系统类型，mount 命令可以挂载的文件系统类型及说明见表 6-19。

表 6-19 mount 命令可挂载的文件系统类型及说明

可挂载的文件系统类型	说 明
auto	自动检测文件系统并进行挂载
ext2	Linux 中使用最多的文件系统
ext3	ext2 基础上的日志文件系统
iso9660	CDROM 标准文件系统
hpfs	OS/2 使用的高性能文件系统，但在 Linux 系统中只能作为只读文件系统使用
minix	Linux 的早期版本所采用的文件系统，该文件系统分区不能超过 64MB，一般只用于软盘或 RAMDISK
msdos	MS-DOS 使用的文件系统，不支持长文件名
nfs	网络文件系统
ntfs	Windows NT 使用的文件系统
vfat	扩展的 MS-DOS 文件系统，支持长文件名，被 Windows95/98/NT/2000 所采用



如果不带任何选项地使用 `mount` 命令，默认会显示目前已挂载的文件系统，如下所示。

```
# mount
/dev/sda2 on / type ext3 (rw)
none on /proc type proc (rw)
usbdevfs on /proc/bus/usb type usbdevfs (rw)
/dev/sda1 on /boot type ext3 (rw)
none on /dev/pts type devpts (rw,gid=5,mode=620)
none on /dev/shm type tmpfs (rw)
```

6.10.2 `umount` 命令格式

使用 `umount` 命令可以卸载已挂装的文件系统，但不能卸载根分区。`umount` 命令的语法格式如下所示。

```
umount [-hV]
umount -a [-dflnr] [-t vfstype] [-O options]
umount [-dflnr] dir | device [...]
```

其主要选项及其说明如表 6-20 所示。

表 6-20 `umount` 命令选项及说明

选 项	说 明
-V	显示版本信息并退出
-h	显示帮助信息并退出
-v	Verbose 模式
-n	卸载时的信息不写入 <code>/etc/mstab</code> 文件
-r	如果卸载失败，尝试以只读方式挂载
-a	所有在 <code>/etc/mstab</code> 文件中描述的设备均被卸载（对于 2.7 以上版本， <code>proc</code> 文件系统不会被卸载）
-t	指定卸载的文件系统类型， <code>umount</code> 命令会只对指定类型的文件系统进行卸载
-f	强制卸载

6.10.3 软盘的挂载与卸载

使用 `mount` 命令可以对软盘进行挂载，并指定挂载的目录。例如，将软盘挂载到 `/mnt/floppy` 目录下，如果 `/mnt/floppy` 目录不存在，则需要提前创建，命令如下所示。

```
#mkdir /mnt/floppy
#mount /dev/fd0 /mnt/floppy
```

使用 `umount` 命令可以卸载软磁盘，命令如下所示。

```
#umount /dev/fd0
```

也可用挂载点作为参数进行卸载，命令如下所示。

```
#umount /mnt/floppy
```

6.10.4 CDROM 的挂载与卸载

CDROM 的挂载与软盘的方法类似，命令格式如下。

```
mount -t iso9660 /dev/cdrom 光盘挂载点
```

其中 `iso9660` 是光盘的标准文件系统类型。例如，在 `/mnt/cdrom` 目录下挂载光驱，如果

/mnt/cdrom 目录不存在,则需要提前创建,命令如下所示。

```
#mkdir /mnt/cdrom
#mount -t iso9660 /dev/cdrom /mnt/cdrom
```

通常光盘被挂载后,如果不进行卸载,光驱是无法打开的。必须卸载已经安装的光盘文件系统之后,才可以顺利取出光盘。卸载光驱时使用 `umount` 命令,命令格式如下。

```
umount [光盘挂载点或光盘设备文件名]
```

例如,对上例中挂载的光驱进行卸载,命令如下所示。

```
umount /dev/cdrom
```

也可以利用挂载点作为参数进行卸载,命令如下所示。

```
umount /mnt/cdrom
```

在 Linux 中,系统还提供了专门用于弹出或关闭光盘驱动器的命令 `eject`。`eject` 命令会先调用 `umount` 命令对光驱进行卸载,然后弹出光盘。`eject` 命令格式如下。

```
eject [-t] [-n] [光盘挂载点或光盘设备文件名]
```

其中选项“-t”用于关闭指定的光盘驱动器,不会将光驱弹出。直接使用 `eject` 命令而不指定任何参数,将弹出默认的设备。可以使用选项“-n”查看系统默认的弹出设备,如下所示。

```
#eject -n
eject: device is `/dev/hdc'
```

6.10.5 USB 存储设备的挂载与卸载

随着硬件成本的不断降低,USB 存储设备由于具有存取速度快、稳定性高、即插即用、携带方便等优点越来越得到广泛的应用和发展。无论是在 Windows 系统还是在 Linux 系统中,均已提供了对该类设备的支持。通常使用的 USB 存储设备包括 U 盘和移动硬盘,一般不需要安装驱动程序即可在 Linux 系统中使用。但由于 USB 存储设备的标准还不是很统一,Linux 不能保证对所有的 USB 存储设备都能正确识别。

在 Linux 中,USB 存储设备通常是作为 SCSI 设备来使用的。SCSI 设备文件名以“sd”开头,后接设备序号(按字母排序)及分区号(按数字排序)。例如,第一个 SCSI 设备上的第一个主分区表示为“/dev/sda1”,第三个 SCSI 设备的第 2 个主分区表示为“/dev/sdc2”。

对于 U 盘,如果 U 盘中没有进行分区则使用 SCSI 设备的设备名直接进行挂载;如果 U 盘中存在分区,则使用带分区的 SCSI 设备名进行挂载。例如,将 U 盘挂载到/mnt/flash 目录下,命令如下所示。

```
#mkdir /mnt/flash
#mount -t vfat /dev/sdb /mnt/flash
```

卸载时使用 `umount` 命令,命令如下所示。

```
#umount /dev/sdb
```

对于 USB 移动硬盘,在使用 `mount` 命令进行挂载时不能对整个硬盘设备进行挂载,必须指定相应的分区。例如,将移动硬盘第一个分区挂载到/mnt/disk 目录下,命令如下所示。

```
#mkdir /mnt/disk
#mount -t vfat /dev/sdc1 /mnt/disk
```

可以使用命令 `umount` 卸载,如下所示。

```
#umount /dev/sdc1
```



6.11 其他文件和目录管理常用命令

文件和目录管理涉及的命令比较多，在现存各个版本的 Linux 系统中，各命令的功能大体相同。常用的文件和目录管理命令大约有 50 多个，最基本的命令包括 ls、cd、cp、chown、chgrp、find、mkdir、rmdir、mv、ln 等。熟练掌握这些命令是深入学习 Linux 的基础。

6.11.1 文件和目录操作中常用的通配符

在 Linux 文件系统中，可以使用通配符来匹配多个选择。常用的通配符如表 6-21 所示。

表 6-21 常用通配符说明

通 配 符	说 明
*	代表文件中任意长度的任意字符
?	代表文件中的任一个字符
[...]	匹配任意一个中括号中的字符，中括号里可以是一个用破折号分割的字母或数字范围
前导字符串{...}后继字符串	大括号中的字符串逐一匹配前导字符串和后继字符串

例如，当前目录下存在 cars、cat、can、cannon、truck、bus、bike 几个文件，列出所有以字母“c”开头的文件，可使用如下所示命令。

```
#touch car cats can cannon truck bus bike
#ls c*
cars can cannon
```

列出所有以字母“b”开头的文件名，命令如下所示。

```
#ls b*
bike bus
```

列出所有第一个字母为“c”，最后一个字母为“n”的文件，命令如下所示。

```
#ls c*n
can cannon
```

列出包含字母“a”的所有文件，命令如下所示。

```
#ls *a*
cars cat can cannon
```

列出当前目录下的所有文件，命令如下所示。

```
#ls *
bike bus cars cat can cannon truck
```

通配符“?”只能匹配任意一个字符。例如，列出所有第 3 个字母是“n”的文件，命令如下所示。

```
#ls ??n*
can cannon
```

列出第一个字母是“b”，第三个字母是“s”的所有文件，命令如下所示。

```
#ls b?s*
bus
```

中括号“[]”表示一个匹配的字符集，例如，[123456]与[1-6]都表示匹配数字 1、2、3、

4、5 和 6 中之一。大写字母 A 到 D 之间的任意一个字符可用[A-D]表示。多个集合之间可以用逗号分隔，例如[1-10,a-z,A-Z]表示数字 1~10，小写字母 a 到小写字母 z，大写字母 A 到大写字母 Z。一个集合前若有前缀“!”，则表示除了集合包含的字符以外的所有字符组成的集合。例如，所有辅音组成的字符集可写成[!aeiou]。

例如，显示所有以字母 b 或 c 开头的文件名，命令如下所示。

```
#ls [b,c]*
bike bus car cat can cannon
```

例如，列出所有以字母 b 或 c 开头，以字母 s 或 k 结尾的文件，命令如下所示。

```
#ls [b,c]*[s,k]
bus cars
```

大括号“{}”是用来查找文件的一个常用方法。例如，以长格式列出 cars、cans 和 cats 文件信息，可以使用如下命令。

```
#ls -l c{ar,an,at}s
```

6.11.2 attr 命令：XFS 文件系统对象的扩展属性

attr 命令用来设置 XFS 文件系统的扩展属性。扩展属性可以实现用户“名:值”对与 XFS 文件系统目录的关联。该命令的语法格式如下所示。

```
attr [ -L ][ -R ][ -q ][ -s attrname ] [ -V attrvalue ][ -g attrname ][ -r attrname ]
```

其中各选项的具体说明如表 6-22 所示。

表 6-22 attr 命令的选项说明

选 项	说 明
-s attrname	从标准输入设备读入属性值
-g attrname	搜索命名对象并显示输出其属性值
-r attrname	删除相关属性
-L	如果指定-L 选项并且对象是一个符号链接，那么操作的是引用符号链接的对象的属性。如果不指定这个选项，那么操作的是符号链接本身
-R	如果-R 选项指定并且处理过程拥有适当的优先级，操作将发生在 root 用户的属性空间，而不是一般用户的属性空间
-q	如果-q 选项指定，将不打印状态信息
-V attrvalue	设置属性值

6.11.3 basename 命令：显示文件或目录的基本名称

basename 命令将显示目录或文件的基本名称，即输出时去掉目录或文件的路径前缀，以及文件的后缀信息。该命令的语法格式如下所示。

```
basename [--help][--version]
```

其中各选项的具体说明如表 6-23 所示。

表 6-23 basename 命令的选项说明

选 项	说 明
--help	显示 basename 命令的帮助信息
--version	显示 basename 命令的版本信息



basename 命令将显示指定完整路径的最后部分，也就是目录或文件的真正名称。例如，下面完整的路径信息是/home/teacher，执行 basename 命令之后，得到目录的基本名称 teacher，如下所示。

```
# basename /home/teacher //显示目录的基本名称
teacher
```

6.11.4 cd 命令：改变目录

cd (Chage Directory) 命令用于更改当前目录，表 6-24 列出了常见的 cd 命令应用示例。

表 6-24 cd 命令与运行结果

命 令	说 明
cd	切换到当前用户的主目录
cd..	切换到当前目录的上一层目录，例如当前目录为/home/student，使用该命令可以将当前目录移到/home
cd../..	切换到当前目录的上二层目录，例如当前目录为/home/student/student1，使用该命令可以将当前目录移动到/home
Cd~	切换当前目录为当前用户的主目录，适用于任何用户
cd /	切换当前目录到根目录，即返回到/



在 Linux 中，引用目录名、计算名或域名时使用正斜杠“/”，而在 Windows 中使用反斜杠“\”。

例如，当前系统中存在目录结构如图 6-4 所示，其中用户当前目录为/home/student。

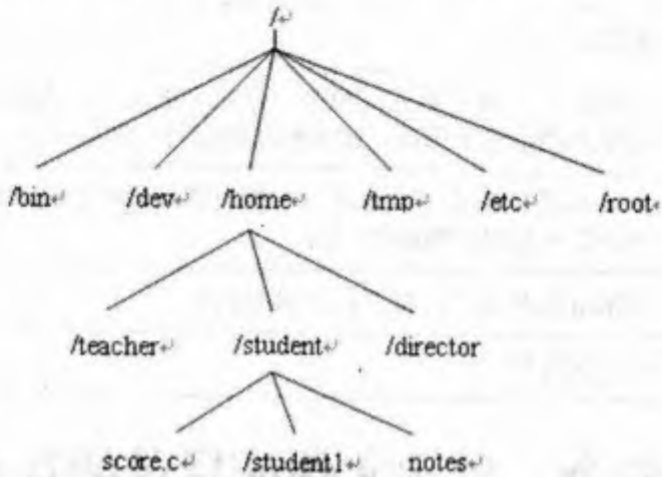


图 6-4 目录结构图

若改变当前目录为/home/director，可以使用相对路径，命令如下。

```
#pwd
/home/student
#cd ../director
#pwd
/home/director
```

也可以使用绝对路径切换到/home/director，命令如下。

```
#pwd
/home/student
#cd /home/director
```



```
#pwd
/home/director
```

将当前目录由/home/stuent 切换为/home/student/student1。由于 student1 目录位于 student 目录下，可以采用相对路径，命令如下。

```
#pwd
/home/student
#cd student1
#pwd
/home/student/student1
```

也可以采用绝对路径，命令如下。

```
#pwd
/home/student
#cd /home/student/student1
#pwd
/home/student/student1
```

从 student1 目录返回根目录，可以使用如下命令。

```
#pwd
/home/student/student1
#cd /
#pwd
/
```

也可以使用相对路径，命令如下。

```
#pwd
/home/student/student1
#cd ../../..
#pwd
/
```

6.11.5 cp 命令：复制文件

cp 命令复制源文件到目标文件，或者复制多个源文件到目标文件夹。cp 命令通常用于实现文件或目录的复制，与 MS-DOS 下的 copy 命令相似。该命令的语法格式如下所示。

```
cp[-a,--archive][--backup[=CONTROL]][-b][--copy-contents][-d][--no-dereference][-f,
--force][--version]
[-i,--interactive][-H][-l,--link][-L,--dereference][-p][--preserve[=ATTR_LIST]][--s
no-preserve=ATTR_LIST]
[--parents][-P][-R,-r,--recursive][--remove-destination][--reply={yes,no,query}][--
sparse=WHEN][--help]
[--strip-trailing-slashes][-s,--symbolic-link][-S,--suffix=后缀][--target-directory=
目录][-u,--update]
[-v,--verbose][-x,--one-file-system]
```

其中各选项的具体说明如表 6-25 所示。

表 6-25 cp 命令的选项说明

选 项	说 明
-a, --archive	等同于-dpR。该选项常在复制目录时使用，该选项保留链接、文件属性，并递归地复制目录
--backup[=CONTROL]	为每个已存在的目标文件创建备份文件
-b	类似--backup，但不接受任何参数
--copy-contents	当使用递归模式时，复制特殊文件的内容
-d	等同于--no-dereference --preserve=link



续表

选 项	说 明
--no-dereference	不会找出符号链接指示的真正目标
-f, --force	如果无法打开已存在的文件，会删除该文件并再尝试打开。如果目标文件或目录已存在，就覆盖它，并且不做提示
-i, --interactive	覆盖文件前需要确认。与-f选项正好相反，在覆盖已有文件时，会让用户回答“Y”来确认
-H	使用命令列中的符号链接指示的真正目标
-l, --link	链接而非复制文件
-L, --dereference	一定先找出符号链接指示的真正目标
-p	等同于--preserve=mode, ownership, timestamps
--preserve[=ATTR_LIST]	若可能，保留指定的文件属性（默认值为mode, ownership, timestamps），额外的属性有links、all
--sno-preserve=ATTR_LIST	不保留指定的文件属性
--parents	复制前先在<目录>创建来源文件路径中的所有目录
-P	等同于“--no-dereference”
-R, -r, --recursive	复制目录及目录内的所有项目。若给出的源目标是一个目录，那么cp将递归复制该目录下所有的子目录和文件，不过这要求目标也是一个目录名
--remove-destination	尝试打开目标文件前先删除已存在的目标文件（与--force选项相对）
--reply={yes,no,query}	指定如何处理已存在的目的地文件
--sparse=WHEN	控制创建sparse文件的方式
--strip-trailing-slashes	删除参数中所有<来源>文件/目录末端的斜杠
-s, --symbolic-link	只创建符号链接而不是复制文件
-S, --suffix=后缀	自行指定备份文件的<后缀>
--target-directory=目录	将所有参数指定的<来源>文件/目录复制至<目录>
-u, --update	只在<来源>文件比目标地文件新，或目标文件不存在时才进行复制
-v, --verbose	详细显示处理的步骤
-x, --one-file-system	不会跨越文件系统进行操作
--help	显示帮助信息并离开
--version	显示版本信息并离开

常用的命令行组合与执行结果见表 6-26。

表 6-26 cp 命令格式与运行结果示例

命 令 格 式	运 行 结 果
cp 文件 1 文件 2	复制源文件 1 的内容到目标文件 2 上，目标文件有一个新的创建日期和索引节点号
cp 多个文件（文件名之间用空格分隔） 目录 1	复制多个文件到目录 1 中
cp -f 文件 1 文件 2	如果已有一个文件 2 存在，该命令不显示任何提示就用文件 1 覆盖文件 2
cp -i 文件 1 文件 2	如果已有一个文件 2 存在，该命令在覆盖文件 2 之前会给出提示
cp -p 目录 1 目录 2	复制目录 1 的内容到目录 2，如果目录 1 下还有子目录，则一并进行复制
cp -u 文件 1 文件 2	如果文件 2 已存在，但文件 1 比文件 2 新，则不显示任何提示就覆盖文件 2

例如，复制当前目录下的文件 `a.doc` 到目录 `/home/my`，命令如下所示。

```
# cp a.doc /home/my/           //复制文件到目录/home/my
# cd /home/my/                 //改变目录到/home/ my
# ls
a.c
```



指定的目标文件夹必须存在，如本例中，指定的目标文件夹 `/home/my` 必须存在。

例如，通过命令 `cp`，复制文件 `a.doc`。由于文件 `a.doc` 在 `/home/my/` 下已经存在，并且复制时指定了选项“`-b`”，所以原来的文件 `a.doc` 将以备份的形式进行存储，存储的名称为 `a.doc~`。命令如下所示。

```
# cp -b a.doc /home/my/       //复制文件，将已经存在的文件进行备份
# cd /home/my/
# ls
a.c a.c~
```

例如，通过命令 `cp`，复制文件并指定备份文件字尾的字符为 `@`，可以看到文件夹 `/home/my/` 下将出现文件 `a.doc@`，命令如下所示。

```
# cp -bv --suffix=@ a.doc /home/my/ //备份时选择字尾的字符
# cd /home/my/
# ls
a.doc a.doc~ a.doc@
```

例如，通过命令 `cp`，复制当前路径下的目录 `Desktop` 到指定路径 `/home/my/`，可以看到在 `/home/my/` 下多了一个目录 `Desktop`，命令如下所示。

```
# cp -fr Desktop/ /home/kk/       //复制文件夹到指定路径
# cd /home/kk/
# ls
a.doc a.doc~ a.doc@ Desktop
```

例如，将当前目录下的文件 `a.txt` 和 `b.txt` 复制到 `/home/teacher` 目录中，命令如下所示。

```
#cp a.txt b.txt /home/user1
```

`cp` 命令可以在同一目录下，换名复制一个文件，源文件保持不变。例如，将当前目录下的 `a.txt` 复制一份且取名为 `b.txt`，放在当前目录下，命令如下所示。

```
#cp a.txt b.txt
```

若源文件是普通文件，则直接拷贝到目标文件；若是目录，需要使用“`-r`”选项才能将整个目录复制到目标位置。例如，复制 `/home/teacher` 目录下所有内容到 `/home/student` 目录下，命令如下所示。

```
#cp -r /home/teacher /home/student
```

6.11.6 chattr 命令：改变文件的属性

`chattr` 命令用于改变 linux 第二个扩展文件系统的属性。该命令的语法格式如下所示。

```
chattr [-R][-V] [-+=AacDdijsSu] [-v version] files
```

其中各选项的具体说明如表 6-27 所示。



表 6-27

chattr 命令的选项说明

选 项	说 明
-R	递归处理指定目录及子目录中的所有文件
-V	冗余模式，显示命令运行时的所有处理过程
-v version	设置文件或目录的版本
+	使文件或目录具有该项属性
-	删除文件或目录具有的该项属性
=	设置文件或目录的该项属性为相应的值
file	指定操作的文件

chattr 命令可用的属性包括以下几项。

- A: 不修改时间。
- a: 文件只能以追加模式写入, 根用户拥有设置或清除该属性的权限。
- b: 不更新文件或目录的最后存取时间。
- c: 自动进行压缩。
- D: 当目录具有 D 属性时, 对目录的修改会直接同步写入磁盘, 该属性等同于 `dirsSync` 命令。
- d: 使用 `dump` 命令时, 不对具有该属性的文件进行备份。
- E: 表明压缩存在错误。
- I: 目录被 Hash 树索引。
- i: 具有 i 属性的文件不可被修改、删除、重命名、建立链接或对文件进行写入操作。
- j: 在向具有 j 属性的文件执行写入操作时, 先记录日志。
- s: 安全删除文件, 文件块被清零并写回到磁盘。
- S: 当一个具有 S 属性的文件被修改时, 将修改同步写入磁盘, 等同于 `sync` 命令。
- T: 具有该属性的目录将被移到目录树的最顶层。
- t: 具有该属性的文件在进行文件合并时, 不会留有碎块。
- u: 当具有 u 属性的文件被意外删除时, 可以恢复。
- X: 压缩文件可以被直接访问。
- Z: 压缩文件损坏。

例如，首先使用命令 `lsattr`，显示文件 `abc.doc` 的属性。可以看到，文件 `abc.doc` 的各项属性为空白。通过命令 `chattr`，修改文件 `abc.doc` 的属性，使其具有追加写入的属性，最后再使用命令 `lsattr` 显示文件 `abc.doc` 的属性。执行命令过程如下所示。

```
# lsattr abc.doc           //显示文件 abc.doc 的属性
----- abc.doc
# chattr +a abc           //使文件具有附加属性
# lsattr abc.doc           //显示文件 abc.doc 的属性
-----a----- abc.doc
```

例如，删除文件 abc.doc 具有的 a 属性，命令如下所示。

```
# lsattr abc.doc           //显示文件 abc.doc 的属性
-----a----- abc.doc
```



```
# chattr -a abc.doc //删除文件 abc.doc 的附加属性
# lsattr abc.doc //显示文件 abc.doc 的属性
----- abc.doc
```

例如，直接设置文件 abc.doc 的属性值，使其具有 a 属性，命令如下所示。

```
# chattr =a abc.doc //设置文件的附加属性
```

6.11.7 cksum 命令：文件的 CRC 校验

cksum 命令将显示文件的校验和文件的字节数。该命令的语法格式如下所示。

```
cksum [--help][--version]
```

其中各选项的具体说明如表 6-28 所示。

表 6-28 cksum 命令的选项说明

选 项	说 明
--help	显示 cksum 命令的帮助信息
--version	显示 cksum 命令的版本信息

例如，对文件进行校验，命令如下所示。

```
# ls -l
总用量 68
-rw-r--r-- 1 root root 960 8月 7 15:43 anaconda-ks.cfg
-rw-r--r-- 1 root root 49492 8月 7 15:38 install.log
-rw-r--r-- 1 root root 7113 8月 7 15:38 install.log.syslog
# cksum anaconda-ks.cfg //计算文件 anaconda-sk.cfg 的校验和及文件长度
1354783707 960 anaconda-ks.cfg
```

6.11.8 chown 命令：改变所有者

chown 命令用于改变文件或目录的拥有者，其语法格式如下所示。

```
chown[-c,--changes][--dereference][--no-dereference][--from=目前所有者:目前组][--f,
--silent,--quiet]
[ --reference=参考文件][ -R, --recursive][ -v, --verbose][ --help][ --version]
```

其中各选项的具体说明如表 6-29 所示。

表 6-29 chown 命令的选项说明

选 项	说 明
-c, --changes	类似--verbose，但只有在有更改时才显示结果
--dereference	受影响的是符号链接所指示的对象，而非符号链接本身
-h, --no-dereference	会影响符号链接本身，而非符号链接所指示的目标（只有当系统支持更改符号链接的所有者时，此选项才有效）
--from=目前所有者:目前组	只当每个文件的所有者和用户组符合选项的指定时，才会更改所有者和用户组。其中一个可以省略，这时已省略的属性就不需要符合原有的属性
-f, --silent, --quiet	去除大部分的错误信息
--reference=参考文件	使用<参考文件>的所属组，而非指定的<组>
-R, --recursive	递归处理所有的文件及子目录，可以一次修改某个目录下所有文件的所有者
-v, --verbose	处理任何文件都会显示信息
--help	显示此帮助信息并离开
--version	显示版本信息并离开



chown 命令可以改变每一个指定文件或目录的用户/用户组的拥有权限。如果只指定一个用户名（或用户 ID），那么该用户将成为指定文件的拥有者，文件的用户组并不改变。如果同时指定用户名和用户组名，那么指定文件的用户和用户组都要改变。

如果用户名后面跟随一个冒号或点号，而不出现用户组名，那么指定文件的用户组也要改变，变为指定用户所在的用户组。如果冒号或点号 and 用户组名称被指定，但是用户名被忽略，那么只有文件的用户组属性改变。在这种情况下，chown 命令的执行效果等同于命令 chgrp 的执行效果。

通过命令 chown，改变目录/test 的用户和用户组信息，使得该目录的用户为 yang，用户组为 yang，而不是原来的用户和用户组 root，命令如下所示。

```
# ls -l
总用量 36
-rwxr-xr-x 1 root root 16368 6月 2 22:42 bubbleSort
-rw-r--r-- 1 root root 232 6月 2 22:42 bubbleSort.bb
-rw-r--r-- 1 root root 380 6月 2 22:42 bubbleSort.bbg
-rw-r--r-- 1 root root 80 6月 2 22:50 bubbleSort.da
-rw-r--r-- 1 root root 85 6月 5 19:36 kk2.c
drwxr-xr-x 2 root root 4096 8月 3 11:59 test
# chown yang:yang test
# ls -l
总用量 36
-rwxr-xr-x 1 root root 16368 6月 2 22:42 bubbleSort
-rw-r--r-- 1 root root 232 6月 2 22:42 bubbleSort.bb
-rw-r--r-- 1 root root 380 6月 2 22:42 bubbleSort.bbg
-rw-r--r-- 1 root root 80 6月 2 22:50 bubbleSort.da
-rw-r--r-- 1 root root 85 6月 5 19:36 kk2.c
drwxr-xr-x 2 yang yang 4096 8月 3 11:59 test
```

在指定的用户名后面跟随一个冒号或点号，而不出现用户组名，那么指定文件或目录的用户组也要改变，变为指定用户所在的用户组，如下所示。

```
# ls -l //显示文件的详细信息
总用量 36
-rwxr-xr-x 1 root root 16368 6月 2 22:42 bubbleSort
-rw-r--r-- 1 root root 232 6月 2 22:42 bubbleSort.bb
-rw-r--r-- 1 root root 380 6月 2 22:42 bubbleSort.bbg
-rw-r--r-- 1 root root 80 6月 2 22:50 bubbleSort.da
-rw-r--r-- 1 root root 85 6月 5 19:36 kk2.c
drwxr-xr-x 2 yang yang 4096 8月 3 11:59 test
# chown root: test //改变文件的用户和用户组
# ls -l
总用量 36
-rwxr-xr-x 1 root root 16368 6月 2 22:42 bubbleSort
-rw-r--r-- 1 root root 232 6月 2 22:42 bubbleSort.bb
-rw-r--r-- 1 root root 380 6月 2 22:42 bubbleSort.bbg
-rw-r--r-- 1 root root 80 6月 2 22:50 bubbleSort.da
-rw-r--r-- 1 root root 85 6月 5 19:36 kk2.c
drwxr-xr-x 2 root root 4096 8月 3 11:59 test
```

如果冒号或点号 and 用户组名称被指定，但是用户名忽略，那么只有文件的用户组属性改变，如下所示。

```
# ls -l
总用量 36
```



```
-rwxr-xr-x 1 root root 16368 6月 2 22:42 bubbleSort
-rw-r--r-- 1 root root 232 6月 2 22:42 bubbleSort.bb
-rw-r--r-- 1 root root 380 6月 2 22:42 bubbleSort.bbg
-rw-r--r-- 1 root root 80 6月 2 22:50 bubbleSort.da
-rw-r--r-- 1 root root 85 6月 5 19:36 kk2.c
drwxr-xr-x 2 root root 4096 8月 3 11:59 test
# chown :users kk2.c
# ls -l
总用量 36
-rwxr-xr-x 1 root root 16368 6月 2 22:42 bubbleSort
-rw-r--r-- 1 root root 232 6月 2 22:42 bubbleSort.bb
-rw-r--r-- 1 root root 380 6月 2 22:42 bubbleSort.bbg
-rw-r--r-- 1 root root 80 6月 2 22:50 bubbleSort.da
-rw-r--r-- 1 root users 85 6月 5 19:36 kk2.c
drwxr-xr-x 2 root root 4096 8月 3 11:59 test
```

例如，将文件 data 的所有者改为 teacher，命令如下所示。

```
$chown teacher data
```

例如，将/root/file1.doc 文件复制到用户 teacher1 的主目录/home/teacher1 中，复制之后可以发现此文件的拥有者仍然是 root，命令如下所示。

```
# ls -l file1.doc
-rw-rw-r-- 1 root root 18 07-27 16:21 file1.doc
# cp file1.doc /home/teacher1
# ls -l /home/teacher1/file1.doc
-rw-rw-r-- 1 root root 18 07-27 16:22 /home/teacher1/file1.doc
```

为此，使用 chown 将 file.doc 文件的所有权赋予 teacher1，命令如下所示。

```
# chown teacher1 /home/teacher1/file1.doc
# ls -l /home/teacher1/file1.doc
-rw-rw-r-- 1 teacher1 root 18 07-27 16:22 /home/teacher1/file1.doc
```

可以看到，file1.doc 文件的所有者已改为 teacher1，但用户组仍为 root。如果希望同时修改 file1.doc 文件的所有者和用户组均为 teacher1，命令如下所示。

```
# chown teacher1:teacher1 /home/teacher1/file1.doc
# ls -l /home/teacher1/file1.doc
-rw-rw-r-- 1 teacher1 teacher1 18 07-27 16:22 /home/teacher1/file1.doc
```

6.11.9 chgrp 命令：更换用户组

chgrp 允许用户修改文件或目录所属的用户组，但该用户要么是根用户要么同时属于被设置的新的用户组，否则无法修改成功。chgrp 命令的语法格式如下所示。

```
chgrp[-c,--changes][--dereference][-h,--no-dereference][-f,--silent,--quiet][--reference=参考文件]
[ -R, --recursive][ -v, --verbose][ --help][ --version]
```

其中各选项的具体说明如表 6-30 所示。

表 6-30 chgrp 命令的选项说明

选 项	说 明
-c, --changes	类似于--verbose，但只有在有更改时才显示结果
--dereference	会影响符号链接所指示的对象，而非符号链接本身
-h, --no-dereference	会影响符号链接本身，而非符号链接所指示的目标（当系统支持更改符号链接的所有者时，此选项才有效）



续表

选 项	说 明
-f, --silent, --quiet	去除大部分的错误信息
--reference=参考文件	使用<参考文件>的所属组，而非指定的<组>
-R, --recursive	递归处理所有的文件及子目录，可以一次修改某个目录下所有文件的用户组
-v, --verbose	处理任何文件都会显示信息
--help	显示帮助信息并离开
--version	显示版本信息并离开

例如，变更文件 test 的用户组属性为 yang，同时输出变更信息，命令如下所示。

```
# chgrp -v test yang //改变文件 test 的用户组属性为 yang
```

例如，根据指定文件的属性改变文件的用户组属性。首先使用命令 ll，显示当前文件夹下文件的详细信息，发现文件 test 所属的用户组为 yang。然后使用命令“chgrp --reference=123.doc test”改变文件 test 的用户组属性，使得文件 test 的用户组属性和参考文件 123.doc 的用户组属性相同。然后再使用命令 ll，列出文件夹下所有文件的详细信息。可以看到，文件 test 的用户组属性变为 root。具体执行过程如下所示。

```
# ll //显示当前文件夹下的文件详细信息
总用量 8
-rw-r--r-- 1 root root 12 4月23 15:30 123.doc
-rw-rw-r-- 1 yang yang 51 5月 9 10:08 test
# chgrp --reference=123.doc test //改变文件 test 的用户组信息
# ll //显示当前文件夹下的文件详细信息
总用量 8
-rw-r--r-- 1 root root 12 4月23 15:30 123.doc
-rw-rw-r-- 1 kk root 51 5月 9 10:08 test
```

例如，将文件 data 的用户组改为 staff，命令如下所示。

```
# chgrp staff data
```



在指定新用户组时，可以使用用户组名也可以使用用户组的 GID。

6.11.10 dirname 命令：显示文件的除名字外的路径

dirname 命令可以显示出指定路径的除了主文件名之外的路径前缀。该命令的语法格式如下所示。

```
dirname [--help][--version]
```

其中各选项的具体说明如表 6-31 所示。

表 6-31 dirname 命令的选项说明

选 项	说 明
--help	显示 dirname 帮助信息
--version	显示 dirname 版本信息

例如，显示指定路径中除了主文件名之外的路径前缀，命令如下所示。

```
# dirname /home/teacher/test.doc //显示除了主文件名称前的路径前缀
/home/teacher
```

6.11.11 fold 命令：自动换行以适应指定列宽

fold 命令用来调整输出的每一行的宽度。该命令的语法格式如下所示。

```
fold [-b,--bytes] [-s,--spaces] [-w,--width=WIDTH] [--help] [--version]
```

其中各选项的主要说明如表 6-32 所示。

表 6-32 fold 命令主要选项说明

选 项	说 明
-b, --bytes	列宽的单位为字节数而不是列数
-s, --spaces	遇到空格换行
-w, --width=WIDTH	指定每列的列数，默认为 80
--help	帮助信息
--version	版本信息

例如，调整文件 acb.doc 的输出列宽为 5，命令如下所示。

```
# cat abc //显示文件 abc.doc 的内容
a
ab
abc
abcd
abcde
abcdef
abcdefg
abcdefgh
abcdefghi
abcdefghij
abcdefghijk
abcdefghijkl
# fold -w 5 abc //调整文件 abc.doc 的列宽为 5 个字符
a
ab
abc
abcd
abcde
abcde
f
abcde
fg
abcde
fgh
abcde
fghi
abcde
fghij
abcde
fghij
```



k
abcde
fghij
kl

6.11.12 find 命令：查找目录或文件

find 命令用于查找文件，其语法格式如下所示。

```
find [起始目录] [搜索条件] [操作]
```

其中，[起始目录]是指，命令将从该目录起，遍历其下所有的子目录，查找满足条件的文件，缺省是当前目录。[搜索条件]是一个逻辑表达式，当表达式为“真”时，搜索条件成立；为“假”时不成立。搜索条件的一般形式见表 6-33。

表 6-33 find 命令搜索条件的一般表达式

搜索条件	说 明
-name ‘字符串’	查找文件名中包含所给字符串的所有文件
-user ‘用户名’	查找属于指定用户的文件
-group ‘用户组名’	查找属于指定用户组的文件
-type x	查找类型为 x 的文件，类型包括 b 块设备文件、c 字符设备文件、d 目录文件、p 命名管道文件、f 普通文件、l 符号链接文件和 s socket 文件
-atime n	查找 n 天以前被访问过的文件
-size n	指定文件大小为 n
-perm	查找符合指定权限值的文件或目录
-mount	要查找文件时不跨越文件系统挂载点
-follow	如果 find 命令遇到符号链接文件，就跟踪到链接所指向的文件
-cpio	对匹配的文件使用 cpio 命令，将文件备份到磁带设备中
-newer file1 ! file2	查找更改时间比文件 file1 新，但比文件 file2 旧的文件
-prune	不在指定的目录中查找，如果同时指定 -depth 选项，那么 -prune 将被 find 命令忽略
-ok	和 exec 作用相同，但在执行每一个命令之前，都会给出提示，由用户来确定是否执行
-depth	在查找文件时，首先查找当前目录，然后再在其他子目录中查找

find 命令的常见操作见表 6-34。

表 6-34 find 命令常用可执行操作

可执行操作	说 明
-exec 命令名 {} \;	不需确认执行命令。注意，“{}”代表找到的文件名，“}”与“\”之间有空格
-print	送往标准输出

例如，从当前目录查找所有以.txt 结尾的文件并在屏幕上显示出来，命令如下所示。

```
$ find . -name '*.txt' -print
```

例如，从根目录查找类型为符号链接的文件，并将其删除，命令如下所示。

```
$ find / -type l -exec rm { } \;
```

例如，从当前目录查找用户 tom 的所有文件并在屏幕上显示，命令如下所示。


```
$ find . -user 'tom' -print
```

例如，显示当前目录中大于 20 字节长的.c 文件名，命令如下所示。

```
$ find . -name '*.c' -size +20c -print
```

显示当前目录中恰好 10 天前访问的文件名，命令如下所示。

```
$ find . -atime 10 -print
```

例如，显示当前目录中不到 10 天前访问的文件名，命令如下所示。

```
$ find . -atime -10 -print
```

例如，查找/home 目录下权限为 640 的文件或目录，命令如下所示。

```
#find /home -perm 640
```

例如，搜索根目录下大于 100KB 的文件，命令如下所示。

```
#find / -size +100K -print
```

例如，搜索根目录下小于 500KB 的文件，命令如下所示。

```
#find / -size -500K -print
```

例如，在当前目录中查找所有文件名以.doc 结尾，且更改时间在 3 天以上的文件，找到后进行删除，且删除前给出提示，命令如下所示。

```
#find . -name '*.doc' -mtime +5 -ok rm { } \;
```

例如，在当前目录下查找所有链接文件，并以长格式显示文件的基本信息，命令行如下所示。

```
# find . -type l -exec ls -l {} \;
lrw-rw-r-- 1 root root 36 07-27 14:34 ./example2
lrw-rw-r-- 1 root root 72 07-27 14:36 ./example3
lrw-rw-r-- 1 root root 36 07-27 14:36 ./example1
```

例如，在当前目录中查找文件名由一个小写字母、一个大写字母、两个数字组成，且扩展名为.doc 的文件，命令如下所示。

```
#find . -name '[a-z][A-Z][0-9][0-9].doc' -print
```

6.11.13 file 命令：显示文件类型

file 命令可以根据参数测试定文件的类别。file 使用三种测试集，依次为：文件系统测试、magic 数字测试和语言测试。其中，文件系统测试成功执行后，将输出文件的类型。该命令的语法格式如下所示。

```
file [-b][-c][-L][-z][-C][-k][ -n][ -f namefile][ -m list][ -v][ --help]
```

其中各选项的具体说明如表 6-35 所示。

表 6-35 file 命令的选项说明

选 项	说 明
-b	简略模式，在输出行中不显示文件名
-c	对 magic 文件进行检测
-L	操作对象为符号链接所指向的对象
-z	检测压缩文件
-C	将 magic.mgc 写入到包含预解析版本的文件中
-k	在找到第一次匹配时不停止，继续搜索
-n	在检测每一文件后强制标准输出刷新



续表

选 项	说 明
-f namefile	从指定的文件读入需要检测的文件名
-m list	指定包含 magic 数字的文件替换列表
-v	显示版本信息
--help	显示帮助信息

例如，显示当前目录下所有文件的文件类型，命令如下所示。

```
# file * //显示当前文件夹下文件的类型
dict:      ASCII English text
fonts.dir: ASCII text
my.doc:     ASCII text
mypaper:    ASCII text
mypaper.bak: ASCII text
mypaper.doc: ASCII text
xx00:      ASCII text
xx01:      ASCII English text
```

6.11.14 filterdiff 命令：从 diff 文件中提取或去除差异

filterdiff 命令用来提取或去除 diff 文件的不同。该命令的语法格式如下所示。

```
filterdiff[-xPAT][-iPAT][--format=context|unified][-z][-n][-pN][--strip=N][--addpre
fix=PREFIX][--version]
[-s][-v][--filter][--list][--grep][--help]
```

其中各选项的具体说明如表 6-36 所示。

表 6-36 filterdiff 命令的选项说明

选 项	说 明
-x PAT	排除匹配 PAT 的文件
-i PAT	仅包含匹配 PAT 的文件
--format=context, unified	设置输出模式
-z	解压.gz 和.bz2 文件
-n	显示行号
-p N	忽略初始路径名
--strip=N	去除初始路径名
--addprefix=PREFIX	指定路径的前缀
-s	显示文件的添加和删除
-v	冗余模式
--filter	以 'filterdiff'方式运行
--list	以 'lsdiff'方式运行
--grep	以'grepdiff'方式运行
--help	显示帮助信息
--version	显示版本信息

6.11.15 findfs 命令：通过卷标或 UUID 查找文件系统

findfs 命令可以通过卷标或 UUID 查找文件系统，找到的设备名将输出到标准输出设备。该命令的语法格式如下所示。

```
findfs [LABEL=<label>] [UUID=<uuid>]
```

其中各选项的具体说明如表 6-37 所示。

表 6-37 findfs 命令的选项说明

选 项	说 明
LABEL=<label>	指定卷标
UUID=<uuid>	指定 UUID
--help	显示 findfs 命令的帮助信息
--version	显示 findfs 命令的版本信息

例如，通过命令 findfs，查找卷标名为 “/” 和 “/boot” 的文件系统，命令如下所示。

```
# findfs LABEL=/ //通过卷标名查找文件系统
/dev/sda2
# findfs LABEL=/boot //通过卷标名查找文件系统
/dev/sda1
```

6.11.16 locate 命令：定位文件或目录

locate 命令查找符合指定模式的文件和目录。指定模式可以是文本也可以采用正则表达式。locate 命令通过已建立的数据库/var/lib/slocate 进行搜索，而不是直接在硬盘当中逐一寻找。因此使用 locate 比使用 find 命令更快、更简便。但因为 locate 是经由数据库来搜寻，而数据库的更新一般是每天一次（多数在夜间进行），所以如果在数据库更新之前搜寻，locate 命令是无法查到用户新建的文件的。locate 命令的语法格式如下所示。

```
locate [-d<path>--database=<path>] [-l<level>] [-n<num>] [-o<file>--output=<file>] [-v--verbose]
[-r<regex>--regex=<regex>] [-q] [-h,--help] [-V,--version]
```

其中各选项的具体说明如表 6-38 所示。

表 6-38 locate 命令的选项说明

选 项	说 明
-d <path> --database=<path>	预设 locate 指定使用的数据库。预设的数据库在/var/lib/slocate 中，名字为 slocate.db
-l <level>	设置安全等级，如下所示。 0：关闭安全检测，可以更快地完成搜索。 1：启动安全检测，该项为默认设定
-n <num>	指定显示结果的数目
-o <file> --output=<file>	指定创建的数据库
-v --verbose	冗余模式，当创建数据库时，显示信息
-r <regex> --regex=<regex>	使用 POSIX 正则表达式搜索数据库
-q	静默模式，不显示错误信息
-h, --help	在线帮助
-V, --version	显示版本信息



例如，通过命令 `locate`，查找含有后缀为“.linux”的字符串的文件，命令如下所示。

```
# locate *.linux //查找文件
/usr/share/doc/shadow-utils-4.0.3/README.linux
/usr/share/doc/gawk-3.1.1/README_d/README.linux
/usr/share/doc/sysklogd-1.4.1/README.linux
/usr/share/doc/ppp-2.4.1/README.linux
/usr/share/doc/star-1.5a08/README.linux
/usr/share/doc/cdrecord-2.0/README.linux
/usr/share/doc/sane-backends-1.0.9/README.linux
/usr/share/doc/ethereal-0.9.8/README.linux
```

例如，查找含有后缀为“.linux”的字符串的文件，且仅输出前 3 行，命令如下所示。

```
# locate -n 3 *.linux
/usr/share/doc/shadow-utils-4.0.3/README.linux
/usr/share/doc/gawk-3.1.1/README_d/README.linux
/usr/share/doc/sysklogd-1.4.1/README.linux
```

例如，查找所有包含“shadow”字符串的文件名，命令如下所示。

```
# locate shadow
/etc/shadow
/usr/bin/pgmdeshadow
/usr/share/icons/gnome/16x16/stock/image/stock_shadow.png
/usr/share/icons/gnome/16x16/stock/text/stock_fontwork-2dshadow.png
... ..
```



与 `find` 命令相比，`locate` 命令从数据库中查找文件，速度比 `find` 指令要快许多。但对于新增或已经被删除的文件，`locate` 命令便无法立即察觉。

6.11.17 ls 或 dir 命令：列目录

`ls`(`list`的缩写)命令可以列出当前目录的内容，`dir`命令是`ls`命令的一个别名，取自 `directory` 的缩写。通常，列出的文件会以不同的颜色进行显示，不同颜色代表不同的文件类型，表 6-39 列出了文件类型与颜色的对应关系。

表 6-39 文件类型与颜色的对应关系

文件类型	颜色
目录	深蓝色
一般文件	浅灰色
执行文件	绿色
图形文件	紫色
链接文件	浅蓝色
压缩文件	红色
FIFO 文件（命名管道）	棕色
设备文件	黄色

`ls` 命令还会对特定类型的文件用符号进行标识，表 6-40 列出了常用的标识符号。

表 6-40 特定文件类型标识符

符 号	说 明
.	表示隐藏文件
/	表示一个目录名
*	表示一个可执行文件
@	表示一个符号链接文件
	表示管道文件
=	表示 socket 文件

ls 命令的语法格式如下所示。

```
ls[-a,--all][-A,--almost-all][--block-size=大小][--author][-b,--escape][-B,--ignore-backups][-c]
[-C][--color[=WHEN]][-d,--directory][-D,--dired][-f][-F,--classify][--full-time][-g]
[-G,--no-group]
[-h,--human-readable][--si][-H,--dereference-command-line][--indicator-style=方式]
[-i,--inode]
[-I,--ignore=样式][-k][-l][-L,--dereference][-m][-n,--numeric-uid-gid][-N,--literal]
[-o][-p,--file-type]
[-q,--hide-control-chars][--show-control-chars][-Q,--quote-name][--quoting-style=方式]
[-r,--reverse]
[-R,--recursive][-s,--size][-S][--sort=WORD][--time=WORD][--time-style=STYLE][-t]
[-T,--tabsize=COLS][-u][-U][-v][-w,--width=COLS][-x][-X][-l][--help][--version][]
```

其中各选项的具体说明如表 6-41 所示。

表 6-41 ls 命令的选项说明

选 项	说 明
-a, --all	不隐藏任何以 . 字符开始的项目
-A, --almost-all	列出除了 . 及 .. 以外的任何项目
--author	显示出每个文件著作者
-b, --escape	以八进制溢出序列表示不可打印的字符
--block-size=大小	块以指定<大小>的字节为单位
-B, --ignore-backups	不列出任何以 ~ 字符结束的项目
-c	配合 -lt：根据 ctime 排序及显示 ctime（文件状态最后更改的时间）。 配合 -l：显示 ctime 但根据名称排序，否则根据 ctime 排序
-C	每栏由上至下列出项目
--color[=WHEN]	控制是否使用色彩分辨文件。WHEN 可以是“ever”、“always”或“auto”其中之一
-d, --directory	遇到目录时列出目录本身而非目录内的文件
-D, --dired	产生适合 Emacs 的 dired 模式使用的结果
-f	不进行排序，-aU 选项生效，-lst 选项失效
-F, --classify	加上文件类型的指示符号（*、/、=、@、其中之一）
--full-time	即 -l --time-style=full-iso
-g	类似-l，但不列出所有者



续表

选 项	说 明
-G, --no-group	不列出任何有关组的信息
-h, --human-readable	以容易理解的格式显示出文件大小（例如 1KB 、234MB 、2GB）
--si	类似 -h，但单位取 1000 的次方而不是 1024
-H, --dereference-command-line	使用命令列中的符号链接指示的真正目标文件
--indicator-style=方式	指定在每个项目名称后加上指示符号<方式>: none（默认）、classify（-F）、file-type（-p）
-i, --inode	显示出每个文件的 inode 号
-I, --ignore=样式	不显示任何符合<样式>的项目
-k	即 --block-size=1K
-l	使用较长格式列出信息，包括文件的类型与权限、链接数、文件所有者、文件所有者所属的组、文件大小、最近修改时间、文件名。详情见图 6-3
-L, --dereference	当显示符号链接的文件信息时，显示符号链接所指示的对象而非符号链接本身的信息
-m	所有项目以逗号分隔，并填满整行行宽
-n, --numeric-uid-gid	类似-l，但列出 UID 及 GID 号
-N, --literal	显示未经处理的项目名称（例如不特别处理控制字符）
-o	类似 -l，但不列出有关组的信息
-p, --file-type	加上文件类型的指示符号（/、=、@、 其中之一）
-q, --hide-control-chars	以 ? 字符代替无法显示的字符
--show-control-chars	直接显示无法显示的字符（这是默认方式，除非调用的程序名称是 ls，而且是在终端机画面输出结果）
-Q, --quote-name	将项目名称括上双引号
--quoting-style=方式	使用指定的引用<方式>显示项目的名称，包括 literal、locale、shell、shell-always、c、escape
-r, --reverse	依相反次序排列
-R, --recursive	同时列出所有子目录层
-s, --size	以块大小为单位列出所有文件的大小
-S	根据文件大小排序
--sort=WORD	以下是可选用的 WORD 和它们代表的相应选项 extension -X status -c none -U time -t size -S atime -u time -t access -u version -v use -u
--time=WORD	显示 WORD 所代表的时间而非修改时间： atime、access、use、ctime 或 status。加上 --sort=time 选项时会以指定时间作为排序索引
--time-style=STYLE	时间的显示风格，包括 full-iso、 long-iso、 iso、 locale、 +FORMAT
-t	按修改时间排序
-T, --tabsize=COLS	设定按 Tab 键每次移动到下一输出字段，而不是 8 列
-u	配合 -lt：显示访问时间而且依访问时间排序。配合-l：显示访问时间但根据名称排序。否则根据访问时间排序

续表

选 项	说 明
-U	不进行排序，依文件系统原有的次序列出项目
-v	根据版本进行排序
-w, --width=COLS	自行指定屏幕宽度而不使用目前的数值
-x	逐行列出项目而不是逐栏列出
-X	根据扩展名排序
-l	每行只列出一个文件
--help	显示此帮助信息并离开
--version	显示版本信息并离开

例如，以缩略格式显示目录内容，命令如下所示。

```
# ls //以缩略格式显示目录内容
cal_txt      finger_txt  id_txt      newgrp_txt  suple_txt   who.txt
finger2_txt  groups_txt  last_txt    suple3_txt  suple_txt~  w.txt
finger3_txt  groups_txt2 ln           suple3_txt~ w
```

例如，以长格式显示目录内容，包括权限、用户名、修改时间等，命令如下所示。

```
# ls -l //以长格式显示目录内容，包括权限、用户名、修改时间等
总计 224
-rw-r--r-- 1 root root 2163 07-24 13:00 cal_txt
-rw-r--r-- 1 root root 212 07-23 06:37 finger2_txt
-rw-r--r-- 1 root root 0 07-23 06:39 finger3_txt
-rw-r--r-- 1 root root 248 07-23 06:31 finger_txt
-rw-r--r-- 1 root root 35 07-23 04:22 groups_txt
-rw-r--r-- 1 root bin 140 07-23 04:32 groups_txt2
-rw-r--r-- 1 root root 144 07-23 04:11 id_txt
-rw-r--r-- 1 root root 1655 07-22 19:19 last_txt
drwxr-xr-x 5 root root 4096 07-26 09:42 ln
-rw-r--r-- 1 root bin 143 07-23 04:24 newgrp_txt
-rw-r--r-- 1 root root 25098 07-26 19:15 suple3_txt
-rw-r--r-- 1 root root 24031 07-26 16:00 suple3_txt~
-rw-r--r-- 1 root root 25198 07-24 20:16 suple_txt
-rw-r--r-- 1 root root 24827 07-24 20:06 suple_txt~
-rw-r--r-- 2 root root 196 07-23 06:48 w
-rw-r--r-- 1 root root 46 07-23 06:48 who.txt
-rw-r--r-- 2 root root 196 07-23 06:48 w.txt
```

例如，以块大小为单位列出所有文件的大小，命令如下所示。

```
# ls -s //显示所用的块数
总计 224
8 cal_txt      8 groups_txt  8 ln          32 suple_txt   8 w.txt
8 finger2_txt  8 groups_txt2 8 newgrp_txt  32 suple_txt~
4 finger3_txt  8 id_txt      32 suple3_txt 8 w
8 finger_txt   8 last_txt    28 suple3_txt~ 8 who.txt
```



ls 命令各选项可以组合使用。



例如，如果需要列出当前目录的所有内容（包括那些以“.”开头的隐含文件），并以“长格式”从屏幕输出文件的详细信息，可以使用选项“-al”。执行命令如下所示。

```
# ls -al /root
drwxr-x--- 18 root root 4096 07-26 16:00 .           //当前目录
drwxr-xr-x 24 root root 4096 07-25 05:43 ..          //父目录
-rw----- 1 root root  997 07-15 20:58 anaconda-ks.cfg //普通文件
-rw----- 1 root root 2827 07-24 21:32 .bash_history //隐藏文件
-rw-r--r-- 1 root root   24 2006-07-13 .bash_logout  //隐藏文件
... ..
```

其中文件名为“.”表示当前目录，对应行列出了当前目录的详细信息。文件名为“..”表示当前目录的上一级目录，即父目录，对应行列出了父目录的详细信息。文件名前有“.”符号的文件表示隐藏文件，只有使用了“-a”选项时隐藏文件才会显示出来。



在查看目录时还有一个经常用到的命令 ll，该命令实质上是命令“ls -l --color=tty”的别名。

6.11.18 ln 命令：链接文件或目录

为了使用、管理的方便和节省磁盘空间，Linux 允许一个物理文件有一个以上的逻辑名，即可以为一个文件创建一个链接文件，用来表示该文件的另一个名字。可为链接中的不同文件指定不同的访问权限，从而实现既可共享，又可安全控制的目的。

ln 命令用来创建链接文件或目录。该命令的语法格式如下所示。

```
ln[--backup[=CONTROL]] [-b] [-d, -F, --directory] [-f, --force] [-i, --interactive] [-n, --no-dereference] [--version]
[-s, --symbolic] [-S, --suffix=SUFFIX] [-v, --verbose] [--target-directory=DIRECTORY] [--help]
```

其中各选项的具体说明如表 6-42 所示。

表 6-42 ln 命令的选项说明

选 项	说 明
--backup[=CONTROL]	对每一个存在的目标文件创建备份
-b	与--backup 相似，但不接受任何参数
-d, -F, --directory	超级用户允许创建目录的硬链接
-f, --force	强制模式。无条件删除存在的目标文件
-i, --interactive	交互模式。删除目标文件前先征求用户的意见
-n, --no-dereference	把符号链接的目标目录视为一般文件
-s, --symbolic	符号链接（默认方式是硬链接）
-S, --suffix=SUFFIX	覆盖普通的备份后缀
-v, --verbose	在建立链接时显示每个文件名称
--target-directory=DIRECTORY	指定创建链接的目录
--help	帮助信息
--version	版本信息

Linux 文件系统中有两类链接文件：一类叫做硬链接，另一类叫做符号链接。硬链接的文件类型标识位与被链接的文件相同。不带选项“-s”使用 ln 命令可以建立硬链接文件，例如对 sysv 文件建立硬链接，命令如下所示。

```
# ls -il sysv
390162 -rw-r--r-- 1 root root 0 07-26 00:51 sysv
# ln sysv syslink
# ls -il sysv slink
390162 -rw-r--r-- 2 root root 0 07-26 00:51 slink
390162 -rw-r--r-- 2 root root 0 07-26 00:51 sysv
```

从本例中可以看出硬链接文件 slink 与被链接的文件 sysv 指向同一个 i 节点（节点编号 390162），硬链接与被链接的文件具有相同的文件类型标识位“-”，建立硬链接后，文件的链接数由 1 变为 2。

实际上硬链接只是源文件的一个硬复制，它们在目录文件中的入口项指向的是同一个 i 节点。只有当硬链接的全部链接被删除时才能够释放此 i 节点。任何对这个文件所做的修改，所有的硬链接都可以同步看到。硬链接的文件必须在同一个文件系统中，且目录不能建立硬链接。

建立符号链接可以使用带选项“-s”的 ln 命令，符号链接只是指定到真实文件的访问路径上，与源文件的 i 节点号不同。如果源文件被删除，符号链接就被损坏。符号链接的文件类型标识位为“l”。例如，为文件 ftpuser 建立符号链接 fuser，命令如下所示。

```
# ls -il ftpuser
390161 -rw-r--r-- 1 root root 0 07-26 01:17 ftpuser
# ln -s ftpuser fuser
# ls -il ftpuser fuser
390161 -rw-r--r-- 1 root root 0 07-26 01:17 ftpuser
390162 lrwxrwxrwx 1 root root 7 07-26 01:18 fuser -> ftpuser
# rm ftpuser
rm: 是否删除 一般空文件 “ftpuser” ? y
# ls -il ftpuser fuser
ls: ftpuser: 没有那个文件或目录
390162 lrwxrwxrwx 1 root root 7 07-26 01:18 fuser -> ftpuser
```

可以看到，ftpuser 与 fuser 的 i 节点号不同（ftpuser 为 390161，fuser 为 390162），fuser 的文件类型标识位为“l”，源文件 ftpuser 被删除后，符号链接文件报错。

与硬链接相比较，符号链接可以跨文件系统建立，并且可以指定到目录。硬链接与符号链接的关系如图 6-5 所示。

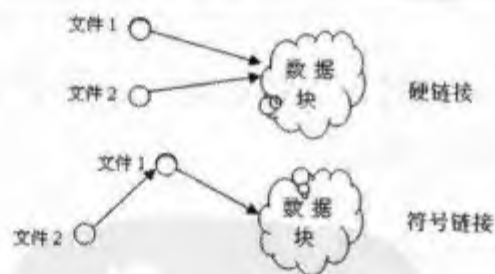


图 6-5 硬链接与符号链接



通过硬链接方式建立的文件类似于已经存在的文件的一个别名，当原文件被删除时，新建的文件仍然可用。通过符号链接建立的文件是以“路径”的形式表示另一个文件，类似于一种快捷方式。

例如，建立对 myfile 文件的硬链接 hardlink，命令如下所示。

```
$ ln myfile hardlink //建立硬链接
```

其中 hardlink 为指定的链接文件，它和文件 myfile 指向物理磁盘中的同一 i 节点，它们的各种属性相同，一个发生变化将引起另一个的变化。只有超级用户并且使用“-d”选项，



才能给目录建立硬链接。如果链接的源是文件而目的是已经存在的目录，则将在目的目录中生成一个链接文件。

例如，建立对 myfile 文件的符号链接 softlink，命令如下所示。

```
$ ln -s myfile softlink //建立符号链接
```

其中 softlink 为指定的链接文件，它以“路径”的形式表示已经存在的文件 myfile。如果文件位置发生变化，则相应的“路径”将发生变化，已建立的符号链接将失效。为了避免这种情况，一般会使用绝对路径建立符号链接。符号链接的目的可以是目录，其路径是相对路径。以“路径”形式表示的符号链接可以跨越磁盘分区，而硬链接不可以跨越磁盘分区。

6.11.19 ln -s 命令：建立到目录的符号链接

ln -s 命令可以把源目录下的文件和子目录，建立到目标目录的符号链接。如果没有指定目标目录，则在当前目录中建立相应的符号链接。该命令的语法格式如下所示。

```
ln -s [ -silent ] [ -ignorelinks ] [ -withrevinfo ] fromdir [ todir ]
```

其中各选项的具体说明如表 6-43 所示。

表 6-43 ln -s 命令的选项说明

选 项	说 明
-ignorelinks	直接建立各个符号链接之间的链接。不采用该选项时，一旦遇到符号链接，默认会维持该链接指向的原始文件。不建议使用该选项，该选项通常是模拟 X11R6 中 C 版本的 ln -s 命令
-silent	静默模式，不显示链接的建立过程
- withrevinfo	适于 RCS、SCCS、CVS 以及 CVS
fromdir	指定源目录
todir	指定目标目录



源目录必须使用绝对路径。

例如，将 test 目录中的文件逐一创建符号链接，并保存到 mytest 目录中。命令执行过程如下所示。

```
#ls //显示当前目录
test mytest
# ll test //显示 test 目录内容
总用量 2876
-rw-r--r-- 1 root root 1451172 8月 7 20:14 dict
-rw-r--r-- 1 root root 2 8月 7 14:01 fonts.dir
-rw-r--r-- 1 root root 289 8月 7 12:38 my.doc
-rw-r--r-- 1 root root 102 8月 7 16:45 mypaper
-rw-r--r-- 1 root root 102 8月 7 16:34 mypaper.bak
-rw-r--r-- 1 root root 102 8月 7 22:57 mypaper.doc
-rw-r--r-- 1 root root 126 8月 8 01:30 patch
-rw-r--r-- 1 root root 0 8月 8 01:31 path
-rw-r--r-- 1 root root 438 8月 7 23:23 xx00
-rw-r--r-- 1 root root 1450734 8月 7 23:23 xx01
# ll mytest
```



```
总用量 0
# ln -s /test /mytest //mytest 为空目录
# ll /mytest //建立链接
//显示 mytest 目录中的链接文件
总用量 0
lrwxrwxrwx 1 root root 10 8月 8 06:54 dict -> /test/dict
lrwxrwxrwx 1 root root 15 8月 8 06:54 fonts.dir -> /test/fonts.dir
lrwxrwxrwx 1 root root 12 8月 8 06:54 my.doc -> /test/my.doc
lrwxrwxrwx 1 root root 13 8月 8 06:54 mypaper -> /test/mypaper
lrwxrwxrwx 1 root root 17 8月 8 06:54 mypaper.bak -> /test/mypaper.bak
lrwxrwxrwx 1 root root 17 8月 8 06:54 mypaper.doc -> /test/mypaper.doc
lrwxrwxrwx 1 root root 11 8月 8 06:54 patch -> /test/patch
lrwxrwxrwx 1 root root 10 8月 8 06:54 path -> /test/path
lrwxrwxrwx 1 root root 10 8月 8 06:54 xx00 -> /test/xx00
lrwxrwxrwx 1 root root 10 8月 8 06:54 xx01 -> /test/xx01
```

6.11.20 lsattr 命令：显示文件属性

lsattr 命令用来显示文件的属性。该命令的语法格式如下所示。

```
lsattr [-R][-V][-a][-d][-v] [files...]
```

其中各选项的具体说明如表 6-44 所示。

表 6-44 lsattr 命令的选项说明

选 项	说 明
-a	显示所有的目录和文件，包括隐藏文件或目录
-R	递归显示指定目录及其子目录的内容
-V	显示版本信息
-v	显示版本号
files	指定的目录或文件。如果不指定，默认为当前目录

例如，通过命令 lsattr，显示当前目录下的文件属性，如下所示。

```
# lsattr
----- ./my.doc
----- ./fonts.dir
----- ./mypaper.bak
----- ./mypaper
----- ./dict
----- ./mypaper.doc
----- ./xx00
----- ./xx01
----- ./patch
----- ./path
```

6.11.21 mkfontdir 命令：创建 x 字体文件目录

mkfontdir 命令针对指定的目录搜寻所有的字体文件，并生成文件 fonts.dir，供 X 服务器和字体服务器使用。fonts.dir 也将被放置在指定的目录中。该命令的语法格式如下所示。

```
mkfontdir [-n] [-x suffix] [-p prefix] [-e encoding-directory-name] [--]
[directory-name ...]
```

其中各选项的具体说明如表 6-45 所示。



表 6-45 mkfontdir 命令的选项说明

选 项	说 明
directory-name	指定搜寻和生成文件的目录
-e encoding-directory-name	指定一个含有编码文件的目录
-n	不搜寻字体，也不写入字体目录文件，用于生成编码目录
-p prefix	指定编码文件生成路径的前缀
-x suffix	搜寻字体文件时忽略 suffix 类型的文件
--	结束选项

例如，在一个目录中通过搜寻字体文件创建文件 fonts.dir，命令如下所示。

```
# mkfontdir
# ls fonts.dir
fonts.dir
#cat fonts.dir
0
```

6.11.22 msgcat 命令：合并消息

msgcat 命令用于连接并合并指定的 PO 文件。在两个或多个指定的 PO 文件中找出共同的翻译消息。利用“--more-than”选项，可以令消息出现次数较多时才被打印。相反，“--less-than”选项则会令消息在出现较少时才打印（例如，“--less-than=2”表示打印只出现一次的消息）。该命令的语法格式如下所示。

```
msgcat[<,--less-than=NUMBER][>,--more-than=NUMBER][-D,--directory=DIRECTORY][-V,--version]
[-u,--unique][-f,--files-from=FILE][-o,--output-file=FILE][-t,--to-code=NAME][-e,--no-escape]
[-E,--escape][--force-po][--no-location][-n,--add-location][-w,--width=NUMBER][--no-wrap]
[-F,--sort-by-file][-h,--help]
```

msgcat 命令各选项的具体说明如表 6-46 所示。

表 6-46 msgcat 命令的选项说明

选 项	说 明
<, --less-than=NUMBER	输出少于指定数量的消息
>, --more-than=NUMBER	输出大于指定数量的消息
-D, --directory=DIRECTORY	在输入文件列表中添加指定的目录
-u, --unique	要求只列出出现一次的消息
-f, --files-from=FILE	从指定文件中读入输入文件列表
-o, --output-file=FILE	输出到指定的文件中
-t, --to-code=NAME	为输出进行编码
-e, --no-escape	不在输出中使用 C 转码序列
-E, --escape	使用逃逸字符

续表

选 项	说 明
--force-po	写 PO 文件，即使为空
--no-locatio	不写入包含 “#: filename:line” 的行
-n, --add-location	添加 “#: filename:line” 行
-w, --width=NUMBER	设置输出行
--no-wrap	不自动换行
-F, --sort-by-file	按文件位置进行排序
-h, --help	显示帮助信息
-V, --version	显示版本信息

6.11.23 msgcomm 命令：匹配消息

msgcomm 命令在两个或多个指定 PO 文件中找出共同的信息。利用 “--more-than” 选项，可以令消息在出现次数较多时才会被输出。相反地，“--less-than” 选项则会令消息在出现次数较少时才被输出。该命令的语法格式如下所示。

```
msgcat [-<, --less-than=NUMBER] [->, --more-than=NUMBER] [--use-first] [-D, --directory=DIRECTORY]
[-u, --unique] [-f, --files-from=FILE] [-o, --output-file=FILE] [-e, --no-escape] [--omit-header]
[-E, --escape] [--force-po] [--no-location] [-n, --add-location] [-w, --width=NUMBER] [--no-wrap]
[-F, --sort-by-file] [-h, --help] [-V, --version]
```

其中各选项的具体说明如表 6-47 所示。

表 6-47 msgcomm 命令的选项说明

选 项	说 明
-f, --files-from=FILE	由指定文件读入输入文件的列表
-D, --directory=DIRECTORY	在指定目录中查找输入文件
-o, --output-file=FILE	将输出写入指定文件
->, --more-than=NUMBER	只打印出现次数多于指定次数的消息，若没有指定次数则默认值为 1
-<, --less-than=NUMBER	只打印出现次数少于指定次数的消息，若没有指定次数则默认值为 1
-u, --unique	要求只列出出现一次的消息
-stringtable-input	输入文件以 NeXTstep/GNUstep.string 语法给出
-e, --no-escape	不在输出中使用 C 转码序列
-E, --escape	在输出中使用 C 转码序列，没有扩展字符
--force-po	就算为空也写入 PO 文件
-i, --indent	使用缩进风格写入 PO 文件
--no-location	不写入#、文件名、行号位置行
-n, --add-location	生成#、文件名、行号位置行



续表

选 项	说 明
--strict	在写入 PO 文件时保持严格统一格式
--stringtable-output	写出 NeXTstep/GNUstep.strings 文件
-w, --width=NUMBER	指定输出页面宽度
--no-wrap	不将超过页面输出宽度的长消息分为多行，即自动换行
-s, --sort-output	输出前排列
-F, --sort-by-file	按文件位置排序输出
--omit-header	不写入带有 msgid 项的文件头
--help	帮助信息
--version	版本信息

6.11.24 msgen 命令：创建英语消息目录

msgen 命令用于创建一个英文翻译目录。输入文件名是最近创建的英文 PO 文件或 PO 模板文件（通常由 xgettext 创建）。未翻译的记录会被填上和 msgid 一样的字符串，并标记为模糊。该命令的语法格式如下所示。

```
msgen[-D,--directory=DIRECTORY][-F,--sort-by-file][--help][--version][--omit-header]
[--no-wrap]
[-o,--output-file=FILE][-e,--no-escape][-i,--indent][--strict][-s,--sort-output][-w,
--width=NUMBER]
[-E,--escape][--force-po][--no-location][--add-location]
```

其中各选项的具体说明如表 6-48 所示。

表 6-48 msgen 命令的选项说明

选 项	说 明
-D, --directory=DIRECTORY	在指定目录中查找输入文件
-o, --output-file=FILE	将输出写入指定文件
-e, --no-escape	不在输出中使用 C 转码序列
-E, --escape	在输出中使用 C 转码序列，没有扩展字符
--force-po	就算为空也写入 PO 文件
-i, --indent	使用缩进风格写入 PO 文件
--no-location	不写入#、文件名、行号位置行
--add-location	生成#、文件名、行号位置行
--strict	在写入 PO 文件时保持严格统一格式
-w, --width=NUMBER	指定输出页面宽度
--no-wrap	不将超过页面输出宽度的长消息分为多行，即自动换行
-s, --sort-output	输出前排列
-F, --sort-by-file	按文件位置排序输出
--help	显示帮助信息
--version	显示版本信息

6.11.25 msginit 命令：初始化消息目录

创建新的 PO 文件，即根据用户环境变量取值初始化元信息。该命令的语法格式如下所示。

```
msginit[-i,--input=INPUTFILE][-o,--output-file=FILE][-stringtable-input][-l,--locale=LL_CC][--no-translator]
      [--stringtable-output][-w,--width=NUMBER][--no-wrap][-h,--help]
```

其中各选项的具体说明如表 6-49 所示。

表 6-49 msginit 命令的选项说明

选 项	说 明
-i, --input=INPUTFILE	指定输入的 POT 文件
-o, --output-file=FILE	指定输出文件
-stringtable-input	输入文件以 NeXTstep/GNUstep.string 语法给出
-l, --locale=LL_CC	设置目标语言
--no-translator	假定 PO 文件自动生成
--stringtable-output	写出 NeXTstep/GNUstep.strings 文件
-w, --width=NUMBER	指定输出页面宽度
--no-wrap	不将超过页面输出宽度的长消息分为多行
-h, --help	显示帮助并退出
-V, --version	输出版本信息并退出

6.11.26 mkdir 命令：创建目录

mkdir 命令用于创建目录，其语法格式如下所示。

```
mkdir [-m, --mode=模式][ -p, --parents][ -v, --verbose][ --help][ --version]
```

其中各选项的具体说明如表 6-50 所示。

表 6-50 mkdir 命令的选项说明

选 项	说 明
-m, --mode=模式	设定权限<模式>（类似 chmod），而不是 rwxrwxrwx 减 umask
-p, --parents	需要时创建上层目录，如目录早已存在则不当作错误。如果目录名中的路径中包含着不存在的子目录，那么就逐一地建立，直到最后的子目录为止
-v, --verbose	每次创建新目录都显示信息
--help	显示帮助信息并离开
--version	显示版本信息并离开

创建目录时，如果目录名前没有指定目录的路径，那么就表示在当前目录下创建；如果有路径名，则在指定的路径下建立。新建的子目录必须不能和已经存在的文件名或目录名重名，如下所示。

```
# pwd //在当前目录下创建子目录 zhang
/home/teacher1
# ls
# mkdir zhang
```



```
# ls
zhang
# mkdir /home/teacher1/yang //使用绝对路径创建子目录 yang
# ls
yang zhang
# mkdir zhang //不能重名地创建
mkdir: 无法创建目录 “zhang”: 文件已存在
```

例如，创建一个新目录 my，命令如下所示。

```
$ ls
$ mkdir new
$ ls
my
```

例如，创建一个新目录 my 且指定该目录的属性为对所有用户可读，命令如下所示。

```
$ mkdir -m a=r my
$ ls -l
总用量 8
dr--r--r-- 2 yang yang 4096 6月25 20 02:15 my
```

在创建子目录的时候，如果子目录的父目录不存在，则无法创建。使用选项“-p”，可以逐级创建目录。例如，在当前目录下创建 li/document 目录，命令如下所示。

```
# pwd
/home/teacher1
#mkdir li/document //由于不存在 li 子目录，li/document 子目录无法创建
mkdir: 无法创建目录 “li/document”: 没有那个文件或目录
# mkdir -p li/document //参数“-p”允许逐级创建目录
# ls
li yang zhang
# cd li
# ls
document
```

例如，创建新目录/usr/yang/example，且指定其权限为 700，命令如下所示。

```
$ mkdir -m 700 /usr/Bob/example
```

6.11.27 mv 命令：移动或更名现有的文件或目录

mv 命令用于移动或重命名指定文件。mv 与 cp 命令明显的不同之处在于：mv 命令是移动文件，文件个数没增加；cp 命令是复制文件，文件个数增加。此外，mv 命令还可以进行文件或目录的改名。mv 命令的语法格式如下所示。

```
mv[--backup[=CONTROL]][-b][-f,--force][-i,--interactive][--reply={yes,no,query}][--strip-trailing-slashes]
[-S,--suffix= 后缀 ][--target-directory= 目录 ][-u,--update][-v, --verbose][--help]
[--version]src dest
```

其中各选项的具体说明如表 6-51 所示。

表 6-51 mv 命令的选项说明

选 项	说 明
--backup[=CONTROL]	为每个已存在的目标文件创建备份文件
-b	类似--backup，但不接受任何参数
-f, --force	覆盖文件前不会进行确认，等于--reply=yes。如果操作要覆盖某个已有的目标文件，不给任何指示

续表

选 项	说 明
-i, --interactive	覆盖文件前必须先确认，等于--reply=query。交互式的操作，如果操作要覆盖某个已有的目标文件，会询问用户是否覆盖
--reply={yes,no,query}	指定如何处理已存在的目标文件
--strip-trailing-slashes	删除参数中所有<来源>文件/目录末端的斜杠
-S, --suffix=后缀	自行指定备份文件的<后缀>
--target-directory=目录	将所有参数指定的<来源>文件/目录移动至<目录>
-u, --update	只在<来源>文件比目标文件新，或目标文件不存在时才会移动
-v, --verbose	详细显示处理的步骤
--help	显示帮助信息并离开
--version	显示版本信息并离开
src	源文件或目录
dest	目标文件或目录

mv 命令使用格式说明如表 6-52 所示。

表 6-52 mv 命令格式说明

命 令 格 式	结 果
mv 文件名 文件名	将源文件名改为目标文件名
mv 文件名 目录名	将文件移动到目标目录
mv 目录名 目录名	目标目录已存在，源目录移动到目标目录，目标目录不存在，改名
mv 目录名 文件名	出错

例如，将 m1.c 文件改名为 m2.c，命令如下所示。

```
$ mv m1.c m.c
```

例如，将 /usr/student 下的所有文件和目录移到当前目录下，命令如下所示。

```
$ mv /usr/student/* .
```

例如，将文件 new 移入 yang 文件夹内，命令如下所示。

```
# ls
teacher new newfile test ws yang
# mv new yang
# ls
teacher newfile test ws yang
# cd yang
# ls
mydir newdoc new test
```

例如，将文件 yy 更名为 mm，命令如下所示。

```
# ls
new newfile test testiso yy
#
# mv yy mm
# ls
new newfile test testiso mm
```



6.11.28 mattrib 命令：修改或显示 MS-DOS 文件的属性

mattrib 命令是 mtools 工具命令集中的命令。该命令模拟 MS-DOS 的 attrib 命令，用于修改或显示 MS-DOS 文件的属性，其语法格式如下所示。

```
mattrib [-a|+a] [-h|+h] [-r|+r] [-s|+s] [-/] [-p] [-X] msdosfile [msdosfiles ...]
```

其中各选项的具体说明如表 6-53 所示。

表 6-53 mattrib 命令的选项说明

选 项	说 明
+/-	(+) 代表打开指定的文件属性，“-”反之
a	归档位，被备份程序用来标识一个文件是不是新文件
r	只读位，用来标识一个只读文件
s	系统位，用来标识一个操作系统文件
h	隐藏位，用来使一个文件在目录中不可见
/	列出包含所有子目录下的文件属性
X	简明方式，列出属性，不含任何空格
-p	重放模式
--help	列出帮助信息
--version	列出版本信息
msdosfiles	指定的 MS-DOS 文件

6.11.29 mc 命令：可视化文件管理程序

mc 命令提供了一个交互式的可视化文件管理程序，具有图形化界面。该命令的语法格式如下所示。

```
mc [-a][-b][-c][-C arg][-f][-k][ --help][ --version]
```

其中各选项的具体说明如表 6-54 所示。

表 6-54 mc 命令的选项说明

选 项	说 明
-a	在行绘制时，关闭图形字符的使用
-C arg	通过命令行指定一个不同的颜色集
-f	显示编译搜索路径
-d	不支持鼠标
-b	黑白显示模式
-c	彩色显示模式
-k	重置热键到默认值
--help	显示帮助信息
--version	显示版本信息

例如，使用 `mc` 对当前目录文件进行管理，执行命令如下所示。

```
#mc
```

执行该命令后，将进入交互式界面，如图 6-6 所示。由于在命令行中没有指定打开的路径，默认为当前目录。



图 6-6 mc 命令的图形界面

例如，如果用户希望对 `/etc` 目录进行管理，可以在 `mc` 命令行中指定查看的目录，命令如下所示。

```
#mc /etc
```

执行该命令后，将进入交互式界面。由于在命令行中指定了打开的路径 `/etc`，所以 `mc` 显示了 `/etc` 目录中的内容，如图 6-7 所示。



图 6-7 mc 命令打开/etc 目录



6.11.30 mcopy 命令：复制 MS-DOS 文件到 UNIX 或从 UNIX 复制 MS-DOS 文件

mcopy 命令是 mtools 工具集中的命令，它用来复制 MS-DOS 文件到 UNIX，或者从 UNIX 复制 MS-DOS 文件。该命令的语法格式如下所示。

```
mcopy [-b][-s][-P][-a][-n][-v][-m][-Q][-T] sourcefile targetfile
mcopy[-b][-s][-P][-a][-n][-v][-m][-Q][-T]sourcefile[sourcefiles...]targetdirectory
mcopy [-t][-n][-v][-m] sourcefile
```

其中各选项的具体说明如表 6-55 所示。

表 6-55 mcopy 命令的选项说明

选 项	说 明
-t	文本文件传输
-b	批处理模式
-s	递归拷贝目录下的子目录的文件
-P	保留复制文件的属性
-a	文本（ASCII）文件传输
-n	覆盖时不进行提示
-m	保留文件的修改时间
-v	冗余模式，显示详细的处理信息
-Q	当复制多个文件时，只要有一个文件复制失败即退出
-T	具有字符集转换的文本传输模式
--help	显示帮助信息
--version	显示版本信息
sourcefile	源文件
targetfile	目标文件

例如，通过命令 mcopy 复制当前目录下的 myproc.c 文件到软盘，命令如下所示。

```
# mcopy myproc.c /dev/fd0 //复制文件
```

6.11.31 md5sum 命令：计算和检测 MD5 信息摘要

md5sum 命令可以计算文件的 MD5 值，并且可以通过 MD5 值检验文件。该命令的语法格式如下所示。

```
md5sum[-c,--check][-b,--binary][-t,--text][--status][--help][--version][-w,--warn]
```

其中各选项的具体说明如表 6-56 所示。

表 6-56 md5sum 命令的选项说明

选 项	说 明
-c, --check	根据指定的列表，检验 MD5
-b, --binary	以二进制的模式读取文件，默认为 MS-DOS/Windows
-t, --text	以文本的方式读取文件

续表

选 项	说 明
--status	除显示状态码外，不进行任何输出
--help	帮助信息
--version	版本信息
-w, --warn	不正确时发出警告



MD5 值是一种哈希值。哈希函数以文件内容为依据，生成一个统一位数的哈希值。

例如，计算当前目录下的文件 dict 和 fonts.dir 的 MD5 值，命令如下所示。

```
# ls
dict    fonts.dir
# md5sum dict fonts.dir           //计算文件 dict、fonts.dir 的 MD5 值
46ce4ea7b689b314373688b835c460ba dict
897316929176464ebc9ad085f31e7284 fonts.dir
```

例如，比较文件 dict 和 /home/dict 是否相同，命令如下所示。

```
# md5sum /dict                   //计算文件 dict 的 MD5 值
46ce4ea7b689b314373688b835c460ba /dict
# md5sum /home/dict              //计算文件 /home/dict 的 MD5 值
46ce4ea7b689b314373688b835c460ba /home/dict
```

从输出结果可以看到，这两个文件的 MD5 值相同。例如，使用 “--check” 选项，检查一组文件中某些文件是否发生了改变，命令如下所示。

```
# md5sum dict my.doc patch > sum //计算文件 dict、my.doc 和 patch 的 MD5 值
# md5sum --check sum
dict: OK
my.doc: OK
patch: OK
# echo "123456" >patch           //改变文件 patch 的内容
# md5sum --check sum             //检验文件的内容是否发生变化
dict: OK
my.doc: OK
patch: FAILED
md5sum: WARNING: 1 of 3 computed checksums did NOT match
```

可以看到，通过 md5sum 命令将文件 dict、my.doc 和 patch 的哈希值存到文件 sum 中。利用 “md5sum-check” 命令检查文件 dict、my.doc 和 patch 中的内容有没有变化。第一次检查结果是文件都没有发生变化。第二次利用 echo 命令，修改了文件 patch 的值，然后再次利用 “md5sum -check” 命令进行检验，发现文件 patch 发生了变化。

6.11.32 mdel 命令：MS-DOS 文件删除

mdel 命令是 mtools 工具集中的命令，它用来删除 MS-DOS 文件。该命令的语法格式如下所示。

```
mdel [-v] msdosfile [ msdosfiles ... ]
```

其中各选项的具体说明如表 6-57 所示。



表 6-57 mdel 命令的选项说明

选 项	说 明
-v	显示 mdel 命令的版本信息
msdosfile	指定删除的文件

例如，删除 MS-DOS 文件 dosfile，命令如下所示。

```
#mdel dosfile //删除 MSDOS 文件
```

6.11.33 mdir 命令：显示 MS-DOS 目录

mdir 命令是 mtools 工具命令集中的命令。该命令是模拟 MS-DOS 的 dir 命令，用来显示目录内容，其语法格式如下所示。

```
mdir [-/] [-f] [-w] [-a] [-b] msdosdir [ msdosdir...]
```

其中各选项的具体说明如表 6-58 所示。

表 6-58 mdir 命令的选项说明

选 项	说 明
-/	递归显示目录及子目录中的内容
-a	显示所有文件，包括隐藏文件
-w	横排模式。只显示文件名，不显示文件长度、时间等信息
-f	快速模式，不计算空闲空间。由于计算空闲空间需要读入 FAT 表并对磁盘进行扫描，对于大容量磁盘所花费的时间也不容忽视。对于 FAT32 文件系统不需要该选项标志
-b	简洁模式，每行显示一个文件名或目录名
msdosfiles	指定显示的 MS-DOS 目录

例如，显示 MS-DOS 目录，包含隐藏文件，命令如下所示。

```
#mdir -a dosdir //显示 DOS 目录
```

6.11.34 mktmp 命令：建立暂存文件

mktmp 命令根据指定的名称创建一个暂存文件，该文件可供 Shell 脚本安全地使用。该命令的语法格式如下所示。

```
mktmp [-d] [-q] [-u]
```

其中各选项的具体说明如表 6-59 所示。

表 6-59 mktmp 命令的选项说明

选 项	说 明
-d	创建一个目录，而不是文件
-u	非安全模式。临时文件在 mktmp 退出时将取消链接
-q	静默模式，发生错误时不显示错误信息

6.11.35 mmove 命令：移动 MS-DOS 文件

mmove 命令是 mtools 工具命令集中的命令。该命令模拟 MS-DOS 下的 move 命令，用来

移动 MS-DOS 文件或者给 MS-DOS 文件改名，其语法格式如下所示。

```
mmove [-vV] file targetfile
      [-vV] file [files...] target_directory
```

其中各选项的具体说明如表 6-60 所示。

表 6-60 mmove 命令的选项说明

选 项	说 明
-v	显示 mmove 版本信息及用法摘要
-V	显示版本信息
targetfile	目标文件
file	源文件
target_directory	目标目录



与 MS-DOS 下的 move 命令不同，mmove 命令可以移动子目录。

例如，通过命令 mmove，将 MS-DOS 文件 file1 改名为 file2，命令如下所示。

```
#mmove file1 file2 //将 MS-DOS 文件改名
```

6.11.36 mren 命令：更改 MS-DOS 文件名称

mren 命令是 mtools 工具命令集中的命令，该命令模拟 MS-DOS 下的 rename 命令，用来更改 MS-DOS 文件名称，其语法格式如下所示。

```
mren [-vV] file targetfile
     [-vV] file [files...] target_directory
```

其中各选项的具体说明如表 6-61 所示。

表 6-61 mmove 命令的选项说明

选 项	说 明
-v	显示 mren 命令版本信息及用法摘要
-V	显示版本信息
targetfile	目标文件
file	源文件
target_directory	目标目录

例如，通过命令 mren，更改 MS-DOS 文件 dosfile1 的名称为 dosfile2，命令如下所示。

```
#mren dosfile1 dosfile2 //更改文件名称
```

6.11.37 mshowfat 命令：显示分配给文件的 FAT 簇信息

mshowfat 命令是 mtools 工具命令集中的命令，它用来显示 MS-DOS 文件的记录。例如，通过命令 mshowfat，显示 MS-DOS 文件 msdoc.doc 文件的 FAT 簇信息，命令如下所示。

```
#mshowfat msdos.doc //显示 MS-DOS 文件记录
```



6.11.38 mtools 命令：显示 mtools 支持的命令

mtools 命令用于显示 mtools 支持的命令集，该命令集中的命令是对 MS-DOS 下对应命令的模拟，用户可以像在 MS-DOS 系统下一样使用这些命令。这将极大地方便使用 MS-DOS 系统的用户。

例如，在命令行下直接执行 mtools 命令，可以看到执行结果如下所示。

```
# mtools //显示 mtools 支持的命令
Supported commands:
mattrib, mbadblocks, mcat, mcd, mcopy, mdel, mdeltree, mdir
mdoctorfat, mdu, mformat, minfo, mlabel, mmd, mmount, mpartition
mrd, mread, mmove, mren, mshowfat, mtoolstest, mtype, mwrite
mzip
```

6.11.39 mtoolstest 命令：测试并显示 mtools 的相关设置

mtoolstest 命令测试并显示 mtools 的相关设置。例如，在 Red Hat Linux 9 下测试并显示 mtools，命令如下所示。

```
# mtoolstest
drive J:
    #fn=0 mode=0 builtin
    file="/dev/sdb4" fat_bits=16
    tracks=0 heads=0 sectors=0 hidden=0
    offset=0x0
    partition=0
    mformat_only

... ..
mtools_fat_compatibility=0 //FAT 兼容设置
mtools_skip_check=0
mtools_lower_case=0 //小写设置
```

6.11.40 paste 命令：对文件进行纵向合并

paste 命令合并指定文件的列，然后将内容输出到标准输出。如果没有指定文件，将从标准输入设备读入内容。该命令的语法格式如下所示。

```
paste [-s, --serial][ -d, --delimiters=LIST][ --help][ --version]
```

其中各选项的具体说明如表 6-62 所示。

表 6-62 paste 命令的选项说明

选 项	说 明
-s, --serial	一次处理一个文件
-d, --delimiters=LIST	指定分隔符
--help	显示 paste 命令的帮助信息
--version	显示 paste 命令的版本信息

例如，通过命令 paste，合并文件 mypaper 和 mydoc，如下所示。

```
# cat mypaper //显示文件 mypaper 的内容
hello
```



```
This is a test paper.
I liko Linux.
It im a very perfecr operating system.
we are Linux lovers.
# cat mydoc //显示文件 mydoc 的内容
0000000000000000
1111111111111111
2222222222222222
3333333333333333
4444444444444444
5555555555555555
# paste mypaper mydoc //按先 mypaper 后 mydoc 合并文件
hello 0000000000000000
This is a test paper. 1111111111111111
I liko Linux. 2222222222222222
It im a very perfecr operating system. 3333333333333333
we are Linux lovers. 4444444444444444
5555555555555555
# paste mydoc mypaper //按先 mydoc 后 mypaper 合并文件
0000000000000000 hello
1111111111111111 This is a test paper.
2222222222222222 I liko Linux.
3333333333333333 It im a very perfecr operating system.
4444444444444444 we are Linux lovers.
5555555555555555
```

例如，通过命令 `paste`，在合并文件 `mydoc` 和 `mypaper` 时，指定两者之间的分隔字符为“#”，如下所示。

```
# paste -d# mydoc mypaper //合并文件的行，并指定间隔符
0000000000000000#hello
1111111111111111#This is a test paper.
2222222222222222#I liko Linux.
3333333333333333#It im a very perfecr operating system.
4444444444444444#we are Linux lovers.
5555555555555555#
```

6.11.41 rcp 命令：远端复制文件或目录

`rcp` 命令用于在主机之间复制文件。该命令的语法格式如下所示。

```
rcp [-p] [-x] [-D port] [-N] [-r]
```

其中各选项的具体说明如表 6-63 所示。

表 6-63 rcp 命令的选项说明

选 项	说 明
-p	复制时保留源文件的修改时间等属性，忽略掩码
-x	加密传输的所有数据
-D port	指定远端主机的端口号
-r	如果指定的源文件为目录，同时复制目录下的文件和子目录下的所有文件
-N	即使在本机复制文件也使用网络连接

例如，将远端主机 192.168.20.122 中 `/home/src/patch` 文件复制到 `/home/src` 目录中，命令



如下所示。

```
# rcp teacher@192.168.20.122: /home/src/patch /home/src //复制文件
```

6.11.42 rm 命令：删除文件或目录

rm 命令用于删除文件或目录，可删除一个目录中的一个或多个文件或目录，也可删除某个目录及其下面的所有文件和子目录。对于链接文件，只是删除该链接，原文件保持不变。rm 命令的语法格式如下所示。

```
rm[-d,--directory][-f,--force][-i,--interactive][-r,-R,--recursive][-v,--verbose][-help][--version]
```

其中各选项的具体说明如表 6-64 所示。

表 6-64 rm 命令的选项说明

选 项	说 明
-d, --directory	删除可能仍有数据的目录（只限超级用户）
-f, --force	略过不存在的文件，不显示任何信息。在删除过程中不给任何指示，直接删除
-i, --interactive	进行任何删除操作前必须先确认。与-f选项相反，为交互式删除，每个文件在删除时都给出提示
-r, -R, --recursive	将参数中列出的全部目录和子目录都递归地删除
-v, --verbose	详细显示处理的步骤
--help	显示帮助信息并离开
--version	显示版本信息并离开

例如，删除文件当前目录下的文件 ls，命令如下所示。

```
# ls -al
总用量 120
drwxr-xr-x  2 root  root    4096  7月 30 11:33 .
drwxr-x--- 25 root  root    4096  8月  3 11:49 ..
-rwxr-xr-x  1 root  root   16368  6月  2 22:42 bubbleSort
-rw-r--r--  1 root  root    232  6月  2 22:42 bubbleSort.bb
-rw-r--r--  1 root  root    380  6月  2 22:42 bubbleSort.bbg
-rw-r--r--  1 root  root    772  6月  2 22:42 bubbleSort.c
-rw-r--r--  1 root  root   1080  6月  2 23:24 bubbleSort.c.gcov
-rw-r--r--  1 root  root     80  6月  2 22:50 bubbleSort.da
-rw-r--r--  1 root  root     85  6月  5 19:36 kk2.c
-rwxr-xr-x  1 root  root   67668  7月 30 11:33 ls      //可执行文件 ls
# rm ls                                     //删除可执行文件 ls
rm: 是否删除一般文件 'ls' ? y
# ls -al
总用量 48
drwxr-xr-x  2 root  root    4096  8月  3 11:53 .
drwxr-x--- 25 root  root    4096  8月  3 11:49 ..
-rwxr-xr-x  1 root  root   16368  6月  2 22:42 bubbleSort
-rw-r--r--  1 root  root    232  6月  2 22:42 bubbleSort.bb
-rw-r--r--  1 root  root    380  6月  2 22:42 bubbleSort.bbg
-rw-r--r--  1 root  root    772  6月  2 22:42 bubbleSort.c
-rw-r--r--  1 root  root   1080  6月  2 23:24 bubbleSort.c.gcov
-rw-r--r--  1 root  root     80  6月  2 22:50 bubbleSort.da
-rw-r--r--  1 root  root     85  6月  5 19:36 kk2.c
#
```


删除文件可以直接使用 `rm` 命令，若删除目录则必须配合选项 “`-r`”，如下所示。

```
# rm pp.c
rm: 是否删除 一般文件 “pp.c”? y
# rm homework
rm: 无法删除目录 “homework”: 是一个目录
# rm -r homework
rm: 是否删除 目录 “homework”? y
```

例如，利用通配符删除多个文件，命令如下所示。

```
# ls -al
总用量 52
drwxr-xr-x  3 root  root    4096  8月  3 11:59 .
drwxr-x--- 25 root  root    4096  8月  3 11:49 ..
-rwxr-xr-x  1 root  root   16368  6月  2 22:42 bubbleSort
-rw-r--r--  1 root  root    232  6月  2 22:42 bubbleSort.bb
-rw-r--r--  1 root  root    380  6月  2 22:42 bubbleSort.bbg
-rw-r--r--  1 root  root    772  6月  2 22:42 bubbleSort.c
-rw-r--r--  1 root  root   1080  6月  2 23:24 bubbleSort.c.gcov
-rw-r--r--  1 root  root     80  6月  2 22:50 bubbleSort.da
-rw-r--r--  1 root  root     85  6月  5 19:36 kk2.c
drwxr-xr-x  2 root  root    4096  8月  3 11:59 test

# rm bubbleSort.c*
rm: 是否删除一般文件 ‘bubbleSort.c’? y
rm: 是否删除一般文件 ‘bubbleSort.c.gcov’? y
# ls -al
总用量 44
drwxr-xr-x  3 root  root    4096  8月  3 12:03 .
drwxr-x--- 25 root  root    4096  8月  3 11:49 ..
-rwxr-xr-x  1 root  root   16368  6月  2 22:42 bubbleSort
-rw-r--r--  1 root  root    232  6月  2 22:42 bubbleSort.bb
-rw-r--r--  1 root  root    380  6月  2 22:42 bubbleSort.bbg
-rw-r--r--  1 root  root     80  6月  2 22:50 bubbleSort.da
-rw-r--r--  1 root  root     85  6月  5 19:36 kk2.c
drwxr-xr-x  2 root  root    4096  8月  3 11:59 test
```

例如，删除当前目录下的所有文件及目录，命令如下所示。

```
# rm -r *
```

6.11.43 rmdir 命令：删除目录

`rmdir` 命令用于删除一个空目录，被删除的目录必须是一个空目录，否则无法删除，将出现错误。`rmdir` 只用于删除目录，无法删除文件。`rmdir` 命令的语法格式如下所示。

```
rmdir [选项] 目录名
```

其中各选项的具体说明如表 6-65 所示。

表 6-65 rmdir 命令的选项说明

选 项	说 明
--ignore-fail-on-non-empty	忽略任何因目录仍有数据而造成的错误
-p	删除目录，然后尝试删除指定路径中的所有上层命令。删除目录下所有的空目录，如果有非空的子目录，则保留下来；如果所有的子目录都被删除了，则删除该目录。例如，“ <code>rmdir -p a/b/c</code> ”的效果等于“ <code>rmdir a/b/c; rmdir a/b; rmdir a</code> ”
-v	运行时，显示详细的处理信息，即处理每个目录时都显示信息
--help	显示帮助信息
--version	显示版本信息



例如，删除 document 子目录，命令如下所示。

```
# ls
document
# cd document
# ls
# rmdir document      //在 document 子目录内无法删除 document 子目录
rmdir: document: 没有那个文件或目录
# cd ..
# rmdir document
# ls
```

若没有 document 的写权限或目录不空则无法删除，如下所示。

```
# ls
li yang zhang
# rmdir li
rmdir: li: 目录非空
```

6.11.44 slocate 命令：安全查找文件或目录

slocate 命令是一种安全查找文件或目录的工具。slocate 命令自己建立一个数据库，直接在自己的数据库里进行查找，极大地提高了查找速度。该命令的语法格式如下所示。

```
slocate[-u][-U<dir>][-e<dir1,dir2,...>][-f<fstype1,...>][-c][-l<level>][-i][-q][-n<num>][-r<regexp>]
[-o<file>][-d<path>][-h][-v][-V]
```

其中各选项的具体说明如表 6-66 所示。

表 6-66 slocate 命令的选项说明

选 项	说 明
-u	创建 slocate 数据库起始路径为根
-U <dir>	在指定目录创建 slocate 数据库
-e <dir1,dir2,...>	从 slocate 数据库中去除指定的路径
-f <fstype1,...>	从 slocate 数据库中去除指定文件系统中的文件
-c	当升级时，解析“/etc/updatedb.conf”文件
-l <level>	安全等级，有以下两值。 0：关闭安全检测，会使搜索更快。 1：启动安全检测，默认设置
-i	搜索时大小写不敏感
-q	静默模式，不输出错误信息
-n <num>	限制显示结果的数目
-r <regexp>	使用基本的 POSIX 正则表达式搜索数据库
-o <file>	指定创建的数据库
-d <path>	指定数据库搜索的路径
-h	显示帮助信息
-v	冗余模式，当创建数据库时显示文件
-V	显示版本信息

例如，通过命令 `slocate`，查找含有字符串 `linux` 的文件，如下所示。

```
# slocate linux //查找文件
/boot/vmlinux-2.4.20-8
/var/www/manual/mod/mod_php4/de/install.linux.html
/var/www/manual/mod/mod_php4/fr/install.linux.html
/var/www/manual/mod/mod_php4/it/install.linux.html
/var/www/manual/mod/mod_php4/ja/install.linux.html
/var/www/manual/mod/mod_php4/ko/install.linux.html
/var/www/manual/mod/mod_php4/install.linux.html
/var/www/manual/mod/mod_php4/pt_BR/install.linux.html
/etc/makedev.d/linux-2.4.x
... ..
```

6.11.45 split 命令：切割文件

利用 `cat` 命令可以将多个文件的内容合并到一个文件中。与之相反，`split` 命令将指定的文件切割成小的文件。该命令的语法格式如下所示。

```
split[-d][-v][-b,--bytes=SIZE][-l,--lines=NUMBER][-C,--line-bytes=SIZE][--help][--version]
```

其中各选项的具体说明如表 6-67 所示。

表 6-67 split 命令的选项说明

选 项	说 明
-v	运行时显示详细的处理信息
-b, --bytes=SIZE	每个分割文件的大小
-l, --lines=NUMBER	每个文件包含的行数。将输入文件每 n 行拆分为一个文件，默认值为 1000
-C, --line-bytes=SIZE	每个文件包含的字节数。以字节为单位进行拆分，设定每个拆分后的文件的大小：b 代表 512KB，k 代表 1KB，m 代表 1MB
-	从标准输入读取数据
--help	帮助信息
--version	版本信息

例如，对文件 `file` 进行拆分，每两行存为一个新文件，新文件名以“`files`”为前缀，命令如下所示。

```
# ls
file
# cat file
aaaaaaaaaaaaaaaa
bbbbbbbbbbbbbbbb
cccccccccccccccc
dddddddddddddddd
eeeeeeeeeeeeeeee
ffffffffffffffff
# split -l 2 file files
```

可以看到，文件 `file` 被拆分成了 `filesaa`、`filesab`、`filesac` 三个文件，如下所示。

```
# ls
file filesaa filesab filesac
```

查看文件 `filesaa`、`filesab` 和 `filesac`，可以看到每个文件包含 2 行，如下所示。



例如，以指定行数分割文件。将文件 my.doc 进行分割，每个文件最多有 5 行，命令如下。

使用选项“-b”可以按大小对文件进行拆分。例如，文件 file 为 90 字节，按每 30 字节一个文件进行拆分，新拆分后的文件以“filebysize”为前缀，如下所示。

可以看到，文件 file 被拆分成了 filebysizeaa、filebysizeab、filebysizeac 三个文件，如下所示。

其中每个文件各为 30 字节，如下所示。

例如，通过命令 `split`，按照字节数分割文件 `my.doc`，每个文件的大小为 30 字节，如下所示。


```

-rw-r--r-- 1 root root 50 8月 7 12:45 xaa
-rw-r--r-- 1 root root 50 8月 7 12:45 xab
-rw-r--r-- 1 root root 50 8月 7 12:45 xac
-rw-r--r-- 1 root root 50 8月 7 12:45 xad
-rw-r--r-- 1 root root 50 8月 7 12:45 xae
-rw-r--r-- 1 root root 39 8月 7 12:45 xaf

```

使用选项“-”可以从标准输入设备读入数据，并按设定值进行拆分。例如，从标准键盘读入数据，每2行拆分成一个新文件，且新文件的前缀为“standinput”，如下所示。

```

# split -l 2 - standinput
aaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaa
bbbbbbbbbbbbbbbbbbbbbbbbbb
bbbbbbbbbbbbbbbbbbbbbbbbbb
cccccccccccccccccccccccccc
cccccccccccccccccccccccccc

```

按 Ctrl+d 或 Ctrl+c 组合键结束文件的输入。使用命令 ls 查看读入的数据是否被拆分，如下所示。

```

# ls
standinputaa  standinputab  standinputac

```

查看文件 standinputaa、standinputab 和 standinputac，可以看到每个文件包含2行，如下所示。

```

# more standinputa?
::::::::::::
standinputaa
::::::::::::
aaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaa
::::::::::::
standinputab
::::::::::::
bbbbbbbbbbbbbbbbbbbbbbbbbb
bbbbbbbbbbbbbbbbbbbbbbbbbb
::::::::::::
standinputac
::::::::::::
cccccccccccccccccccccccccc
cccccccccccccccccccccccccc

```

可以使用命名管道对标准输入进行重新指定，如下所示。

```

# ls -al /etc | split -l 50 - etcfile
# ls
etcfileaa etcfilead etcfileab etcfileae etcfileac etcfileaf

```

该命令以 ls 长格式列出/etc 目录下的所有文件，并把命令 ls 的输出结果通过命名管道输入 split 命令。由 split 命令按着每50行另存为一个新文件，对输入的文件列表进行拆分。新拆分的文件以“etcfile”为前缀，生成了 etcfileaa、etcfilead、etcfileab、etcfileae、etcfileac 和 etcfileaf，总共6个文件。

6.11.46 stat 命令：显示 inode 内容

stat 命令用来显示文件的详细信息。该命令的语法格式如下所示。



stat[-f,--filesystem][-c--format=FORMAT][-L,--dereference][-t,--terse][--help][--version]
其中各选项的具体说明如表 6-68 所示。

表 6-68 stat 命令的选项说明

选 项	说 明
-f, --filesystem	显示文件系统状态而不是文件状态
-c --format=FORMAT	使用指定的格式，而不是默认
-L, --dereference	跟随链接
-t, --terse	以简洁格式显示信息
--help	显示帮助信息并离开
--version	显示版本信息并离开

例如，通过命令 stat，显示文件 dict2 的详细信息，命令如下所示。

```
# ls //显示文件列表
123 dict2 mybook my.doc mypaper mypaper.doc patch sum sum2 xx01
dict fonts.dir mydoc mylist mypaper.bak paper.diff path sum1 xx00
# stat dict2 //显示文件信息
File: 'dict2'
Size: 1451172 Blocks: 2848 IO Block: 4096 Regular File
Device: 802h/2050d Inode: 814423 Links: 1
Access: (0644/-rw-r--r--) Uid: ( 0/ root) Gid: ( 0/ root)
Access: 2008-08-08 09:01:29.000000000 +0800
Modify: 2008-08-08 08:06:36.000000000 +0800
Change: 2008-08-08 08:06:36.000000000 +0800
```

例如，通过命令 stat，显示文件系统/dev/sda 的详细信息，命令如下所示。

```
# stat /dev/sda
File: '/dev/sda'
Size: 0 Blocks: 0 IO Block: 4096 Block Device
Device: 802h/2050d Inode: 66914 Links: 1 Device type: 8,0
Access: (0660/brw-rw----) Uid: ( 0/ root) Gid: ( 6/ disk)
Access: 2003-01-30 18:24:37.000000000 +0800
Modify: 2003-01-30 18:24:37.000000000 +0800
Change: 2008-08-07 14:34:34.000000000 +0800
```

6.11.47 scp 命令：远程复制文件

scp 命令用于在网络中的不同主机之间复制文件。该命令的语法格式如下所示。

```
scp [-q] [-4][-6] [-F config] [-P port][-i identity][--help]
[[user@]host1:]file1 [...] [[user@]host2:]file2
```

其中各选项的具体说明如表 6-69 所示。

表 6-69 scp 命令的选项说明

选 项	说 明
-4	强制使用 IPv4 协议
-6	强制使用 IPv6 协议
-q	不显示处理信息
-v	运行时显示详细的处理信息

续表

选 项	说 明
-F config	指定配置文件
-i identity	指定认证文件
-P port	指定端口号
--help	显示帮助信息
user	用户名
host1, host2	主机名
file1, file2	复制的文件名

例如，通过命令 scp，从主机 192.168.20.122 复制文件/root/test 到远端主机 192.168.20.123 的目录/home/my/下，命令如下所示。

```
# scp -v root@192.168.20.122:/root/test yang@192.168.20.123:/home/my///复制文件到远端主机
Executing: /usr/bin/ssh -v -x -o'ClearAllForwardings yes' -t -l root 192.168.20.122 scp
-v /root/test 'yang@192.168.20.123:/home/my/'
OpenSSH_4.0p1, OpenSSL 0.9.7f 22 Mar 2005 //建立 OpenSSL 连接
debug1: Reading configuration data /etc/ssh/ssh_config //读取 ssh 配置文件
debug1: Applying options for *
debug1: Connecting to 192.168.20.122 [192.168.20.122] port 22.
debug1: Connection established.
debug1: permanently_set_uid: 0/0
debug1: identity file /root/.ssh/identity type -1
debug1: identity file /root/.ssh/id_rsa type -1
debug1: identity file /root/.ssh/id_dsa type -1
debug1: Remote protocol version 2.0, remote software version OpenSSH_4.0
debug1: match: OpenSSH_4.0 pat OpenSSH*
debug1: Enabling compatibility mode for protocol 2.0 //使用适合于协议 2.0 的兼容模式
debug1: Local version string SSH-2.0-OpenSSH_4.0
debug1: SSH2_MSG_KEXINIT sent
debug1: SSH2_MSG_KEXINIT received
debug1: kex: server->client aes128-cbc hmac-md5 none
debug1: kex: client->server aes128-cbc hmac-md5 none
debug1: SSH2_MSG_KEX_DH_GEX_REQUEST(1024<1024<8192) sent
debug1: expecting SSH2_MSG_KEX_DH_GEX_GROUP
debug1: SSH2_MSG_KEX_DH_GEX_INIT sent
debug1: expecting SSH2_MSG_KEX_DH_GEX_REPLY
debug1: Host '192.168.20.122' is known and matches the RSA host key.
debug1: Found key in /root/.ssh/known_hosts:1
debug1: ssh_rsa_verify: signature correct //SSH 的 RSA 验证通过
debug1: SSH2_MSG_NEWKEYS sent
debug1: expecting SSH2_MSG_NEWKEYS
debug1: SSH2_MSG_NEWKEYS received
debug1: SSH2_MSG_SERVICE_REQUEST sent
debug1: SSH2_MSG_SERVICE_ACCEPT received
debug1: Authentications that can continue: publickey,gssapi-with-mic,password
debug1: Next authentication method: gssapi-with-mic
debug1: An invalid name was supplied
Cannot determine realm for numeric host address
debug1: An invalid name was supplied
Cannot determine realm for numeric host address
```



```
debug1: Next authentication method: publickey
debug1: Trying private key: /root/.ssh/identity
debug1: Trying private key: /root/.ssh/id_rsa
debug1: Trying private key: /root/.ssh/id_dsa
debug1: Next authentication method: password
root@192.168.20.122's password:
debug1: Authentication succeeded (password). //成功认证
debug1: channel 0: new [client-session]
debug1: Entering interactive session.
debug1: Sending command: scp -v /root/test yang@192.168.20.123:/home/my/
Executing: program /usr/bin/ssh host 192.168.20.123, user yang, command scp -v -t
/home/my/ /建立 ssh 连接
OpenSSH_4.0p1, OpenSSL 0.9.7f 22 Mar 2005
debug1: Reading configuration data /etc/ssh/ssh_config
debug1: Applying options for *
debug1: Connecting to 192.168.20.123 [192.168.20.123] port 22.
debug1: Connection established. //连接已建立
debug1: permanently_set_uid: 0/0
debug1: identity file /root/.ssh/identity type -1
debug1: identity file /root/.ssh/id_rsa type -1
debug1: identity file /root/.ssh/id_dsa type -1
debug1: Remote protocol version 1.99, remote software version OpenSSH_3.5p1
debug1: match: OpenSSH_3.5p1 pat OpenSSH_3.*
debug1: Enabling compatibility mode for protocol 2.0 //启用兼容模式
debug1: Local version string SSH-2.0-OpenSSH_4.0
debug1: SSH2_MSG_KEXINIT sent
debug1: SSH2_MSG_KEXINIT received
debug1: kex: server->client aes128-cbc hmac-md5 none
debug1: kex: client->server aes128-cbc hmac-md5 none
debug1: SSH2_MSG_KEX_DH_GEX_REQUEST(1024<1024<8192) sent
debug1: expecting SSH2_MSG_KEX_DH_GEX_GROUP
debug1: SSH2_MSG_KEX_DH_GEX_INIT sent
debug1: expecting SSH2_MSG_KEX_DH_GEX_REPLY
The authenticity of host '192.168.20.123 (192.168.20.123)' can't be established.
RSA key fingerprint is e6:f3:8a:13:82:e8:72:0f:51:e2:26:3b:ea:f4:37:c9.
Are you sure you want to continue connecting (yes/no)? yes//确认是否建立连接
Warning: Permanently added '192.168.20.123' (RSA) to the list of known hosts.
debug1: ssh_rsa_verify: signature correct
debug1: SSH2_MSG_NEWKEYS sent
debug1: expecting SSH2_MSG_NEWKEYS
debug1: SSH2_MSG_NEWKEYS received
debug1: SSH2_MSG_SERVICE_REQUEST sent
debug1: SSH2_MSG_SERVICE_ACCEPT received
debug1: Authentications that can continue: publickey,password,keyboard-interactive
debug1: Next authentication method: publickey
debug1: Trying private key: /root/.ssh/identity
debug1: Trying private key: /root/.ssh/id_rsa
debug1: Trying private key: /root/.ssh/id_dsa
debug1: Next authentication method: keyboard-interactive
debug1: Authentications that can continue: publickey,password,keyboard-interactive
debug1: Next authentication method: password
yang@192.168.20.123's password: //输入口令
debug1: Authentication succeeded (password).
debug1: channel 0: new [client-session]
debug1: Entering interactive session. //进入交互会话
debug1: Sending command: scp -v -t /home/my/ //传输 scp 命令
Sending file modes: C0644 220 123
```



```
123
220    0.2KB/s  00:00
debug1: client_input_channel_req: channel 0 rtype exit-status reply 0
debug1: channel 0: free: client-session, nchannels 1
debug1: fd 0 clearing O_NONBLOCK
debug1: fd 1 clearing O_NONBLOCK
debug1: Transferred: stdin 0, stdout 0, stderr 0 bytes in 0.1 seconds
debug1: Bytes per second: stdin 0.0, stdout 0.0, stderr 0.0
debug1: Exit status 0
debug1: client_input_channel_req: channel 0 rtype exit-status reply 0
debug1: channel 0: free: client-session, nchannels 1
Connection to 192.168.20.122 closed.           //关闭连接
debug1: Transferred: stdin 0, stdout 0, stderr 37 bytes in 9.0 seconds
debug1: Bytes per second: stdin 0.0, stdout 0.0, stderr 4.1
debug1: Exit status 0
```

6.11.48 sum 命令：计算文件的校验码

sum 命令可以用来计算文件的校验码，同时显示块数。该命令的语法格式如下所示。

```
sum [-r][-s, --sysv][--help][--version]
```

其中各选项的具体说明如表 6-70 所示。

表 6-70 sum 命令的选项说明

选 项	说 明
-r	默认项，使用 BCD 校验算法，每块 1024 字节
-s, --sysv	使用 System V 校验算法，每块 512 字节
--help	显示版本信息
--version	显示版本信息

例如，通过命令 sum ， 计算文件 dict2 的校验码，如下所示。

```
# ll dict2
-rw-r--r--  1 root  root    1451172  8月  8 08:06 dict2
# sum dict2           //计算文件的校验码
20805 1418
```

6.11.49 tmpwatch 命令：删除临时文件

tmpwatch 命令可以用来删除最近一段时间内没有使用过的临时文件。该命令的语法格式如下所示。

```
tmpwatch[-u,--atime][-m,--mtime][-c,--ctime][-a,--all][-d,--nodirs][-f,--force][-t,
--test][-s,--fuser]
[-v,--verbose]
```

其中各选项的具体说明如表 6-71 所示。

表 6-71 tmpwatch 命令的选项说明

选 项	说 明
-u, --atime	基于文件的访问时间删除文件，默认值
-m, --mtime	基于文件的修改时间删除文件
-c, --ctime	基于文件 i 节点的修改时间删除文件，默认值



续表

选 项	说 明
-a, --all	删除所有文件，不仅删除常规文件和目录
-d, --nodirs	不删除目录
-f, --force	强制删除
-t, --test	不删除文件，仅进行测试
-s, --fuser	在删除文件之前，使用 fuser 命令检测该文件是否正被使用
-v, --verbose	冗余模式，具有 2 级冗余模式（连接两次使用该选项）

例如，通过命令 `tmpwatch`，删除/tmp 目录下 100 小时内没有使用的文件，该命令如下所示。

```
# tmpwatch 100 /tmp/
```

6.11.50 touch 命令：更新文件或目录时间

使用 `touch` 命令更新指定目录或文件的存取时间为当前时间，即改变文件的时间戳。如果 `file` 文件不存在，则创建该文件。`touch` 命令的语法格式如下所示。

```
touch[-a] [-BSEC, --backward=SEC] [-c, --no-create] [-d, --date=STRING] [-FSEC, --forward=SEC] [-f] [-m]
[-r, --reference=文件] [-tSTAMP] [--time=WORD] [--help] [--version]
```

其中各选项的具体说明如表 6-72 所示。

表 6-72 touch 命令的选项说明

选 项	说 明
-a	更改存取时间
-B SEC, --backward=SEC	时间回调 SEC 秒
-c, --no-create	不创建任何文件
-d, --date=STRING	使用指定日期，而不是当前时间
-F SEC, --forward=SEC	时间提前 SEC 秒
-f	忽略
-m	只更新变动的的时间
-r, --reference=文件	使用指定<文件>的时间属性而非目前的时间
-t STAMP	使用 [[CC]YY]MMDDhhmm[.ss] 格式的时间而非目前的时间
--time=WORD	使用 WORD 指定的时间：access、atime、use 等于 -a 选项的效果，而 modify、mtime 等于 -m 选项的效果
--help	显示帮助信息并离开
--version	显示版本信息并离开



-d 和 -t 选项可接受不同的时间/日期格式。

例如，使用 touch 命令创建 sample 文件，如下所示。

```
# ls -a
. . .
# touch example
# ls -a
. . . example
```

使用 touch 文件更新时间戳，命令如下：

```
# ls -al
drwxrwxr-x 2 root root 4096 07-27 00:47 .
drwxr-xr-x 6 root root 4096 07-27 00:47 ..
-rw-rw-r-- 1 root root 0 07-27 00:47 example //创建时间为 07-27 00:47
# touch example
# ls -l example
-rw-rw-r-- 1 root root 0 07-27 00:48 example //修改时间戳为 07-27 00:48
```

例如，更新文件 my.doc 的时间为当前时间，命令如下所示。

```
# ll //显示文件信息
总用量 40
-rw-r--r-- 1 root root 10659 5月 9 09:37 my.doc
-rw-r--r-- 1 root root 97 5月 15 21:11 abc.doc
drwxr-xr-x 2 root root 4096 3月 30 14:30 TEST
drwxr-xr-x 9 root root 4096 5月 3 17:51 helloworld
# touch my.doc //更新文件存取时间
# ll //显示文件信息
总用量 40
-rw-r--r-- 1 root root 10659 5月 30 17:22 my.doc
-rw-r--r-- 1 root root 97 5月 15 21:11 abc.doc
drwxr-xr-x 2 root root 4096 3月 30 14:30 TEST
drwxr-xr-x 9 root root 4096 5月 3 17:51 helloworld
```

例如，更新文件的存取时间为指定时间 2008 年 8 月 8 日，命令如下所示。

```
# ls -l //显示文件信息
总用量 36
-rwxr-xr-x 1 root root 16368 6月 2 22:42 bubbleSort
-rw-r--r-- 1 root root 232 6月 2 22:42 bubbleSort.bb
-rw-r--r-- 1 root root 380 6月 2 22:42 bubbleSort.bbg
-rw-r--r-- 1 root root 80 6月 2 22:50 bubbleSort.da
-rw-r--r-- 1 root users 85 6月 5 19:36 kk2.c
drwxr-xr-x 2 root root 4096 8月 3 11:59 test
# touch test -t 200808081500.00 //更新文件存取时间
# ls -l
总用量 36
-rwxr-xr-x 1 root root 16368 6月 2 22:42 bubbleSort
-rw-r--r-- 1 root root 232 6月 2 22:42 bubbleSort.bb
-rw-r--r-- 1 root root 380 6月 2 22:42 bubbleSort.bbg
-rw-r--r-- 1 root root 80 6月 2 22:50 bubbleSort.da
-rw-r--r-- 1 root users 85 6月 5 19:36 kk2.c
drwxr-xr-x 2 root root 4096 2008-08-08 test
```

6.11.51 tree 命令：以树状图显示目录内容

tree 命令将列出指定目录下的所有文件，包括子目录里的所有文件。该命令的语法格式如下所示。

```
tree[-a][-d][-f][-i][-l][-x][-Ppattern][-p][-s][-u][-g][-D][-F][-q][-N][-t][-n][-C][-A]
```



其中各选项的具体说明如表 6-73 所示。

表 6-73 tree 命令的选项说明

选 项	说 明
-a	显示所有文件
-d	仅列出目录
-f	每一文件输出完全的路径前缀
-i	目录树不显示缩进行，通常与选项-f 联合使用
-l	跟随符号链接
-x	保持在当前的文件系统
-P pattern	仅列出符合指定模式的文件
-p	显示每个文件的权限
-s	显示文件的大小
-u	显示用户名或 UID
-g	显示用户组名或 GID
-D	显示上次修改时间
-F	在目录前添加“/”，在 socket 文件前添加“=”，在可执行文件前添加“*”，在 FIFO 文件前添加“ ”
-q	对于有问题的文件，在文件名前打印非打印字符
-N	显示非打印字符
-t	根据上次修改时间进行排序，而不是根据字母表序列排序
-n	关闭颜色显示
-C	开启颜色显示
-A	显示缩进行时，开启 ASCII 绘图符

例如，通过命令 tree，显示/root 目录的目录树，如下所示。

```
# tree /test //显示指定的目录树
/test
|-- 123
|-- backup
|-- backup1
|-- backup2
|-- backup3
|-- dict
|-- dict2
|-- fonts.dir
|-- info
|-- my.doc
|-- mybook
|-- mydoc
|-- mylist
|-- mypaper
|-- mypaper.bak
|-- mypaper.doc
|-- paper.diff
```



```

|-- patch
|-- path
|-- student
|   |-- Zhangsan          //student 目录下的子目录 Zhangsan
|   |   |-- 123.c
|   |   |-- homework.c
|-- sum
|-- sum1
|-- sum2
|-- xx00
|-- xx01

```

2 directories, 26 files //共2个目录, 26个文件

例如, 通过选项 “-A”, 以 ASCII 绘图字符显示/test 目录树, 命令如下所示。

```

# tree -A /test          //显示目录树
/test
|__ 123
|__ backup
|__ backup1
|__ backup2
|__ backup3
|__ dict
|__ dict2
|__ fonts.dir
|__ info
|__ my.doc
|__ mybook
|__ mydoc
|__ mylist
|__ mypaper
|__ mypaper.bak
|__ mypaper.doc
|__ paper.diff
|__ patch
|__ path
|__ student
|   |__ Zhangsan          //student 目录下的子目录 Zhangsan
|   |   |__ 123.c
|   |   |__ homework.c
|__ sum
|__ sum1
|__ sum2
|__ xx00
|__ xx01

```

2 directories, 26 files //共2个目录, 26个文件

例如, 在显示目录树的同时, 显示每一文件或目录的权限, 命令如下所示。

```

# tree -p /test          //显示带权限的目录树
/test
|-- [-rw-r--r--] 123
|-- [-rw-r--r--] backup
|-- [-rw-r--r--] backup1
|-- [-rw-r--r--] backup2
|-- [-rw-r--r--] backup3
|-- [-rw-r--r--] dict
|-- [-rw-r--r--] dict2

```




```

|-- [-rw-r--r--] fonts.dir
|-- [-rw-r--r--] info
|-- [-rw-r--r--] my.doc
|-- [-rw-r--r--] mybook
|-- [-rw-r--r--] mydoc
|-- [-rw-r--r--] mylist
|-- [-rw-r--r--] mypaper
|-- [-rw-r--r--] mypaper.bak
|-- [-rw-r--r--] mypaper.doc
|-- [-rw-r--r--] paper.diff
|-- [-rw-r--r--] patch
|-- [-rw-r--r--] path
|-- [drwxr-xr-x] student
| |-- [drwxr-xr-x] Zhangsan //student 目录下的子目录 Zhangsan
| | `-- [-rw-r--r--] 123.c
| `-- [-rw-r--r--] homework.c
|-- [-rw-r--r--] sum
|-- [-rw-r--r--] sum1
|-- [-rw-r--r--] sum2
|-- [-rw-r--r--] xx00
`-- [-rw-r--r--] xx01
2 directories, 26 files //共 2 个目录, 26 个文件

```

例如, 在显示目录树的同时, 显示每一文件或目录的所属用户和用户组, 以及文件大小, 命令如下所示。

```

# tree -sug /test
/test
|-- [root root 91] 123
|-- [root root 70] backup
|-- [root root 70] backup1
|-- [root root 70] backup2
|-- [root root 70] backup3
|-- [root root 1451172] dict
|-- [root root 1451172] dict2
|-- [root root 2] fonts.dir
|-- [root root 70] info
|-- [root root 289] my.doc
|-- [root root 103] mybook
|-- [root root 91] mydoc
|-- [root root 103] mylist
|-- [root root 102] mypaper
|-- [root root 102] mypaper.bak
|-- [root root 102] mypaper.doc
|-- [root root 253] paper.diff
|-- [root root 7] patch
|-- [root root 0] path
|-- [root root 4096] student
| |-- [root root 4096] Zhangsan //student 目录下的子目录 Zhangsan
| | `-- [root root 0] 123.c
| `-- [root root 0] homework.c
|-- [root root 120] sum
|-- [root root 40] sum1
|-- [root root 39] sum2
|-- [root root 438] xx00
`-- [root root 1450734] xx01
2 directories, 26 files //共 2 个目录, 26 个文件

```


6.11.52 which 命令：查找文件

which 命令用来查找文件，其查找范围由环境变量\$PATH 设置。该命令的语法格式如下所示。

```
which[--version,-[vV]][--help][--skip-dot][--skip-tilde][--show-tilde][--tty-only][
--all,-a]
[--read-alias,-i][--skip-alias][--read-functions][--skip-functions]
```

其中各选项的具体说明如表 6-74 所示。

表 6-74 which 命令的选项说明

选 项	说 明
--version, -[vV]	显示版本信息
--help	显示帮助信息
--skip-dot	跳过路径中以“.”开始的目录
--skip-tilde	跳过路径中以“~”开始的目录
--show-dot	不添加符号“.”到当前的输出目录中
--show-tilde	对非根用户在主目录中显示符号“~”
--tty-only	如果不是在 tty 进行操作，立即停止处理
--all, -a	输出所有的匹配，而不是仅输出第一个
--read-alias, -i	从标准输入设备读入别名列表
--skip-alias	忽略别名
--read-functions	从标准输入设备读取 Shell 函数
--skip-functions	不从标准输入设备读入函数

例如，使用 which 命令查找文件 myself，但是在指定的路径下没有找到文件 myself，执行命令过程如下所示。

```
# which linux
# which myself //查找文件
/usr/bin/which:no myself in(/usr/kerberos/sbin:/usr/kerberos/bin:/usr/local/bin:/
bin:/usr/bin:/usr/X11R6/bin:/home/teacher/bin) //未找到
#
```

例如，通过命令 which，查找文件 ls，命令如下所示。

```
# which ls //查找文件
alias ls='ls --color=tty'
/bin/ls
```

6.12 文件和目录管理命令的综合应用

文件和目录管理是 Linux 系统管理的重要组成部分。本节将综合运用多个命令对使用中经常出现的问题进行分析处理。



6.12.1 无法卸载已挂载的文件系统

如果用户无法卸载一个已经挂载的文件系统，并且系统总是提示“device is busy”，说明该文件系统正被某个程序使用。必须先退出正在使用该文件系统的程序，才能完成卸载。可以使用 `lsof` 命令查看是否有任何程序正在使用该文件系统，命令如下所示。

```
#lsof | grep 文件系统的挂载点
```

从命令的执行结果可以找出当前使用该文件的程序。停止该程序的运行，然后再次卸载该文件系统。

6.12.2 查询设备上采用的未知文件系统

可以使用 `blkid` 命令对设备上所采用的文件系统类型进行查询。`blkid` 命令主要用来对系统中的块设备（包括交换分区）所使用的文件系统类型、LABEL、UUID 等信息进行显示。例如，查看 `/dev/sda1` 设备所采用的文件系统类型，命令如下所示。

```
# blkid /dev/sda1
/dev/sda1: LABEL="/boot" UUID="bc360f28-cae0-45fd-821b-d4b3600e3858" SEC_TYPE="ext2"
TYPE="ext3"
```

直接使用 `blkid` 命令可以列出当前系统中所有已挂载文件系统的类型，如下所示。

```
# blkid
/dev/mapper/VolGroup00-LogVol01: TYPE="swap"
/dev/mapper/VolGroup00-LogVol00: UUID="034dae0c-363c-41a1-97e8-b2e6933f7c0a" SEC_TYPE="ext2" TYPE="ext3"
/dev/sda1: LABEL="/boot" UUID="bc360f28-cae0-45fd-821b-d4b3600e3858" SEC_TYPE="ext2" TYPE="ext3"
/dev/VolGroup00/LogVol01: TYPE="swap"
/dev/cdrom: LABEL="TomatoWinXP_SP2_v3.0" TYPE="iso9660"
```

6.12.3 恢复已删除的文件

如果在命令行下误删除了某个文件，该文件是无法从回收站中找回的，必须使用命令进行恢复。如果系统使用的是 `ext3`，由于 `ext3` 文件系统在执行删除任务时会清除指向删除文件的信息节点指针，因此一般无法进行恢复。而 `ext2` 文件系统在删除某一文件时只是对该块标记为未使用，只要该块没有被其他文件所占用，那么可以使用 `debugfs` 命令试着进行恢复。

恢复前应先将包含要恢复文件的分区进行卸载。例如，恢复 `/boot` 目录中的 `example` 文件，首先使用 `umount` 命令卸载 `/boot`，如下所示。

```
#umount /boot
```

然后运行 `debugfs` 命令，并在提示符下使用 `ls` 命令，列出从该目录下删除的文件，如下所示。

```
#debugfs /dev/sda1
debugfs 1.39 (29-May-2006)
debugfs: ls -ld //使用 ls 命令
  2  40755 (2)    0    0  1024  5-Aug-2007 09:43 .
  2  40755 (2)    0    0  1024  5-Aug-2007 09:43 ..
 11  40700 (2)    0    0 12288 15-Jul-2007 19:43 lost+found
8033 40755 (2)    0    0  1024 15-Jul-2007 20:58 grub
 18 100600 (1)    0    0 2318736 15-Jul-2007 19:59 initrd-2.6.18-8.el
5xen.img
```



```
12 100644 (1) 0 0 868062 27-Jan-2007 03:51 System.map-2.6.18-8
.el5xen
13 100644 (1) 0 0 61053 27-Jan-2007 03:51 config-2.6.18-8.el5x
en
14 100644 (1) 0 0 84906 27-Jan-2007 03:51 symvers-2.6.18-8.el5
xen.gz
15 100644 (1) 0 0 2076303 27-Jan-2007 03:51 vmlinuz-2.6.18-8.e
l5xen
16 100755 (1) 0 0 608564 27-Jan-2007 04:01 xen-syms-2.6.18-8.e
l5
17 100644 (1) 0 0 274752 27-Jan-2007 03:12 xen.gz-2.6.18-8.el5
< 211> 0 (1) 0 0 0 example
debugfs:
```

在“<>”之间显示的是文件信息节点编号“211”，可以使用下面的命令恢复 example 文件。
#debugfs: dump <211> /home/teacher1/example_dump.txt

通过使用 dump 命令，将 example 文件的内容复制到了/home/teacher1/example_dump.txt 文件里。



如果“<>”之间显示的是零值，则不可以进行恢复。

6.12.4 无法引导文件系统

如果系统出现较为严重的错误，导致无法引导文件系统，可以尝试从引导盘启动系统，然后对受损的文件系统进行修复。如果没有定制引导盘，也可以使用 Red Hat Linux 9 安装光盘套装中的第 1 张光盘，通过 Linux Rescue 模式，启动 Linux 系统进行修复。

例如，将 Red Hat Linux 9 第 1 张安装盘放入光盘驱动器，然后设置系统从光盘启动。从光盘引导系统后，出现如图 6-8 所示的界面。

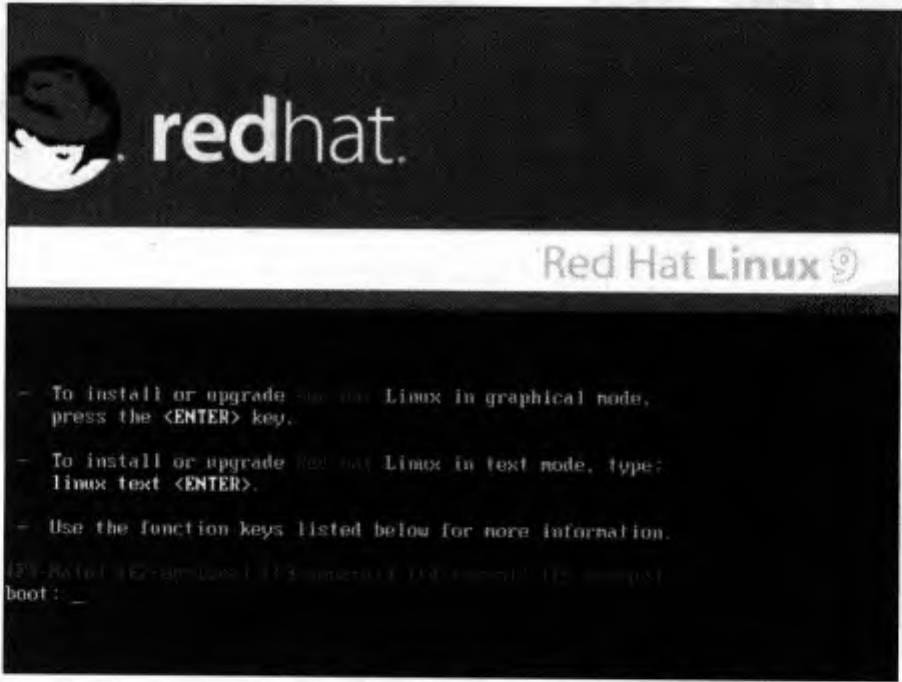


图 6-8 从光盘引导 Linux 文件系统

按 F5 键进入救援模式（Linux Rescue），如图 6-9 所示。在提示符下键入“linux rescue”，



按 Enter 键，启动救援模式。



图 6-9 键入 “linux rescue” 进入救援模式

选择语言、键盘以及是否启用网络后，弹出图 6-10 所示的【救援】选择对话框。

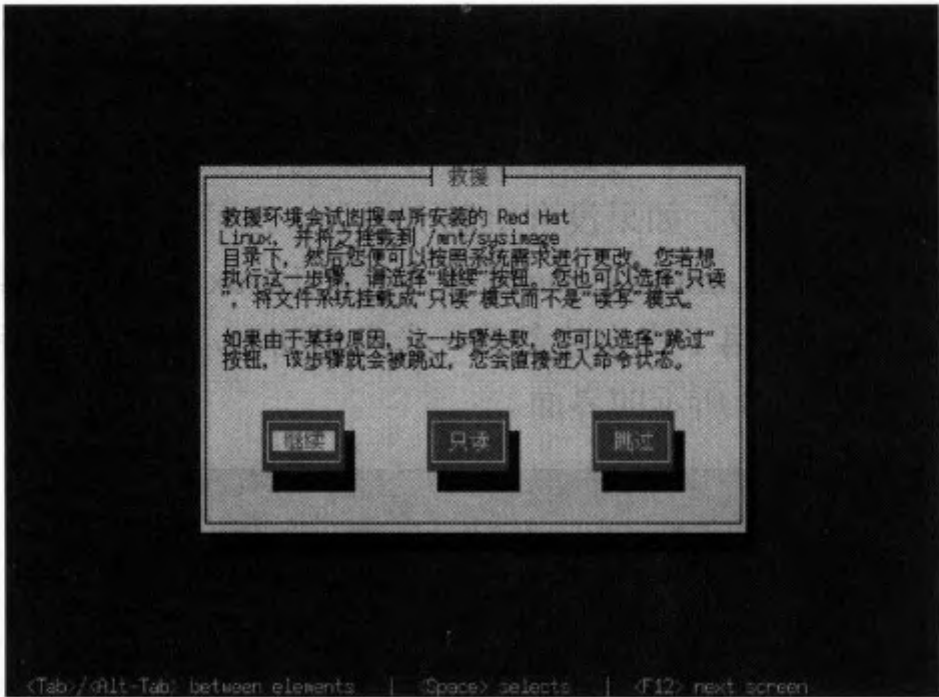


图 6-10 Linux 救援模式选项

其中包括三个选项。

- **【继续】**：如果选择**【继续】**选项，Red Hat Linux 9 将搜索已安装的硬盘，所有查找到的文件系统均被安装在/mnt/sysimage 目录中。
- **【只读】**：**【只读】**选项与**【继续】**选项几乎完全相同，差别仅在于**【只读】**选项会将查找到的文件系统以只读模式加以挂载。
- **【跳过】**：**【跳过】**选项直接进入一个命令 Shell。

单击**【继续】**选项，弹出图 6-11 所示的**【救援】**对话框。

从**【救援】**对话框可以看到：

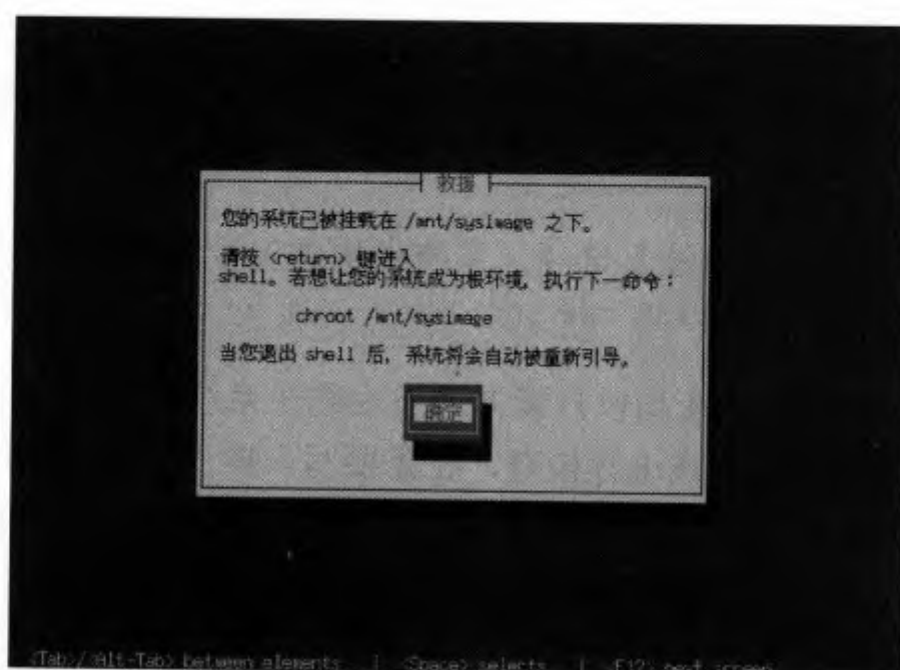


图 6-11 从【继续】选项进入救援模式

- 硬盘的文件系统被挂载到了/mnt/sysimage 目录下，如果用户仍然希望使用自己以前系统的根环境，可以在进入 Shell 后使用命令“chroot /mnt/sysimage”；
- 使用命令 exit 可以从 Shell 中退出，退出后系统自动重启。

在图 6-11 中，选择【确定】，系统会进入一个命令 Shell，如下所示。

```
Your system is mounted under the /mnt/sysimage directory.
When finished please exit from the shell and your system will reboot.
#
```

使用 e2label 命令可以进一步查看分区被重新挂载的位置。例如，查看/sda1 的挂载点，命令行如下所示。

```
# e2label /dev/sda1
/boot
```

可以看到，/dev/sda1 被挂载到了/boot 目录下。

进入/mnt/sysimage 目录，查看原来的文件系统，如下所示。

```
# cd /mnt/sysimage
# ls
dev lib mnt proc selinux sys
var bin exit lost+found opt root share tmp
boot home media password sbin srv usr
```

如果有一个或多个文件系统没有得到安装，可以用 vi 编辑器打开 fstab 配置文件，检查其中的设置和格式本身的完整性，如下所示。

```
# vi /mnt/sysimage/etc/fstab
```

可以使用 fsck 命令修复一个受损的文件系统。例如，修复/dev/sda2，可以使用下面的命令。

```
# fsck /dev/sda2
```

如果在救援模式下需要访问 Linux 手册页面，但由于顶级根目录已不是原来文件系统分级结构的最顶层，所以无法激活 Linux 手册页面。此时可以运行 chroot 命令，更改顶级根目录的位置，执行命令如下所示。

```
# chroot /mnt/sysimage
# cd /
```



```
# ls
dev lib mnt proc selinux sys
var bin exit lost+found opt root share tmp
boot home media password sbin srv usr
# man
What manual page do you want?
```

如果在图 6-10 中选择了【只读】选项，系统也将进入救援模式，系统提示如下信息。

```
When finished please exit from the shell and your system will reboot.
#
```

【只读】选项表明，原文件系统均以只读方式被安装在系统中。如果用户有多个文件系统，比较安全的作法应该是以只读方式进行检查，在需要写入操作时，再使用 `mount` 命令重新挂载需要写入操作的文件系统。例如，重新挂载 `/dev/sda2` 分区，使其能够执行写入操作，如下所示。

```
# mount -w -o remount /dev/sda2
```

如果在图 6-10 中选择了【跳过】选项，系统会直接进入一个 Shell。该选项是将一个最小的根系统和内核装入到一个 RAM 盘中，不安装任何文件系统，仅提供给用户一个 Shell。用户需要完全依靠手动挂载、修复文件系统，这需要用户对 `mount`、`mkdir`、`cp`、`move`、`fdisk`、`fsck` 等命令十分熟悉。

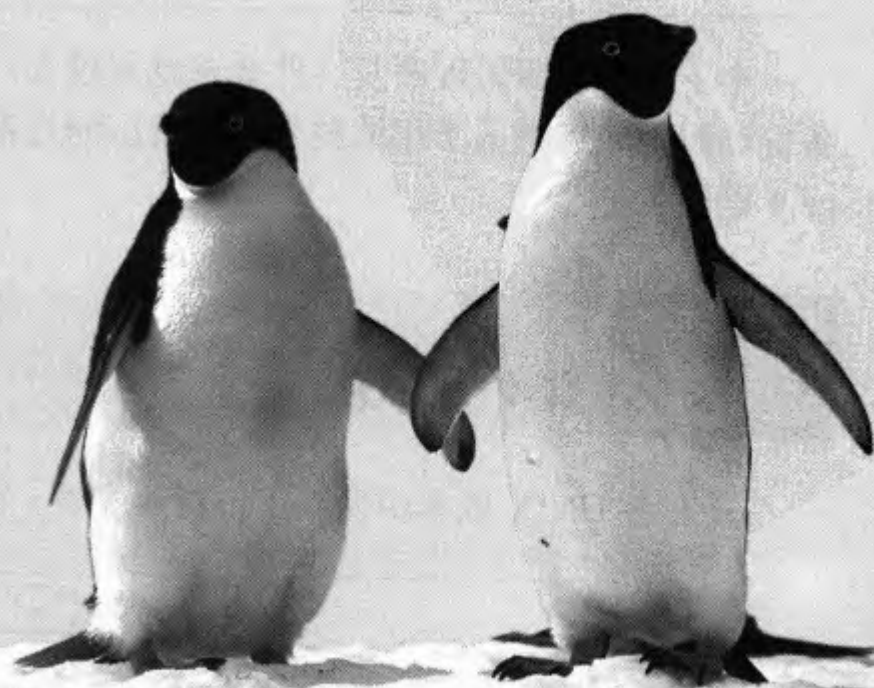
LINUX

第7章

备份与压缩

没有任何系统是绝对可靠的，防止数据丢失最切实可行的方法是定期进行数据备份。同时，适当对数据进行压缩，可以有效提高存储空间的使用率，尤其是对需要备份的数据，通常是备份与数据压缩同步进行。

为保护数据，实现系统资源的有效利用，Linux 提供了大量的用于备份与压缩的工具，比较流行的如 tar、gzip、zip、bzip2 等。





7.1 Linux 备份与压缩的策略

在 Linux 系统中，对系统目录的备份是一种有效的保护手段，但并不是所有目录都需要备份，表 7-1 列出了系统目录备份的一个常规建议。

表 7-1 备份系统目录

1. 必须备份或需要频繁备份的目录	
/etc	系统配置文件存放的目录
/home	用户的目录空间
/root	根用户的主目录
/var	日志、邮件、打印机队列等存放的目录
/var/www	www 服务器使用的目录
/var/ftp	ftp 服务器使用的目录
2. 不需要经常备份或只在全系统备份时才需要备份的目录	
/bin	存放操作系统中的可执行程序（主要是一些命令）的目录
/boot	存放引导文件、内核文件的目录
/dev	存放设备文件的目录（包括硬盘、软盘等）
/lib	存放库文件、pam 和内核模块
/usr	存放用户使用的系统命令和应用程序等信息的目录
/tmp	临时文件目录
/opt	第三方和 FHS 软件存放目录
/lost+found	文件系统中被损坏或未正常链接的文件存放目录
/sbin	存放系统启动时需要执行的程序
3. 不需要备份或很少备份的目录	
/mnt	文件系统的挂载点
/proc	存放内核、操作系统的动态信息

由于现在的应用程序及文件普遍越来越大，为了节省磁盘空间，减少网络传输代价，在备份过程中一般采用了压缩技术。通常压缩与备份是同步进行的，常用的命令包括 tar、zip、gzip 等。

7.2 tar 命令

tar 在各 UNIX 版本中受到了广泛的支持，有着非常悠久的历史。tar 是 Tape Archive 的缩

写，最早是与磁带机联系在一起的，用于将系统中需要备份的数据打包归档到磁带中，在需要时再把备份的数据从磁带中恢复回来。随着计算机硬盘容量不断增大，CD-R 及移动磁盘的广泛使用，tar 命令已不仅仅局限在磁带机，而更多地应用在磁盘备份中。tar 命令本身只进行打包不进行压缩，其主要功能是将多个文件或目录打包在一个文件里，便于传输和保存。为了减少备份文件的大小，节省存储空间，tar 命令经常和许多压缩程序配合使用。tar 命令的语法格式如下所示。

```
tar      [-c, --create][ --same-owner][ -w, --interactive, --confirmation][ -x,
--extract, --get] [ --version][ --help]
      [ -Z, --compress, --uncompress][ -z, --compress, --uncompress][ -t, --list] [ -i,
--ignore-zeros]
      [ -A, --catenate, --concatenate][ --checkpoint][ -c, --create][ --delete][ -d, --diff,
--compare]
      [ -r, --append][ -u, --update][ -x, --extract, --ge][ -t, --list][ --atime-preserve][ -b,
--block-size N]
      [ -k, --keep-old-files][ -l, --one-file-system][ -L, --tape-length N][ -m,
--modification-time] [-h, --dereference]
      [ -M, --multi-volume][ -N, --after-date DATE, --newer DATE][ -O, --to-stdout] [-g,
--listed-incremental F]
      [ -p, --same-permissions, --preserve-permissions][ --remove-files][ --totals][ -v,
--verbose]
      [ -V, --label NAME][ -W, --verify][ --use-compress-program PROG][ -G, --incremental]
```

其中各选项的具体说明如表 7-2 所示。

表 7-2 tar 命令的选项说明

选 项	说 明
-c, --create	创建新的备份文件
--same-owner	创建解压文件时，具有相同的所有者设置
-w, --interactive, --confirmation	对于每一操作要求确认
-x, --extract, --get	对文件进行解压缩
-Z, --compress, --uncompress	指定压缩为.Z 格式
-z, --compress, --uncompress	指定压缩为.gz 格式
-t, --list	查询包中内容
-A, --catenate, --concatenate	新增 tar 文件到已存在的归档包中
--checkpoint	读压缩文件时显示目录名
-c, --create	建立新的归档包
--delete	从归档包中删除
-d, --diff, --compare	查找压缩文件与文件系统之间的差别
-r, --append	添加文件到归档包文件的尾部
-u, --update	仅向归档包文件中添加时间更新的文件
-x, --extract, --ge	从归档包文件中提取文件
-t, --list	显示归档包内容
--atime-preserve	不改变归档包的时间
-b, --block-size N	指定块大小 为 N×512 字节



续表

选 项	说 明
-k, --keep-old-files	保留已存在的文件，不进行写覆盖
-l, --one-file-system	当创建归档文件时，保留本地文件系统
-L, --tape-length N	在写入 N×1024 字节后，更换磁带
-m, --modification-time	不提取文档修改时间
-M, --multi-volume	创建/提取/显示多卷归档
-N, --after-date DATE, --newer DATE	仅保留比指定时间更新的文件
-O, --to-stdout	提取文件到标准输出设备
-p, --same-permissions, --preserve-permissions	提取所有权限信息
--remove-files	添加到归档包之后，删除源文件
--totals	显示总字节数
-v, --verbose	冗余模式
-V, --label NAME	创建归档包的卷标
-W, --verify	在写入之后，对归档包进行校验
--use-compress-program PROG	指定压缩程序
-G, --incremental	创建/提取/显示 旧 GNU 格式的增量备份
-g, --listed-incremental F	创建/提取/显示 新 GNU 格式的增量备份
-h, --dereference	不归档链接文件，直接归档指向的源文件
-i, --ignore-zeros	忽略 0 字节大小的块
--version	版本信息
--help	帮助信息



tar 命令通常用来压缩和解压文件，但 tar 本身不具有压缩和解压功能，它需要调用其他命令来实现压缩和解压功能。

7.2.1 打包和解包的常规操作

tar 命令选项一共有 70 多个，表 7-2 列出的是常用的几十个选项，各选项可以配合在一起使用。例如，对当前目录下的所有文件和目录进行打包，生成 example.tar 备份文件，命令如下所示。

```
# ls
directory1 file1 file2 file3
# tar -cvf example.tar *
directory1/
directory1/file4
file1
file2
file3
# ls
```



```
directory1 example.tar file1 file2 file3
```

可以看到，已经生成了 `example.tar` 打包文件，要解开此文件只需将选项 “c” 改为 “x”，如下所示。

```
# tar -xvf example.tar
directory1/
directory1/file4
file1
file2
file3
```

在打包的时候如果使用了绝对路径，恢复时 `tar` 会自动将文件或目录恢复到原来的路径下，如果路径不存在就重新创建。为了避免这种情况发生，打包时应尽可能地进到子目录下，然后再执行 `tar` 命令，如下所示。

```
$cd /myshare
$tar -cvf /dev/sda1 *
```

`tar` 命令不支持分卷，不具有磁盘修复功能，但可以用 `tar` 将目录打包成一个文件，如下所示。

```
# ls directory1/
file4
# tar cvf directory1.tar directory1/
directory1/
directory1/file4
# ls
directory1 directory1.tar
```

7.2.2 查看 tar 包中的内容

可以使用 `tar` 命令中的选项 “-tf” 查看包中的内容。例如，对目录 `directory1` 进行打包，生成打包文件 `directory1.tar`，通过 “-tf” 查看包中内容，命令如下所示。

```
# ls directory1/                                //显示目录 directory1 中的内容
file4
# tar -cvf directory1.tar directory1/           //对目录 directory1 打包
directory1/
directory1/file4
# ls                                            //显示目录中的内容
directory1 directory1.tar
# tar -tf directory1.tar                       //查看 directory1.tar 包中的内容
directory1/
directory1/file4
```

7.2.3 打包链接文件

对于链接文件，`tar` 命令默认只打包链接，不会对源文件打包。如果需要对源文件打包，必须使用选项 “-h”。

例如，当前目录下有 `file1` 和 `file1_ln` 两个文件，`file1_ln` 是 `file1` 文件的链接文件，如下所示。

```
# ls -al
drwxr-xr-x 3 root root 4096 08-08 20:46 .
drwxr-xr-x 5 root root 4096 08-08 20:34 ..
-rw-r--r-- 1 root root 0 08-08 20:16 file1
lrwxrwxrwx 1 root root 5 08-08 20:46 file1_ln -> file1
# tar -cvf test1.tar *
```



```
file1
file1_ln
# tar -tvf test1.tar
-rw-r--r-- root/root          0 2007-08-08 20:16:34 file1
lrwxrwxrwx root/root          0 2007-08-08 20:46:04 file1_ln -> file1
```

由于没有使用“-h”选项，上例中只对链接本身进行了打包，下面使用“-h”选项，对链接源文件进行打包。

```
# tar -hcvf test1.tar *
file1
file1_ln
# tar -tvf test1.tar
-rw-r--r-- root/root          0 2007-08-08 20:16:34 file1
-rw-r--r-- root/root          0 2007-08-08 20:16:34 file1_ln
```

7.2.4 向包中添加新文件

对于打包后的文件，如果要在包中增加新的文件，只需使用“-r”选项，无需重新打包全部文件。例如，在上例生成的 test1.tar 中加入新的文件 file2，命令行如下。

```
# tar -rvf test1.tar file2
file2
# tar -tvf test1.tar
-rw-r--r-- root/root          0 2007-08-08 20:16:34 file1
-rw-r--r-- root/root          0 2007-08-08 20:16:34 file1_ln
-rw-r--r-- root/root          0 2007-08-08 20:16:39 file2
```

7.2.5 生成.tar.gz 压缩包

在 Linux 系统中，gzip 程序可以用来实现压缩，生成以“.gz”为后缀的压缩包。在使用 tar 命令进行打包的同时，配合使用“-z”选项，也可以同步生成“.gz”的压缩包。例如，将当前目录下的所有文件和目录打包并压缩，再保存为 tmp.tar.gz 文件，命令如下所示。

```
# ls
directory1 file1      file2      file3
# tar -zcvf tmp.tar.gz *           //对当前目录进行压缩
directory1/
directory1/file4
file1
file2
file3
# ls                               //可以看到已生成 tmp.tar.gz 压缩包
directory1 file1      file2      file3      tmp.tar.gz
```

使用“-ztf”选项可以查看压缩包中的内容，命令如下。

```
# tar -ztf tmp.tar.gz
directory1/
directory1/file4
file1
file2
file3
```

使用“-zxvf”选项可以对后缀为“.tar.gz”的压缩包解压缩，命令行如下。

```
# tar -zxvf tmp.tar.gz
directory1/
directory1/file4
```



```
file1
file2
file3
```

在上例中，也可以对 tmp.tar.gz 文件先用 gzip 命令解压，然后再使用 tar 命令解包，同样可以完成压缩包的展开。

7.3 gzip 与 gunzip 命令

Linux 中一种极为常见的压缩格式是“.gz”，该格式的压缩文件由 gzip 程序生成，由 gunzip 程序来解压缩。gzip 具有较高的压缩率，但只能逐个生成压缩文件，无法将多个文件压缩并打包成一个文件，所以 gzip 经常和 tar 命令配合使用。即先用 tar 对多个文件打包，然后再用 gzip 进行压缩，通常会生成后缀为“.tar.gz”或“.tgz”的文件。gzip 命令的语法格式如下所示。

```
gzip [-c][-d][-f][-h][-l][-L][-n][-N][-r][-t][-v][-V][-1][-9] [-S suffix] [filename]
```

gzip 的选项比较多，常用选项说明见表 7-3。

表 7-3 gzip 选项说明

选 项	说 明
-c --stdout	输出到标准输出设备上，原文件内容不变
-d --decompress	解压缩
-f --force	对输出文件强制写覆盖并对链接文件进行压缩
-h --help	显示帮助信息
-l --list	列出压缩包的内容
-L --license	显示软件许可权
-n --no-name	不保存或恢复原文件的文件名和时间戳
-N --name	保存或恢复原文件的文件名和时间戳
-q --quiet	禁止警告
-r --recursive	对目录进行递归操作
-S .suf --suffix .suf	使用自定义的压缩文件后缀名
-t --test	检测压缩包的完整性
-v --verbose	冗余模式
-V --version	显示版本号
-1 --fast	快速压缩
-9 --best	最佳压缩（压缩率最高）
filename	指定文件名



7.3.1 常规压缩与解压缩操作

可以直接使用 `gzip` 命令对文件进行压缩，但不可以对目录进行压缩。压缩后的文件以原文件名为主文件名，以“.gz”为后缀，同时会自动删除原文件。例如，当前目录下有 `file1`、`file2` 文件，对 `file1` 文件进行压缩，命令行如下所示。

```
# ls
file1 file2
# gzip file1
# ls
file1.gz file2
```

也可以同时对多个文件进行压缩，如下所示。

```
# ls
file1.gz file2 file3
# gzip file2 file3
# ls
file1.gz file2.gz file3.gz
```

解压使用 `gunzip` 命令，解压后原压缩文件会被自动删除。例如，对 `file1.gz` 文件解压缩，命令如下所示。

```
# gunzip file1.gz
# ls
file1 file2.gz file3.gz
```

也可以对多个文件一起解压缩。例如，对 `file2.gz`、`file3.gz` 解压缩，命令如下所示。

```
# gunzip file2.gz file3.gz
# ls
file1 file2 file3
```

`gzip` 命令使用选项“-d”，也可以完成对文件的解压缩，如下所示。

```
# ls
file1.gz file2.gz file3.gz
# gzip -d *.gz
# ls
file1 file2 file3
```

7.3.2 查看.gz 压缩包中的内容

使用 `gzip` 命令，配合“-l”选项，可以在.gz 压缩包未解压状态下查看包的基本信息，包括已压缩字节、未压缩字节、压缩率、压缩前文件名等。

例如，对当前目录下的.gz 文件进行查询，显示包的基本信息，命令如下所示。

```
# ls
file1.gz file2.gz file3.gz
# gzip -l *
gzip: directory1 is a directory -- ignored
      compressed      uncompressed  ratio uncompressed_name
          32             6 -33.3% file1
          26             0   0.0% file2
          26             0   0.0% file3
          84             6 -900.0% (totals)
```

7.3.3 自定义压缩包后缀

使用 `gzip` 压缩文件后，默认生成的后缀为“.gz”，但配合使用选项“-S”，可以自定义压

缩文件的后缀名。

例如，对当前目录下的 file1 文件进行压缩，且自定义后缀名为“.teacher”，命令如下所示。

```
# ls
file1 file2 file3
# gzip -S .teacher file1
# ls
file1.teacher file2 file3
```

解压时也必须说明自定义的后缀名，命令如下所示。

```
# gunzip -S .teacher file1.teacher
# ls
file1 file2 file3
```

也可以使用 gzip 命令解压，如下所示。

```
#gzip -S .teacher -d file1.teacher
```

7.3.4 zcat 命令：显示压缩文件内容

zcat 命令用于查看压缩文件的内容，可以将压缩文件的内容输出到标准输出设备。该命令的语法格式如下所示。

```
zcat [ -f ][ -h ][ -L ][ -V ] [ name ]
```

其中各选项的具体说明见表 7-4。

表 7-4 zcat 选项说明

选 项	说 明
-f --force	对输出文件强制写覆盖并对链接文件进行压缩
-h --help	显示帮助信息
-L --license	显示软件许可权
-V --version	显示版本号
name	指定文件名

例如，显示压缩文件 vip.gz 文件的内容，命令如下所示。

```
# zcat vip.gz //显示压缩文件 vip.gz 的内容
This is ../info/vip, produced by makeinfo version 4.0 from vip.texi.

INFO-DIR-SECTION Emacs
START-INFO-DIR-ENTRY
* VIP: (vip).      An older VI-emulation for Emacs.
END-INFO-DIR-ENTRY
... ..
```

7.4 zip 与 unzip 命令

zip 是 Linux 下另一个被广泛使用的压缩程序。使用 zip 压缩后的文件名后缀是“.zip”，解压缩时使用 unzip 命令。



7.4.1 使用 zip 生成压缩包

zip 格式的文件在 Windows 系统中也被广泛使用，Windows 下著名的 winzip 程序就是用来专门处理 zip 文件的工具。zip 命令的语法格式如下。

```
zip [-f][-u][-d][-r][-l][-9][-q][-v][-z][-@][-x][-i][-j][-A][-T]
[-y][-e][-k][-m][-n][-t mmddyyyy] [-n]
[zip 文件] [源文件...]
```

如果没有指定源文件和 zip 文件，系统默认对标准输入进行压缩，并从标准输出导出压缩。zip 的选项说明见表 7-5。zip 文件如果没有指定后缀，则默认后缀为“.zip”。

表 7-5 zip 命令的选项说明

选 项	说 明
-f	仅对变化的文件进行更新
-u	只有文件被改动或有新文件时才进行升级
-d	删除 zip 文件中的条目
-r	子目录递归
-l	将 LF 转换为 CR LF
-9	最佳压缩（压缩率最高）
-q	静默模式
-v	冗余模式
-z	给 zip 压缩文件加注释
-@	从标准输入读入文件名
-x	不包括列出的文件
-i	包括列出的文件
-j	不包括子目录
-A	调整为自展开的 exe 文件
-T	检测文件的完整性
-y	对于链接文件仅保存符号链接
-e	加密
-k	用 8.3 格式转换 zip 文件中的文件名，并且文件名改为大写
-m	创建压缩文件后，自动删除源文件
-n	直接放入压缩包，不进行压缩

例如，对当前目录下的文件 file1、file2、file3 进行压缩，压缩后生成 file.zip 文件，执行命令如下所示。

```
# ls
directory1 file1 file2 file3
# zip file.zip file1 file2 file3
adding: file1 (deflated 74%)
adding: file2 (deflated 40%)
adding: file3 (deflated 39%)
```



```
# ls
directory1 file1 file2 file3 file.zip
```

在对目录进行压缩时，如果目录中存在子目录，则 `zip` 默认只将目录名放入压缩包，并不进入子目录进行压缩，如下所示。

```
# ls
directory1 file1 file2 file3
# zip filezip.zip *
  adding: directory1/ (stored 0%)
  adding: file1 (deflated 74%)
  adding: file2 (deflated 40%)
  adding: file3 (deflated 39%)
# ls
directory1 file1 file2 file3 filezip.zip
```

```
# ls -l
drwxr-xr-x 2 root root 4096 08-08 20:17 directory1
-rw-r--r-- 1 root root  73 08-09 02:33 file1
-rw-r--r-- 1 root root  72 08-09 02:33 file2
-rw-r--r-- 1 root root  77 08-09 02:34 file3
-rw-r--r-- 1 root root 623 08-09 02:48 filezip.zip
```

如果希望连同子目录中的文件一并进行压缩，需使用选项 “-r”。接上例，对当前目录中的所有文件、子目录及子目录下的文件一并进行压缩，命令如下所示。

```
# ls
directory1 file1 file2 file3
# zip -r filezip.zip *
  adding: directory1/ (stored 0%)
  adding: directory1/file4 (deflated 22%)
  adding: file1 (deflated 74%)
  adding: file2 (deflated 40%)
  adding: file3 (deflated 39%)
# ls -l
drwxr-xr-x 2 root root 4096 08-08 20:17 directory1
-rw-r--r-- 1 root root  73 08-09 02:33 file1
-rw-r--r-- 1 root root  72 08-09 02:33 file2
-rw-r--r-- 1 root root  77 08-09 02:34 file3
-rw-r--r-- 1 root root 800 08-09 02:49 filezip.zip
```

如果只需要对指定目录中的文件进行压缩，忽略子目录，需要使用选项 “-j”，命令如下所示。

```
# ls
directory1 file1 file2 file3
# zip -j filezip.zip *
  adding: file1 (deflated 74%)
  adding: file2 (deflated 40%)
  adding: file3 (deflated 39%)
# ls -l
drwxr-xr-x 2 root root 4096 08-08 20:17 directory1
-rw-r--r-- 1 root root  73 08-09 02:33 file1
-rw-r--r-- 1 root root  72 08-09 02:33 file2
-rw-r--r-- 1 root root  77 08-09 02:34 file3
-rw-r--r-- 1 root root 491 08-09 02:49 filezip.zip
```

有时，`zip` 压缩文件需要在其他平台上解压缩，为了保持和其他平台的兼容性，最好使用比较通用的 8.3 命名格式来重新对文件命名，然后再进行压缩。可以使用选项 “-k” 完成转



换。选项“-k”除了将文件名改为 8.3 格式外，还会将文件名调整为大写。该选项只影响压缩包内的文件名，对于源文件名不做更改，如下所示。

```
# ls
directory1 file0000000000003.document file1 file2
# zip -k file3.zip *
adding: DIRECTOR/ (stored 0%)
adding: FILE0000.DOC (deflated 39%)
adding: FILE1 (deflated 74%)
adding: FILE2 (deflated 40%)
# ls
directory1 file0000000000003.document file1 file2 file3.zip
```



8.3 格式是文件名的常用命名方法之一。其文件名分为主文件名和扩展文件名两部分，主文件名最长 8 字符，扩展文件名最长 3 字符。通常也把文件的扩展文件名称为文件的后缀。

对于 JPG、GIF、RMVB、MPEG 等格式的媒体文件，由于其自身已经采用了一定的压缩技术，如果再使用 zip 进行压缩，效果不大而且浪费时间。可以使用“-n”选项，对该类文件只打包，不进行压缩。文件类型之间需用冒号分隔。例如，对当前目录下的文件进行压缩，忽略对“.JPG”、“.GIF”格式文件的压缩，命令行如下所示。

```
# ls
directory1 DSC03191.JPG DSC03192.JPG DSC06479.JPG DSC09.GIF file1
# zip -n .JPG : .GIF files.zip *
zip warning: name not matched: .GIF
zip warning: name not matched: files.zip
adding: directory1/ (stored 0%)
adding: DSC03191.JPG (stored 0%) //直接打包不压缩
adding: DSC03192.JPG (stored 0%) //直接打包不压缩
adding: DSC06479.JPG (stored 0%) //直接打包不压缩
adding: DSC09.GIF (deflated 74%) //直接打包不压缩
adding: file1 (deflated 40%)
```

选项“-@”允许从标准输入设备读入文件名。如果需要压缩的文件很多，无法在一行列出所有文件名，可以使用该选项，输入完毕按 Ctrl+d 组合键结束，如下所示。

```
# zip -@ filezip.zip
file1
file2
file3
adding: file1 (deflated 74%)
adding: file2 (deflated 40%)
adding: file3 (deflated 39%)
```

如果某一目录下需要压缩的文件占绝大多数，只需在选择源文件时使用选项“-x”，将个别不需要压缩的文件排除即可，如下所示。

```
# ls
file1 file2 file3
# zip filezip.zip * -x file3 //跳过 file3
adding: file1 (deflated 74%)
adding: file2 (deflated 40%)
```

如果需要压缩某一日期之后的文件，可以使用选项“-t”对日期进行设定。日期格式为 mmddyy，其中 mm 表示月份，dd 表示日，yy 表示年的后两位。例如，在当前目录下，对 2007

年8月9日之后创建的文件进行压缩, 可以使用如下命令。

```
#zip -t 080907 files.zip *
```

选项“-9”表示最佳压缩, 即压缩率最高。zip 的压缩率共分9个等级, 1级最低, 即压缩速度快, 但节省存储空间少; 9级最高, 压缩速度慢, 但可节省较多存储空间。zip 默认的压缩等级为6, 下面是对不同压缩等级、压缩率的比较。

```
# zip -9 test1.zip file1
adding: file1 (deflated 40%)
# zip -1 test2.zip file1
adding: file2 (deflated 36%)
```

对于软链接文件, zip 在压缩前会自动寻找该链接指向的源文件, 并对源文件进行压缩, 压缩后链接关系不再存在, 如下所示。

```
# ls -l
-rw-r--r-- 1 root root 25 08-09 04:07 file1
lrwxrwxrwx 1 root root 5 08-09 04:20 file1_ln -> file1
# zip file1.zip file1 file1_ln //对 file1 及 file1 的链接文件
file1_ln 进行压缩
adding: file1 (deflated 40%)
adding: file1_ln (deflated 40%)
# unzip file1.zip //对 file1.zip 文件解压
Archive: file1.zip
inflating: file1
inflating: file1_ln
# ls
file1 file1_ln file1.zip
# ls -l
-rw-r--r-- 1 root root 25 08-09 04:07 file1
-rw-r--r-- 1 root root 25 08-09 04:07 file1_ln //已不再是链接文件
-rw-r--r-- 1 root root 298 08-09 04:21 file1.zip
```

如果只希望对链接文件自身进行压缩, 解压后仍然保持其链接属性, 可以使用选项“-y”, 命令如下所示。

```
# zip -y file1.zip file1 file1_ln
adding: file1 (deflated 40%)
adding: file1_ln (stored 0%)
# unzip file1.zip
Archive: file1.zip
inflating: file1
linking: file1_ln -> file1
finishing deferred symbolic links: //自动恢复链接关系
file1_ln -> file1
# ls -l
-rw-r--r-- 1 root root 25 08-09 04:07 file1
lrwxrwxrwx 1 root root 5 08-09 04:27 file1_ln -> file1 //解压后仍是链接文件
-rw-r--r-- 1 root root 288 08-09 04:25 file1.zip
```

在用 zip 生成压缩文件的同时, 源文件通常保持不变。使用选项“-m”, 可以在生成压缩文件后, 自动删除源文件, 进一步节省存储空间。例如, 对当前目录下的 file1、file2 文件进行压缩, 生成 file.zip 文件, 同时删除 file1、file2 文件, 命令如下所示。

```
# ls
file1 file2 file3
# zip -m file.zip file1 file2
adding: file1 (deflated 74%)
```



```
adding: file2 (deflated 40%)
# ls -l
-rw-r--r-- 1 root root 77 08-09 02:34 file3
-rw-r--r-- 1 root root 800 08-09 02:49 file.zip //文件 file1、file2 已删除
```

7.4.2 使用 unzip 进行解压缩

在 Linux 系统中，使用 unzip 命令来对 zip 压缩文件进行解压缩。unzip 命令的语法格式如下。

```
unzip [-Z] [-l] [-L] [-t] [-x] [filename]
```

其中各选项的具体说明如表 7-6 所示。

表 7-6 unzip 命令的选项说明

选 项	说 明
-Z	以 zipinfo 格式显示压缩文件内的信息，包括文件数目、已压缩和未压缩字节数、压缩比等
-l	以简略格式列出压缩文件的基本信息，包括文件名、修改日期时间、未压缩文件的大小等。如果存在注释，也一并显示
-L	如果压缩文件是从不区分大小写的文件系统中创建的，例如 MS-DOS 操作系统，则将文件名全部改为小写，并添加 “^” 前缀
-t	通过 CRC 校验对 zip 压缩包进行检测
-x	用于排除解压缩包中的特定文件
filename	指定.zip 文件名

例如，对当前目录下的 file.zip 压缩文件进行解压缩，命令行如下所示。

```
# unzip file1.zip //对 file1.zip 文件解压
Archive: file1.zip
  inflating: file1
  inflating: file2
# ls
file1 file2 file1.zip
```

如果希望排除压缩包中的某个特定文件，可以使用选项 “-x”，如下所示。

```
# ls
file1.zip
# unzip file1.zip -x file2 //对 file1.zip 文件解压，但不对其中包含的 file2 进行解压
Archive: file1.zip
  inflating: file1
# ls
file1 file1.zip
```

如果使用 “-l” 选项，可以以简略格式查看压缩包的内容，命令行如下。

```
# unzip -l files.zip
Archive: files.zip
Length Date Time Name
-----
0 08-08-07 20:17 directory1/
2130896 08-02-07 05:58 DSC03191.JPG
2145453 08-02-07 05:58 DSC03192.JPG
1305640 08-02-07 05:58 DSC06479.JPG
73 08-09-07 02:33 DSC09.GIF
25 08-09-07 03:33 file1
```



```
0 08-09-07 04:26 test/
5582827 08-09-07 03:33 :.zip
-----
11164914 8 files
```

7.4.3 zipinfo 命令：显示压缩文件信息

zipinfo 命令可以用来显示.zip 压缩文件的信息。该命令的语法格式如下所示。

```
zipinfo [-l][-2][-s][-m][-l][-v][-h][-T] file[.zip] [-x xlist...]
```

其中各选项的具体说明如表 7-7 所示。

表 7-7 zipinfo 命令的选项说明

选 项	说 明
-l	只显示文件名。每行显示一个文件名。不包括标题、注释、尾注等
-2	显示文件名。每行显示一个文件名，且包括标题、注释、尾注等
-h	显示标题行，包括文件名、大小、文件数目
-T	显示时间设置
-v	以冗余模式显示 zip 文件信息
-z	显示注释
-x xfile(s)	匹配格式的文件不处理
-s	以短 UNIX “ls -l” 格式列出 zip 文件信息
-m	以中等 UNIX “ls -l” 格式列出 zip 文件信息
-l	以长 UNIX “ls -l” 格式列出 zip 文件信息
-T	以短十进制格式 (yyymmdd.hhmmss) 显示文件的时间、日期
file	指定查询的文件

例如，通过命令 zipinfo，显示压缩文件 inter.zip 的信息，如下所示。

```
# ls
backup backup1 backup3
# zip file.zip backup backup1 backup3
adding: backup (deflated 10%)
adding: backup1 (deflated 10%)
adding: backup3 (deflated 10%)
# ls
backup backup1 backup3 file.zip
# zipinfo file.zip //显示压缩文件的信息
Archive: file.zip 581 bytes 3 files
-rw-r--r-- 2.3 unx 70 tx defN 9-Aug-08 08:32 backup
-rw-r--r-- 2.3 unx 70 tx defN 9-Aug-08 08:32 backup1
-rw-r--r-- 2.3 unx 70 tx defN 9-Aug-08 08:32 backup3
3 files, 210 bytes uncompressed, 189 bytes compressed: 10.0%
```

例如，通过命令 “zipinfo -v”，显示压缩 file.zip 文件中每个文件的详细信息，如下所示。

```
# zipinfo -v file.zip //显示每个文件的信息
Archive: file.zip 581 bytes 3 files
End-of-central-directory record:
-----
Actual offset of end-of-central-dir record: 559 (0000022Fh)
```



Expected offset of end-of-central-dir record: 559 (0000022Fh)
(based on the length of the central directory and its expected offset)

This zipfile constitutes the sole disk of a single-part archive; its central directory contains 3 entries. The central directory is 197 (000000C5h) bytes long, and its (expected) offset in bytes from the beginning of the zipfile is 362 (0000016Ah).

There is no zipfile comment. //无 zipinfo 注释

Central directory entry #1:

backup //backup 文件信息

offset of local header from start of archive: 0 (00000000h) bytes
file system or operating system of origin: UNIX
version of encoding software: 2.3
minimum file system compatibility required: MS-DOS, OS/2 or NT FAT
minimum software version required to extract: 2.0
compression method: deflated
compression sub-type (deflation): normal
file security status: not encrypted //文件未加密
extended local header: no
file last modified on (DOS date/time): 2008 Aug 9 08:32:12
file last modified on (UT extra field modtime): 2008 Aug 9 08:32:12 local
file last modified on (UT extra field modtime): 2008 Aug 9 00:32:12 UTC
32-bit CRC value (hex): 93fdc581
compressed size: 63 bytes
uncompressed size: 70 bytes
length of filename: 6 characters //文件名长6字符
length of extra field: 13 bytes
length of file comment: 0 characters
disk number on which file begins: disk 1
apparent file type: text
UNIX file attributes (100644 octal): -rw-r--r--
MS-DOS file attributes (00 hex): none

The central-directory extra field contains:

- A subfield with ID 0x5455 (universal time) and 5 data bytes.
The local extra field has UTC/GMT modification/access times.
- A subfield with ID 0x7855 (UNIX UID/GID) and 0 data bytes.

There is no file comment. //无文件注释

Central directory entry #2:

backup1 //backup1 文件信息

offset of local header from start of archive: 120 (00000078h) bytes
file system or operating system of origin: UNIX
version of encoding software: 2.3
minimum file system compatibility required: MS-DOS, OS/2 or NT FAT
minimum software version required to extract: 2.0
compression method: deflated


```

compression sub-type (deflation):          normal
file security status:                      not encrypted
extended local header:                     no
file last modified on (DOS date/time):      2008 Aug 9 08:32:12
file last modified on (UT extra field modtime): 2008 Aug 9 08:32:12 local
file last modified on (UT extra field modtime): 2008 Aug 9 00:32:12 UTC
32-bit CRC value (hex):                    93fdc581
compressed size:                           63 bytes      //压缩大小为 63 字节
uncompressed size:                         70 bytes
length of filename:                        7 characters  //文件名长度为 7 个字符
length of extra field:                     13 bytes
length of file comment:                    0 characters
disk number on which file begins:           disk 1
apparent file type:                        text
UNIX file attributes (100644 octal):        -rw-r--r--
MS-DOS file attributes (00 hex):            none

```

The central-directory extra field contains:

- A subfield with ID 0x5455 (universal time) and 5 data bytes.
The local extra field has UTC/GMT modification/access times.
- A subfield with ID 0x7855 (UNIX UID/GID) and 0 data bytes.

There is no file comment.

//无文件注释

Central directory entry #3:

backup3

//backup3 文件信息

```

offset of local header from start of archive: 241 (000000F1h) bytes
file system or operating system of origin:    UNIX
version of encoding software:                 2.3
minimum file system compatibility required:    MS-DOS, OS/2 or NT FAT
minimum software version required to extract:  2.0
compression method:                           deflated
compression sub-type (deflation):              normal
file security status:                          not encrypted
extended local header:                         no
file last modified on (DOS date/time):          2008 Aug 9 08:32:12
file last modified on (UT extra field modtime): 2008 Aug 9 08:32:12 local
file last modified on (UT extra field modtime): 2008 Aug 9 00:32:12 UTC
32-bit CRC value (hex):                        93fdc581
compressed size:                              63 bytes      //压缩大小为 63 字节
uncompressed size:                           70 bytes      //未压缩大小为 70 字节
length of filename:                           7 characters  //文件名长度为 7 个字符
length of extra field:                        13 bytes
length of file comment:                       0 characters
disk number on which file begins:              disk 1
apparent file type:                           text
UNIX file attributes (100644 octal):            -rw-r--r--
MS-DOS file attributes (00 hex):                none

```

The central-directory extra field contains:

- A subfield with ID 0x5455 (universal time) and 5 data bytes.
The local extra field has UTC/GMT modification/access times.



- A subfield with ID 0x7855 (UNIX UID/GID) and 0 data bytes.

There is no file comment. //无注释

使用 `unzip` 命令配合 “-Z” 选项，可以依据 `zipinfo` 格式查看压缩包的内容，执行命令如下所示。

```
# unzip -Z files.zip
Archive: files.zip 11159281 bytes 8 files
drwxr-xr-x 2.3 unx 0 bx stor 8-Aug-07 20:17 directory1/
-rw-r--r-- 2.3 unx 2130896 bx defN 2-Aug-07 05:58 DSC03191.JPG
-rw-r--r-- 2.3 unx 2145453 bx defN 2-Aug-07 05:58 DSC03192.JPG
-rw-r--r-- 2.3 unx 1305640 bx defN 2-Aug-07 05:58 DSC06479.JPG
-rw-r--r-- 2.3 unx 73 tx defN 9-Aug-07 02:33 DSC09.GIF
-rw-r--r-- 2.3 unx 25 tx defN 9-Aug-07 03:33 file1
drwxr-xr-x 2.3 unx 0 bx stor 9-Aug-07 04:26 test/
-rw-r--r-- 2.3 unx 5582827 bx stor 9-Aug-07 03:33 :.zip
8 files, 11164914 bytes uncompressed, 11158237 bytes compressed: 0.1%
//由于包含多幅图片，压缩率为 0.1%
```

7.5 bzip2 与 bunzip2 命令

与 `gzip` 类似，`bzip2` 也是一种常用的压缩工具。`bzip2` 压缩后的文件一般具有后缀“`.bz2`”，可以使用 `bunzip` 对其解压缩。

7.5.1 使用 bzip2 生成压缩包

`bzip2` 命令使用 Burrows-Wheeler 块分类文本压缩算法压缩文件，并用 Huffman 进行编码。该压缩算法比传统的 LZ77/LZ78-based 压缩算法效果更好，接近 PPM 统计压缩算法。`bzip2` 不具有将多个文件或目录进行打包的功能，只能单纯地对文件进行压缩。在产生后缀为“`.bz2`”的压缩文件后，`bzip2` 默认将自动删除源文件。`bzip2` 命令的语法格式如下所示。

```
bzip2 [-h --help] [-d --decompress] [-z --compress] [-k --keep] [-f --force] [-t
--test] [-c --stdout] [-q --quiet]
[-v --verbose] [-L --license] [V --version] [-s --small] [-1 .. -9] [--fast] [--best]
```

`bzip2` 的选项说明如表 7-8 所示。

表 7-8 bzip2 选项说明

选 项	说 明
-h --help	显示帮助信息
-d --decompress	强制解压缩
-z --compress	强制压缩
-k --keep	保留源文件
-f --force	允许写覆盖
-t --test	检测压缩文件的完整性，但是不解压它们
-c --stdout	输出到标准输出设备
-q --quiet	静默模式，不输出警告信息

续表

选 项	说 明
-v --verbose	冗余模式。运行时，输出详细的处理信息
-L --license	显示软件许可协议
-V --version	显示软件版本
-s --small	使用较小的内存
-l .. -9	设置压缩等级 1~9
--fast	快速压缩（等级 1），当压缩时将块的大小设置为 100KB
--best	最佳压缩（等级 9），当压缩时将块的大小设置为 900KB

例如，压缩文件 abc.doc，压缩后生成文件 abc.doc.bz2，原文件 abc.doc 不再存在，命令如下所示。

```
# bzip2 abc.doc //压缩文件 abc
```

例如，压缩 abc.doc 文件，将产生的压缩数据输出到标准输出，命令如下所示。

```
# bzip2 -c abc //压缩文件 abc，将产生的压缩数据输出到标准输出
```

例如检查压缩文件 abc.doc.bz2 的完整性，命令如下所示。

```
# bzip2 -t abc.doc.bz2 //检查文件 abc.doc
```

例如，对当前目录下的文件 file1、file2、file3 进行压缩，命令如下所示。

```
#ls
file1 file2 file3
# bzip2 *
# ls
file1.bz2 file2.bz2 file3.bz2
```

可以看到，在生成 file1.bz2、file2.bz2 和 file3.bz2 压缩文件后，源文件已经被自动删除。在压缩过程中没有任何提示，如果希望看到压缩过程中的提示信息，可以使用选项 “-v”，如下所示。

```
# ls
file1 file2 file3
# bzip2 -v *
file1: 0.681:1, 11.745 bits/byte, -46.81% saved, 47 in, 69 out.
file2: 1.170:1, 6.839 bits/byte, 14.52% saved, 62 in, 53 out.
file3: 1.033:1, 7.742 bits/byte, 3.23% saved, 62 in, 60 out.
# ls
file1.bz2 file2.bz2 file3.bz2
```

如果希望在生成压缩文件后，源文件不被删除，可以使用带选项 “-k” 的 bzip2 命令，如下所示。

```
#ls
file1 file2 file3
# bzip2 -k *
# ls
file1 file2 file3
file1.bz2 file2.bz2 file3.bz2
```

7.5.2 使用 bunzip2 进行解压缩

对 bzip2 文件解压缩可以使用 bunzip2 命令。bunzip2 命令相当于 “bzip2 -d”，将解压所



有指定的文件。解压缩后，原 bzip2 文件会自动被删除。在解压时，不是被 bzip2 创建的文件将被忽略，并产生警告信息。文件被解压后的形式如下。

- filename.bz2 变成 filename。
- filename.bz 变成 filename。
- filename.tbz2 变成 filename.tar。
- filename.tbz 变成 filename.tar。
- anyothername 变成 anyothername.out。

bunzip 命令的语法格式如下所示。

```
bunzip2 [-h --help] [-d --decompress] [-z --compress] [-k --keep] [-f --force] [-t --test] [-c --stdout] [-q --quiet]
[-v --verbose] [-L --license] [-V --version] [-s --small] [-1 .. -9] [--fast] [--best]
```

其中各选项的说明如表 7-9 所示。

表 7-9 bunzip2 命令的选项说明

选 项	说 明
-f --force	强制解压文件，覆盖当前存在的同名的文件
-k --keep	解压后保留源文件，默认情况下不保留源文件
-s --small	减少程序执行时占用的内存（大约 2500KB）
-v --verbose	运行时，输出详细的处理信息
-c --stdout	将解压后的数据输出到标准输出
-L --license -V --version	显示版本信息
-h, --help	显示帮助信息

例如，解压 abc.doc.bz2 文件，执行命令如下所示。

```
# bunzip2 abc.doc.bz2 //解压文件 abc.bz2
```

解压之后，将生成文件 abc.doc，原文件 abc.doc.bz2 将自动被删除。

例如，解压文件 abc.doc.bz2，生成文件 abc.doc，文件 abc.doc 的内容为“i am a student”（第一行），“i am from Beijing”（第二行）。使用“-c”选项后，系统将解压后的数据直接输出到标准输出设备上，命令执行过程如下所示。

```
# bunzip2 -c abc.doc.bz2 //解压文件 abc..doc.bz2 并将解压后的数据输出到标准输出
i am a student
i am from Beijing
```

例如，对当前目录下的所有.bz2 压缩文件进行解压，命令如下所示。

```
# ls
file1.bz2 file2.bz2 file3.bz2
# bunzip2 *
# ls
file1 file2 file3
```



解压缩也可以使用带选项“-d”的 bzip2 命令。例如，对当前目录下的 file1 文件进行压缩和解压缩，命令如下所示。


```
# bzip2 file1
# ls
file1.bz2
# bzip2 -d file1.bz2
# ls
file1
```

7.5.3 bzip2recover 命令：恢复损坏的 bzip2 文件

bzip2recover 命令可以修复损坏的 bz2 文件的数据。如果 bzip2 文件有损坏，可以试着用 bzip2recover 命令进行恢复。该命令的语法格式如下所示。

```
bzip2recover [filename]
```

其中，选项 filename 指定进行处理的文件名。例如，对 test 目录下的压缩文件 abc.doc.bz2 进行修复，命令如下所示。

```
# bzip2recover /test/abc.doc.bz2 //修复.bz2 文件
bzip2recover 1.0.2: extracts blocks from damaged .bz2 files.
bzip2recover: searching for block boundaries ...
    block 1 runs from 80 to 421
bzip2recover: splitting into blocks
    writing block 1 to 'rec00001abc.bz2' ...
bzip2recover: finished
```

例如，对当前目录下的 file.bz2 文件进行修复，命令如下所示。

```
# ls
file.bz2
# bzip2recover file.bz2
bzip2recover 1.0.3: extracts blocks from damaged .bz2 files.
bzip2recover: searching for block boundaries ...
    block 1 runs from 80 to 358
bzip2recover: splitting into blocks
    writing block 1 to 'rec00001file.bz2' ...
bzip2recover: finished
# ls
file.bz2  rec00001file.bz2 //生成了恢复文件 rec00001file.bz2
# bunzip2 rec00001file.bz2
# ls
file.bz2  rec00001file
# cat rec00001file // rec00001file 中记录了 file 文件的内容片段
aaaaaaaaaaaaaaaaaaaaaaaaaaaaa
bbbbbbbbbbbbbbbbbbbbbbbbbbbbb
cccccccccccccccccccccccccccc
ddddddddddddddddddddddddddddd
eeeeeeeeeeeeeeeeeeeeeeeeeeeee
fffffffffffffffffffffffffffffff
```

可以看出，bzip2recover 命令实际是对需要修复的文件按块进行搜索，将每一块保存到一个文件名为“recoverXXXXX.bz2”的文件中。每一个这样的文件中，都记录了源文件内容的一些片段。

7.6 dump 与 restore 命令

dump 命令与 tar 类似，长久以来一直是 UNIX 系统中最为常用的备份工具之一。与 tar



不同，`dump` 支持分卷和增量备份（所谓增量备份是指仅备份最近一次备份以来修改过的文件，也称差异备份）。`restore` 命令用来恢复已备份的文件，可以从备份磁带或由 `dump` 生成的备份文件中恢复原文件。

7.6.1 使用 `dump` 制作备份

当用 `dump` 命令制作一个文件系统的备份时，必须通过选项来指定 `dump` 的级别、备份的介质及需要备份的文件系统。`dump` 命令的语法格式如下所示。

```
dump [-t date] [-u] [-s feet] [-n] [-d density] [-f file] [-b kbperdump] [-B records] [0-9] [-W] [-w]
```

表 7-10 列出了 `dump` 命令常用的选项说明。

表 7-10 dump 命令的选项说明

选 项	说 明
-t date	指定增量备份的基准时间，在这个时间之后修改或增加的任何文件将得到备份。该选项将使 <code>dump</code> 忽略 <code>/etc/dumpdates</code> 文件中的设定值。 <code>date</code> 的格式可以参考手册中的 <code>ctime</code>
-u	将备份的细节记录在 <code>/etc/dumpdates</code> 文件中
-s feet	指定磁带以英尺为单位长度
-n	当 <code>dump</code> 操作需要用户参与，例如更换磁带时， <code>dump</code> 就向该用户组的所有成员发出提示信息
-d density	设定磁带的密度，默认每英寸 1600bit
-f file	指定 <code>dump</code> 所写入的文件或设备
-b kbperdump	指定每个 <code>dump</code> 记录的 KB 数
-B records	指定每个卷包含的 <code>dump</code> 记录数，该记录数表示一盘磁带能装载的数据容量
0-9	<code>dump</code> 级别，分为 0~9 级，其中级别 0 表示备份所有文件，1~9 表示增量备份。对于不为 0 的备份等级，表示仅对上次同等级或更低级别的备份后修改过的文件进行备份
-W	列出需要备份的文件系统
-w	列出需要备份的单个文件

例如，使用 `dump` 命令将第一个 IDE 硬盘上第一个文件系统 (`/dev/hda1`) 完全备份 (level 0) 到磁带上，命令如下所示。

```
#dump -ouf /dev/tape /dev/hda1
```

使用 `dump` 命令也可以直接实现对目录的备份。例如，对当前目录下的 `zip` 子目录进行备份，备份文件名为 `dumpzip`，命令如下所示。

```
# dump -f dumpzip zip/
DUMP: Date of this level dump: Thu Aug 9 09:18:33 2007
DUMP: Dumping /dev/mapper/VolGroup00-LogVol100 (/ (dir myshare/zip)) to dumpzip
DUMP: Label: none
DUMP: Writing 10 Kilobyte records
DUMP: mapping (Pass I) [regular files]
DUMP: mapping (Pass II) [directories]
DUMP: estimated 522 blocks.
DUMP: Volume 1 started with block 1 at: Thu Aug 9 09:18:34 2007
DUMP: dumping (Pass III) [directories]
DUMP: dumping (Pass IV) [regular files]
DUMP: Closing dumpzip
```



```
DUMP: Volume 1 completed at: Thu Aug 9 09:18:35 2007
DUMP: Volume 1 560 blocks (0.55MB)
DUMP: Volume 1 took 0:00:01
DUMP: Volume 1 transfer rate: 560 kB/s
DUMP: 560 blocks (0.55MB) on 1 volume(s)
DUMP: finished in less than a second
DUMP: Date of this level dump: Thu Aug 9 09:18:33 2007
DUMP: Date this dump completed: Thu Aug 9 09:18:35 2007
DUMP: Average transfer rate: 560 kB/s
DUMP: DUMP IS DONE
```

dump 能够备份文件系统中的所有文件（全备份），也可以有选择地备份最近修改过的文件，具有增量备份功能。增量备份可以节省大量的时间和系统资源，使用 dump 命令中的级别设置可以很好地完成增量备份。表 7-11 列出了一个 dump 等级设置的标准范例。

表 7-11 dump 等级设置范例

星 期	dump 等级	星 期	dump 等级
星期日	等级 0，全备份	星期一	等级 9，增量式备份
星期二	等级 8，增量式备份	星期三	等级 7，增量式备份
星期四	等级 6，增量式备份	星期五	等级 5，增量式备份
星期六	等级 4，增量式备份		

该时间表中，从星期日开始，每一次的增量备份都比前一次大，所有前一次增量备份中的文件以及自那以后修改过的文件都需要进行备份。整体运行的效果是从星期日开始，修改过的所有文件在每天的增量备份中都得到了备份。当出现磁盘损坏等需要恢复备份的情况时，按该时间表方法可以节省大量时间和精力。例如，如果周三发生了磁盘损坏，需要恢复备份，只需恢复上周日全备份数据和本周二的增量备份数据即可。

7.6.2 使用 restore 恢复备份

restore 命令用来恢复已备份的文件，可以从备份磁带或由 dump 生成的备份文件中恢复原文件。restore 的命令格式与 dump 相似，如下所示。

```
restore [-r][ -C][ -x filelist][ -T file][ -R][ -i][ -y][ -v][ -b blocksize][ -D
filesystem][ -T directory][ -N][ -m][ -f file][ -t]
```

其中各选项的具体说明如表 7-12 所示。

表 7-12 restore 命令选项说明

选 项	说 明
-r	恢复整个 dump 备份
-C	将 dump 备份的内容同硬盘上的文件进行比较，通常用来检查恢复是否成功
-x filelist	仅从 dump 备份中恢复指定的文件或目录，其中 filelist 是待提取的文件或目录列表
-T file	列出 dump 备份文件的内容。如果指定了文件或目录，则仅列出与之相关的内容
-R	从一组备份磁带中的某一特定磁带开始恢复
-i	以交互方式恢复备份
-y	如果 restore 在恢复备份时遇到损坏的数据块，不进行提示继续恢复



续表

选 项	说 明
-v	冗余模式，每恢复一个文件都会显示相关信息
-b blocksize	指定 dump 块大小
-D filesystem	当使用-C 选项时，指定被比较的文件系统的名字
-T directory	指定 restore 使用的临时文件
-N	不释放文件，仅打印文件名
-m	根据 i 节点数释放，而不是根据文件名
-f file	从指定文件读入备份文件，如果通过网络进行工作，格式为“host:file”或“user@host:file”
-t	列出备份文件中的文件名



restore 命令执行 dump 命令的相反操作。restore 命令可以恢复一个文件系统的完全备份，单个文件或子树也可以从全部或部分的备份中恢复。restore 命令还可以通过网络工作，具体的细节请参看“-f”选项。

例如，通过命令 dump，将/test 目录下的数据备份到文件 mydump，然后通过命令 restore，查看备份文件列表，命令执行过程如下所示。

```
# dump -0 -f mydump /test //对 test 目录进行备份
DUMP: Date of this level 0 dump: Sat Aug 9 10:00:54 2008
DUMP: Dumping /dev/sda2 (/ (dir test)) to mydump
DUMP: Added inode 8 to exclude list (journal inode)
DUMP: Added inode 7 to exclude list (resize inode)
DUMP: Label: /
DUMP: mapping (Pass I) [regular files]
DUMP: mapping (Pass II) [directories]
DUMP: estimated 5165 tape blocks.
DUMP: Volume 1 started with block 1 at: Sat Aug 9 10:00:54 2008
DUMP: dumping (Pass III) [directories]
DUMP: dumping (Pass IV) [regular files]
DUMP: Closing mydump
DUMP: Volume 1 completed at: Sat Aug 9 10:00:55 2008
DUMP: Volume 1 5240 tape blocks (5.12MB)
DUMP: Volume 1 took 0:00:01
DUMP: Volume 1 transfer rate: 5240 kB/s
DUMP: 5240 tape blocks (5.12MB) on 1 volume(s)
DUMP: finished in 1 seconds, throughput 5240 kBytes/sec
DUMP: Date of this level 0 dump: Sat Aug 9 10:00:54 2008 //备份 0 的时间
DUMP: Date this dump completed: Sat Aug 9 10:00:55 2008
DUMP: Average transfer rate: 5240 kB/s //平均传输速率
DUMP: DUMP IS DONE
# restore -t -f mydump //查看备份文件mydump 中的
文件列表
Dump date: Sat Aug 9 10:00:54 2008
Dumped from: the epoch
Level 0 dump of / (dir test) on localhost.localdomain:/dev/sda2
Label: /
      2 .
814424 ./test
```



```

814425      ./test/my.doc
814426      ./test/fonts.dir
814428      ./test/mypaper.bak
814430      ./test/mypaper
814427      ./test/dict
814432      ./test/mypaper.doc
814429      ./test/xx00
814431      ./test/xx01
814433      ./test/patch

```

例如，查看当前目录下备份文件 `dumpzip` 中备份文件的列表，命令如下所示。

```

# restore -t -f dumpzip
Dump date: Thu Aug 9 09:18:33 2007
Dumped from: the epoch
Level 0 dump of / (dir myshare/zip) on localhost.localdomain:/dev/mapper/VolGroup00-
LogVol100
Label: none
      2
390145      ./myshare
390188      ./myshare/zip
390174      ./myshare/zip/file2
390189      ./myshare/zip/file3
390192      ./myshare/zip/file1.bz2
390212      ./myshare/zip/file.bz2
390206      ./myshare/zip/rec00001file1
390213      ./myshare/zip/rec00001file

```



`dump` 和 `restore` 命令虽然多年来被广泛使用，但由于其对正在变化的文件系统（如日志、电子邮件、数据库等）的处理上还存在可靠性和稳定性问题，以及 `dump` 与 `restore` 版本自身的兼容性问题，使得该命令大多仍然应用在磁带等老式介质的备份上。

7.7 compress 与 uncompress 命令

`compress` 命令也是一种常用的压缩工具。使用 `compress` 命令压缩后的文件，一般具有后缀“.Z”。压缩后的文件可以使用 `uncompress` 命令对其解压缩。

7.7.1 使用 compress 生成压缩文件

`compress` 命令通过使用 Lempel-Ziv 算法对指定文件进行压缩。每一个被压缩的文件将具有扩展名“.Z”，并保存与原文件相同的所有权、读写权限和更新时间。`compress` 命令的语法格式如下所示。

```
compress [ -f ] [ -v ] [ -c ] [ -V ] [ -r ] [-d] [ -b bits ] [ name ... ]
```

其中各选项的具体说明如表 7-13 所示。

例如，通过命令 `compress`，压缩文件 `mydump`，产生压缩文件 `mydump.Z`，且原文件 `mydump` 被自动删除。命令执行过程如下所示。



表 7-13 compress 命令选项说明

选 项	说 明
-c	保留原始文件，压缩结果输出到标准输出
-d	解压缩
-f	强制压缩
-r	处理指定目录及其子目录下的所有文件
-v	运行时，输出详细的处理过程，包括每一文件的压缩率
-b bits	按指定数值的压缩效率进行压缩。压缩效率表示压缩的比例，值越大压缩比例越高。数值在 9~16 之间
-V	显示版本信息
name	指定压缩的文件。如果没有指定文件，那么标准输入将被压缩，并且输出到标准输出

```
# ls
mydump
# compress mydump
# ls
mydump.Z test
```

例如，通过命令 `compress` 命令，解压文件 `mydump.Z`，得到文件 `mydump`，且压缩文件 `mydump.Z` 被自动删除。命令执行过程如下所示。

```
# compress -d mydump.Z
# ls
mydump
```

例如，压缩文件 `mydump`，但保留原始文件，并将结果输出到标准输出。命令如下所示。

```
# compress -c mydump
```

例如，按指定压缩比例对文件 `mydump` 进行压缩，命令如下所示。

```
# compress -b 7 mydump
```

例如，强制对目录 `test` 进行压缩，命令如下所示。

```
# compress -rf test/
```

例如，解压目录 `/test` 下的所有压缩文件，命令如下所示。

```
# compress -rfd test/
```

7.7.2 使用 `uncompress` 进行解压缩

`uncompress` 命令用来解压 `.Z` 文件。该命令的语法格式如下所示。

```
uncompress [ -f ] [ -v ] [ -c ] [ -V ] [ name ... ]
```

其中各选项的具体说明如表 7-14 所示。

表 7-14 uncompress 命令选项说明

选 项	说 明
-c	输出结果到标准输出
-f	强制执行
-v	运行时显示详细的处理信息
-V	版本信息
name	指定解压文件

例如，通过命令 `uncompress`，解压文件 `dump.Z`，命令执行过程如下所示。

```
# ls
mydump
# compress mydump                //对 mydump 文件进行压缩
# ls
mydump.Z
# uncompress mydump.Z            //解压 mydump.Z 文件
# ls
mydump
```

7.8 其他常用备份与压缩命令

除了可以使用 `tar`、`gzip/ungzip`、`zip/unzip` 等常用压缩与备份命令外，Linux 还提供了一些针对特殊用途的备份与压缩命令，如 `cpio`、`dd`、`ar` 等。

7.8.1 ar 命令：从归档文件中建立、修改或提取文件

`ar` 命令可以将一组文件合并到一个归档文件（包文件）中，也可以从建立的归档文件中提取文件，归档文件中的文件原有属性不变。该命令的语法格式如下所示。

```
ar [d][m][p][q][r][t][x][a][b][c][f][i][o][s][S][u][v][V]
```

其中各选项的具体说明如表 7-15 所示。

表 7-15 ar 命令选项说明

选 项	说 明
d	删除归档文件中的文件
m	通过移动，改变归档文件中文件的次序
p	显示归档文件中的文件内容
q	向归档文件追加文件
r	向归档文件中添加文件
t	显示归档文件中的文件列表
x	从归档文件中提取文件
a	将新文件追加在归档文件中指定的文件之后
b	将新文件追加在归档文件中指定的文件之前
c	创建归档文件
f	截去过长的归档文件名
i	与 b 选项作用相同，将新文件追加在归档文件中指定的文件之前
o	保留归档文件中文件的时间信息
s	将对象文件索引写入归档文件中
S	不产生归档符号表
u	与 r 一起使用，仅向归档文件中加入更新的文件
v	运行命令时输出详细的信息
V	显示版本信息



例如,用 gcc 编译 C 源文件 abc.c, 产生目标文件 abc.o, 命令如下所示。

```
gcc -o abc.o abc.c //编译文件 abc.c
```

用 gcc 编译 C 源文件 123.c, 产生目标文件, 如下所示。

```
gcc -o 123.o abc.c //编译文件 123.c
```

使用 ar 命令创建归档文件 mylib, 把生成的模块加入其中, 将 mylib 作为库文件使用, 命令如下所示。

```
#ar rv mylib abc.o 123.o //创建归档文件 mylib
abc.o
123.o
```

可以看到, 通过 ar 命令创建了库文件 mylib, 它包含两个成员文件 abc.o 和 123.o。mylib 也同时具有 abc.o 和 123.o 模块所具有的功能, 在利用 gcc 进行编译链接时, 可以使用选项 “-llibrary” 进行加载。

例如, 显示归档文件 mylib 的内容, 命令如下所示。

```
#ar t lib.a //显示归档文件 mylib 的信息
abc.o
123.o
```

可以看到, mylib 库中的模块为 abc.o 和 123.o。

例如, 删除归档文件 mylib 中的文件 abc.o, 命令如下所示。

```
#ar d mylib abc.o //删除档案文件的文件
ar t mylib
123.o
```

删除库 lib 中的模块 a.o, 重新显示库 lib 中的模块, 可以发现只剩下 b.o。

7.8.2 cpio 命令: 备份文件

cpio 与 tar 程序相似, 但 cpio 不仅可以实现分卷备份, 还可以跳过磁带上的坏区。cpio 经常使用两个选项: -o 和 -i。其中, 选项 “-o” 表示从标准输入设备读入文件名, 归档后从标准输出设备输出; 选项 “-i” 与之相反, 用于恢复备份。cpio 命令经常和命名管道、输入输出重定向联合使用来实现备份。

例如, 对当前目录下所有的 “.txt” 文件进行备份, 备份到 backup_txt 文件中, 命令如下所示。

```
# ls
file1.txt file2.txt file3.txt file4.doc
# ls *.txt | cpio -o > backup_txt
1 block
# ls
backup_txt file1.txt file2.txt file3.txt file4.doc
```

恢复备份需要使用选项 “-i”。例如, 对上例中创建的备份文件 backup_txt 进行恢复, 命令如下所示。

```
# ls
backup_txt
# cpio -i < backup_txt
1 block
# ls
backup_txt file1.txt file2.txt file3.txt
```


7.8.3 dd 命令：磁盘复制命令

dd 命令用于复制文件，可以根据参数选项进行转换和格式化。该命令的语法格式如下所示。

```
dd [bs=BYTE][ cbs=BYTES][ conv=KEYWORDS][ count=BLOCKS][ ibs=BYTES][ if=FILE]
[ obs=BYTES]
[ of=FILE][ seek=BLOCKS][ skip=BLOCKS][ --help][ --version]
```

其中各选项的具体说明如表 7-16 所示。

表 7-16 dd 命令选项说明

选 项	说 明
bs=BYTE	强制设置输入输出的字节数，ibs=BYTES 且 obs=BYTES
cbs=BYTES	设置一次转换的字节数
conv=KEYWORDS	转换逗号分隔的关键词列表
count=BLOCKS	复制指定的输入区块数
ibs=BYTES	设置每次读取的字节数
if=FILE	指定文件作为读取的对象
obs=BYTES	设置每次输出的字节数
of=FILE	结果输出到指定的文件
seek=BLOCKS	在开始输出时，忽略指定的区块数，不进行输出
skip=BLOCKS	在开始输入时，忽略指定的区块数，不进行读取
--help	显示帮助信息
--version	显示版本信息

例如，通过命令 dd 对文件进行复制，输入的文件为 abc.doc，输出的文件为 abc.doc.bak，压缩之后原文件 abc.doc 仍然存在。命令执行过程如下所示。

```
# dd if=abc.doc of=abc.doc.bak //压缩文件
读入了 0+1 个块
输出了 0+1 个块
```

例如，指定输入输出字节数为 256，命令如下所示。

```
# dd if=abc.doc of=abc.doc.bak bs=256 //压缩文件
读入了 0+1 个块
输出了 0+1 个块
```

例如，指定从输出复制的块数为 1，命令如下所示。

```
# dd if=abc.doc of=abc.doc.bak bs=256 cout=1 //压缩文件
读入了 0+1 个块
输出了 0+1 个块
```

7.8.4 使用 cp 命令制作软盘镜像

由于软盘易于损坏，不适合长期保存，因此一般需要将软盘中的内容制作成镜像文件备份在硬盘中，在需要使用时，再将镜像文件恢复到软盘中。在 Red Hat Linux 9 系统中，可以使用 cp 命令完成镜像文件的制作，命令格式如下。

```
cp /dev/fd0 镜像文件名
```



例如，将软盘中的内容制作成镜像文件 `myfloppy.img` 保存在 `/home/backup` 目录下，命令如下所示。

```
#cp /dev/fd0 /home/backup/myfloppy.img
```

在需要使用软盘时，可以再次使用 `cp` 命令将镜像文件恢复到软盘中，如下所示。

```
#cp /home/backup/myfloppy.img /dev/fd0
```

在 Red Hat Linux 9 中，也可以直接将镜像文件挂载到文件系统中，从而可以像直接使用软盘一样读取其中的内容。可以使用 `mount` 命令对镜像文件进行挂载。例如，将上例中的镜像文件挂载到 `/mnt/floppy_img` 目录下，命令如下所示。

```
#mount -o loop myfloppy.img /mnt/floppy_img
```

如果 `/mnt/floppy_img` 目录不存在，需要提前创建。

7.8.5 使用 `cp` 命令制作光盘镜像

ISO 文件是目前广泛使用的光盘镜像标准格式。为了方便传输和数据保存，通常可以将光盘中的内容制作成 ISO 文件存储到磁盘中，在需要使用时再将镜像文件刻录到光盘中。在 Windows 中可以使用 WinISO、Alcohol 等软件从光盘生成 ISO 文件，在 Linux 中与制作软盘镜像一样，可以直接使用 `cp` 命令。其命令格式如下所示。

```
cp /dev/cdrom 镜像文件名
```

例如，将光盘中的内容制成镜像文件，文件名为 `mycdrom.iso`，命令如下所示。

```
#cp /dev/cdrom mycdrom.iso
```

可以使用 `cdrecord` 命令将制作完成的镜像文件刻录到光盘，命令如下所示。

```
cdrecord -v speed=刻录速度 dev=刻录机设备号 镜像文件名
```

刻录机设备号可以使用“`cdrecord -scanbus`”命令获得。例如，将上例中的镜像文件刻录到光盘中，命令如下所示。

```
#cdrecord -v speed=8 dev=0,0 mycdrom.iso
```

在 Linux 中，不仅可以将光盘制成镜像文件，还可以将系统中的任何文件或目录制作成镜像文件。这需要使用命令 `mkisofs`，命令格式如下。

```
mkisofs -r -o 镜像文件名 路径名
```

例如，将 `/home/backup` 目录下的所有内容制成镜像文件，文件名为 `mybackup.iso`，命令如下所示。

```
#mkisofs -r -o mybackup.iso /home/backup
```

在 Linux 系统中，镜像文件也可以直接挂载到系统中，从而可以直接从镜像文件中读取数据，无需再将其刻录到光盘。挂载镜像文件需要使用 `mount` 命令，格式如下。

```
mount -o loop 镜像文件名 挂载点
```

例如，将上例中制作的 `mybackup.iso` 文件挂载到 `/mnt/backup` 目录下，命令如下所示。

```
mount -o loop mybackup.iso /mnt/backup
```

7.8.6 `lha` 命令：压缩或解压缩 `.lhz` 文件

`lha` 命令是从 `lharc` 命令演变而来的压缩和解压缩程序，文件压缩后默认的扩展名为“`.lhz`”。该命令的语法格式如下所示。

```
lha [a][c][e][f][l][p][v][x][u][d][i][z][g][0/1/2] archive_file
```

其中各选项的具体说明如表 7-17 所示。

表 7-17 lha 命令选项说明

选 项	说 明
a	压缩文件
c	重新构建压缩文件
e	解开压缩文件
f	执行 lha 时不询问，强行覆盖已有的文件
l	显示压缩文件的信息
p	从压缩文件输出到标准输出设备
v	显示压缩文件的详细信息
x	解压缩文件
u	升级压缩文件中较新的文件
d	在执行 a/u/c 操作后，删除文件
i	解压缩时，忽略压缩文件指定的路径，将文件解压后放在现行的目录下，与 e/x 联合使用
z	新文件加入压缩文件时，不将文件压缩，与 a/u 联合使用
g	兼容格式
0/1/2	设置压缩文件不同的级别。“0”表示不记录每个文件的路径，“1”为预设值，“2”较节省空间
archive_file	指定的压缩文件名

例如，将名字为 myword 的文件压缩，压缩后保存为名 myword.lhz 的压缩文件，命令如下所示。

```
# lha a myword.lhz myword
```

例如，将当前目录中的所有文件一起压缩，压缩为名为 myback.lhz，命令如下所示。

```
# ls
mybook mydoc my.doc mylist mypaper mypaper.bak mypaper.doc
# lha -a myback.lhz *
my.doc - Frozen(60%) o
mybook - Frozen(87%) o
mydoc - Frozen(23%) o
mylist - Frozen(87%) o
mypaper - Frozen(83%) o
mypaper.bak - Frozen(83%) o
mypaper.doc - Frozen(81%) o
# ls
myback.lhz mybook mydoc my.doc mylist mypaper mypaper.bak mypaper.doc
```

例如，将当前目录中的压缩文件 myback.lhz 解压到当前目录里，命令如下所示。

```
# ls
myback
# lha x myback.lhz
my.doc - Melted : o
mybook - Melted : o
mydoc - Melted : o
mylist - Melted : o
mypaper - Melted : o
mypaper.bak - Melted : o
mypaper.doc - Melted : o
```



```
# ls
myback mybook mydoc my.doc mylist mypaper mypaper.bak mypaper.doc
```

例如，显示 myback.lhz 压缩包中的文件信息，命令如下所示。

```
# lha l myback.lhz
PERMSSN  UID  GID      SIZE  RATIO  STAMP      NAME
-----
-rw-r--r--  0/0      289  60.9% Aug  9 15:37 my.doc
-rw-r--r--  0/0      103  87.4% Aug  9 15:37 mybook
-rw-r--r--  0/0       91  23.1% Aug  9 15:37 mydoc
-rw-r--r--  0/0      103  87.4% Aug  9 15:37 mylist
-rw-r--r--  0/0      102  83.3% Aug  9 15:37 mypaper
-rw-r--r--  0/0      102  83.3% Aug  9 15:37 mypaper.bak
-rw-r--r--  0/0      102  81.4% Aug  9 15:37 mypaper.doc
-----
Total      7 files      892  70.6% Aug  9 15:40
```

7.8.7 unarj 命令：解压文件

unarj 命令用来解压缩.arj 文件。该命令的语法格式如下所示。

```
unarj [e][l][t][x] archive[.arj]
```

其中各选项的具体说明如表 7-18 所示。

表 7-18 unarj 命令选项说明

选 项	说 明
e	解压.arj 文件
l	列出压缩包内容
t	对指定压缩包文件进行检测
x	解压时保留原路径，即对路径也进行解压
archive	指定压缩文件

例如，对 abc.arj 文件解压，命令如下所示。

```
# unarj e abc.arj //解压文件
```

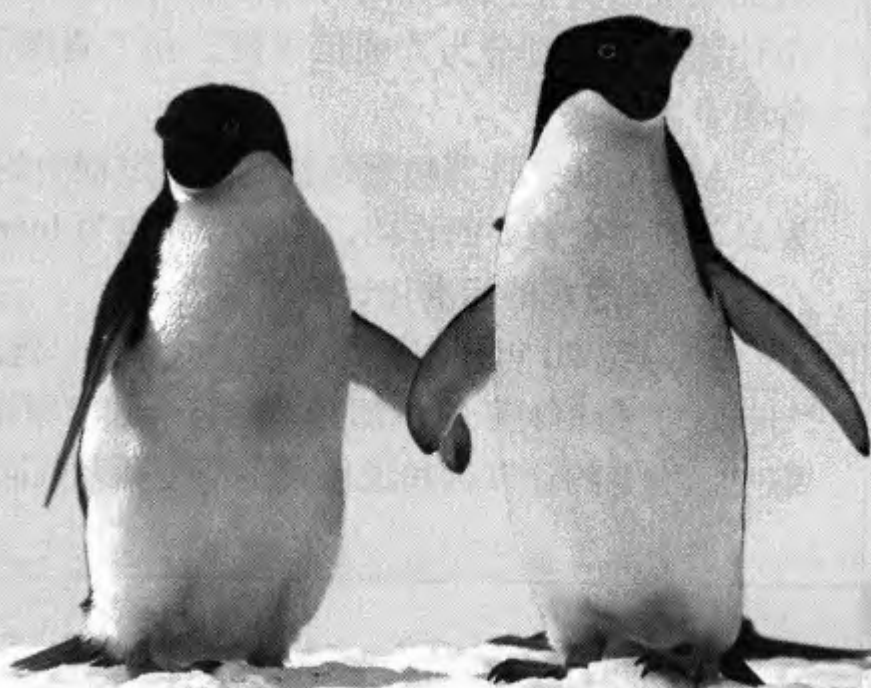

LINUX

第8章

网络管理与相关应用

Linux 具备强大的网络功能,可与当前绝大多数主流的网络操作系统保持良好的兼容性。Linux 沿袭 UNIX 系统,采用 TCP/IP 作为主要的网络通信协议,同时也提供对 IPX/SPS、AppleTalk、ISDN、PPP、SLIP、PLIP、ATM 等协议的支持。由于 Linux 性能稳定,因此许多 ISP 都采用 Linux 架设网络服务器,如 WWW、FTP、Mail Server、DNS 等。在对 Linux 主机进行基本的网络配置之后,就可以使用该主机与其他主机进行网络连接通信。

为方便网络管理和网络应用,Linux 提供了大量的实用命令,从而从各个方面进一步增强了 Linux 的网络功能。





8.1 计算机网络的发展

计算机网络是将地理位置分散的多台计算机按着通信协议有机地联接起来, 以实现计算机之间的信息交流、资源共享和协同工作的计算机复合系统。

计算机网络最早出现在 20 世纪 50 年代中期, 源于美国的半自动地面防空系统 (SAGE)。当时, SAGE 系统能够将远距离的雷达和测控设备的信息经过通信线路汇集到一台 IBM 计算机上进行处理和控制。通常 SAGE 被认为是计算机技术和通信技术相结合的最初尝试。

在 20 世纪 60 年代中后期, 美国国防部 (Department of Defense, 缩写为 DoD) 日益认识到其对计算机网络的依赖性。出于“如果敌人破坏了我们的网络会发生什么情况? 我们会无法访问我们的计算机吗?” 这样的考虑, 1969 年美国高级研究计划署 (ARPA, Advanced Research Project Agency) 组织和成功研制了世界上公认的第一个远程计算机网络 ARPAnet。

ARPAnet 在 1969 年建成时仅有 4 个试验节点, 到 1971 年 2 月已扩充为具备 15 个节点、23 台主机的计算机网络, 并正式投入使用。由于现代计算机网络的许多概念和方法都起源于 ARPAnet, 因此, 人们通常将其视为计算机网络的起源, 同时也是 Internet 的起源。计算机网络发展到今, 一般可分为以下 4 个阶段。

1. 面向终端的计算机通信网络

计算机技术与通信技术结合, 形成了计算机网络的雏形。此时的计算机网络仅仅是以单台计算机为中心的远程联机系统, 因此也被称为“面向终端的计算机通信网络”。美国在 1963 年投入使用的飞机订票系统 SABRE-I, 就是这类系统的典型代表之一。此系统以一台中心计算机为网络的主体, 将全美范围内的 2000 多个售票终端通过电话线连接到中心计算机上, 从而实现了联网订票。

2. 初级计算机网络

随着计算机与通信技术的进一步发展, 以及计算机网络体系结构与协议的深入研究, 形成了最初的计算机网络, 此时的计算机网络一般被称为“初级计算机网络”。其中, 20 世纪 60 年代后期到 20 世纪 70 年代初期发展起来的美国高级研究计划署的 ARPAnet 就是这类系统的典型代表。此时的计算机网络由若干个计算机互联而成。同时, ARPAnet 将一个计算机网络划分为“通信子网”和“资源子网”两大部分, 目前的计算机网络仍沿用这种组合方式。

ARPAnet 是计算机网络技术发展过程中的一个里程碑, 它的研究成果对促进网络技术的发展具有十分重要的作用, ARPAnet 也为 Internet 的形成奠定了基础。

3. 开放式的标准化计算机网络

20 世纪 70 年代末至 20 世纪 90 年代, 逐渐形成了“开放式标准化计算机网络”。这里的“开放式”是相对于以往那些只能符合独家网络厂商要求的各自封闭的系统而言。在开放式网络中, 所有的计算机和通信设备都遵循着共同认可的国际标准, 从而可以保证不同厂商的网

络产品可以在同一网络中顺利地进行通信。事实上,目前存在着两种占主导地位的网络体系结构,一种是 ISO (International Standards Organization, 国际标准化组织) 的 OSI (开放式系统互联) 体系结构; 另一种是 TCP/IP (传输控制协议 / 网际协议) 体系结构, 它也是事实上的网络标准。

4. 新一代的计算机网络

互联网 (Internet) 的进一步发展要求新一代计算机网络必须提供高速、大容量、安全的综合性数字信息传递机制。通过使用 IPv6 技术, 新一代互联网的 IP 地址空间将进一步扩展, 现有的 IPv4 地址空间紧缺问题将得到解决; 通过采用多层次路由结构、分层目录管理等技术, 新一代互联网将有可能解决目前网络管理的无政府状态。总之, 正在研究与发展的“新一代的计算机网络”将向着全面互联、高速和智能化发展, 并将继续得到更为广泛的发展和应用。

8.2 网络基本类型

对计算机网络进行分类的标准很多。例如, 按网络的覆盖范围分为局域网 LAN (Local Area Network)、城域网 MAN (Metropolitan Area Network)、广域网 WAN (Wide Area Network) 和互联网 (Internet); 按网络的拓扑结构分类分为星型结构、环型结构、总线型结构、星型和总线型结合的复合型结构等。这些划分标准都从不同角度对计算机网络特征进行了描述。

8.2.1 按地理覆盖范围

按地理覆盖范围划分是一种通用网络划分标准。按该标准可以把各种网络类型划分为局域网、城域网、广域网和互联网 4 种。局域网一般应用于办公楼群、校园网、工厂及企事业单位等相对较小的区域内; 城域网通常应用于一个大型城市或都市地区; 广域网络也称为远程网, 所覆盖的范围比城域网更大, 一般是在不同城市之间实现互联; 而互联网是全球范围内的计算机的互联互通。

● 局域网 (LAN)。

局域网, 顾名思义即指在局部地区范围内使用的网络, 其覆盖的地区范围较小, 一般连接距离在几公里以内, 最大距离不超过 10 公里, 一般用于组建一个部门或单位的内部网络。通常所说的“LAN”就是指局域网。

局域网是在小型计算机和微型计算机大量推广使用之后才逐渐发展起来的计算机网络。一方面, 局域网拓扑结构简单, 组网灵活方便, 易于管理与配置; 另一方面局域网速率高、延迟小、成本低廉, 因此, 深受广大用户的欢迎。局域网是目前最常见、应用最广也是最活跃的一种网络。

IEEE 的 802 标准委员会按介质访问方式定义了多种局域网, 主要包括: 以太网 (Ethernet)、令牌环网 (Token Ring)、光纤分布式接口网络 (FDDI)、异步传输模式网 (ATM) 以及无线局域网 (WLAN)。

● 城域网 (MAN)。



城域网采用 IEEE802.6 标准，是介于局域网与广域网之间的一种大范围的高速网络，连接距离可以在 10~100 公里。

随着局域网的广泛使用，人们逐渐要求扩大局域网的使用范围，或者要求将已经使用的局域网互相连接起来，使其成为一个规模较大的城市范围内的网络，即城域网。例如，将同一个城市内的政府机构的局域网、医院的局域网、电信的局域网以及公司企业的局域网连接起来组成城域网，从而实现大量用户、多种信息资源的互联互通。

城域网与局域网相比覆盖的范围更大，连接的计算机数量也更多，从某种意义上讲，城域网是局域网在地理范围上的延伸。光纤连接的引入使城域网中的高速互连成为可能。城域网既可以是私人网，也可以是公用网；既可以支持数据和语音传输，也可以与有线电视相连。

城域网多采用 ATM 技术做为骨干网。ATM 采用固定长度（53 字节）的“信元交换”技术来替代传统的“包交换”技术，可以为不同的应用提供 25、51、155 和 622Mbit/s 几种不同的传输速率。由于 ATM 没有共享介质或包传递带来的延时，因此非常适合音频和视频数据的传输。但是，ATM 需要专门的软硬件，部署成本相对比较高。

由于各种原因，城域网的特有技术并没能在世界各国迅速地推广，相反，在实践中人们通常使用广域网的技术去构建与城域网目标、范围相当的网络。

● 广域网（WAN）。

广域网也称远程网（Wide Area Network），其所覆盖的范围比城域网（MAN）更广，可以跨越城市、地区、国家甚至洲际，地理距离可从几百公里到几千公里。广域网通常以连接不同地域的大型主机系统或局域网为主要目的。由于距离较远，信息衰减比较严重，所以这种网络一般需要租用专线或者自行铺设专线（例如使用光纤、双绞线、同轴电缆、微波、卫星、红外以及激光等）。因为所连接的用户多，总带宽有限，所以用户的终端连接速率一般较低，通常为 9.6Kbit/s~45Mbit/s。比较典型的广域网应用如中国移动的网络运营系统（包括计费系统、信息系统、综合结算系统和综合账务系统等）。

● 互联网（Internet）

互联网又称为“因特网”，是世界上最大的计算机网络。互联网通过使用统一的协议（TCP/IP）将全球成千上万的计算机网络（广域网与广域网之间、广域网与局域网之间、局域网与局域网之间）连接起来，使得各网络之间可以交换信息，共享资源。由于互联网由全球范围内的计算机网络构成，对于互联网普通用户而言，Internet 拥有不计其数的网络资源，用户可以随时从 Internet 上获得所需的信息。

综上所述，各种计算机网络参数比较如表 8-1 所示。

表 8-1 局域网/城域网/广域网/互联网参数比较

网 络 类 型	缩 写	数据传输率	覆 盖 范 围	典型应用
局域网	LAN	4Mbit/s~10Gbit/s	<10km	校园、楼宇
城域网	MAN	50Kbit/s~100Mbit/s	10~100km	城市
广域网	WAN	9.6Kbit/s~45Mbit/s	100~1000km	城市、国家、洲
互联网	Internet	9.6Kbit/s~45Mbit/s	全球	全球互联

8.2.2 按拓扑结构

网络拓扑结构指网络中各个节点相互联接的方式，主要有星型拓扑结构、总线拓扑结构、环型拓扑结构和混合型拓扑结构。

- 星型拓扑结构。

星型拓扑是目前在局域网中应用得最为普遍的一种结构。由于在该种网络结构中，各个网络节点通过点对点方式连接到中央节点上，整体呈现出星状分布状态，因此被称为星型拓扑结构，如图 8-1 所示。星型拓扑结构中某一普通网络节点出现故障不会波及其他节点，但是一旦中心节点（通常由 Hub 或交换机组成）出现问题，其所在网络将陷入瘫痪。

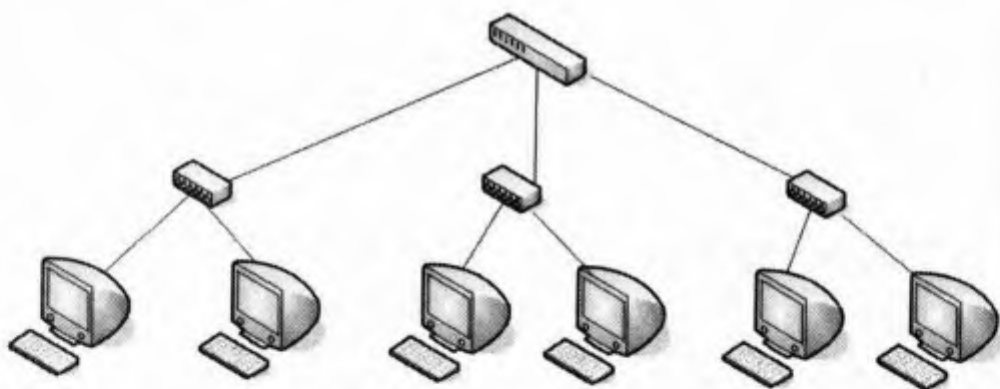


图 8-1 星型网络拓扑结构图

星型拓扑结构主要应用于 IEEE 802.2、IEEE 802.3 标准以太网中，目前用得最多的传输介质是双绞线，如常见的五类双绞线、超五类双绞线等。

星型拓扑结构具有容易实现、易扩展、维护方便、传输数据快等优点，但同时也存在设备成本高、可靠性较低、资源共享能力差等不足。



该种网络出现故障时，可以先从集线器或交换机等中央节点查起。每一台计算机在集线器或交换机上都有指示灯对应。如果是 10Mbit/s/100Mbit/s 自适应，则适应 10Mbit/s 连接时亮黄色灯，适应 100Mbit/s 连接时亮绿色灯，无连接时，灯不亮。

- 总线拓扑结构。

总线拓扑结构采用一根公用总线作为传输介质，所有设备都直接与总线相连，其结构如图 8-2 所示。总线上的信息多以基带形式串行传递，其传递方向总是从发送信息的节点开始向两端扩散，因此采用总线拓扑结构的网络又称为广播式计算机网络。

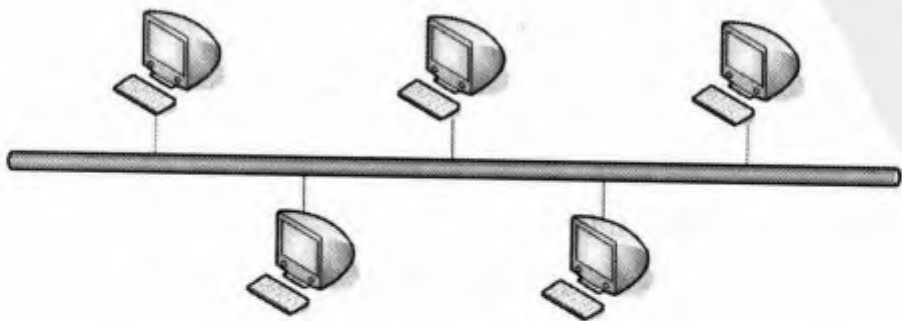


图 8-2 总线型网络拓扑结构图



在总线拓扑结构中，所有的工作站和服务器都连接到总线上，无中心节点，各个节点的地位是平等的。由于各节点是共用总线带宽的，所以在传输速度上会随着接入网络的节点的增多而下降。单个节点失效不影响整个网络的正常通信，但是如果总线出现故障，则整个网络或相关主干网段就会瘫痪。

总线拓扑结构具有组网费用低、可靠性高、易扩展（但所能连接的节点数量有限）等优点，但由于是共享总线，一次仅能允许一个节点发送数据，其他节点必须等待总线空闲后才能发送，因此存在数据传输率较低、总线负担过重等问题。而且该种类型的网络一旦发生连接错误，维护和检测都很不方便。

总线所采用的物理介质一般为同轴电缆（包括粗缆和细缆），少数也使用光纤。

● 环型拓扑结构。

环型拓扑一般仅适用于 IEEE 802.5 的令牌网（Token ring network）。在这种网络结构中，所有节点首尾相接，连接成一个闭环，整个网络发送的信息就是在这个环中传递，如图 8-3 所示。网络中的节点若想发送信息，必须获得“令牌”，“令牌”会在环型连接中被依次传递。

在环型拓扑中传送的数据要经过每个节点，如果有一个节点出现故障，将会造成整个网络的瘫痪。同时，环型拓扑结构中对故障节点的定位较难，维护起来非常不便。

环路上各节点都是自举控制，因而控制软件简单，但环路必须闭合才能正常工作，这使得该网络不便于扩充。

令牌环一般由同轴电缆或光纤组成。

● 混合型拓扑结构。

混合型拓扑结构从星型结构和总线型结构演变而来，兼具星型结构和总线型结构的特点，是一种综合布线方式，如图 8-4 所示。

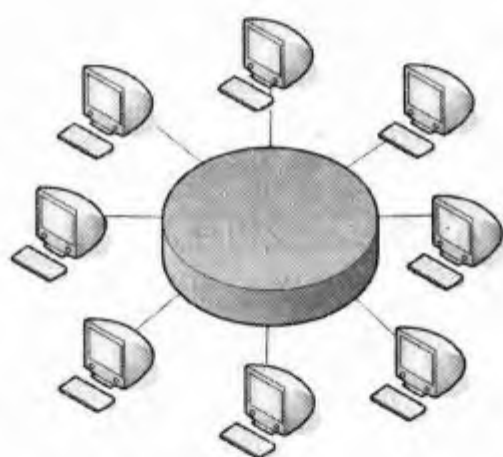


图 8-3 环型网络拓扑结构图

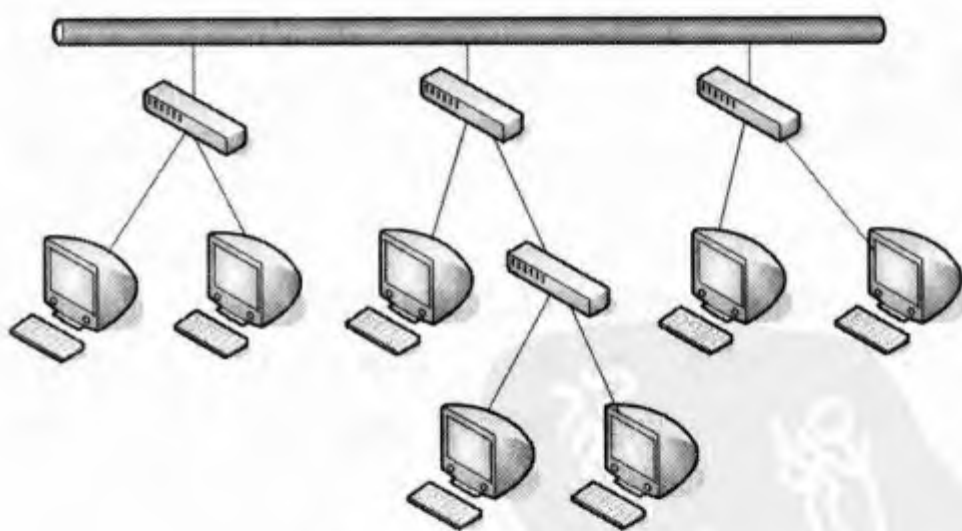


图 8-4 混合型网络拓扑结构图

混合拓扑结构更能满足较大网络的扩展需求，既突破了星型网络在传输距离上的局限，又解决了总线型网络在连接节点数量上的限制。但是，由于混合拓扑继承了总线型网络拓扑结构的特点，同样会存在较难维护的问题。

8.3 网络体系结构

计算机网络的体系结构是指该网络及其子系统所能完成功能的精确定义，通常采用层次结构进行描述。OSI/RM 和 TCP/IP 是目前两个主要的参考模型。

8.3.1 OSI/RM 参考模型

OSI/RM（开放系统互连参考模型）是 ISO（International Standards Organization）在网络通信方面所定义的连接异种计算机的开放协议标准。基于这个开放的模型，各网络设备厂商就可以遵照共同的标准来开发网络产品，最终实现彼此兼容。

整个 OSI/RM 模型共分 7 层，从下往上依次为：物理层、数据链路层、网络层、传输层、会话层、表示层和应用层，如图 8-5 所示。

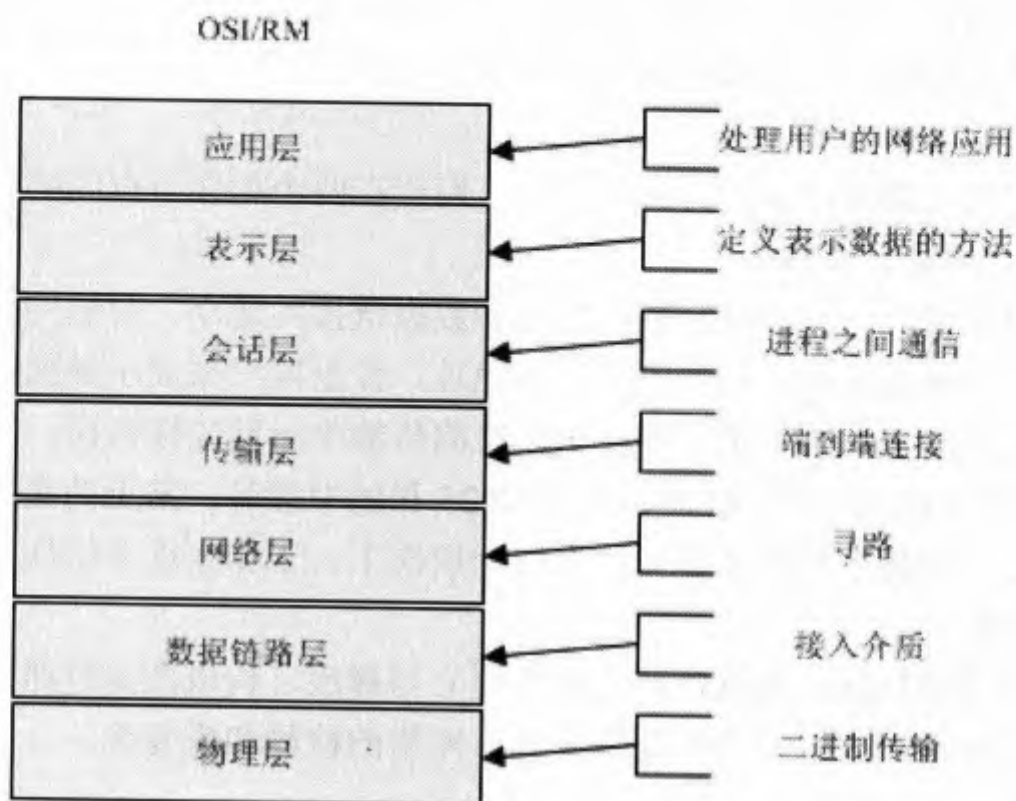


图 8-5 OSI/RM 参考模型

其中相邻层之间可以通信，第 1~3 层属于通信子网的范畴，第 5~7 层属于资源子网的功能范畴，第 4 层是中间层，连接上下 3 层。当接收数据时，数据是自下而上传输；当发送数据时，数据是自上而上传输。虽然这种通信流程垂直通过各层次，但每一层都在逻辑上能够直接与远程计算机系统的相应层直接通信。为构建这种层次间的逻辑连接，需要发送端在每一层都要在数据报文前增加报文头。该报文头只能被远程计算机的相应层识别和使用。接收端接收到数据报文后会删去报文头，每一层都删去该层负责的报文头，最后将数据传递给相关的应用程序，其过程如图 8-6 所示。

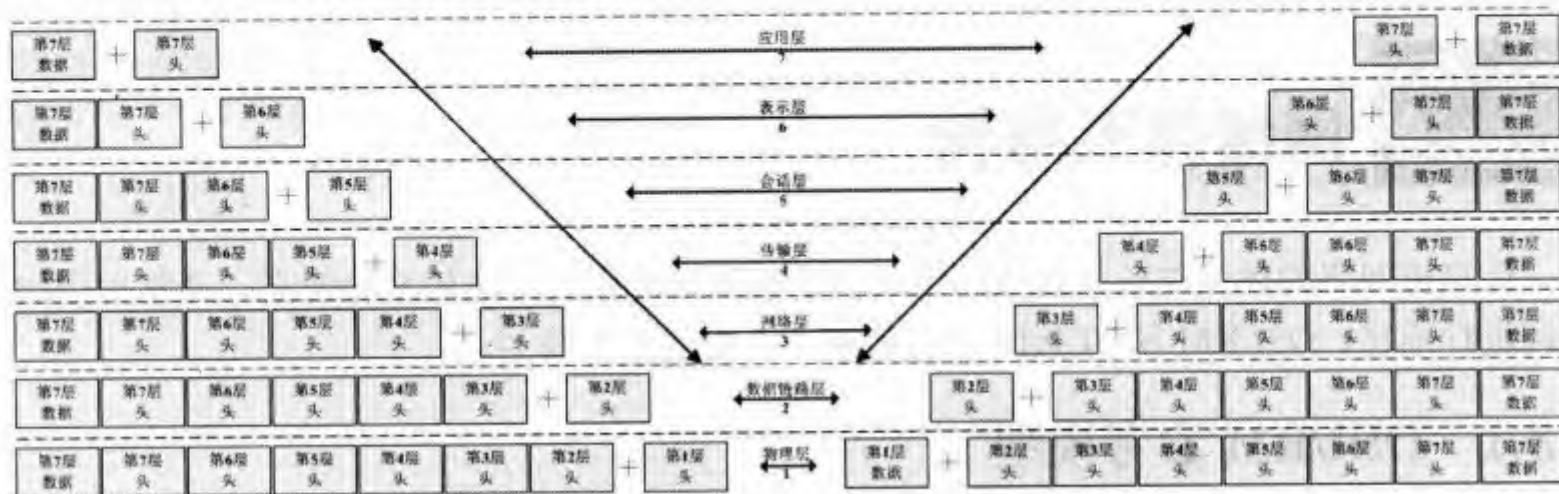


图 8-6 OSI/RM 层间通信原理

各层功能如下。

● 物理层。

物理层位于整个 OSI 参考模型的最低层，其主要任务是为上一层的数据链路层提供一个物理连接，透明地传送比特流。物理层建立在物理介质上，提供机械和电气接口，包括电缆、物理端口以及附属设备。物理层提供的服务包括：物理连接、物理服务数据单元顺序化（接收物理实体收到的比特顺序，与发送物理实体所发送的比特顺序相同）和数据电路标识。

● 数据链路层。

数据链路层建立在物理传输能力的基础上，用于在两个相邻节点间的链路上无差错地传送以帧（Frame）为单位的数据。

数据链路层的主要任务就是进行数据封装和数据链接的建立。封装的数据信息由 4 部分组成，其中地址段含有发送节点和接收节点的地址，控制段用来表示数据链接帧的类型，数据段包含实际要传输的数据，差错控制段用来检测传输中出现的帧错误。

数据链路层可使用的协议有 SLIP、PPP、X25 和帧中继等。常见的集线器和低档的交换机以及 Modem 之类的拨号设备都是工作在这个层次上。工作在这个层次上的交换机通常被称为“第二层交换机”。

数据链路层的功能包括：数据链路连接的建立与释放、构成数据链路数据单元、数据链路连接的分裂、定界与同步、顺序和流量控制、差错的检测和恢复等。

● 网络层。

网络层属于 OSI 中的第 3 层，解决的是网际的通信问题，即选择合适的路由和交换节点，透明地向目的节点交付发送节点所发送的分组。网络层的主要功能是寻路，即选择到达目标主机的最佳路径，并沿该路径传送数据包。

除此之外，网络层还具有流量控制和拥塞控制的功能。现在较高档的交换机也可直接工作在这个层次上，由于该层具备路由功能且属于第 3 层，通常被称为“第三层交换机”。

网络层的功能包括：建立和拆除网络连接、路径选择和中继、网络连接多路复用、分段和组块、服务选择和传输以及流量控制。

● 传输层。

传输层在网络两个节点之间建立端到端的通信信道，主要解决的是数据在网络之间的传

输质量问题。传输层提供两端点之间可靠、透明的数据传输，执行端到端的差错控制、顺序控制和流量控制，管理多路复用和解复用。传输层是计算机通信体系结构中关键的一层，主要涉及的是网络传输协议，如 TCP 协议。

传输层的功能包括：映像传输地址到网络地址、多路复用与分割、传输连接的建立与释放、分段与重新组装、组块与分块。

● 会话层。

会话层提供面向通信的逻辑用户接口。会话可能是一个用户通过网络登录到一个主机，或一个正在建立的用于传输文件的连接。

会话层的功能主要有：会话连接到传输连接的映射、数据传送、会话连接的恢复和释放、会话管理、令牌管理和活动管理。

● 表示层。

表示层主要解决用户信息的语法表示问题，如用于文本文件的 ASCII 和 EBCDIC 表示形式。如果通信双方用不同的数据表示方法，彼此之间就不能互相理解。表示层就是用于屏蔽这种不同之处。表示层的功能主要有：数据语法转换、语法表示、连接管理、数据加密和数据压缩。

● 应用层。

应用层是 OSI 参考模型的最高层，直接面对用户的具体应用。应用层包含用户应用程序执行通信任务所需要的协议和功能，如电子邮件和文件传输等应用。

在 7 层网络中，第 1、2 层处理网络信道问题，第 3、4 层处理传输服务问题，第 5、6、7 层处理应用服务问题。

8.3.2 TCP/IP 参考模型

与 OSI 参考模型不同，TCP/IP 模型更侧重于互联设备间的数据传送，而不是严格的功能层次划分，因而为协议的具体实现留下很大的余地。一般而言，OSI 参考模型比较适合用于解释互联网络通信机制，而 TCP/IP 是互联网络协议事实上的标准。从体系结构角度看，TCP/IP 参考模型分为 4 层：网络接口层、网络层、传输层和应用层。其层次结构与 OSI 参考模型的对应关系如图 8-7 所示。

可以看到，TCP/IP 模型中的应用层在功能上与 OSI/RM 模型的最上面 3 层相当，网络接口层与 OSI/RM 模型的最下面 2 层相当。TCP/IP 参考模型是在 TCP/IP 协议出现很久以后才发展起来的，由于其更强调功能分布而不是严格的功能层次的划分，因此比 OSI/RM 模型更灵活。TCP/IP 各层含义及功能如下所述。

● 网络接口层（Network Interface Layer）。

TCP/IP 模型将与物理网络相关的部分称为网络接口层，与 OSI/RM 的物理层和数据链路

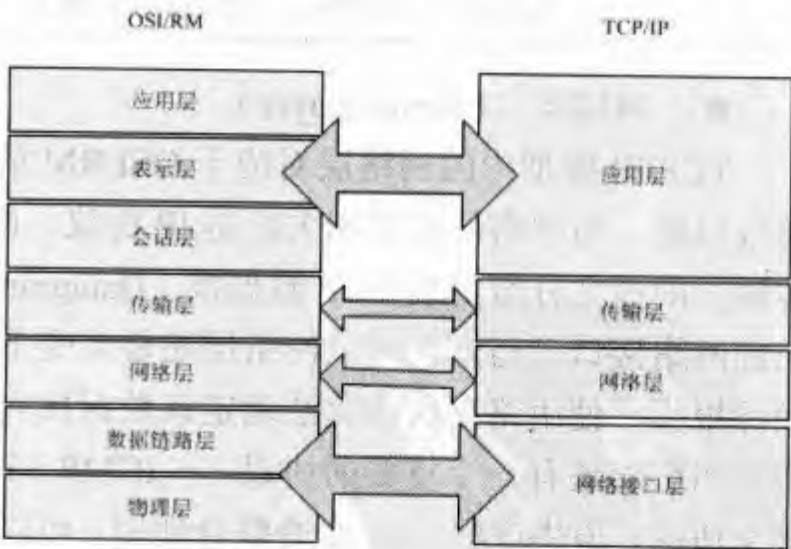


图 8-7 TCP/IP 与 OSI/RM 对应关系



层相对应。负责接收上层递交来的数据报，并把该数据报发送到指定的网络上，同时负责接收来自网络的数据报，并传递给上层。

网络接口层也称为 TCP/IP 链路层，与 Ethernet（以太网）、Token Ring（令牌环网）以及 ATM（异步传输模式）等网络接入技术密切相关。各种网络接入技术的说明如表 8-2 所示，其中 Ethernet 技术近年得到了充发的发展和广泛的应用。

表 8-2 主要网络接入技术说明

类 型	说 明
Ethernet	标准以太网，遵守 IEEE802.3 标准。最早由 Xerox（施乐）公司创建，在 1980 年由 DEC、Intel 和 Xerox 三家公司联合制定的标准。采用 CSMA/CD（带有冲突检测的载波侦听多路访问）访问控制方法，理论传输速度最高可达 10Mbit/s。以太网主要有两种传输介质：双绞线和同轴电缆
Fast Ethernet	快速以太网，遵守 IEEE802.3u 标准，理论传输速度最高可达 100Mbit/s，传输介质要求 5 类或 5 类以上的电缆
GB Ethernet	吉比特以太网，遵守 IEEE802.3z 标准，理论传输速度最高可达 1000Mbit/s，传输介质一般采用光纤
10GB Ethernet	10G 以太网，理论传输速度最高可达 10000Mbit/s，目前还处于研发阶段，还没有得到实际应用
Token Ring	令牌环网，IBM 公司于 70 年代创建，遵守 IEEE802.5 标准。数据传输速度为 4Mbit/s 或 16Mbit/s，新型的快速令牌环网速度可达 100Mbit/s。令牌环网的传输方法在物理上采用了星型拓扑结构，但逻辑上仍是环型拓扑结构。由于只有具有令牌的计算机才被允许传递数据，因此比 Ethernet 效率更高
FDDI	FDDI 的英文全称为“Fiber Distributed Data Interface”，即“光纤分布式数据接口”，使用基于 IEEE802.5 令牌环标准的令牌传递 MAC 协议。可提供多达 100Mbit/s 的数据传输速度，支持长达 2km 的多模光纤，最多可容纳 1000 个节点
ATM	ATM 的英文全称为“Asynchronous Transfer Mode”，即异步传输模式。ATM 是一种信元交换技术（使用 53 字节固定长度的信元进行交换），由于没有共享介质或包传递带来的延时，非常适合音频和视频数据的传输。ATM 具有不同的速率，分别为 25、51、155、622Mbit/s，从而为不同的应用提供不同的速率
PPP	PPP 的英文全称为“Point-to-Point Protocol”，即点对点协议。PPP 由 IETF（Internet Engineering Task Force）定义，用于取代串行 IP（Series Line Internet Potocol, SLIP），支持多种协议，有错误检测和链路管理功能。目前利用 PPP 通过调制解调器接入 Internet，仍是最为普遍的方法
WLAN	WLAN 英文全称为“Wirrless Local Area Network”，即无线局域网。WLAN 遵守 802.11 系列标准，该系列共分 4 个标准，分别为 802.11b、802.11a、802.11g 和 802.11z。其中，802.11b 传输速度为 11Mbit/s，802.11a 标准的连接速度可达 54Mbit/s。802.11g 兼容 802.11b 与 802.11a 两种标准。802.11z 专门加强了无线局域网安全

● 网络层（Internet Layer）。

TCP/IP 模型中的网络层对应于 OSI/RM 的网络层，用于解决互联网中计算机到计算机的通信问题。与网络层密切相关的是 IP 协议，即 Internet Poctocol（网际协议）。网络层把来自传输层的报文分组封装在 IP 数据报（Datagram）中，然后按路由选择算法将数据报发送到相应的网络接口。与此同时，网络层还要接受下层递交过来的数据报，校验数据报的有效性，删除报头，使用路由选择算法确定该数据报应该由本地处理，还是转发出去。

网络层还有一个重要的协议——ICMP（Internet Control Message Protocol），即网际控制报文协议。该协议与 ping 命令联合使用，可以查看当前计算机节点与本地网络上其他节点的连通性。

● 传输层（Transport Layer）。

TCP/IP 模型中的传输层对应于 OSI/RM 的传输层，其任务是提供应用程序之间端到端（End-to-End）的通信。传输层将要发送的报文或数据流分成更小的段，即报文分组（Packet），然后把每个报文分组连同报文目的地址一并递交给网络层。与此同时，传输层也接收来自下

层的报文分组，重组为完整的报文或数据流。传输层对数据流具有一定的调节作用，确保其完整、正确，并按顺序递交。

TCP/IP 模型中的传输层主要包括 TCP（Transmission Control Protocol，即传输控制协议）和 UDP（User Datagram Protocol，用户数据报协议）两个协议。1974 年，在 ARPAnet 诞生后的短短 5 年里，Vinton Cerf 和 Robert Kahn 发明了 TCP。TCP 在 IP 之上提供了一个可靠的、连接式的协议，可以提供比其他协议更多的保护。TCP 要求在提供相应服务前必须先建立连接（三次握手机制），因此 TCP 也被称为面向连接的协议（Connection-oriented）。而 UDP 数据包传输基于尽力递交，没有差错修正、重传、重新排序等机制，无需先建立连接，因此 UDP 也被称为无连接协议（Connectionless）。

● 应用层。

TCP/IP 将 OSI/RM 传输层之上的所有层统称为应用层。在应用层，用户调用访问网络的应用程序，应用程序与传输层协议相配合，用以发送和接收用户所需的数据。

TCP/IP 应用层协议比较多，常见的有 HTTP、FTP、POP3 等。每个应用层协议通常会使用一些指定的端口（端口类似于电视机中的频道，将应用程序指定到正确的端口，就可以接收和发送与该端口相关的数据，TCP/IP 有 65536 个可用端口）。一些特定用途的端口由 Internet Assigned Numbers Authority（www.iana.org）分配，如 HTTP 使用 80，FTP 使用 21，以及 POP3 使用 110。

表 10-3 列举了几个主要的 TCP/IP 应用层协议及其相关端口。

表 8-3 主要的 TCP/IP 应用层协议

协 议	端 口	说 明
FTP	21	实现文件传输的基本协议，利用 FTP 协议可以把本地文件上传到网络上的另一台计算机上（FTP 服务器），也可从网络上（FTP 服务器）下载所需的文件。目前有许多软件站点就是通过 FTP 协议来为用户提供下载服务的
HTTP	80	用于网页的超文件链接协议（Hypertext Transfer Protocol）
HTTPS	443	Secure HTTP
SSH	22	Secure Shell，加密计算机之间的通信，是 Telnet 的安全性版本
Telnet	23	Telnet 用明文方式建立计算机之间的通信连接，可以登录到远程计算机上，并进行信息访问，包括读取所需的数据库、进行联机游戏、建立对话服务以及访问电子公告牌，但只能进行字符类操作和会话
SMTP	25	简单邮件传输协议（Simple Mail Transfer Protocol），用于发送电子邮件，负责邮件的发送、分拣和存储
POP3	110	用于接收电子邮件的 POP3 协议（Post Office Protocol 3），负责将邮件通过 SLIP/PPP 协议连接下载到用户计算机上
SNMP	161	用于网络管理的简单网络管理协议（Simple network Management protocol）
IPP	631	互联网打印协议（Internet Print Protocol），与 Common UNIX Print System（CUPS）相关
SWAT	901	Samba 服务的 Web 管理工具
NFS	2049	用于 Linux/UNIX 计算机之间文件的共享
IMAP	143	使用 Internet 消息访问协议的邮件阅读器

/etc/services 文件中列出了 Red Hat Linux 9 中所使用的 TCP/IP 应用层协议的完整列表，



包括服务的名称、关联的端口号以及相关说明，如下所示。

```
#vi /etc/services
...
# 21 is registered to ftp, but also used by fsp
ftp      21/tcp
ftp      21/udp      fsp fspd
ssh      22/tcp      # SSH Remote Login Protocol
ssh      22/udp      # SSH Remote Login Protocol
telnet   23/tcp
telnet   23/udp
# 24 - private mail system
lmtpl    24/tcp      # LMTPL Mail Delivery
lmtpl    24/udp      # LMTPL Mail Delivery
smtp     25/tcp      mail
smtp     25/udp      mail
...
```

8.4 网络配置基本内容

在 Linux 主机上，通过对网络基本内容的配置，可以将计算机连接到网络上，实现与其他计算机之间的通信。网络配置通常涉及主机名、IP 地址、子网掩码、网关地址、域名服务器地址等内容。

8.4.1 主机名

主机名用于在网络上标识一台计算机，通常主机名在网络中是惟一的。

8.4.2 IP 地址

发送到 Internet 上的数据之所以能够找到目的计算机，是因为任何一个连接到 Internet 上的计算机都有一个惟一的网络地址。为了确保 Internet 上的每一个网络地址始终是惟一的，国际网络信息中心（NIC）会根据网络的大小为每一个申请者统一分配 IP 地址。

IP 地址有 IPv4 和 IPv6 两个版本。IPv4 由于其设计上的局限，目前除了美国外，各国都出现 IP 地址短缺现象。IPv6 主要解决 IPv4 协议中 IP 地址远远不够的现象。

IPv4 使用点分十进制数来描述地址，共 32 位长，由 4 个分段的十进制数组成。例如，用二进制描述的 32 位地址如下。

```
0 1 1 1 1 1 1 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 1 1 1 1
```

为了方便阅读，通常将 32 位地址进行分组，每 8 位为一组，如下所示。

```
0 1 1 1 1 1 1 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 1 1 1 1
```

最后，将每个 8 位数据转换成十进制，并用小数点隔开。IPv4 点分十进制描述的地址如下所示。

```
126 . 136 . 1 . 47
```

与记忆二进制位串（如 011111101000100000000000100101111）相比，记忆 IP 地址 126.136.1.47 更加容易。

IPv6 采用冒分十六进制来描述地址，共 128 位，由 8 个分段的十六进制数组成，如下所示。

FEDC : BA98 : 7654 : 3210 : FEDC : BA98 : 7654 : 3210

个别字段中前面的 0 可以不写, 但是每段必须至少有一位数字。



当 IPv4 和 IPv6 节点混用时，可以采用另一种表示形式，如下所示。

x : x : x : x : x : x : d . d . d . d

其中 x 是地址中 6 个高阶 16 位段的十六进制值, d 是地址中 4 个低阶 8 位段的十进制值 (标准 IPv4 表示)。例如

0 : 0 : 0 : 0 : 0 : 0 : 13 . 1 . 68 . 3

```
0 : 0 : 0 : 0 : 0 : F F F F : 1 2 9 . 1 4 4 . 5 2 . 3 8
```

也可以写成压缩格式，如下所示。

: : 1 3 . 1 . 6 8 . 3

: : F F F F . 1 2 9 . 1 4 4 . 5 2 . 3 8

IPv4 地址由网络号（网络 ID）和主机号（主机 ID）两部分构成。如上所述，可以将 IPv4 地址划分为 4 个分段的十进制数，如图 8-8 所示。

按照 IPv4 协议规定, 互联网上的 IPv4 地址共有 A、B、C、D、E 共 5 类。下面主要介绍 IPv4 地址分类。

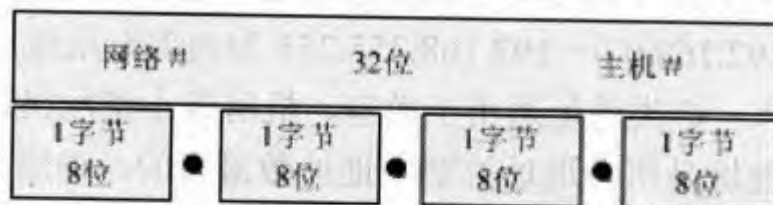


图 8-8 IPv4 地址划分

- A 类 IP 地址。

A 类地址是最大的地址组，用前面 8 位来标识网络号（规定最前面一位为“0”），其余 24 位标识主机地址，如下所示。

```
0 n n n n n n n h h h h h h h h h h h h h h h h h h h h h
```

其中，字母“n”表示网络位，字母“h”表示主机位。可以看到，A类地址的第一段取值（也即网络号）可以是“00000001~01111111”之间的任一数字，转换为十进制后即为1~126之间的任一数字。其余各段合在一起表示主机号。由于对主机号没有做硬性规定，所以A类IP地址范围为“1.0.0.0~128.255.255.255”，其中127.0.0.1专门用于回环地址，127.0.0.0专门用于回环网络，10.0.0.0~10.255.255.254是内部网络地址。

A 类地址主要针对大型政府网络,全世界总共只有 126 个可用的 A 类网络,每个 A 类网络最多可以连接多达 16 777 214 台计算机。下面是一些 A 类地址网络号。

10 . 0 . 0 . 0

44 . 0 . 0 . 0

101 . 0 . 0 . 0

120 . 0 . 0 . 0

● B 类 IP 地址。

B 类地址用前面 16 位来标识网络号, 其中最前面两位规定为 “10”, 其余 16 位标识主机号, 如下所示。

1 0 n n n n n n n n n n n n n n n h h h h h h h h h h h h h h h h h h

可以看到，B 类地址的第 1 段取值为“10000000~10111111”之间，转换成十进制后即为 128~191 之间。第 1 段、第 2 段合在一起表示网络号，第 3 段、第 4 段合在一起标识网络上的主机号。由于对主机号没有做硬性规定，所以 B 类地址的范围为“128.0.0.0~



191.255.255.255”，其中 172.16.0.0~172.31.255.254 地址段专门用于内部网络。

B 类地址适用于中等规模的网络，全世界大约有 16 000 个 B 类网络，每个 B 类网络最多可以连接 65 534 台计算机。

下面是一些 B 类地址网络号。

```
137 . 55 . 0 . 0
129 . 33 . 0 . 0
190 . 254 . 0 . 0
150 . 0 . 0 . 0
168 . 30 . 0 . 0
```

● C 类 IP 地址。

C 类地址用前面 24 位来标识网络号（其中最前面三位规定为“110”），其余 8 位标识主机号，如下所示。

```
1 1 0 n n n n n n n n n n n n n n n n n n n n h h h h h h h h
```

可以看到，C 类地址的第 1 段取值为“11000000~11011111”之间，转换成十进制后即为 192~223。第 1 段、第 2 段、第 3 段合在一起表示网络号，第 4 段标识网络上的主机号。由于对主机号没有做硬性规定，所以 C 类地址范围为“192.0.0.0~223.255.255.255”，其中 192.168.0.0~192.168.255.255 为内部专用地址段。

C 类地址适用于教室、机房等小型网络，每个 C 类网络最多可以有 254 台计算机。这类地址是所有地址类型中地址数最多的，但这类网络所允许连接的计算机数也是最少的。下面是一些 C 类网络号。

```
204 . 238 . 7 . 0
192 . 153 . 186 . 0
199 . 0 . 44 . . 0
191 . 0 . 0 . 0
222 . 222 . 31 . 0
```

● D 类 IP 地址。

D 类地址用于多重广播组，一个多重广播组可能包括 1 台或多台主机，也可能没有主机。D 类地址的最高位为“1110”，如下所示。

```
1 1 1 0 n n n n n n n n n n n n n n n n n n n n n h h h h h h h h
```

其中第 1 段取值为“11100000~11101111”，转换成十进制即为 224~239，其余各位用于设定客户机参加的特定组，取值址范围为“224.0.1.1~239.255.255.255”。在多重广播操作中没有网络或主机位，数据包将被传送到网络中选定的主机子集中，只有注册了多重广播地址的主机才能接收到数据包。

● E 类地址。

E 类地址是实验性地址，保留作为以后使用。E 类地址的最高位为“1111”，第 1 段取值为“11110000~11110111”之间，转换成十进制即为 240~247。248~254 暂无规定。

A 类、B 类、C 类网络是最常使用的网络，其地址结构组成如图 8-9 所示。

A 类、B 类、C 类网络地址特点如表 8-4 所示。

A 类	网络	主机	主机	主机
B 类	网络	网络	主机	主机
C 类	网络	网络	网络	主机

图 8-9 A 类、B 类、C 类网络地址结构

表 8-4 A 类、B 类、C 类网络地址特点

类 别	网络标识位	网 络 位 数	主 机 位 数	网 络 数	地 址 数
A	0	8	24	126	16 777 214
B	10	16	16	16 384	65 534
C	110	24	8	2 097 152	254

8.4.3 子网与子网掩码 (subnet mask)

子网掩码主要用于标识子网是如何划分的，即地址的哪一部分包含子网的网络号，哪一部分是网络中的主机号。掩码与 IP 地址一样，由 32 位组成，用点分十进制来描述。缺省情况下，掩码包含两个域：网络域和主机域，分别对应网络号和主机号。

通常将 IP 地址的网络位号全改为“1”，将主机位号全改为“0”，即为该 IP 地址的子网掩码。例如，将 IP 地址为“192.168.57.128”的网络位号“192.168.57”全改为“1”，将主机位号“57”全改为“0”，可得该 IP 地址的子网掩码为“255.255.255.0”。

A 类、B 类、C 类地址的标准子网掩码分别为：A 类为 255.0.0.0，B 类为 255.255.0.0，C 类为 255.255.255.0。例如，某 IP 地址为“172.16.56.45”，由于首字段为“172”，属于 B 类地址“128~191”范围内，所以该 IP 地址为 B 类地址，由此可推得其子网掩码为“255.255.0.0”，与“172.16.56.45”进行逻辑与运算，可得该地址所在网络的网络号为“172.16”。

利用子网掩码和 IP 地址，也可以计算 IP 地址对应的网络号。例如，已知某 IP 地址为“10.1.1.182”，子网掩码为“255.0.0.0”，则通过将“10.1.1.182”和“255.0.0.0”进行逻辑与运算，可得该 IP 地址的网络号为“10”。

8.4.4 广播地址 (broadcast address)

广播地址使用户能将消息一次性传递到自己所在网络的全体成员中。广播地址设定规则为：主机部分被设置为 255（二进制位全为 1），网络部分保持不变。例如，某 IP 地址为 192.68.56.6，其中“192.168.56”表示网络地址，占用了前 24 位，后 8 位是主机地址。将后 8 位全部设置为 1 就可以得到其广播地址为“192.168.56.255”。

8.4.5 网关地址 (gateway)

主机的 IP 地址设置正确后可以和同网段的其他主机进行通信，但还不能与不同网段的外网主机进行直接通信。为了与外部网络进行通信，需要正确设置网关地址。

网关通常是提供外部网络连接的路由器，一般至少有两个网络接口：一个连接局域网，另一个提供外网连接。

对于需要连接外部网络的主机，需要正确设置本地局域网内至少一个网关的 IP 地址，任何不同网段主机间进行的通信都将通过网关进行。

8.4.6 域名服务器地址 (DNS)

仅仅正确设置了 IP 地址和网关地址，只能保证用户通过 IP 地址和其他主机进行通信。



为了能够使用更为简易的主机域名进行通信，需要指定至少一个 DNS 服务器的 IP 地址。所有的域名解析任务都会通过指定的 DNS 服务器来完成。

实际上，DNS 服务器上存放着主机 IP 地址与其主机域名之间的对应关系。DNS 服务器收到域名解析请求之后，就将域名解析为 IP 地址，然后反馈回去。使用域名将大大减轻用户记忆的负担。例如，访问北京理工大学，就可以输入地址 `http://www.bit.edu.cn`，而不用记忆其 IP 地址 `202.112.144.31`。

8.4.7 DHCP 服务器

网络中每一台计算机都拥有惟一的 IP 地址。主机的 IP 地址可分为静态地址和动态地址两类。

静态地址一般由用户手工设定（指定 IP 地址、子网掩码等），设定成功后永久生效。为了保证正常的网络通信，静态地址的设定通常需要咨询网络管理员。

动态地址则由用户指定的 DHCP 服务器负责自动分配。用户每次接入网络时，由 DHCP 服务器从其地址池中动态选择一个没被使用的 IP 地址分配给该用户临时使用。当用户退出网络时（关机或断开网络连接），所使用的 IP 地址会被释放，可由 DHCP 服务器重新分配。



DHCP 是动态主机分配协议，主要用于简化主机 IP 地址的分配和管理。用户可以利用 DHCP 服务器管理动态分配的 IP 地址及其他相关的网络配置，如 DNS、WINS 等。

8.5 网络管理常用命令

Red Hat Linux 9 中提供了大量的网络管理命令，利用这些命令可以对网络进行快速配置，也可以协助诊断网络故障。

8.5.1 apachectl 命令：apache HTTP 服务器控制接口

apachectl 命令是 apache HTTP 服务器的一个控制接口。通过 apachectl 命令可以控制 apache HTTP 服务器的开启、关闭、重启等操作。apachectl 命令的语法格式如下所示。

```
apachectl [configtest][ graceful][ fullstatus][ restart][ start][ stop][ help][ status]
```

其中各选项的具体说明如表 8-5 所示。

表 8-5 apachectl 命令的选项说明

选 项	说 明
configtest	执行配置文件语法检测
graceful	通过发送 SIGUSR1 信号，重启 apache 服务，保留原有连接
fullstatus	列出服务器的详细状态信息
restart	重启 apache 服务器

续表

选 项	说 明
start	启动服务器
stop	关闭服务器
help	帮助信息
status	显示服务器的主要状态报告

例如，启动 apache HTTP 服务器，命令如下所示。

```
# apachectl start //启动 apache HTTP 服务器
```

例如，重新启动 apache HTTP 服务器，命令如下所示。

```
# apachectl restart //重新启动 apache HTTP 服务器
```

例如，检查 apache HTTP 服务器配置文件的语法是否正确，命令如下所示。

```
# apachectl configtest //检查配置文档的语法是否正确
Syntax OK
```

例如，关闭 apache HTTP 服务器，命令如下所示。

```
# apachectl stop //关闭 apache HTTP 服务器
```

8.5.2 arp 命令：控制系统 ARP 缓存

arp 命令可以实现从 IP 地址到以太网 MAC 地址之间的转换，其语法格式如下所示。

```
arp [-a [hostname], --display [hostname]] [-n, --numeric] [-e] [-s hostname hw_addr,
--set hostname]
[-d hostname, --delete hostname] [-i If, --device If] [-f filename, --file filename] [-v,
--verbose]
```

其中各选项的主要说明如表 8-6 所示。

表 8-6 arp 命令主要选项说明

选 项	说 明
-a [hostname], --display [hostname]	以 BSD 默认格式显示 arp 表中所有记录项
-n, --numeric	显示 IP 地址
-e	以 Linux 默认格式显示 arp 表中所有记录项
-s hostname hw_addr, --set hostname	设置一个新的 arp 记录项
-d hostname, --delete hostname	删除一个 arp 记录项
-i If, --device If	指定网络接口
-f filename, --file filename	从指定文件读取新的记录项
-v, --verbose	冗余模式

例如，查询 arp 表中的所有记录项，命令如下所示。

```
# arp -a
Computer1.bit.edu.cn (192.168.255.1) at 00:50:56:C0:00:01 [ether] on eth0
Computer2.bit.edu.cn (192.168.255.5) at 00:50:DA:8C:00:46 [ether] on eth0
Computer3.bit.edu.cn (192.168.255.5) at 00:E0:D0:18:A6:C7 [ether] on eth0
```

以 Linux 默认格式显示所有的 arp 表记录项，命令如下所示。

```
# arp -e
Address          HWtype  HWaddress      Flags Mask    Iface
```



192.168.255.254	ether	00:50:56:E8:11:7E	C	eth0
192.168.255.1	ether	00:50:56:C0:00:01	C	eth0

8.5.3 arpwatch 命令：监听 ARP 记录

arpwatch 命令跟踪 MAC 地址和 IP 地址的对应情况，并通过 E-mail 的形式报告当前的变化。arpwatch 使用 pcap 来侦听本地网卡的 arp 数据包。该命令的语法格式如下所示。

```
arpwatch [-d] [-N] [-f datafile] [-i interface] [-n] [-r file] [-u] [-e username] [-s username]
```

其中各选项的主要说明如表 8-7 所示。

表 8-7 arpwatch 命令主要选项说明

选 项	说 明
-d	允许调试
-N	关闭报告
-f datafile	指定 MAC 地址与 IP 地址数据库文件，默认为 arp.dat
-i interface	指定监听的网卡
-n	指定附加的本地网络
-r file	不从网络上监听 ARP 信息，而是从文件中读取 ARP 的记录信息
-u	指定用户和用户组
-e username	arpwatch 将发送邮件到指定用户，而不是默认的用户 root
-s username	arpwatch 将指定的用户名作为返回地址发送邮件，而不是默认的用户 root

arp 命令涉及的系统信息如表 8-8 所示。

表 8-8 arp 命令常见系统信息

信 息	说 明
ethernet broadcast	主机的 MAC 地址是广播地址
ip broadcast	主机的 IP 地址是广播地址
bogon	源 IP 地址不是本地子网地址
ethernet mismatch	源 MAC 地址与 ARP 数据包里的地址不匹配
reused old ethernet address	MAC 地址发生变化
suppressed DECnet flip flop	禁止“flip flop”报告

例如，监听网卡 eth0 的 ARP 信息，命令如下所示。

```
# arpwatch -i eth0 //监听网卡 eth0 上的 ARP 信息
```

例如，监听网卡 eth0 的 ARP 信息，并将相关信息记录到文件 arp.log 中，命令如下所示。

```
# arpwatch -i eth0 -f arp.log //监听网卡 eth0，将监听到的信息记录到 arp.log 文件中
```

8.5.4 arping 命令：向邻居主机发送 ARP 请求

arping 命令可以在指定网卡上向指定的相邻主机发送 ARP 请求。该命令的语法格式如下所示。


```
arping [-A][ -b][ -D][ -f][ -h][ -q][ -U][ -V][ -c count][ -w deadline][ -I interface][ -s source]
```

其中各选项的主要说明如表 8-9 所示。

表 8-9 arping 命令主要选项说明

选 项	说 明
-A	与-U 相似，但使用 ARP 回复包，而不是 ARP 请求包
-b	仅由 MAC 级广播发送
-D	复制地址检测模式
-f	得到第一个回复时就退出
-h	帮助信息
-q	运行命令时，不显示调试信息
-U	主动的 ARP 模式，不要求回复
-V	版本信息
-c count	发送指定数目的 ARP 请求包
-w deadline	设置超时时间
-I interface	使用指定的以太网设备发送 ARP 请求包，默认情况下使用 eth0
-s source	指定源 IP 地址

例如，向指定主机 192.168.20.147 发送 ARP 请求，命令如下所示。

```
# arping 192.168.20.147 //向 IP 地址为 192.168.20.147 的主机发送 ARP 请求
ARPING 192.168.20.147 from 192.168.20.122 eth0
Unicast reply from 192.168.20.147 [00:13:46:72:98:E6] 3.310ms
Unicast reply from 192.168.20.147 [00:E0:4C:E0:48:38] 37.110ms
Unicast reply from 192.168.20.147 [00:E0:4C:E0:48:38] 126.480ms
Unicast reply from 192.168.20.147 [00:E0:4C:E0:48:38] 134.795ms
Unicast reply from 192.168.20.147 [00:E0:4C:E0:48:38] 96.589ms
Unicast reply from 192.168.20.147 [00:E0:4C:E0:48:38] 176.594ms
Unicast reply from 192.168.20.147 [00:E0:4C:E0:48:38] 112.922ms
Sent 6 probes (1 broadcast(s))
Received 7 response(s)
```

当运行以上命令时，会向指定主机 192.168.20.147 连续不停地发送 ARP 数据包，直到用户手动终止。

例如，向指定主机 192.168.20.147 发送 ARP 请求，当收到第一个 ARP 回复包时，退出 arping，命令如下所示。

```
# arping -f 192.168.20.147 //向指定主机发送 ARP 请求包，收到第一个包后退出
ARPING 192.168.20.147 from 192.168.88.76 eth0
Unicast reply from 192.168.20.147 [00:13:46:72:98:E6] 0.954ms
Sent 1 probes (1 broadcast(s))
Received 1 response(s)
#
```

例如，向指定主机 192.168.20.147 发送 ARP 请求，并设定发送 ARP 的包的数目为 5，命令如下所示。



```
# arping -c 5 192.168.20.147 //向指定主机发送 ARP 数据包，发送数据包的个数为 5
ARPING 192.168.20.147 from 192.168.88.76 eth0
Unicast reply from 192.168.20.147 [00:13:46:72:98:E6] 15.842ms
Unicast reply from 192.168.20.147 [00:E0:4C:E0:48:38] 144.305ms
Unicast reply from 192.168.20.147 [00:E0:4C:E0:48:38] 96.410ms
Unicast reply from 192.168.20.147 [00:E0:4C:E0:48:38] 127.986ms
Unicast reply from 192.168.20.147 [00:E0:4C:E0:48:38] 157.652ms
Unicast reply from 192.168.20.147 [00:E0:4C:E0:48:38] 181.159ms
Sent 5 probes (1 broadcast(s))
Received 6 response(s)
#
```

例如，向指定主机 192.168.20.147 发送数据包，但不输出相关的调试信息，命令如下所示。

```
# arping -q 192.168.20.147 //向主机 192.168.20.147 发送 ARP 包，但不输出调试信息
```

8.5.5 cu 命令：调用其他主机系统

cu 命令类似于终端播号程序，可以调用其他主机系统，进行主机间的通信，也可以实现没有校验的文件传输。该命令的语法格式如下所示。

```
cu [-d][ -e, --parity=even][ -h, --halfduplex][ -n, --prompt][ -o, --parity=
odd][ --parity=none][ --nostop] [ -a port]
[ -c phone-number, --phone phone-number][ -E char, --escape char][ -I file, --config
file][ -l line, --line line]
[ -s speed, --speed speed][ -v, --versio][ -x type, --debug type][ -z system, --system
system]
[ -p port, --port port] [ --help] [--parity=none]
```

其中各选项的主要说明如表 8-10 所示。

表 8-10 cu 命令主要选项说明

选 项	说 明
-d	调试模式
-e, --parity=even	使用偶数检验
-h, --halfduplex	半双工模式
-n, --prompt	交互模式，等待用户输入电话号码
-o, --parity=odd	奇数检验
--parity=none	不使用奇偶检验
--nostop	关闭流量控制
-c phone-number, --phone phone-number	指定电话号码
-E char, --escape char	指定脱离字符
-I file, --config file	从指定的配置文件中读取文件
-l line, --line line	指定线路
-s speed, --speed speed	设置连线速率
-v, --version	版本信息
-x type, --debug type	打开部分调试

续表

选 项	说 明
-z system, --system system	指定连接调用的主机
-p port, --port port	指定端口
-a port	等同于--port port
--help	版主信息
--parity=none	关闭同位检测

例如，显示 cu 命令的版本信息，命令如下所示。

```
# cu -v
cu: Taylor UUCP 1.06.1, copyright (C) 1991, 92, 93, 94, 1995 Ian Lance Taylor
```

8.5.6 gaim 命令：即时信息传输客户端

gaim 命令用于即时信息的传输。gaim 是基于 GTK 的即时信息（IM）客户端，主要针对 AMI，但也支持流行的 IM 协议，如 MSN、jabber、Yahoo、ICQ、IRC、Napster、Zephyr 和 Gadu-Gadu。gaim 命令的语法格式如下所示。

```
gaim [-a, --acct][ -d, --debug][ -f, --file=FILE][ -u, --user=NAME][ -h, --help][ -v, --version]
```

其中各选项的主要说明如表 8-11 所示。

表 8-11 gaim 命令主要选项说明

选 项	说 明
-a, --acct	开启账号编辑器而不是登录窗口
-d, --debug	调试信息
-f, --file=FILE	指定配置文件，默认为 ~/.gaimrc
-u, --user=NAME	使用指定的账户名
-h, --help	帮助信息
-v, --version	版本信息

例如，通过命令 gaim，启动 gaim，如下所示。

```
# gaim //启动 gaim
```

8.5.7 hostname 命令

hostname 命令用来显示或设置当前系统的主机名，这个名字被许多网络程序用来标识主机。该命令的语法格式如下所示。

```
hostname [-s, --short][ -a, --alias][ -i, --ip-address][ -f, --fqdn, --long][ -d, --domain][ -y, --yp, --nis][ -v, --verbose]
[ -F, --file filename][ --help][ -V, --version]
```

其中各选项的具体说明如表 8-12 所示。



表 8-12 hostname 命令的选项说明

选 项	说 明
-s, --short	短主机名
-a, --alias	别名
-i, --ip-address	IP 地址
-f, --fqdn, --long	长主机名
-d, --domain	DNS 域名
-y, --yp, --nis	NIS 域名
-v, --verbose	运行时，显示详细的处理过程
-F, --file filename	从指定文件读取主机名
--help	帮助信息
-V, --version	版本信息

例如，通过命令 `hostname`，显示主机名称，如下所示。

```
# hostname //显示主机名
localhost.localdomain
#
```

例如，显示短主机名，命令如下所示。

```
# hostname -s //显示短主机名
localhost
#
```

例如，显示主机别名，命令如下所示。

```
# hostname -a //显示主机别名
localhost
#
```

例如，显示主机 IP 地址，命令如下所示。

```
# hostname -i //显示主机 IP 地址
127.0.0.1
#
```

例如，显示主机域名，命令如下所示。

```
# hostname -d //显示主机域名
localhost
#
```

例如，显示 NIS 域名，命令如下所示。

```
# hostname -y //显示 NIS 域名
(none) //未设置
#
```

例如，设置主机名为 `teacherhost`，命令如下所示。

```
# hostname teacherhost //设置主机名称
# hostname
teacherhost
#
```

例如，使用 `hostname` 命令修改当前系统的主机名称为 `redhat9`，如下所示。

```
# hostname redhat9
# hostname
redhat9
```


8.5.8 host 命令：dns 查询

host 命令是一个执行 DNS 查询的简单工具，可以将指定的名字转换为 IP 地址。当不指定参数时，将显示 host 命令用法的摘要。该命令的语法格式如下所示。

```
host [ -aCdlnrTwv ] [ -c class ] [ -N ndots ] [ -R number ] [ -t type ] [ -W wait ]
name [ server ]
```

其中各选项的主要说明如表 8-13 所示。

表 8-13 host 命令主要选项说明

选 项	说 明
-a	等于-v
-C	在所有列出的认证域名服务器上查找 SOA 记录
-d	等于-v
-l	列出模式
-n	根据 RFC1886，反向查询 IPv6 地址使用 IP6.INT 域名和“nibble”标签
-i	反向查找
-N	设备在名称中使用的点数
-r	不使用递归处理
-R	指定一次查询使用的 UDP 包数
-T	默认使用 UDP，该选项设置使用 TCP 进行查询
-v	运行时，显示详细的处理信息
-w	永远等待回复
-W	指定等待回复的时间
-4	仅用于 IPv4 的传输查询
-6	仅用于 IPv6 的传输查询
-c	指定类的 DNS 查询
-t	指定查询类型，包括 a、all、mx、ns

例如，显示命令 host 的帮助信息，命令如下所示。

```
# host //host 帮助信息
Usage: host [-aCdlriTwv] [-c class] [-N ndots] [-t type] [-W time]
        [-R number] hostname [server]
-a is equivalent to -v -t *
-c specifies query class for non-IN data
-C compares SOA records on authoritative nameservers
-d is equivalent to -v
-l lists all hosts in a domain, using AXFR
-i IP6.INT reverse lookups
-N changes the number of dots allowed before root lookup is done
-r disables recursive processing
-R specifies number of retries for UDP packets
-t specifies the query type
-T enables TCP/IP mode
```



```
-v enables verbose output
-w specifies to wait forever for a reply
-W specifies how long to wait for a reply
-4 use IPv4 query transport only
-6 use IPv6 query transport only
```

例如, 通过命令 `host`, 查询主机 `www.bit.edu.cn` 的 IP 地址, 通过 IP 地址 `202.112.144.31`, 查询主机的域名, 命令如下所示。

```
# host www.bit.edu.cn           //查询主机名称
www.bit.edu.cn has address 202.204.80.38
# host 202.204.80.38           //查询 IP 地址
Host 38.80. 204.202.in-addr.arpa not found: 3(NXDOMAIN)
#
```

例如, 通过命令 `host`, 查询详细的 DNS 信息, 可以看到有两个 DNS 服务器, 如下所示。

```
# host -a www.bit.edu.cn       //查询详细的 DNS 信息
Trying "www.bit.edu.cn"
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 50411
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 2
;; QUESTION SECTION:
;www.bit.edu.cn.             IN      ANY
;; ANSWER SECTION:
www.bit.edu.cn.             10800   IN      A       202.204.80.38
;; AUTHORITY SECTION:
bit.edu.cn.                 10800   IN      NS      dns.bit.edu.cn.
bit.edu.cn.                 10800   IN      NS      sns.bit.edu.cn.
;; ADDITIONAL SECTION:
dns.bit.edu.cn.             10800   IN      A       202.204.80.30
sns.bit.edu.cn.             10800   IN      A       202.204.80.65
Received 127 bytes from 202.204.80.38#53 in 10 ms
```

例如, 使用指定的 DNS 服务器 `jupiter.njtu.edu.cn` 查询主机 `www.bjtu.edu.cn`, 命令如下所示。

```
# host -a www.bjtu.edu.cn jupiter.njtu.edu.cn //指定服务器进行 DNS 查询
Trying "www.bjtu.edu.cn"
Using domain server:
Name: jupiter.njtu.edu.cn
Address: 202.112.144.65#53
Aliases:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 15236
;; flags: qr rd; QUERY: 1, ANSWER: 0, AUTHORITY: 5, ADDITIONAL: 7
;; QUESTION SECTION:
;www.bjtu.edu.cn.          IN      ANY
;; AUTHORITY SECTION:
edu.cn.                    85190   IN      NS      NS2.NET.edu.cn.
edu.cn.                    85190   IN      NS      NS2.CUHK.HK.
edu.cn.                    85190   IN      NS      dns2.edu.cn.
edu.cn.                    85190   IN      NS      DENEK.DFN.DE.
edu.cn.                    85190   IN      NS      DNS.edu.cn.
;; ADDITIONAL SECTION:
DNS.edu.cn.                171217  IN      A       202.112.0.35
DNS.edu.cn.                171217  IN      AAAA    2001:da8:1:100::20
NS2.NET.edu.cn.            84817   IN      A       202.112.0.33
NS2.NET.edu.cn.            84817   IN      AAAA    2001:250:c006::2
NS2.CUHK.HK.               55619   IN      A       137.189.6.21
dns2.edu.cn.               85190   IN      A       202.112.0.13
DENEK.DFN.DE.              84817   IN      A       192.76.176.9
Received 279 bytes from 202.112.144.65#53 in 10 ms
```


例如，通过命令“host -t MX bjtu.edu.cn”，查询邮件交换机的记录，如下所示。

```
# host -t MX bjtu.edu.cn           //查询邮件交换机记录
bjtu.edu.cn mail is handled by 10 smg.njtu.edu.cn.
#
```

8.5.9 httpd 命令：apache http 服务器程序

httpd 是一个 apache 超文本传输协议的服务器程序，通常其作为一个独立的守护进程来运行。当使用时，httpd 会根据需要创建一个子进程池或多个线程来处理请求。该命令的语法格式如下所示。

```
httpd [-t ] [-D parameter ] [-d serverroot ] [-f config ] [-c directive ] [ -h ]
[ -l ][ -V][ -E file] [-L][ -e level]
[ -t -D DUMP_VHOSTS]
```

其中各选项的主要说明如表 8-14 所示。

表 8-14 httpd 命令主要选项说明

选 项	说 明
-t	对配置文件进行语法检查
-D parameter	设置配置参数
-d serverroot	指定服务根目录
-f config	指明配置文件中的命令
-C directive	在读取配置文件前处理指示
-c directive	读取配置文件之后处理指示
-h	显示简要帮助信息
-l	显示编辑到服务器中的模块列表
-V	显示版本信息
-e level	显示启动错误等级
-t -D DUMP_VHOSTS	根据配置文件中的解析结果，显示配置信息
-L	显示可用的配置指令
-E file	将启动错误写入指定的文件

例如，通过命令 httpd，检查配置文件的语法错误，如下所示。

```
# httpd -t           //检查配置文件的语法错误
Syntax OK
#
```

例如，通过命令 httpd，启动 httpd 进程，如下所示。

```
# ps -A|grep httpd    //查找 httpd 进程，发现未启动
# httpd               //启动 httpd
# ps -A|grep httpd    //查找 httpd 进程
11414 ?      00:00:04 httpd
11427 ?      00:00:00 httpd
11428 ?      00:00:00 httpd
11429 ?      00:00:00 httpd
11430 ?      00:00:00 httpd
11431 ?      00:00:00 httpd
```



```
11432 ?      00:00:00 httpd
11433 ?      00:00:00 httpd
11434 ?      00:00:00 httpd
```

例如，通过命令 `httpd`，显示编译模块列表，如下所示。

```
# httpd -l          //显示编译模块
Compiled in modules:
  core.c
  prefork.c
  http_core.c
  mod_so.c
```

例如，显示 `httpd` 的可用配置命令信息。通过命令 `httpd` 的选项 “-L”，显示配置命令的信息，并将结果重定向到文件 `temp` 中，再通过 `tail` 命令显示文件 `temp` 的最后 20 行信息。命令执行过程如下所示。

```
# httpd -L > temp | tail -n 20 temp          //显示配置文件信息，输出最后 20 行
Maximum number of children alive at the same time
Allowed in *.conf only outside <Directory>, <Files> or <Location>
ServerLimit (prefork.c)
Maximum value of MaxClients for this run of Apache
Allowed in *.conf only outside <Directory>, <Files> or <Location>
KeepAliveTimeout (http_core.c)
Keep-Alive timeout duration (sec)
Allowed in *.conf only outside <Directory>, <Files> or <Location>
MaxKeepAliveRequests (http_core.c)
Maximum number of Keep-Alive requests per connection, or 0 for infinite
Allowed in *.conf only outside <Directory>, <Files> or <Location>
KeepAlive (http_core.c)
Whether persistent connections should be On or Off
Allowed in *.conf only outside <Directory>, <Files> or <Location>
LoadModule (mod_so.c)
a module name and the name of a shared object file to load it from
Allowed in *.conf only outside <Directory>, <Files> or <Location>
LoadFile (mod_so.c)
shared object file or library to load into the server at runtime
Allowed in *.conf only outside <Directory>, <Files> or <Location>
```

8.5.10 ifup 命令

`ifup` 命令用于启动指定的非活动网卡，与 “`ifconfig up`” 命令相似。例如，启动本地回环网络接口，命令如下所示。

```
# ifup lo
# ifconfig lo
lo      Link encap:Local Loopback
        inet addr:127.0.0.1  Mask:255.0.0.0
        UP LOOPBACK RUNNING MTU:16436 Metric:1
        RX packets:1357002 errors:0 dropped:0 overruns:0 frame:0
        TX packets:1357002 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:0
        RX bytes:92723242 (88.4 Mb)  TX bytes:92723242 (88.4 Mb)
```

8.5.11 ifdown 命令

`ifdown` 命令用于关闭指定的活动网卡，与 “`ifconfig down`” 命令相似。例如，关闭本地

回环网络接口，命令如下所示。

```
# ifdown lo                                //关闭本地回环接口
# ifconfig lo                               // 查看 lo 是否已处于关闭状态
lo      Link encap:Local Loopback
        LOOPBACK MTU:16436 Metric:1        //lo 已处于关闭状态
        RX packets:1358256 errors:0 dropped:0 overruns:0 frame:0
        TX packets:1358256 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:0
        RX bytes:92809009 (88.5 Mb) TX bytes:92809009 (88.5 Mb)
```

8.5.12 ifconfig 命令：显示或配置网络设备

ifconfig 命令类似于 Windows 下的 ipconfig，用于获取和修改网络接口配置信息。当系统启动时，将调用此命令配置网络设备。此外，当进行调试或者系统的网络环境发生变化时，可以调用此命令进行相应的网络配置。ifconfig 命令的语法格式如下所示。

```
ifconfig [-a] [-i] [-v] [-s] <interface> [[<AF>] <address>]
        [add <address>[/<prefixlen>]]
        [del <address>[/<prefixlen>]]
        [[-]broadcast [<address>]] [[-]pointopoint [<address>]]
        [netmask <address>] [dstaddr <address>] [tunnel <address>]
        [outfill <NN>] [keepalive <NN>]
        [hw <HW> <address>] [metric <NN>] [mtu <NN>]
        [[-]trailers] [[-]arp] [[-]allmulti]
        [multicast] [[-]promisc]
        [mem_start <NN>] [io_addr <NN>] [irq <NN>] [media <type>]
        [txqueuelen <NN>]
        [[-]dynamic]
        [up|down] ...
```

ifconfig 命令可以指定许多选项，主要选项的说明如表 8-15 所示。

表 8-15 ifconfig 命令主要选项说明

选 项	说 明
-a	显示所有接口信息，包括活动和非活动的
-v	以冗余模式显示详细信息
-s	以短列表格式显示接口信息，每个接口只显示一行摘要数据
up	激活一个不活动的接口
down	与 up 相反，关闭一个接口
netmask addr	为一个指定接口设置网络掩码
[-]broadcast [addr]	为一个指定接口设置广播地址
irq addr	设置 IRQ 地址
mem_start addr	设置共享内存的起始地址
media type	设置介质类型
address	设置指定接口的 IP 地址
dstaddr addr	对于点对点链接，设置远端 IP 地址
interface	显示一个指定接口的信息



续表

选 项	说 明
[-]promisc	设置是否支持网卡的混杂模式，如果选择此选项，网卡将接收网络中的所有数据包
io_addr addr	设置 IO 地址空间的起始地址
[-]arp	设置指定的网卡是否支持 ARP 协议
[-]allmulti	设置是否支持多播模式，如果选择此选项，网卡将接收网络中所有的多播数据包
add addr/prefixlen	给指定网卡配置 IPv6 地址
del addr/prefixlen	删除指定网卡的 IPv6 地址
tunnel aa.bb.cc.dd	创建一个新的 SIT 设备
hw class address	配置网卡的硬件地址
mtu N	设置网卡的传输单元
txqueuelen length	设置发送队列的长度
[-]pointopoint [addr]	点对点模式
multicast	为网卡设置组播标志
metric N	设置接口的单位度量

例如，不带任何选项使用 `ifconfig` 命令，可以显示当前系统中活动的网卡信息，如下所示。

```
# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0C:29:1B:EA:CC
          inet addr:192.168.20.122  Bcast:192.168.20.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:39218  errors:0  dropped:0  overruns:0  frame:0
          TX packets:33358  errors:0  dropped:0  overruns:0  carrier:0
          collisions:0 txqueuelen:100
          RX bytes:2644538 (2.5 Mb)  TX bytes:12293795 (11.7 Mb)
          Interrupt:10 Base address:0x2000
```

例如，使用“`ifconfig -a`”命令可以显示系统中所有网卡的信息，包括活动和非活动的，如下所示。

```
# ifconfig -a
eth0      Link encap:Ethernet  HWaddr 00:0C:29:1B:EA:CC
          inet addr:192.168.20.122  Bcast:192.168.20.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:39209  errors:0  dropped:0  overruns:0  frame:0
          TX packets:33352  errors:0  dropped:0  overruns:0  carrier:0
          collisions:0 txqueuelen:100
          RX bytes:2643998 (2.5 Mb)  TX bytes:12292586 (11.7 Mb)
          Interrupt:10 Base address:0x2000

lo        Link encap:Local Loopback
          LOOPBACK MTU:16436  Metric:1
          RX packets:1358256  errors:0  dropped:0  overruns:0  frame:0
          TX packets:1358256  errors:0  dropped:0  overruns:0  carrier:0
          collisions:0 txqueuelen:0
          RX bytes:92809009 (88.5 Mb)  TX bytes:92809009 (88.5 Mb)
```

使用“`ifconfig [interface]`”命令可以显示指定接口的信息（无论该接口是否处于活动状

态)。例如, 显示回环网络接口信息, 命令如下所示。

```
# ifconfig lo
lo      Link encap:Local Loopback
        inet addr:127.0.0.1  Mask:255.0.0.0
        UP LOOPBACK RUNNING  MTU:16436  Metric:1
        RX packets:12181919 errors:0 dropped:0 overruns:0 frame:0
        TX packets:12181919 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:0
        RX bytes:841096584 (802.1 Mb)  TX bytes:841096584 (802.1 Mb)
```

使用“ifconfig [interface] down”命令关闭指定接口。例如, 关闭本地回环网络接口, 命令如下所示。

```
# ifconfig          //显示当前接口状态信息
eth0      Link encap:Ethernet  HWaddr 00:0C:29:1B:EA:CC          //接口eth0信息
        inet addr:192.168.20.122  Bcast:192.168.20.255  Mask:255.255.255.0
        UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
        RX packets:322177 errors:0 dropped:0 overruns:0 frame:0
        TX packets:289496 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:100
        RX bytes:26535001 (25.3 Mb)  TX bytes:54769313 (52.2 Mb)
        Interrupt:10 Base address:0x2000

lo      Link encap:Local Loopback          //本地回环接口信息
        inet addr:127.0.0.1  Mask:255.0.0.0
        UP LOOPBACK RUNNING  MTU:16436  Metric:1
        RX packets:12182451 errors:0 dropped:0 overruns:0 frame:0
        TX packets:12182451 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:0
        RX bytes:841132970 (802.1 Mb)  TX bytes:841132970 (802.1 Mb)

# ifconfig lo down          //关闭显示本地回环接口
# ifconfig          //显示当前接口状态信息
eth0      Link encap:Ethernet  HWaddr 00:0C:29:1B:EA:CC          //eth0 接口信息
        inet addr:192.168.20.122  Bcast:192.168.20.255  Mask:255.255.255.0
        UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
        RX packets:322213 errors:0 dropped:0 overruns:0 frame:0
        TX packets:289519 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:100
        RX bytes:26537845 (25.3 Mb)  TX bytes:54772551 (52.2 Mb)
        Interrupt:10 Base address:0x2000

#
```

与之相反, 使用“ifconfig [interface] up”命令可以启动指定接口。例如, 启动本地回环网络接口, 命令如下所示。

```
# ifconfig          //显示接口信息
eth0      Link encap:Ethernet  HWaddr 00:0C:29:1B:EA:CC
        inet addr:192.168.20.122  Bcast:192.168.20.255  Mask:255.255.255.0
        UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
        RX packets:322644 errors:0 dropped:0 overruns:0 frame:0
        TX packets:289751 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:100
        RX bytes:26580011 (25.3 Mb)  TX bytes:54803094 (52.2 Mb)
        Interrupt:10 Base address:0x2000
```



```
# ifconfig lo up           //启动本地回环接口
# ifconfig                 //显示接口信息
eth0    Link encap:Ethernet HWaddr 00:0C:29:1B:EA:CC           //eth0 接口信息
        inet addr:192.168.20.122 Bcast:192.168.20.255 Mask:255.255.255.0
        UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
        RX packets:322681 errors:0 dropped:0 overruns:0 frame:0
        TX packets:289780 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:100
        RX bytes:26582839 (25.3 Mb) TX bytes:54806496 (52.2 Mb)
        Interrupt:10 Base address:0x2000

lo      Link encap:Local Loopback                             //已启动本地回环接口
        inet addr:127.0.0.1 Mask:255.0.0.0
        UP LOOPBACK RUNNING MTU:16436 Metric:1
        RX packets:12247891 errors:0 dropped:0 overruns:0 frame:0
        TX packets:12247891 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:0
        RX bytes:845608489 (806.4 Mb) TX bytes:845608489 (806.4 Mb)
```

使用“ifconfig [interface] address”命令可以指定接口的 IP 地址。例如，指定 eth0 的 IP 地址为 192.168.20.122，命令如下所示。

```
# ifconfig eth0 192.168.20.122
# ifconfig eth0
eth0    Link encap:Ethernet HWaddr 00:0C:29:1B:EA:CC
        inet addr:192.168.20.122 Bcast:192.168.20.255 Mask:255.255.255.0
        UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
        RX packets:39683 errors:0 dropped:0 overruns:0 frame:0
        TX packets:33804 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:100
        RX bytes:2673560 (2.5 Mb) TX bytes:12334481 (11.7 Mb)
        Interrupt:10 Base address:0x2000
```

例如，为网卡接口 eth0 设置地址 192.168.20.7，子网掩码为 255.255.255.0，执行命令如下所示。

```
# ifconfig eth0 192.168.20.7 netmask 255.255.255.0 //为网卡 eth0 配置 IPv4 地址和子网掩码
```

例如，启动和关闭指定网卡 eth0，命令如下所示。

```
# ifconfig eth0 up           //启动网卡 eth0
# ifconfig eth0 down         //关闭网卡 eth0
```

例如，为网卡 eth0 配置和删除 IPv6 地址，命令如下所示。

```
# ifconfig eth0 add 3ffe:3240:8007:1005::2/64 //为网卡 eth0 配置 IPv6 地址
# ifconfig eth0 del 3ffe:3240:8007:1005::2/64 //删除网卡 eth0 的 IPv6 地址
```

例如，使用 ifconfig 命令配置一台安装 3 块网卡的主机，第 1 块网卡连接网络 192.168.214.0，第 2 块网卡连接网络 192.168.202.0，第 3 块网卡连接网络 192.168.200.0，则 3 块网卡的配置命令如下所示。

```
ifconfig eth0 192.168.214.0 broadcast 192.168.214.255 netmask 255.255.255.0
ifconfig eth1 192.168.202.0 broadcast 192.168.202.255 netmask 255.255.255.0
ifconfig eth2 192.168.200.0 broadcast 192.168.200.255 netmask 255.255.255.0
```

8.5.13 iptables 命令：ipv4 的包过滤和 nat 的管理

iptables 命令用来设置、维护或检测 linux 内核的 IP 数据包表。iptables 定义多个数据表，

每一个数据表包含了一定数量的链表，用户也可以自定义链表。每一个链表中定义了相应的数据包的处理规则。数据表包括以下几种类型。

- filter: 包过滤表。
- mangle: 高级表，用于完成指定的包替换。
- nat: 转发表。

iptables 命令的语法格式如下所示。

```
iptables [--append, -A chain][ -t, --table][ --delete, -D chain][ --flush, -F
[chain]][ --insert, -I chain]
[ --list, -L [chain]][ -R, --replace][ --new, -N chain][ -P, --policy chain target]
[ --version, -V]
[ --delete-chain -X [chain]][ --destination, -d][ --in-interface,
-i][ --out-interface , -o][ --verbose, -v]
[ --proto, -p][ --source, -s][ --match , -m match][ --jump, -j target][ --numeric ,
-n][ --line-numbers]
```

其中各选项的主要说明如表 8-16 所示。

表 8-16 命令主要选项说明

选 项	说 明
--append, -A chain	向链表中添加规则
-t, --table	指定使用的匹配数据表
--delete, -D chain	从链表中删除规则
--flush, -F [chain]	清除所有链表中的规则
--insert, -I chain	在链表中指定位置插入新规则
--list, -L [chain]	列出链表中的规则
-R, --replace	替换链表中的规则
--new, -N chain	新建用户自定义链表
-P, --policy chain target	设定指定对象的链表策略
--delete-chain -X [chain]	删除用户自定义链表
--destination, -d	目的地址
--in-interface, -i	选择网卡
--out-interface , -o	显示网卡名称
--proto, -p	选择通信协议
--source, -s	源地址
--match , -m match	扩展匹配
--jump, -j target	指定规则对象
--numeric , -n	以数字形式输出地址和端口号。
--line-numbers	显示行号
--verbose , -v	冗余模式
--version, -V	显示版本信息

例如，显示过滤表（filter 表）中的规则，命令如下所示。



```
# iptables -t filter -L //显示过滤表
Chain INPUT (policy ACCEPT)
target    prot opt source                destination
ACCEPT    0    --  anywhere              anywhere
ACCEPT    0    --  anywhere              anywhere
ACCEPT    icmp --  anywhere              anywhere
ACCEPT    udp  --  anywhere              anywhere      udp dpt:2304
ACCEPT    tcp  --  anywhere              anywhere      tcp dpts:ftp:telnet
ACCEPT    tcp  --  anywhere              anywhere      tcp dpt:www
ACCEPT    tcp  --  anywhere              anywhere      tcp dpt:8888
ACCEPT    0    --  anywhere              anywhere      state RELATED,ESTABLISHED
Chain FORWARD (policy ACCEPT)
target    prot opt source                destination
ACCEPT    0    --  anywhere              192.168.88.88
ACCEPT    0    --  192.168.88.0/24      anywhere
ACCEPT    0    --  anywhere              anywhere      state RELATED,ESTABLISHED
Chain OUTPUT (policy ACCEPT)
target    prot opt source                destination
```

例如，显示高级表中包含的规则，命令如下所示。

```
# iptables -t mangle -L //显示高级表
Chain PREROUTING (policy ACCEPT)
target    prot opt source                destination
Chain INPUT (policy ACCEPT)
target    prot opt source                destination
Chain FORWARD (policy ACCEPT)
target    prot opt source                destination
Chain OUTPUT (policy ACCEPT)
target    prot opt source                destination
Chain POSTROUTING (policy ACCEPT)
target    prot opt source                destination
```

例如，显示转发表中定义的规则，命令如下所示。

```
# iptables -t nat -L //显示转发包过滤表
Chain PREROUTING (policy ACCEPT)
target    prot opt source                destination
DNAT      tcp  --  anywhere              ip1ab604      tcp dpt:3389
to:192    .168.88.88:3389
Chain POSTROUTING (policy ACCEPT)
target    prot opt source                destination
MASQUERADE 0    --  192.168.88.0/24      anywhere
SNAT      0    --  192.168.88.88      anywhere      to:202.112.146.70
Chain OUTPUT (policy ACCEPT)
target    prot opt source                destination
```

例如，通过命令“iptables -P INPUT DROP”，设置 INPUT 链表的默认规则为 DROP，即与 INPUT 链的规则不匹配的数据包将被丢弃，如下所示。

```
# iptables -P INPUT DROP //设置 INPUT 链表的规则
```

例如，通过命令“iptables -P OUTPUT DROP”，设置 OUTPUT 链表的默认规则为 DROP，即与 OUTPUT 链表中的规则不匹配的数据包将被丢弃，如下所示。

```
# iptables -P OUTPUT DROP //设置 OUTPUT 链表的规则
```

例如，通过命令“iptables -P FORWARD DROP”，设置 FORWARD 链表的默认规则为 DROP，即与 FORWARD 链表的规则不匹配的数据包将被丢弃，如下所示。

```
# iptables -P FORWARD DROP //设置 FORWARD 链表的规则
```


例如，通过命令“iptables -A INPUT -s 192.168.20.7 -j ACCEPT”，添加一条新规则，接受源地址是 192.168.20.7 的数据包，如下所示。

```
# iptables -A INPUT -s 192.168.20.7 -j ACCEPT //添加规则
```

例如，通过命令“iptables -A INPUT -s 192.168.20.7 -j DROP”，添加一条新规则，丢弃 INPUT 中源地址是 192.168.40.7 的数据包，如下所示。

```
# iptables -A INPUT -s 192.168.40.7 -j DROP //添加规则
```

例如，通过命令“iptables -D INPUT --dport 80 -j DROP”，添加一条新规则，丢弃前往 80 端口的数据包，如下所示。

```
# iptables -D INPUT --dport 80 -j DROP //删除规则
```

例如，通过命令“iptables -D INPUT 4”，删除 INPUT 链中的第 4 条规则，如下所示。

```
# iptables -D INPUT 4 //删除规则
```

例如，通过命令“iptables -I INPUT 2 -s 192.168.20.7 -j ACCEPT”，在 INPUT 链中的 2 号规则的位置上插入一条新的规则，其内容是接受源于 192.168.20.7 的数据包，如下所示。

```
# iptables -I INPUT 1 -s 192.168.20.7 -j ACCEPT //插入规则
```

例如，通过命令“iptables -R INPUT 3 -s 192.168.88.7 -j ACCEPT”，更换 INPUT 链中的 3 号规则为新规则，其内容是接受源于 192.168.88.7 的数据包，如下所示。

```
# iptables -R INPUT 3 -s 192.168.88.7 -j ACCEPT //更换规则
```

8.5.14 iptables-save 命令：保存 ip 表

iptables-save 命令用于输出 IP table 表到标准输出。该命令的语法格式如下所示。

```
iptables-save [-c][-t]
```

其中各选项的主要说明如表 8-17 所示。

表 8-17 iptables-save 命令主要选项说明

选 项	说 明
-c	包含当前所有数据包的值和字节数
-t	指定 table 表类型

例如，通过命令 iptables-save，显示当前 iptables 的设置，如下所示。

```
iptables-save //显示当前 iptables 的设置
# Generated by iptables-save v1.3.6 on Thu May 24 04:40:16 2007
*mangle
:PREROUTING ACCEPT [834359:490508468]
:INPUT ACCEPT [66789:18462492]
:FORWARD ACCEPT [613800:454909818]
:OUTPUT ACCEPT [24584:15999458]
:POSTROUTING ACCEPT [638382:470909124]
COMMIT
# Completed on Thu May 24 04:40:16 2007
# Generated by iptables-save v1.3.6 on Thu May 24 04:40:16 2007
*nat
:PREROUTING ACCEPT [71744143:7018491963]
:POSTROUTING ACCEPT [32953:1853286]
:OUTPUT ACCEPT [32163:1791912]
-A PREROUTING -d 202.112.146.70 -p tcp -m tcp --dport 3389 -j DNAT --to-destination 192.168.88.88:3389
-A POSTROUTING -s 192.168.88.0/255.255.255.0 -o eth1 -j MASQUERADE
```




```
-A POSTROUTING -s 192.168.88.88 -j SNAT --to-source 202.112.146.70
COMMIT
# Completed on Thu May 24 04:40:16 2007
# Generated by iptables-save v1.3.6 on Thu May 24 04:40:16 2007
*filter
:INPUT ACCEPT [26710398:9570338513]
:FORWARD ACCEPT [15991:19168654]
:OUTPUT ACCEPT [4969768:2145883292]
-A INPUT -i lo -j ACCEPT
-A INPUT -i eth0 -j ACCEPT
-A INPUT -i eth1 -p icmp -j ACCEPT
-A INPUT -i eth1 -p udp -m udp --dport 2304 -j ACCEPT
-A INPUT -i eth1 -p tcp -m tcp --dport 21:23 -j ACCEPT
-A INPUT -i eth1 -p tcp -m tcp --dport 80 -j ACCEPT
-A INPUT -i eth1 -p tcp -m tcp --dport 8888 -j ACCEPT
-A INPUT -i eth1 -m state --state RELATED,ESTABLISHED -j ACCEPT
-A FORWARD -d 192.168.88.88 -j ACCEPT
-A FORWARD -s 192.168.88.0/255.255.255.0 -j ACCEPT
-A FORWARD -i eth1 -m state --state RELATED,ESTABLISHED -j ACCEPT
COMMIT
# Completed on Thu May 24 04:40:16 2007
```

例如，通过命令“iptables-save -t filter”，显示 filter 表的设置，如下所示。

```
# iptables-save -t filter //filter 表的设置
# Generated by iptables-save v1.3.6 on Thu May 24 04:44:31 2007
*filter
:INPUT ACCEPT [26711145:9570386824]
:FORWARD ACCEPT [15991:19168654]
:OUTPUT ACCEPT [4970272:2145916938]
-A INPUT -i lo -j ACCEPT
-A INPUT -i eth0 -j ACCEPT
-A INPUT -i eth1 -p icmp -j ACCEPT
-A INPUT -i eth1 -p udp -m udp --dport 2304 -j ACCEPT
-A INPUT -i eth1 -p tcp -m tcp --dport 21:23 -j ACCEPT
-A INPUT -i eth1 -p tcp -m tcp --dport 80 -j ACCEPT
-A INPUT -i eth1 -p tcp -m tcp --dport 8888 -j ACCEPT
-A INPUT -i eth1 -m state --state RELATED,ESTABLISHED -j ACCEPT
-A FORWARD -d 192.168.88.88 -j ACCEPT
-A FORWARD -s 192.168.88.0/255.255.255.0 -j ACCEPT
-A FORWARD -i eth1 -m state --state RELATED,ESTABLISHED -j ACCEPT
COMMIT
# Completed on Thu May 24 04:44:31 2007
```

例如，通过命令“iptables-save -t mangle”，显示当前 mangle 表的设置，如下所示。

```
# iptables-save -t mangle //显示 mangle 表的设置
# Generated by iptables-save v1.3.6 on Thu May 24 04:46:08 2007
*mangle
:PREROUTING ACCEPT [851347:495665608]
:INPUT ACCEPT [68335:18578218]
:FORWARD ACCEPT [624216:459405852]
:OUTPUT ACCEPT [25298:16048556]
:POSTROUTING ACCEPT [649512:475454256]
COMMIT
# Completed on Thu May 24 04:46:08 2007
#
```

例如，通过命令“iptables-save -t nat”，显示当前 nat 表的设置，如下所示。


```
# iptables-save -t nat //显示 nat 表的设置
# Generated by iptables-save v1.3.6 on Thu May 24 04:47:19 2007
*nat
:PREROUTING ACCEPT [71751530:7019235259]
:POSTROUTING ACCEPT [32954:1853338]
:OUTPUT ACCEPT [32164:1791964]
-A PREROUTING -d 202.112.146.70 -p tcp -m tcp --dport 3389 -j DNAT --to-destination 192.168.88.88:3389
-A POSTROUTING -s 192.168.88.0/255.255.255.0 -o eth1 -j MASQUERADE
-A POSTROUTING -s 192.168.88.88 -j SNAT --to-source 202.112.146.70
COMMIT
# Completed on Thu May 24 04:47:19 2007
```

8.5.15 iwconfig 命令：配置无线网络设备

iwconfig 为系统配置无线网络设备或显示无线网络设备的状态信息。iwconfig 命令类似于 ifconfig 命令，但是 iwconfig 配置的对象是无线网卡，如设置无线通信频段。该命令的语法格式如下所示。

```
iwconfig
[essid][ nwid][ freq][ channel][ sens][ mode][ ap][ nick[name]][ rate/bit[rate]][ rts[_threshold]] [ retry]
[ frag[mentation_threshold]][ key/enc[ryption] [power][ txpower]
[ commit][ --help][ --version]
```

其中各选项的主要说明如表 8-18 所示。

表 8-18 iwconfig 命令主要选项说明

选 项	说 明
essid	设置 ESSID（网络名）
nwid	设置网络 ID（域名 ID）
freq	设置无线网络通信频率
channel	设置无线网络通信的信道
sens	设置无线网络设备的感知阈值
mode	设置无线网络设备的通信模式
ap	强迫无线网卡向给定地址的接入点注册
nick[name]	为网卡设定别名
rate/bit[rate]	设定无线网卡的传输速率。
rts[_threshold]	在传输数据包之前，RTS/CTS 增加一次握手，确保信道是正常的
frag[mentation_threshold]	设置数据包分段大小
key/enc[ryption]	加密模式
power	无线网卡的电源设置
txpower	对于支持多传输功率的网卡，设定功率大小
retry	定义 MAC 重发机制
commit	强制命令立即生效
--help	显示帮助信息
--version	显示版本信息



例如，显示当前无线网络设备的状态信息，命令如下所示。

```
#iwconfig //显示无线网络信息
lo      no wireless extensions.

eth0    no wireless extensions.
```

ESSID 用来标识不同的网络通信区域，只有 ESSID 相同的网卡才能直接通信，设置无线网卡的 ESSID 为 “mywirelessnetwork”，命令如下所示。

```
#iwconfig eth0 essid "mywirelessnetwork" // 设置无线网卡的 ESSID 为 "mywirelessnetwork"
```

例如，设置网络 ID 为 “AB34”，命令如下所示。

```
#iwconfig eth0 nwid AB34 //设置网络 ID
```

只有在同一频段或信道的无线网络设备才能直接通信。例如，设置通信频段为 2.422G，信道为 3，命令如下所示。

```
#iwconfig eth0 freq 2.422G //设置 eth0 的通信频段为 2.422G
#iwconfig eth0 channel 3 //设置 eth0 的信道为 3
```

例如，设置无线网卡的通信模式为 Ad-hoc（该类网络仅由通信节点组成，没有访问点），命令如下所示。

```
#iwconfig eth0 mode Ad-Hoc //设置通信方式为 Ad-hoc
```

例如，设置无线网卡的通信模式 managed（可管理方式，即通信节点通过访问点访问网络），命令如下所示。

```
#iwconfig eth0 mode Managed //设置通信方式为 managed
```

例如，强制无线网卡注册到访问点 “00:60:1D:01:23:45”，命令如下所示。

```
#iwconfig eth0 ap 00:60:1D:01:23:45 //强制注册到访问点
```

例如，设置无线网卡的别名为 “my Linux node”，命令如下所示。

```
#iwconfig eth0 nickname "my Linux Node" //设置别名
```

例如，设置无线网卡的速率为 5.5M，命令如下所示。

```
# iwconfig eth0 rate 5.5M //设置速率为 5.5Mbit/s
```

例如，设置无线网卡的速率为 2.5M，命令如下所示。

```
# iwconfig eth0 rate 2.5M //设置速率为 2.5Mbit/s
```

例如，设置 MAC 重发的时间间隔为 16 秒，命令如下所示。

```
iwconfig eth0 retry 16
```

8.5.16 mesg 命令：控制终端的写入操作

mesg 命令可以控制其他用户能否使用 write 命令将信息发送到当前用户使用的终端上。该命令的语法格式如下所示。

```
mesg [n][y]
```

其中各选项的主要说明如表 8-19 所示。

表 8-19 mesg 命令主要选项说明

选 项	说 明
y	允许其他用户发送信息到当前终端
n	禁止其他用户发送信息到当前终端

例如，允许其他用户（如终端 pts/3）发送信息到当前终端（pts/4），命令如下所示。


```
# mesg y //设置为允许
如果此时在另一个终端 pts/3 上使用 write 命令向 pts/4 发送信息，命令如下所示。
$ write root pts/4 //在另一个终端上写入
hello
hello //用 Ctrl+D 结束
```

此时在 pts/3 终端将收到如下信息。

```
Message from teacher@localhost on pts/3 at 09:57 //在被写入终端上的输出
hello
helloEOF
```

例如，禁止其他用户发送信息到当前终端（pts/4），命令如下所示。

```
# mesg n //设置为禁止
如果此时在另一个终端 pts/3 上写入 write 命令向 pts/4 发送信息，将得到如下信息。
$ write root pts/4
write: root has messages disabled on pts/4 // write 写入被禁止
```

8.5.17 mingetty 命令：精简版的agetty

mingetty 命令是精简版的 agetty，是适用于虚拟终端上的登录程序。不同于 agetty 命令，mingetty 不适合串口。该命令的语法格式如下所示。

```
mingetty [--noclear] [--long-hostname]
```

其中各选项的主要说明如表 8-20 所示。

表 8-20 mingetty 命令主要选项说明

选 项	说 明
--noclear	在没有提示输入登录名之前，不清除屏幕
--long-hostname	完整显示通过 gethostname 函数得到的全部主机名，默认只显示到第一个“.”处

mingetty 命令可识别的转义字符如表 8-21 所示。

表 8-21 mingetty 命令可识别的转义字符

转 义 字 符	说 明
\d	插入当前日期（本地时间）
\l	插入一行
\m	插入主机的体系结构名称，与命令“uname -m”相同
\n	插入主机的网络节点主机名，与命令“uname -n ”相同
\o	插入域名
\r	插入操作系统版本信息，与命令“uname -r ”相同
\t	插入当前时间（本地时间）
\s	插入操作系统名称
\u	插入登录的用户数目。仅插入数字，无单位
\v	插入操作系统版本信息，与命令“uname -v”相同

8.5.18 minicom 命令：串口通信程序

minicom 命令是一个开源、友好的串行通信程序。minicom 与 TELIX 非常相似，可以在



绝大多数 UNIX 环境下使用，其特点包括：具有播号目录（支持自动重播）、支持 UUCP 风格的锁文件、具有脚本语言解释器等。该命令的语法格式如下所示。

```
minicom [-s, --setup][ -o, --noinit][ -m, --metakey][ -M, --metakey8][ -z,
--statline][ -l, --ansi][ -w, --wrap]
[ -a, --attrib=on/off][ -t, --term=TERM][ -c, --color=on/of][ -S, --script=SCRIPT][ -d,
--dial=ENTRY]
[ -p, --ptty=TTY][ -C, --capturefile=FILE][ -8, --8bit][ configuration]
```

其中各选项的主要说明如表 8-22 所示。

表 8-22 minicom 命令主要选项说明

选 项	说 明
-s, --setup	不进行初始化，直接进入配置菜单
-o, --noinit	程序运行时跳过初始化代码
-m, --metakey	设置 Alt 或 Meta 键作为特殊命令键
-M, --metakey8	与-m 类似，但是 Meta 键设置为字符的高 8 位
-z, --statline	使用终端状态行，仅适于终端
-l, --ansi	假定屏幕使用 IBM-PC 字符集
-w, --wrap	自动换行
-a, --attrib=on/off	使用反转或高亮属性
-t, --term=TERM	设置终端机环境变量
-c, --color=on/of	ANSI 风格颜色
-S, --script=SCRIPT	在启动时运行脚本
-d, --dial=ENTRY	在启动时从指定的播号目录中读取号码，进行播号
-p, --ptty=TTY	连接到伪终端
-C, --capturefile=FILE	启动时打开指定的捕获文件
-8, --8bit	8 位清除模式（适用于日本）
configuration	指定配置文件

8.5.19 mkfifo 命令：创建管道

mkfifo 命令用于根据指定的名字创建 FIFO 管道，其语法格式如下所示。

```
mkfifo [NAME][ -m, --mode=MODE][ --help][ --version]
```

其中各选项的主要说明如表 8-23 所示。

表 8-23 mkfifo 命令主要选项说明

选 项	说 明
NAME	指定管道名称
-m, --mode=MODE	设置创建的管道的权限模式
--help	打印帮助信息
--version	打印版本信息

例如，创建一个具有只读权限的管道 myfifo，命令如下所示。

```
# mkfifo -m 444 myfifo
# ll myfifo
pr--r--r-- 1 root root 0 8 10 07:50 myfifo
```

8.5.20 mtr 命令：网络诊断工具

mtr 命令是集 traceroute 命令和 ping 命令的功能于一体的网络诊断程序。该命令的语法格式如下所法。

```
mtr [-i SECONDS, --interval SECONDS] [-g, --gtk] [-n, --no-dns] [-p BYTES] [-c COUNT] [-r, --report]
[-v, --version] [-h, --help] [HOSTNAME]
```

其中各选项的主要说明如表 8-24 所示。

表 8-24 mtr 命令主要选项说明

选 项	说 明
HOSTNAME	指定要诊断的主机
-h, --help	帮助信息
-v, --version	版本信息
-r, --report	报告模式，以-c 指定的循环数进行统计
-c COUNT	指定-r 模式的循环次数
-p BYTES	指定探测包的大小
-n, --no-dns	强迫显示数字格式的 IP 地址
-g, --gtk	指定使用基于 X11 窗口的 GTK+接口
-i SECONDS, --interval SECONDS	指定 ICMP 回应请求的时间间隔

例如，诊断到主机 192.168.20.147 之间的网络状况，命令如下所示。

```
# mtr 192.168.20.147
Matt's traceroute [v0.52]
localhost.localdomain Fri Aug 29 01:26:10 2008
Keys: D - Display mode R - Restart statistics Q - Quit
Last 50 pings
Scale: .:979 ms 1:949 ms 2:919 ms 3:849 ms a:669 ms b:499 ms c:199 ms
```

8.5.21 nc 命令：TCP/UDP 连接与侦听

nc 命令是 TCP/UDP 的实用工具，可以创建 TCP 链接、发送 UDP 数据包、侦听 TCP 或 UDP 套接口、进行端口扫描、追踪源路由等。nc 命令通常应用于以下几方面。

- 简单 TCP 代理。
- 基于 Shell 脚本的 HTTP 客户端和服务端。
- 网络守护进程检测。
- 基于安全路由的连接检测。

nc 命令的语法格式如下所示。



```
nc [-e command][ -g intermediate-host][ -l][ -n][ -u][ -v][ -z][ -h][ -i seconds][ -o
filename][ -p port][ -w timeout]
[ -s hostname/ip-address]
```

其中各选项的主要说明如表 8-25 所示。

表 8-25 nc 命令主要选项说明

选 项	说 明
-e command	执行指定的命令
-g intermediate-host	在松散源路由路径中指定一跳
-l	侦听模式
-n	使用 IP 地址，不进行 DNS 查询
-u	使用 UDP，而不是 TCP
-v	运行时显示详细的处理信息
-z	扫描侦听进程。默认不显示连接拒绝信息，若希望显示，需使用选项“-v”两次
-h	帮助信息
-i seconds	延迟时间
-o filename	指定十六进制日志文件
-p port	指定源端口号
-s hostname/ip-address	指定源地址
-w timeout	设置超时时间

例如，通过命令 nc 对主机 192.168.20.122 的端口 1~40 进行 TCP 扫描，设定超时时间为 2 秒，且显示详细过程信息，如下所示。

```
# nc -v -v -z -w2 192.168.20.122 1-40 //TCP 端口扫描
(UNKNOWN) [192.168.20.122] 40 (?) : Connection refused
(UNKNOWN) [192.168.20.122] 39 (rlp) : Connection refused
(UNKNOWN) [192.168.20.122] 38 (?) : Connection refused
(UNKNOWN) [192.168.20.122] 37 (time) : Connection refused
(UNKNOWN) [192.168.20.122] 36 (?) : Connection refused
(UNKNOWN) [192.168.20.122] 35 (?) : Connection refused
(UNKNOWN) [192.168.20.122] 34 (?) : Connection refused //对 192.168.20.122 的端口 34
的连接被拒绝
(UNKNOWN) [192.168.20.122] 33 (?) : Connection refused
(UNKNOWN) [192.168.20.122] 32 (?) : Connection refused
(UNKNOWN) [192.168.20.122] 31 (?) : Connection refused
(UNKNOWN) [192.168.20.122] 30 (?) : Connection refused
(UNKNOWN) [192.168.20.122] 29 (?) : Connection refused
(UNKNOWN) [192.168.20.122] 28 (?) : Connection refused
(UNKNOWN) [192.168.20.122] 27 (?) : Connection refused
(UNKNOWN) [192.168.20.122] 26 (?) : Connection refused
(UNKNOWN) [192.168.20.122] 25 (smtp) : Connection refused
(UNKNOWN) [192.168.20.122] 24 (?) : Connection refused
(UNKNOWN) [192.168.20.122] 23 (telnet) open //192.168.20.122 的端口 23 打开
(UNKNOWN) [192.168.20.122] 22 (ssh) open //192.168.20.122 的端口 22 打开
(UNKNOWN) [192.168.20.122] 21 (ftp) : Connection refused
```



```
(UNKNOWN) [192.168.20.122] 20 (ftp-data) : Connection refused
(UNKNOWN) [192.168.20.122] 19 (chargen) : Connection refused
(UNKNOWN) [192.168.20.122] 18 (msp) : Connection refused
(UNKNOWN) [192.168.20.122] 17 (gotd) : Connection refused
(UNKNOWN) [192.168.20.122] 16 (?) : Connection refused
(UNKNOWN) [192.168.20.122] 15 (netstat) : Connection refused
(UNKNOWN) [192.168.20.122] 14 (?) : Connection refused
(UNKNOWN) [192.168.20.122] 13 (daytime) : Connection refused
(UNKNOWN) [192.168.20.122] 12 (?) : Connection refused
(UNKNOWN) [192.168.20.122] 11 (sysstat) : Connection refused
(UNKNOWN) [192.168.20.122] 10 (?) : Connection refused
(UNKNOWN) [192.168.20.122] 9 (discard) : Connection refused//对 192.168.20.122 的端口 9
的连接被拒绝
(UNKNOWN) [192.168.20.122] 8 (?) : Connection refused
(UNKNOWN) [192.168.20.122] 7 (echo) : Connection refused
(UNKNOWN) [192.168.20.122] 6 (?) : Connection refused
(UNKNOWN) [192.168.20.122] 5 (rje) : Connection refused
(UNKNOWN) [192.168.20.122] 4 (?) : Connection refused
(UNKNOWN) [192.168.20.122] 3 (?) : Connection refused
(UNKNOWN) [192.168.20.122] 2 (?) : Connection refused
(UNKNOWN) [192.168.20.122] 1 (tcpmux) : Connection refused//对 192.168.20.122 的端口 1
的连接被拒绝
```

可以看到,有些端口是打开的(如端口 22、23),有些则不可以连接。

例如,通过命令 `nc`,对主机 192.168.20.122 的端口 1~10 进行 UDP 扫描,设定超时时间为 2 秒,且显示详细过程信息,如下所示。

```
# nc -u -vv -z -w2 192.168.20.122 1-25 //使用 UDP 进行端口扫描
(UNKNOWN) [192.168.20.148] 10 (?) : Connection refused
(UNKNOWN) [192.168.20.148] 9 (discard) : Connection refused
(UNKNOWN) [192.168.20.148] 8 (?) : Connection refused
(UNKNOWN) [192.168.20.148] 7 (echo) : Connection refused
(UNKNOWN) [192.168.20.148] 6 (?) : Connection refused
(UNKNOWN) [192.168.20.148] 5 (rje) : Connection refused
(UNKNOWN) [192.168.20.148] 4 (?) : Connection refused
(UNKNOWN) [192.168.20.148] 3 (?) : Connection refused
(UNKNOWN) [192.168.20.148] 2 (?) : Connection refused
(UNKNOWN) [192.168.20.148] 1 (tcpmux) : Connection refused
```

例如,通过命令 `nc`,连接到远端主机 192.168.20.148 的 80 端口号上。用户输入任意字符并按回车键后,可以看到 HTML 代码,命令执行过程如下所示。

```
# nc -vv 192.168.20.148 80 //连接远端主机
(UNKNOWN) [192.168.20.148] 80 (http) open
aa //用户输入
<a href="/phpMyAdmin/index.php">Start phpMyAdmin</a><hr>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"DTD/xhtml1-transitional.dtd">
<html><head>
<style type="text/css"><!-- //页面风格设置
body {background-color: #ffffff; color: #000000;}
body, td, th, h1, h2 {font-family: sans-serif;}
pre {margin: 0px; font-family: monospace;}
a:link {color: #000099; text-decoration: none; background-color: #ffffff;}
a:hover {text-decoration: underline;}
table {border-collapse: collapse;}
```



```
.center {text-align: center;}
.center table { margin-left: auto; margin-right: auto; text-align: left;}
.center th { text-align: center !important; }
td, th { border: 1px solid #000000; font-size: 75%; vertical-align: baseline;}
h1 {font-size: 150%;}
h2 {font-size: 125%;}
.p {text-align: left;}
.e {background-color: #ccccff; font-weight: bold; color: #000000;}
.h {background-color: #9999cc; font-weight: bold; color: #000000;}
.v {background-color: #cccccc; color: #000000;}
.vr {background-color: #cccccc; text-align: right; color: #000000;}
img {float: right; border: 0px;}
hr {width: 600px; background-color: #cccccc; border: 0px; height: 1px; color: #000000;}
/--></style>
<title>phpinfo()</title></head>                                     //页面标题是 phpinfo( )
... ..
```

8.5.22 netconfig 命令：设置各项网络功能

netconfig 命令用来设置网络环境，既可以通过命令行选项完成网络配置，也可以提供图形化界面。通过 netconfig 命令可以设置本地主机名称、域名、IP 地址、网络掩码、网关及域名服务器。该命令的语法格式如下所示。

```
netconfig
[--bootproto=(dhcp|bootp|none)][ --gateway=STRING][ --ip=STRING][ --nameserver=STRING]
[      --netmask=STRING][      --hostname=STRING][      --domain=STRING][      -d,
--device=STRING][ --nodns]
[ -?, --help][ --usage][ ][ ][ ][ ][ ][ ][ ][ ]
```

其中各选项的主要说明如表 8-26 所示。直接在命令行下执行 netconfig 命令，可以进入图形界面，比较快捷地完成网络配置。

表 8-26 netconfig 命令主要选项说明

选 项	说 明
--bootproto=(dhcp bootp none)	使用的启动协议，可选项包括 dhcp、bootp 和 none
--gateway=STRING	设置网关
--ip=STRING	设置 IP 地址
--nameserver=STRING	设置域名服务器
--netmask=STRING	设置子网掩码
--hostname=STRING	设置主机名
--domain=STRING	设置域名
-d, --device=STRING	指定网络设备
--nodns	设置不进行 DNS 查询
-?, --help	显示帮助信息
--usage	显示用法摘要

例如，在命令行下直接执行 netconfig 命令，进行网络设置，如下所示。


```
# netconfig //网络配置
```

该命令执行之后出现图 8-10 所示的【Network configuration】界面，单击【Yes】按钮后进入【Configuration TCP/IP】界面，如图 8-11 所示。

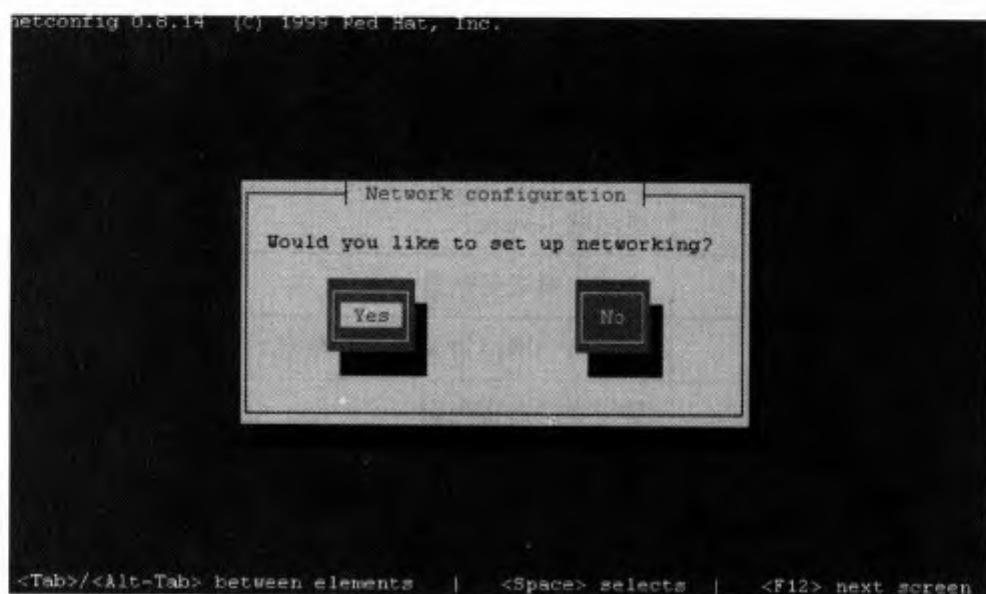


图 8-10 【Network configuration】界面

在图 8-11 所示界面中，可以根据提示信息设置网络的 IP 地址、子网掩码、默认网关、主 DNS 服务器地址。如果使用 DHCP 进行动态地址配置，需要选择【Use dynamic IP configuration】。

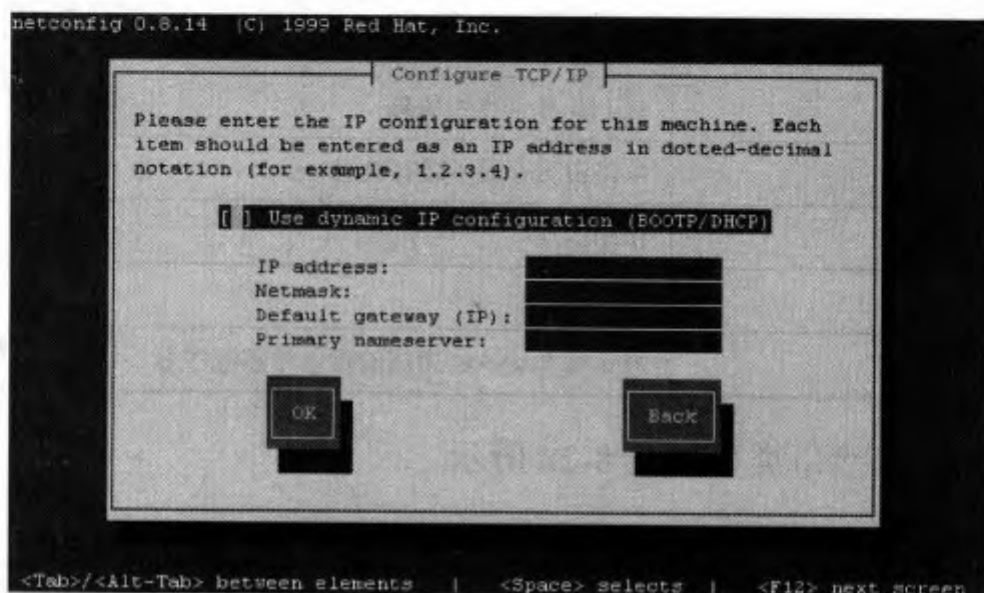


图 8-11 【Configuration TCP/IP】界面

完成网络配置之后，单击【OK】按钮可以保存网络配置。如果希望配置立即生效，需要重启系统或重新激活网卡。

8.5.23 netstat 命令：显示网络状态

netstat 命令用于显示网络连接状态、路由表信息、接口统计信息等网络基本情况，其语法格式如下所示。

```
netstat [--route , -r][ --interface=iface , -i][ --groups , -g][ --statistics , -s][ --masquerade , -M] [ -p, --program]
```



```
[ --verbose  
-n] [ --numeric-hosts [ --numeric-port [ -e, --extend  
[ -c, --continuous [ -l, --listening [ -a, --all [ -F ] [ -C ] [ -h, --help ] [ -V, --version ]
```

其中各选项的主要说明如表 8-27 所示。

表 8-27 netstat 命令主要选项说明

选项	说明
--route, -r	显示路由表
--interface=iface, -i	显示网卡列表
--groups, -g	显示组播成员信息
--statistics, -s	显示每一网络协议的统计信息
--masquerade, -M	显示隐藏链接列表
--verbose, -v	运行时显示详细的处理信息
--numeric, -n	不解析主机名，直接显示 IP 地址
--numeric-hosts	显示主机 IP 地址
--numeric-port	显示端口名
--numeric-users	显示用户 ID
-e, --extend	显示更多信息
-c, --continuous	连续刷新列表
-l, --listening	显示侦听的服务套接口
-a, --all	显示所有的套接口（包括侦听与非侦听）
-F	从 FIB 显示路由信息
-C	从路由缓存显示缓存信息
-h, --help	帮助信息
-V, --version	版本信息
-p, --program	显示每个 socket 所属的进程号和程序名

netstat 命令支持的套接口类型如表 8-28 所示。

表 8-28 netstat 命令支持的套接口类型

类型表示	说明
-t	TCP 类型
-u	UDP 类型
--raw	RAW 类型
--UNIX	UNIX 类型
--ax25	AX25 类型
--ipx	ipx 类型
--netrom	netrom 类型

netstat 命令支持的地址族如表 8-29 所示。

表 8-29 netstat 命令支持的地址族

地 址 类 型	说 明
inet	DARPA 网络
inet6	IPv6
ax25	AMPR AX.25
netrom	AMPR NET/ROM
ipx	Novell IPX
ddp	Appletalk DDP
x25	CCITT X.25

例如，通过使用“-r”选项，显示本地主机的路由表，命令如下所示。

```
# netstat -r //显示路由表
Kernel IP routing table
Destination      Gateway          Genmask          Flags   MSS Window  irtt Iface
192.168.20.0      *                255.255.255.0    U        0 0        0 eth0
169.253.0.0       *                255.255.0.0      U        0 0        0 eth0
default           192.168.20.2    0.0.0.0          UG       0 0        0 eth0
```

例如，显示网络的详细状况，包括是否处于侦听状态和连接状态，命令如下所示。

```
# netstat -a //显示详细的网络状况
Active Internet connections (servers and established) //活动的网络连接
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp      0      0 *:32768                 *:*                     LISTEN // 侦
听态
tcp      0      0 localhost.localdo:32769 *:*                     LISTEN
tcp      0      0 localhost.localdoma:783 *:*                     LISTEN
tcp      0      0 *:sunrpc                 *:*                     LISTEN
tcp      0      0 *:x11                     *:*                     LISTEN
tcp      0      0 *:ssh                     *:*                     LISTEN
tcp      0      0 localhost.localdoma:ipp *:*                     LISTEN
tcp      0      0 *:telnet                 *:*                     LISTEN
tcp      0      0 localhost.localdom:smtp *:*                     LISTEN
tcp      0      2 192.168.20.122:telnet 192.168.20.148:1285    ESTABLISHED // 已
建立连接
Active UNIX domain sockets (servers and established) //活动的 UNIX 域名 socket
Proto RefCnt Flags   Type       State      I-Node Path
UNIX 2 [ ACC ] STREAM LISTENING 3150 /tmp/orbit-root/linc-a29-0-5a618b4eb7e8f
UNIX 2 [ ACC ] STREAM LISTENING 3191 /tmp/orbit-root/linc-a2f-0-400a11a5bd79
UNIX 2 [ ACC ] STREAM LISTENING 3208 /tmp/orbit-root/linc-a31-0-7e0502e42de74
UNIX 2 [ ACC ] STREAM LISTENING 3211 /tmp/orbit-root/linc-a2d-0-400a11a52f3f2
UNIX 2 [ ACC ] STREAM LISTENING 3285 /tmp/orbit-root/linc-a33-0-7e0502e4ebe55
UNIX 2 [ ACC ] STREAM LISTENING 3358 /tmp/orbit-root/linc-a42-0-81acb56dc8a6
UNIX 2 [ ACC ] STREAM LISTENING 3374 /tmp/orbit-root/linc-a38-0-81acb5112350
... ..
```

例如，显示本地主机 UDP 的连接状况，命令如下所示。

```
# netstat -nu //显示 UDP 联机状况
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
udp      0      0 127.0.0.1:32768         127.0.0.1:32768        ESTABLISHED
```

例如，显示本地主机 UDP 端口号的使用状况，命令如下所示。



```
# netstat -apu //显示 UDP 端口号使用情况
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address      Foreign Address    State      PID/Program name
udp      0      0 *:2304             *:                 26541/poweron
udp      0      0 localhost:32768     localhost:32768    ESTABLISHED 3723/postmaster
udp      0      0 *:32769            *:                 3891/rpc.statd
udp      0      0 *:32770            *:                 26541/poweron
udp      0      0 *:675              *:                 3891/rpc.statd
udp      0      0 *:sunrpc            *:                 3201/portmap
```

例如，显示网卡相关列表，命令如下所示。

```
# netstat -i //显示网卡列表
Kernel Interface table
Iface      MTU Met  RX-OK RX-ERR RX-DRP RX-OVR    TX-OK TX-ERR TX-DRP TX-OVR Flg
eth0       1500 0    114420 0      0      0      1538 0      0      0 BMRU
lo         16436 0     1752 0      0      0      1752 0      0      0 LRU
```

例如，显示组播成员信息，命令如下所示。

```
# netstat -g //显示组播关系
IPv6/IPv4 Group Memberships
Interface      RefCnt Group
-----
lo              1      ALL-SYSTEMS.MCAST.NET
eth0            1      224.0.0.251
eth0            1      ALL-SYSTEMS.MCAST.NET
lo              1      ff02::1
eth0            2      ff02::1:ff32:451c
eth0            1      ff02::1
```

例如，通过命令 netstat，显示网络的统计信息，包括 IP、TCP、UDP、ICMP、TCPEXT 等数据包的接收情况，命令如下所示。

```
# netstat -s //显示网络统计信息
Ip: //IP 包的统计情况
  75963 total packets received
    0 forwarded
    0 incoming packets discarded
  75713 incoming packets delivered
  75352 requests sent out
    77 outgoing packets dropped

Icmp: //ICMP 包的统计情况
  153 ICMP messages received
    0 input ICMP message failed.
  ICMP input histogram:
    destination unreachable: 153
  100 ICMP messages sent
    0 ICMP messages failed
  ICMP output histogram:
    destination unreachable: 100

Tcp: //Tcp 包的统计情况
  1999 active connections openings
  1887 passive connection openings
    0 failed connection attempts
    0 connection resets received
    1 connections established
  75642 segments received
  75089 segments send out
```



```
26 segments retransmitted
0 bad segments received.
96 resets sent
Udp:                                     //Udp 包的统计情况
10 packets received
100 packets to unknown port received.
0 packet receive errors
163 packets sent
... ..
```

例如，通过命令 netstat，查看当前系统中所有处于侦听状态的 socket，命令如下所示。

```
# netstat -l
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp      0      0 localhost.localdomain:2208 *:.*                    LISTEN
tcp      0      0 *:sift-uft              *:.*                    LISTEN
tcp      0      0 *:netbios-ssn           *:.*                    LISTEN
tcp      0      0 *:sunrpc                 *:.*                    LISTEN
tcp      0      0 *:ftp                    *:.*                    LISTEN
... ..
```

8.5.24 nslookup 命令：查询 DNS

nslookup 命令是一个查找 Internet 域名服务器的程序，主要用于测试 DNS 服务器是否工作正常。除此之外，还可以对域名和 IP 地址进行查询。nslookup 有两种模式：交互模式和非交互模式。在非交互模式下，该命令的语法格式如下所示。

```
nslookup [IP][HOSTNAME]
```

其中选项“IP”及“HOSTNAME”表示等待解析的 IP 地址或主机名。在命令行下，直接执行 nslookup 命令将进入交互模式，交互模式下的常用内部命令如表 8-30 所示。

表 8-30 nslookup 的主要内部命令说明

内部命令	说明
exit	退出
host	查找主机信息
server<服务器>	指定 DNS 服务器
set<关键词>	改变查询信息的状态

例如，在交互模式下查询北京理工大学网站（www.bit.edu.cn）的 IP 地址，使用 nslookup 命令如下。

```
#nslookup
// “>” 是 nslookup 命令环境的提示符，输入待查询的域名 www.bit.edu.cn
> www.bit.edu.cn
Server: dns.bj.unicomcdma.com
Address: 220.192.0.130
//以上为所使用的 DNS 服务器
Non-authoritative answer:
Name: www.bit.edu.cn
Address: 202.204.80.38
//输入待查询的 IP 地址
> 202.204.80.38
```



```
Server: dns.bj.unicomcdma.com
Address: 220.192.0.130
//以上为所使用的 DNS 服务器
Name: www.bit.edu.cn.80.204.202.in-addr.arpa
Address: 202.204.80.38
//使用“exit”命令退出 nslookup 命令环境
>exit
#
```

也可以不进入 nslookup 命令交互模式，直接使用命令查询 IP 地址或域名，如下所示。

```
#nslookup www.bit.edu.cn
Server: dns.bj.unicomcdma.com
Address: 220.192.0.130

Non-authoritative answer:
Name: www.bit.edu.cn
Address: 202.204.80.38
```

查询 IP 地址为 202.204.80.38 的服务器的域名，命令行如下。

```
#nslookup 202.204.80.38
Server: dns.bj.unicomcdma.com
Address: 220.192.0.130

Name: www.bit.edu.cn.80.204.202.in-addr.arpa
Address: 202.204.80.38
```

8.5.25 route 命令：显示并设置路由

route 命令用于显示和动态修改系统当前的路由表，其原始的用途是为特定主机或网络设置静态路由。可以通过 route 命令的 add 或 del 选项，改变路由表设置。route 命令的语法格式如下所示。

```
route [-v] [-n] [-e] [-F] [-C] [del] [add] [target] [-net] [-host] [--help] [--version]
```

其中各选项的主要说明如表 8-31 所示。

表 8-31 route 命令主要选项说明

选 项	说 明
-v	运行时，显示详细的处理信息
-n	不解析主机名称
-e	使用 netstat 格式显示路由表
-F	使用内核 FIB 路由表
-C	使用内核路由缓存
del	删除路由
add	添加路由
target	目标网络或主机，可以使用主机/网络名或 IP 地址
-net	目标网络
-host	目标主机
--help	帮助信息
--version	版本信息

route 命令支持的地址族如表 8-32 所示。

表 8-32 route 命令支持的地址族

地 址 类 型	说 明
inet	DARPA 互联网
inet6	IPv6
ax25	AMPR AX.25
netrom	AMPR NET/ROM
ipx	Novell IPX
ddp	Appletalk DDP
X25	CCITT X.25

例如，不带任何选项直接使用 route 命令，可以显示系统当前的路由信息，如下所示。

```
# route
Kernel IP routing table
Destination      Gateway          Genmask         Flags   Metric  Ref    Use  Iface
192.168.255.0    *                255.255.255.0   U        0        0      0   eth0
default          192.168.255.254 0.0.0.0         UG        0        0      0   eth0
```

例如，使用“route add”命令添加 172.16.0.0 网络的路由信息，如下所示。

```
# route add -net 172.16.0.0 netmask 255.255.0.0
# route
Kernel IP routing table
Destination      Gateway          Genmask         Flags   Metric  Ref    Use  Iface
192.168.255.0    *                255.255.255.0   U        0        0      0   eth0
172.16.0.0       *                255.255.0.0     U        0        0      0   eth0
default          192.168.255.254 0.0.0.0         UG        0        0      0   eth0
//已经成功添加 172.16.0.0 网络的路由信息
```

使用“route add default gw 网关 IP 地址 dev 接口”和“route del default gw 网关 IP 地址”命令格式，可以添加和删除系统当前路由表中的默认网关信息。

例如，在当前路由表中添加默认网关 192.168.255.254，命令行如下。

```
# route
Kernel IP routing table
Destination      Gateway          Genmask         Flags   Metric  Ref    Use  Iface
192.168.255.0    *                255.255.255.0   U        0        0      0   eth0
172.16.0.0       *                255.255.0.0     U        0        0      0   eth0
#route add default gw 192.168.255.254 dev eth0 //添加默认网关
# route
Kernel IP routing table
Destination      Gateway          Genmask         Flags   Metric  Ref    Use  Iface
192.168.255.0    *                255.255.255.0   U        0        0      0   eth0
172.16.0.0       *                255.255.0.0     U        0        0      0   eth0
default          192.168.255.254 0.0.0.0         UG        0        0      0   eth0
```

例如，使用“route del default gw 网关 IP 地址”命令格式，删除已存在的默认网关，执行命令如下所示。

```
#route del default gw 192.168.255.254 dev eth0
# route
```



```
Kernel IP routing table
Destination      Gateway          Genmask          Flags   Metric  Ref    Use    Iface
192.168.255.0    *                255.255.255.0    U        0        0        0    eth0
172.16.0.0       *                255.255.0.0      U        0        0        0    eth0
//已经删除默认的网关
```

例如，在网络 192.168.200.0 中有一台主机，该主机有两块网卡，IP 地址分别为 192.168.200.10 和 172.16.0.111，分别连接网络 192.168.200.0 和 172.16.0.0。要在该网络的主机 192.168.200.56 中增加到达 172.16.0.0 的路由，命令如下所示。

```
#route add -net 172.16.0.0 gw 172.16.10.111 netmask 255.255.0.0 metric 1
```

例如，通过命令“route add -net 10.0.0.0 netmask 255.0.0.0 reject”，增加一条屏蔽路由，目的地为 10.x.x.x 的路由将被拒绝，如下所示。

```
# route add -net 10.0.0.0 netmask 255.0.0.0 reject //增加一条屏蔽路由
# route //查看路由
Kernel IP routing table
Destination      Gateway          Genmask          Flags   Metric  Ref    Use    Iface
192.168.20.0     *                255.255.255.0    U        0        0        0    eth0
169.254.0.0     *                255.255.0.0      U        0        0        0    eth0
10.0.0.0         ~                255.0.0.0        !        0        -        0    -
127.0.0.0        *                255.0.0.0        U        0        0        0    lo
224.0.0.0        *                240.0.0.0        U        0        0        0    eth0
default          192.168.20.2     0.0.0.0          UG        0        0        0    eth0
```

8.5.26 ping 命令：检测主机（ipv4）

ping 命令使用 ICMP 协议，可以向指定的主机发送 ICMP 包，并等待回复，以确定指定的主机是否存在。ping 命令主要用于测试网络的连通性，其基本格式如下。

```
ping [-c count][ -s packetsize][ -b][ -i interval][ -q][ -S sndbuf][ -t][ -W timeout][ -T timestamp option] destination
```

其中主要选项的说明如表 8-33 所示。

表 8-33 ping 命令主要选项说明

选 项	说 明
-c count	测试中发出的分组数。如果不指定 count，ping 命令会连续发送测试分组，直到按 Ctrl+c 组合键强行中断该命令
-s packetsize	以字节为单位指定分组报文的大小，缺省为 56 字节
-b	允许 ping 广播地址
-i interval	指定分组发送的间隔时间，只有根用户可以指定小于 0.2 秒的时间间隔
-q	静默模式，只显示最后的统计信息
-S sndbuf	指定 socket 发送缓冲大小
-t	设置 TTL（IP 生存期）
-W timeout	定义等待响应的时间
-T timestamp option	设置指定的 IP 时间戳
destination	指定检测的主机

例如，检测一台主机 Computer1 到另一台主机 Computer2 之间是否连通，可以在主机 Computer1 上执行下面的命令。

```
# ping Computer2
PING Computer2.bit.edu.cn (192.168.200.2) 56(84) bytes of data.
64 bytes from Computer2.bit.edu.cn (192.168.200.2): icmp_seq=1 ttl=64 time=0.068 ms
64 bytes from Computer2.bit.edu.cn (192.168.200.2): icmp_seq=2 ttl=64 time=0.029 ms
64 bytes from Computer2.bit.edu.cn (192.168.200.2): icmp_seq=3 ttl=64 time=0.029 ms
64 bytes from Computer2.bit.edu.cn (192.168.200.2): icmp_seq=4 ttl=64 time=0.031 ms
//按“Ctrl+c”键终止ping命令
--- Computer2.bit.edu.cn ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2999ms
rtt min/avg/max/mdev = 0.029/0.039/0.068/0.017 ms
```

发送过量的测试分组并不能很好地利用网络和系统资源，一般对于一次测试，发送 5 个分组报文已经足够。使用选项“-c”可以指定发送分组的数量，如下所示。

```
# ping -c 5 192.168.255.128
PING 192.168.255.128 (192.168.255.128) 56(84) bytes of data.
64 bytes from 192.168.255.128: icmp_seq=1 ttl=64 time=0.192 ms
64 bytes from 192.168.255.128: icmp_seq=2 ttl=64 time=0.030 ms
64 bytes from 192.168.255.128: icmp_seq=3 ttl=64 time=0.031 ms
64 bytes from 192.168.255.128: icmp_seq=4 ttl=64 time=0.032 ms
64 bytes from 192.168.255.128: icmp_seq=5 ttl=64 time=0.032 ms
--- 192.168.255.128 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 3998ms
rtt min/avg/max/mdev = 0.030/0.063/0.192/0.064 ms
```

使用“-s”选项可以指定发送分组的大小。例如，向 192.168.255.128 发送 6 个发组，每个分组大小为 6553 字节，命令如下所示。

```
# ping -s 6553 -c 5 192.168.255.128
PING 192.168.255.128 (192.168.255.128) 6553(6581) bytes of data.
6561 bytes from 192.168.255.128: icmp_seq=1 ttl=64 time=1.63 ms
6561 bytes from 192.168.255.128: icmp_seq=2 ttl=64 time=0.034 ms
6561 bytes from 192.168.255.128: icmp_seq=3 ttl=64 time=0.034 ms
6561 bytes from 192.168.255.128: icmp_seq=4 ttl=64 time=0.033 ms
6561 bytes from 192.168.255.128: icmp_seq=5 ttl=64 time=0.034 ms
--- 192.168.255.128 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 5003ms
rtt min/avg/max/mdev = 0.033/0.300/1.635/0.597 ms
```

为了防止大数据包攻击，ping 命令不允许发送过大的分组，最大分组不能超过 65507 字节，如下所示。

```
# ping -s 65535 192.168.255.128
Error: packet size 65535 is too large. Maximum is 65507
```

例如，通过命令 ping，检测到主机 www.sina.com.cn 的链路是否存在。用“-i”选项指定周期性发送包的间隔为 3 秒，用“-s”选项指定发送包的大小为 1024 字节，用“-t”选项指定 TTL 值为 25，用“-c”选项指定发送 8 个数据包。执行命令如下所示。

```
# ping -c 8 -i 3 -s 1024 -t 25 www.sina.com.cn //检测主机是否存在
PING jupiter.sina.com.cn (202.205.3.130) 1024(1052) bytes of data.
1032 bytes from 202.205.3.130: icmp_seq=0 ttl=52 time=6.86 ms
1032 bytes from 202.205.3.130: icmp_seq=1 ttl=52 time=5.43 ms
1032 bytes from 202.205.3.130: icmp_seq=2 ttl=52 time=5.79 ms
```



```
1032 bytes from 202.205.3.130: icmp_seq=3 ttl=52 time=9.41 ms
1032 bytes from 202.205.3.130: icmp_seq=4 ttl=52 time=7.30 ms
1032 bytes from 202.205.3.130: icmp_seq=5 ttl=52 time=11.5 ms
1032 bytes from 202.205.3.130: icmp_seq=6 ttl=52 time=5.48 ms
1032 bytes from 202.205.3.130: icmp_seq=7 ttl=52 time=5.86 ms
--- jupiter.sina.com.cn ping statistics ---
8 packets transmitted, 8 received, 0% packet loss, time 23074ms
rtt min/avg/max/mdev = 5.437/7.220/11.598/2.068 ms, pipe 2
```

8.5.27 ping6 命令：检测主机（ipv6）

ping6 命令与 ping 类似，只是 ping6 针对于 IPv6，而 ping 命令针对 IPv4。该命令的语法格式如下所示。

```
ping6 [-n][ -R][ -v][ -r][ -Q tos][ -I interface address][ -c count][ -s
packet size][ -b][ -i interval][ -q] [ -W timeout]
[ -S sndbuf][ -t] [ -T timestamp option][ destination]
```

其中各选项的主要说明如表 8-34 所示。

表 8-34 ping6 命令主要选项说明

选 项	说 明
-n	使用数值形式，不进行域名解析
-R	记录路由
-v	运行时显示详细的处理信息
-r	不通过网关传送包
-Q tos	设置服务质量
-I interface address	指定网卡
-c count	测试中发出的分组数。如果不指定 count，ping 命令会连续发送测试分组，直到按 Ctrl+c 组合键强行中断该命令
-s packet size	以字节为单位指定分组报文的大小，缺省为 56 字节
-b	允许 ping 广播地址
-i interval	指定分组发送的间隔时间，只有根用户可以指定小于 0.2 秒的时间间隔
-q	静默模式，只显示最后的统计信息
-S sndbuf	指定 socket 发送缓冲大小
-t	设置 TTL（IP 生存期）
-W timeout	定义等待响应的时间
-T timestamp option	设置指定的 IP 时间戳
destination	指定检测的主机

例如，通过命令 ping6 向主机 300c:da8:205:130:21c:20ef:fe32:45bc 发送测试包，检测该主机是否可达，如下所示。

```
# ping6 300c:da8:205:130:21c:20ef:fe32:45bc //检测主机是否可达
PING 300c:da8:205:130:21c:20ef:fe32:45bc (300c:da8:205:130:21c:20ef:fe32:45bc) 56
```



```
data bytes
 64 bytes from 300c:da8:205:130:21c:20ef:fe32:45bc: icmp_seq=0 ttl=64 time=6.64 ms
 64 bytes from 300c:da8:205:130:21c:20ef:fe32:45bc: icmp_seq=1 ttl=64 time=0.218 ms
 64 bytes from 300c:da8:205:130:21c:20ef:fe32:45bc: icmp_seq=2 ttl=64 time=0.053 ms
 64 bytes from 300c:da8:205:130:21c:20ef:fe32:45bc: icmp_seq=3 ttl=64 time=0.052 ms
 64 bytes from 300c:da8:205:130:21c:20ef:fe32:45bc: icmp_seq=4 ttl=64 time=0.051 ms
 64 bytes from 300c:da8:205:130:21c:20ef:fe32:45bc: icmp_seq=5 ttl=64 time=0.053 ms
 64 bytes from 300c:da8:205:130:21c:20ef:fe32:45bc: icmp_seq=6 ttl=64 time=0.045 ms
 64 bytes from 300c:da8:205:130:21c:20ef:fe32:45bc: icmp_seq=7 ttl=64 time=0.051 ms
 64 bytes from 300c:da8:205:130:21c:20ef:fe32:45bc: icmp_seq=8 ttl=64 time=0.052 ms
 64 bytes from 300c:da8:205:130:21c:20ef:fe32:45bc: icmp_seq=9 ttl=64 time=0.052 ms
 64 bytes from 300c:da8:205:130:21c:20ef:fe32:45bc: icmp_seq=10 ttl=64 time=0.051 ms
--- 300c:da8:205:130:21c:20ef:fe32:45bc ping statistics ---
11 packets transmitted, 11 received, 0% packet loss, time 10003ms //发送 11 个测试包,收到 11 个, 丢失率为 0
rtt min/avg/max/mdev = 0.045/0.665/6.645/1.891 ms, pipe 2
```

8.5.28 pppd 命令：点对点连接的守护进程

PPP (Point-to-Point Protocol)，即点对点协议，提供了在串行点对点连接中传输数据包的方法。PPP 由三部分组成：串行连接的数据包封装方法、扩展的连接控制协议（LCP）以及用于建立和配置不同网络层协议的网络控制协议族(NCP)。数据包的封闭由内核驱动完成，pppd 命令提供基本的 LCP、认证支持和 IPCP（用于创建和配置 Internet 协议的 NCP）。

pppd 命令可以用来建立主机间的 PPP 链接，其语法格式如下所示。

```
pppd [auth][ call name][ connect script][ disconnect script][ ipx][noauth][ nolog]
[ connect-delay n] [ crtscts]
    [ maxconnect n][ idle n][ logfile filename][ device][ speed][ loc:rem]
[ defaultroute][ modem][ file]
[ connect]
```

其中各选项的主要说明如表 8-35 所示。

表 8-35 pppd 命令主要选项说明

选 项	说 明
auth	在发送或接收数据包前要求认证
call name	从/etc/ppp/peers/name 中读选项信息
connect script	使用脚本建立串行点对点连接
disconnect script	在 pppd 终止连接后执行指定的脚本
ipx	允许 IPXCP 和 IPX 协议
noauth	不要求认证
nolog	不记录 log 信息
connect-delay n	连接脚本运行完毕后等待指定的 n 毫秒
maxconnect n	指定连接使用的时间
idle n	设定 pppd 命令超时设置
logfile filename	附加日志信息到指定的文件
device	指定通信设备



续表

选 项	说 明
speed	指定波特率
loc:rem	设置本地或远端 IP 地址
connect	执行指定的 Shell 命令建立串行连接
crtsets	使用硬件 RTS/CTS 流量控制
defaultroute	添加默认路由
modem	使用调制解调器控制连接
file	从指定文件读取选项设置

例如，通过命令 `pppd` 尝试建立 PPP 连线，这会出现一些乱码，尝试失败后将返回到命令行，如下所示。

```
# pppd //建立 ppp 链接
~?}#!}!!} }4}"&} } } } %}&?It_}'")({}"??~?}#!}!!} }4}"&} } } } %}&?It_}'")({
}"??~?}#!}!!} }4}"&} } } } %}&?It_}'")({}"??~?}#!}!!} }4}"&} } } } %}&?It_}'")
({}"??~?}#!}!!} }4}"&} } } } %}&?It_}'")({}"??~?}#!}!!} }4}"&} } } } %}&?It_}'")
)({}"??~?}#!}!!} }4}"&} } } } %}&?It_}'")({}"??~?}#!}!!} }4}"&} } } } %}&?It_}'")
")({}"??~
... ..
```

8.5.29 pppstats 命令：显示 PPP 统计信息

`pppstats` 命令用来显示 PPP 连线的状态，其语法格式如下所示。

```
pppstats [-a][ -r][ -v][ -z][ -c count][ -w wait]
```

其中各选项的主要说明如表 8-36 所示。

表 8-36 pppstats 命令主要选项说明

选 项	说 明
-a	显示绝对数值
-r	显示额外的统计信息，对压缩率进行统计
-v	显示额处的统计信息，包括 Van Jacobson TCP 压缩算法的性能
-z	显示使用中的包压缩算法的性能
-c count	重复显示的次数
-w wait	每次显示之间暂停指定时间，即设置显示的周期

例如，通过命令 `pppstats`，显示 `ppp` 的连线信息，命令如下所示。

```
#pppstats //显示 ppp 连线信息
```

8.5.30 rdate 命令：通过网络获取时间

`rdate` 命令用于通过 TCP/IP 网络连接到一台 RFC 868 时间服务器，可以获取时间或对系统时间进行设置。该命令的语法格式如下所示。

```
rdate [-s] [-p] [-u] [-l][host]
```

其中各选项的主要说明如表 8-37 所示。

表 8-37 rdate 命令主要选项说明

选 项	说 明
-p	显示从远端主机返回的时间
-s	根据远端主机的时间，设置本地的系统时间
-u	传输协议使用 UDP 协议，而不是 TCP
-l	使用 syslog 显示错误信息
host	指定获取时间的主机

例如，通过命令 rdate，从远端主机 192.168.20.7 获取系统时间，如下所示。

```
# rdate 192.168.20.7 //从 192.168.20.7 获取时间
```

8.5.31 smbd 命令：Samba 服务端

smbd 命令是 Samba 服务器的守护进程，通过使用 SMB（或 CIFS）协议，可以给 Windows 用户提供文件共享和打印服务。

SMB（Server Message Block）通信协议是微软和英特尔在 1987 年制定的协议，主要是作为 Microsoft 网络的通信协议。简单地说，该协议是在局域网中共享目录和打印机的一种协议。SMB 不仅可以工作在 TCP/IP 之上，也可以用在其他网络协议之上，如 IPX、NetBEUI 等。SMB 协议使用了 NetBIOS 的应用程序接口（Application Program Interface, API），通过 SMB 协议，客户端应用程序可以在各种不同的网络环境下访问服务器，并提出服务请求。

SMB 与 LanManager 协议兼容，可以支持 LanManager 客户端（包括 MS-DOS 下的 MSCIENT3.0，Windows 工作组等）。

Samba 的服务器程序实现了 SMB 协议，通过运行 smbd 守护进程，可以把 Linux 系统配置成一台 Samba 服务器，从而可以在安装和使用 SMB 协议的机器之间实现共享文件和打印机。smbd 命令的语法格式如下所示。

```
smbd [-a] [-D] [-i] [-o] [-P] [-h] [-?] [-V] [-d debuglevel] [-s services file] [-O socket options]
```

其中各选项的主要说明如表 8-38 所示。

表 8-38 smbd 命令主要选项说明

选 项	说 明
-a	附加到日志文件
-D	后台执行，默认
-i	交互模式
-o	覆盖写日志文件，不附加到文件尾部
-P	测试
-h	帮助信息
-?	显示用法信息



续表

选 项	说 明
-V	显示版本信息
-d debuglevel	日志/调试文件的主文件名
-s services file	服务文件的文件名
-O socket options	Socket 选项

例如，在后台启动 Samba 服务器，命令如下所示。

```
#smbd -D //启动 Samba 服务端
```

8.5.32 ssh 命令：远程登录程序（ssh 客户端）

ssh 命令是 ssh 的客户端程序，用来登录远程主机，其语法格式如下所示。

```
ssh [-1][-2][-3][-4][-6][-f][-g][-n][-q][-v][-x][-X][-C][ -D port][ -l login_name][ -I smartcard_device][ -i identity_file][ -b bind_address][hostname ][ user@hostname]
```

其中各选项的主要说明如表 8-39 所示。

表 8-39 ssh 命令主要选项说明

选 项	说 明
-1	强制 ssh 命令使用协议 1
-2	强制 ssh 命令使用协议 2
-4	强制使用 IPv4 协议
-6	强制使用 IPv6 协议
-f	在命令被执行前将 ssh 转到后台
-g	允许远端主机连接本地转发端口
-b bind_address	指定接口
-i identity_file	读取 RSA 或 DSA 认证所需的私钥
-I smartcard_device	定义使用的智能卡设备
-p port	指定连接远端主机的端口
-n	重定向标准输入
-q	静默模式不显示处理信息
-v	运行时显示详细的处理信息
-x	关闭 X11 转发
-X	启动 X11 转发
-C	对所有数据进行压缩
-D port	指定一个本地应用级动态转发端口
-l login_name	指定登录远端主机的用户名

例如，通过命令 ssh，登录远程主机 192.168.20.76，如下所示。


```
# ssh 192.168.20.76 //远程登录主机
root@192.168.20.76's password:
Last login: Tue May 29 09:22:49 2008 from 192.168.20.122
#
```

例如，以用户 **teahcer** 身份登录远程主机 192.168.20.76，命令如下所示。

```
# ssh 192.168.20.76 -l teacher //远程登录主机
teacher@192.168.20.76's password:
Last login: Fri Dec 20 16:04:41 2008
$
```

8.5.33 statserial 命令：显示串口连接状态

statserial 命令可以用于显示串口的连接状态信息。statserial 显示标准的 9 针或 25 针串口列表，显示线路的连接状态，这对于调试串口或调制解调器错误将非常有帮助。通过按 **Ctrl+C** 组合键，可以终止 statserial 命令的状态显示。该命令的语法格式如下所示。

```
statserial [-n][-d ][-x] [device-name]
```

其中各选项的主要说明如表 8-40 所示。

表 8-40 statserial 命令主要选项说明

选 项	说 明
-n	默认每秒刷新一次，设置该项表示不进行刷新
-d	以十进制数显示状态，位编码格式如图 8-12 所示，其中 XXX 表示没有被使用
-x	以十六进制数显示状态
device-name	设备文件的全称，如果没有指定，默认值为环境变量中的 MODEM，否则为/dev/cual



图 8-12 statserial 命令的十进制输出结果

例如，通过命令 statserial，显示串口/dev/ttyS1 的详细信息，如果不按组合键 **Ctrl+C** 终止，系统将循环连续显示串口的状态，如下所示。

```
# statserial /dev/ttyS1 //显示串口状态
Device: /dev/ttyS1
Signal Pin Pin Direction Status Full
Name (25) (9) (computer) Name
-----
FG 1 - - - Frame Ground
TxD 2 3 out - Transmit Data
RxD 3 2 in - Receive Data
RTS 4 7 out 1 Request To Send
CTS 5 8 in 1 Clear To Send
DSR 6 6 in 1 Data Set Ready
GND 7 5 - - Signal Ground
DCD 8 1 in 1 Data Carrier Detect
DTR 20 4 out 1 Data Terminal Ready
RI 22 9 in 0 Ring Indicator
```



例如，显示串口/dev/ttyS1 的状态，但只显示一次，不进行连续循环显示，命令如下所示。

```
# statserial -n /dev/ttyS1 //显示一次串口状态
Device: /dev/ttyS1
Signal Pin Pin Direction Status Full
Name (25) (9) (computer) Name
-----
FG 1 - - - Frame Ground
TxD 2 3 out - Transmit Data
RxD 3 2 in - Receive Data
RTS 4 7 out 1 Request To Send
CTS 5 8 in 1 Clear To Send
DSR 6 6 in 1 Data Set Ready
GND 7 5 - - Signal Ground
DCD 8 1 in 1 Data Carrier Detect
DTR 20 4 out 1 Data Terminal Ready
RI 22 9 in 0 Ring Indicator
```

8.5.34 slrn 命令：新闻阅读程序

slrn 命令用来阅读和管理新闻组，其语法格式如下所示。

```
slrn [-C][ -a][ -d][ -n][ -p N][ -h host[:port]][ -i config-file][ -m][ --debug
file][ --help][ --version]
```

其中各选项的主要说明如表 8-41 所示。

表 8-41 slrn 命令主要选项说明

选 项	说 明
-C	彩色模式
-a	当检测新闻时，读活动文件
-d	从新闻服务器取得组描述
-n	不检查新的组
-p N	设置端口号
-h host[:port]	指定服务器及端口
-i config-file	指定配置文件
-m	强制鼠标支持
-f newsrc-file	使用指定的新闻资源文件
--debug file	将调试信息写入指定文件
--help	帮助信息
--version	版本信息

例如，连接新闻组服务器 news.usc.net，并指定使用新闻资源文件/root/.jnewsrc，命令如下所示。

```
#slrn -h news.usc.net -f /root/.jnewsrc //连接新闻组服务器
```

8.5.35 talk 命令：与其他用户交谈

talk 命令用来进行系统中在线用户间的交谈。talk 是可视化通信程序，可以将命令行复制

给另一个在线用户，该命令的语法格式如下所示。

```
talk [person] [ttyname]
```

其中各选项的主要说明如表 8-42 所示。

表 8-42 talk 命令主要选项说明

选 项	说 明
[person]	用户名，如 root
[ttyname]	如 tty1

例如，同其他用户进行交谈，命令如下所示。

```
# who //查看当前登录用户
teacher      tty1      May 30 10:00
root         pts/1      May 30 09:16 (192.168.88.7)
root         pts/2      May 30 09:28 (192.168.88.7)
root         :0         May 30 09:58
root         pts/3      May 30 09:59 (:0.0)
# talk teaher tty4 //同用户 teacher 交谈
```

8.5.36 tcpdump 命令：显示网络中的 TCP 数据包

tcpdump 命令用于显示网络中传输的 TCP 数据包的信息，只有系统管理员才能执行此指令。该命令的语法格式如下所示。

```
tcpdump [-a] [-d] [-dd] [-ddd] [-e] [-l] [-n] [-nn] [-N] [-O] [-q] [-t] [-v] [-X] [-c] [-F] [-i] [-r] [-w]
```

其中各选项的主要说明如表 8-43 所示。

表 8-43 tcpdump 命令主要选项说明

选 项	说 明
-a	显示名称，将网络或主机地址转换为主机名
-d	按可读方式转换数据包匹配编码，输出到标准输出设备
-dd	数据包匹配编码按 C 语言格式
-ddd	数据包匹配编码按十进制格式
-e	显示链路层数据包头
-l	输出缓冲区，适用于一边获取数据一边显示，如“tcpdump -l tee dat”或“tcpdump -l > dat & tail -f dat”
-n	不将主机地址转换为主机名，即以数字形式显示
-nn	不将协议、端口等转换为名称，使用数字形式进行显示
-N	不显示域名
-O	不进行包匹配代码优化
-q	快速输出，不显示详细的处理信息
-t	不显示时间
-v	运行时显示详细的处理信息，更高级别分别为“-vv”、“-vvv”
-X	以十六进制显示
-c	收到指定数目的数据包后停止输出



续表

选 项	说 明
-F	从指定文件读入过滤表达式
-i	侦听指定的网络接口
-r	从指定文件读入包数据
-w	把原始数据包写入指定文件

例如，显示当前系统捕获的 TCP 数据包的相关信息，命令如下所示。

```
# tcpdump //显示当前系统的 TCP 包信息
tcpdump: listening on eth0
18:03:23.047649 192.168.20.148.1285 > 192.168.20.122.telnet: . ack 1570810744 win 64836 (DF)
18:03:23.047725 192.168.20.122.telnet > 192.168.20.148.1285: P 1:29(28) ack 0 win 5840 (DF) [tos 0x10]
18:03:23.269603 192.168.20.148.1285 > 192.168.20.122.telnet: . ack 29 win 64808 (DF)
18:03:23.269680 192.168.20.122.telnet > 192.168.20.148.1285: P 29:228(199) ack 0 win 5840 (DF) [tos 0x10]
18:03:23.484041 192.168.20.148.1285 > 192.168.20.122.telnet: . ack 228 win 64609 (DF)
18:03:23.484886 192.168.20.122.telnet > 192.168.20.148.1285: P 228:422(194) ack 0 win 5840 (DF) [tos 0x10]
18:03:23.706905 192.168.20.148.1285 > 192.168.20.122.telnet: . ack 422 win 64415 (DF)
18:03:23.706978 192.168.20.122.telnet > 192.168.20.148.1285: P 422:618(196) ack 0 win 5840 (DF) [tos 0x10]
//按 Ctrl+c 结束
11 packets received by filter
0 packets dropped by kernel
```

例如，连续显示 5 个 TCP 数据包，然后自动退出 tcpdump，命令如下所示。

```
# tcpdump -c 5 //显示 5 个数据包
tcpdump: listening on eth0
18:31:04.936498 192.168.20.148.1285 > 192.168.20.122.telnet: . ack 1570839920 win 64721 (DF)
18:31:04.936589 192.168.20.122.telnet > 192.168.20.148.1285: P 1:29(28) ack 0 win 5840 (DF) [tos 0x10]
18:31:05.197105 192.168.20.148.1285 > 192.168.20.122.telnet: . ack 29 win 64693 (DF)
18:31:05.197188 192.168.20.122.telnet > 192.168.20.148.1285: P 29:228(199) ack 0 win 5840 (DF) [tos 0x10]
18:31:05.645565 192.168.20.148.1285 > 192.168.20.122.telnet: . ack 228 win 64494 (DF)
6 packets received by filter
0 packets dropped by kernel
```

例如，连续显示 5 个 TCP 数据包，然后自动退出 tcpdump，且显示输出时不显示时间，命令如下所示。

```
#tcpdump -c 5 -t //显示 5 个数据包但不显示时间
tcpdump: listening on eth0
192.168.20.148.1285 > 192.168.20.122.telnet: . ack 1570840548 win 64093 (DF)
192.168.20.122.telnet > 192.168.20.148.1285: P 1:29(28) ack 0 win 5840 (DF) [tos 0x10]
192.168.20.148.1285 > 192.168.20.122.telnet: . ack 29 win 65535 (DF)
192.168.20.122.telnet > 192.168.20.148.1285: P 29:196(167) ack 0 win 5840 (DF) [tos 0x10]
192.168.20.148.1285 > 192.168.20.122.telnet: . ack 196 win 65368 (DF)
6 packets received by filter
0 packets dropped by kernel
```

例如，通过命令 tcpdump，以精简模式显示 5 个 tcp 数据包，如下所示。


```
# tcpdump -c 5 -q //精简模式显示 5 个数据包
tcpdump: listening on eth0
18:36:42.264093 192.168.20.148.1285 > 192.168.20.122.telnet: tcp 0 (DF)
18:36:42.264178 192.168.20.122.telnet > 192.168.20.148.1285: tcp 28 (DF) [tos 0x10]
18:36:42.487237 192.168.20.148.1285 > 192.168.20.122.telnet: tcp 0 (DF)
18:36:42.487331 192.168.20.122.telnet > 192.168.20.148.1285: tcp 159 (DF) [tos 0x10]
18:36:42.708503 192.168.20.148.1285 > 192.168.20.122.telnet: tcp 0 (DF)
```

例如，显示指定网卡接口 eth0 中的 TCP 数据包，且显示 5 个数据包后退出，命令如下所示。

```
# tcpdump -c 5 -i eth0 //显示指定网卡的 TCP 数据包
tcpdump: listening on eth0
18:45:01.913753 192.168.20.148.1285 > 192.168.20.122.telnet: . ack 1570845725 win 65374 (DF)
18:45:01.913833 192.168.20.122.telnet > 192.168.20.148.1285: P 1:29(28) ack 0 win 5840 (DF) [tos 0x10]
18:45:02.122718 192.168.20.148.1285 > 192.168.20.122.telnet: . ack 29 win 65346 (DF)
18:45:02.123767 192.168.20.122.telnet > 192.168.20.148.1285: P 29:228(199) ack 0 win 5840 (DF) [tos 0x10]
18:45:02.349466 192.168.20.148.1285 > 192.168.20.122.telnet: . ack 228 win 65147 (DF)
6 packets received by filter
0 packets dropped by kernel
```

8.5.37 telnet 命令：使用 telnet 进行远程登录

telnet 命令可以使用 TELNET 协议进行远程登录，其语法格式如下所示。

```
telnet [-8] [-E] [-K] [-a] [-d] [-e escapechar] [-f] [-k realm] [-l user] [-n tracefile] [host [port]]
```

其中各选项的主要说明如表 8-44 所示。

表 8-44 telnet 命令主要选项说明

选 项	说 明
-8	使用 8 位数据路径
-a	自动登录
-d	调试模式
-K	关闭自动登录
-e escape char	设置 telnet 初始脱离字符
-E	不对脱离字符进行识别
-k realm	如果进行 Kerberos 认证，指定域名
-l user	指定用户名
-n tracefile	打开追踪文件，记录追踪信息
host	指定登录的主机
port	指定端口号

如果没有在 telnet 命令行中指定任何参数选项，telnet 将进入命令模式，提示符为“telnet>”。在命令模式下，telnet 可以接受并执行如表 8-45 所示的内部命令。



表 8-45 telnet 可使用的内部命令

选 项	说 明
open host [-a] [-l user] [-port]	连接到指定的主机，其中 host 表示主机名，-a 表示自动连接，-l user 表示指定的用户名，-port 表示指定连接的端口
quit	退出
set argument value	设置 telnet 环境变量的值。其中 argument 表示环境变量，value 表示值
unset argument value	清除 telnet 环境变量的值
status	显示当前 telnet 的状态
z	挂起 telnet
logout	重新登录

例如，通过命令 telnet，登录远程主机 192.168.20.122。直接在命令行中指定登录的主机地址，如下所示。

```
# telnet 192.168.20.122 //远程登录
Trying 192.168.20.122...
Connected to 192.168.20.122 (192.168.20.122).
Escape character is '^J'.
Red Hat Linux release 9 (Shrike)
Kernel 2.4.20-8 on an i686
login: teacher //输入用户名
Password: //输入密码
Last login: Sun Aug 10 17:44:56 from 192.168.20.148
$ //登录成功
```

例如，通过命令 telnet，登录远程主机 192.168.20.122。首先进入命令模式，然后通过命令“open”建立连接，如下所示。

```
# telnet
telnet> open 192.168.20.122
Trying 192.168.20.122...
Connected to 192.168.20.122 (192.168.20.122).
Escape character is '^J'.
Red Hat Linux release 9 (Shrike)
Kernel 2.4.20-8 on an i686
login: teacher //输入用户名
Password: //输入密码
Last login: Sun Aug 10 19:20:21 from 192.168.20.122
$ //登录成功
```

例如，直接运行命令 telnet，进入 telnet 命令模式，然后通过命令“status”显示当前连接建立情况，如下所示。

```
# telnet
telnet> status
No connection. //无连接
telnet>
```

8.5.38 testparm 命令：测试 samba 配置文件

samba 服务的配置文件为 smb.conf，testparm 命令通过读取该文件，可以测试 samba 服务的配置是否正确无误。该命令的语法格式如下所示。

testparm [-s] [-h] [-x] [-L <servername>] config filename [hostname hostIP]
其中各选项的主要说明如表 8-46 所示。

表 8-46 testparm 命令主要选项说明

选 项	说 明
-s	如果不指定该选项，testparm 在显示服务器角色类型之后将等待用户回车，然后继续显示配置信息。使用该选项之后，不等待用户回车，直接显示全部信息
-h	显示帮助信息
-x	显示没有默认值的参数
-L servername	设置%L 宏
configfilename	指定需要检测的配置文件
hostname	指定用户名
hostIP	指定 IP 地址

例如，测试 samba 配置，且在显示服务器角色类型之后暂停显示，等待回车后继续，命令如下所示。

```
# testparm                                     //测试 samba 配置
Load smb config files from /etc/samba/smb.conf
Processing section "[homes]"
Processing section "[printers]"
Loaded services file OK.
Press enter to see a dump of your service definitions //按回车键继续

# Global parameters
[global]                                     //全局参数设置
    coding system =
    client code page = 850
    code page directory = /usr/share/samba/codepages
    workgroup = MYGROUP
    netbios name =
    netbios aliases =
    ... ..
[homes]                                     //用户主目录设置
    comment = Home Directories
    valid users = %S
    read only = No
    create mask = 0664
    directory mask = 0775
    browseable = No

[printers]                                 //打印机参数设置
    comment = All Printers
    path = /var/spool/samba
    printable = Yes
    browseable = No
```

可以看到，在执行过程中显示提示信息“Press enter to see a dump of your service definitions”，等待用户确认后才能继续显示。



例如，不显示提示信息并等待用户确认，直接显示全部的 `samba` 配置信息，命令如下所示。

```
# testparm -s //检测 samba 配置，不显示提示
Load smb config files from /etc/samba/smb.conf
Processing section "[homes]"
Processing section "[printers]"
Loaded services file OK.
# Global parameters
[global] //全局参数设置
    coding system =
    client code page = 850
    code page directory = /usr/share/samba/codepages
    workgroup = MYGROUP
    netbios name =
    netbios aliases =
    ... ..
[homes] //用户主目录设置
    comment = Home Directories
    valid users = %S
    read only = No
    create mask = 0664
    directory mask = 0775
    browseable = No

[printers] //打印机参数设置
    comment = All Printers
    path = /var/spool/samba
    printable = Yes
    browseable = No
```

8.5.39 `tracpath` 命令：追踪路径

`tracpath` 命令用来实现对主机连接到目的地址时所经过的路由的追踪。该命令与 `traceroute` 命令相似，只是执行该命令不需要 `root` 权限，其语法格式如下所示。

```
tracpath destination [ port]
```

其中，选项“`destination`”表示目的主机，“`port`”表示端口号。

例如，通过命令 `tracpath`，显示连接到目的地址 `www.sohu.com` 的路由，执行过程如下所示。

```
# tracpath www.sohu.com //显示连接路由
1: 192.168.20.76 (192.168.20.76) 0.219ms pmtu 1500
1: 192.168.20.1 (192.168.20.1) 2.295ms
2: 202.112.146.2 (202.112.146.2) 3.384ms
3: 10.10.146.1 (10.10.146.1) 3.444ms
4: 10.10.147.1 (10.10.147.1) 19.639ms
5: 10.20.70.1 (10.20.70.1) 10.353ms
6: 10.10.42.2 (10.10.42.2) 8.194ms
7: 202.112.42.65 (202.112.42.65) 3.761ms
8: 202.112.62.241 (202.112.62.241) 4.146ms
9: 202.112.53.74 (202.112.53.74) 4.066ms
10: 202.112.38.82 (202.112.38.82) 3.822ms
11: 202.205.13.218 (202.205.13.218) 6.186ms
... ..
```




tracpath 还有一个针对 IPv6 的功能相似的命令 tracpath6。例如，对主机 (3ffe:2400:0:109::2) 进行路径追踪，命令如下所示。

```
# tracpath6 3ffe:2400:0:109::2
1?: [LOCALHOST]                pmtu 1500
1: dust.inr.ac.ru                0.411ms
2: dust.inr.ac.ru                asymm 1 0.390ms pmtu 1480
2: 3ffe:2400:0:109::2            463.514ms reached
... ..
```

8.5.40 traceroute 命令：显示到达目标网络主机的路由包

traceroute 命令追踪网络数据包，可以显示到达目标网络主机的路由包，通常用于检测到达目的地的路由状况。

traceroute 命令在每个 TTL 值发送 3 个分组，如果有一个发出的分组没有接收到响应，traceroute 就显示 1 个“*”，如果有响应则显示响应路由的名字和地址。traceroute 以毫秒为单位计算分组的往返时间，其语法格式如下所示。

```
traceroute [-d][-I][-v][-z][-r][-f][-F][-d][-g][-i][-m][-n][-p][-t]
```

其中各选项的主要说明如表 8-47 所示。

表 8-47 traceroute 命令主要选项说明

选 项	说 明
-d	开启调试功能
-I	使用 ICMP
-v	运行时，显示详细的处理信息
-z	设置两次探测之间的时间间隔
-r	对于相邻的网络，不经过路由表路由
-f	设置 TTL 值
-F	设置不对包进行分片
-d	启动 socket 级调试
-g	指定松散源路由网关，默认最大值为 8
-i	指定网络接口
-m	设置最大 TTL，默认最大为 30
-n	以数字方式显示每一跳地址
-p	设置 UDP 端口号，默认为 33434
-t	设置服务类型 QOS

例如，使用 traceroute 命令对到达 www.bit.edu.cn 的路由进行跟踪，且使用 IP 地址表示



每一跳主机，如下所示。

```
#tracerout -n www.bit.edu.cn //追踪路由
traceroute to www.bit.edu.cn (202.204.80.38), 30 hops max, 40 byte packets
 1  220.192.0.1(220.192.0.1)      352 ms  379 ms  359 ms
 2  220.192.0.20(220.192.0.20)   358 ms  359 ms  339 ms
 3  220.192.0.222(220.192.0.222) 332 ms  357 ms  339 ms
 4  192.168.32.2(192.168.32.2)   333 ms  339 ms  359 ms
 5  * * *                        //已丢失
...
30  * * *
```

可以看到，从第 5 行开始路由丢失（没有取得相应路由信息），从而显示 3 个“*”，直到其跳数达到 30 为止。

例如，使用 `traceroute` 命令对到达 `www.baidu.com` 的路由进行跟踪，如下所示。

```
# traceroute www.baidu.com //追踪路由
traceroute to www.a.shifen.com (211.94.144.100), 30 hops max, 40 byte packets
 1  202.112.146.2 (202.112.146.2) 1.427 ms  2.398 ms  13.981 ms
 2  10.10.146.1 (10.10.146.1) 1.553 ms  12.339 ms  16.526 ms
 3  10.10.147.1 (10.10.147.1) 14.018 ms  19.101 ms  19.803 ms
 4  10.20.70.1 (10.20.70.1) 13.360 ms  19.112 ms  20.123 ms
 5  10.10.42.2 (10.10.42.2) 3.863 ms  3.688 ms  3.741 ms
 6  202.112.42.65 (202.112.42.65) 1.574 ms  1.588 ms  1.289 ms
 7  202.112.62.241 (202.112.62.241) 2.315 ms  2.099 ms  1.640 ms
 8  202.112.53.178 (202.112.53.178) 1.739 ms  1.553 ms  1.252 ms
 9  202.38.123.22 (202.38.123.22) 155.748 ms  155.701 ms  155.364 ms
10  211.94.44.2 (211.94.44.2) 153.914 ms  153.561 ms  153.683 ms
11  211.94.68.110 (211.94.68.110) 155.247 ms  154.164 ms  154.652 ms
12  211.94.68.126 (211.94.68.126) 157.155 ms  157.394 ms  159.635 ms
13  * * *                        //已丢失
14  * * *
```

例如，使用 `traceroute` 命令对到达 `allspice.lcs.mit.edu` 的路由进行跟踪，如下所示。

```
#traceroute allspice.lcs.mit.edu
traceroute to allspice.lcs.mit.edu (18.26.0.115), 30 hops max
 1  helios.ee.lbl.gov (128.3.112.1) 0 ms  0 ms  0 ms
 2  lilac-dmc.Berkeley.EDU (128.32.216.1) 19 ms  19 ms  19 ms
 3  lilac-dmc.Berkeley.EDU (128.32.216.1) 39 ms  19 ms  19 ms
 4  ccngw-ner-cc.Berkeley.EDU (128.32.136.23) 19 ms  39 ms  39 ms
 5  ccn-nerif22.Berkeley.EDU (128.32.168.22) 20 ms  39 ms  39 ms
 6  128.32.197.4 (128.32.197.4) 59 ms  119 ms  39 ms
 7  131.119.2.5 (131.119.2.5) 59 ms  59 ms  39 ms
 8  129.140.70.13 (129.140.70.13) 80 ms  79 ms  99 ms
 9  129.140.71.6 (129.140.71.6) 139 ms  139 ms  159 ms
10  129.140.81.7 (129.140.81.7) 199 ms  180 ms  300 ms
11  129.140.72.17 (129.140.72.17) 300 ms  239 ms  239 ms
12  * * *
13  128.121.54.72 (128.121.54.72) 259 ms  499 ms  279 ms
14  * * *
15  * * *
16  * * *
17  * * *
18  ALLSPICE.LCS.MIT.EDU (18.26.0.115) 339 ms  279 ms  279 ms //已找到
```


8.5.41 tty 命令：显示标准输入设备名称

tty 命令用来查询当前使用的终端的名称，其语法格式如下所示。

```
tty [-s, --silent, --quiet][ --help][ --version]
```

其中各选项的主要说明如表 8-48 所示。

表 8-48 tty 命令主要选项说明

选 项	说 明
-s, --silent, --quiet	不显示处理信息，仅返回状态值
--help	帮助信息
--version	版本信息

例如，通过命令 tty，显示当前终端的名称，如下所示。

```
# tty //显示终端名
/dev/pts/3
```

8.5.42 wall 命令：发送信息

wall 命令可以向所有终端用户发送广播信息，其语法格式如下所法。

```
wall [ message ]
```

选项 “message” 表示广播信息。

例如，向所有用户发送信息 “Hello, I am Sullivan”，命令如下所示。

```
# wall hello, I am Sullivan //广播消息

Broadcast message from root (pts/3) (Sun Aug 10 21:37:41 2008):

hello, I am Sullivan
#
```

例如，向所有用户发送 “Hello, I am Kelly”，命令如下所示。

```
# wall
Hello, I am Kelly
//按 Ctrl+d 退出 wall
Broadcast message from root (pts/3) (Sun Aug 10 21:39:03 2008):
Hello, I am Kelly
#
```

8.5.43 wget 命令：从互联网上下载资源

wget 是一个自由软件，它采用非交互方式从互联网上下载文件。wget 支持 HTTP、HTTPS 和 FTP 协议。该命令的语法格式如下所示。

```
wget [-c, --continue][ -nc, --no-clobber][ -h, --help][ -x, --force-directorie][ -b,
--background][ -F, --force-html]
[ -e command, --execute command][ -o logfile, --output-file=logfile][ -a logfile,
--append-output=logfile]
[ -d, --debug][ -q, --quiet][ -v, --verbose][ -i file, --input-file=file][ -t number,
--tries=number][URL]
[-Q quota, --quota=quota]
```



其中各选项的主要说明如表 8-49 所示。

表 8-49 wget 命令主要选项说明

选 项	说 明
-c, --continue	支持断点续传功能
-nc, --no-clobber	不覆盖已经存在的文件
-h, --help	帮助信息
-x, --force-directorie	强制建立目录
-b, --background	后台执行
-e command, --execute command	执行指定的命令
-o logfile, --output-file=logfile	将日志信息写入指定文件
-a logfile, --append-output=logfile	追加到日志文件中
-d, --debug	开启调试模式
-q, --quiet	静默模式
-v, --verbose	冗余模式，运行时，显示详细的处理信息
-i file, --input-file=file	从指定文件读取 URL
-F, --force-html	将读入的文件视为 HTML
-t number, --tries=number	设置重试的次数
-Q quota, --quota=quota	指定下载配额
URL	指定下载地址

例如，通过命令 wget，从网站 192.168.20.148 上下载资料，如下所示。

```
# wget 192.168.20.148 //下载文件
--22:07:21-- http://192.168.20.148/
=> `index.html.1'
Connecting to 192.168.20.148:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [text/html]

[ <=> ] 39,199 6.23M/s

22:07:21 (6.23 MB/s) - `index.html.1' saved [39199]
#
```

8.5.44 ytalk 命令：多用户聊天程序

ytalk 命令可以用于和在线的其他用户交谈，也可以通过指定主机名和用户名，与不同主机上的用户交谈。该命令的语法格式如下所示。

```
ytalk [-x] [-s] [-Y] [-i] [-h hostname_or_ip] [username]
```

其中各选项的主要说明如表 8-50 所示。

表 8-50 ytalk 命令主要选项说明

选 项	说 明
-x	关闭图形界面
-s	在 Shell 中开启 ytalk 窗口
-i	关闭自动邀请端口
-h hostname_or_ip	指定本地主机的名称或 IP 地址
-Y	对于 yes/no 问题需要大写 Y 或 N 进行回答
username	指定交谈的用户，可采用如表 8-51 所示的表示方式

表 8-51 ytalk 命令中指定交谈用户的表示方式

用户名表示方式	说 明
name	当前主机上的用户
name@host	其他主机上的用户
name#tty	指定终端上的用户
name#tty@host	其他主机上特定终端上的用户
name@host#tty	与 “name#tty@host” 相同
aliasname	主机别名，在.ytalkrc 中定义

例如，首先查看当前在线用户，命令如下所示。

```
# who //显示用户信息
root :0 Aug 10 09:08
root pts/3 Aug 10 19:43 (192.168.20.148)
root pts/0 Aug 10 22:23 (192.168.20.148)
teacher pts/1 Aug 10 22:24 (192.168.20.148)
```

例如，当前用户 root 通过命令 “ytalk teacher”，同用户 teacher 进行交谈，如下所示。

```
# ytalk teacher //同用户 teacher 交谈
```

执行该命令后，root 用户进入等待状态，如图 8-13 所示。



图 8-13 root 用户在等待其他用户加入



此时，用户 teacher 所在终端将显示如下信息。

```
Message from Talk_Daemon@localhost.localdomain at 22:24 ...  
talk: connection requested by root@localhost.localdomain.  
talk: respond with: talk root@localhost.localdomain
```

用户 teacher 执行命令 ytalk，与 root 用户建立连接。此时，在 root 用户及 teacher 用户所在终端都将出现交互界面，两用户可以在此环境下开始聊天，如图 8-14 和图 8-15 所示。



图 8-14 root 用户的聊天界面



图 8-15 teacher 用户的聊天界面

无论在什么情况下，在 ytalk 界面下按下 ESC 键将出现如图 8-16 所示【Main Menu】菜单。按字母“q”键将退出 ytalk 程序，按字母“a”键将添加一个新用户，按字母“d”键将删除一个用户，按字母“u”键将显示用户列表。



图 8-16 【Main Menu】主菜单

8.6 FTP 常用命令

FTP 的全称是文件传输协议（File Transfer Protocol），它能够在 Internet 上提供文件传输服务，是现今广大用户获得丰富 Internet 资源的重要手段。FTP 能够实现文件共享、远程文件访问、对用户透明的文件存储技术以及高效可靠的数据传输。

FTP 采用客户端/服务器模型，客户端和服务端使用 TCP 建立连接。和大多数的应用层协议不同，FTP 协议在两台主机之间建立了两条通信链路，分别是：控制连接和数据连接。控制连接主要负责传送在会话过程中用户发送的 FTP 命令和 FTP 服务的响应，数据连接主要负责传输数据。

在服务器端，FTP 服务器有两个预分配的端口号，分别为 20 和 21。其中，端口 20 用于发送和接收 FTP 数据（ASCII 码或二进制文件），该数据端口只在传输数据时打开，并在传输结束时关闭。端口 21 用于发送和接收 FTP 的控制信息，FTP 服务器通过侦听这个端口来判断是否有 FTP 客户的连接请求，FTP 会话建立后，端口 21 的连接在会话期间将始终保持打开状态。

在客户端，当发出请求时将动态分配端口号，选择范围为 1024~65535。当一个 FTP 会话开始后，客户端程序打开一个控制端口（如 1038），该端连接到服务器上的 21 端口，接收和发送控制命令；需要传输数据时，客户端再打开第 2 个端口（如 1039），连接到服务器上的 20 端口。每当开始传输文件时，客户端程序会打开一个新的数据端口，在文件传输完毕后，再自动释放该端口。

8.6.1 ftp 命令：文件传输协议

ftp 命令是 ARPANET 标准文件传输协议的用户接口，允许用户从远端网络站点下载文件



或传输文件到远端网络站点。该命令的语法格式如下所示。

```
ftp [-v] [-d] [-i] [-n] [-g] [-k realm] [-x] [-u] [host]
```

其中各选项的主要说明如表 8-52 所示。

表 8-52 ftp 命令主要选项说明

选 项	说 明
-d	启动调试模式
-u	关闭自动认证
-e	不记录历史指令
-i	关闭交互模式
-x	在成功认证之后，协商密钥
-n	关闭自动登录功能
-p	传输文件的模式为被动模式
-v	程序运行时，显示详细的处理信息
-k realm	使用 Kerberos v4 认证时，从 realm 中得到信息
host	FTP 服务器的主机名/IP 地址

在客户端访问 FTP 服务器时，如果没有在命令行给出服务器的主机名或 IP，则客户端将出现“ftp>”提示符，等待用户输入 ftp 内部命令。

在“ftp>”提示符下输入“？”，可以查看 ftp 常用内部命令。表 8-53 列出了常见的 ftp 命令及其功能。

表 8-53 常见 ftp 内部命令及其说明

内 部 命 令	说 明
ls	显示服务器上的目录
get	从服务器下载指定文件到客户端
put	从客户端传送指定文件到服务器
open	连接 ftp 服务器
quit	断开连接并退出 ftp 服务器
cd directory	改变服务器的当前目录为 directory
lcd directory	改变本地当前目录为 directory
bye	退出 ftp 命令状态
ascii	设置文件传输方式为 ASCII 模式
binary	设置文件传输方式为二进制模式
!	执行本地主机的命令
cd	切换远端 ftp 服务器上的目录
cdup	上一层目录
close	在不结束 ftp 进程的情况下，关闭与 ftp 服务器的链接
delete	删除远端 ftp 服务器上的文件

续表

内部命令	说明
get	下载
hash	显示#
mdelete	删除文件，模糊匹配
mget	下载文件，模糊匹配
mkdir	在远端 ftp 服务器上，建立文件夹
mput	上传文件，模糊匹配
newer	下载时，检测是不是新文件
prompt	关闭交互模式
put	上传
pwd	显示当前目录

例如，建立到 FTP 服务器 192.168.20.2 的连接，命令如下所示。

```
# ftp
ftp> open 192.168.20.122 //建立 ftp 连接
Connected to 192.168.20.122.
220 (vsFTPd 1.1.3)
530 Please login with USER and PASS.
530 Please login with USER and PASS.
KERBEROS_V4 rejected as an authentication type
Name (192.168.20.122:root): teacher //输入用户名
331 Please specify the password. //输入密码
Password:
230 Login successful. Have fun.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> ls //列出当前目录
227 Entering Passive Mode (192,168,20,122,103,132)
150 Here comes the directory listing.
-rw-r--r-- 1 0 0 289 Aug 10 15:38 my.doc
-rw-r--r-- 1 0 0 1005 Aug 10 15:38 myback
-rw-r--r-- 1 0 0 103 Aug 10 15:38 mybook
-rw-r--r-- 1 0 0 91 Aug 10 15:38 mydoc
-rw-r--r-- 1 0 0 103 Aug 10 15:38 mylist
-rw-r--r-- 1 0 0 102 Aug 10 15:38 mypaper
-rw-r--r-- 1 0 0 102 Aug 10 15:38 mypaper.bak
-rw-r--r-- 1 0 0 102 Aug 10 15:38 mypaper.doc
drwxr-xr-x 2 0 0 4096 Aug 10 15:40 test
-rw-r--r-- 1 0 0 2051981 Aug 10 15:38 test.bak
226 Directory send OK.
ftp> cd test //更改目录
250 Directory successfully changed.
ftp> pwd //显示远端当前目录
257 "/home/teacher/test"
ftp> cdup //返回上级目录
250 Directory successfully changed.
ftp> pwd //显示远端当前目录
```



```
257 "/home/teacher"
ftp> lcd /yy //改变本地目录
Local directory now /yy
ftp> !pwd //显示本地目录
/yy
ftp> bye //结束 ftp 连接
221 Goodbye.
```

例如，从 FTP 服务器上传/下载文件，命令如下所示。

```
# ftp
ftp> open 192.168.20.122 //建立连接
Connected to 192.168.20.122.
220 (vsFTPD 1.1.3)
530 Please login with USER and PASS.
530 Please login with USER and PASS.
KERBEROS_V4 rejected as an authentication type
Name (192.168.20.122:root): teacher //输入用户名
331 Please specify the password.
Password: //输入密码
230 Login successful. Have fun.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> ls //显示远端当前目录
227 Entering Passive Mode (192,168,20,122,24,151)
150 Here comes the directory listing.
-rw-r--r-- 1 0 0 289 Aug 10 15:38 my.doc
-rw-r--r-- 1 0 0 1005 Aug 10 15:38 myback
-rw-r--r-- 1 0 0 103 Aug 10 15:38 mybook
-rw-r--r-- 1 0 0 91 Aug 10 15:38 mydoc
-rw-r--r-- 1 0 0 103 Aug 10 15:38 mylist
-rw-r--r-- 1 0 0 102 Aug 10 15:38 mypaper
-rw-r--r-- 1 0 0 102 Aug 10 15:38 mypaper.bak
-rw-r--r-- 1 0 0 102 Aug 10 15:38 mypaper.doc
drwxr-xr-x 2 0 0 4096 Aug 10 15:40 test
-rw-r--r-- 1 0 0 2051981 Aug 10 15:38 test.bak
226 Directory send OK.
ftp> !ls //显示本地当前目录
anaconda-ks.cfg index.html install.log minicom.log
dump.tmp index.html.1 install.log.syslog myfifo
ftp> get mypaper.doc //下载文件 mypaper.doc
local: mypaper.doc remote: mypaper.doc
227 Entering Passive Mode (192,168,20,122,172,100)
150 Opening BINARY mode data connection for mypaper.doc (102 bytes).
226 File send OK.
102 bytes received in 0.00019 seconds (5.3e+02 Kbytes/s)
ftp> !ls //显示本地当前目录
anaconda-ks.cfg index.html install.log minicom.log mypaper.doc //已下载
mypaper.doc
dump.tmp index.html.1 install.log.syslog myfifo
ftp> mget test* //下载文件名以 test 开头的文件
mget test? y //下载 test
227 Entering Passive Mode (192,168,20,122,23,179)
550 Failed to open file.
mget test.bak? y //下载 test.bak
227 Entering Passive Mode (192,168,20,122,175,191)
```



```

150 Opening BINARY mode data connection for test.bak (2051981 bytes).
226 File send OK.
2051981 bytes received in 0.016 seconds (1.3e+05 Kbytes/s)
ftp> !ls //显示本地当前目录
anaconda-ks.cfg index.html install.log minicom.log mypaper.doc
dump.tmp index.html.1 install.log.syslog myfifo test.bak
ftp> put index.html //上传 index.html 文件
local: index.html remote: index.html
227 Entering Passive Mode (192,168,20,122,245,46)
150 Ok to send data.
226 File receive OK.
39199 bytes sent in 0.0005 seconds (7.6e+04 Kbytes/s)
ftp> ls //显示远端当前目录
227 Entering Passive Mode (192,168,20,122,148,230)
150 Here comes the directory listing.
-rw-r--r-- 1 500 500 39199 Aug 10 15:54 index.html //已上传
-rw-r--r-- 1 0 0 289 Aug 10 15:38 my.doc
-rw-r--r-- 1 0 0 1005 Aug 10 15:38 myback
-rw-r--r-- 1 0 0 103 Aug 10 15:38 mybook
-rw-r--r-- 1 0 0 91 Aug 10 15:38 mydoc
-rw-r--r-- 1 0 0 103 Aug 10 15:38 mylist
-rw-r--r-- 1 0 0 102 Aug 10 15:38 mypaper
-rw-r--r-- 1 0 0 102 Aug 10 15:38 mypaper.bak
-rw-r--r-- 1 0 0 102 Aug 10 15:38 mypaper.doc
drwxr-xr-x 2 0 0 4096 Aug 10 15:40 test
-rw-r--r-- 1 0 0 2051981 Aug 10 15:38 test.bak
226 Directory send OK.
ftp> mput install.* //上传多个文件名以 install 开头的文件
mput install.log? y //上传 install.log
227 Entering Passive Mode (192,168,20,122,200,192)
150 Ok to send data.
226 File receive OK.
49492 bytes sent in 0.059 seconds (8.2e+02 Kbytes/s)
mput install.log.syslog? y //上传 install.log.syslog
227 Entering Passive Mode (192,168,20,122,38,61)
150 Ok to send data.
226 File receive OK.
ftp> ls //显示远端当前目录
227 Entering Passive Mode (192,168,20,122,187,110)
150 Here comes the directory listing.
-rw-r--r-- 1 500 500 39199 Aug 10 15:54 index.html
-rw-r--r-- 1 500 500 49492 Aug 10 15:55 install.log
-rw-r--r-- 1 500 500 7113 Aug 10 15:55 install.log.syslog
-rw-r--r-- 1 0 0 289 Aug 10 15:38 my.doc
-rw-r--r-- 1 0 0 1005 Aug 10 15:38 myback
-rw-r--r-- 1 0 0 103 Aug 10 15:38 mybook
-rw-r--r-- 1 0 0 91 Aug 10 15:38 mydoc
-rw-r--r-- 1 0 0 103 Aug 10 15:38 mylist
-rw-r--r-- 1 0 0 102 Aug 10 15:38 mypaper
-rw-r--r-- 1 0 0 102 Aug 10 15:38 mypaper.bak
-rw-r--r-- 1 0 0 102 Aug 10 15:38 mypaper.doc
drwxr-xr-x 2 0 0 4096 Aug 10 15:40 test
-rw-r--r-- 1 0 0 2051981 Aug 10 15:38 test.bak
226 Directory send OK.
7113 bytes sent in 0.012 seconds (5.9e+02 Kbytes/s)

```




```
ftp> bye //结束本次 ftp
221 Goodbye.
#
```

例如，尝试使用“open”和“close”命令建立和断开连接，执行命令如下所示。

```
# ftp 192.168.20.122 //建立连接
Connected to 192.168.20.122.
220 (vsFTPd 1.1.3)
530 Please login with USER and PASS.
530 Please login with USER and PASS.
KERBEROS_V4 rejected as an authentication type
Name (192.168.20.122:root): teacher //输入用户名
331 Please specify the password.
Password: //输入密码
230 Login successful. Have fun.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> ls //显示远端当前目录
227 Entering Passive Mode (192,168,20,122,254,4)
150 Here comes the directory listing.
-rw-r--r-- 1 500 500 39199 Aug 10 15:54 index.html
-rw-r--r-- 1 500 500 49492 Aug 10 15:55 install.log
-rw-r--r-- 1 500 500 7113 Aug 10 15:55 install.log.syslog
-rw-r--r-- 1 0 0 289 Aug 10 15:38 my.doc
-rw-r--r-- 1 0 0 1005 Aug 10 15:38 myback
-rw-r--r-- 1 0 0 103 Aug 10 15:38 mybook
-rw-r--r-- 1 0 0 91 Aug 10 15:38 mydoc
-rw-r--r-- 1 0 0 103 Aug 10 15:38 mylist
-rw-r--r-- 1 0 0 102 Aug 10 15:38 mypaper
-rw-r--r-- 1 0 0 102 Aug 10 15:38 mypaper.bak
-rw-r--r-- 1 0 0 102 Aug 10 15:38 mypaper.doc
drwxr-xr-x 2 0 0 4096 Aug 10 15:40 test
-rw-r--r-- 1 0 0 2051981 Aug 10 15:38 test.bak
226 Directory send OK.
ftp> close //关闭连接
221 Goodbye.
ftp> ls //显示远端当前目录
Not connected. //显示无连接
ftp> open //打开连接
(to) 192.168.20.122
Connected to 192.168.20.122.
220 (vsFTPd 1.1.3)
530 Please login with USER and PASS.
530 Please login with USER and PASS.
KERBEROS_V4 rejected as an authentication type
Name (192.168.20.122:root): teacher //输入用户名
331 Please specify the password.
Password: //输入密码
230 Login successful. Have fun.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> ls //显示远端当前目录
227 Entering Passive Mode (192,168,20,122,25,106)
150 Here comes the directory listing.
-rw-r--r-- 1 500 500 39199 Aug 10 15:54 index.html
```



```

-rw-r--r-- 1 500 500 49492 Aug 10 15:55 install.log
-rw-r--r-- 1 500 500 7113 Aug 10 15:55 install.log.syslog
-rw-r--r-- 1 0 0 289 Aug 10 15:38 my.doc
-rw-r--r-- 1 0 0 1005 Aug 10 15:38 myback
-rw-r--r-- 1 0 0 103 Aug 10 15:38 mybook
-rw-r--r-- 1 0 0 91 Aug 10 15:38 mydoc
-rw-r--r-- 1 0 0 103 Aug 10 15:38 mylist
-rw-r--r-- 1 0 0 102 Aug 10 15:38 mypaper
-rw-r--r-- 1 0 0 102 Aug 10 15:38 mypaper.bak
-rw-r--r-- 1 0 0 102 Aug 10 15:38 mypaper.doc
drwxr-xr-x 2 0 0 4096 Aug 10 15:40 test
-rw-r--r-- 1 0 0 2051981 Aug 10 15:38 test.bak
226 Directory send OK.
ftp> bye //退出 ftp 连接
221 Goodbye.
#

```

例如，联合使用“delete”“mkdir”及“hash”命令，如下所示。

```

# ftp 192.168.20.122 //建立 ftp 连线
Connected to 192.168.20.122.
220 (vsFTPD 1.1.3)
530 Please login with USER and PASS.
530 Please login with USER and PASS.
KERBEROS_V4 rejected as an authentication type
Name (192.168.20.122:root): teacher //输入用户名
331 Please specify the password.
Password: //输入密码
230 Login successful. Have fun.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> ls //显示远端当前目录
227 Entering Passive Mode (192,168,20,122,253,113)
150 Here comes the directory listing.
-rw-r--r-- 1 500 500 39199 Aug 10 15:54 index.html
-rw-r--r-- 1 500 500 49492 Aug 10 15:55 install.log
-rw-r--r-- 1 500 500 7113 Aug 10 15:55 install.log.syslog
-rw-r--r-- 1 0 0 289 Aug 10 15:38 my.doc
-rw-r--r-- 1 0 0 1005 Aug 10 15:38 myback
-rw-r--r-- 1 0 0 103 Aug 10 15:38 mybook
-rw-r--r-- 1 0 0 91 Aug 10 15:38 mydoc
-rw-r--r-- 1 0 0 103 Aug 10 15:38 mylist
-rw-r--r-- 1 0 0 102 Aug 10 15:38 mypaper
-rw-r--r-- 1 0 0 102 Aug 10 15:38 mypaper.bak
-rw-r--r-- 1 0 0 102 Aug 10 15:38 mypaper.doc
drwxr-xr-x 2 0 0 4096 Aug 10 15:40 test
-rw-r--r-- 1 0 0 2051981 Aug 10 15:38 test.bak
226 Directory send OK.
ftp> mkdir mybox //在远端当前目录中创建目录 mybox
257 "/home/teacher/mybox" created
ftp> ls //显示远端当前目录
227 Entering Passive Mode (192,168,20,122,45,55)
150 Here comes the directory listing.
-rw-r--r-- 1 500 500 39199 Aug 10 15:54 index.html
-rw-r--r-- 1 500 500 49492 Aug 10 15:55 install.log
-rw-r--r-- 1 500 500 7113 Aug 10 15:55 install.log.syslog

```




```
-rw-r--r-- 1 0 0 289 Aug 10 15:38 my.doc
-rw-r--r-- 1 0 0 1005 Aug 10 15:38 myback
-rw-r--r-- 1 0 0 103 Aug 10 15:38 mybook
drwxr-xr-x 2 500 500 4096 Aug 10 16:18 mybox //已创建 mybox 目录
-rw-r--r-- 1 0 0 91 Aug 10 15:38 mydoc
-rw-r--r-- 1 0 0 103 Aug 10 15:38 mylist
-rw-r--r-- 1 0 0 102 Aug 10 15:38 mypaper
-rw-r--r-- 1 0 0 102 Aug 10 15:38 mypaper.bak
-rw-r--r-- 1 0 0 102 Aug 10 15:38 mypaper.doc
drwxr-xr-x 2 0 0 4096 Aug 10 15:40 test
-rw-r--r-- 1 0 0 2051981 Aug 10 15:38 test.bak
226 Directory send OK.
ftp> cd mybox //改变远端当前目录为 mybox
250 Directory successfully changed.
ftp> ls //显示远端当前目录
227 Entering Passive Mode (192,168,20,122,164,109)
150 Here comes the directory listing.
226 Directory send OK.
ftp> !ls //显示本地当前目录
anaconda-ks.cfg index.html install.log minicom.log mypaper.doc
dump.tmp index.html.1 install.log.syslog myfifo test.bak
ftp> put index.html //上传 index.html
local: index.html remote: index.html
227 Entering Passive Mode (192,168,20,122,136,22)
150 Ok to send data.
226 File receive OK.
39199 bytes sent in 0.00023 seconds (1.7e+05 Kbytes/s)
ftp> ls //显示远端当前目录
227 Entering Passive Mode (192,168,20,122,27,200)
150 Here comes the directory listing.
-rw-r--r-- 1 500 500 39199 Aug 10 16:18 index.html //已成功上传
226 Directory send OK.
ftp> delete index.html //删除 index.html
250 Delete operation successful.
ftp> ls //显示远端当前目录
227 Entering Passive Mode (192,168,20,122,32,118)
150 Here comes the directory listing. //已删除
226 Directory send OK.
ftp> hash //设置哈希,即上传或下载时显示“#”以表示进程
Hash mark printing on (1024 bytes/hash mark).
ftp> mput install* //上传多个文件名以 install 开头的文件
mput install.log? y
227 Entering Passive Mode (192,168,20,122,183,54)
150 Ok to send data.
##### //显示上传进度
226 File receive OK.
49492 bytes sent in 0.00022 seconds (2.2e+05 Kbytes/s)
mput install.log.syslog? y
227 Entering Passive Mode (192,168,20,122,28,114)
150 Ok to send data.
##### //显示上传进度
226 File receive OK.
7113 bytes sent in 8.6e-05 seconds (8.1e+04 Kbytes/s)
ftp> ls //显示远端当前目录
227 Entering Passive Mode (192,168,20,122,226,131)
```



```
150 Here comes the directory listing.
-rw-r--r--    1 500      500      49492 Aug 10 16:20 install.log
-rw-r--r--    1 500      500      7113 Aug 10 16:20 install.log.syslog
#
226 Directory send OK.
ftp> bye
221 Goodbye
#
```

//结束 ftp

8.6.2 ncftp 命令：传输文件

ncftp 命令为 Internet 标准文件传输协议提供了一个强大的、可扩展的接口。ncftp 在性能和便捷性方面具有的优势，使其具有取代 ftp 命令的趋势。该命令的语法格式如下所示。

```
ncftp [-u XX][ -p XX][ -P XX][ -j XX][ -h][ -v][ -F]
```

其中各选项的主要说明如表 8-54 所示。

表 8-54 ncftp 命令主要选项说明

选 项	说 明
-u XX	指定用户名
-p XX	指定密码
-P XX	指定端口号
-j XX	指定账号
-h	帮助信息
-v	版本信息
-F	显示 ncftp 防火墙参数



ncftp 不支持 http 代理（如 squid 或 Netscape Proxy Server 等代理服务器）。

在客户端访问 FTP 服务器时，如果没有在命令行给出服务器的主机名或 IP，则客户端将出现“ncftp>”提示符，等待用户输入 ncftp 内部命令。

通过在“ncftp>”提示符下输入“？”，可以查看 ncftp 常用内部命令。表 8-55 列出了常见的 ncftp 命令及其说明。

表 8-55 常见 ncftp 内部命令及其说明

选 项	说 明
!	执行本地机器的命令
ascii	传输模式为 ASCII
binary	传输模式为二进制模式
bye	结束 ftp
cd	切换远端 ftp 服务器上的目录
cdup	上一层目录



续表

选 项	说 明
close	在不结束 ftp 进程的情况下，关闭与 ftp 服务器的连接
delete	删除远端 ftp 服务器上的文件
get	下载
hash	显示#
lcd	更改本地目录
ls	列出远端 ftp 服务器上的文件列表
mdelete	删除文件，模糊匹配多个文件
mget	下载文件，模糊匹配多个文件
mkdir	在远端 ftp 服务器上，建立文件夹
mput	上传文件，模糊匹配多个文件
newer	下载时，检测是不是新文件
open	打开连线
prompt	关闭交互模式
put	上传
pwd	显示当前目录
quit	退出

例如，显示 ncftp 所有内部命令的列表，并查询内部命令的帮助信息，命令如下所示。

```
#ncftp
ncftp> ? //显示 ncftp 所有内部命令的列表
Commands may be abbreviated. 'help showall' shows hidden and unsupported
commands. 'help <command>' gives a brief description of <command>.

ascii      cat        help       lpage      open       quit       show
bgget      cd         jobs       lpwd       page       quote      site
bgput      chmod     lcd        lrename    passive    rename     type
bgstart    close     lchmod     lrm        pdir       rhelp      umask
binary     debug     lls        lrmdir     pls        rm         version
bookmark   dir       lmkdir     ls         put        rmdir
bookmarks  get       lookup     mkdir      pwd        set

ncftp> help lcd //显示 lcd 命令的帮助信息
lcd: changes local working directory.
Usage: lcd <directory>

For details, please see the manual ("man ncftp" at your regular shell prompt
or online at http://www.ncftp.com/ncftp/doc/ncftp.html).
ncftp> help lookup //显示 lookup 命令的帮助信息
lookup: looks up information in the host database.
Usage: lookup [-v|-V] <host or IP number> [<more hosts or IP numbers>]
For details, please see the manual ("man ncftp" at your regular shell prompt
or online at http://www.ncftp.com/ncftp/doc/ncftp.html).
ncftp>
```

例如，通过命令“ncftp -F”，显示当前主机的防火墙设置，如下所示。


```

# ncftp -F                                //显示 ncftp 防火墙参数
# NcFTP firewall preferences
# =====
#
# If you need to use a proxy for FTP, you can configure it below.
# If you do not need one, leave the ``firewall-type'' variable set
# to 0. Any line that does not begin with the ``#' character is
# considered a configuration command line.
#
# NOTE: NcFTP does NOT support HTTP proxies that do FTP, such as "squid"
#       or Netscape Proxy Server. Why? Because you have to communicate with
#       them using HTTP, and this is a FTP only program.
#
# Types of firewalls:                      //防火墙类型
# -----
#
# type 1: Connect to firewall host, but send "USER user@real.host.name"
#
# type 2: Connect to firewall, login with "USER fwuser" and
#         "PASS fwpassword", and then "USER user@real.host.name"
#
# type 3: Connect to and login to firewall, and then use
#         "SITE real.host.name", followed by the regular USER and PASS.
#
# type 4: Connect to and login to firewall, and then use
#         "OPEN real.host.name", followed by the regular USER and PASS.
#
# type 5: Connect to firewall host, but send
#         "USER user@fwuser@real.host.name" and
#         "PASS pass@fwpass" to login.
#
# type 6: Connect to firewall host, but send
#         "USER fwuser@real.host.name" and
#         "PASS fwpass" followed by a regular
#         "USER user" and
#         "PASS pass" to complete the login.
#
# type 7: Connect to firewall host, but send
#         "USER user@real.host.name fwuser" and
#         "PASS pass" followed by
#         "ACCT fwpass" to complete the login.
#
# type 8: Connect to firewall host, but send "USER user@real.host.name:port"
#
# type 9: Connect to firewall host, but send "USER user@real.host.name port"
#
# type 0: Do NOT use a firewall (most users will choose this). //类型 0: 不使用防火墙
#
firewall-type=0                                //变量“firewall-type”设定为 0, 即不使用防火墙
#
#
# The ``firewall-host'' variable should be the IP address or hostname of
# your firewall server machine.
#

```




```
firewall-host=firewall.localdomain
#
#
#
# The ``firewall-user'' variable tells NcFTP what to use as the user ID
# when it logs in to the firewall before connecting to the outside world.
#
firewall-user=root                //变量“firewall-user”可指定用户名
#
#
#
# The ``firewall-password'' variable is the password associated with
# the firewall-user ID. If you set this here, be sure to change the
# permissions on this file so that no one (except the superuser) can
# see your password. You may also leave this commented out, and then
# NcFTP will prompt you each time for the password.
#
firewall-password=fwpass          //变量“firewall-password”可指定密码
#
#
#
# Your firewall may require you to connect to a non-standard port for
# outside FTP services, instead of the internet standard port number (21).
#
firewall-port=21                  //变量“firewall-port”可指定非标准端口
#
#
#
# You probably do not want to FTP to the firewall for hosts on your own
# domain. You can set ``firewall-exception-list'' to a list of domains
# or hosts where the firewall should not be used. For example, if your
# domain was ``probe.net'' you could set this to ``.probe.net''.
#
# If you leave this commented out, the default behavior is to attempt to
# lookup the current domain, and exclude hosts for it. Otherwise, set it
# to a list of comma-delimited domains or hostnames. The special token
# ``localdomain'' is used for unqualified hostnames, so if you want hosts
# without explicit domain names to avoid the firewall, be sure to include
# that in your list.
#
firewall-exception-list=.localdomain,localhost,localdomain //设定不使用防火墙的
域或主机
#
#
#
# You may also specify passive mode here. Normally this is set in the
# regular $HOME/.ncftp/prefs file. This must be set to one of
# "on", "off", or "optional", which mean always use PASV,
# always use PORT, and try PASV then PORT, respectively.
#
#passive=on                      //指定被动模式
#
#
#
```



```
# NOTE: This file was created for you on Fri May 25 09:19:16 2007
#       by NcFTP 3.2.0. Removing this file will cause the next run of NcFTP
#       to generate a new one, possibly with more configurable options.
#
# ALSO: A /usr/local/etc/ncftp.firewall file, if present, is processed before this file,
#       and a /usr/local/etc/ncftp.firewall.fixed file, if present, is processed after.
#
```

例如，以用户 teacher 登录 ftp 服务器 192.168.20.122，teacher 用户的密码为“654321”，登录服务器后下载文件。命令执行过程如下所示。

```
# ncftp -u teacher -p 654321 192.168.20.122 //登录 ftp
NcFTP 3.1.5 (Oct 13, 2002) by Mike Gleason (ncftp@ncftp.com).
Connecting to 192.168.20.122...
(vsFTPD 1.1.3)
Logging in...
Login successful. Have fun.
Sorry, I don't have help.
Logged in to 192.168.20.122.
ncftp /home/teacher > ls //显示远端当前目录
index.html          mybook              mylist              test/
install.log         mybox/             mypaper            test.bak
install.log.syslog  mydoc              mypaper.bak
myback              my.doc             mypaper.doc
ncftp /home/teacher > !ls //显示本地当前目录
mm mydump.bz2 test test.bak
ncftp /home/teacher > !pwd //显示本地当前目录路径
/yy
ncftp /home/teacher > pwd //显示远端当前目录路径
ftp://teacher:PASSWORD@192.168.20.122
ncftp /home/teacher > get mylist //下载文件 mylist
mylist:
103.00 B 313.35 kB/s
ncftp /home/teacher > !ls //显示本地当前目录
mm mydump.bz2 mylist test test.bak //已下载成功
ncftp /home/teacher > mget mypaper* //下载文件名以 mypaper 开头的多个文件
mypaper:
102.00 B 456.92 kB/s
mypaper.bak:
102.00 B 686.96 kB/s
mypaper.doc:
102.00 B 913.85 kB/s
ncftp /home/teacher > !ls //显示本地当前目录
mm mydump.bz2 mylist mypaper mypaper.bak mypaper.doc test test.bak //已成功下载
ncftp /home/teacher > bye //结束 ftp
#
```

例如，使用 ncftp 登录 ftp 服务器，执行创建目录、删除文件、下载文件等操作，如下所示。

```
# ncftp -u teacher 192.168.20.122 //以 teacher 用户身份登录服务器 192.168.20.122
NcFTP 3.1.5 (Oct 13, 2002) by Mike Gleason (ncftp@ncftp.com).
Connecting to 192.168.20.122...
(vsFTPD 1.1.3)
Logging in...
Password requested by 192.168.20.122 for user "teacher".

Please specify the password.

Password: ***** //输入密码

Login successful. Have fun.
```




```
Sorry, I don't have help.
Logged in to 192.168.20.122.
ncftp /home/teacher > ls
index.html          mybook
install.log         mybox/
install.log.syslog  mydoc
myback              my.doc
ncftp /home/teacher > mkdir mywork
ncftp /home/teacher > ls
index.html          mybook
install.log         mybox/
install.log.syslog  mydoc
myback              my.doc
ncftp /home/teacher > cd mywork
Directory successfully changed.
ncftp /home/teacher/mywork > ls
ncftp /home/teacher/mywork > !ls
hh test test.bak
ncftp /home/teacher/mywork > mput test.bak
test.bak:
ncftp /home/teacher/mywork > ls
test.bak
ncftp /home/teacher/mywork > delete test.bak
ncftp /home/teacher/mywork > ls
ncftp /home/teacher/mywork > bye
#
```

//显示远端当前目录

mylist	test/
mypaper	test.bak
mypaper.bak	
mypaper.doc	

//在远端服务器创建 mywork 目录

mylist	mywork/
mypaper	test/
mypaper.bak	
test.bak	
mypaper.doc	

//改变远端当前目录为 mywork

//显示远端当前目录

//显示本地当前目录

//上传 test.bak

1.96 MB 43.87 MB/s

//显示远端当前目录

//删除远端当前目录中的 test.bak

//显示远端当前目录

//结束 ftp

例如，使用匿名账号登录 ftp 服务器，无需输入用户名和密码，但需 ftp 服务器支持匿名登录，命令如下所示。

```
# ncftp 192.168.20.122
NcFTP 3.1.5 (Oct 13, 2002) by Mike Gleason (ncftp@ncftp.com).
Connecting to 192.168.20.122...
(vsFTPD 1.1.3)
Logging in...
Login successful. Have fun.
Sorry, I don't have help.
Logged in to 192.168.20.122.
ncftp / > ls
pub/
ncftp / > pwd
ftp://192.168.20.122
ncftp / > bye
#
```

//登录 ftp 服务器

//显示远端当前目录

//显示远端当前目录的路径

//结束 ftp

8.6.3 ncftpget 命令：下载文件

ncftpget 命令用来直接获取远端 ftp 服务器上的文件。ncftpget 命令基于命令行，不用进入交互 Shell 方式即可方便地下载文件，因此非常适合在脚本中使用。该命令的语法格式如下所示。

```
ncftpget [-u XX] [-p XX] [-P XX] [-j XX] [-d XX] [-a] [-t XX] [-v, -V] [-f XX] [-c] [-A] [-z/-Z] [-E] [-F] [-DD] [-R] [-B XX] [-W XX] [-W XX] [-X XX] [-Y XX] [-r XX] [-b] [-bb] [-T] local-dir remote-path-names
```

其中各选项的主要说明如表 8-56 所示。

表 8-56 ncftpget 命令主要选项说明

选 项	说 明
-u XX	指定用户名
-p XX	指定密码
-P XX	使用指定的端口号，而不是默认 FTP 端口号
-j XX	使用指定的账号作为用户/密码的补充
-d XX	使用指定的文件作为调试登录
-a	使用 ASCII 码传输模式，而不是二进制
-t XX	指定时间后超时
-v/-V	使用进度条
-f XX	从指定的文件读入主机名、用户名、密码信息
-c	将结果输出到标准输出设置而不是写入文件
-A	追加到本地文件，而不是覆盖
-z/-Z	尝试继续传输
-E	使用常规（PORT）数据连接
-F	使用被动（PASV）数据连接
-DD	在成功下载后，删除远端源文件
-R	递归模式，复制整个目录树
-B XX	设置 TCP/IP socket 缓存大小
-W XX	在登录后发送原始 FTP 命令
-X XX	传输文件后发送原始 FTP 命令
-Y XX	在退出登录前发送原始 FTP 命令
-r XX	尝试登录的次数
-b	在后台运行
-bb	类似于-b 选项，但是仅提交批处理任务
-T	在下载整个目录树中，不使用自动 TAR 模式
local-dir	指定存放下载文件的本地目录
remote-path-names	指定远端路径名

例如，下载远端主机 192.168.20.122 上的文件 index.html 到本地根目录，执行命令如下所示。

```
# ncftpget -u teacher -p 654321 192.168.20.122 / index.html
//index.html:                               38.28 kB   35.17 MB/s
#
```

例如，下载远端主机 192.168.20.122 上的所有文件到本地目录/home 下，执行命令如下所示。

```
# ncftpget -u teacher -p 654321 192.168.20.122 /home *
/home/index.html:                             38.28 kB   71.21 MB/s
ncftpget index.html.1: server said: Failed to open file. //无法打开文件
/home/install.log:                             48.33 kB   51.36 MB/s
```



/home/install.log.syslog:	6.95 kB	19.61 MB/s
/home/mypaper.doc:	102.00 B	58.59 kB/s
/home/test.bak:	1.96 MB	92.63 MB/s

8.6.4 ncftpls 命令：显示文件目录

ncftpls 命令用来显示远端 ftp 服务器的文件列表。ncftpls 命令基于命令行，不用进入交互 Shell 方式即可方便地显示文件目录，因此非常适合在脚本中使用。该命令的语法格式如下所示。

```
ncftpls [-u XX][ -p XX][ -P XX][ -j XX][ -d XX][ -t XX][ -f XX][ -K][ -E][ -F][ -W XX][ -X XX][ -Y XX]
[-r XX] ftp://url.style/host/path/name/
```

其中各选项的主要说明如表 8-57 所示。

表 8-57 ncftpls 命令主要选项说明

选 项	说 明
-u XX	指定用户名
-p XX	指定密码
-P XX	使用指定的端口号，而不是默认 FTP 端口号
-j XX	使用指定的账号作为用户/密码的补充
-d XX	使用指定的文件作为调试登录
-t XX	指定时间后超时
-f XX	从指定的文件读入主机名、用户名、密码信息
-K	显示磁盘使用情况
-E	使用常规（PORT）数据连接
-F	使用被动（PASV）数据连接
-W XX	在登录后发送原始 FTP 命令
-X XX	传输文件后发送原始 FTP 命令
-Y XX	在退出登录前发送原始 FTP 命令
-r XX	尝试登录的次数
ftp://url.style/host/path/name/	指定 ftp 服务的地址、路径

例如，显示远端 ftp 服务器 192.168.20.122 的文件列表，命令如下所示。

```
# ncftpls -u teacher -p 654321 ftp://192.168.20.122 /显示文件列表
index.html
install.log
install.log.syslog
my.doc
myback
mybook
#
```

例如，以文件的长格式形式显示远端 ftp 服务器 192.168.20.122 的文件列表，命令如下所示。

```
# ncftpls -l -u teacher -p 654321 ftp://192.168.20.122
-rw-r--r-- 1 500 500 39199 Aug 10 15:54 index.html
```


-rw-r--r--	1	500	500	49492	Aug 10 15:55	install.log
-rw-r--r--	1	500	500	7113	Aug 10 15:55	install.log.syslog
-rw-r--r--	1	0	0	289	Aug 10 15:38	my.doc
-rw-r--r--	1	0	0	1005	Aug 10 15:38	myback
-rw-r--r--	1	0	0	103	Aug 10 15:38	mybook
drwxr-xr-x	2	500	500	4096	Aug 10 16:20	mybox

8.6.5 ncftpput 命令：上传文件

ncftpput 命令用来直接上传文件到远端 ftp 服务器。ncftp 命令基于命令行，不用进入交互 Shell 方式即可方便地显示文件目录，因此非常适合在脚本中使用。该命令的语法格式如下所示。

```
ncftpput [-u XX][ -p XX][ -P XX][ -j XX][ -d XX][ -a][ -t XX][ -v, -V][ -f
XX][ -c][ -A[ -z/-Z][ -E][ -F][ -DD][ -R][ -B XX][ -y]
[-W XX][ -W XX][ -X XX][ -Y XX][ -r XX][ -b][ -bb][ -T] remote-directory local-files
```

表 8-58 ncftpput 命令主要选项说明

选 项	说 明
-u XX	指定用户名
-p XX	指定密码
-P XX	使用指定的端口号，而不是默认 FTP 端口号
-j XX	使用指定的账号作为用户/密码的补充
-d XX	使用指定的文件作为调试登录
-a	使用 ASCII 码传输模式，而不是二进制
-t XX	指定时间后超时
-v/-V	使用进度条
-f XX	从指定的文件读入主机名、用户名、密码信息
-c	将结果输出到标准输出设置而不是写入文件
-A	追加到本地文件，而不是覆盖
-z/-Z	尝试继续传输
-E	使用常规（PORT）数据连接
-F	使用被动（PASV）数据连接
-DD	在成功下载后，删除远端源文件
-R	递归模式，复制整个目录树
-B XX	设置 TCP/IP socket 缓存大小
-W XX	在登录后发送原始 FTP 命令
-X XX	传输文件后发送原始 FTP 命令
-Y XX	在退出登录前发送原始 FTP 命令
-r XX	尝试登录的次数
-y	在远端主机上保留时间戳



续表

选 项	说 明
-b	在后台运行
-bb	类似于-b 选项，但是仅提交批处理任务
-T	在下载整个目录树中，不使用自动 TAR 模式
local-files	指定上传的文件
remote-directory	指定远端路径名

例如，将本地当前目录下的所有文件上传到远端服务器 192.168.20.122 的当前目录中，且用户名为 teacher，密码为“654321”，命令如下所示。

```
# ncftpput -u teacher -p 654321 192.168.20.122 ./ * //上传文件
anaconda-ks.cfg:          960.00 B    2.54 MB/s
dump.tmp:                 478.00 B    2.30 MB/s
index.html:               38.28 kB   68.59 MB/s
index.html.1:             38.28 kB  102.98 MB/s
install.log:              48.33 kB   71.51 MB/s
install.log.syslog:        6.95 kB   38.54 MB/s
minicom.log:              35.00 B   210.99 kB/s
#
# ls //显示当前目录
anaconda-ks.cfg index.html install.log minicom.log
dump.tmp index.html.1 install.log.syslog
```

8.6.6 tftp 命令：传输文件

tftp 命令是简单的文本模式的 ftp 命令，其语法格式如下所示。

```
tftp [-v][-V] [host]
```

其中各选项的主要说明如表 8-59 所示。

表 8-59 tftp 命令主要选项说明

选 项	说 明
-v	运行时，显示详细的处理信息
-V	显示 tftp 命令的版本信息
host	指定 ftp 服务器

在客户端访问 FTP 服务器时，如果没有在命令行给出服务器的主机名或 IP，则客户端将出现“tftp>”提示符，等待用户输入 tftp 内部命令。

通过在“tftp>”提示符下输入“?”，可以查看 tftp 常用内部命令。表 8-60 列出了常见的 tftp 命令及其说明。

表 8-60 常见 tftp 内部命令及其说明

选 项	说 明
connect	连接到远端 tftp 服务器
mode	文件传输模式
put	上传文件

续表

选 项	说 明
get	下载文件
quit	退出
verbose	显示详细的处理信息
trace	显示包路径
status	显示当前状态信息
binary	二进制传输模式
ascii	ASCII 传输模式
rexmt	设置包传输的超时时间
timeout	设置重传的超时时间
?	显示帮助信息
help	帮助信息

例如，显示 tftp 内部命令列表，并查询指定命令的帮助信息，命令如下所示。

```
# tftp
tftp> ? //显示 tftp 内部命令列表
tftp-hpa 0.32
Commands may be abbreviated. Commands are:

connect      connect to remote tftp
mode         set file transfer mode
put          send file
get          receive file
quit         exit tftp
verbose      toggle verbose mode
trace        toggle packet tracing
status       show current status
binary       set mode to octet
ascii        set mode to netascii
rexmt        set per-packet transmission timeout
timeout      set total retransmission timeout
?            print help information
help         print help information
tftp> help put //显示 put 命令的帮助信息
tftp-hpa 0.32
send file
tftp> help get //显示 get 命令的帮助信息
tftp-hpa 0.32
receive file
tftp>quite
#
```

例如，使用 tftp 命令连接到 ftp 服务器 192.168.20.122，并通过“status”命令查询连接状态，通过“binary”命令设置二进制传输模式，如下所示。

```
# tftp
tftp> connect
```



```
(to) 192.168.20.122
tftp> status //当前状态
Connected to 192.168.20.122.
Mode: netascii Verbose: off Tracing: off
Rexmt-interval: 5 seconds, Max-timeout: 25 seconds
tftp> binary //二进制模式
tftp> quit //退出
#
```

8.7 UUCP 常用命令

UUCP 系统是一组程序，可以完成系统之间的文件传输、命令执行、使用情况统计、系统安全保护等功能。

UUCP 是各种 UNIX 版本包括 Linux 都可用的标准网络系统，而且 UUCP 部署简便，成本低廉，只需要一根电缆连接两个系统就可建立 UUCP。如果要在相距数百或数千公里的两个系统间传输数据，只需要两个具有拨号功能的调制解调器即可。也正基于此，UUCP 是类 UNIX 系统中使用最广泛的网络实用系统。

UUCP 相关命令包括 uucico、uucp、uupick、uuto、uulog、uuname、uuc 及 uustat 等。

8.7.1 uucico 命令：UUCP 文件传输守护进程

uucico 命令用来处理 UUCP 以及 UUX 的文件传输任务，其语法格式如下所示。

```
uucico [-c, --quiet][ -D, --nodetach][ -f, --force][ -r0, --slave][ -r1, --master][ -i
type, --stdin type] [ --version][ --help]
[ -s system, --system system][ -S system][ -f, --force][ -p port, --port port][ -q,
--nouuxqt][ -I file, --config file]
[ -u name, --login name]
```

其中各选项的主要说明如表 8-61 所示。

表 8-61 uucico 命令主要选项说明

选 项	说 明
-c, --quiet	不将错误信息写入日志文件
-D, --nodetach	始终保持与终端的连接
-f, --force	强制执行，忽略任何请求等待
-r0, --slave	从属方式
-r1, --master	主动方式
-i type, --stdin type	指定连接端口类型
-s system, --system system	调用命名系统
-S system	调用命令系统，忽略任何请求等待
-f, --force	忽略任何请求等待和系统调用
-p port, --port port	指定调用端口或侦听端口
-q, --nouuxqt	不启动 uuxqt 守护进程

续表

选 项	说 明
-I file, --config file	指定配置文件
-u name, --login name	指定用户
--version	版本信息
--help	帮助信息

例如，启动 uucico，且运行方式为主动式，命令如下所示。

```
# uucico -rl //启动 uucico
```

8.7.2 uucp 命令：在 UNIX 系统间传送文件

uucp 命令用来在 UNIX 系统之间传送文件，其语法格式如下所示。

```
uucp [-c,--nocopy] [-C,-p,--copy] [-d,--directories] [f,--nodirectories] [-g,--grade
grade] [-m,--mail] [-t,--uuto]
[ -n,--notify user] [ -R,--recursive] [ -r,--nouucico] [ -s,--status
file] [-j,--jobid] [-W,--noexpand] [-I,--config file]
[ -u,--usage name] [ -x,--debug debug] [ -v,--version] [ --help]
[source-file] [ destination-file]
[ destination-directory]
```

其中各选项的主要说明如表 8-62 所示。

表 8-62 uucp 命令主要选项说明

选 项	说 明
-c,--nocopy	不将本地文件复制到 spool 目录
-C,-p,--copy	复制本地文件到 spool 目录
-d,--directories	创建所需的目录
-f,--nodirectories	不创建目录
-g,--grade grade	设置任务等级
-m,--mail	报告通过 mail 进行复制的状态信息
-n,--notify user	报告通过 mail 复制到远端用户的状态
-R,--recursive	递归复制目录
-r,--nouucico	不启动 uucico 守护进程
-s,--status file	报告文件完成状态
-j,--jobid	报告任务
-W,--noexpand	不将当前目录添加到远端文件名中
-t,--uuto	模拟 UUTO
-u,--usage name	设置用户名
-x,--debug debug	设置调试等级
-I,--config file	设置使用的配置文件
-v,--version	显示版本信息



续表

选 项	说 明
--help	显示帮助信息并退出
source-file	指定源文件
destination-file	指定目标文件
destination-directory	指定目标目录

例如，通过命令 `uucp`，将本机/home 目录下的文件，传送到主机 teacher 下 UUCP 公用目录下的 tmp 目录下，如下所示。

```
# uucp -d -R /home/ teacher!~/tmp //传送文件
```

8.7.3 uupick 命令：处理 UUCP 文件

uupick 命令用来处理 UUCP 传输进来的文件，其语法格式如下所示。

```
uupick [-s,--system system] [-x,--debug debug] [-I,--config file] [-v,--version] [--help]
```

其中各选项的主要说明如表 8-63 所示。

表 8-63 uupick 命令主要选项说明

选 项	说 明
-s,--system system	仅处理来自指定系统的文件
-x,--debug debug	设置调试等级
-I,--config file	指定配置文件
-v,--version	版本信息
--help	帮助信息

8.7.4 uuto 命令：文件传送到远端主机

uuto 命令是一个脚本命令，用于传送文件到远端的 UUCP 主机，其语法格式如下所示。

```
uuto [-c,--nocopy] [-C,-p,--copy] [-d,--directories] [f,--nodirectories] [-g,--grade grade] [-m,--mail] [-t,--uuto] [-n,--notify user] [-R,--recursive] [-r,--nouucico] [-s,--status file] [-j,--jobid] [-W,--noexpand] [-I,--config file] [-u,--usage name] [-x,--debug debug] [-v,--version] [--help] [file] [dest]
```

其中各选项的主要说明如表 8-64 所示。

表 8-64 uuto 命令主要选项说明

选 项	说 明
-c,--nocopy	不将本地文件复制到 spool 目录
-C,-p,--copy	复制本地文件到 spool 目录
-d,--directories	创建所需的目录
-f,--nodirectories	不创建目录
-g,--grade grade	设置任务等级

续表

选 项	说 明
-m,--mail	报告通过 mail 进行复制的状态信息
-n,--notify user	报告通过 mail 复制到远端用户的状态
-R,--recursive	递归复制目录
-r,--nouucico	不启动 uucico 守护进程
-s,--status file	报告文件完成状态
-j,--jobid	报告任务
-W,--noexpand	不将当前目录添加到远端文件名中
-t,--uuto	模拟 UUTO
-u,--usage name	设置用户名
-x,--debug debug	设置调试等级
-I,--config file	设置使用的配置文件
-v,--version	显示版本信息
--help	显示帮助信息并退出
file	指定文件
dest	指定远端主机

例如，通过命令 uuto，传送本地文件/home/teacher/mybook.doc 到远端主机 yang 下的 tmp 目录下，如下所示。

```
# uuto /home/teacher/mybook.doc yang!tmp //传送文件
```

8.7.5 uulog 命令：显示 uucp 日志信息

uulog 命令显示 UUCP 日志文件中的记录信息，其语法格式如下所示。

```
uulog [-n,--lines] [ -s,--system] [ -f system,--follow=system] [ -u,--user user] [ -F,--follow] [ -S,--statslog] [ -D,--debuglog] [ -X,--debug debug] [ -I,--config file] [ -v,--version] [ --help]
```

其中各选项的主要说明如表 8-65 所示。

表 8-65 uulog 命令主要选项说明

选 项	说 明
-n,--lines	显示从日志结尾开始的指定行数
-s,--system	显示命名系统的记录
-f system,--follow=system	为指定的命名系统添加记录
-u,--user user	显示命名用户的记录
-F,--follow	为所有系统添加记录
-S,--statslog	显示统计文件
-D,--debuglog	显示调试文件
-X,--debug debug	显示调试等级



续表

选 项	说 明
-I,--config file	设置配置文件
-v,--version	显示版本信息
--help	显示帮助信息

例如，通过命令 `uulog`，显示 `uucp` 日志信息，如下所示。

```
# uulog //显示 uucp 记录信息
uucico - - (2008-08-11 04:46:32.81 6391) Incoming call (login root port pts/1)
uucico - - (2008-08-11 04:48:32.87 6391) ERROR: Timeout
uucico - - (2008-08-11 05:17:11.19 6460) No work
```

8.7.6 uuname 命令：显示 uucp 远端主机

`uuname` 命令用于显示 UUCP 远端主机，其语法格式如下所示。

```
uuname [-a,--aliases][ -l,--local][ -I,--config file][ --help][ -v,--version]
```

其中各选项的主要说明如表 8-66 所示。

表 8-66 uuname 命令主要选项说明

选 项	说 明
-a,--aliases	别名显示
-l,--local	输出本机名称
-I,--config file	指定配置文件
--help	显示 <code>uuname</code> 命令帮助信息
-v,--version	显示 <code>uuname</code> 命令的版本信息

通过命令 `uuname`，显示 UUCP 远端主机名，如下所示。

```
# uuname //显示 uucp 远端主机名
```

8.7.7 uustat 命令：显示 UUCP 状态

`uustat` 命令可以显示当前的 UUCP 状态，还可以显示、删除或启动 UUCP 任务，其语法格式如下所示。

```
uustat [-a,--all][ B,--mail-lines num][ -c,--command command][ -C,--not-command
command][ -I,--config file]
[ -e,--executions][ -i,--prompt][ -k,--kill
job][ -K,--kill-all][ -m,--status][ -M,--mail][ -N,--notify][ --help]
[ -o,--older-than hours][ -p,--ps][ -q,--list][ -Q,--no-list][ -r,--rejuvenate
job][ -R,--rejuvenate-all]
[ -s,--system system][ -S,--not-system system][ -u,--user user][ -U,--not-user user]
[ -v,--version]
[ -W,--comment comment][ -y,--younger-than hours][ -x,--debug debug]
```

其中各选项的主要说明如表 8-67 所示。

表 8-67 uustat 命令主要选项说明

选 项	说 明
-a,--all	列出所有 UUCP 任务
-B,--mail-lines num	显示邮件信息的行数
-c,--command command	显示指定命令的请求列表
-C,--not-command command	显示非指定命令的请求列表
-e,--executions	显示队列执行情况，而不是任务请求
-i,--prompt	在删除每一个列出的任务时，进行提示
-k,--kill job	删除指定的 UUCP 任务
-K,--kill-all	删除每一个列出的任务
-m,--status	执行所有远端主机的状态
-M,--mail	通过电子邮件向 UUCP 管理者报告每一个列出的任务
-N,--notify	通过电子邮件向请求者报告每一个列出的任务
-o,--older-than hours	列出所有比指定时间旧的任务
-p,--ps	显示所有持有 UUCP 锁的进程状态
-q,--list	显示每一系统的任务数
-Q,--no-list	不列出任务
-r,--rejuvenate job	更新指定的 UUCP 任务
-R,--rejuvenate-all	更新每一个列出的任务
-s,--system system	列出指定系统的所有任务
-S,--not-system system	列出非指定系统的所有任务
-u,--user user	列出指定用户的所有任务
-U,--not-user user	列出非指定用户的所有任务
-W,--comment comment	在邮件信息中对包含文件进行注释
-y,--younger-than hours	列出所有比指定时间更新的任务
-x,--debug debug	设置调试等级
-I,--config file	设置配置文件
-v,--version	显示版本信息
--help	显示帮助信息

例如，通过命令 uustat，显示所有 UUCP 任务，如下所示。

```
# uustat -a //显示所有任务
```

8.7.8 uux 命令：通过 UUCP 在远端主机上执行命令

uux 命令通过 UUCP 在远端主机上执行命令，所请求执行的命令不会立即被执行，而是排入队列，由 uucico 守护进程最终完成命令的执行。该命令的语法格式如下所示。

```
uux [-, -p, --stdi][ -c, --nocopy][ -C, --copy][ -b][ -l, --link][ -j, --jobid][ -a address, --requestor address][ -I file, --config file][ --help][ -v, --version] command
```



其中各选项的主要说明如表 8-68 所示。

表 8-68 uux 命令主要选项说明

选 项	说 明
-, -p, --stdi	从标准输入读取命令
-c, --nocopy	不复制本地文件到 spool 目录
-C, --copy	复制本地文件到 spool 目录
-b	显示状态
-l, --link	链接本地文件到 spool 目录
-j, --jobid	显示任务 ID
-a address, --requestor address	向指定的邮件地址报告任务状态
-l file, --config file	指定配置文件
--help	帮助信息
-v, --version	版本信息
command	指定执行的命令

8.8

电子邮件

电子邮件服务是 Internet 上最基本的服务之一，用户可以很方便地读、写和转发信件，实现信息的传递与交流。与电话相比，电子邮件服务不要求通信双方实时在线，收件人可以根据实际需要随时上网从邮件服务器上收取电子邮件；与传统的邮政服务相比，电子邮件传输速度快，且费用很低。正是因为电子邮件快速、便捷而经济的特点，使其成为除 WWW 服务之外人们最常用的互联网应用。

一个完整的 E-mail 系统一般由三个部分组成，包括用户邮件代理、邮件服务器和邮件协议。

1. 用户邮件代理

用户邮件代理（Mail User Agent，简称 MUA）是用户与电子邮件系统的接口，负责用户和邮件服务器之间的交互工作。大多数情况下，MUA 就是运行在客户端上的应用程序，其作用是将邮件发送到邮件服务器上 and 从邮件服务器上接收邮件。

2. 邮件服务器

邮件服务器是电子邮件系统的核心，其主要功能是发送和接收邮件，并向发件人告知邮件的传送情况。根据其功能，邮件服务器分为邮件传输服务器（SMTP 服务器）和邮件接收服务器（POP3 或 IMAP4 服务器）。

3. 电子邮件协议

电子邮件协议包含一系列标准规范，是 TCP/IP 协议体系中的一部分，主要内容包括邮件格式、邮件的传输、邮件的接收等。

一个普通用户使用电子邮件服务过程的如下：用户 A 向用户 B 发送一封 E-mail，用户 B 收到该邮件并阅读邮件内容。在这个过程中，用户 A 和用户 B 各自拥有一个 E-mail 地址，

这个 E-mail 地址是用户向各自邮件服务器注册后得到的，地址的格式为 aaa@example.com 和 bbb@test.com。用户 A 把按照电子邮件格式协议编写好的邮件发往用户 B，发送时该邮件将从用户 A 的主机被发送到用户 A 的邮件服务器 example.com，再由 example.com 将邮件转发至用户 B 所在的邮件服务器 test.com。test.com 收到该邮件后，将其保存下来。用户 B 接收邮件时，利用客户端登陆 test.com 邮件服务器，下载并浏览邮件。

在这个过程中，用户从 MUA 发送的邮件，是通过邮件传输代理（Mail Transport Agent，简称 MTA）进行转发的。MTA 根据邮件接收方地址中的域名在网络上查找 DNS 服务器，选择最佳的路径进行传输。有时候需要通过多个 MTA 的接力，才能够将邮件正确地传输到目的服务器。

前面提到，邮件服务器一般可分为邮件传输服务器（SMTP 服务器）和邮件接收服务器（POP3 或 IMAP4 服务器）。SMTP 服务器具有接收其他 SMTP 服务器发送的邮件和发送邮件到其他 SMTP 服务器的功能，因此所有的 SMTP 服务器都可以被称为 MTA。

目的邮件服务器接收到邮件后，如果判断邮件就是本服务器用户的邮件，则把邮件投递到用户的邮箱中。当用户访问服务器的时候就会发现有新邮件到达。这个邮件投递的服务称为邮件投递代理（Mail Delivery Agent，简称 MDA），主要负责把邮件放入用户邮箱。

8.8.1 fetchmail 命令：收取邮件

fetchmail 命令用于从 POP、IMAP、ETRN 或 ODMR-capable 收取邮件。该命令从远端邮件服务器获取邮件并将之发送到本地客户端。然后，可以使用通常的邮件用户代理，如 mutt、elm 或 Mail 处理收到的邮件。可以设置 fetchmail 以守护进程方式运行，周期性地收取邮件。该命令的语法格式如下所示。

```
fetchmail [-?, --help] [-V, --version] [-c, --check] [-s, --silent] [-v, --verbose] [-d,
--daemon] [-q, --quit] [-m, --mda]
    [-L, --logfile] [--syslog] [--invisible] [-f, --fetchmailrc] [-r, --folder]
[--showdots] [--bsmtp] [--lmtpl]
    [-i, --idfile] [--postmaster] [--nobounce] [--ssl] [--sslkey] [--sslcrt] [--sslpr
oto] [--plugin] [--plugout]
    [-p, --protocol] [-U, --uidl] [-P, --port] [--auth] [-t, --timeout] [-E, --envelope] [-Q,
--qvirtual] [--tracepolls]
    [-u, --username] [-a, --all] [-K, --nokeep] [-k, --keep] [-F, --flush] [-n,
--norewrite] [-l, --limit]
    [-w, --warnings] [-S, --smtp host] [--fetchdomains] [-D, --smtp address] [--smtp name] [-Z,
--antispam]
    [-b, --batchlimit] [-B, --fetchlimit] [-e, --expunge]
```

其中各选项的主要说明如表 8-69 所示。

表 8-69 fetchmail 命令主要选项说明

选 项	说 明
-?, --help	显示帮助信息
-V, --version	显示版本信息
-c, --check	不取信，仅检查信息
-s, --silent	静默模式



续表

选 项	说 明
-v, --verbose	冗余模式，输出调试信息
-d, --daemon	以守护进程方式运行
-q, --quit	杀死守护进程
-L, --logfile	指定日志文件
--syslog	当以守护进程方式运行时，使用 syslog 记录所有信息
--invisible	不写入收到的信件，使用主机缓存
-f, --fetchmailrc	指定替换的运行控制文件
-i, --idfile	指定替换 UID 文件
--postmaster	指定上次使用的收件箱
--nobounce	重新定向用户与邮箱
--ssl	使用 SSL 加密会话
--sslkey	SSL 私钥文件
--sslcert	SSL 客户端认证
--sslproto	强制 SSL 协议（ssl2/ssl3/tls1）
--plugin	指定建立连接的外部命令
--plugout	指定打开 SMTP 连接的外部命令
-p, --protocol	指定收信协议
-U, --uidl	强制使用 UIDLs（仅适用于 pop3）
-P, --port	连接 TCP/IP 端口
--auth	认证类型
-t, --timeout	服务器超时
-E, --envelope	信封地址头
-Q, --qvirtual	从本地用户 ID 中删除前缀
--tracepolls	对收到的邮件头，添加追踪信息
-u, --username	指定服务器登录用户
-a, --all	收取全部信息
-K, --nokeep	在收取邮件之后，删除新邮件
-k, --keep	在收取邮件之后，保存新邮件
-F, --flush	从服务器上删除旧信息
-n, --norewrite	不重写头地址
-l, --limit	超出指定大小，不收取
-w, --warnings	指定发出警告通知的时间间隔
-S, --smtp host	设置 SMTP 转发主机
--fetchdomains	收取指定域的邮件

续表

选 项	说 明
-D, --smtpaddress	设置使用 SMTP 的投递域
--smtpname	设置 SMTP 全名 username@domain
-Z, --antispam,	设置反垃圾邮箱的响应值
-b, --batchlimit	设置 SMTP 连接的批处理限制
-B, --fetchlimit	设置连接服务器收取信件的限制
-e, --expunge	设置最大删除
-m, --mda	设置 MDA
--bsmtp	设置输出的 BSMTP 文件
--lmtmp	使用 LMTP (RFC2033) 进行投递
-r, --folder	指定远端目录名
--showdots	在日志文件中显示进度, 用 “.” 表示

例如，通过命令 `fetchmail` 到邮件服务器 `mail.bit.edu.cn` 下载邮件。通过选项 “-p” 指定传输协议为 POP3，通过选项 “-u” 指定用户名为 `sullivan`。执行命令如下所示。

[illegible]

例如，通过命令 `fetchmail` 到邮件服务器 `mail.bit.edu.cn` 下载邮件，通过选项 “-p” 指定传输协议为 POP3，通过选项 “-u” 指定用户名为 `sullivan`，通过选项 “-a” 设定下载时不区分邮件新旧全部下载，通过选项 “-k” 设定在邮件服务器上保留原邮件。命令执行过程如下所示。



```
# fetchmail -ak -p POP3 -u sullivan mail.bit.edu.cn //下载邮件
Enter password for sullivan@mail.bit.edu.cn:
183 messages for 06145891 at mail.bjtu.edu.cn (30069496 octets).
reading message sullivan@mail.bit.edu.cn:1 of 183 (2268 octets) .. not flushed
reading message sullivan@mail.bit.edu.cn:2 of 183 (2949 octets) .. not flushed
reading message sullivan@mail.bit.edu.cn:3 of 183 (12573 octets) ..... not
flushed
reading message sullivan@mail.bit.edu.cn:4 of 183 (7550 octets) ..... not flushed
reading message sullivan@mail.bit.edu.cn:5 of 183 (25220 octets) .....
not flushed
reading message sullivan@mail.bit.edu.cn:6 of 183 (9214 octets) ..... not flushed
reading message sullivan@mail.bit.edu.cn:7 of 183 (3118 octets) ... not flushed
reading message sullivan@mail.bit.edu.cn:8 of 183 (2889 octets) .. not flushed
reading message sullivan@mail.bit.edu.cn:9 of 183 (76784 octets) .....
..... not flushed
reading message sullivan@mail.bit.edu.cn:10 of 183 (1350 octets) . not flushed
reading message sullivan@mail.bit.edu.cn:11 of 183 (1827 octets) . not flushed
reading      message      sullivan@mail.bit.edu.cn:12      of      183      (215465
octets) .....
.....
reading message sullivan@mail.bit.edu.cn:98 of 12 (6568 octets) .fetchmail: terminated
with signal 2
You have new mail in /var/spool/mail/root
#
```

8.8.2 mail 命令：收发邮件

mail 命令是个邮件管理工具，可以收取或发送电子邮件，其语法格式如下所示。

```
mail [-i] [-I] [-iv] [-n] [-N] [-l] [-s] [-c] [-b] [-f] [-u]
```

其中各选项的主要说明如表 8-70 所示。

表 8-70 mail 命令主要选项说明

选 项	说 明
-i	忽略中断信号
-I	强制 mail 运行于交互模式
-v	冗余模式，运行时显示详细的处理信息
-n	启动时禁止读取文件 /etc/mail.rc
-N	不显示邮件标题
-l	交互模式
-s	指定邮件主题
-c	指定副本邮箱地址
-b	指定暗抄邮箱地址
-f	读取指定邮箱
-u	指定用户

在命令行直接输入 mail 命令，将显示当前用户的邮件信息并进入交互模式。在出现 “\$” 提示符后，等待用户输入 mail 内部命令。mail 的常用内部命令如表 8-71 所示。

表 8-71 mail 的内部命令及说明

选 项	说 明
数字	选择对应的邮件
+	下一封
-	上一封
h	显示标题
q	退出 mail 程序，退出前删除标记的邮件
x	退出 mail 程序
?	帮助信息
p<邮件号>	读取指定邮件号邮件
d<邮件号>	删除邮件
u<邮件号>	恢复邮件
R<邮件号>	回复邮件
r<邮件号>	回复邮件，既寄给发件人，也寄给其他收件人
m<邮箱地址>	发邮件到指定邮箱

例如，将邮件发送到 sullivan@bit.edu.cn，邮件主题为“Hello”，内容为“Hello ,I am Sullivan. Welcome to China.”，将副本发送到 Kelly@bit.edu.cn，执行命令如下所示。

```
# mail teacher@localhost //发送邮件
Subject: Hello //输入邮件主题
Hello ,I am Sullivan. Welcome to China. //输入邮件内容
Cc: Kelly@bit.edu.cn //输入抄送地址
#
```

例如，查看当前用户的新邮件列表，并查看第 11 封、第 12 封邮件内容，命令如下所示。

```
#mail //当前用户的邮件信息
Mail version 8.1 6/6/93. Type ? for help.
"/var/spool/mail/root": 17 messages 17 new //未读邮件标题列表
>N 1 root@localhost.local Thu Aug 7 10:28 19/838 "Anacron job 'cron.daily'"
N 2 root@localhost.local Fri Aug 8 04:02 24/3705 "LogWatch for localhost.localdomain"
N 3 root@localhost.local Fri Aug 8 04:07 24/1018 "Cron <root@localhost> run-parts
/etc/cr"
N 4 root@localhost.local Sat Aug 9 04:02 24/1663 "LogWatch for localhost.localdomain"
N 5 root@localhost.local Sat Aug 9 04:04 24/1018 "Cron <root@localhost> run-parts
/etc/cr"
N 6 MAILER-DAEMON@localh Sun Aug 10 09:06 73/2704 "Warning: could not send message
for pas"
N 7 MAILER-DAEMON@localh Sun Aug 10 09:06 102/3268 "Warning: could not send message
for pas"
N 8 MAILER-DAEMON@localh Sun Aug 10 09:06 69/2534 "Warning: could not send message
for pas"
N 9 root@localhost.local Sun Aug 10 09:06 52/1584 "LogWatch for localhost.localdomain"
N 10 root@localhost.local Sun Aug 10 09:06 24/1021 "Cron <root@localhost> run-parts
/etc/cr"
N 11 MAILER-DAEMON@localh Sun Aug 10 09:06 69/2533 "Warning: could not send message
```



```
for pas"
  N 12 root@localhost.local Sun Aug 10 09:06 20/851 "new station"
  N 13 root@localhost.local Sun Aug 10 09:07 20/852 "new station"
  N 14 uucp@localhost.local Sun Aug 10 21:34 22/835 "Execution failed"
  N 15 uucp@localhost.local Sun Aug 10 21:34 23/889 "UUCP execution files saved after
failur"
  N 16 root@localhost.local Mon Aug 11 04:02 105/2974 "LogWatch for
localhost.localdomain"
  N 17 root@localhost.local Mon Aug 11 04:04 24/1021 "Cron <root@localhost> run-parts
/etc/cr"
& 11 //选择查看第 11 封邮件
Message 11:
From root@localhost.localdomain Sun Aug 10 09:07:00 2008
Date: Sat, 9 Aug 2008 21:50:23 +0800
From: root@localhost.localdomain (Arpwatch)
To: root@localhost.localdomain
Subject: new station

    hostname: <unknown>
    ip address: 192.168.20.122
    ethernet address: 0:c:29:1b:ea:cc
    ethernet vendor: <unknown>
    timestamp: Saturday, August 9, 2008 21:50:23 +0800
& + //查看下一封邮件
Message 12:
From uucp@localhost.localdomain Sun Aug 10 21:34:10 2008
Date: Sun, 10 Aug 2008 21:34:10 +0800
From: uucp <uucp@localhost.localdomain>
To: root@localhost.localdomain
Subject: Execution failed

Message from UUCP on localhost Sun Aug 10 21:34:10 2008

Your execution request failed because you are not permitted to execute
teacher
on this system.
Execution requested was:
teacher ls
& - //返回上一封邮件
Message 11:
From root@localhost.localdomain Sun Aug 10 09:07:00 2008
Date: Sat, 9 Aug 2008 21:50:23 +0800
From: root@localhost.localdomain (Arpwatch)
To: root@localhost.localdomain
Subject: new station

    hostname: <unknown>
    ip address: 192.168.20.122
    ethernet address: 0:c:29:1b:ea:cc
    ethernet vendor: <unknown>
    timestamp: Saturday, August 9, 2008 21:50:23 +0800
& q //退出 mail
Held 17 messages in /var/spool/mail/root
#
```


例如, 查看当前用户的新邮件列表, 并删除第 11 封信件, 回复第 12 封邮件, 命令如下所示。

```
# mail
Mail version 8.1 6/6/93. Type ? for help.
"/var/spool/mail/root": 13 messages 13 unread           //显示未读邮件列表
>U 1 root@localhost.local Thu Aug 7 10:28 20/848 "Anacron job 'cron.daily'"
  U 2 root@localhost.local Sat Aug 9 04:02 46/1673 "LogWatch for localhost.localdomain"
  U 3 root@localhost.local Sat Aug 9 04:04 25/1028 "Cron <root@localhost> run-parts
/etc/cr"
  U 4 MAILER-DAEMON@localh Sun Aug 10 09:06 74/2714 "Warning: could not send message
for pas"
  U 5 MAILER-DAEMON@localh Sun Aug 10 09:06 103/3278 "Warning: could not send message
for pas"
  U 6 MAILER-DAEMON@localh Sun Aug 10 09:06 70/2544 "Warning: could not send message
for pas"
  U 7 root@localhost.local Sun Aug 10 09:06 53/1594 "LogWatch for
localhost.localdomain"
  U 8 root@localhost.local Sun Aug 10 09:06 25/1031 "Cron <root@localhost> run-parts
/etc/cr"
  U 9 MAILER-DAEMON@localh Sun Aug 10 09:06 70/2543 "Warning: could not send message
for pas"
  U 10 root@localhost.local Sun Aug 10 09:06 21/861 "new station"
  U 11 uucp@localhost.local Sun Aug 10 21:34 24/899 "UUCP execution files saved after
failur"
  U 12 root@localhost.local Mon Aug 11 04:02 106/2984 "LogWatch for localhost.localdomain"
  U 13 root@localhost.local Mon Aug 11 04:04 25/1031 "Cron <root@localhost> run-parts
/etc/cr"
& d 11           //删除第 11 封邮件
& h           //显示未读邮件列表
  U 1 root@localhost.local Thu Aug 7 10:28 20/848 "Anacron job 'cron.daily'"
  U 2 root@localhost.local Sat Aug 9 04:02 46/1673 "LogWatch for localhost.localdomain"
  U 3 root@localhost.local Sat Aug 9 04:04 25/1028 "Cron <root@localhost> run-parts
/etc/cr"
  U 4 MAILER-DAEMON@localh Sun Aug 10 09:06 74/2714 "Warning: could not send message
for pas"
  U 5 MAILER-DAEMON@localh Sun Aug 10 09:06 103/3278 "Warning: could not send message
for pas"
  U 6 MAILER-DAEMON@localh Sun Aug 10 09:06 70/2544 "Warning: could not send message
for pas"
  U 7 root@localhost.local Sun Aug 10 09:06 53/1594 "LogWatch for localhost.localdomain"
  U 8 root@localhost.local Sun Aug 10 09:06 25/1031 "Cron <root@localhost> run-parts
/etc/cr"
  U 9 MAILER-DAEMON@localh Sun Aug 10 09:06 70/2543 "Warning: could not send message
for pas"
  U 10 root@localhost.local Sun Aug 10 09:06 21/861 "new station"
  >U 12 root@localhost.local Mon Aug 11 04:02 106/2984 "LogWatch for
localhost.localdomain"
  U 13 root@localhost.local Mon Aug 11 04:04 25/1031 "Cron <root@localhost> run-parts
/etc/cr"
& r 12           //回复第 12 封邮件
To: root@localhost.localdomain root@localhost.localdomain
Subject: Re: LogWatch for localhost.localdomain

Thanks!
Cc: Kelly@bit.edu.cn           //抄送给 Kelly
```



```
&q
#
```

8.8.3 mailq 命令：显示发件箱的邮件

mailq 命令用来显示发件箱中的邮件，等同于“sendmail -b”。mailq 成功执行时返回值为 0，返回值大于 0 时，表示出现错误。该命令的语法格式如下所示。

```
mailq [-Ac][-v]
```

其中各选项的主要说明如表 8-72 所示。

表 8-72 mailq 命令主要选项说明

选 项	说 明
-Ac	显示在/etc/mail/submit.cf 文件中的邮件队列，而不是/etc/mail/sendmail.cf 中定义的邮件队列
-v	冗余模式，显示详细信息

例如，通过命令 mailq ，显示待发送邮件详细信息，如下所示。

```
# mailq -v //显示待发送的邮件
150d 867 1HAVys-0005CP-Ti <root@localhost>
      root@localhost

143d 867 1HDTJi-0001Uc-Sl <root@localhost>
      root@localhost

120d 867 1HLoHc-0001DN-F6 <root@localhost>
      root@localhost

118d 867 1HMXEQ-0001Yp-Pn <root@localhost>
      root@localhost
#
```

8.8.4 mutt 命令：Mutt 用户邮件代理

mutt 命令是一个基于文本的小巧且功能强大的邮件管理器，可以读取邮件，支持彩色终端、MIME。该命令的语法格式如下所示。

```
mutt [-a file][ -b address][ -c address][ -e command][ -f mailbox][ -F muttrc][ -h][ -H
draft][ -i include][ -n][ -p][ -R]
[ -m type [ -s subject][ -v][ -x][ -y][ -z][ -Z]
```

其中各选项的主要说明如表 8-73 所示。

表 8-73 mutt 命令主要选项说明

选 项	说 明
-a file	使用 MIME，附加文件
-b address	指定暗抄地址
-c address	指定抄送地址
-e command	指定配置命令
-f mailbox	指定加载的邮箱
-F muttrc	指定初始化文件，默认为 ~/.muttrc
-h	显示帮助信息

续表

选 项	说 明
-H draft	指定包含标题和内容的草稿文件
-i include	指定包含到邮件体中的文件
-m type	指定默认邮件类型
-n	忽略系统配置文件
-p	继续被延迟的邮件
-R	以只读模式打开邮件
-s subject	指定邮件的标题
-v	显示版本信息
-x	模拟邮箱组合方式
-y	通过邮箱命令，在启动 mutt 时显示指定的邮件列表
-z	与选项“-f”联合使用，如果邮箱中没有邮件，不启动 mutt
-Z	在启动 mutt 时显示包含新邮件的邮箱，如果没有新邮件，不启动 mutt

例如，在命令行执行 mutt 命令，进入如图 8-17 所示界面。



图 8-17 显示邮件列表

用 UP/DOWN 方向键选择要读取的邮件。例如，选择第 2 封邮件然后按回车键，查看第 2 封邮件的内容，如图 8-18 所示。

输入“?”显示帮助信息，如图 8-19 所示。



```
i: 离开 -: 上一页 <Space>: 下一页 v: 显示附件. d: 删除 r: 回复 j: 下一个 ? : 帮助
Date: Sat, 8 Aug 2008 04:02:11 +0800
From: root <root@localhost.localdomain>
To: root@localhost.localdomain
Subject: LogWatch for localhost.localdomain

##### LogWatch 4.3.1 (01/13/03) #####
Processing Initiated: Sat Aug 9 04:02:07 2008
Date Range Processed: yesterday
Detail Level of Output: 0
Logfiles for Host: localhost.localdomain
#####

----- Kernel Begin -----
WARNING: Kernel Errors Present
end_request: I/O error, dev 02:00 (floppy), sector 0
----- Kernel End -----

----- pam_unix Begin -----
su:
Sessions Opened:
(uid=0) -> news: 1 Time(s)
----- pam_unix End -----

Aug 8 07:25:29 end_request: I/O error, dev 02:00 (floppy), sector 0
Aug 8 07:26:17 end_request: I/O error, dev 02:00 (floppy), sector 0
Aug 8 07:26:34 end_request: I/O error, dev 02:00 (floppy), sector 0

##### LogWatch End #####

-o P- 2/13: root LogWatch for localhost.localdomain -- (all)
```

图 8-18 查看邮件

```
i: 离开 -: 上一页 <Space>: 下一页 ? : 帮助
^B M jurlview\n call urlview to extract URLs out of a message
^D delete-thread 删除所有在序列中的信件
^E edit-type edit attachment content type
^F forget-passphrase 消除记忆体中的 PGP 通行密码
<Tab> next-new 跳到下一封新的信件
^K extract-keys 提取 PGP 公共钥匙
<Return> display-message 显示一封信件
^N next-thread 跳到下一个序列
^P previous-thread 跳到上一个序列
^R read-thread 标记现在的序列为已读取
^T untag-pattern 反标记符合某个格式的信件
^U undelete-thread 反删除这个序列中的所有信件
<Esc><Tab> previous-new 跳到上一个新的信件
<Esc>C decode-copy 制作可解码的 (text/plain) 拷贝
<Esc>P check-traditional-pgp check for classic pgp
<Esc>V collapse-all 打开/关闭 所有的序列
<Esc>b M /-b search in message bodies
<Esc>c change-folder-readonly 用只读模式打开另一个文件夹
<Esc>d delete-subthread 删除所有在子序列中的信件
<Esc>e resend-message 用这封信件作为新信件的模板
<Esc>k mail-key 寄出 PGP 公共钥匙
<Esc>l show-limit 显示目前有作用的限制样式
<Esc>n next-subthread 跳到下一个子序列
<Esc>p previous-subthread 跳到上一个子序列
<Esc>r read-subthread 标记现在的子序列为已读取
<Esc>s decode-save 制作可解码的拷贝 (text/plain) 并且删除之
<Esc>t tag-thread 标记目前的序列
<Esc>u undelete-subthread 反删除这个子序列中的所有信件
<Esc>v collapse-thread 打开/关闭 目前的序列
<Space> display-message 显示一封信件
$ sync-mailbox 储存变动到信箱
^ toggle-write 切换是否重新写入邮箱中
@ display-address 显示寄信人的完整位址
C copy-message 拷贝一封信件到某个文件或信箱
D delete-pattern 删除符合某个格式的信件
F flag-message 切换某封信件的 '重要' 旗标
G fetch-mail 取回 POP 伺服器上的信件
J next-entry 移动到下一项资料
K previous-entry 移到上一项资料
L list-reply 回覆给某一个指定的邮件列表
N toggle-new 切换将讯息切换为 'new' 旗标
Index 的求助 (33%)
```

图 8-19 显示 mutt 帮助信息

8.8.5 pine 命令：收发邮件

pine 命令用来收发邮件，它具有交互式界面，操作比较方便，可以同时给多个人发送邮件。pine 支持 MIME，可以访问远端或本地的邮箱。该命令的语法格式如下所示。

```
pine[-conf][-i][-o][-r][-z][-attachfile][-attachlistfile-list][-f folder][-h][-sort order][ -url url][ -v] [ -p config-file]
```

其中各选项的主要说明如表 8-74 所示。

表 8-74 pine 命令主要选项说明

选 项	说 明
-conf	产生配置文件，并输出到标准输出
-i	启动时直接进入收件箱
-o	只读模式
-r	使用限制或演示模式
-z	使用 Ctrl+z 组合键可挂起 pine
-attach file	粘贴附件
address	发送邮件地址
-attachlist file-list	粘贴附件列表
-f folder	打开指定的目录
-h	帮助信息
-p config-file	指定个人配置文件
-sort order	指定收件箱中文件的排序方式，可选方式包括时间、主题、大小、发件地址、收件地址等，默认按到达时间排序
-url url	打开指定的 URL
-v	版本信息

例如，在命令行下直接输入 pine，将进入图 8-20 所示的 pine 主界面。

在图 8-20 所示界面中，主窗口显示菜单，下面是快捷方式。按 UP/DOWN 方向键选择【FOLDER LIST】，将进入文件夹选择界面，如图 8-21 所示。

图 8-21 所示的文件夹选择界面包括三个选项：【INBOX】、【sent-mail】和【saved-message】。选择【INBOX】，进入图 8-22 所示的收件箱界面。

在图 8-22 所示的收件箱界面中，可以看到收件箱中的所有邮件列表。利用方向键选择需要查看的邮件，直接按回车键，可以打开所选择的邮件，如图 8-23 所示。

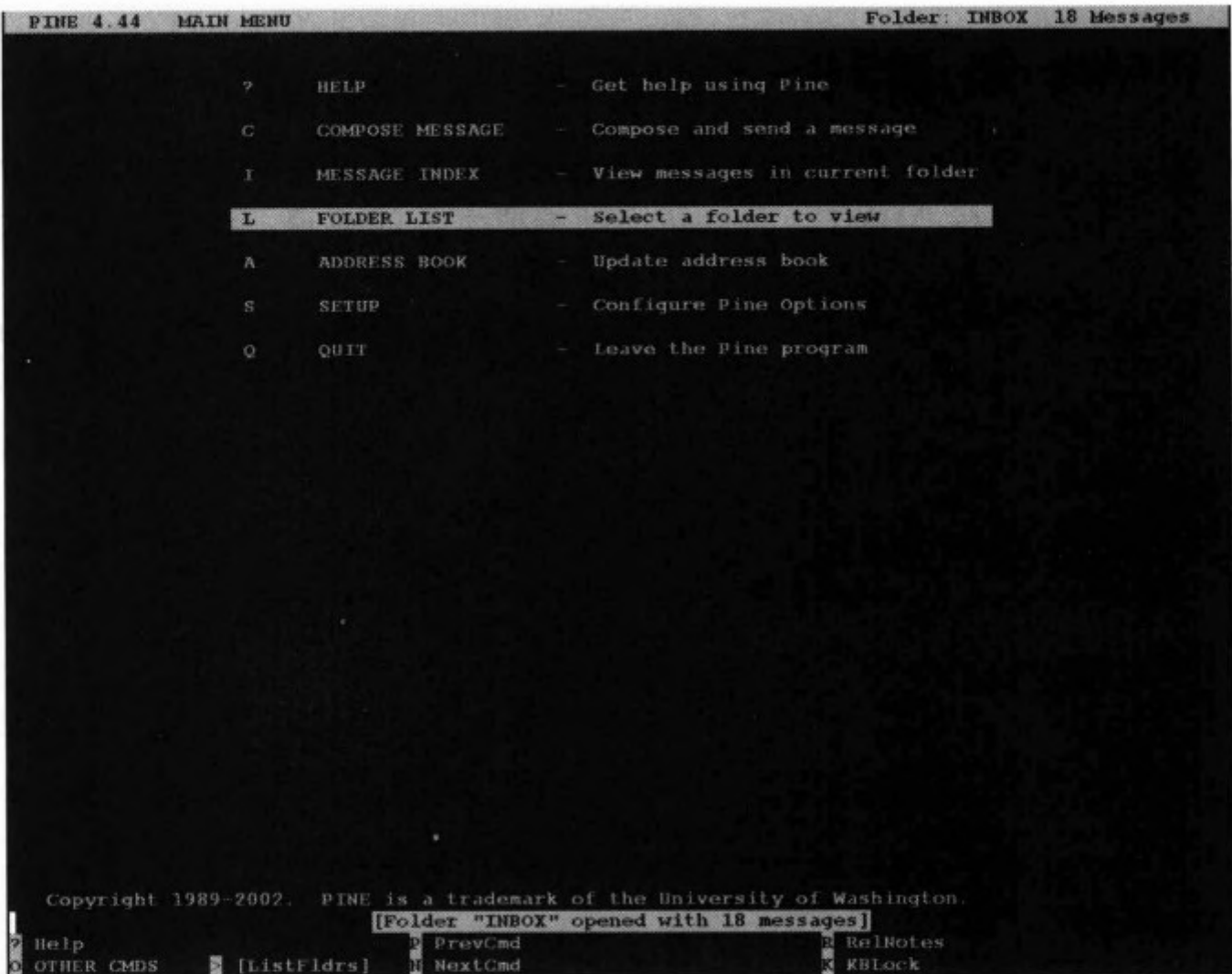


图 8-20 pine 的主界面



图 8-21 文件夹选择界面

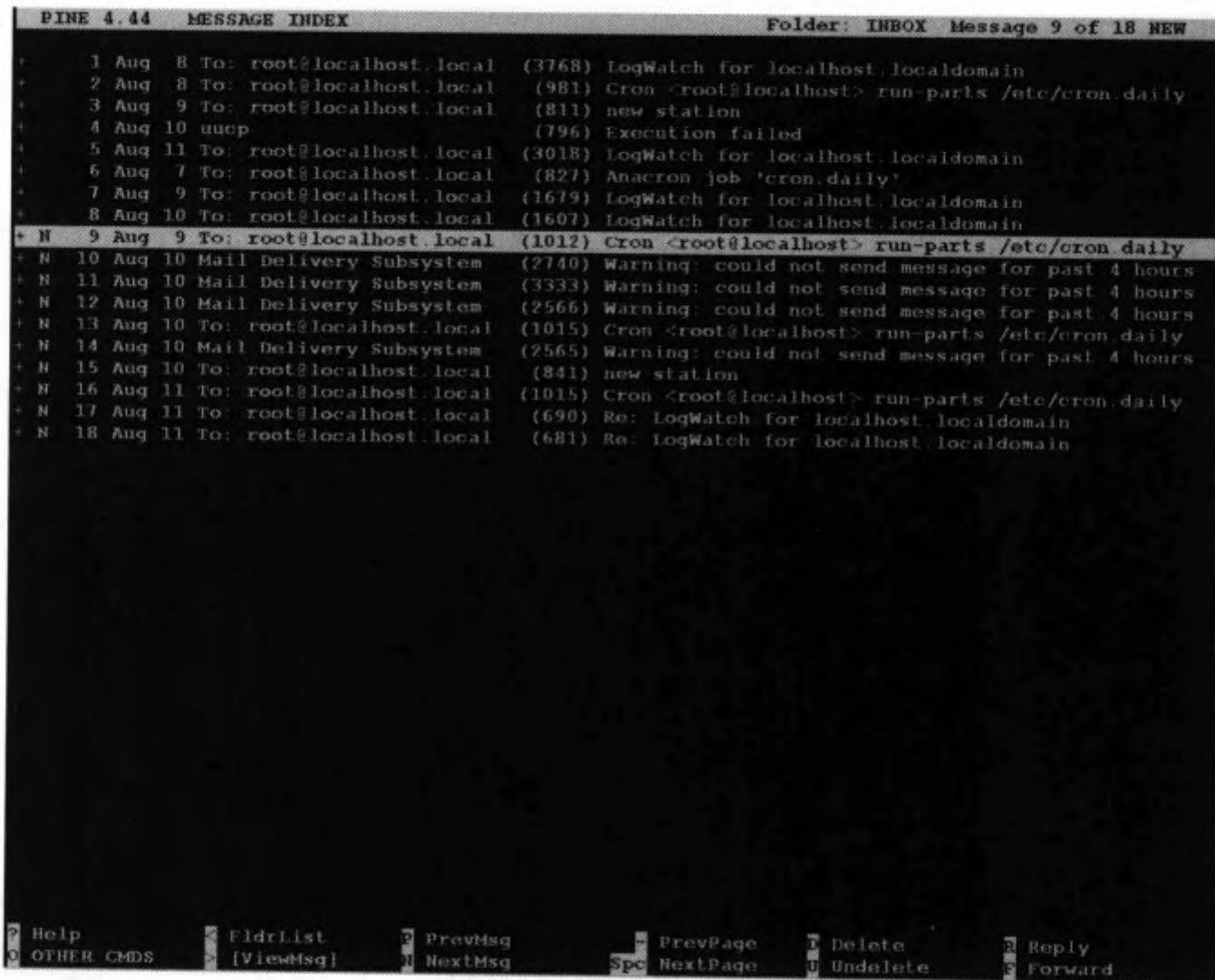


图 8-22 收件箱邮件列表

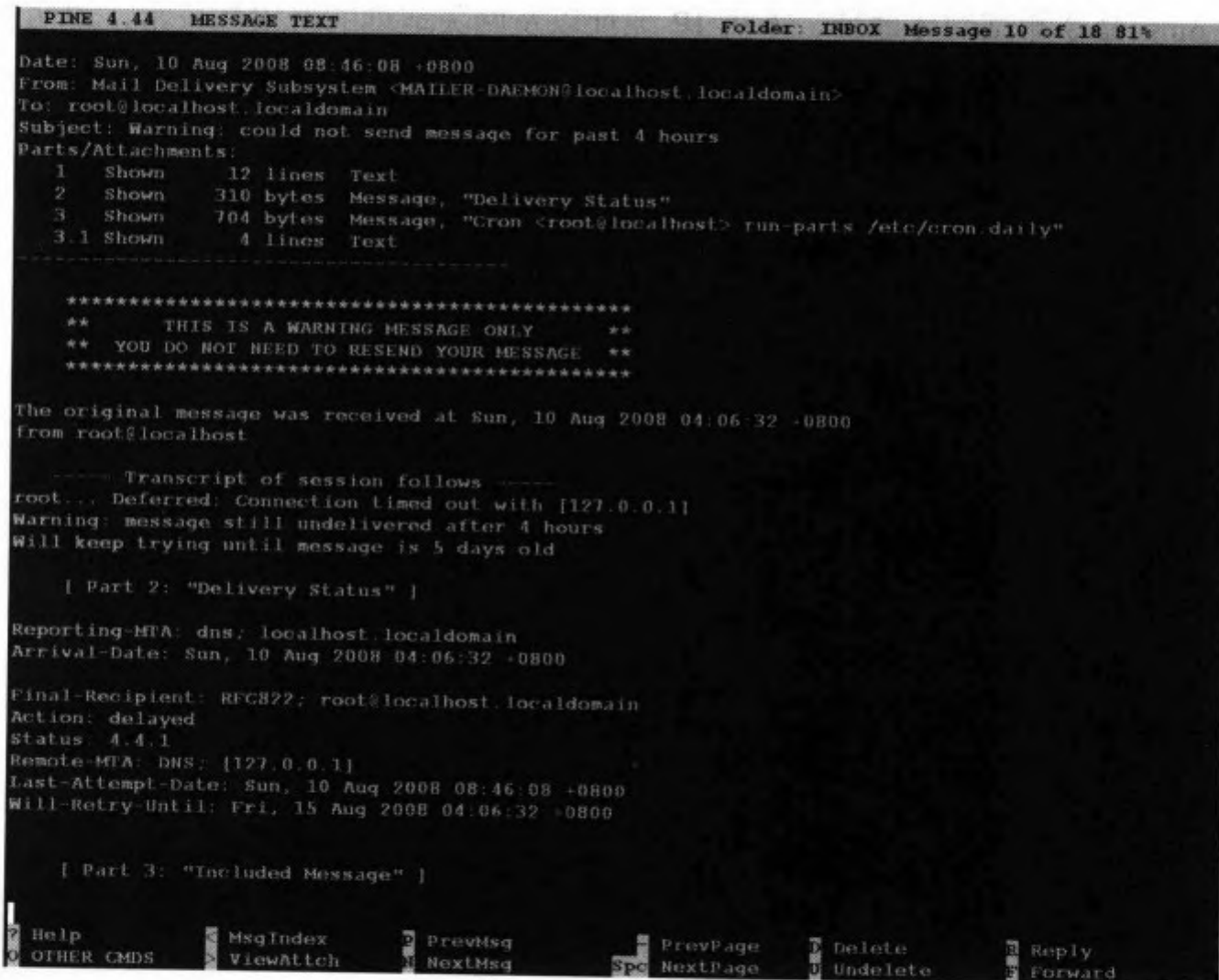


图 8-23 阅读邮箱中的邮件



8.9 利用常用命令分析局域网连通故障

当网络不通时，联合使用 `ping`、`netstat`、`nslookup` 及 `traceroute` 命令可以进行故障的分析和诊断。

- 使用 `ping` 命令测试回环地址、本机 IP 地址和网关地址。

使用“`ping 127.0.0.1`”命令和“`ping` 本机 IP 地址”命令检查本机的 TCP/IP 协议是否正确，网卡是否正常工作。

如果 `ping` 回环地址正确，说明 TCP/IP 协议没有问题，否则需要重新安装 TCP/IP 协议。

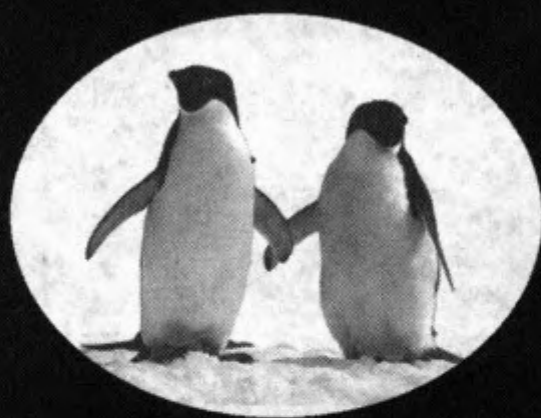
如果 `ping` 本机的 IP 地址结果正确，说明网卡配置正确，否则需对网卡的软硬件进行检查。

检查硬件，检查网卡和与之相连的交换机上的指示灯是否都亮，如果有一边不亮，说明在其相应那边的设备可能有问题，也可能是网线有问题。

若硬件无故障，则检测网卡驱动程序是否安装正确。用户需要先确认其所安装的系统内核是否支持该网卡。Red Hat Linux 9 和最近两年出品的大多数硬件是兼容的，用户可以到 Red Hat 的官方网站上 (<https://hardware.redhat.com/>) 查找最新支持的硬件列表进行兼容性确认。如果不兼容，则上网查找相应的驱动程序更新或补丁。如果系统内核支持该网卡，用户需要对驱动程序的设置做进一步的检查。通常可以先卸载驱动程序，然后再尝试重新安装。

- 使用 `netstat`、`nslookup`、`traceroute` 命令检查路由、DNS 的设置。

如果 `ping` 命令测试回环地址、本机 IP 地址以及网关地址的结果都正常，说明故障出现在网络层之上。接下来，联合使用 `netstat`、`nslookup`、`traceroute` 命令分别检查路由、DNS 设置是否正确。



LINUX

中篇 编辑器

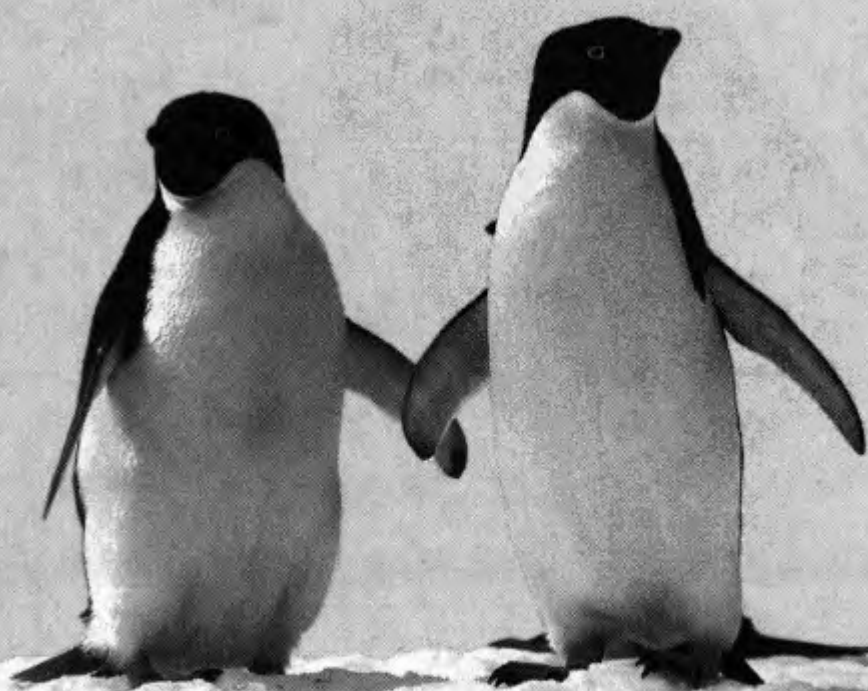
本篇详细讲述了 Linux 下常用的编辑器，包括 vi、emacs、grep、gawk、sed 等，同时介绍了在编辑器中广泛使用的正则表达式。对一些专门用途的编辑命令，如 aspell、join、sort、more 等，也做了细致的讲解。

LINUX

第9章

正则表达式

正则表达式 (regular expression) 适合完成一些复杂的模式匹配。在 Linux 中, 一些强大的实用程序, 如 vi、grep、sed 和 awk 等, 都提供了对正则表达式的支持。



9.1 正则表达式介绍

正则表达式可以追溯到对人类神经系统的早期研究阶段。当时的神经生理学家 Warren McCulloch 和 Walter Pitts 曾研究出用一种以数学方式来描述神经网络的方法。1956 年一位叫 Stephen Kleene 的数学家在 McCulloch 和 Pitts 研究成果的基础上，发表了一篇标题为“神经网络事件的表示法”的论文，正式引入了正则表达式的概念。

正则表达式是一种依赖于自身，专门为文本的模式匹配而设计的一种语言。尽管正则表达式比较复杂，但是其语法规则却非常灵活和强大。事实上，正则表达式只是字符串模式，提供了一种描述一段文本模式的方法，可以利用正则表达式方便地检查一个字符串是否含有某个子字符串，将匹配的子字符串进行替换，或者从某个字符串中取出符合某个条件的子字符串等。

正则表达式在数据有效性验证方面也有着重要的作用。使用正则表达式对数据的有效性进行验证，可以检验所接收到的任何数据是否合法。

9.2 元字符

正则表达式与一般的数学表达式十分相似，元字符相当于数学表达式中定义的运算符。

9.2.1 元字符的定义

简单地讲，一个正则表达式由正则字符（如字母、逗号、数字等）和元字符（如星号、圆括号等）构成，其中元字符与正则字符联合构成了一个文本模式的程序性描述。因此，正则表达式允许对文本内的模式和子模式进行匹配。

最简单的正则表达式是只包括一个字符的表达式，如 `w`，它可以与字符串 `where`、`power`、`how` 等匹配（其中都包含字母“`w`”）。可以将几个字母组合成为更长的表达式，如 `wh`，它可以与 `what`、`who`、`where` 等字符串匹配。

表 9-1 中列出了在 `vi`、`grep`、`sed`、`awk` 中都可以使用的正则表达式元字符。

表 9-1 正则表达式中常用的元字符及说明

元 字 符	符 号 名 称	说 明
\	反斜杠	转义字符，例如“\t”将匹配一个制表符 Tab，而不匹配字母“t”
	管道	表示“或”，即匹配其中任何一个，例如，“book desk”将匹配“book”或“desk”
[]	方括号	表示匹配指定范围内的其中一个字符，例如，“[abc]”将匹配包含 a、b 和 c 中任意一个字符
[^]	方括号+插入符	匹配不在指定组内的字符，例如，“[^abc]”表示匹配不包含字母 a、b 和 c 的任意字符



续表

元 字 符	符 号 名 称	说 明
()	小括号	可以将正则字符和元字符或表达式进行组合, 例如, “(book desk)s” 将匹配 “books” 或 “desks”
{ }	大括号	为前导表达式指定一个最小或最大匹配的数目, 例如, “a{3,4}” 将匹配 “aaa” 和 “aaaa”
^	插入符	用于指定匹配字符串的头部, 也称行首定位符。例如, “^hello” 将匹配 “hello world” 和 “hello Mr. yang” 等开头为 “hello” 的字符串。若 “^” 出现在中括号内开始位置, 则表示 “非” 或 “排除”, 例如, “[^a-z]” 表示匹配除小写字母以外的任意一个字符
\$	美元符	用于指定匹配字符串的尾部, 也称行尾定位符。例如, “world\$” 将匹配 “hello world” 和 “PHP world” 等结尾为 “world” 的字符串
*	星号	匹配 0 个或多个前导表达式, 例如, “a*” 将匹配 0~n 个字符 “a” 组成的字符串
?	问号	匹配 0 个或 1 个前导表达式, 例如, “a?” 将匹配 “a” 或 “aa”
\<	反斜杠+小于号	词首定位符, 例如, “\<abc” 表示所有包含以 abc 开头的单词的行
\>	反斜杠+大于号	词尾定位符, 例如, “\>abc” 表示所有包含以 abc 结尾的单词的行
.	点号	匹配任意一个字符
-	减号	用于指明字符范围, 例如, “[a-c]” 将匹配包含 a、b 和 c 中任意一个字符



正则表达式中的元字符不会被识别为正常字符。如果希望将元字符在正则表达式中当作正常字符使用, 必须使用一个反斜杠来转义。例如, 要表示一个普通字符 “\$”, 则在正则表达式中应写为 “\\$, 要表示一个普通字符 “.”, 在正则表达式中应写为 “\.”。

9.2.2 元字符的应用示例

例如, 在 vi (vim) 中打开 example.txt 文件。查找文件中的字符串 “self”, 在 vi (vim) 末行模式下输入以下命令。

```
:/self/
```

回车后, vi (vim) 开始搜索, 搜索结果如图 9-1 所示。



图 9-1 使用正则表达式查找指定的字符串

例如，在末行模式下，查找文件 `example.txt` 中出现的所有字符串“MSN”，并将其替换为“mobile sensor network”，在末行模式下输入以下命令。

```
:1,$ s/MSN/mobile sensor network/g
```

回车后，vi (vim) 开始搜索并替换，共替换了 8 次，结果如图 9-2 所示。



```
1.Mobile sensor network (mobile sensor network) is composed of a group of mobile nodes equipp
ed with wireless
devices. To assure the sensor performance, mobile sensor network should have self-deploying and
self-healing
capability. In this paper a center-attraction-field based self-deployment algorithm (CAFSA)
is proposed to solve the network partition and coverage holes which aroused by traditional
deployment methods based on artificial-potential-field. Simulation results show that CAFSA
exhibit excellent performance and enforce the network coverage.
2.Mobile sensor network (mobile sensor network) is composed of a group of mobile nodes equipp
ed with wireless
devices. To assure the sensor performance, mobile sensor network should have self-deploying and
self-healing
capability. In this paper a center-attraction-field based self-deployment algorithm (CAFSA)
is proposed to solve the network partition and coverage holes which aroused by traditional
deployment methods based on artificial-potential-field. Simulation results show that CAFSA
exhibit excellent performance and enforce the network coverage.
3.Mobile sensor network (mobile sensor network) is composed of a group of mobile nodes equipp
ed with wireless
devices. To assure the sensor performance, mobile sensor network should have self-deploying and
self-healing
capability. In this paper a center-attraction-field based self-deployment algorithm (CAFSA)
is proposed to solve the network partition and coverage holes which aroused by traditional
deployment methods based on artificial-potential-field. Simulation results show that CAFSA
exhibit excellent performance and enforce the network coverage.
4.Mobile sensor network (mobile sensor network) is composed of a group of mobile nodes equipp
ed with wireless
devices. To assure the sensor performance, mobile sensor network should have self-deploying and
self-healing
capability. In this paper a center-attraction-field based self-deployment algorithm (CAFSA)
is proposed to solve the network partition and coverage holes which aroused by traditional
deployment methods based on artificial-potential-field. Simulation results show that CAFSA
exhibit excellent performance and enforce the network coverage.
```

第 8 行 20,2 全部

图 9-2 使用正则表达式执行替换操作

例如，查找文件 `example.txt` 中所有包含以“center”开头的字符串的行，其中用到了元字符“*”，命令如下所示。

```
# grep 'center*' example.txt
capability. In this paper a center-attraction-field based self-deployment algorithm
(CAFSA)
capability. In this paper a center-attraction-field based self-deployment algorithm
(CAFSA)
capability. In this paper a center-attraction-field based self-deployment algorithm
(CAFSA)
capability. In this paper a center-attraction-field based self-deployment algorithm
(CAFSA)
```

例如，查找 `example.txt` 文件中所有包含单词“performance”的行，在正则表达式中用到了元字符“\<”和“\>”，命令如下所示。

```
# grep '\<performance\>' example.txt
devices. To assure the sensor performance, MSN should have self-deploying and
self-healing
exhibit excellent performance and enforce the network coverage.
devices. To assure the sensor performance, MSN should have self-deploying and
self-healing
exhibit excellent performance and enforce the network coverage.
devices. To assure the sensor performance, MSN should have self-deploying and
self-healing
exhibit excellent performance and enforce the network coverage.
```



devices. To assure the sensor performance, MSN should have self-deploying and self-healing
exhibit excellent performance and enforce the network coverage.

9.3 字符集和类

字符集和类都具有集合概念，用于表示一个取值范围。通常系统会对一些特殊的字符集预先进行定义，称其为字符类。

9.3.1 字符集和类的定义

字符集表示一个字符范围，与精确匹配相比，其功能更为强大。字符集可以匹配属于指定类型的任何字符。事实上，字符集表达的是一种通配关系。

例如，要匹配一个元音字符，需要建立一个包含所有元音字符的字符集，正则表达式可以写为：

```
[AEIOUaeiou]
```

用连字符可以表示一个字符范围，如下所示。

- [a-zA-Z]：表示匹配任意一个字母。
- [A-Z]：表示匹配任意一个大写字母。
- [a-z]：表示匹配任意一个小写字母。
- [0-9]：表示匹配任意一个数字。
- [0-9*\+]：表示匹配数字、星号、加号中的任意一个。

例如，要匹配一个由一个大写字母和一位数字组成的字符串，如“A1”、“B1”或“D3”，可以建立如下正则表达式。

```
^[A-Z] [0-9]$
```

如果在中括号内使用符号“^”，则表示“取非”。以下是符号“^”的应用示例。

- [^A-Z]：表示匹配任意一个除大写字母之外的字符。
- [^a-z]：表示匹配任意一个除小写字母之外的字符。
- [^0-9]：表示匹配任意一个除数字之外的字符。
- [^0-9*\+]：表示匹配除数字、星号和加号之外的任意一个字符。

为了方便编程，系统会提供一些预定义的字符集，通常称之为字符类。字符类会对指定的字符范围进行定义，标准的字符类如表 9-2 所示。

表 9-2 正则表达式中内置的字符类

字 符 类	说 明
[:alpha:]	小写及大写字母，等价于[a-zA-Z]
[:alnum:]	小写和大写字母以及数字，等价于[a-zA-Z0-9]
[:cntrl:]	控制字符，如制表符（Tab）、换行符或反斜线
[:digit:]	数字 0～9，等价于[0-9]

续表

字 符 类	说 明
[:graph:]	ASCII 码在 33~126 之间的可打印字符
[:lower:]	小写字母，等价于[a-z]
[:punct:]	标点符号，包括 “?”、“.”、“,”、“:”、“;”、“!”、“~”、“!”、“@”、“#”、“\$”、“%”、“^”、“&”、“*”、“(”、“)”、“[”、“]”、“{”、“}”、“/”、“-”、“_”
[:upper:]	大写字母，等价于[A-Z]
[:space:]	空白字符，包括空格、水平制表符、垂直制表符、换行、换页、回车
[:xdigit:]	十六进制字符，等价于[a-fA-F0-9]



中括号仅表示匹配指定字符集中的一个字符。

9.3.2 元字集和类的应用示例

例如，查找包含“数字+句号”的行，在正则表达式中用到了字符集[0-9]来代表数字，命令如下所示。

```
# grep '[0-9]\.' example.txt
1.Mobile sensor network (MSN) is composed of a group of mobile nodes equipped with wireless
2.Mobile sensor network (MSN) is composed of a group of mobile nodes equipped with wireless
3.Mobile sensor network (MSN) is composed of a group of mobile nodes equipped with wireless
4.Mobile sensor network (MSN) is composed of a group of mobile nodes equipped with wireless
```

例如，查找包含“数字+句号”的行，在正则表达式中用到了数字类[:digit:]代表所有数字，命令如下所示。

```
# grep '[:digit:]\.' example.txt
1.Mobile sensor network (MSN) is composed of a group of mobile nodes equipped with wireless
2.Mobile sensor network (MSN) is composed of a group of mobile nodes equipped with wireless
3.Mobile sensor network (MSN) is composed of a group of mobile nodes equipped with wireless
4.Mobile sensor network (MSN) is composed of a group of mobile nodes equipped with wireless
```

9.4 重复

在很多情况下，某些字符或字符集会不止一次地出现，可以在正则表达式中使用指定的符号来表示这种重复。



9.4.1 重复的定义

重复有多种表示方式，不同的表示方式由不同的符号所指向，这些符号如下所示。

- 符号 “*”: 表示匹配 0 个或多个前导表达式。
- 符号 “?”: 表示匹配 0 个或 1 个前导表达式。
- 符号 “+”: 表示匹配 1 个或多个前导表达式。
- 大括号 “{}”: 表示为前导表达式指定一个最小或最大匹配的数目。

这些符号都跟在一个前导表达式的后面，其中，大括号 “{}” 有以下三种用法。

- {x,y}: 表示前导表达式至少出现 x 次，但不超过 y 次。
- {x,}: 表示前导表达式出现 x 次或更多次。
- {x}: 表示前导表达式只出现 x 次。

以下是一些重复的典型示例。

(1) 例如，以下正则表达式表示可以匹配任意一个或多个字母组成的字符串。

```
[[:alpha:]]+
```

该正则表达式可以匹配 “Welcome to Beijing”、“bed” 等字符串。

(2) 例如，以下正则表达式表示可以匹配任意 0 个或多个字母组成的字符串。

```
[[:alpha:]]*
```

该正则表达式可以匹配 “beep”、“b” 等字符串以及空字符串。

(3) 例如，以下正则表达式表示任意 0 个或一个字母组成的字符串。

```
[[:alpha:]]?
```

该正则表达式可以匹配 “a”、“b” 等字符串以及空字符串。

(4) 例如，以下正则表达式表示可以匹配任意 2 个字母组成的字符串。

```
[[:alpha:]]{2}
```

该正则表达式可以匹配 “ac”、“bd” 等字符串。

(5) 例如，以下正则表达式表示可以匹配任意 2 个或 3 个字母组成的字符串。

```
[[:alpha:]]{2,3}
```

该正则表达式可以匹配 “ab”、“bee” 等字符串。

(6) 例如，以下正则表达式表示可以匹配任意 2 个、3 个或 4 个字母组成的字符串。

```
[[:alpha:]]{2,4}
```

该正则表达式可以匹配 “ab”、“bee”、“abcde” 等字符串。

(7) 例如，以下正则表达式表示可以匹配任意 2 个或多个字母组成的字符串。

```
[[:alpha:]]{2,}
```

该正则表达式可以匹配 “ab”、“bee”、“abcde”、“ABCDEFG” 等字符串。

9.4.2 重复的应用示例

例如，在 example.txt 文件中，查找存在连续 11 个小写字母（如 performance）组成的单词的行，其正则表达式用到了 “{x}”，命令如下所示。

```
# grep '[a-z]\{11\}' example.txt
devices. To assure the sensor performance, MSN should have self-deploying and
self-healing
is proposed to solve the network partition and coverage holes which aroused by
```



```

traditional
  exhibit excellent performance and enforce the network coverage.
  devices. To assure the sensor performance, MSN should have self-deploying and
self-healing
  is proposed to solve the network partition and coverage holes which aroused by
traditional
  exhibit excellent performance and enforce the network coverage.
  devices. To assure the sensor performance, MSN should have self-deploying and
self-healing
  is proposed to solve the network partition and coverage holes which aroused by
traditional
  exhibit excellent performance and enforce the network coverage.
  devices. To assure the sensor performance, MSN should have self-deploying and
self-healing
  is proposed to solve the network partition and coverage holes which aroused by
traditional
  exhibit excellent performance and enforce the network coverage.

```

可以看到，搜索结果中每一行都包含了一个 11 个字母组成的单词 “performance”。

9.5 子表达式

与算术运算中使用小括号类似，在正则表达式中也可以使用小括号将字母、数字、元字符等进行组合后，再与其他字母、数字、元字符或其他组合联合使用，组成更长的表达式。

9.5.1 子表达式的定义

在正则表达式中，可以使用小括号将一个长的表达式分隔成几个子表达式或将字符、元字符、子表达式组合在一起。以下是一些典型示例。

(1) 例如，以下正则表达式将匹配包含 0 个或多个单词 “very” 的字符串。

```
(very) * good
```

该正则表达式可以匹配 “good”、“very good”、“very very good”、“very very very good” 等字符串。

(2) 例如，以下正则表达式表示可以匹配任意 2 个、3 个或 4 个单词 “very” 组成的字符串。

```
(very){2,4}high
```

该正则表达式可以匹配字符串 “very very high”、“very very very high”、“very very very very high”。

(3) 例如，以下的正则表达式可以匹配 “wy”、“wz”、“xy” 或 “xz”。

```
([wx])([yz])
```

9.5.2 子表达式的应用示例

例如，在 example.txt 文件中查找包含字符串 “center-” 的行，在正则表达式中将 “center-”



作为一个子表达式看待，命令如下所示。

```
# egrep '(center-)' example.txt
capability. In this paper a center-attraction-field based self-deployment algorithm
(CAFSA)
capability. In this paper a center-attraction-field based self-deployment algorithm
(CAFSA)
capability. In this paper a center-attraction-field based self-deployment algorithm
(CAFSA)
capability. In this paper a center-attraction-field based self-deployment algorithm
(CAFSA)
```

9.6 定位字符串的开始和末尾

使用正则表达式中的“^”和“\$”符号可以确定一个指定的字符序列是否出现在字符串的开始、结尾，或者在两个位置都出现。

9.6.1 开始和末尾的定义

插入符号“^”用于匹配字符串的头部，表示指定的字符序列必须出现在被匹配字符串的开始位置；美元符号“\$”用于匹配字符串的尾部，表示指定的字符序列必须出现在被匹配字符串的结束位置。以下是一些典型示例。

(1) 例如，以下正则表达式是在字符串开始处匹配“Welcome”。

```
^Welcome
```

该正则表达式可以匹配“Welcome to Beijing”、“Welcome to Harbin”等字符串。

(2) 例如，以下正则表达式是在字符串结束处匹配“est”。

```
est$
```

该正则表达式可以匹配“best”、“slowest”、“pest”等字符串。

(3) 例如，以下正则表达式匹配所有 3 个字母的字符串。

```
^[[:alpha:]]{3}$
```

该正则表达式可以匹配“pet”、“let”、“bed”等字符串。

(4) 例如，以下正则表达式匹配由 2 个以上的字母、数字或下划线组成的字符串。

```
^[A-Za-z0-9_]{2,}$
```

该正则表达式可以匹配“be”、“3def”、“df4d_e”等字符串。

(5) 例如，以下正则表达式匹配字符串“aaaa”。

```
^a{4}$
```

(6) 例如，以下正则表达式匹配字符串“aa”、“aaa”或“aaaa”。

```
^a{2,4}$
```

(7) 例如，以下正则表达式匹配字符串“a”、“aa”或“aaa”。

```
^a{1,3}$
```

(8) 例如，以下正则表达式匹配字符串“aa”、“aaa”或“aaaa”等起始位置多于 2 个 a 的字符串。

```
^a{2,}$
```


(9) 例如，以下正则表达式匹配包含两个以上字母 a 的字符串。

```
a{2, }
```

该正则表达式可以匹配 “baae”、“ef2aaa”、“e4aaaa5d” 等字符串。

(10) 例如，以下正则表达式匹配两个制表符。

```
\t{2}
```

(11) 例如，以下正则表达式匹配所有两个以上的字符。

```
.{2, }
```

该正则表达式可以匹配 “er”、“456”、“yuf4” 等字符串。

(12) 例如，以下正则表达式匹配所有的正整数。

```
^[0-9]{1, }$
```

该正则表达式可以匹配 “34”、“67”、“0” 等字符串。

(13) 例如，以下正则表达式匹配所有的整数，其中 “-” 表示负号。

```
^\-{0,1}[0-9]{1, }$
```

该正则表达式可以匹配 “-134”、“67”、“0” 等字符串。由于元字符 “?” 与 {0,1} 等价，都表示 “匹配 0 个或 1 个前导表达式”，所以该正则表达式也可以改写为：

```
^\-?[0-9]{1, }$
```

(14) 例如，以下正则表达式匹配所有的小数，其中 “-” 表示负号，“.” 表示小数点。

```
^\-{0,1}[0-9]{1, }\.[0,1][0-9]{0, }$
```

该正则表达式可以匹配 “-13.4”、“6.7”、“0.9” 等字符串。其中元字符 “?” 与 {0,1} 等价，都表示 “匹配 0 个或 1 个前导表达式”，所以该正则表达式也可以改写为：

```
^\-?[0-9]{1, }\.[0,1][0-9]{0, }$
```

9.6.2 开始和末尾的应用示例

例如，在 example.txt 文件中查找所有以字符串 “CAFSA” 结尾的行，在正则表达式中用到了元字符 “\$”，命令如下所示。

```
# grep 'CAFSA$' example.txt
deployment methods based on artificial-potential-field. Simulation results show that
CAFSA
deployment methods based on artificial-potential-field. Simulation results show that
CAFSA
deployment methods based on artificial-potential-field. Simulation results show that
CAFSA
deployment methods based on artificial-potential-field. Simulation results show that CAFSA
```

9.7 分支

在正则表达式中，使用符号 “|” 表示一个分支。分支具有 “或” 的意义，表示可以在多个选择中任选一个或多个。

9.7.1 分支的定义

分支的表示符号为 “|”，通常称为管道。分支表示可以匹配管道左右任何一个。例如，“a|b”



等价于 “[ab]”。以下是一些典型示例。

(1) 例如，以下正则表达式将匹配包含 “edu”、“mil” 或 “gov” 的字符串。

```
edu|mil|gov
```

该正则表达式可以匹配 “bit.edu.cn”、“ep.mil” 和 “guangxi.gov” 等字符串。

(2) 例如，以下正则表达式将匹配以 “er” 或 “est” 结尾的字符串。

```
(er|est)$
```

该正则表达式可以匹配 “higher”、“best” 和 “slowest” 等字符串。

(3) 例如，以下正则表达式将匹配字符串 “goodness” 或 “goodful”。

```
^good(ness|ful)$
```

(4) 例如，以下正则表达式将匹配字符串 “a”、“b” 或 “c” 中的任意一个。

```
[ab]|c
```

该正则表达式等价于 [abc]。

(5) 例如，以下正则表达式将匹配包含单词 “pucket” 或 “bucket” 的字符串。

```
(p|b)ucket
```

该正则表达式可以匹配 “bucket”、“puckets” 和 “buckets” 等字符串。

(6) 例如，以下的正则表达式将匹配包含单词 “warning” 或以 “ing” 结尾的字符串。

```
warning|ing$
```

该正则表达式可以匹配 “watching”、“warning again”、“beginning” 等字符串。

9.7.2 分支的应用示例

例如，在 example.txt 文件中查找包含数字 1、2、3 或 4 的行，其正则表达式中使用了分支符号 “|”，命令如下所示。

```
# egrep '1|2|3|4' example.txt
1.Mobile sensor network (MSN) is composed of a group of mobile nodes equipped with wireless
2.Mobile sensor network (MSN) is composed of a group of mobile nodes equipped with wireless
3.Mobile sensor network (MSN) is composed of a group of mobile nodes equipped with wireless
4.Mobile sensor network (MSN) is composed of a group of mobile nodes equipped with wireless
```

可以看到，搜索结果显示的行中均包含数字 1、2、3 或 4。

9.8 匹配特殊字符

正则表达式中的特殊字符，如 “.”、“\$” 等，是不能当做普通字符进行匹配的。要匹配这些特殊字符必须使用反斜杠 “\” 进行转义。例如，要匹配一个 “\$”，则在正则表达式中应表示为 “\\$”；要匹配一个 “.”，则在正则表达式中应表示为 “\.”；而如果要匹配一个反斜杠，则必须用两个反斜杠 “\\” 来表示。

正则表达式中需要转义的特殊字符包括：“^”、“\$”、“(”、“)”、“.”、“{”、“}”、“[”、“]”、“*”、“?”、“+”、“\”、“-”、“|”。

当正则表达式出现在字符串中时，该字符串最好使用单引号字符串表示法（即用单引号

将正则表达式包括起来)，如果使用双引号字符串表示法（即用双引号将正则表达式包括起来），在一些特殊应用环境中将带来一些复杂性。

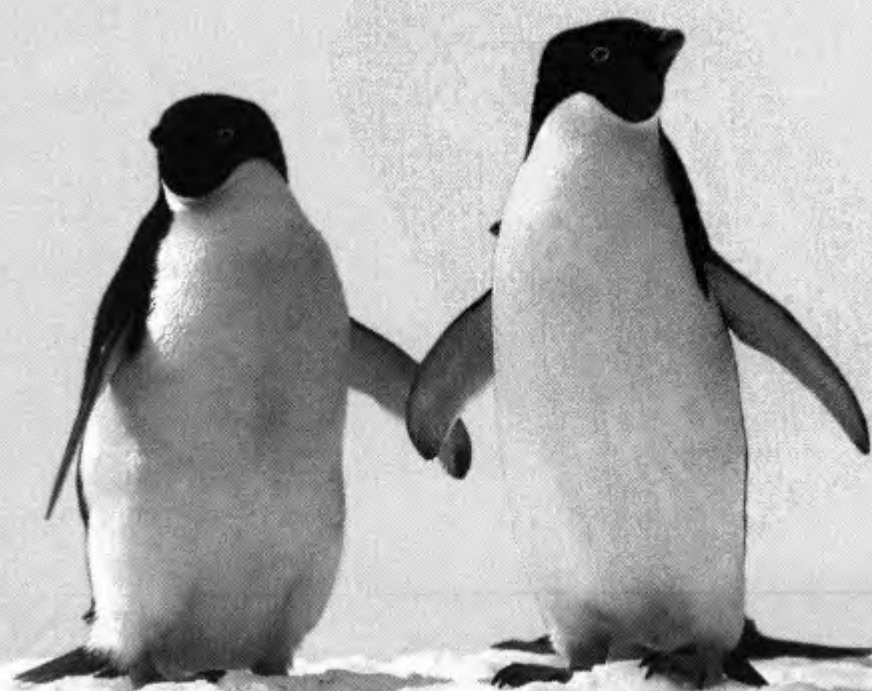
例如，在 PHP 中，如果在双引号字符串表示的正则表达式中希望包含一个反斜杠字符，则该反斜杠需要表示为 4 个反斜杠“\\”才能正确解析为一个普通的反斜杠字符。因为 PHP 在对双引号字符串进行解析时，首先会将该双引号字符串中的 4 个反斜杠解析为 2 个，然后再按正则表达式的规则进行解析，将 2 个反斜杠解析为 1 个反斜杠。对于美元符号“\$”也是如此。如果希望匹配一个“\$”，在 PHP 双引号字符串表示的正则表达式中需要写为“\\\$”。

LINUX

第10章

vi (vim) 编辑器

文本编辑器是 Linux 操作系统中的重要工具。其中，vi 是使用最多的文本编辑器，可以在任何 Shell 中使用。vi 是“visual editor（可视编辑器）”的缩写，发音为/vee eye/。可以说，vi 在类 UNIX 编程界有着广泛的影响。



10.1 vi (vim) 编辑器的历史

编辑器也称文本编辑器，是用于文本处理的程序。与字处理程序相比（如 Open Office.orgWrite），编辑器不具有加粗、居中、斜体等排版功能，所以可以快速高效地创建和编辑文本。

vi 并不是最早的文本编辑器，最早的文本编辑器是 ed。由于 ed 只是一种行编辑器，即一次只能书写或修改一程序，还不是一个屏幕文本编辑器，因而并没有得到大多数程序员的认可。

vi 是一种屏幕编辑器，即一次可以看到整个屏幕的文本，可以在屏幕中随意移动光标并选择文本的某一部分，也可以向上或向下移动数据视图，以浏览文本的不同部分。当启动 vi 编辑器后，vi 将指定的文件内容复制到称为缓冲区的内存空间中以供用户编辑。如果没有指定打开的文件，则创建一个空的缓冲区。



缓冲区只是文件在内存中的临时存放处，启动 vi 的时候，vi 将要编辑的文件从磁盘读入缓冲区。在会话过程中，vi 将所做的修改保存在缓冲区中而不同时修改磁盘上的文件，直到将缓冲区内容写回到磁盘。在结束会话即退出 vi 时，缓冲区将被清除，因此在退出 vi 前如果不保存文件，对文件所做的修改将不复存在，所以在修改文件后一定要注意存盘后再退出。

vi 在 1976 年随 BSD 软件发布，现在在一些现代的 4.4BSD 系统，如 BSD/OS、Free BSD、NetBSD 等中都可以找到 vi。vi 还有几个具有扩展功能的版本，比较知名的有 vim (www.vim.org)、vile (dickey.his.com/vile/vile.html)、elvis (elvis.the-title-red-haired-girl.org) 和 nvi (www.bostic.com/vi)，同时还有移植到 Windowsey 及 MacOS 上的版本。其中 vim 最为流行，在许多 Linux 系统上都可以找到。所有这些版本功能类似，共享一个从最初 vi 版本继承而来的核心命令集。

vim 是 vi 的增强版本，几乎与所有 vi 命令都兼容，可以运行在多个平台上，如 Windows、Macintosh、OS/2、UNIX 和 Linux。程序员通常会用 vim 编写 C、HTML、Java 等代码。



在一些系统上，命令 vi 运行的是 vim 的最小版本，该版本比 vim 精简且加载速度快，但具有较少的特性。



10.2 启动与退出 vi (vim)

vi (vim) 的启动与退出极为方便，有专门的命令完成此项操作，而且有多种启动和退出的方法，可根据需要选择使用。

10.2.1 启动 vi (vim)

可以通过直接运行 vi 或 vim 命令启动 vi 或 vim。如果在启动 vi (vim) 时指定编辑的文件，可将文件名作为命令 vi (vim) 的参数。例如，启动 vi，对文件 mybook 进行编辑，命令如下所示。

```
#vi mybook
```

当按下回车键后，命令行消失，出现图 10-1 所示的编辑界面。屏幕左侧的符号“~”表示该行为空，未经过用户编辑。对于用户编辑过的行，其对应行的“~”符号将会消失。在显示区域的底部（状态栏）显示状态信息，状态信息包括错误消息、删除和添加的文本块信息、文件状态信息等。从图 10-1 可以看到，当前屏幕左下角的状态栏显示了当前编辑的文件名。



用“~”表示的空行并不是文件中的空白行。文件中的空白行依然是文件中的一部分内容，而用“~”表示的空行并不在文件之内。



图 10-1 启动 vim

如果用户在启动 vim 时没有指定文件名，vim 会给出简要的帮助信息，如图 10-2 所示。如果希望获得更详细的帮助只需键入命令“:help”；如果要退出 vim，可键入命令“:q”；如果希望查看版本信息，可键入命令“help version6”。



图 10-2 启动 vim 时未指定文件名

除了以上述方式启动 vi (vim) 之外，还有其他一些方便打开和编辑操作的启动方式。表 10-1 中列出了一些启动 vi 的常用方式。

表 10-1 启动 vi 的方式

命 令	说 明
vi filename , vim filename	从 filename 文件的第一行开始编辑，即光标默认停留在文件第一行第一个字符处
vi +n filename, vim +n filename	从 filename 文件的第 n 行开始编辑，即光标默认停留在文件第 n 行第一个字符处
vi + filename, vim + filename	从 filename 文件的最后一行开始编辑，即光标默认停留在文件最后一行的第一个字符处
vi +/pattern filename, vim +/pattern filename	从 filename 文件的第一个包含字符串 “pattern” 的行开始编辑，即光标默认停留在文件 filename 中第一次出现字符串 “pattern” 的位置
vi -r filename, vim -r filename	在系统崩溃后恢复 filename
vi -R filename, vim -R filename	以只读方式编辑 filename



如果只希望浏览文件的内容而不对文件进行编辑，可以使用命令 view。例如，浏览文件 filename，命令如下所示。

```
view filename
```

当用 vi (vim) 打开并编辑一个文件时，该文件会被自动锁定。如果此时另一用户（或当前用户从另一终端或窗口）也希望打开该文件进行编辑，将出现图 10-3 所示的提示信息。可以按提示进行选择。

- 以只读方式打开：按字母 o 键，以只读方式打开该文件，不会影响其他用户对该文件的同时操作。
- 直接编辑：按字母 e 键，可以直接编辑该文件，但在存盘时与其他同时操作该文件的用户可能存在冲突。



- 恢复：按字母 r 键。如果出现该界面是由于上一次编辑此文件时文件崩溃引起的，选择该项可以恢复修改内容。
- 退出：按字母 q 键，直接退出 vi (vim)。

```
E325: 注意
发现交换文件 ".bubble.c.swp"
所有者: root    日期: Thu Jun  5 09:16:47 2008
文件名: ~root/bubble.c
修改过: 否
用户名: root    主机名: localhost.localdomain
进程 ID: 9606 (正在执行)
当打开文件时 ".bubble.c"
日期: Mon Jun  2 22:13:20 2008

(1) 可能有另一个程序也在编辑同一个文件。
    如果是这样，请注意不要一起写入，不然你的努力都会付诸东流。
    退出，或是继续编辑。

(2) 上一次编辑此文件时崩溃
    如果是这样，请用 ":recover" 或 "vim -rbubble.c"
    恢复修改内容（进一步说明请看 ":help recovery"）。
    如果该恢复的都已经恢复了，请直接删除此交换文件 ".bubble.c.swp"
    以避免再看到此信息。

交换文件 ".bubble.c.swp" 已经存在了！
以只读方式打开(O!），直接编辑(E），恢复(R），退出(Q）：
```

图 10-3 打开一个已经锁定的文件

10.2.2 退出 vi (vim)

退出 vi (vim) 可按下 ESC 键，然后输入下面的命令，退出 vi (vim) 后将立即返回到 Shell 提示符。

```
:q
```

当按下 ESC 键并输入冒号（“:”）时，vi (vim) 会把光标移动到屏幕的底部，此时输入命令 q 将退出 vi (vim)。如果输入命令:q!，将强行退出（不提示保存此次编辑的任何修改）。“存盘退出”需要使用命令:wq。“另存为”可使用下面的命令。

```
:w filename
```

其中，filename 为指定的文件名。

10.3 vi (vim) 的运行模式

vi (vim) 有以下三种工作模式。

- 命令模式：也称普通模式。命令模式下，允许用户在文本中四处移动光标，浏览文本内容，通过命令对文本进行删除、复制等操作，但不能直接编辑修改文本内容。
- 插入模式：也称文本模式、编辑模式或输入模式。在插入模式下，允许用户直接编辑输入任何文本信息，并且将编辑的内容直接显示在屏幕上。
- 末行模式：也称最后行模式。先按 ESC 键切换到命令模式下，然后再按冒号（“:”）键，就可以切换到末行模式下。此时，光标始终处于屏幕底部，当用户输入完命令并按回车键后，vi (vim) 会将光标恢复到原来在文本中的位置。在末行模式下，用

用户可以输入命令，完成保存文件、载入其他文件、保存文件或退出等操作。

默认情况下，vi (vim) 会显示其所处的模式。例如，当 vi (vim) 处于插入模式时，屏幕的左下角将显示如下信息。

```
--插入--
```

3 种模式之间的关系如图 10-4 所示。

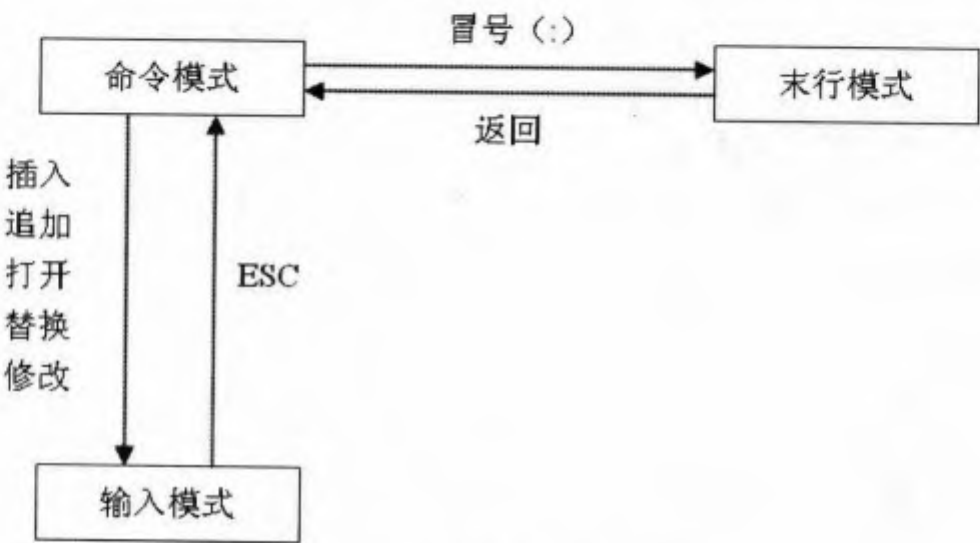


图 10-4 vi (vim) 中的模式

10.3.1 命令模式

用户可以输入命令，通过命令实现移动光标、删除或修改部分文本，以及如复制、粘贴、退出等操作。命令一经输入就被立即执行，不需要按回车键。大多数命令在屏幕中无回显，即用户在执行命令时，通常只能看到光标位置、当前行或模式状态的变化，无法在屏幕中直接看到自己执行的命令。

通过输入表 10-2 所示命令，可将 vi (vim) 由命令模式切换到插入模式。用户刚启动 vi (vim) 后，vi (vim) 默认处于命令模式下，而且不管在什么模式下，只要按一下 ESC 键，即可将 vi (vim) 切换到命令模式下。

表 10-2 切换到插入模式的命令

命 令	说 明
i	在当前字符之前插入文本
I	在当前行的行首插入文本
a	在当前字符之后追加文本
A	在当前行的行尾追加文本
o	在当前行的下面添加一新行
O	在当前行的上面添加一新行



由于 vi (vim) 大小写敏感，所以在输入命令时，应注意区分大小写，vi (vim) 会将字母相同但大小写不同的两个命令认为是不同的命令。



10.3.2 插入模式

在命令模式下，按下字母“a”、“o”或“i”（或“A”、“O”、“I”）键即可切换到插入模式。如果不能确定 vi（vim）当前是处于何种模式，只需按下 ESC 键，然后再按下字母“a”或“o”或“i”（或“A”、“O”、“I”）键。

在插入模式下，用户按下的任何键都被当作文本，用户键入的字符只要是可打印字符就可直接插入到文本的光标处。如果屏幕显示的内容不是用户所键入的，说明 vi（vim）当前不处于插入模式，需要重新切换模式。

在输入文本时，由于 vi（vim）会限制文本向右侧持续扩展，因此在每行的行尾需要按回车键，否则 vi（vim）会自动换行到下一行，从而引起物理行不能与逻辑行对应的问题。

在插入模式下，按下 ESC 键，vi（vim）将切换回命令模式。

10.3.3 末行模式

在命令模式下按下冒号键即可切换到末行模式。此时光标停留在最后一行上，并等待用户输入所需执行的末行命令。当用户输入命令后，按回车键，命令即可被执行。

例如，在末行模式下，输入命令“set number”并按回车键后，文本 bubbleSort.c 在每行的左端显示对应的行号，如图 10-5 所示。

```
1 |include <stdio.h>
2 |void bubbleSort( int list[], int size )
3 |{
4 |    int i, j, temp, flag = 1;
5 |    while (flag)
6 |    {
7 |        flag = 0;
8 |        for ( i = (size-1) ; i >= 0 ; i-- )
9 |        {
10 |            for ( j = 0 ; j <= i ; j++ )
11 |            {
12 |                if ( list[j-1] > list[j] )
13 |                {
14 |                    temp = list[j-1];
15 |                    list[j-1] = list[j];
16 |                    list[j] = temp;
17 |                    flag = 1;
18 |                }
19 |            }
20 |        }
21 |    }
22 |}
23 |int main( )
24 |{
25 |    int myList[10]={8, 5, 7, 9, 10, 3, 2, 1, 7, 0};
26 |    int i;
27 |
28 |    bubbleSort(myList, 10 );
29 |
30 |    for ( i = 0 ; i < 10 ; i++ )
31 |    {
32 |        printf("%d ", myList[i]);
33 |    }
: set number
```

图 10-5 在文本前添加行号

10.4 vi（vim）中的常用命令

vi（vim）中包含丰富的命令，这些命令简单、通用，是 vi（vim）的优点所在。熟练掌握

握这些命令可以极大地加快文本处理速度。

10.4.1 插入文本命令

在命令模式下，插入文本命令包括 `i` 命令和 `I` 命令，可以在当前光标位置或当前行的行首插入文本。小写的插入命令（“`i`”）在当前光标处插入文本，原来位于光标处的字符向后移动一个位置。大写的插入命令（“`I`”）在当前光标所在行的行首插入文本，该行的其他文本依次向后移动。

`i` 命令与 `I` 命令的主要区别在于插入的起始位置，而不在于插入的内容。键入插入文本命令后，文本进入插入模式，按 `ESC` 键可切换回命令模式。

10.4.2 附加文本命令

在命令模式下，附加文本命令包括 `a` 命令和 `A` 命令，可以在当前光标位置之后或当前行的行尾附加文本。小写的附加命令（“`a`”）在当前光标之后添加文本，原来光标处的字符保持不变。大写的附加命令（“`A`”）在当前光标所在行的行尾添加文本，该行的原文本保持不变。

键入附加文本命令后，文本进入插入模式，按 `ESC` 键可切换回命令模式。

10.4.3 添加新行命令

在命令模式下，添加新行命令包括 `o` 命令和 `O` 命令，可以实现在文本中添加一个新行。小写的添加新行命令（“`a`”）在当前光标所在行的下方添加一新行。大写的添加新行命令（“`O`”）在当前光标所在行的上面一行添加一新行。如果当前行跨越多个屏幕行，则在包含回车符的屏幕行之后或之前打开新一行，而不是在包含光标的屏幕行之前或之后添加新行。

键入添加新行命令后，文本进入插入模式，按 `ESC` 键可切换回命令模式。

10.4.4 光标移动命令

为了编辑文本，需将光标移动到需要编辑的文本位置。光标移动命令只有在命令模式下才有效。在执行光标移动命令之后，`vi` (`vim`) 仍然处于命令模式，可以继续移动光标。通常也可以通过光标移动命令实现对文本的浏览。为明确移动方向，光标向前/向后的约定如图 10-6 所示。

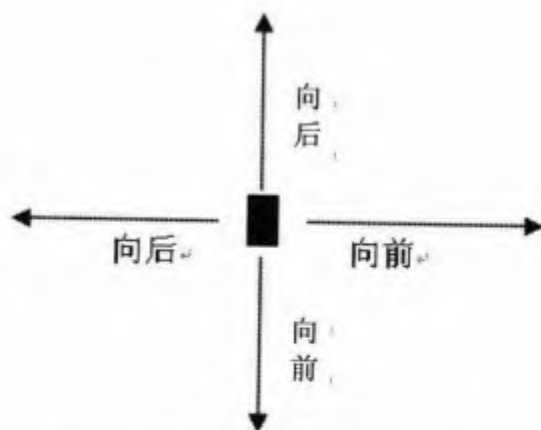


图 10-6 光标前后位置的约定



常用的光标移动命令如表 10-3 所示。

表 10-3 光标移动命令

命 令	说 明
k, ↑	将光标上移一行，该行为逻辑行，可能包含多个物理行。如果上一行包含不少于当前行的字符数，则光标移到上一行的同一相对位置处。如果上一行比当前行短，则移动到上一行的行尾
j, ↓	将光标下移一行，该行为逻辑行，可能包含多个物理行。如果下一行包含不少于当前行的字符数，则光标移到下一行的同一相对位置处。如果下一行比当前行短，则移动到下一行的行尾
-	移动光标到上一行的行首，如果当前行为最上一行，则光标不移动
+	移动光标到下一行的行首，如果当前行为最下一行，则光标不移动
h,Backspace, ←	将光标左移一个字符，需要连续移动时，可以重复按该键，也可以直接指定移动的字符数，例如，“7h”表示向左移动 7 个字符
l,Space, →	将光标右移一个字符，需要连续移动时，可以重复按该键，也可以直接指定移动的字符数，例如，“7l”表示向右移动 7 个字符
0,	移动光标到当前行的行首
\$	移动光标到当前行的行尾
w, W	w 命令将光标前移动到下一个单词的第一个字母（标点符号也被当作一个单词），如果下一个单词位于下一行，那么该命令也可以将光标移动到下一行。可以直接指定移动的单词数，例如，“7w”表示向前移动 7 个单词。W 命令与 w 类似，也是向前移动光标，但 W 键以空白为分隔符（包括标点符号）来移动光标
b, B	b 命令将光标向后移动到上一个单词的第一个字母（标点符号也被当作一个单词），如果上一个单词位于上一行，那么该命令也可以将光标移动到上一行。可以直接指定移动的单词数，例如，“7b”表示向后移动 7 个单词。B 命令与 b 类似，也是向后移动光标，但 B 键以空白为分隔符（包括标点符号）来移动光标
e, E	e 命令表示 把光标移到下一个单词的末尾字符，E 命令与 e 相似，只是以空白为分隔符进行移动
(,)	)命令可以将光标向前移动到下一个句子的开始，(命令可以将光标移动到上一个句子的开始
{, }	}命令可以将光标向前移动到下一个段落的开始位置，{命令可以将光标移动到上一个段落的开始位置
H	将光标移动到屏幕顶部第一行的最左端
M	将光标移动到屏幕中间
L	将光标移动到屏幕的尾部



vi (vim) 对“单行”的长度没有限制，仅受限于可用内存容量。而文件的总长度受限于可用的磁盘和内存空间。

10.4.5 删除命令

在命令模式下，删除命令分两种，一种是字符删除命令，另一种是块删除命令。

删除字符使用 x/X 命令，x 命令可以删除当前字符，X 命令可以删除当前光标左侧字符。如果在使用命令前指定了重复次数，可连续删除多个字符。例如，命令“3w”表示删除从当前光标位置开始向后的 3 个字符，命令“3W”表示删除当前光标之前的 3 个字符。

命令 d/D 可以把指定的文本块从工作缓冲区中删除。命令 D 表示删除从当前光标开始到本行末尾一段字符，命令 d 还需要配合其他命令组合来确定删除文本块的大小。表 10-4 中列

出了常用的删除命令及说明。

表 10-4 常用删除命令及其说明

命 令	说 明
x	删除当前字符
X	删除当前光标左侧字符
dl	删除当前光标处的字符，与命令 x 相同
d0	从行的开始处删除
d\$	删除到行的末尾
D	删除到行的末尾
d^	从行开始的第 1 个字符开始删除（不包括空格和 TAB）
dw	删除到单词的末尾
d5w	删除到第 5 个单词的末尾
dtc	删除到当前行字符 c 的下一出现
dfc	删除到当前行字符 c 的下一出现（包括 c）
d/word	删除到单词 word 的下一出现
d3{	从前面的第 3 段删除到当前位置
d{	从本段的开始位置删除到当前位置
db	从单词的开始处删除
dW	删除到以空白为分隔的单词的末尾
dB	从以空白为分隔的单词的首字符开始删除
d5B	从以空白为分隔的前面第 5 个单词的首字符开始删除
d)	删除到句子的末尾
d4)	删除到第 4 个句子的末尾
d}	删除到段落末尾
d4}	删除到后面的第 4 段末尾
dd	删除一行
3dd	删除当前行开始的 3 行
dL	删除到当前屏幕的最后一行（包括最后一行）
dH	从当前屏幕的第一行开始删除

10.4.6 修改命令

修改文本命令 c/C 应用在命令模式下，可以删除指定的文本，并使 vi (vim) 进入插入模式。在编辑完新文本后，按 ESC 键可切换回命令模式，完成本次修改操作。如果进入插入模式后没有对文本进行修改，则该命令等同于删除命令。

命令 C 表示从光标位置删除到行的结尾并切换到插入模式，命令 c 还需要配合其他命令组合来确定删除文本块的大小，并切换到插入模式。表 10-5 中列出了一些常用的修改命令及



说明。

表 10-5 常用修改命令说明及示例

命 令	说 明
cl	删除当前光标处的字符，并进入插入模式
c0	从行的开始处删除，并进入插入模式
c\$	删除到当前行的末尾，并进入插入模式
C	删除到行的末尾，并进入插入模式
c^	从行开始的第 1 个字符开始删除（不包括空格和 Tab），删除后进入插入模式
cw	删除到单词的末尾，删除后进入插入模式
c5w	删除到第 5 个单词的末尾，删除后进入插入模式
ctc	删除到当前行字符 c 的下一次出现，删除后进入插入模式
cfc	删除到当前行字符 c 的下一次出现（包括 c），删除后进入插入模式
c/word	删除到单词 word 的下一次出现，删除后进入插入模式
c3{	从前面的第 3 段删除到当前位置，删除后进入插入模式
c{	从本段的开始位置删除到当前位置，删除后进入插入模式
cb	从单词的开始处删除，删除后进入插入模式
cW	删除到以空白为分隔的单词的末尾，删除后进入插入模式
cB	从以空白为分隔的单词的首字符开始删除，删除后进入插入模式
c5B	从以空白为分隔的前面第 5 个单词的首字符开始删除，删除后进入插入模式
c)	删除到句子的末尾，删除后进入插入模式
c4)	删除到第 4 个句子的末尾，删除后进入插入模式
c}	删除到段落末尾，删除后进入插入模式
c4}	删除到后面的第 4 段末尾，删除后进入插入模式
cc	删除当前行，删除后进入插入模式
3cc	删除当前行开始的 3 行，删除后进入插入模式
cL	删除到当前屏幕的最后一行（包括最后一行），删除后进入插入模式
cH	从当前屏幕的第一行开始删除，删除后进入插入模式

10.4.7 更改大小写命令

在命令模式下，代字符（“~”）用来更改字母的大小写，可以将小写字母改为大写，也可以将大写字母改为小写。在代字符前加入数字可以指定要修改大小写的字符数目。例如，“3~”表示将改变从当前光标所在字符开始的 3 个字符的大小写。如果指定的数目超出了当前行的行尾，则修改到行尾结束。

10.4.8 替换命令

在命令模式下，替换命令 s/S 可以删除指定的文本，并使 vi（vim）进行插入模式。命令

s 只替换当前字符，即删除当前光标所在位置的字符并切换到插入模式。命令 S 与命令 cc 类似，可以删除当前行，并切换到插入模式。编辑完文本后，按 ESC 键可切换为命令模式，完成替换操作。

在命令 s 前加入数字可以指定要替换字符的数目。例如，“3s”表示删除从当前光标所在字符开始的 3 个字符，并切换到插入模式。如果指定的数目超出了当前行的行尾，则删除到行尾结束。表 10-6 列出了常用的替换命令及说明。

表 10-6 常用替换命令及其说明	
命 令	说 明
s	用一个或多个字符替换当前光标位置处的字符
S	用一个或多个字符替换当前行
10s	用一个或多个字符替换从当前字符开始的 10 个字符

10.4.9 合并命令

在命令模式下，合并命令可将当前行的末尾与下一行连接起来，并在两行之间插入一个空格，合并后光标定位到该空格处。如果当前行以句号结束，则会在两行之间插入两个空格。

例如，合并前两行文本，命令如下所示。

```
Hello, everybody! I am Sullivan.  
I like Linux.
```

合并后，光标停在了原来第一行文本的末尾，如下所示。

```
Hello, everybody! I am Sullivan. I like Linux.
```

10.4.10 句点命令

在命令模式下，句点命令将重复执行最近一次的修改命令。例如，如果刚刚用命令 dd 删除了当前一行，那么命令“.”将继续删除接下来的一行；如果刚刚插入了一行文本“I like Linux”，则命令“.”将再次在光标位置插入文本“I like Linux”。

10.4.11 撤销命令

vi (vim) 包括 1 个通用缓冲区和 26 个命名缓冲区，vi (vim) 会将最近的修改保存在通用缓冲区中，撤销命令实际上是从通用缓冲区中取出文本并进行恢复操作。在编辑、修改文本之后，在命令模式下直接使用撤销命令可以恢复到文本修改之前的状态。撤销修改命令包括 u 命令和 U 命令。

命令 u 可以撤销上一次的编辑操作。执行一次 u 命令只能撤销上一次对文本的操作。如果删除一行后又插入了某些字符，执行一次 u 命令只能删除插入的字符。如果希望恢复删除的行，则需要再执行一次 u 命令。

命令 U 可以撤销对当前行的所有修改，将文本恢复到启动修改之前的状态（即使期间进行了很多次修改）。



撤销命令也可以被撤销。

10.4.12 查找字符命令

在命令模式下，命令 `f` 可以在当前行从当前光标处开始查找指定的字符，并将光标移动到该字符出现的位置。如果没有找到，则不移动光标。`F` 命令与 `f` 类似，只是在本行开始位置到当前光标所在位置之间查找。

命令 `t` 在查找字符时，会将光标定位在指定字符出现位置的前一个字符位置，而命令 `T` 在查找字符时会将光标定位在指定字符出现的后一字符位置。

查找命令可以与其他命令组合使用。例如，命令“`d2fq`”表示从当前光标所在单词位置开始搜索，删除 2 次出现字符“`q`”的单词之间的单词。例如，使用命令“`d2fq`”对下面字符串进行处理。

```
bbbbb ccccccqcccc wwwwww ddddqdddd eeeeeqeeee
```

执行“`d2fq`”命令后，删除了两个包含“`q`”字符的单词之间的单词，结果如下所示。

```
bbbbb eeeeeqeeee
```

10.4.13 读入文件命令

在末行模式下，使用命令 `:r` 可以将指定的文件读入缓冲区。新读入的文件并不覆盖缓冲区中的原文件内容，而是被插入到指定位置之后。如果没有指定插入的位置，则默认插入到当前光标所在行的下方。该命令的语法格式如下所示。

```
: [ address ] r [ filename ]
```

其中，`address` 表示插入的位置，可以是行号，`filename` 表示插入的文件名。

例如，位置 0 表示将新读入的文件插入到缓冲区的起始位置（即最顶端）。下面命令将文件 `filename` 读入到当前缓冲区的起始位置。

```
: 0 r filename
```

例如，将文件 `filename` 插入到当前缓冲区第 4 行的下面，命令如下所示。

```
: 4 r filename
```



如果在使用 `:r` 命令时没有指定读入的文件名，默认再次读入当前文件。

10.4.14 写入文件命令

在末行模式下，使用命令 `:w` 可以将缓冲区中的部分或全部内容写入到磁盘。命令 `:w` 的语法格式如下所示。

```
: [ address ] w [ ! ] [ filename ]
```

其中，`address` 指定写入磁盘的文本范围，以行号表示，`filename` 表示写入到磁盘的文件

名。如果不指定地址和文件名，默认把整个缓冲区的内容写入到正在编辑的文件中，即更新磁盘上的当前文件。若指定的 filename 文件已经存在，将无法写入。“!”表示强制执行，可以进行写入覆盖。命令:w 还有另外一种语法格式，如下所示。

```
: [ address ] w >> filename
```

其中，address 表示把缓冲区中指定的一部分内容写入磁盘，以行号表示，>>filename 表示向已有的文件 filename 追加文本。

例如，将当前缓冲区中 3~8 行写入到文件 temp 中，命令如下所示。

```
:3,8 w temp
```

例如，将当前缓冲区中 4~10 行追加到文件 temp 中，命令如下所示。

```
:4,10 w >> temp
```

10.4.15 定位到指定行命令

在命令模式下，输入命令“行号+G”可以将当前光标定位到缓冲区中指定的行。如果没有指定行号，默认将光标定位到缓冲区最后一行。命令 G 不需要缓冲区显式地给出行号，即，即使不显示行号也可以定位到指定的行。

例如，在命令模式下，输入“3G”即可将光标定位到缓冲区第 3 行。



在末行模式下，输入命令“set number”并按回车键后，可以在缓冲区中显式地给出每行的行号。

10.4.16 复制命令

复制命令实际是把指定的文本内容复制到通用缓冲区。复制命令包括 y 和 Y 两个命令，需要在命令模式下使用。

命令 yy 可以复制一行，默认为当前行。如果需要复制多行，可以在命令 yy 前指定行数。例如，下面的命令复制从当前行开始的 3 行文本。

```
3yy
```

命令 Y 与命令 yy 类似，可以将当前行复制到通用缓冲区。如果需要复制多行，可以在命令 Y 前指定行数。例如，下面的命令复制从当前行开始的 5 行文本。

```
5Y
```

10.4.17 粘贴命令

粘贴命令实际是把通用缓冲区中的内容粘贴到当前位置。粘贴命令包括 p 和 P 两个命令，需要在命令模式下使用。

命令 p 可以把通用缓冲区的内容插入到当前字符之后，而命令 P 则把通用缓冲区的内容插入到当前字符之前。可以连接多次使用 p 或 P 命令来实现多次粘贴。

10.4.18 重复命令

如果撤销某个命令后又想重新执行该命令，可以在命令模式下使用 Ctrl+R 组合键，也可



以在末行模式下输入命令:redo，并按回车键，vi（vim）会重新执行已被撤销的命令。重复命令也可以被连续执行多次。

10.4.19 显示状态信息命令

显示文件的状态信息通常有两种方式：在命令模式下按 Ctrl+G 键和在末行模式下使用命令:f。

末行模式下，使用命令:f 可以显示文件的状态信息，包括文件名、行数、当前行号、当前行之前的行数占缓冲区中行数的百分比。例如，图 10-7 中通过命令:f，显示了文件 bubbleSort.c 的状态信息。

```
#include <stdio.h>
void bubbleSort( int list[], int size )
{
    int i, j, temp, flag = 1;
    while (flag)
    {
        flag = 0;
        for ( i = (size-1); i >= 0; i-- )
            for ( j = 0; j <= i; j++ )
                if ( list[j] > list[j+1] )
                {
                    temp = list[j];
                    list[j] = list[j+1];
                    list[j+1] = temp;
                    flag = 1;
                }
    }
}

int main( )
{
    int myList[] = {10, 9, 8, 7, 6, 5, 4, 3, 2, 1};
    int i;

    bubbleSort(myList, 10);

    for ( i = 0; i < 10; i++)
        printf("%d ", myList[i]);
}
```

"bubbleSort.c" 行数 35 字节 284

图 10-7 文件 bubbleSort.c 的状态信息

10.4.20 保存和退出命令

常用的保存与退出命令共有 6 个，具体说明如表 10-7 所示。其中，除命令 ZZ 是在命令模式下使用外，其他命令都是应用在末行模式下，即需要先键入冒号（“:”）在屏幕底部打开一个命令行，然后再输入命令并按回车键。

表 10-7 保存与退出命令说明

命 令	说 明
:w	保存缓冲区内容，但不退出。保存后磁盘中的原始文件被修改，不能恢复
:w filename	将缓冲内容保存到指定的文件 filename 中，原文件未被修改，相当于“另存为”。在保存之后，不退出，仍处在编辑刚保存的文件状态
ZZ	保存并退出，在命令模式下使用
:q	退出，但如果缓冲区已被修改将提示错误信息，不能退出
:wq	保存缓冲区内容并且退出。与命令模式下的 ZZ 命令效果相同
:q!	不保存缓冲区内容，直接退出。退出时系统不进行警告

10.5 查找与替换字符串

vi (vim) 可以在整个缓冲区查找与正则表达式匹配的字符串。在命令模式下，键入斜杠 (/) 后跟待查找的字符串（称为搜索字符串），按回车键，vi (vim) 即可开始搜索。状态行将同步显示输入的斜杠和搜索字符串。如果搜索成功，光标会停留在首次匹配该字符串的第一个字符处。如果向后搜索到文件的末尾仍没有找到搜索的字符串，则折回到缓冲区的开始继续搜索。同样，在向前搜索到文件头仍没有找到搜索的字符串，则返回缓冲区的末尾继续搜索。

按 n 键可以向后重复上一次搜索，按 N 键可以向前重复上次搜索。

10.5.1 普通查找与增量查找

在末行模式下，使用命令 “:set noincsearch” 和 “:set incsearch” 可以设置普通查找或增量查找。

在普通查找方式下（默认的查找方式），输入斜杠和搜索字符串并按回车键后，vi (vim) 便开始搜索。光标会移动到搜索字符串首次出现的位置。设置普通查找使用如下命令。

```
:set noincsearch
```

在增量查找方式下输入斜杠后，随着搜索字符的依次输入，vi (vim) 会高亮显示已输入字符串从当前光标开始的首次匹配位置，按回车键后，光标移动到首次匹配位置。设置增量查找使用如下命令。

```
:set incsearch
```

10.5.2 替换字符串

在末行模式下的替换命令 (s) 具有查找与修改双重功能。替换命令首先查找某个字符串（匹配搜索字符串的字符串），然后修改该字符串。末行模式下的替换命令 s 的语法格式如下所示。

```
:[g] [address] s/ search/replacement [ /option ]
```

其中，g 表示在所有行进行查找和替换，否则只对第一次查找结果进行替换。address 表示查找的范围，如果不指定 address，默认只搜索当前行。有关 address 的应用示例见表 10-8。search 表示搜索字符串，replace 表示替换字符串，“/” 为分隔符。



分隔符可以使用除字母、数字、空白符和反斜杠之外的任意其他字符，但应保持 search 前后的分隔符相同。

表 10-8

address 应用示例

address 示例	说 明
3	第 3 行
22, 100	第 22~ 100 行之间（包括第 22 行及 100 行）
5, .	从第 5 行开始至当前行



续表

address 示例	说 明
5,\$	从第 5 行开始至末尾
%	整个缓冲区
g/pattern/	包含字符串 pattern 的所有行
/pattern/	搜索到的首次包含字符串 pattern 的行
.,+20	从当前行开始向后的连续 20 行
.,-10	从当前行开始向前的连续 10 行



在 address 中，“.”代表当前行，“\$”代表缓冲区的最后一行，“%”代表整个缓冲区。表 10-9 中例出了一些查找与替换的示例。

表 10-9 查找与替换示例

命 令	说 明
:s/string1/string2	将当前行中第一次搜索到的字符串 string1 替换为 string2
:1,s/string1/string2/g	将当前行之前所有行中的字符串 string1 替换为 string2，其中 g 表示在每一行中都允许进行多次替换
:1,\$s/string1/string2/g	将所有行中出现的字符串 string1 替换为 string2
:%s/string1/string2/g	将所有行中出现的字符串 string1 替换为 string2
.,+10s/string1/string2/g	将从当前行开始连续 10 行内出现的字符串 string1 替换为 string2
:/string/s/string1/string2/g	将首次出现字符串 string 的行中的字符串 string1 替换为 string2
:g /string/s/string1/string2	在所有行中将包含 string 字符串的行中的第 1 次出现 string1 的字符串替换为 string2
:g /string/s/string1/string2/g	在所有行中将包含 string 字符串的行中出现的所有 string1 的字符串替换为 string2
5.,s/string1/string2	将第 5 行到当前行之间所有行中出现的 string1 替换为 string2

10.6 窗口操作

vi (vim) 允许打开、创建、关闭多个窗口，可以在不同窗口中分别编辑不同的文件或同一文件的不同部分。窗口操做极大地方便了多文件操作，提高了文本处理的效率。

10.6.1 窗口操作的快捷方式

vi (vim) 提供了实现窗口操作的一些快捷方式，如下所示。

- 组合键 Ctrl+w+s：用于打开编辑同一个文件的另一个窗口，即用于实现窗口的水平拆分。
- 组合键 Ctrl+w+n：可以打开一个新窗口编辑一个空文件。
- 组合键 Ctrl+w+w：实现在多个窗口之间的切换。

10.6.2 窗口水平拆分

在末行模式下，通过命令 `split` 可以实现窗口的水平拆分。例如，在末行模式下使用命令 `split` 将当前打开的文档 `bubbleSort.c` 进行水平拆分，如下所示。

```
:split
```

该命令执行后，屏幕被分割为上下两个窗口，可以用来浏览文档的不同部分，如图 10-8 所示。



图 10-8 使用 `split` 命令将窗口进行水平拆分

如果在使用命令 `split` 的同时指定一个文件名，可以创建一个新窗口，并在窗口中载入指定的文件；如果指定的文件不存在，将在窗口中打开一个具有指定文件名称的空文件。例如，在末行模式下使用命令 `split` 将指定的文档 `kk2.c` 载入到一个新的窗口中，如下所示。

```
:split kk2.c
```

该命令执行后，屏幕被分割为上下两个窗口，上部窗口中显示新打开的文件 `kk2.c`，下部窗口中显示原来编辑的文件 `bubbleSort.c`，如图 10-9 所示。



图 10-9 使用 `split` 命令打开一个新窗口



10.6.3 窗口垂直拆分

在末行模式下，通过命令 `vsplit` 可以实现窗口的垂直拆分。例如，在末行模式下使用命令 `vsplit` 将当前打开的文档 `bubbleSort.c` 进行垂直拆分，如下所示。

`:vsplit`
该命令执行后，屏幕被分割为左右两个窗口，可以用来浏览文档的不同部分，如图 10-10 所示。

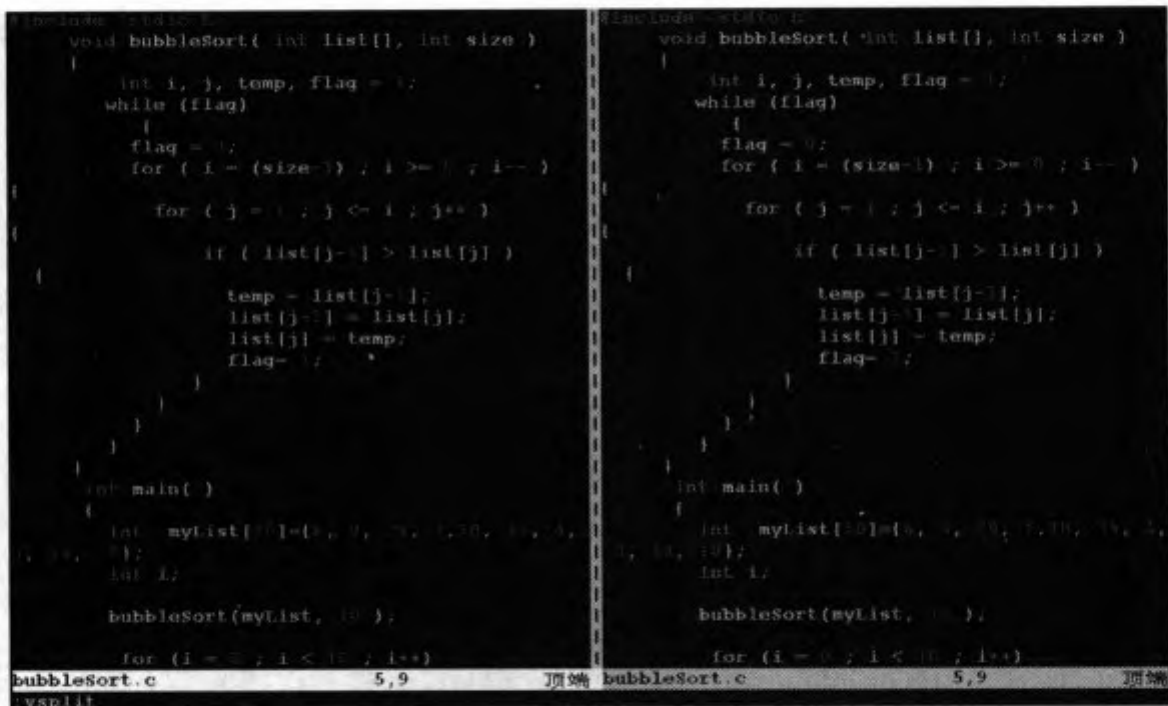


图 10-10 使用 `vsplit` 命令将窗口进行垂直拆分

如果在使用命令 `vsplit` 的同时指定一个文件名，可以创建一个新窗口，并在窗口中载入指定的文件；如果指定的文件不存在，将在窗口中打开一个具有指定文件名称的空文件。例如，在末行模式下使用命令 `vsplit` 将指定的文档 `kk2.c` 载入到一个新的窗口中，如下所示。

`:vsplit kk2.c`
该命令执行后，屏幕被分割为左右两个窗口，左部窗口中显示新打开的文件 `kk2.c`，右部窗口中显示原来打开的文件 `bubbleSort.c`，如图 10-11 所示。



图 10-11 使用 `vsplit` 命令打开一个新窗口

10.6.4 关闭窗口

- 在 vi (vim) 中关闭窗口有如下几种方法。
- 关闭指定的窗口：将光标转换到希望关闭的窗口，然后按照关闭单个窗口的方法关闭窗口。例如，在末行模式下使用命令:q。
 - 关闭所有窗口：在任何窗口的末行模式下使用命令:qall。
 - 关闭除当前窗口之外的窗口：将光标移动到希望保持打开状态的窗口下，在末行模式下使用命令:only。

10.7 使用 Shell 命令

vi (vim) 中有多种执行 Shell 命令的方式，并且可以将 Shell 命令的执行结果插入到当前编辑的文件中。

10.7.1 启动一个新 Shell

在末行模式下输入命令:sh，可以派生一个新的交互式 Shell，如下所示。

:sh

回车后将出现一个新 Shell。执行完 Shell 命令之后，执行 exit 命令或按 Ctrl+d 组合键可以退出 Shell，返回到 vi (vim)。

10.7.2 在 vi (vim) 中执行 Shell

vi (vim) 在末行模式下也可以直接执行 Shell 命令，格式如下所示。

:! command

其中，command 表示执行的 Shell 命令。当 command 执行完毕后，按回车键返回 vi(vim)。例如，在末行模式下执行如下命令。

:! ls -al

回车后，“ls -al”命令被执行，如图 10-12 所示。按回车后将返回 vi (vim)。

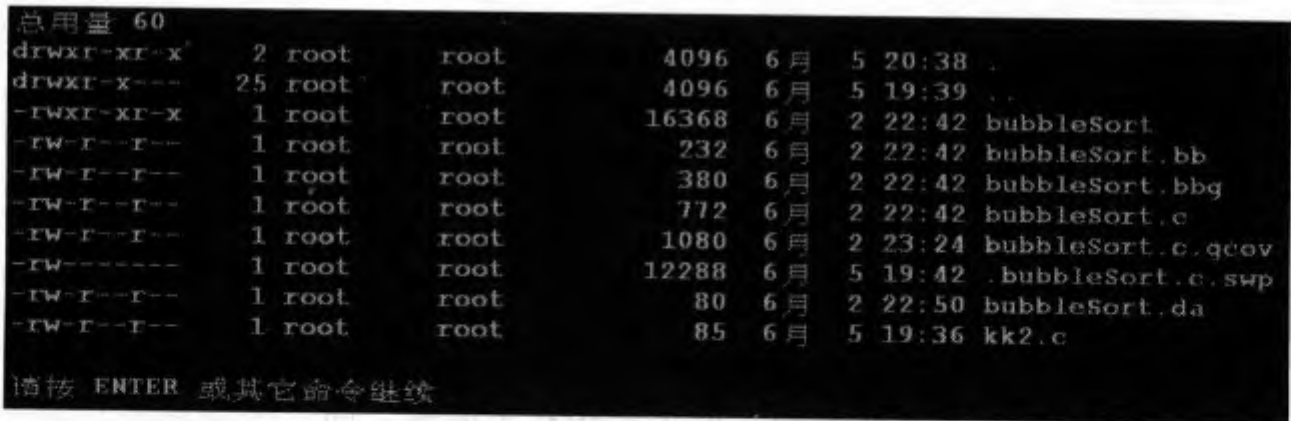


图 10-12 在 vi (vim) 中执行 Shell 命令

在 vi (vim) 中执行 Shell 命令，还可以将执行结果返回插入到当前编辑的文件中，在末行模式下的语法格式如下所示。



```
:.!command
```

在命令模式下的语法格式如下所示。

```
!!command
```

例如，将 Shell 命令“ls -al”的执行结果插入到当前编辑文件的当前光标位置处，在命令模式下执行如下所示命令。

```
!! ls -al
```

执行完毕后，如图 10-13 所示，可以看到“ls -al”命令的执行结果被插入到当前编辑文件的当前光标位置处。

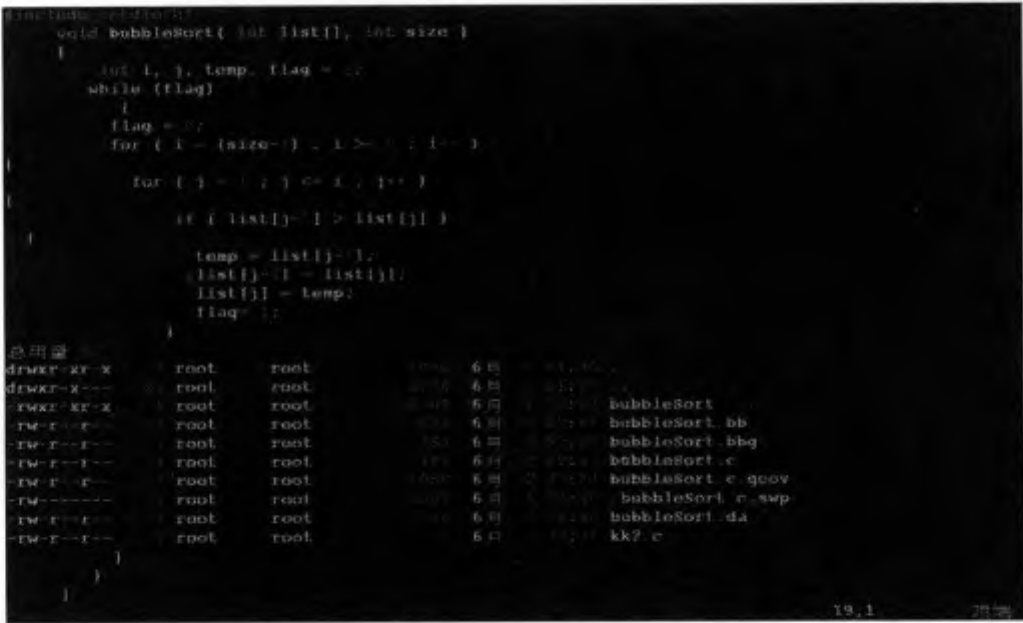


图 10-13 将 Shell 命令结果插入到文件中

10.8 其他常用操作

为了更方便地使用 vi (vim)，还应掌握一些常用的操作方法，包括屏幕重绘、滚动屏幕以及建立和使用标记。

10.8.1 屏幕重绘

有时由于某些程序与 vi (vim) 缓冲区出现冲突，影响了 vi (vim) 的显示效果，可能导致屏幕扭曲、变形，甚至出现乱码。这时，只需按 ESC 键将 vi (vim) 切换到命令模式下，然后按组合键 Ctrl+L，即可重绘屏幕。

10.8.2 滚动屏幕

当一个文件的行数超出屏幕所能显示的最大行数时，需要上下滚动文本才能观察到全部内容。滚动的方向是相对于文本而言的，而不是相对于屏幕。向下滚动意味着向文件的结尾处移动，向上滚动意味着向文件的起始位置移动。

滚动屏幕命令应用于命令模式下，常用的滚动命令有 6 个，都是包含控制键 (Ctrl) 的组合键，详细说明如表 10-10 所示。

表 10-10 常用的滚动命令

命 令	说 明
Ctrl+y	向上滚动一行。当向上滚动时，光标位置不变，除非光标是位于屏幕最底部，此时光标上移一行
Ctrl+e	向下滚动一行。当向下滚动时，光标位置不变，除非光标是位于屏幕的最顶部，此时光标下移一行
Ctrl+u	向上滚动半屏。在滚动之后，光标位于目标行的行首，如果光标距离顶部不足半屏，则光标停留在屏幕的顶部
Ctrl+d	向下滚动半屏。在滚动之后，光标位于目标行的行首，如果光标距离底部不足半屏，则光标停留在屏幕的底部
Ctrl+f	向下滚动整屏。在滚动之后，光标位于目标行的行首
Ctrl+b	向上滚动整屏。在滚动之后，光标位于目标行的行首

10.8.3 建立和使用标记

建立和使用标记可以更容易地实现文本的定位。在命令模式下使用命令“m+标记”可以给当前行设置标记，其中标记可以采用任意字符。例如，若对文本中的第 3 行设置标记“t”，只需将光标移动到第 3 行，然后在命令模式下键入命令“mt”。



vi (vim) 在退出时不保存标记。

标记被设置后，可以像使用行号一样定位标记。定位标记时在末行模式下使用“'+标记”，可以直接定位到标记所在行的行首。例如，定位上例设置的 t 标记所在行，只需在末行模式下输入以下命令。

```
: 't
```

按回车键后，即定位到第 3 行的行首。例如，删除从当前行到标记 t 之间的所有行，命令如下所示。

```
:d 't
```

例如，将从标记 t 开始到当前行中出现的所有字符串 string1 替换为 string2，命令如下所示。

```
: 't, . s /string1/string2/g
```

在定位标记时，在末行模式下使用“`+标记”，可以定位到所标记的字符，而不是“'+标记”方式下定位到标记所在行的行首。例如，删除从当前行到标记 t 所标记字符之间的所有文本，命令如下所示。

```
:d `t
```

10.9 获得帮助

在末行模式下，输入命令“:help”可以看到屏幕被分为上下两个窗口，如图 10-14 所示。上部窗口显示帮助信息，下部窗口显示当前打开的文件。

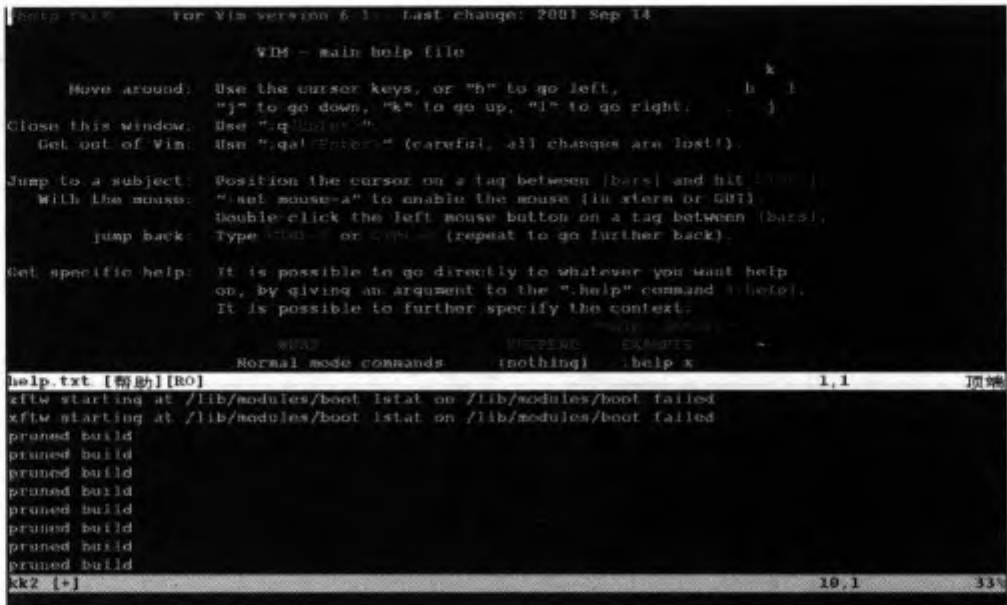


图 10-14 vi (vim) 帮助信息主界面

可以看到，上部窗口的底部显示了帮助信息来自文件 `help.txt`，下部窗口的底部显示了当前打开文件为 `kk2`。在末行模式下输入命令 `“:q”` 可以关闭帮助窗口，返回到当前打开的文件窗口。

在末行模式下，通过输入主题词可以查找相关的帮助主题。经常查找的主题词包括 `insert`（插入）、`delete`（删除）、`opening-window`（打开窗口）等。输入命令 `“:help doc-file-list”` 可以查看帮助文件的列表。输入命令 `“:help delete”` 可以获得一些与删除命令相关的帮助信息，如图 10-15 所示。

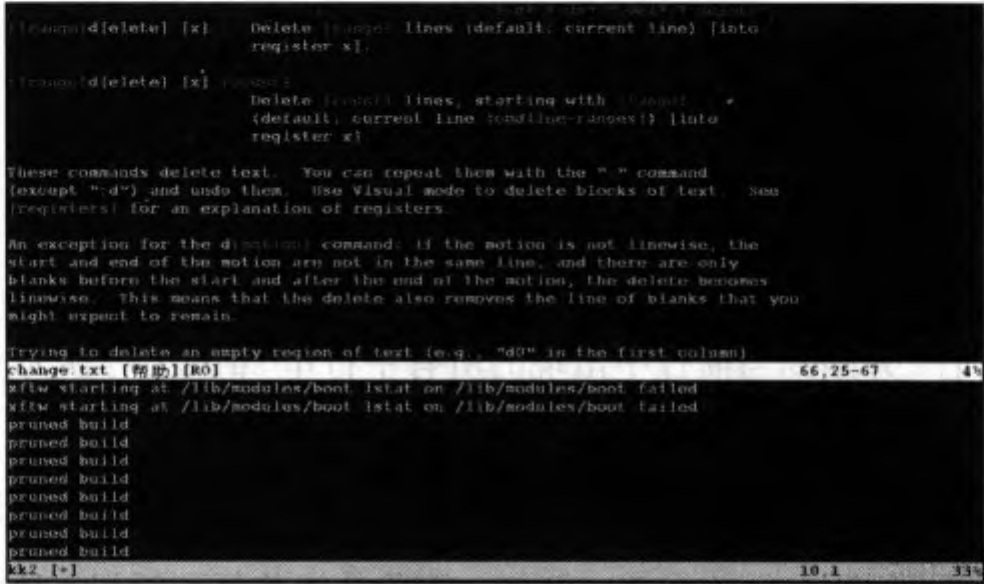


图 10-15 关于删除命令的帮助信息

当滚动浏览帮助信息时，用户有时会看到由两条竖线括起来的单词，如图 10-15 中的 `|cmdline-ranges|`，这表示该单词是一个活动链接，将光标移动到这些活动链接并按 `Ctrl+]键`即可跳到其对应的链接文本。按 `Ctrl+o 键`可以返回前一页面。



vim 是一个自由软件，但程序的开发者建议用户在使用这些自由软件的过程中，能为乌干达的孩子们捐一点款，通过输入命令 `“:help uganda”` 可以看到有关捐助的详细信息。

UNIX

第11章

emacs 编辑器

emacs (editing macros), 即宏编辑器, 是在 20 世纪 60 年代的文本编辑器 TECO (Text Editor and Corrector) 的基础上发展起来的, 并扩展了其中的一系列命令和宏。

emacs 编辑器用 C 语言编写, 包括一个完整的 Lisp 解释器, 完全支持 X Window 系统和鼠标, 支持世界上的主要语言。如今, emacs 正不断朝着所见即所得 (what you see is what you get) 字处理程序的方向发展, 致力于为初学者提供更便捷的服务, 已经完全超越了最初的 TECO。

emacs 能够为各种编程语言提供语法上的帮助。emacs 自带交互教程以及非常完备的在线文档, 可以在 emacs 的默认启动屏幕中调出。





11.1 emacs 概述

emacs 是一个屏幕编辑器，编辑文本直接显示在屏幕上，当输入命令或插入新文本时，显示的内容即相应地改变。但与 vim 相比，vim 更适合于完成小型任务，如邮件的快速回复、系统配置的简单修改等，而 emacs 更适合于处理复杂事务，如同时修改多个文件、调用其他程序等。

与 vim 不同的是，emacs 不需要在插入模式和命令模式间进行切换，所以 emacs 被称为无模式编辑器。emacs 的命令通常需要配合使用 Ctrl 键或其他特殊约定的键（如 Alt 键），因此不需要对模式进行追踪，即不需要确认当前是处于命令模式还是插入模式。由于 emacs 不使用普通字符作为命令，所以可以简便地将普通字符插入到正在编辑的文本中。对大多数用户来说，这种操作更方便，也更自然。

对缓冲区内容的移动和修改，emacs 与 vim 类似，也具有丰富的、可扩展的命令集合，而且这一命令集合可以根据需要随时修改和定制，甚至任何一个键都可以与任何命令绑定。键的绑定通常保存在 emacs 的初始化文件中，在某个会话期间，也可以交互地动态改变。本章描述的所有绑定键都是当前 GNU emacs 版本下标准的绑定键，该版本还可以支持许多可视化和面向鼠标的一些功能。

emacs 与 vim 最大的区别是 emacs 允许用户使用 Lisp 编写新命令或者重写原来的命令。Stallman 称此功能为联机扩展，但是只有熟练掌握 Lisp 的专业人员才能使用该功能。更常见的方式是向 emacs 文件中添加许多额外的调试命令，这些命令在 emacs 启动时被加载。

11.1.1 emacs 简介

1956 年，MIT 的 John McCarthy 开发了 Lisp 语言，Richard Stallman 和 Guy Steele 作为 MIT Lisp 组织的成员于 1975 年合作开发了 emacs 编辑器。利用 Lisp 编程语言的优势，emacs 更倾向于对功能的定制和扩展。经过不断的努力，emacs 的功能也逐渐强大，内置了收发电子邮件、执行 Shell 命令、编译程序并调试错误以及游戏等工具。

随着时间的推移，emacs 不断得到发展，经历了前后二十多次重大的修订后，如今已成长为 GNU 的主流版本。目前，emacs 已经移植到 MS-DOS、Windows 95 及 Windows NT 环境中。

作为一个区分语言的编辑器，emacs 具有一些可以开启/关闭的特殊功能，可以用来编辑文本、nroff、Tex、Lisp、C 和 Fortran 等文件。

emacs 还有一个具有更好图形界面的版本 xemacs（其继承自 emacs 19），如图 11-1 所示。可以通过 <http://www.xemacs.org> 访问 xemacs 主页。


```

XEmacs <insert spiffy graphic logo here>
XEmacs 21.4 (patch 12) "Portable Code" (linux, Mule) of Thu Feb 20 2003 on porky
^C-^ means the control key, ^M-^ means the meta key

Distribution, copying license, warranty:

Please visit the XEmacs website at http://www.xemacs.org !

M-? C-w: XEmacs comes with ABSOLUTELY NO WARRANTY
M-? C-l: conditions to give out copies of XEmacs
M-? C-d: how to get the latest version

---
Copyright (C) 1985-2001 Free Software Foundation, Inc.
Copyright (C) 1990-1994 Lucid, Inc.
Copyright (C) 1993-1997 Sun Microsystems, Inc. All Rights Reserved.
Copyright (C) 1994-1996 Board of Trustees, University of Illinois
Copyright (C) 1995-2001 Ben Wing

Zh-GB/EUC-----XEmacs: *scratch* (Lisp Interaction)-----All-----
Loading /usr/share/xemacs/site-packages/lisp/site-start.d/w3m-init.el...done

```

图 11-1 xemacs 主界面

11.1.2 启动和退出

emacs 编辑器具有许多功能，其完整使用手册有三十多章。然而，用户仅需使用一个相对较小的命令子集就能完成很多工作。例如，输入下面的命令，使用标准的 emacs 配置来编辑文件 example。

```
#emacs -nw -q example
```

其中，选项“-nw”表示不使用 X 接口，选项“-q”表示不从用户的主目录下读取“~/.emacs”或“default.el”初始化文件，而使用标准的、未定制的 emacs。有时这对初学者和那些想绕过 emacs 初始化文件的用户来说很有用。该命令在启动 emacs 时，如果文件 example 已存在，将在窗口中显示该文件的内容；如果不存在此文件，emacs 将显示空白屏幕，屏幕底部显示“New file”（如图 11-2 所示）。

从图 11-2 可以看到，屏幕启动时具有一个单独的窗口。窗口有上下两个反白显示的文本条。屏幕顶部的文本条列出了可用选项，但它不是菜单，不能用鼠标或别的方式访问。

窗口底部反白显示的文本条，被称为模式行（mode line）。模式行显示的信息包括：文件修改指示符、当前文件名称、模式说明、当前光标位置指示符、缓冲区位置。

- 文件修改指示符是靠近最左边的两个星号字符。星号字符表示文件中的数据已经改变。如果文件没有改变，看到的将会是连字号而不是星号。
- 文件名对应磁盘上的文件名称。
- 光标位置指示符显示当前光标所在行的行号。当浏览整个文档时，光标位置指示符会不断地改变。

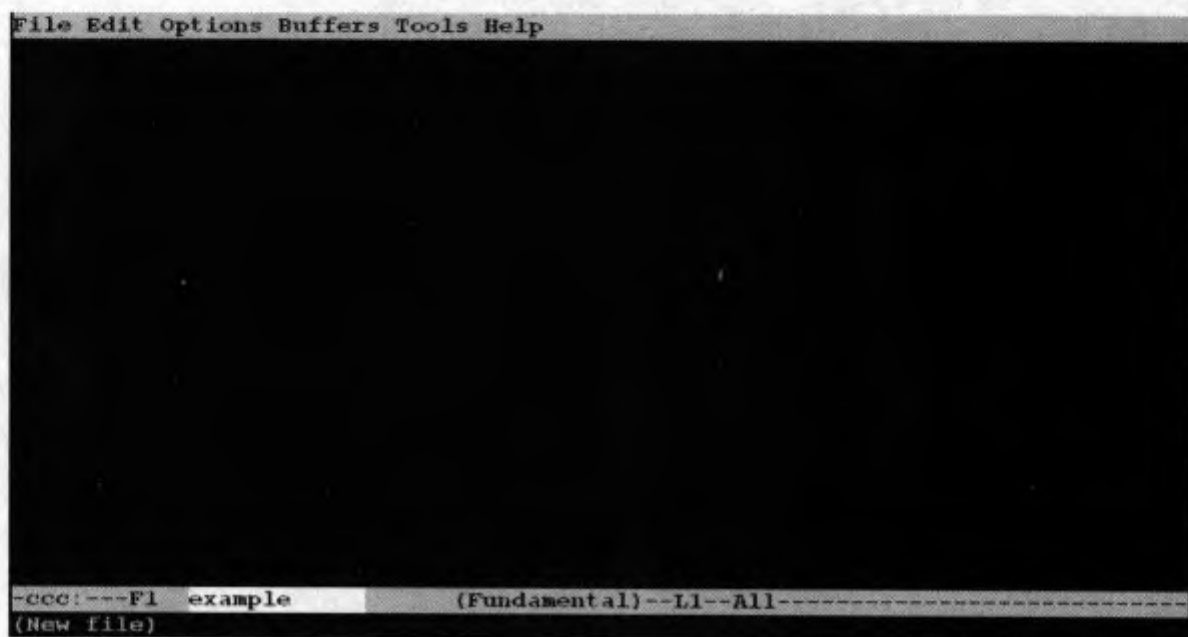


图 11-2 使用标准配置启动 emacs

- 缓冲区位置给出了当前窗口距离缓冲区底部的距离。如果窗口内的所有文本可见，则缓冲区位置显示为“**All**”。对于较大的文档，如果只有文件首行可见，显示为“**Top**”；如果文件最后一行可见，显示为“**Bot**”；如果文件的中间部分在当前窗口，则显示百分比。当打开多个窗口时，每个窗口将对应一个模式行。

在屏幕的底部，emacs 留有一个单独的行，被称为回显区（echo area）或小缓冲区（mini buffer），也称为状态行，用来显示较短的消息或特殊的单行命令。

退出 emacs 的命令由双键序列 Ctrl+X 和 Ctrl+C 组成。在 X 环境中，单击菜单栏中的【File】菜单，然后选择【Exit Emacs】也可退出 emacs，如图 11-3 所示。如果在编辑会话期间用户修改了文件内容，那么退出 emacs 时，emacs 会询问是否保存所做的修改。

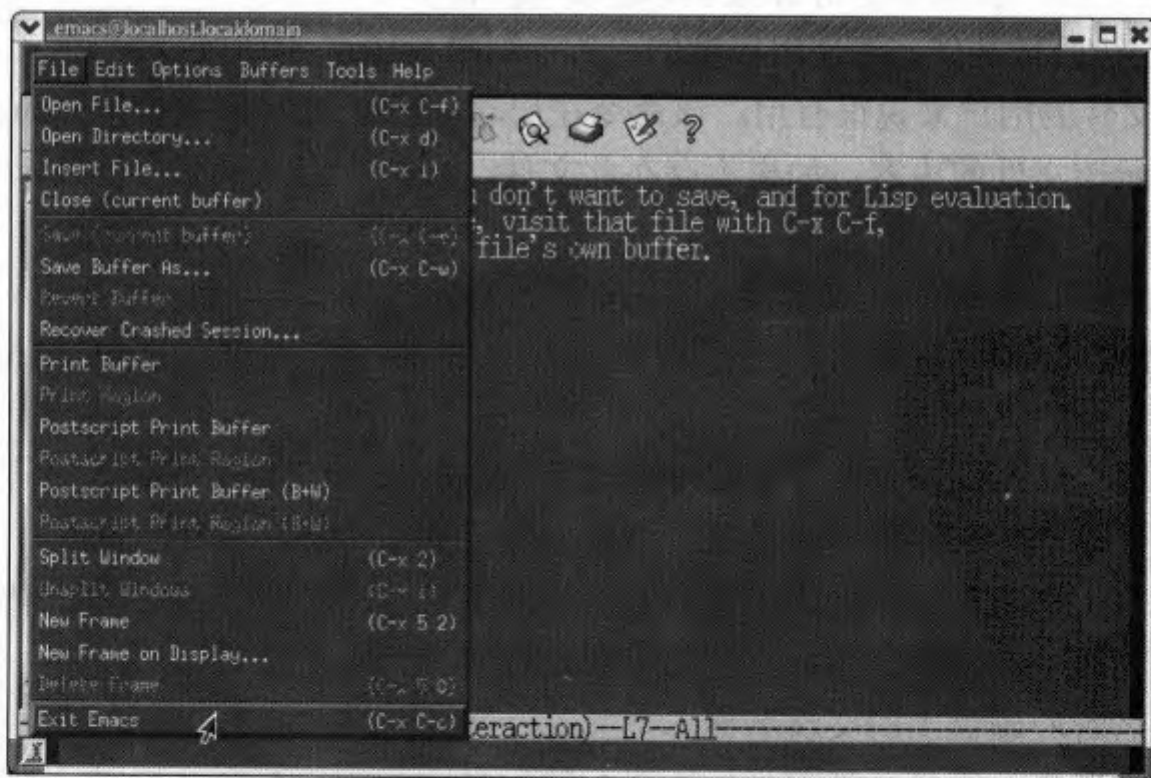


图 11-3 退出 emacs



按下 Ctrl+G 组合键，可以将已经输入一半的命令取消，或者停止正在运行的命令。在回显区，emacs 编辑器将显示 “Quit” 并等待下一个命令的输入。

11.1.3 新建一个文件

emacs 有文本模式和帮助模式之分。如果在启动 emacs 时提供了一个文件名，emacs 就进入文本模式，可以打开并编辑文本。如果指定的文件已经存在，加载后，该文件内容将显示在缓冲区窗口中，文本的首行显示在窗口顶部。如果该文件不存在，就加载一个空缓冲区，在回显区将出现一个新文件消息(即“New file”)。如果在启动 emacs 时没有提供文件名，emacs 就在帮助模式中打开。例如，在 X 环境下 (Red Hat Linux 9 图形环境中)，直接在命令行执行 emacs 命令，如下所示。

```
#emacs
```

执行该命令后将出现如图 11-4 所示界面，由于没有指定打开的文件名，emacs 进入帮助模式。

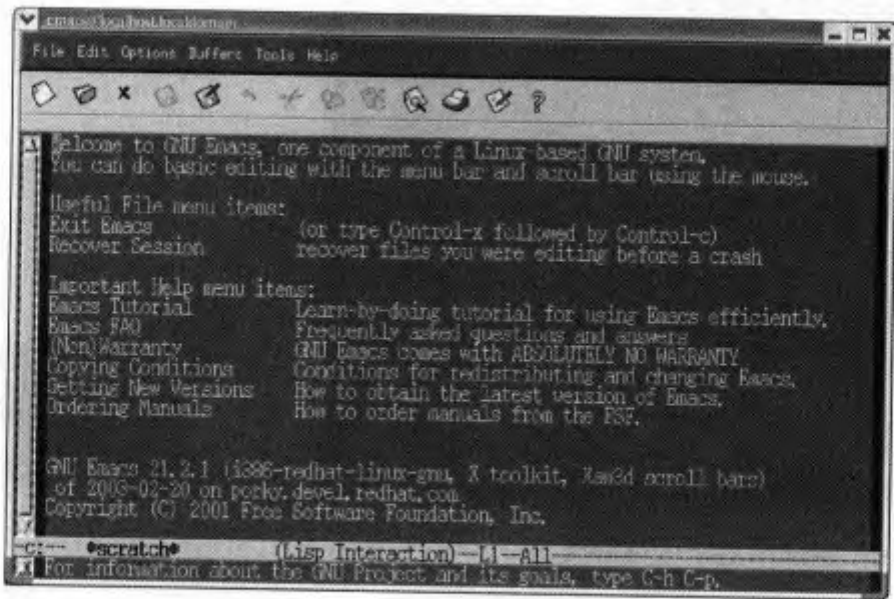


图 11-4 emacs 帮助模式

在图 11-4 所示界面中，按任意键或单击鼠标将进入文本编辑模式，如图 11-5 所示。输入文本仅需在文本模式下从键盘键入数据。要删除数据，可以使用删除键 (Delete 键)。

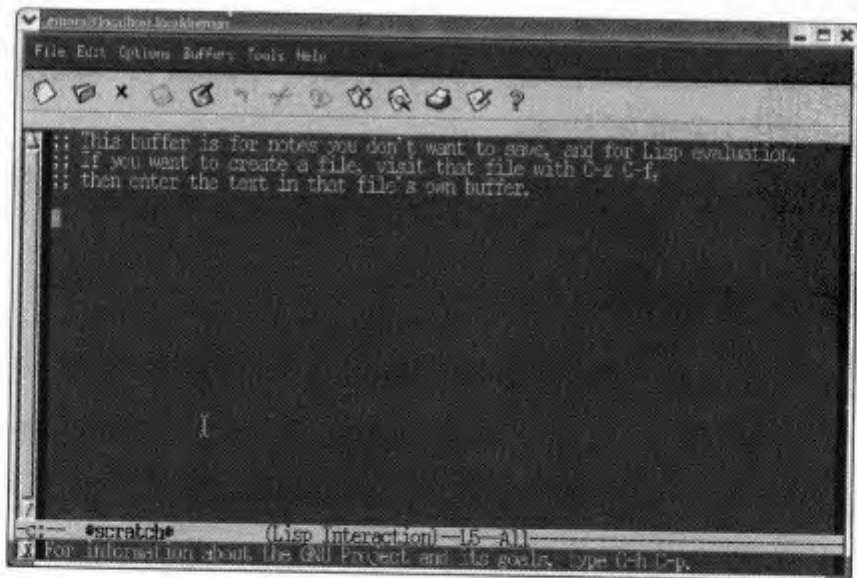


图 11-5 emacs 文本模式



11.1.4 保存文件

当使用 emacs 编辑文本时，应养成不定期地对文档进行保存的良好习惯，以免在出现系统崩溃等情况时造成重大损失。

emacs 保存命令很简单，依次键入 Ctrl+x 和 Ctrl+s 即可，这样，缓冲区内容将被保存到与其关联的文件。在 X 环境下，使用【File】菜单中的【Save】选项也可保存 emacs。

当退出 emacs 时，如果从上次保存之后又进行了修改，退出时将出现图 11-6 所示的对话框，要求用户对所做的修改进行处理。其中，【Yes】表示保存后退出；【No】表示放弃修改并退出，即缓冲区关联文件的内容将不会改变，会话期间的工作将被完全丢弃；【Quit】表示取消；【Save Buffer And Quit】表示保存缓冲区并退出；【Save All Buffers】表示保存所有缓冲区；【Display The Current Buffer】表示显示当前缓冲区内容。

如果在图 11-6 所示对话框中选择了【No】，将进一步弹出确认对话框，如图 11-7 所示。

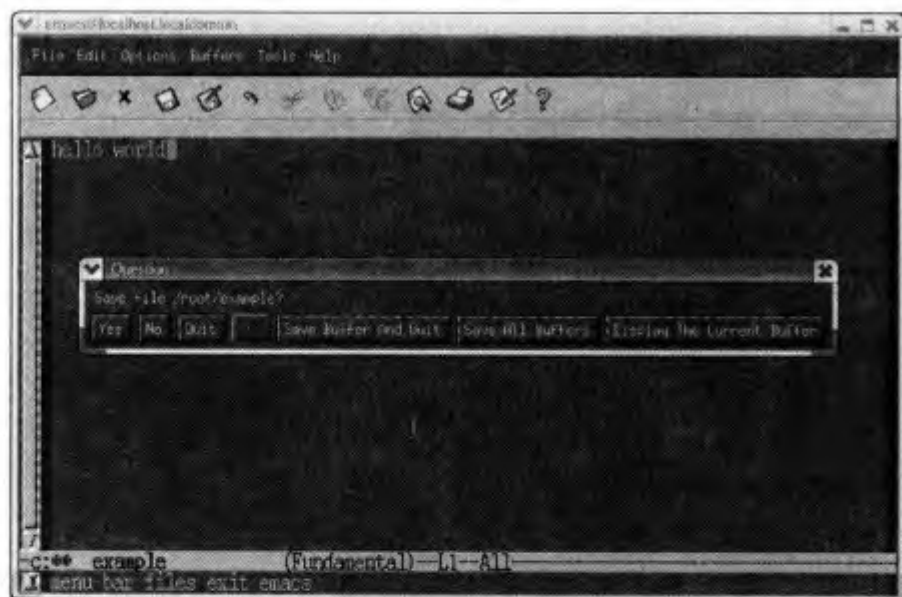


图 11-6 退出前提示保存修改



图 11-7 确认放弃修改

11.2 文件编辑

emacs 与 Windows 中的 Word 在对文本的编辑操作方面有许多相似之处。熟悉 Word 的用户可以很顺利地在 emacs 中完成编辑任务。

emacs 中定义了许多编辑命令，熟练掌握这些命令可以进一步加快编辑操作。

11.2.1 按键的表示与使用

为了区分命令和文本，emacs 使用命令键。emacs 有两种命令键：控制字符和元字符。控制字符通常使用键盘上的 Ctrl 键，元字符通常使用 Alt 键。当使用命令键时，emacs 可以识别它并把下一个键作为命令而不是文本，即命令键把编辑器置于命令模式而不是文本模式。

emacs 命令集中的大多数命令都是以非打印字符 Ctrl 或 Alt 形式存在，大多数 emacs 文档，包括联机帮助一般用大写字母 C 和 M 分别代表 Ctrl 和 Alt 键。一些最常用的特殊键用一

些相对简单的缩略词表示，如 RET (RETURN)、LFD (LINEFEED)、DEL (DELETE)、ESC (ESCAPE)、SPC (SPACE) 和 TAB。表 11-1 中显示了一些 emacs 键示例。

表 11-1		emacs 键表示	
字 符		emacs 中的表示	
a		a	
Shift +a		A	
Ctrl+a		C-a	
Ctrl+A		C-a ， 与 Ctrl+a 等价	
Alt+a		M-a	
Alt+A		M-A	
Ctrl+Alt+a		C-M-a	
Alt+Ctrl+a		M-C-a	

11.2.2 光标的移动

可以使用鼠标或直接使用键盘上的方向键定位光标。如果没有方向键，可以使用 emacs 命令。根据指定的移动单位（如字符、字、行、句子、段落等），可以将光标在文本中向前或向后移动。在任何光标移动命令之前可以加上一个重复因子（Ctrl+U+数值），使得光标在文本中移动由数字指定的文本单位数。

- 按字符移动光标。

按方向键中的右键或 Ctrl+F 组合键，光标将向前（或者说是向右）移动一个字符。如果光标位于行的末尾，则该命令将自动换行到下一行。配合使用“Ctrl+U”键可以一次移动多个字符。例如，命令“Ctrl+U 5 Ctrl+F”将光标向右移动 5 个字符。

按下方向键中的向左键或 Ctrl+B 组合键，光标将向后（或者说是向左）移动一个字符。命令“Ctrl+U 7 Ctrl+B”将向左移动 7 个字符。

- 按字移动光标。

使用 Alt+f 组合键，光标将向前（向右）移动一个字。光标停留在它原来所在字的第一个不属于该字的字符上。命令“Ctrl+U 4 Alt+f”将光标定位到向右第 4 个字末尾的空格处。

按下 Alt+b 组合键，光标将向后移动一个字，光标停留在它原来所在字的第一个字母上。如果光标原来就在某个字的第 1 个字母上，那么该命令 Alt+b 将把光标移到前一个字的第一个字母上。

- 按行移动光标。

按下 Ctrl+A 组合键，光标将被移到它所在行的行首。按下 Ctrl+E 组合键光标将被移到它所在行的行尾。按下方向键中的向上键或 Ctrl+P 组合键，光标将被移到其所在行正上方一行中的相应位置。按下方向键中的向下键或者 Ctrl+N 组合键，光标将被移到其所在行正下方一行中的相应位置。

类似于其他光标移动命令，在 Ctrl+P 和 Ctrl+N 组合键之前使用 Ctrl+U 命令，可以将光标移动多行。



- 按句子移动光标。

按下 Alt+a 组合键将把光标移到其所在句子的开始处，按下 Alt+e 组合键将把光标移到其所在句子的末尾处。

- 按段落移动光标。

按下 Alt+{组合键将把光标移到其所在段落的开始处，按下 Alt+}组合键将把光标移到其所在段落的末尾处。在这些命令之前使用 Ctrl+U 命令，可以将光标移动多行和多个段落。

- 按窗口位置移动光标。

Alt+r 组合键将把光标移到窗口中间一行的开始处。配合使用 Ctrl+U 命令，可以定位光标的位置。例如，命令“Ctrl+U 0 Alt+r”将把光标移到窗口最顶一行的开始处。其中的数字可以由用户希望移到的行号替换，如果使用减号“-”，则可以将光标移到窗口最后一行的开始处。

11.2.3 输入文本

光标位于窗口中或小缓冲区中。所有的输入和几乎所有的编辑工作都在光标处进行。当输入普通字符时，emacs 将在光标处插入输入的字符，已有文本将向右移动，这样字符就不会丢失。如果输入的字符太多使文本超过了窗口的右边界，那么 emacs 将在窗口的右端放置一个反斜杠“\”（对于 X 环境将在右侧显示一个向左的小箭头，在下一行左侧显示一个向右的小箭头，如图 11-8 所示），并自动换行到下一行。虽然反斜杠显示在屏幕上，但不是文件的内容，不会被打印。

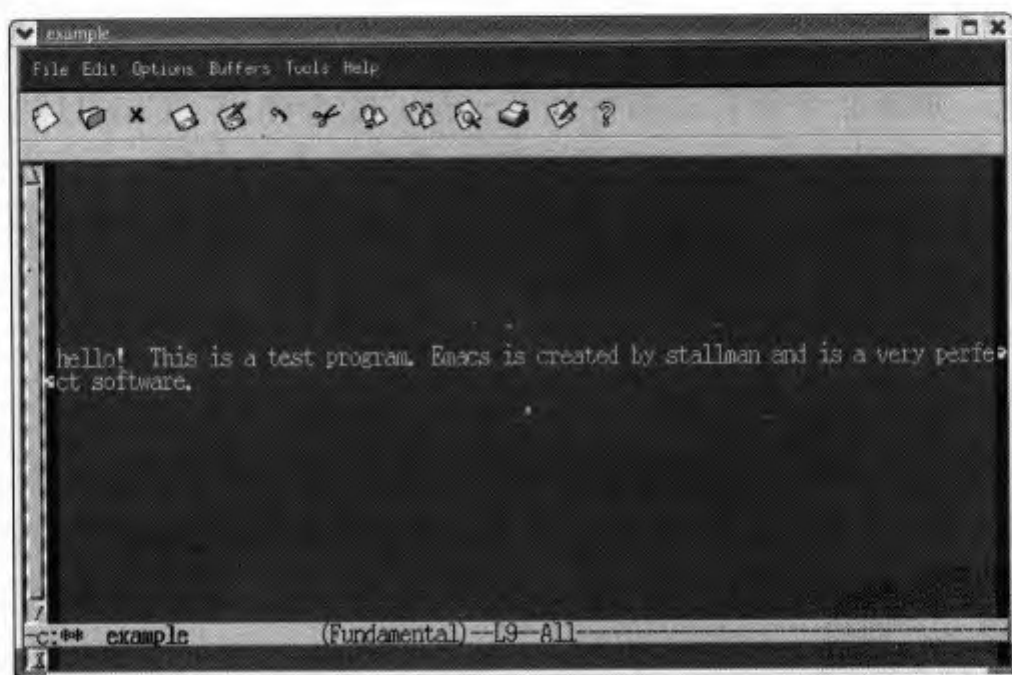


图 11-8 emacs 的自动换行

11.2.4 文本的删除

按下 Ctrl+D 组合键将删除光标所在字符，光标不动，行中的其他文本将向左移动来替代被删除的字符，效果与 DELETE 键一致。BACKSPACE 用于将光标左边的字符删除，光标和行中的其他文本将一起左移。要将某行和它上面一行连接起来，可以将光标定位到下面一行的第一个字符外，然后按下 BACKSPACE 键即可。

11.2.5 查找和替换

在【Edit】菜单中选择【Search】|【Search...】可以完成查找操作，如图 11-9 所示。在回显区输入待查找的字符串（如“http”），按回车键后，光标将停留在匹配的位置。

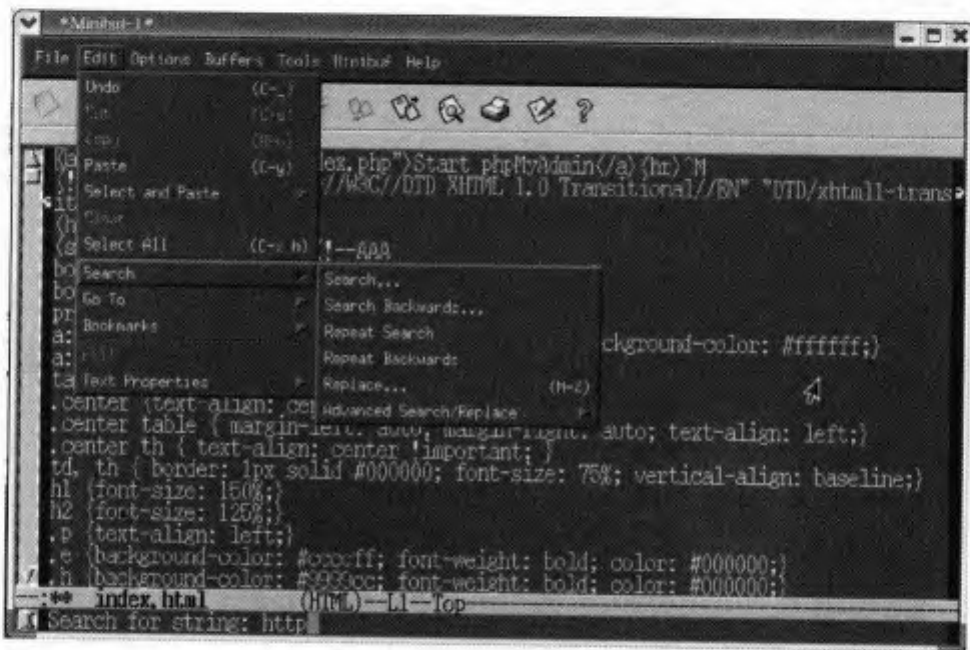


图 11-9 查找操作

在【Edit】菜单中选择【Search】|【Replace...】可以完成查找及替换操作，在回显区输入待查找的模式，回车后输入替换后的字符串即可，如图 11-10 所示。

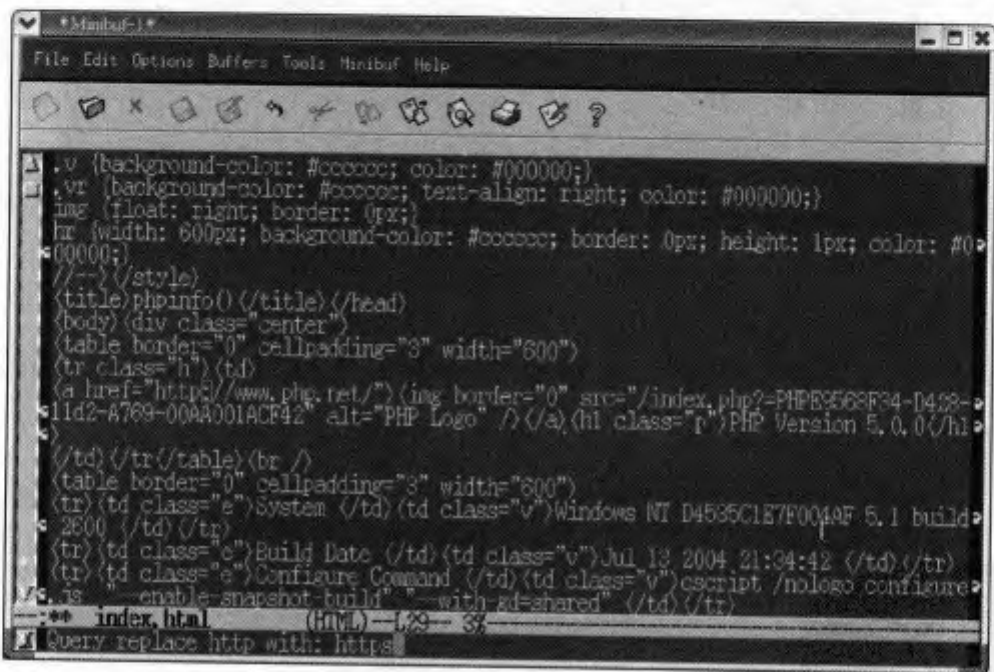


图 11-10 查找及替换

11.3 缓冲区和窗口

emacs 窗口是一个查看缓冲区的视窗。emacs 屏幕最初显示为一个单独的窗口，但此屏幕空间可以被分为两个或更多个窗口。在屏幕中，光标所在窗口即为当前窗口，通过窗口可以



查看当前缓冲区的内容。

11.3.1 拆分窗口

每个窗口一次可以查看一个缓冲区。在当前窗口中输入命令 **Ctrl+X b**，并输出缓冲区名称，可以切换到指定的缓冲区。与 Word 中的窗口拆分十分相似，多个窗口可以同时查看一个缓冲区，且每个窗口还可以查看同一个缓冲区的不同部分。



缓冲区可以在没有打开窗口的情况下存在。

拆分窗口可以使用命令 **Ctrl+X 2**，或者在 X 环境下单击【File】菜单中的【Split】选项。当前窗口将被拆分成上下两部分，新的窗口位于原来窗口的上面。例如，图 11-11 所示是对窗口进行了拆分，上下两个窗口分别显示了文件 **index.html** 的不同部分。

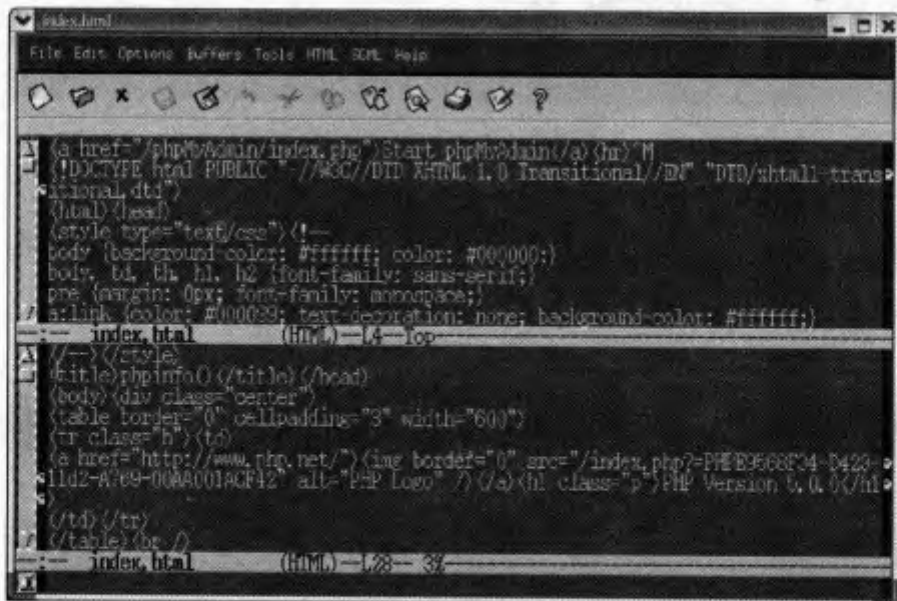


图 11-11 窗口的水平拆分

使用命令 **Ctrl+X 3** 可将当前窗口分成两个并排的窗口。例如图 11-12 中，将显示文件 **index.html** 的窗口进行了左右拆分。

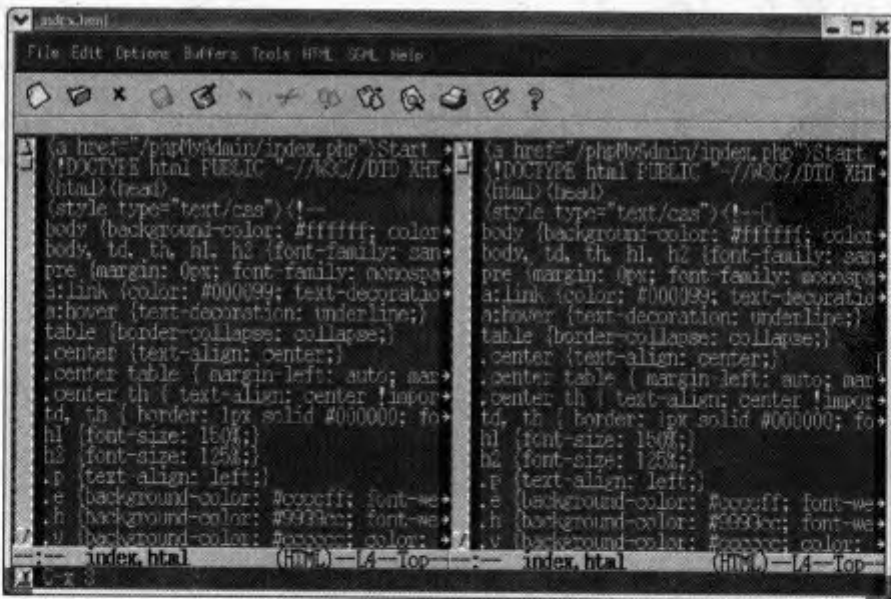


图 11-12 窗口的垂直拆分

联合使用水平拆分与垂直拆分可以将当前窗口细分为若干个小窗口，每个窗口中既可以显示同一文件，也可以显示不同文件。例如，图 11-13 中首先对窗口进行水平拆分，拆分成上下两部分，然后对下面窗口又进行了垂直拆分。上部窗口显示文件 index.html，左下部窗口显示文件 example，右下部窗口显示 Messages 信息。

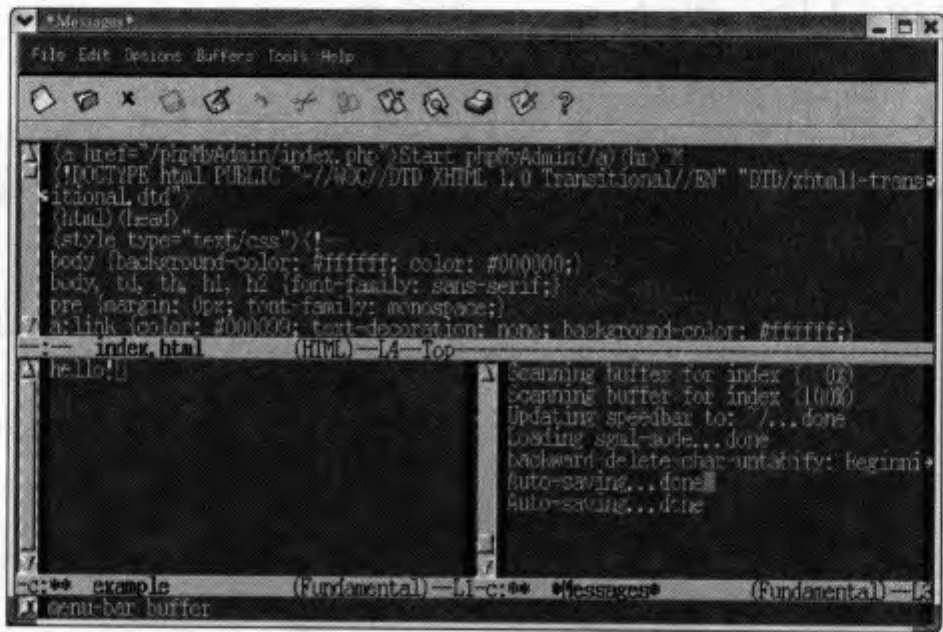


图 11-13 窗口的水平及垂直拆分

11.3.2 同时使用多个缓冲区进行工作

在编辑会话中，同时在缓冲区中载入两个文件要比退出 emacs 返回到 Shell，然后再启动 emacs 来编辑另一个文件更为方便。如果用户在使用 emacs 编辑某个文件的同时希望编辑另一个文件，则可以依次键入命令 Ctrl+X 和 Ctrl+F，将新文件载入到 emacs 的某个新缓冲区。在 X 环境中可以使用【File】菜单中的【Open File】选项。emacs 将提示用户输入新文件的名字，emacs 将给出用户要输入文件名的部分路径，该路径通常是用户的当前工作目录。用户输入文件名之后，emacs 将新文件读入到某个新缓冲区，并在当前窗口显示该缓冲区内容。

例如如图 11-14 中，通过执行窗口拆分操作将当前窗口分为上下两部分，在上部窗口载入了文件 example，而在下部窗口中载入了文件 index.html。

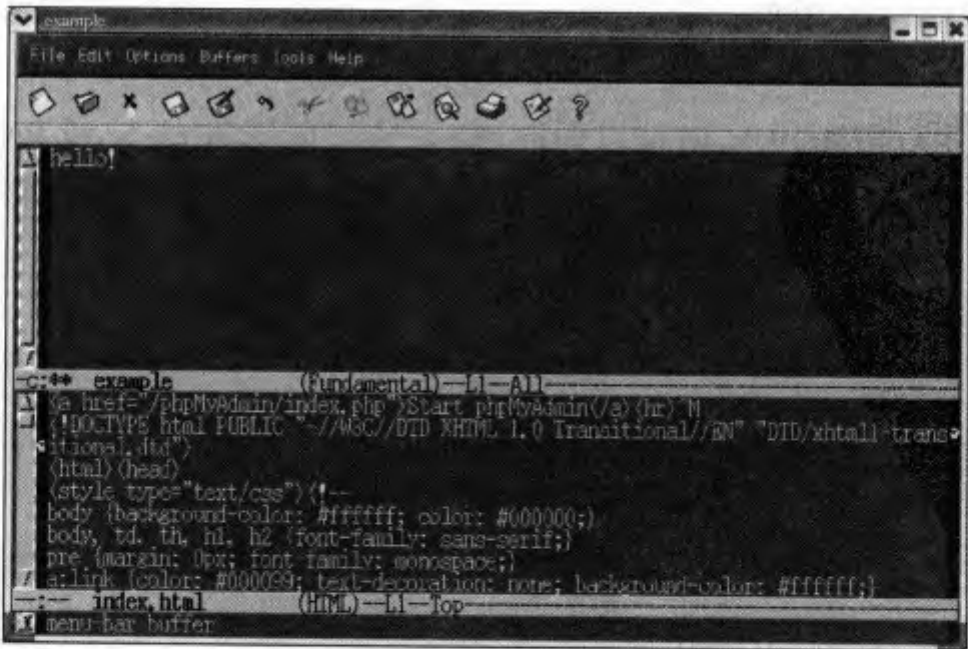


图 11-14 同时编辑多个文件



通过【Buffer】菜单可以切换当前使用的缓冲区。单击【Buffer】菜单中的【List All Buffers】选项，可以显示当前的全部缓冲区列表，如图 11-15 所示。

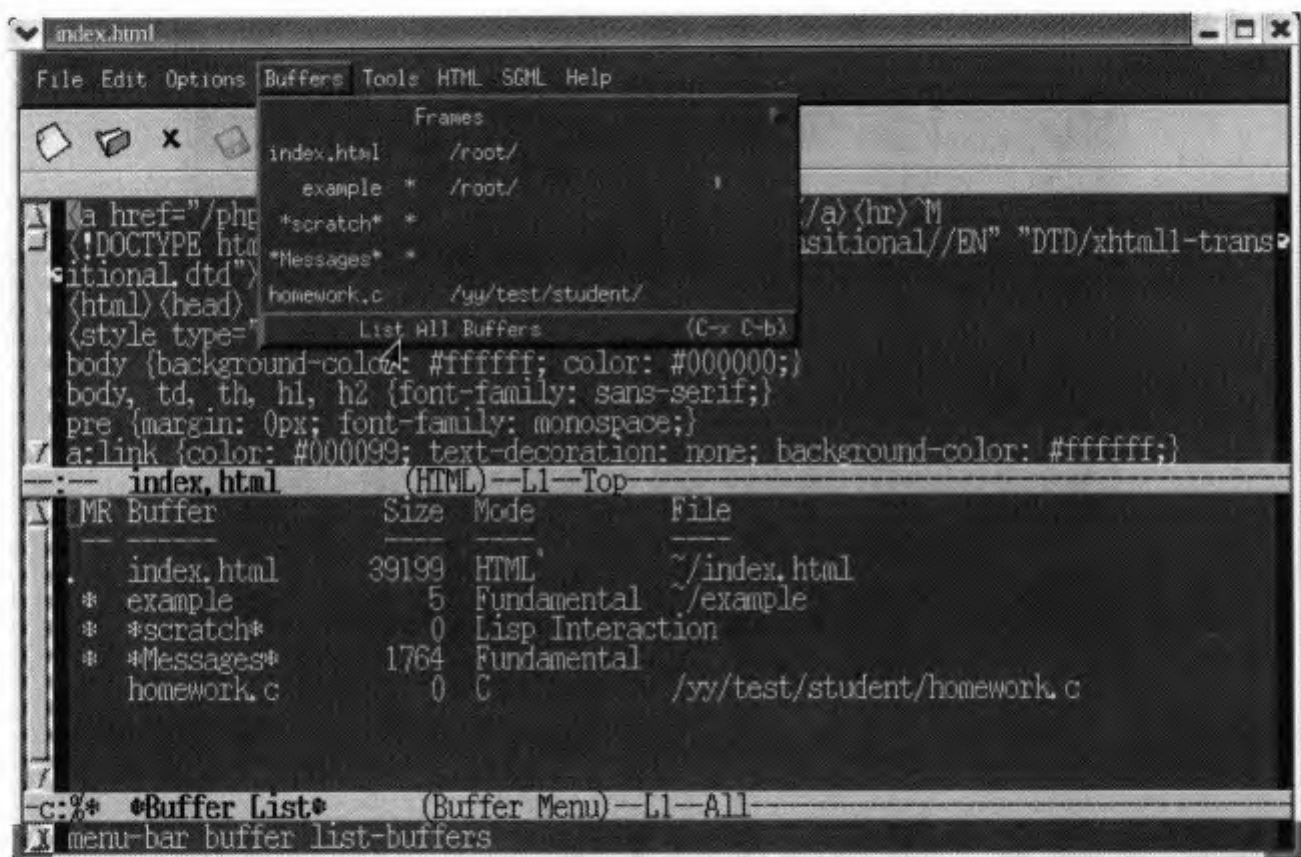


图 11-15 缓冲区列表

从图 11-15 可以看到，当前缓冲区中载入了“index.html”、“example”、“Scratch”、“Message”和“homework”。

11.3.3 在缓冲区中滚动

缓冲区通常要比窗口大许多，为了查看整个缓冲区的内容，需要向上、向下移动缓冲区的内容，以使用户感兴趣的部分显示在窗口中。向后滚动表示将文本内容向下移动，新行内容将出现在窗口的顶部，向前滚动表示文本内容向上移动，新行内容出现在窗口的底部。

使用 Ctrl+V 或 PAGEDOWN 键可以向前滚动一屏，使用 Alt+V 或 PAGEON 键可以向后滚动一屏。当屏幕上出现乱码时，可以使用 Ctrl+L 重绘屏幕。重绘屏幕将把光标移动到窗口的中心位置。

11.4 emacs 的扩展工具

在 emacs 的【Tool】菜单中有许多扩展工具，包括查找文件、编译文件、执行 Shell 命令、Debug、拼写检查、收发邮件、版本控制、显示日历等，如图 11-16 所示。

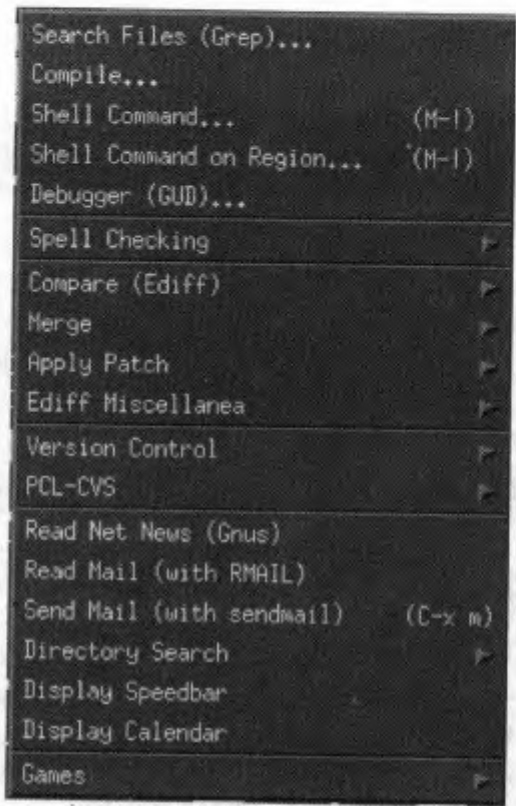


图 11-16 【Tool】菜单中的菜单项

11.4.1 显示日历

在 emacs 的【Tool】菜单中，单击【Display Calendar】选项可以在当前窗口中拆分出一个日历窗口，显示前后连续三个月的日历，且光标停留在当天位置处，如图 11-17 所示。

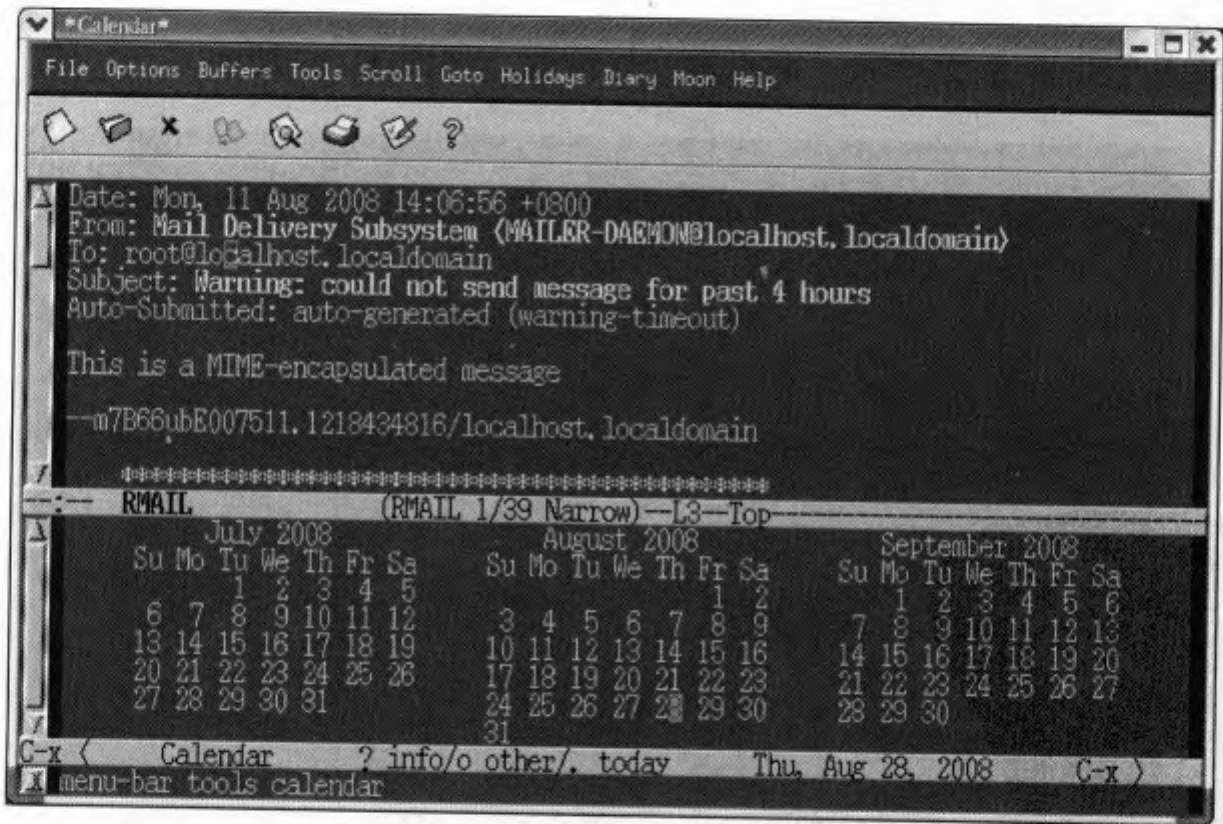


图 11-17 在 emacs 中显示日历

11.4.2 执行 Shell 命令

在 emacs 的【Tool】菜单中，单击【Shell Command...】选项，然后在回显区输入要执行



的 Shell 命令，当前窗口中拆分出一个 Shell 命令输出窗口，显示命令执行结果。
例如，下面示例中，在回显区输入 Shell 命令“ls-al”，显示结果如图 11-18 所示。

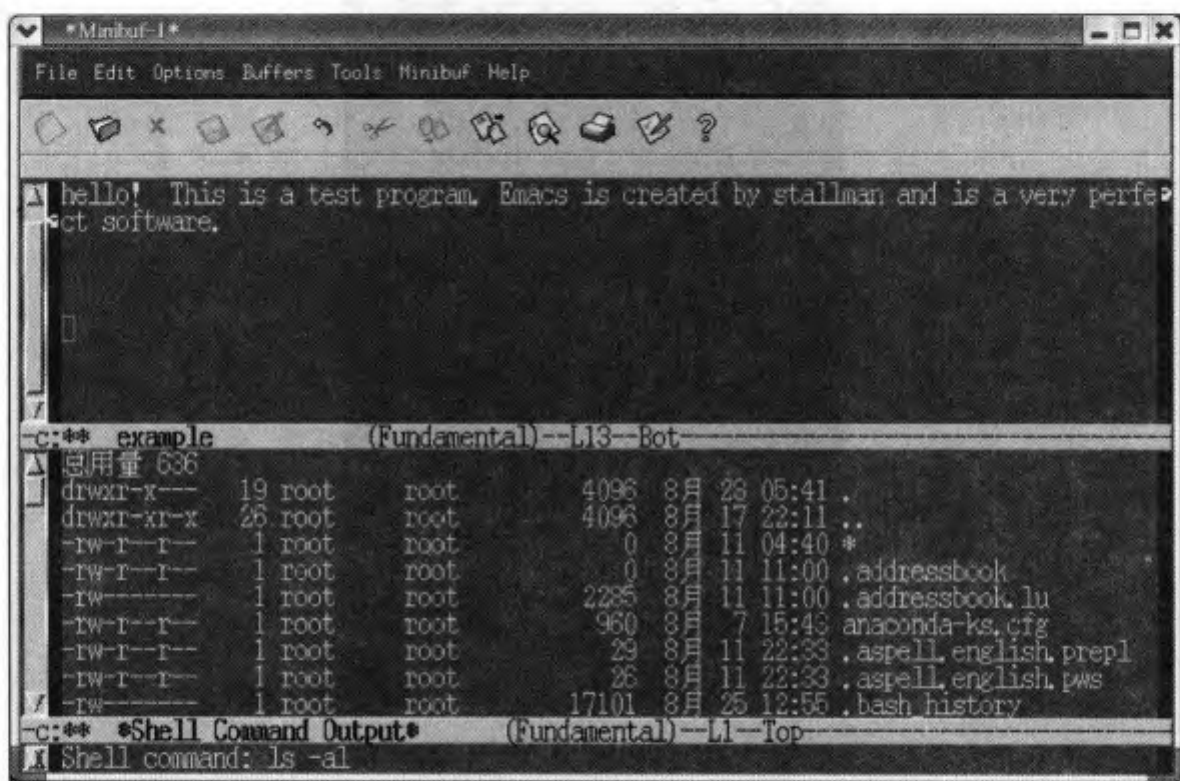


图 11-18 在 emacs 中执行 Shell 命令

11.4.3 调用游戏

在 emacs 的【Tool】菜单中，单击【Games】级联菜单可以看到 emacs 中内置的游戏列表，包括贪吃蛇、汉诺塔、黑盒子等。
例如，单击【Snake】菜单项，调用贪吃蛇游戏，如图 11-19 所示。

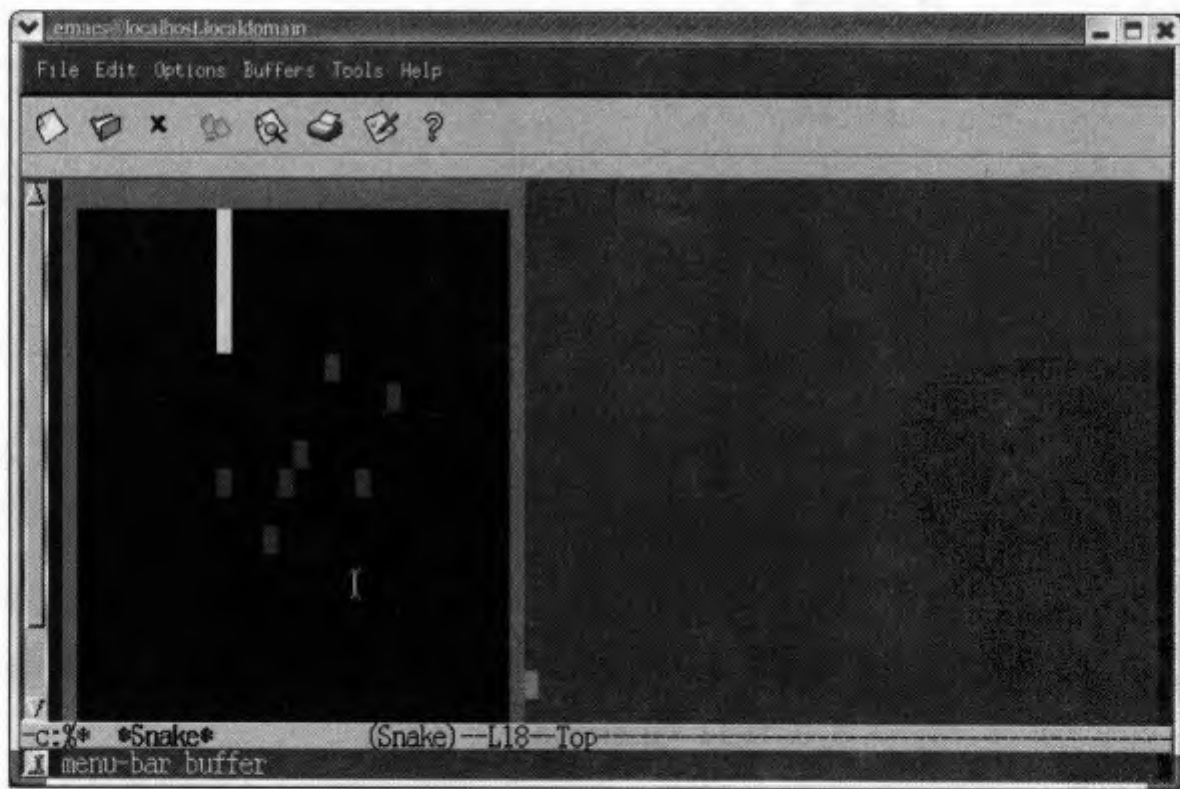


图 11-19 在 emacs 中运行游戏

游戏运行结束，可以看到游戏的得分情况，如图 11-20 所示。

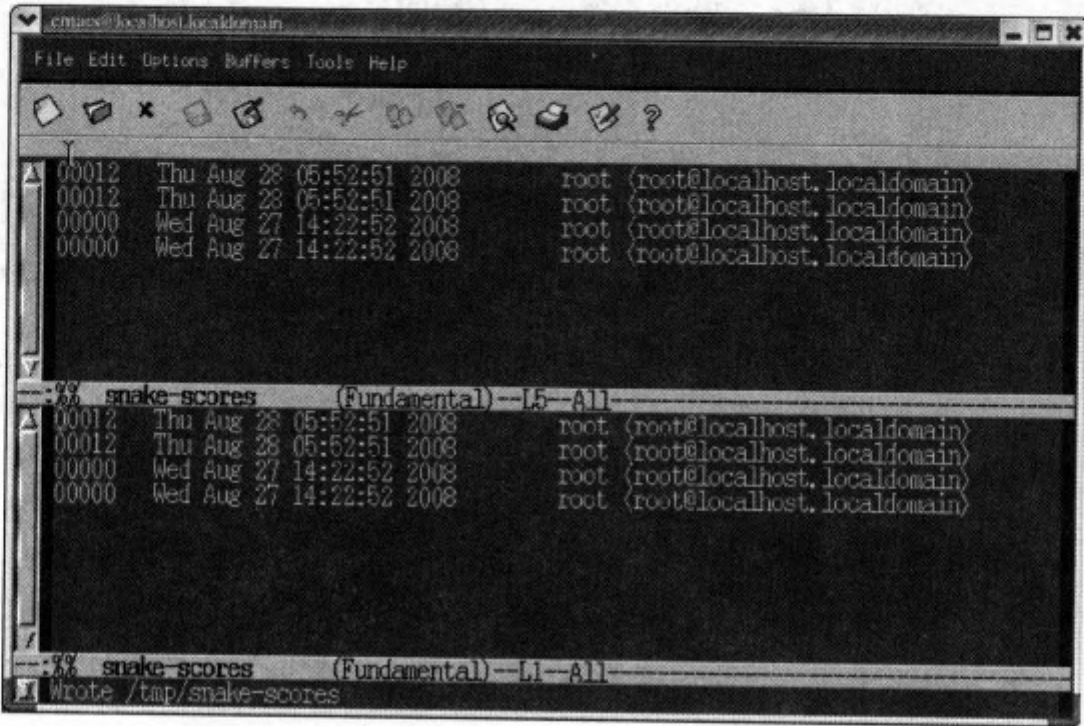


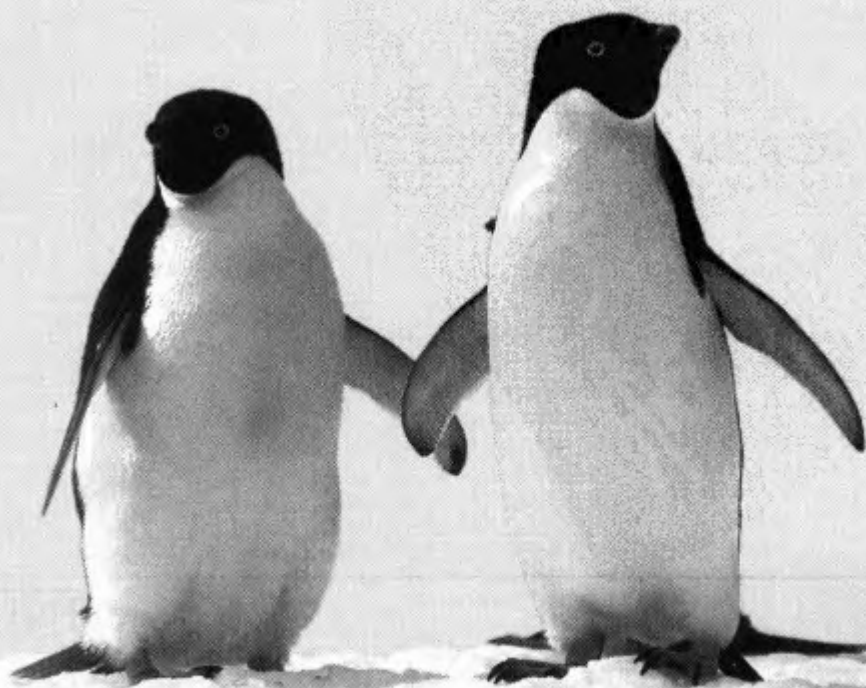
图 11-20 snake 的得分情况列表

LINUX

第12章

grep 命令

grep 是一个最初用于 UNIX 操作系统的命令行工具。在给出文件列表或标准输入后，grep 会对匹配一个或多个正则表达式的文本进行搜索，并只输出匹配的文本。正如 grep 的英文解释“Global search Regular Expression and Print out the line（全面搜索正则表达式并把行打印出来）”，它是一种强大的文本搜索工具，是 Linux/UNIX 中使用最广泛的命令之一。



12.1 grep 介绍

UNIX 的 `grep` 家族包括 `grep`、`egrep` 和 `fgrep`。`egrep` 和 `fgrep` 与 `grep` 基本上是一样的程序。通过调用不同的参数，`grep` 可以实现大多数 `grep` 家族具有的功能。在 Linux 中使用 GNU 版本的 `grep`，可以通过 `-E`、`-F` 命令行选项来使用 `egrep` 和 `fgrep` 的功能。读者可以在任何一个 GNU 镜像的 `/gnu/grep` 子目录中，下载稳定版本的 GNU `grep`。

12.1.1 grep 的含义

`grep` 的历史可以追溯到 UNIX 文本编辑器 `ed` 的类似命令，如下所示。

```
: g/re/p
```

这个命令搜索整个文件中与给定正则表达式 (`re`) 匹配的文本行，选项 “`g`” 表示文件中的所有行，选项 “`p`” 是指打印命令。

`grep` 命令功能强大，以下是一个常见的 `grep` 用法。

```
# grep beef menu.txt
```

在这个例子中，`grep` 会返回 “`menu.txt`” 中所有包含 “`beef`” 的文本行。需要注意的是，`grep` 不会返回匹配 “`Beef`” (字母 `B` 大写) 的文本行，因为 `grep` 在默认情况下大小写敏感。而通过选项设置，`grep` 能够改变或增加一些特定的功能，如下所示。

```
# grep -i beef menu.txt
```

这个命令会返回所有匹配 “`beef`”、“`Beef`”、“`BEef`” 或其他混合大小写的拼写。

使用 `grep` 能够不启动编辑器就可以运行查找匹配，并通过选项和模式的设置，实现包括显示不匹配的文本行，查找或排除搜索的文件，以及用不同的方式在输出中进行注释等多种灵活、独特的功能。

12.1.2 grep 的工作原理

对输入 (如文件、键盘) 的每一行文本，`grep` 命令进行如下操作。

- (1) 将该行文本读入相应的缓冲区，该缓冲区又被称为模式空间。
- (2) 对模式空间应用正则表达式进行对比。
- (3) 如果匹配，则该行信息从模式空间被复制到标准输出 (显示器)。

`grep` 命令会对每一行文本重复以上 3 个操作步骤，其语法格式如下。

```
grep [选项] 字符模式 [文件名 1, 文件名 2...]
```

简单地说，`grep` 在一个或多个文件 (文件名 1、文件名 2……) 中搜索字符模式。如果该模式包含空格，则必须加上引号。模式后的其他字符串将被看作文件名。搜索的结果输出到屏幕上，并不影响原文件的内容。

例如

```
# grep root /var/www/html/config.inc.php
$DBUSER="root";
```



```
# grep root /var/www/html/config.inc
grep: /var/www/html/config.inc: 没有那个文件或目录
```

上例中，grep 在/var/www/html/config.inc.php 文件中搜索模式“root”。搜索成功后，文件中包含“root”的一行被显示到屏幕上；如果没有找到该模式，则没有任何输出；如果搜索的文件是不合法的，则屏幕中将显示相应的错误提示。

例如

```
# ps -ef |grep mysql
root 2249      1  0 14:55 ?                00:00:00 /bin/sh /usr/bin/safe_mysqld --defaults-
file=/ etc/ my. cnf
mysql      2279  2249  0 14:55 ?                00:00:00 [mysqld]
root      2798  2733  0 15:20 pts/0    00:00:00 grep mysql
```

使用命令“ps -ef”显示系统中所有运行的进程。以上例子将 ps 命令的输出结果传送给 grep，grep 搜索所有包含 mysql 的行。

grep 命令支持很多正则表达式元字符，能够进一步定义搜索模式。此外，grep 命令还提供了多种选项，用以修改搜索或显示的方式。

12.2 grep 与正则表达式

grep 通过对多种正则表达式元字符的支持，能够灵活地定义搜索。正则表达式元字符集有两种版本：基本集和扩展集。grep 使用的是正则表达式元字符的基本集。egrep 使用的是正则表达式元字符的扩展集，将在 12.6 节进一步介绍。

12.2.1 grep 的正则表达式元字符

grep 的正则表达式元字符基本集包括^、\$、*、[]、[^]、\>和\<。表 12-1 所示为 grep 所使用的正则表达式元字符的基本集及说明。

表 12-1 grep 的正则表达式元字符及说明

元 字 符	说 明
^	行首定位符，用于指定匹配字符串的头部。例如，'^grep'匹配所有以 grep 开头的行
\$	行尾定位符，用于指定匹配字符串的尾部。例如，'grep\$'匹配所有以 grep 结尾的行
.	匹配任意一个字符。例如，'g.p'匹配所有字符串的第一个字符是 g，紧跟 2 个字符，最后以 p 结尾
*	匹配 0 个或多个前导字符。例如，'*grep'匹配所有一个或多个空格后紧跟 grep 的行
[]	表示匹配指定范围内的其中一个字符。例如，'[Gg]rep'匹配 Grep 和 grep
[^]	匹配不在指定范围内的字符。例如，'[^A-FH-Z]rep'匹配不包含以 A-F 和 H-Z 的一个字母开头，其后紧跟 rep 的行
\<	词首定位符。例如，'\<grep'匹配包含以 grep 开头的单词的行

续表

元 字 符	说 明
\>	词尾定位符。例如，'grep\>'匹配包含以 grep 结尾的单词的行
x\{m\}	重复字符 x，m 次。例如，'o\{5\}'匹配包含 5 个 o 的行
x\{m,\}	重复字符 x，至少 m 次。例如，'o\{5,\}'匹配至少有 5 个 o 的行
x\{m,n\}	重复字符 x，至少 m 次，不多于 n 次，例如，'o\{5,10\}'匹配 5~10 个 o 的行

下面举例说明如何使用这些元字符。例如，显示/etc/passwd 中以字符串 “root” 开头的行，命令如下所示。

```
# grep ^root /etc/passwd
root:x:0:0:root:/root:/bin/bash
```

例如，显示所有以 “h” 结尾的行，命令如下所示。

```
# grep h$ /etc/passwd
root:x:0:0:root:/root:/bin/bash
rpm:x:37:37::/var/lib/rpm:/bin/bash
amanda:x:33:6:Amanda user:/var/lib/amanda:/bin/bash
netdump:x:34:34:Network Crash Dump user:/var/crash:/bin/bash
mysql:x:27:27:MySQL Server:/var/lib/mysql:/bin/bash
postgres:x:26:26:PostgreSQL Server:/var/lib/pgsql:/bin/bash
pvm:x:24:24::/usr/share/pvm3:/bin/bash
teacher:x:500:500:~/home/teacher:/bin/bash
```

例如，使用 “.” 进行一个单字符匹配。从系统的词典中匹配所有以 “a” 开头且以 “e” 结尾，中间包含 2 个字符的单词，命令如下所示。

```
grep '\<a..e\>' /usr/share/dict/words
abbe
able
ache
acme
acne
acre
ague
aide
alee
aloe
ante
apse
axle
```

在使用方括号 “[” 和 “]” 时，能够匹配任何在方括号中的单个字符；如果方括号中的第一个字符是 “^”，则匹配任何不在方括号中的字符。例如，显示所有包含一个 “y” 或 “h” 字符的行，命令如下所示。

```
# grep [yh] /etc/group
sys:x:3:root,bin,adm
tty:x:5:
wheel:x:10:root
gopher:x:30:
nobody:x:99:
floppy:x:19:
sshd:x:74:
```



```
nfsnobody:x:65534:
apache:x:48:
mysql:x:27:
privoxy:x:73:
teacher:x:500:
```

例如，显示不包含字母 a~k，且后紧跟“pple”的单词，命令如下所示。

```
# grep [^a-k]pple /usr/share/dict/words
Doppler
supple
supplement
supplemental
supplementary
supplemented
supplementing
supplements
suppleness
topple
toppled
topples
```

若要匹配多个字符，则使用星号。例如，从系统的词典中选择所有以“c”开头且以“o”结尾的单词，命令如下所示。

```
# grep '\<c.*o\>' /usr/share/dict/words
calico
calypso
canto
cargo
casino
coco
commando
concerto
coo
cuckoo
```

而如果希望在一个文件或输出中找到包含星号字符的行，应加上引号，如下所示。

```
# grep '*' /etc/profile
for i in /etc/profile.d/*.sh ; do
```

此外，可以使用 **grep** 命令在几个文本文件中查找某一字符串。例如，在“/usr/src/linux/Documentation”目录下搜索带字符串“Custom”的文件，命令如下所示。

```
# grep Custom /usr/src/linux/Documentation/*
/usr/src/linux/Documentation/moxa-smartio: 3.5 Custom configuration
/usr/src/linux/Documentation/watchdog.txt: Custom Linux Driver And Program
Development
```

结果显示，文件“moxa-smartio”和“watchdog.txt”中，包含字符串“Custom”。

例如，显示所有包含每个字符串至少有 20 个连续字母的单词的行，命令如下所示。

```
# grep '[a-z]\{20\}' /usr/share/dict/words
antidisestablishmentarianism
electroencephalogram
electroencephalograph
electroencephalography
Mediterraneanization
Mediterraneanizations
Nondeterministically
```


12.2.2 grep 正则实例

接下来通过一个实例，进一步说明 `grep` 正则表达式的使用。以下是 `Weather.txt` 文件的内容。

```
# cat Weather.txt
Sep 26 Today      Partly cloudy.      High around 65.0F.Winds NNW at 5 to 10 mph
Sep 26 Tonight    Mostly cloudy skies. Low 53.0F.Winds WNW at 5 to 10 mph
Sep 27 Tomorrow    Mostly cloudy.      High around 70.0F.Winds W at 5 to 10 mph
Sep 27 Tomorrow night Mostly cloudy skies. Low 53.0F.Winds light and variable
Sep 28 Sunday      Sunshine.            Highs in the low 70s and lows in the low 50s
Sep 29 Monday      A few clouds.        Highs in the mid 70s and lows in the low 50s
Sep 30 Tuesday      Abundant sunshine.   Highs in the mid 70s and lows in the mid 50s
Oct 1 Wednesday    More sun than clouds. Highs in the mid 70s and lows in the upper 50s
Oct 2 Thursday      Sunshine.            Highs in the low 70s and lows in the mid 50s
Oct 3 Friday        Chance of showers.   Highs in the upper 60s and lows in the mid 50s
Oct 4 Saturday      Times of sun and clouds. Highs in the low 70s and lows in the mid 50s
Oct 5 Sunday        Chance of showers.   Highs in the low 70s and lows in the mid 50s
```

例如，匹配 `Weather.txt` 文件中所有包含正则表达式 `Sunshine` 的行，命令如下所示。

```
# grep Sunshine Weather.txt
Sep 28 Sunday      Sunshine.            Highs in the low 70s and lows in the low 50s
Oct 2 Thursday      Sunshine.            Highs in the low 70s and lows in the mid 50s
```

例如，匹配所有以 `W` 开头的文件中，包含正则表达式 `Monday` 的行。`W*` 表示所有以 `W` 开头的文件，在这个例子中，文件名是 `Weather.txt` 和 `WeatherReport`，命令如下所示。

```
# grep Monday W*
WeatherReport:Sep 29 Monday      Mostly cloudy.
Weather.txt:Sep 29 Monday        A few clouds.      Highs in the mid 70s and lows
in the low 50s
```

例如，匹配所有以 `S` 开头的行，本例中用到了 “`^`” 行首定位符，如下所示。

```
# grep '^S' Weather.txt
Sep 26 Today      Partly cloudy.      High around 65.0F.Winds NNW at 5 to 10 mph
Sep 26 Tonight    Mostly cloudy skies. Low 53.0F.Winds WNW at 5 to 10 mph
Sep 27 Tomorrow    Mostly cloudy.      High around 70.0F.Winds W at 5 to 10 mph
Sep 27 Tomorrow night Mostly cloudy skies. Low 53.0F.Winds light and variable
Sep 28 Sunday      Sunshine.            Highs in the low 70s and lows in the low 50s
Sep 29 Monday      A few clouds.        Highs in the mid 70s and lows in the low 50s
Sep 30 Tuesday      Abundant sunshine.   Highs in the mid 70s and lows in the mid 50s
```

例如，使用 `grep` 在 `Weather` 和 `Weather.txt` 文件中搜索 `Monday`，如下所示。

```
# grep Monday Weather Weather.txt
grep: Weather: 没有那个文件或目录
Weather.txt:Sep 29 Monday        A few clouds.      Highs in the mid 70s and lows in the low 50s
```

在本例中，`Weather` 和 `Weather.txt` 是文件名，`Monday` 是字符模式。`Monday` 和 `Weather` 之间的空格，将导致 `grep` 在 `Weather` 和 `Weather.txt` 中搜索 `Monday`。若要搜索包含 “`Monday Weather`” 的行，则需要加上引号，如下所示，`grep` 将在 `Weather.txt` 中搜索包含 “`Monday Weather`” 的行。

```
# grep 'Monday Weather' Weather.txt
```

例如，匹配包含第一个字符是 `5`，后紧跟一个点，接着是任意字符串的行，命令如下所示。

```
# grep '5\...' Weather.txt
```



```
Sep 26 Today          Partly cloudy.          High around 65.0F.Winds NNW at 5 to 10 mph
```

需要注意的是，除非点的前面有一个反斜杠，否则它就表示任意字符。若加了反斜杠，点就不再特殊，而仅表示一个点。

例如，匹配包含字符串“.0”的行，命令如下所示。

```
# grep '\.0' Weather.txt
Sep 26 Today          Partly cloudy.          High around 65.0F.Winds NNW at 5 to 10 mph
Sep 26 Tonight        Mostly cloudy skies.   Low 53.0F.Winds WNW at 5 to 10 mph
Sep 27 Tomorrow        Mostly cloudy.         High around 70.0F.Winds W at 5 to 10 mph
Sep 27 Tomorrow night Mostly cloudy skies.   Low 53.0F.Winds light and variable
```

例如，匹配所有以 S 或 O 开头的行。“^”锚定行的开头，中括号内任何一个字符都将被匹配，如下所示。

```
# grep '^[SO]' Weather.txt
Sep 26 Today          Partly cloudy.          High around 65.0F.Winds NNW at 5 to 10 mph
Sep 26 Tonight        Mostly cloudy skies.   Low 53.0F.Winds WNW at 5 to 10 mph
Sep 27 Tomorrow        Mostly cloudy.         High around 70.0F.Winds W at 5 to 10 mph
Sep 27 Tomorrow night Mostly cloudy skies.   Low 53.0F.Winds light and variable
Sep 28 Sunday          Sunshine.              Highs in the low 70s and lows in the low 50s
Sep 29 Monday          A few clouds.          Highs in the mid 70s and lows in the low 50s
Sep 30 Tuesday          Abundant sunshine.    Highs in the mid 70s and lows in the mid 50s
Oct 1 Wednesday        More sun than clouds.  Highs in the mid 70s and lows in the upper 50s
Oct 2 Thursday          Sunshine.              Highs in the low 70s and lows in the mid 50s
Oct 3 Friday           Chance of showers.     Highs in the upper 60s and lows in the mid 50s
Oct 4 Saturday          Times of sun and clouds. Highs in the low 70s and lows in the mid 50s
Oct 5 Sunday           Chance of showers.     Highs in the low 70s and lows in the mid 50s
```

例如，匹配第一个字母是大写 O，后面紧跟两个字母及一个空格，再跟 1~3 之间的一个数字的字符串的行，命令如下所示。

```
# grep '[O][a-z][a-z] [1-3]' Weather.txt
Oct 1 Wednesday        More sun than clouds.  Highs in the mid 70s and lows in the upper 50s
Oct 2 Thursday          Sunshine.              Highs in the low 70s and lows in the mid 50s
Oct 3 Friday           Chance of showers.     Highs in the upper 60s and lows in the mid 50s
```

例如，匹配所有包含一个或多个大写 N 的字符串的行，命令如下所示。

```
# grep 'NN*' Weather.txt
Sep 26 Today          Partly cloudy.          High around 65.0F.Winds NNW at 5 to 10 mph
Sep 26 Tonight        Mostly cloudy skies.   Low 53.0F.Winds WNW at 5 to 10 mph
```

例如，匹配所有包含每个字符串至少有 9 个连续字母的行，命令如下所示。

```
# grep '[a-z]\{9\}' Weather.txt
Oct 1 Wednesday        More sun than clouds.  Highs in the mid 70s and lows in the upper 50s
```

例如，匹配所有以 Sep 开始的单词的行，命令如下所示。

```
# grep '\<Sep' Weather.txt
Sep 26 Today          Partly cloudy.          High around 65.0F.Winds NNW at 5 to 10 mph
Sep 26 Tonight        Mostly cloudy skies.   Low 53.0F.Winds WNW at 5 to 10 mph
Sep 27 Tomorrow        Mostly cloudy.         High around 70.0F.Winds W at 5 to 10 mph
Sep 27 Tomorrow night Mostly cloudy skies.   Low 53.0F.Winds light and variable
Sep 28 Sunday          Sunshine.              Highs in the low 70s and lows in the low 50s
Sep 29 Monday          A few clouds.          Highs in the mid 70s and lows in the low 50s
Sep 30 Tuesday          Abundant sunshine.    Highs in the mid 70s and lows in the mid 50s
```

例如，匹配所有包含单词 High 的行，命令如下所示。


```
# grep '\<High\>' Weather.txt
Sep 26 Today      Partly cloudy.      High around 65.0F.Winds NNW at 5 to 10 mph
Sep 27 Tomorrow   Mostly cloudy.      High around 70.0F.Winds W at 5 to 10 mph
```

因为“\<”、“\>”分别锚定了单词的开头和结尾，因此，文件中包含单词“Highs”的行都未能匹配。

12.3 grep 的选项

grep 命令可以通过一些选项来定义搜索或搜索结果的显示方式。不同版本的 grep 所能支持的选项可能不同，在使用时可以通过检查 man 页面获得完整的选项列表。表 12-2 介绍了一些常用的 grep 选项，并举例说明了它们的用法。

表 12-2 常用 grep 选项及功能说明

元 字 符	说 明
-#	同时显示匹配行上下的#行，#表示数字
-b	在每行前打印字符偏移量
-c	只打印匹配的行数，不显示匹配的内容
-f File, --file=File	从 FILE 文件中提取模板。空文件中包含 0 个模板，所以什么都不匹配
-h, --no-filename	当搜索多个文件时，不显示匹配文件名前缀
-i, --ignore-case	忽略大小写差别
-q, --quiet	取消显示，只返回退出状态。0 则表示找到了匹配的行
-l, --files-with-matches	打印匹配模板的文件清单
-L, --files-without-match	打印不匹配模板的文件清单
-n, --line-number	在匹配的行前面打印行号
-s, --silent	不显示关于不存在或无法读取文件的错误信息
-v, --revert-match	反检索，只显示不匹配的行
-w, --word-regexp	如果被\<和\>引用，就把表达式作为一个单词搜索
-V, --version	显示软件版本信息

以下是 Weather.txt 文件的内容。

```
# cat Weather.txt
Sep 26 Today      Partly cloudy.      High around 65.0F.Winds NNW at 5 to 10 mph
Sep 26 Tonight    Mostly cloudy skies. Low 53.0F.Winds WNW at 5 to 10 mph
Sep 27 Tomorrow   Mostly cloudy.      High around 70.0F.Winds W at 5 to 10 mph
Sep 27 Tomorrow night Mostly cloudy skies. Low 53.0F.Winds light and variable
Sep 28 Sunday     Sunshine.           Highs in the low 70s and lows in the low 50s
Sep 29 Monday     A few clouds.       Highs in the mid 70s and lows in the low 50s
Sep 30 Tuesday    Abundant sunshine.  Highs in the mid 70s and lows in the mid 50s
Oct 1 Wednesday   More sun than clouds. Highs in the mid 70s and lows in the upper 50s
```



```
Oct 2  Thursday  Sunshine.           Highs in the low 70s and lows in the mid 50s
Oct 3  Friday    Chance of showers.  Highs in the upper 60s and lows in the mid 50s
Oct 4  Saturday  Times of sun and clouds.  Highs in the low 70s and lows in the mid 50s
Oct 5  Sunday    Chance of showers.           Highs in the low 70s and lows in the mid 50s
```

例如，使用选项“-n”在每个匹配的行前打印行号，如下所示。

```
# grep -n 'High' Weather.txt
1:Sep 26 Today      Partly cloudy.           High around 65.0F.Winds NNW at 5 to 10 mph
3:Sep 27 Tomorrow   Mostly cloudy.           High around 70.0F.Winds W at 5 to 10 mph
5:Sep 28 Sunday     Sunshine.                Highs in the low 70s and lows in the low 50s
6:Sep 29 Monday     A few clouds.            Highs in the mid 70s and lows in the low 50s
7:Sep 30 Tuesday    Abundant sunshine.       Highs in the mid 70s and lows in the mid 50s
8:Oct 1  Wednesday  More sun than clouds.    Highs in the mid 70s and lows in the upper 50s
9:Oct 2  Thursday   Sunshine.                Highs in the low 70s and lows in the mid 50s
10:Oct 3 Friday     Chance of showers.       Highs in the upper 60s and lows in the mid 50s
11:Oct 4 Saturday   Times of sun and clouds. Highs in the low 70s and lows in the mid 50s
12:Oct 5 Sunday     Chance of showers.       Highs in the low 70s and lows in the mid 50s
```

例如，使用选项“-i”关闭大小写敏感，可以匹配任意大小写的字符，命令如下所示。

```
# grep -i 'oct' Weather.txt
Oct 1  Wednesday  More sun than clouds.    Highs in the mid 70s and lows in the upper 50s
Oct 2  Thursday   Sunshine.                Highs in the low 70s and lows in the mid 50s
Oct 3  Friday     Chance of showers.       Highs in the upper 60s and lows in the mid 50s
Oct 4  Saturday   Times of sun and clouds. Highs in the low 70s and lows in the mid 50s
Oct 5  Sunday     Chance of showers.       Highs in the low 70s and lows in the mid 50s
```

例如，选项“-l”使得 `grep` 只显示匹配的文件名，并不显示匹配的行。该例中，文件 `WeatherReport` 和 `Weather.txt` 都包含 `Monday` 字符串，如下所示。

```
# grep -l Monday *
WeatherReport
Weather.txt
```

例如，选项“-c”使得 `grep` 只显示有多少行能够匹配，命令如下所示。

```
# grep -c mid Weather.txt
7
```



匹配的结果是包含字符模式的行数，而非字符模式出现的次数。在 `Weather.txt` 文件中，总共有 7 行包含 `mid`，而 `mid` 总共在该文件中出现了 8 次。

例如，使用选项“-V”显示 `grep` 的版本，命令如下所示。

```
# grep -V
grep (GNU grep) 2.5.1
```

```
Copyright 1988, 1992-1999, 2000, 2001 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

例如，选项“-#”能够显示匹配的该行以及上下各#行的内容，其中“#”表示数字。本例中，显示了匹配 `Thursday` 的行，及其上下各三行，如下所示。

```
# grep -3 Thursday Weather.txt
Sep 29 Monday      A few clouds.           Highs in the mid 70s and lows in the low 50s
```



```
Sep 30 Tuesday Abundant sunshine. Highs in the mid 70s and lows in the mid 50s
Oct 1 Wednesday More sun than clouds. Highs in the mid 70s and lows in the upper 50s
Oct 2 Thursday Sunshine. Highs in the low 70s and lows in the mid 50s
Oct 3 Friday Chance of showers. Highs in the upper 60s and lows in the mid 50s
Oct 4 Saturday Times of sun and clouds. Highs in the low 70s and lows in the mid 50s
Oct 5 Sunday Chance of showers. Highs in the low 70s and lows in the mid 50s
```

例如,“-b”选项在匹配行前打印字符偏移量,命令如下所示。

```
# grep -b Thursday Weather.txt
697:Oct 2 Thursday Sunshine. Highs in the low 70s and lows in the mid 50s
```

例如,使用“-v”选项显示所有不包含字符模式的行,如下所示。

```
# grep -v Oct Weather.txt
Sep 26 Today Partly cloudy. High around 65.0F.Winds NNW at 5 to 10 mph
Sep 26 Tonight Mostly cloudy skies. Low 53.0F.Winds WNW at 5 to 10 mph
Sep 27 Tomorrow Mostly cloudy. High around 70.0F.Winds W at 5 to 10 mph
Sep 27 Tomorrow night Mostly cloudy skies. Low 53.0F.Winds light and variable
Sep 28 Sunday Sunshine. Highs in the low 70s and lows in the low 50s
Sep 29 Monday A few clouds. Highs in the mid 70s and lows in the low 50s
Sep 30 Tuesday Abundant sunshine. Highs in the mid 70s and lows in the mid 50s
```

本例中, grep 显示所有不包含 Oct 的行。该选项常用于从文本中删除特定的行。删除特定行的方法是,首先将输出重定向到一个临时文件,然后再将临时文件重命名为原来的名字,如下所示。

```
# grep -v Oct Weather.txt > temp
# mv temp Weather.txt
mv: 是否覆盖 'Weather.txt' ? y
# cat Weather.txt
Sep 26 Today Partly cloudy. High around 65.0F.Winds NNW at 5 to 10 mph
Sep 26 Tonight Mostly cloudy skies. Low 53.0F.Winds WNW at 5 to 10 mph
Sep 27 Tomorrow Mostly cloudy. High around 70.0F.Winds W at 5 to 10 mph
Sep 27 Tomorrow night Mostly cloudy skies. Low 53.0F.Winds light and variable
Sep 28 Sunday Sunshine. Highs in the low 70s and lows in the low 50s
Sep 29 Monday A few clouds. Highs in the mid 70s and lows in the low 50s
Sep 30 Tuesday Abundant sunshine. Highs in the mid 70s and lows in the mid 50s
```

经过以上操作,在 Weather.txt 文件中,已经删除了包含 Oct 的全部行。

例如,选项“-w”使得 grep 只匹配按照单词方式匹配字符模式的行,而不是作为单词一部分匹配字符模式的行,如下所示。

```
# grep -w High Weather.txt
Sep 26 Today Partly cloudy. High around 65.0F.Winds NNW at 5 to 10 mph
Sep 27 Tomorrow Mostly cloudy. High around 70.0F.Winds W at 5 to 10 mph
```

本例中,包含单词 High 的行被匹配,而包含 Highs 的行不会被匹配。

需要搜索的文件不止一个, grep 会在匹配结果里的每一行前加上文件名。例如,在文件 WeatherReport 和 Weather.txt 中搜索 Monday,如下所示。

```
# grep Monday WeatherReport Weather.txt
WeatherReport:Sep 29 Monday Mostly cloudy.
Weather.txt:Sep 29 Monday A few clouds. Highs in the mid 70s and lows in the low 50s
```

而选项“-h”使得 grep 不打印文件名,如下所示。

```
# grep -h Monday WeatherReport Weather.txt
Sep 29 Monday Mostly cloudy.
Sep 29 Monday A few clouds. Highs in the mid 70s and lows in the low 50s
```



12.4 使用管道的 grep

grep 命令通常配合管道符 (|) 来作为其他命令的输入。例如，统计指定文件中包含某字符串的行数、字数和字节数，如下所示。

```
# cat example
aa bb cc dd
aa bb ff
ee
# grep 'bb' example |wc
      2      7     21
```

grep 命令除了作为其他命令的输入，也可以作为一些命令（如 ls、ps）的输出。例如，在当前运行的进程中查找 vi 程序的进程信息，命令行如下。

```
# ps aux | grep vi
root      5716  0.0  0.2  4956   736 pts/1    T   Jul25   0:00 vi
root     20681  0.3  0.4  4960  1012 pts/1    T   14:21   0:00 vi
root     20689  5.0  0.2  4132   668 pts/1    R+  14:22   0:00 grep vi
```

此外，还可以使用管道完成一系列匹配。例如，用 grep 命令检查哪个用户没有使用 bash，但使用 nologin 的账号不会显示出来，如下所示。

```
# grep -v bash /etc/passwd | grep -v nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
news:x:9:13:news:/etc/news:
nut:x:57:57:Network UPS Tools:/var/lib/ups:/bin/false
ldap:x:55:55:LDAP User:/var/lib/ldap:/bin/false
mailman:x:41:41:GNU Mailing List Manager:/var/mailman:/bin/false
```

12.5 grep 的退出状态

grep 还可以用于 Shell 脚本，因为 grep 通过返回一个状态值来说明搜索的状态。如果模式搜索成功，则返回 0；如果搜索不成功，则返回 1；如果搜索的文件不存在，则返回 2。利用这些返回值就可进行一些自动的文本处理工作。

在下面的例子中，grep 命令在 /etc/passwd 中搜索字符串 aaa。由于没有找到，退出状态为 1，如下所示。

```
# grep 'aaa' /etc/passwd
# echo $?
1
```

grep 命令在 /etc/passwd 中搜索字符串 root 成功，grep 的退出状态为 0，如下所示。

```
# grep 'root' /etc/passwd
root:x:0:0:root:/root:/bin/bash
operator:x:11:0:operator:/root:/sbin/nologin
```



```
# echo $?
0
```

而如果搜索的文件不存在，则退出状态为 2，如下所示。

```
# grep 'root' /etc/pass
grep: /etc/pass: 没有那个文件或目录
# echo $?
2
```



在 TC 和 C Shell 中的变量 status，以及 Bourne 和 Korn Shell 中的变量?都表示最后执行命令的状态。

12.6 egrep 与 fgrep

与 grep 非常相似的还有两个命令：一个是“egrep”，表示扩展 grep (extend grep)，执行效率比 grep 高，但需占用较大内存空间，用于搜索更复杂的正则表达式；另一个是“fgrep”，占用空间比 egrep 小，且速度也比 grep 快，但只能处理简单的字符串，不能处理正则表达式。这 3 个命令的结构、功能类似，大部分参数可以共享使用。本节将重点介绍它们之间的区别。

12.6.1 egrep

egrep 使用正则表达式元字符的扩展集，能够搜索更复杂的正则表达式。egrep 与 GNU grep 的“-E”选项相同，都能够使用正则表达式元字符的扩展集。表 12-3 所示为 egrep 使用的正则表达式元字符扩展集及说明。

表 12-3 grep 的正则表达式扩展集元字符及说明

元 字 符	说 明
+	匹配一个或多个先前的字符。如'[a-z]+able'，匹配一个或多个小写字母后跟 able 的串，如 loveable、enable，disable 等
?	匹配零个或多个先前的字符。如'gr?p' 匹配 gr 后跟一个或没有字符，然后是 p 的行
a b c	匹配 a 或 b 或 c。如 grep sed 匹配 grep 或 sed
()	分组符号，如 love(able rs)ov+匹配 loveable 或 lovers，匹配一个或多个 ov
x{m},x{m,},x{m,n}	作用同 x{m\},x{m,\},x{m,n\}

为了在不同国家的字符编码中保持一致，POSIX (The Portable Operating System Interface) 增加了特殊的字符类，如[:alnum:] 表示小写和大写字母以及数字，等价于[a-zA-Z0-9]。grep 和 egrep 命令都能够支持 POSIX 的字符类。

以下是几个例子。grep、egrep 和“grep -E”都将搜索两个数字后紧跟一个点的字符串所在行，如下所示。

```
# grep '[:space:][:digit:][:digit:]\.' Weather.txt
Sep 26 Today      Partly cloudy.      High around 65.0F.Winds NNW at 5 to 10 mph
Sep 26 Tonight    Mostly cloudy skies. Low 53.0F.Winds WNW at 5 to 10 mph
Sep 27 Tomorrow   Mostly cloudy.      High around 70.0F.Winds W at 5 to 10 mph
```



```

Sep 27 Tomorrow night Mostly cloudy skies. Low 53.0F.Winds light and variable
# grep -E '[[[:space:]][[[:digit:]][[[:digit:]]\.' Weather.txt
Sep 26 Today Partly cloudy. High around 65.0F.Winds NNW at 5 to 10 mph
Sep 26 Tonight Mostly cloudy skies. Low 53.0F.Winds WNW at 5 to 10 mph
Sep 27 Tomorrow Mostly cloudy. High around 70.0F.Winds W at 5 to 10 mph
Sep 27 Tomorrow night Mostly cloudy skies. Low 53.0F.Winds light and variable
# egrep '[[[:space:]][[[:digit:]][[[:digit:]]\.' Weather.txt
Sep 26 Today Partly cloudy. High around 65.0F.Winds NNW at 5 to 10 mph
Sep 26 Tonight Mostly cloudy skies. Low 53.0F.Winds WNW at 5 to 10 mph
Sep 27 Tomorrow Mostly cloudy. High around 70.0F.Winds W at 5 to 10 mph
Sep 27 Tomorrow night Mostly cloudy skies. Low 53.0F.Winds light and variable

```

接下来将通过一些例子，说明 **egrep** 与 **grep** 的不同。

例如，使用 **egrep**，匹配所有包含 **Monday** 或 **Friday** 的行，如下所示。

```

# egrep 'Monday|Friday' Weather.txt
Sep 29 Monday A few clouds. Highs in the mid 70s and lows in the low 50s
Oct 3 Friday Chance of showers. Highs in the upper 60s and lows in the mid 50s

```

在 GNU 版本的 **grep** 中，可以通过“-E”选项，打开对扩展元字符的支持，实现与 **egrep** 相同的功能，如下所示。

```

# grep -E 'Monday|Friday' Weather.txt
Sep 29 Monday A few clouds. Highs in the mid 70s and lows in the low 50s
Oct 3 Friday Chance of showers. Highs in the upper 60s and lows in the mid 50s

```

在 **grep** 命令中，不支持正则表达式元字符扩展集，在扩展集中表示“或”的竖线“|”不能够被识别，因此，在 **Weather.txt** 文件中搜索 **Monday|Friday** 字符串，没有匹配结果，如下所示。

```

# grep 'Monday|Friday' Weather.txt

```

需要注意的是，在 GNU 的标准 **grep** 中，如果在字符前加一个反斜杠，则该字符就会被当作是扩展正则表达式，因此，最后一个例子中，“\|”将竖线变为了正则表达式扩展集元字符，从而与 **egrep** 和“**grep -E**”实现相同的功能，如下所示。

```

# grep 'Monday\|Friday' Weather.txt
Sep 29 Monday A few clouds. Highs in the mid 70s and lows in the low 50s
Oct 3 Friday Chance of showers. Highs in the upper 60s and lows in the mid 50s

```

例如，以下几个语句，都能够匹配文件中包含一个或多个 6 的行。

```

# egrep '6+' Weather.txt
Sep 26 Today Partly cloudy. High around 65.0F.Winds NNW at 5 to 10 mph
Sep 26 Tonight Mostly cloudy skies. Low 53.0F.Winds WNW at 5 to 10 mph
Oct 3 Friday Chance of showers. Highs in the upper 60s and lows in the mid 50s
# grep -E '6+' Weather.txt

```

```

Sep 26 Today Partly cloudy. High around 65.0F.Winds NNW at 5 to 10 mph
Sep 26 Tonight Mostly cloudy skies. Low 53.0F.Winds WNW at 5 to 10 mph
Oct 3 Friday Chance of showers. Highs in the upper 60s and lows in the mid 50s
# grep '6\+' Weather.txt

```

```

Sep 26 Today Partly cloudy. High around 65.0F.Winds NNW at 5 to 10 mph
Sep 26 Tonight Mostly cloudy skies. Low 53.0F.Winds WNW at 5 to 10 mph
Oct 3 Friday Chance of showers. Highs in the upper 60s and lows in the mid 50s

```

例如，以下命令匹配包含两个或多个连续 **NN** 的行。

```

# egrep '(NN)+' Weather.txt
Sep 26 Today Partly cloudy. High around 65.0F.Winds NNW at 5 to 10 mph
# grep -E '(NN)+' Weather.txt

```

```

Sep 26 Today Partly cloudy. High around 65.0F.Winds NNW at 5 to 10 mph
# grep '\(NN\)+' Weather.txt

```

```

Sep 26 Today Partly cloudy. High around 65.0F.Winds NNW at 5 to 10 mph

```


12.6.2 fgrep

`fgrep` 命令行语法格式跟 `grep` 相似，但它不识别任何正则表达式元字符，因为它搜索字符串而不是搜索匹配正则表达式的模式。`fgrep` 命令使用快速的压缩算法。`$`、`*`、`[`、`|`、`(` 和 `\` 等字符串被 `fgrep` 命令按字面意思解释——一个插入符号就表示插入符号，一个美元符号就表示美元符号，等等。“`-F`”选项使得 GNU `grep` 与 `fgrep` 相同。

例如，在当前目录下所有以 `.c` 字符串结尾的文件中搜索字符串 `strcpy`，命令如下所示。

```
fgrep strcpy *.c
```

例如，在当前目录下所有以 `.c` 字符串结尾的文件中搜索字符串 `strcpy`，命令如下。

```
fgrep -l strcpy *.c
```

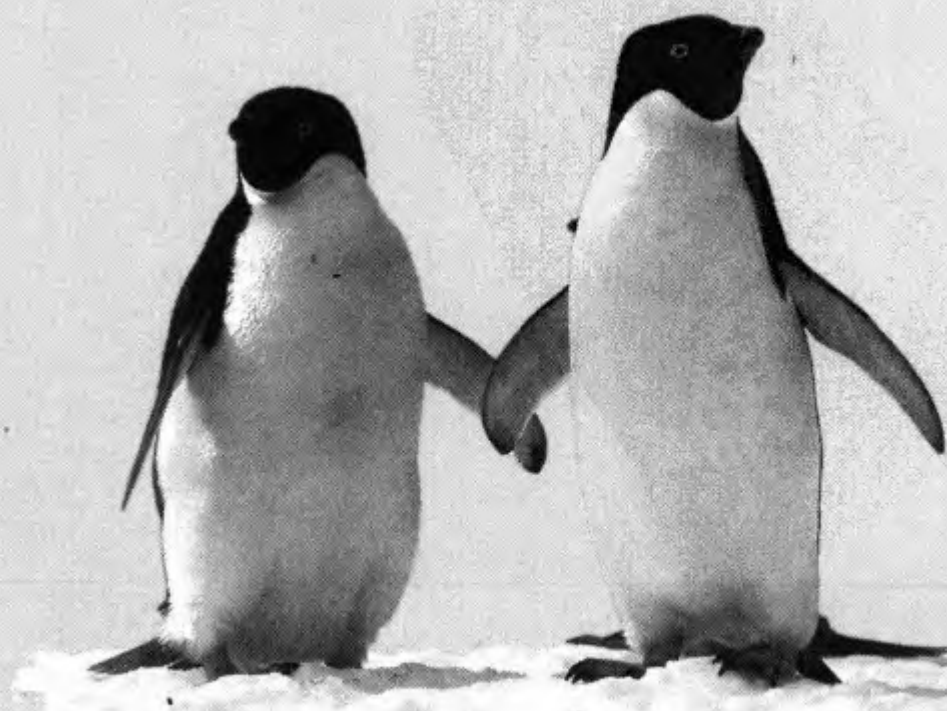
LINUX

第13章

gawk 与 sed

`awk` 是模式扫描和处理语言，又称过滤器，是 Linux/UNIX 下用来操纵数据和产生报告的程序语言。其处理的数据可以来自标准输入、一个或多个文件，或者其他命令的输出。`awk` 可用于命令行的简单操作，也可以写入大的应用程序。因为 `awk` 可以操纵数据，所以它是 Shell 脚本和管理小型数据库必需的工具。`awk` 之所以被广泛应用还因为它吸收了 C 语言的许多优点。`gawk` 是 `awk` 的 GNU 版本。

`sed` 是流编辑器，可以实现在 `vi` 和 `ex` 里一样的编辑任务。Linux 使用 GNU 版本的 `sed`，版权属于自由软件基金会，该版本与 UNIX 提供的版本在功能上几乎完全一致。`sed` 相当小巧，可以利用管道对标准输入/输出的数据进行编辑和组合。使用 `-V` 选项可以查看 `sed` 的帮助信息，如下所示。




```
# sed -V
GNU sed version 4.0.5
Copyright (C) 2002 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE,
to the extent permitted by law.
```

13.1 过滤器 gawk

gawk 是可以完成模式扫描和数据处理的语言。gawk 搜索一个或多个文件，查找其中匹配模式的记录（通常是行），通过执行指定的动作来处理这些记录。例如，每次发现匹配的时候，将记录写到标准输出，或者递增某个计数器的值。

gawk 程序可以使用变量、函数、算术运算符、关系运算符、关联数组、控制语句和 C 语言的 `printf` 语句，而高级 gawk 编程可以利用 `getline` 语句输入数据，使用协进程让 gawk 与其他程序进行数据交换，或者通过网络连接与运行在远程系统上的程序交换数据。

gawk 工具中的很多结构都来自 C 编程语言。它包含下列特性。

- 灵活的格式
- 支持条件语句
- 具有循环结构
- 支持数值变量
- 支持字符串变量
- 支持正则表达式
- 支持关系表达式
- 具有 C 语言的 `printf` 语句
- 支持协进程（coprocess）
- 支持网络数据交换

13.1.1 gawk 概述

awk 写于 1977 年，其名称取自其作者姓氏的第一个字母，其作者是 Alfred V Aho、Peter J. Weinberger 和 Brian W Kernighan。在 1985 年的升级版中加入了用户自定义函数、动态正则表达式和处理输入文件等功能，使得 awk 可以处理大型程序。awk 有多个版本，包括旧版 awk、nawk（new awk，新版本）、gawk（GNU 版本）以及 POSIX awk 等。

gawk 是 GNU awk 的缩写，它遵守 POSIX 1003.2 关于命令语言的定义，以及 Aho、Kernighan 和 Weinberger 对于 awk 的定义。gawk 提供了 Bell 实验室的 awk 扩展和一些 GNU 的最新扩展。使用 gawk 的“-version”选项可以显示版本信息。在大多数系统上，命令 awk 指的是旧版本，nawk 指的是新版本，而 gawk 指的是 GNU 版本。



本章基于 gawk，该版本的 awk 是 Linux 发行套件的一部分，在 Red hat Linux 9 中，awk 命令实质上是 gawk 命令的链接。

gawk 逐行扫描文件，从第一行到最后一行寻找匹配特定模式的行，并在这些行上运行指定的动作。与过程式编程语言相反，gawk 属于数据驱动语言，即描述想要处理的数据并告诉 gawk 当它发现这些数据时将要做的事情。

gawk 程序由包含模式和动作的单行或多行文本组成，格式如下。

```
pattern { action }
```

模式 (pattern) 从输入中选择文本行。针对模式选中的所有行，gawk 执行相应的动作 (action)。如果程序行并没有包含模式，gawk 将选中输入中的所有行。如果程序行没有包含动作，gawk 将选中的行输出到标准输出。

本章在介绍 gawk 命令时，很多示例都使用了 example 和 student 数据文件。在 example 文件中，从左至右依次为每名学生的姓名、出生年份、性别、课程、分数，所有空白字符都由单个 Tab 组成 (该文件没有包含任何空格)。该文件内容如下所示。

```
# cat example
Zhangsan      1977      male      computer      83
lisi          1989      male      math          99
Wanglijiang   1990      female    chinese       78
Xuliang       1977      male      economic      89
Xuxin         1986      female    english       99
Wangxuebing   1978      male      math          89
lichang       1989      male      math          99
Wanglijiang   1990      female    chinese       78
Zhangsansan   1977      male      computer      83
langxuebing   1978      male      math          89
lisibao       1989      male      math          99
xiaobao       1990      female    chinese       78
```

在 student 文件中，从左至右依次为每名学生的姓名、学号、出生日期、出生地所有字段之间以冒号分隔。该文件内容如下所示。

```
# cat student
Tom Helleen:10501173:93-09-11:Harbin
Mary Degens:10501179:95-05-22:Peking
Suli Vanlen:10501170:90-09-17:Harbin
Tom Changle:10501176:88-01-11:Harbin
Billy Black:10501175:93-02-18:Harbin
John Hellen:10501172:93-04-13:Peking
Sam Possion:10501171:98-11-19:Tianjin
```



awk (gawk 的基础) 的作者 (Alfred V Aho、Peter J. Weinberger 和 Brian W Kernighan) 为了将 awk 设计得易于使用，部分牺牲了 awk 的执行速度。

13.1.2 gawk 的语法

gawk 从命令行上指定的文件或标准输入设备获取输入数据。gawk 的语法格式如下所示。

```
gawk [options][program][file-list]
gawk [options] -f program-file [file-list]
```

其中，“program”是用户在命令行中所包含的 gawk 程序。“program-file”是存放 gawk 程序文件的名称。将较长或较复杂的程序放置在文件中可以减少错误和重复输入。在命令行上使用 gawk，就可以编写出简短的 gawk 程序，而不用创建单独的 program-file 文件。为了阻止 Shell 将 gawk 命令解释成 Shell 命令，要将 program 用单引号引起来。除非将 gawk 的输出重定向，否则将被送往标准输出。

“file-list”包含 gawk 要处理的普通文件的路径名，这些文件就是输入的数据文件。如果用户没有指定 file-list，gawk 将从标准输入获取输入，或者由 getline 或协进程指定输入。表 13-1 列出了 gawk 常用选项说明。

表 13-1 gawk 命令选项说明

选 项	说 明
--field-seperator fs, -F fs	将 fs 作为输入字段分隔符（FS 变量）的值
--field program-file, -f program-file	从 program-file 命名的文件中而不是标准输入中读取 gawk 程序。用户可以在命令行上多次指定这个选项
--help, -W help	显示 gawk 的帮助信息
--lint, -W lint	对不正确或不利于移植的结构发出警告
--posix, -W posix	运行 POSIX 兼容版 gawk，该选项引入了一些限制，详细内容请查看 gawk 的 man 页

如果用户编写的 gawk 程序比较长，那么可以将程序放在一个文件中，然后在命令行上看引用该文件。使用 gawk 命令中的“-f”或“--file”选项可以指定引用的 gawk 程序文件。例如，以下是 gawk 程序文件，用于输出标题信息（代码 head_student）。

```
1 BEGIN {
2   print "          学生基本信息统计          "
3   print "-----"
4 }
5 {print}
```

使用 gawk 的“-f”选项引入 head_student 文件，对文件 example 进行处理，如下所示。

```
# gawk -f head_student example
          学生基本信息统计
-----
Zhangsan      1977      male      computer      83
lisi          1989      male      math           99
Wanglijiang   1990      female   chinese        78
Xuliang       1977      male     economic       89
Xuxin         1986      female   english        99
Wangxuebing   1978      male     math           89
lichang       1989      male     math           99
Wanglijiang   1990      female   chinese        78
Zhangsansan   1977      male     computer       83
langxuebing   1978      male     math           89
```



```
lisibao      1989      male      math      99
xiaobao      1990      female    chinese    78
```

除了在命令行使用 `gawk` 命令的“-f”选项调用 `gawk` 程序外，还可以编写独立脚本程序。独立脚本文件需要具备读权限和执行权限，且需在程序首行添加“`#!/bin/gawk -f`”。该行代码告诉 Shell 使用“`/bin/gawk -f`”运行该脚本。命令如下所示。

```
1 #!/bin/gawk -f          //程序第 1 行的“#!/bin/ gawk -f”表示调用“gawk -f”来执行
脚本程序
2 #filename:head2_student //程序第 2 行的“#filename: head2_student”表示该程序的名称
为 head2_student
3 BEGIN {                //第 3~6 行是 BEGIN 模式
4 print "                学生基本信息统计                " //显示输出“学生基本信息统计”
5 print "-----"      //显示输出分隔符
6 }                      // BEGIN 模式结束
7 {print}                //显示输出所有内容
```

该程序的执行示例如下所示。

```
# chmod u+x head2_student //赋予文件 head2_student 可执行权限
# ./head2_student example //带参数“example”执行 head2_student
      学生基本信息统计
-----
Zhangsan      1977      male      computer    83
lisi          1989      male      math        99
Wanglijiang   1990      female    chinese     78
Xuliang       1977      male      economic    89
Xuxin         1986      female    english     99
Wangxuebing   1978      male      math        89
lichang       1989      male      math        99
Wanglijiang   1990      female    chinese     78
Zhangsansan   1977      male      computer    83
langxuebing   1978      male      math        89
lisibao       1989      male      math        99
xiaobao       1990      female    chinese     78
```

可以看到，在数据前添加了标题信息“学生基本信息统计”。

13.1.3 awk 的语言基础

`gawk` 工具是 UNIX `awk` 的 GNU 版本。为了方便起见，很多 Linux 系统(包括 Red Hat Linux 9)将 `/bin/awk` 链接至 `/bin/gawk`，这样用户就可以使用两者之中的任何一个来运行程序。链接关系如下所示。

```
# ll /bin/awk
lrwxrwxrwx 1 root root 4 8月 7 14:34 /bin/awk -> gawk
```

`gawk` 程序(来自命令行或程序文件)由一行或多行文本构成，其中包含一个模式和动作，格式如下。

```
pattern { action }
```

模式(pattern)用来从输入中选取文本行。对于由模式选中的每行文本，`gawk` 命令都执行动作(action)进行处理。动作两边的大括号将动作与模式区分开来。如果程序行中没有包含模式，`gawk` 将选取输入中的所有行。如果程序行没有包含动作，`gawk` 将把选中的行复制到标准输出。

`gawk` 的处理流程如下。开始时，`gawk` 将输入(来自 file-list 或标准输入)的第一行与程

序中的每个模式进行比较。如果某个模式选中某一行（如果存在一次匹配的话），那么 gawk 将执行与模式关联的动作。如果没有选中的行，gawk 将不执行动作。当 gawk 完成了对输入中第一行文本的比较之后，将对输入中的下一行重复上面的过程。gawk 将继续对输入中后续的每行文本执行这个比较过程，直到其读完了所有的输入。

如果多个模式选中了同一行文本，gawk 将按照这些模式出现在程序中的顺序分别执行与每个模式关联的动作。gawk 有可能多次向标准输出发送来自输入的同一行文本。

13.1.4 模式

用斜杠把正则表达式包装起来，就可以将其作为模式使用。例如，下面示例程序中，使用一个正则表达式作为模式，但没有指明动作。

```
/math/
```

gawk 将从数据输入（本例为文件 example）中选择包含字符串“math”的所有行。由于没有指定动作，默认为 print。以下是该程序的执行示例。

```
# gawk '/math/' example
```

lisi	1989	male	math	99
Wangxuebing	1978	male	math	89
lichang	1989	male	math	99
langxuebing	1978	male	math	89
lisibao	1989	male	math	99

可以看到，通过该程序将所有课程名为“math”（数字）的学生的相关信息显示出来。

运算符“~”用于测试某个字段或变量是否匹配正则表达式。运算符“!~”用于测试不匹配。例如，使用匹配运算符“~”来选择 example 文件第一个字段中包含字符“l”的行，执行结果如下所示。

```
# gawk '$1~/l/' example
```

lisi	1989	male	math	99
Wanglijiang	1990	female	chinese	78
Xuliang	1977	male	economic	89
lichang	1989	male	math	99
Wanglijiang	1990	female	chinese	78
langxuebing	1978	male	math	89
lisibao	1989	male	math	99

例如，下面示例中使用运算符“~”搜索 1990 年出生的学生，且将其分数加 5，执行结果如下所示。

```
# gawk ' {if ($2~/1990/) $5=$5+5 ; print}' example
```

Zhangsan	1977	male	computer	83
lisi	1989	male	math	99
Wanglijiang	1990	female	chinese	83
Xuliang	1977	male	economic	89
Xuxin	1986	female	english	99
Wangxuebing	1978	male	math	89
lichang	1989	male	math	99
Wanglijiang	1990	female	chinese	83
Zhangsansan	1977	male	computer	83
langxuebing	1978	male	math	89
lisibao	1989	male	math	99
xiaobao	1990	female	Chinese	83



例如，下面示例中使用运算符 “!~” 搜索不是 1990 年出生的学生，且将其分数减 5，执行结果如下所示。

```
# gawk ' {if ($2!~/1990/) $5=$5-5 ; print}' example
Zhangsan 1977 male computer 78
lisi 1989 male math 94
Wanglijiang 1990 female chinese 78
Xuliang 1977 male economic 84
Xuxin 1986 female english 94
Wangxuebing 1978 male math 84
lichang 1989 male math 94
Wanglijiang 1990 female chinese 78
Zhangsansan 1977 male computer 78
langxuebing 1978 male math 84
lisibao 1989 male math 94
xiaobao 1990 female chinese 78
```

BEGIN 和 END 是两个独特的模式，分别是在 gawk 开始处理之前和处理完毕之后要执行的命令。在处理所有输出之前，gawk 执行 BEGIN 模式关联的动作，在处理完之后执行 END 模式关联的动作。

例如，下面 gawk 程序中将显示题标的代码放在 BEGIN 模式中，执行结果如下所示。

```
# gawk 'BEGIN{print "-----学生基本信息表总成绩-----"} {print}' example
-----学生基本信息表总成绩-----
Zhangsan 1977 male computer 83
lisi 1989 male math 99
Wanglijiang 1990 female chinese 78
Xuliang 1977 male economic 89
Xuxin 1986 female english 99
Wangxuebing 1978 male math 89
lichang 1989 male math 99
Wanglijiang 1990 female chinese 78
Zhangsansan 1977 male computer 83
langxuebing 1978 male math 89
lisibao 1989 male math 99
xiaobao 1990 female chinese 78
```

例如下面示例中，计算文件 example 中记录的学生成绩总和，在 BEGIN 模式中显示标题、设置变量 total 的初值，在 END 模式中显示变量 total 的值。执行过程如下所示。

```
# gawk 'BEGIN{print "统计学生总成绩";total=0} {print $5; total=total+$5} END{printf "
学生总成绩为: %.2f",total}' example
统计学生总成绩
83
99
78
89
99
89
99
78
83
89
99
78
学生总成绩为: 1063.00
```

逗号 “,” 是范围运算符。如果在一个 gawk 程序行上用逗号将两个模式隔开，则 gawk

选取的第一行是匹配第一个模式的文本行，选取的最后一行是匹配第二个模式的文本行。如果没有匹配第二个模式的文本行，gawk 将选取直到输入数据末尾的所有文本行。在 gawk 找到第二个模式之后，将再次查找第一个模式以再次开始这个过程。

例如，查找文件 example 中包含 “Zhangsan” 与 “Zhangsansan” 之间的文本行，如下所示。

```
# gawk -F: '/Zhangsan/,/Zhangsansan/' example
Zhangsan      1977      male      computer      83
lisi          1989      male      math          99
Wanglijiang   1990      female    chinese       78
Xuliang       1977      male      economic      89
Xuxin         1986      female    english       99
Wangxuebing   1978      male      math          89
lichang       1989      male      math          99
Wanglijiang   1990      female    chinese       78
Zhangsansan   1977      male      computer      83
```

例如，查找文件 example 中包含 “1977” 与 “1989” 之间的文本行，如下所示。

```
# gawk -F: '/1977/,/1989/' example
Zhangsan      1977      male      computer      83
lisi          1989      male      math          99
Xuliang       1977      male      economic      89
Xuxin         1986      female    english       99
Wangxuebing   1978      male      math          89
lichang       1989      male      math          99
Zhangsansan   1977      male      computer      83
langxuebing   1978      male      math          89
lisibao       1989      male      math          99
```

从显示结果可以看到，在找到第一次匹配后，gawk 将开始再次查找新的匹配，直至文件结束。

13.1.5 动作

如果 gawk 匹配某个模式，它将执行 gawk 命令的动作部分所指定的动作。如果没有指定动作，gawk 将执行默认动作，即 print 命令（可以用 { print } 显式表示）。这个动作将记录（通常是一行）从输入复制到标准输出。

如果在 print 命令后面带上参数，gawk 将只显示用户指定的参数。这些参数可以是变量或字符串常量。可以将 print 命令的输出发送到文件（>）、追加到文件（>>）或者通过管道将其发送给另一个程序的输入（“|”）。协进程（“|&”）是一个双向管道，可以与运行在后台的程序交换数据。

除非用逗号将 print 命令中的各输出项区分开，否则 gawk 将它们连接起来输出。逗号使得 gawk 用输出字段分隔符（变量 FS 的值，默认是空格符）将各项分隔开。



通过用分号将多个动作隔开，可以在同一行上包含多个动作。

例如，下面是一个简单的 gawk 程序，该程序由单行代码组成，其中没有指定模式，仅使用了一个 print 动作。

```
{ print }
```

由于没有指定模式，所以 gawk 会选择输入中的所有行作为待处理的数据，而 print 命令没有任何参数，意味着直接显示输出选中的每一行。该程序的实际作用是显示输出指定文件



的内容。执行示例如下所示。

```
# gawk '{print}' example
Zhangsan      1977      male      computer      83
lisi          1989      male      math           99
Wanglijiang   1990      female    chinese        78
Xuliang       1977      male      economic       89
Xuxin         1986      female    english        99
Wangxuebing   1978      male      math           89
lichang       1989      male      math           99
Wanglijiang   1990      female    chinese        78
Zhangsansan   1977      male      computer       83
langxuebing   1978      male      math           89
lisibao       1989      male      math           99
xiaobao       1990      female    chinese        78
```

可以看到，显示输出文件 `example` 的所有内容。

13.1.6 注释

`gawk` 工具不处理以符号“#”开头的程序行中的任何内容。用户可以在程序注释文本前面添加这个符号，为 `gawk` 程序加入注释。

13.1.7 变量

尽管不需要在使用 `gawk` 变量之前声明变量，但用户可以选择向这些变量赋初始值。没有赋值的数值变量被初始化为 0，而字符串变量则被初始化为空串。除了用户变量（`user variable`）之外，`gawk` 还维护了程序变量（`program variable`）。在 `gawk` 程序的模式部分和动作部分中均可以使用用户变量和程序变量。表 13-2 列出了一些程序变量。

表 13-2 变量说明

变 量	说 明
\$0	当前记录（作为单个变量）
\$1-\$n	当前记录中的字段
FILENAME	当前输入文件名（null 表示标准输入）
FS	输入字段的分隔符
NF	当前记录的字段数目
NR	当前记录的记录编号，每处理完一个记录，NR 的值就自动加 1
OFS	输出字段分隔符（默认为空格符）
ORS	输出记录分隔符（默认为换行符）
RS	输入记录分隔符（默认为换行符）

除了可以在程序中初始化变量之外，还可以在命令行上使用“`--assign`”（或“`-v`”）选项初始化变量。如果某个变量的值在 `gawk` 的两次运行之间发生改变，这个特性就非常有用。

默认情况下，输入记录和输出记录的分隔符均为换行符。因此，`gawk` 将每行输入作为单独的一条记录，并在每条输出记录后面追加一个换行符。默认情况下，输入字段分隔符为空格和 Tab。默认的输出字段分隔符是空格。在任意时刻都可以修改分隔符的值，具体方法是

通过在程序或命令行中使用“--assign”（或“-v”）选项，将新的值指派给与这些分隔符相关的变量。

变量 NR 中包含了当前行的记录号（行号）。下面示例中利用变量 NR，将 example 文件中学生出生年份大于等于 1989 的学生的行号打印输出，执行过程如下所示。

```
# gawk '$2 >= 1989{print NR }' example
2
3
7
8
11
12
```

13.1.8 记录和字段

gawk 把输入文件看作一个具有一定格式和结构的连续字符串来处理的。默认每一个以换行符结束的行为一个记录。

记录中的每一个以空格或 Tab 分隔的字符串，称为字段（也称为“域”）。每一行记录的字段数目是不一样的，变量 NF 中保存了当前记录的字段数目。变量 FS 保存了字段使用的分隔符，默认为空格或 Tab。可以在 BEGIN 语句中或命令行上使用“-F”选项来修改 FS 的设置，从而修改分隔符。

例如，通过在命令行中使用“-F”选项，重新设定字段分隔符为冒号“:”，对文件 student 的第 1 字段、第 4 字段进行输出，如下所示。

```
# cat student //显示文件 student 的内容
Tom Helleen:10501173:93-09-11:Harbin
Mary Degens:10501179:95-05-22:Peking
Suli Vanlen:10501170:90-09-17:Harbin
Tom Changle:10501176:88-01-11:Harbin
Billy Black:10501175:93-02-18:Harbin
John Hellen:10501172:93-04-13:Peking
Sam Possion:10501171:98-11-19:Tianjin
# gawk -F: '/Mary/{ print $1,$4}' student
Mary Degens Peking
```

“-F”选项后的符号“:”设定了新的字段分隔符为冒号。从输出结果可以看到，该程序在文件 student 中查找包含字符串“Mary”的行，并以冒号作为字段分隔符，输出该行的第 1 和第 4 个字段。

若字段分隔符由多个字符组成，可以将其放在方括号内进行指定。例如，下面示例中将“:”和“-”均视为字段分隔符，用于提取所有学生的出生年份（即以“:”和“-”为分隔符的第 3 个字段），执行结果如下所示。

```
# gawk -F[: -] '{ print $3}' student
93
95
90
88
93
93
98
```



13.1.9 输出字段分隔符

默认的输出字段分隔符为空格，保存在变量 OFS 中。在用 print 语句进行输出打印时，指定的多个字段之间以逗号分隔，而逗号表示的就是 OFS 的值。

例如，下面示例中对 student 文件中查找包含字符串“Tom”的行，并以冒号作为字段分隔符，输出该行的第 2 和第 3 个字段。输出时，\$2 与 \$3 之间使用一个空格作为分隔符，执行结果如下所示。

```
# gawk -F[:] '/Tom/{ print $2,$3}' student
10501173 93-09-11
10501176 88-01-11
```

可以看到，由于在 print 语句的“\$2”和“\$3”之间使用了逗号，所以在输出时第 2 和第 3 个字段之间留有空格。

相反地，下面示例中没有在 print 语句的“\$2”和“\$3”之间使用逗号，所有输出字段都挤在了一起。

```
# gawk -F[:] '/Tom/{ print $2 $3}' student
1050117393-09-11
1050117688-01-11
```

例如，变量 OFS 中保存了输出字段间使用的分隔符，以下示例中将输出字段的分隔符设置为 Tab（转义字符为“\t”），输出 student 文件中第 1、2、3 字段的内容，如下所示。

```
# gawk 'BEGIN{FS=":";OFS="\t"} { print $1,$2,$3}' student
Tom Helleen      10501173      93-09-11
Mary Degens      10501179      95-05-22
Suli Vanlen      10501170      90-09-17
Tom Changle      10501176      88-01-11
Billy Black      10501175      93-02-18
John Hellen      10501172      93-04-13
Sam Possion      10501171      98-11-19
```

13.1.10 重定向输出

gawk 默认将结果输出到屏幕，但若希望将输出保存到一个文件中，可以使用输出重定向。使用符号“>”可以重新定向输出。

例如，下面程序中使用输出重定向操作，将文件 example 中记录的 1989 年出生的学生信息保存到文件 year1989，将 1977 年出生的学生信息保存到文件 year1977 中，如下所示（代码 redirectout）。

```
1 /1989/ {print > "year1989"}
2 /1977/ {print > "year1977"}
3 END{ print "输出完毕!"}
```

该程序执行结果如下所示。

```
# gawk -f redirectout example //调用 gawk 命令执行 redirectout 程序文件，处理数据文件 example
输出完毕!
# cat year1977 //显示生成的文件 year1977 的内容
Zhangsan      1977      male      computer      83
Xuliang       1977      male      economic      89
Zhangsansan   1977      male      computer      83
# cat year1989 //显示生成的文件 year1989 的内容
```



```
lisi      1989   male   math      99
lichang   1989   male   math      99
lisibao   1989   male   math      99
```

13.1.11 函数

大多数流行的编程语言中都有函数（function）的概念，函数是一段完成指定功能的、被命名的代码片段。表 13-3 列出了 gawk 所提供的一些用来操作数值和字符串的函数。

表 13-3 gawk 中的函数

函 数 名	说 明
length(str)	返回 str 中的字符个数，如果没能带参数，则返回当前记录中的字符个数
int(num)	返回 num 的整数部分
index(str1,str2)	返回 str2 在 str1 中的位置，如果 str2 不存在返回 0
split(str,arr,del)	用 del 作为定界符，将 str 的元素放置到数组 arr[1]...arr[n]中，返回数组元素个数
sprintf(fmt,args)	根据 fmt 格式化 args 并返回格式化后的字符串，模仿 C 编程语言中相应的同名函数
substr(str,pos,len)	返回 str 中从 pos 开始长度为 len 个字符的字符串
tolower(str)	返回 str 的副本，且其中所有大写字母被替换成相应的小写字母
toupper(str)	返回 str 的副本，且其中所有小写字母被替换成相应的大写字母

例如，如果用户不带任何参数使用 length 函数，该函数将返回当前文本行中的字符个数，包括字段分隔符。下面示例中，通过 length 函数将每一行记录的长度添加到每行的开头，然后通过管道（“|”）将输出从 gawk 输入到 sort，使输出行按每行的长度由小到大进行排列，如下所示。

```
# gawk '{print length, $0 }' example | sort -n
24 lisi      1989   male   math      99
27 lichang   1989   male   math      99
27 lisibao   1989   male   math      99
30 langxuebing 1978   male   math      89
30 Wangxuebing 1978   male   math      89
30 Xuliang    1977   male   economic  89
30 Xuxin      1986   female english    99
30 Zhangsan   1977   male   computer   83
32 xiaobao    1990   female chinese   78
33 Zhangsansan 1977   male   computer   83
35 Wanglijiang 1990   female chinese   78
35 Wanglijiang 1990   female chinese   78
```

其中，变量\$0 表示当前文本行，sort 命令中的选项“-n”表示按数值排序。

13.1.12 关联数组

关联数组是 gawk 最强大的功能之一。关联数组使用字符串作为索引。在使用关联数组时，用户可以用数值字符串作为索引来模仿传统的数字索引数组。

可以像将某个值指派给任何其他 gawk 变量一样，将值指派给关联数组中的某个元素，其语法格式如下。

```
array [string] =value
```

其中，array 为数组的名称，string 为用户将要赋值的元素在数组中的索引，value 为将要



指派给该元素的值。

可以将特殊的 for 结构用于关联数组，语法格式如下。

```
for (elem in array) action
```

for 结构将遍历数组中的元素，elem 表示数组中每个元素的值，array 为数组名称，action 为 gawk 对数组中每个元素所采取的动作。可以在 action 中使用 elem 变量。

例如，下面程序中，依据文件 example 中第 1 个字段建立关联数组 name，数组索引为第 1 个字段的内容，且数组的值为第 5 个字段的内容，即由 name[lisibao]=99, name[Wanglijiang]=78, name[Zhangsansan]=83 等组成。建立关联数组后，显示数组索引及数组元素的值，具体代码如下所示（代码 arrayexample）。

```
1 #filename:arrayexample
2 gawk '{name[$1]=$5} END{ for (elem in name) print elem" 的成绩为: " name[elem]}' example
```

该程序的执行结果如下所示。

```
#chmod u+x arrayexample
# ./arrayexample
lisibao 的成绩为: 99
Wanglijiang 的成绩为: 78
Zhangsansan 的成绩为: 83
Xuliang 的成绩为: 89
xiaobao 的成绩为: 78
langxuebing 的成绩为: 89
Wangxuebing 的成绩为: 89
Zhangsan 的成绩为: 83
lisi 的成绩为: 99
Xuxin 的成绩为: 99
lichang 的成绩为: 99
```

例如，下面程序中依据文件 example 中指定的字段建立关联数组 arr，数组索引为指定字段的内容，且数组的值为第 1 个字段的内容。建立关联数组后，显示数组索引及数组元素的值，具体代码如下所示（代码 arrayexample2）。

```
1 #filename:arrayexample2          //注释，标明该程序名为 arrayexample2
2 if [ $# != 2 ]                  //第2~6行为 if...then...fi 结构，判断条件为 "$# != 2"，即变量$#的
                                   //值不为 2 时执行第 4、5 行，如果等于 2 则直接执行第 7 行
3 then                            // if 结构中的 then 语句
4   echo "Usage: arrayexample2 field file"    //显示输出 "Usage: arrayexample2 field
file"
5   exit 1                        //退出
6 fi                              // if 结构中的 fi 语句
7 gawk -F: < $2 '{arr['$1']=$1} END{ for (elem in arr) print elem"<----->" arr[elem]}'
```

//gawk 命令，其

//中“-F:”表示分隔符为“:”；“< \$2”表示将位置参数 2 指定的文件名作为 gawk 的输入；“\$1”表示用位置

//参数 1 的值指定索引字段（前导“\$”符号使 gawk 将 Shell 所替换的内容翻译为字段编号）。“arr['\$1']=\$1”表

//示将字段 1 的内容赋值给数组 arr。“for (elem in arr) print elem"<----->" arr[elem]”表示

输出数组 arr 的索引及元

//素值

该程序的执行结果如下所示。

```
#chmod u+x arrayexample2          //赋予文件 arrayexample2 执行权限
# ./arrayexample2 2 student        //执行程序 arrayexample2，且将第 2 个字段作为数组索引，数据文
件为 student
10501175<----->Billy Black
```



```
10501176<----->Tom Changle
10501179<----->Mary Degens
10501170<----->Suli Vanlen
10501171<----->Sam Possion
10501172<----->John Hellen
10501173<----->Tom Helleen
# ./arrayexample2 3 student      //执行程序 arrayexample2, 且将第 3 个字段作为数组索引, 数
据文件为 student
88-01-11<----->Tom Changle
98-11-19<----->Sam Possion
90-09-17<----->Suli Vanlen
93-09-11<----->Tom Helleen
93-04-13<----->John Hellen
95-05-22<----->Mary Degens
93-02-18<----->Billy Black
```

13.1.13 printf 命令：格式化输出

可以使用 printf 命令代替 print 控制 gawk 产生的输出。gawk 版的 printf 类似于 C 语言中的 printf，其语法格式如下。

```
printf "control-string", arg1, arg2, ..., argn
```

control-string 决定了 printf 如何格式化 arg1、arg2...argn。这些参数既可以是变量也可以是其他表达式。可以在 control-string 中使用 “\n” 来表示换行符，使用 “\t” 来表示 Tab。control-string 包含了转换规格，每个参数对应一个。转换规格的语法如下。

```
%[-][x[.y]]conv
```

其中，“-”表示输出左对齐，“x”表示最小字段宽度，“y”表示数字中小数点右边的位数。“conv”指示数值转换的类型，这些类型如表 13-4 所示。

表 13-4 数值转换类型

conv (转换类型)	说 明
d	十进制
e	指数表示
f	浮点数字
g	使用 f 或 e 中较短的那个
o	无符号八进制
s	字符串
x	无符号十六进制

例如，对文件 student 中有 2 人以上来自同一出生地的城市名称进行显示输出。显示输出时使用 printf 语句，且按字符串格式进行输出。程序代码如下所示。

```
1 { FS=":"      //变量 FS 保存了输入字段的分隔符，此处设定 “:” 为分隔符
2   count[$4]++ //根据输入文件的第 4 个字段建立关联数组，即第 4 个字段作为关联数组的索引，
//数组元素的值为该字段内容重复出现的次数。“++”表示自加，每重复出现一次自加 1
3 }           //与第 1 行的左大括号匹配
4 END {      //第 4~12 行为 END 模式
5   for (item in count) //第 5~11 行为 for 循环。每次循环从关联数组 count 中读取一个元素
```



```
6      {                                //表示 for 循环体开始
7          if ( count[item] >= 2)      //第 7~10 行为 if 结构，判断条件为每一数组元素值是
//否大于等于 2。当大于等于 2 时，执行第 9 行，否则跳过 if 结构
8          {                            // if 结构开始
9              printf "城市%-20s 出现了%3s 次\n",item,count[item] //使用printf命令显示输出
数组元素
//的索引（城市名）及元素值（重复次数）。"%-20s"表示按字符串进行输出，且左对齐，字段宽为 20 个字符。
// "%3s"表示按字符串进行输出，且右对齐，字段宽为 3 个字符
10         }                            // if 结构结束
11     }                                // for 循环结构体结束
12 }                                    // END 模式结束
```

该程序的执行结果如下所示。

```
# cat student                                //显示文件 student 的内容
Tom Helleen:10501173:93-09-11:Harbin
Mary Degens:10501179:95-05-22:Peking
Suli Vanlen:10501170:90-09-17:Harbin
Tom Changle:10501176:88-01-11:Harbin
Billy Black:10501175:93-02-18:Harbin
John Hellen:10501172:93-04-13:Peking
Sam Possion:10501171:98-11-19:Tianjin
# gawk -f citycount student                //使用 gawk 命令执行程序 citycount，处理数据输入文件
student
城市 Harbin          出现了  3 次
城市 Peking          出现了  2 次
```

13.1.14 关系运算符

在 gawk 中，可以使用表 13-5 中列出的关系运算符执行数值比较和字符串比较。可以使用布尔运算符“||”或“&&”来组合任何模式。

表 13-5 关系运算符

运 算 符	说 明
<	小于
<=	小于等于
==	等于
!=	不等于
>=	大于等于
>	大于

例如，对文件 student 中出生地为哈尔滨（“Harbin”）的学生信息进行显示，执行命令如下所示。

```
# cat student                                //文件 student 的内容
Tom Helleen:10501173:93-09-11:Harbin
Mary Degens:10501179:95-05-22:Peking
Suli Vanlen:10501170:90-09-17:Harbin
Tom Changle:10501176:88-01-11:Harbin
Billy Black:10501175:93-02-18:Harbin
John Hellen:10501172:93-04-13:Peking
Sam Possion:10501171:98-11-19:Tianjin
```


可以看到，文件 student 由 4 个字段组成，以冒号为字段间的分隔符，各字段依次表示学生姓名、学号、出生日期、籍贯。搜索籍贯为“Harbin”的学生记录需要使用关系运算符“==”，如下所示。

```
# gawk -F: '$4=="Harbin"' student
Tom Helleen:10501173:93-09-11:Harbin
Suli Vanlen:10501170:90-09-17:Harbin
Tom Changle:10501176:88-01-11:Harbin
Billy Black:10501175:93-02-18:Harbin
```

由于第 4 个字段为字符串，所以在 gawk 程序中的等于运算符中使用了双引号。如果是数字之间进行比较，则无需加双引号。

例如，查找文件 student 中，学号为“10501173”的记录，程序代码及执行结果如下所示。

```
# gawk -F: '$2 == 10501173' student
Tom Helleen:10501173:93-09-11:Harbin
```

例如，查找文件 student 中，学号大于等于“10501175”的记录，程序代码及执行结果如下所示。

```
# gawk -F: '$2 >= 10501175' student
Mary Degens:10501179:95-05-22:Peking
Tom Changle:10501176:88-01-11:Harbin
Billy Black:10501175:93-02-18:Harbin
```

例如，查找文件 student 中，学号大于等于“10501172”且小于“10501176”的记录，程序代码及执行结果如下所示。

```
# gawk -F: '$2 >= 10501172 && $2 <=10501176' student
Tom Helleen:10501173:93-09-11:Harbin
Tom Changle:10501176:88-01-11:Harbin
Billy Black:10501175:93-02-18:Harbin
John Hellen:10501172:93-04-13:Peking
```

13.1.15 算术运算符

gawk 中定义了大量的算术运算符，可以方便地执行算术运算，包括加、减、乘、除、取余、自加、自减，以及赋值、加赋值、减赋值、乘赋值、除赋值、取余赋值等。表 13-6 中列出了 gawk 的算术运算符。

表 13-6 gawk 的算术运算符

运 算 符	说 明
*	将运算符前面的表达式与后面的表达式相乘
/	将运算符前面的表达式与后面的表达式相除
%	将运算符前面的表达式与后面的表达式相除，然后取余
+	将运算符前面的表达式与后面的表达式相加
-	将运算符前面的表达式与后面的表达式相减
=	将运算符后面的表达式的值指派给前面的变量
++	将运算符前面的变量递增
--	将运算符前面的变量递减
+=	将运算符后面的表达式与前面的变量相加，将结果赋值给前面的变量



续表

运 算 符	说 明
-=	将运算符后面的表达式与前面的变量相减，将结果赋值给前面的变量
*=	将运算符后面的表达式与前面的变量相乘，将结果赋值给前面的变量
/=	将运算符后面的表达式与前面的变量相除，将结果赋值给前面的变量
%=	将运算符后面的表达式与前面的变量相除，将余数赋值给前面的变量

例如，下面示例中使用了赋值、加、减、除等运算，对文件 example 中的学生成绩进行统计，最后显示学生的平均成绩，代码如下所示（代码 report）。

```
1 #!/bin/bash           //程序第 1 行的“#!/bin/bash”表示调用 bash 来执行脚本程序
2 #filename:report      //程序第 2 行的“#filename: report”表示该程序的名称为 report
3 if [ $# = 0 ]         //第 3~7 行为 if...then...fi 结构。判断条件为变量 $# 是否等于 0，如果等于
0 则执行               //第 5、6 行，否则跳出 if 结构。其中“$#”表示脚本的位置参数个数。该
                       //如果当前脚本 report 是否指定了一个位置参数（数据文件名），如果没有
                       //提示信息并直接退出程序
                       // if 结构中的 then 语句
4 then                 //显示输出“必须指定一个数据文件！”
5   echo "必须指定一个数据文件！"
6   exit 1             //退出当前脚本
7 fi                   // if 结构中的 fi 语句
8 (date; cat $1) |      //将命令 date 及命令“cat $1”的执行结果通过管道“|”
输出给 gawk            //命令。“date; cat $1”表示先执行 date 命令，再执
                       //（其中$1表示位置参数 1 指定的文件名）
                       //使用 gawk 命令对指定的数据文件进行处理
                       // BEGIN 模式，变量 total 赋初值 0
9 gawk '               //如果变量
10 BEGIN {total=0}     // NR 等于 1（即当前行
11 NR == 1 { print "统计时间：",$2,$3,"$6;print "-----"} //的
NR 等于 1（即当前行  //行号为 1，NR 表示当前记录的记录编号），则显示输出“统计时间：”，$2，$3，“$6”，其中$2 为月份，$3
                       //为日
                       //期，$6 为年份。“print “-----””用于显示输出分隔符
12 NR > 1 { print $1 ":" $5 ; total+= $5} //如果变量
NR 大于 1（即当前行  //的行号大于 1，NR 表示当前记录的记录编号），则显示输出第 1 字段（学生姓名）、第 5 字段（学生成绩），并
                       //将第 5 字段的内容累加到变量 total
13 END {print "-----";printf "平均分为： %4.2f\n", total/(NR-1)} // END 模
式，“print “-----” //用于显示输出分隔符，“printf “平均分为： %4.2f\n”，total/(NR-1)”用于计算输出学生成绩的平均值。
                       //其中“%4.2f”
                       //表示按浮点数进行输出，字段宽度为 4，保留 2 位小数，“\n”表示回车，“total/(NR-1)”用于计算平均值
14 '                  // gawk 代码结束
```

该代码的执行结果如下所示。

```
#chmod u+x report      //赋予文件 report 执行权限
#./report              //不带参数执行程序 report
必须指定一个数据文件
#./report example      //执行程序 report，且指定处理的文件为 example
统计时间： 8 月 24,2008
-----
Zhangsan:83
lisi:99
Wanglijiang:78
```



```

Xuliang:89
Xuxin:99
Wangxuebing:89
lichang:99
Wanglijiang:78
Zhangsansan:83
langxuebing:89
lisibao:99
xiaobao:78
-----

```

平均分为: 88.58

例如, 下面示例中对 `student` 文件中的以 “:” 为字段分隔符的第 4 个字段进行了赋值运算, 将第 4 个字段的内容统一赋值为 `Harbin`, 最后显示输出修改后结果。执行命令如下所示。

```

# gawk -F: '$4="Harbin"' student
Tom Helleen 10501173 93-09-11 Harbin
Mary Degens 10501179 95-05-22 Harbin
Suli Vanlen 10501170 90-09-17 Harbin
Tom Changle 10501176 88-01-11 Harbin
Billy Black 10501175 93-02-18 Harbin
John Hellen 10501172 93-04-13 Harbin

```

可以看到, 第 4 个字段统一被修改为 `Harbin`。

13.1.16 gawk 的流程控制

流程控制语句可以改变 `gawk` 程序中命令的执行顺序。常用的流程控制结构包括 `if...else`、`while` 和 `for`。另外, `break` 和 `continue` 语句与控制结构连用也可以修改命令的执行顺序。

1. if...else 结构

`if...else` 控制结构测试由 `condition` 返回的状态, 并根据该状态转移控制。`if...else` 结构的语法如下所示, `else` 部分是可选的。

```

if ( condition )
    { commands }
[ else
    { commands } ]

```

其中 “`condition`” 为判断条件, “`commands`” 为命令列表, 如果仅为单行命令, 就不需要在命令前后加大括号。当 `condition` 为真, 执行 `condition` 之后的 `commands`; 如果为假, 则执行 `else` 之后的 `commands`。`else` 分支可以省略。

例如, 下面示例是一个简单的 `if` 结构的 `gawk` 程序, 用于判断第 5 个字段的值是否大于 50000, 如果大于 50000 则输出当前行。由于仅有一行命令, 没有使用大括号, 代码如下所示。

```
if ( $5 <= 50000 ) print $0
```

例如, 下面示例是一个 `if...else` 结构的 `gawk` 程序, 用于判断变量 `name` 是否等于 “`Zhangsan`”。如果相等则显示 “`name is Zhangsan`”, 否则显示 “`name is not Zhangsan`”。由于仅有一个命令, 同样没有必要使用大括号, 代码如下所示 (代码 `ifelse`)。

```

1 #filename: ifelse
2 BEGIN {

```



```
3     name="John"
4     if ( name == "Zhangsan" )
5         print " name is Zhangsan"
6     else
7         print " name is not Zhangsan, it is ", nam
8     }
```

该程序的执行结果如下所示。

```
# gawk -f ifelse
name is not Zhangsan, it is John
```

2. while 循环结构

当条件为真时，while 结构将循环执行指定的命令，直到条件为假。while 结构的语法格式如下所示。

```
while ( condition )
{ commands }
```

其中，“condition”为判断条件，“commands”为命令列表，如果仅为单行命令，那么就不需要在命令前后加大括号。只要 condition 为真，while 循环遍历执行 commands。

下面是一个使用简单 while 结构的 gawk 程序，用来显示 2 的 1~8 次幂。由于示例中的 while 循环体包括了多个语句，所以使用了大括号，代码如下所示。

```
1 #filename:while1           //注释，说明该脚本的文件名为 while1
2 BEGIN {                   //第 2~9 行为 gawk 的 BEGIN 模块
3     n = 1                  //变量 n 赋值 1
4     while ( n <= 8 )       //第 4~8 行为 while 循环结构，循环控制条件为“n <= 8”，即当变
量 n 小于                        //等于 8 时继续循环，否则退出 while 循环
5     {                     //表示 while 循环体开始
6         print "2^" n "=", 2**n //输出变量 n 的值，以及 2 的 n 次幂
7         n++                //变量 n 加 1
8     }                     //表示 while 循环体结束
9 }                          //表示 BEGIN 模块结束
```

该程序的执行过程如下所示。

```
# gawk -f while1
2^1= 2
2^2= 4
2^3= 8
2^4= 16
2^5= 32
2^6= 64
2^7= 128
2^8= 256
```

3. for 循环结构

for 循环是另一个常用的循环结构，尤其适于明确知道循环执行次数的情况。for 循环结构的语法格式如下所示。

```
for ( init ; condition; increment )
{ commands }
```

for 结构起初执行 init 语句，该语句通常用于将计数器（也称循环控制变量）设置为 0 或 1。然后，只要 condition 为真就会循环遍历 commands。在每次循环之后，执行 increment 语句。

例如，下面的 gawk 程序 for1 所做的事情与前例 while1 程序类似，用来计算 2 的 1~8 次幂。但此处使用了 for 结构，从而使得程序变得更加简单，如下所示。


```

1 #filename: for1                                //注释,说明该脚本的文件名为 for1
2 BEGIN {                                         //第2~5行为 gawk 的 BEGIN 模块
3     for ( n=1; n <= 8; n++) //第3~4行为 for 循环结构,循环控制条件为“n <= 8”,
//即当变量 n 小于等于 8 时继续循环,否则退出 for 循环。变量 n 赋初值 1,且每次循环之后自加 1
4         print "2^" n "=", 2**n //输出变量 n 的值,以及 2 的 n 次幂
5     }                                           //表示 BEGIN 模块结束

```

该程序的执行过程如下所示。

```

# gawk -f for1
2^1= 2
2^2= 4
2^3= 8
2^4= 16
2^5= 32
2^6= 64
2^7= 128
2^8= 256

```

对于关联数组的处理, gawk 工具还支持另外一种 for 语法,如下所示。

```

for (var in array)
{commands}

```

该 for 结构循环遍历一个名为 array 的关联数组中的所有元素,每次循环,都将 array 相应元素的索引值赋值给 var。

4. break

break 语句将控制转移到 for 或 while 循环之外,用于结束其所在的最内层循环的执行。

5. continue

continue 语句将控制转移到 for 或 while 循环的末尾处,使其所在的最内层循环继续执行下一次迭代。

13.1.17 getline: 控制输入

通常 gawk 从命令行上指定的文件或标准输入中获取输入中数据。使用高级命令 getline,还可以在从哪里以及如何读取输入数据方面有更多的选择。

例如,编写程序 gline,读入用户的一行输入,然后直接显示输出该行文本,如下所示(代码 gline)。

```

1 BEGIN {                                         //第1~4行为 BEGIN 模式
2     getline kk                                 //读入用户的一行输入并保存到变量 kk 中
3     print kk                                  //输出变量 kk 的值
4 }                                              // BEGIN 模式结束

```

该程序的执行过程如下所示。

```

# gawk -f gline                                //使用 gawk 命令执行程序 gline
aaaaaaaaaa                                     //输入“aaaaaaaaaa”
aaaaaaaaaa                                     //显示 aaaaaaaaaa
# gawk -f gline                                //使用 gawk 命令执行程序 gline
aaaaaaaaaa bbbbbbbb ccccccc dddddddddd        //输入“aaaaaaaaaa bbbbbbbb ccccccc dddddddddd”
aaaaaaaaaa bbbbbbbb ccccccc dddddddddd        //显示 aaaaaaaaaa bbbbbbbb ccccccc dddddddddd

```



当给程序 gline 的数据多于一行时,其仍仅处理第一行数据,如下所示。



```
# cat testdata //显示文件 testdata 的内容
11111111111111111111
22222222222222222222
33333333333333333333
44444444444444444444
55555555555555555555
# gawk -f gline < testdata //使用 gawk 命令执行程序 gline, 且将文件 testdata 作为数据输入
11111111111111111111
#
```

例如下面程序中, 在 BEGIN 块中使用 while 循环遍历输入文件中的所有数据行。getline 每次循环读取一行, 再通过 print 输出到标准输出设备。具体代码如下所示 (代码 gline2)。

```
1 BEGIN { //第 1~8 行为 BEGIN 模式
2     i=1; //定义变量 i, 赋初值为 1
3     while ( getline LINE ) //第 3~7 行为 while 循环。循环控制条件由 getline 命令实现。
//每次循环 getline 从指定的数据文件中读入一行数据, 循环会在读取完文件中的所有数据后结束
4     { // while 循环体开始
5         print i": "LINE //输入变量 i 及变量 LINE 的值
6         i++ //变量 i 自加 1
7     } // while 循环体结束
8 } // BEGIN 模式结束
```

该程序的执行过程如下所示。

```
# gawk -f gline2 < testdata //使用 gawk 命令执行程序 gline2, 且将文件 testdata 作为数据输入
1: 11111111111111111111
2: 22222222222222222222
3: 33333333333333333333
4: 44444444444444444444
5: 55555555555555555555
```

在程序 gline2 中为了输出行号定义了变量 i, 每次循环变量 i 自加 1, 从而实现行号的递增。事实上, 可以使用变量 NR 直接输出行号, 从而简化程序, 如下所示 (代码 gline3)。

```
1 BEGIN {
2
3     while ( getline LINE )
4     {
5         print NR": "LINE
6     }
7 }
```

该代码的执行过程如下所示。

```
# gawk -f gline3 < testdata //使用 gawk 命令执行程序 gline3, 且将文件 testdata 作为数据输入
1: 11111111111111111111
2: 22222222222222222222
3: 33333333333333333333
4: 44444444444444444444
5: 55555555555555555555
```

13.1.18 协进程

协进程 (coprocess) 是指与另一个进程并行运行的进程。gawk 从版本 3.1 开始就可以通过协进程方式直接与某个后台进程进行信息交换。协进程可用于客户端/服务器环境中 (例如构建 SQL 前端与后端), 或者通过网络与远程系统交换数据。

gawk 通过在程序名称前面添加双向管道运算符 “|&” 来标识在后台运行的协进程。而协

进程必须是一个过滤器（例如从标准输入设备读入并写入到标准输出设备），在每完成一行的数据处理之后必须清空输出，而不是在缓存了多行数据后再统一输出。gawk 程序通过双向管道建立与协进程的连接，可以从协进程中读取或写入数据。

例如，下面的 Shell 程序 `toupper` 用于将输入的字符转换成大写，代码如下所示（代码 `toupper`）。

```
1 #!/bin/bash           //程序第1行的“#!/bin/bash”表示调用 bash 来执行脚本程序
2 #filename:toupper     //程序第2行的“#filename: toupper”表示该程序的名称为 toupper
3 while read LINE       //第3~6行为 while...do...done 循环结构。每次循环 read 命令会从数据输入读取
//行数据保存到变量 LINE 中。当 read 命令读完所有输入数据，循环结束
4 do                   // while 循环中的 do 语句，表示循环体开始
5   echo $LINE | tr '[a-z]' '[A-Z]' //使用 tr 命令将保存在变量 LINE 中的字符转换为大写字母，并通过
//echo 命令显示输出
6 done                // while 循环中的 done 语句，表示循环体结束
```

该程序的执行过程如下所示。

```
#chmod u+x toupper      //赋予文件 toupper 执行权限
# echo abcdef | ./toupper //通过管道将“abcdef”输出给 toupper
ABCDEF
```

可以看到，通过程序 `toupper`，将小写字符串“abcdef”转换为“ABCDEF”。

程序 `gline4` 调用 `toupper` 作为协进程，对从指定文件读入的数据转换成大写，并输出到标准输出设备上（代码 `gline4`），如下所示。

```
1 { print $0 |& "./toupper" //将当前行通过双向管道“|&”输出到当前目录下的程序 toupper 进行处
//理。其中“$0”表示当前记录行，程序名两边必须带双引号
2   "./toupper" |& getline LINE //将来自 toupper 的标准输出重定向给 getline 命令，getline 命令将获得
//的数据保存到变量 LINE 中
3   print LINE }           //输入变量 LINE 的值
```

例如，定义数据文件 `testdata2`，其中内容全部由小写字母组成。通过 `gawk` 命令执行 `gline4` 程序对 `testdata2` 进行处理，执行过程如下所示。

```
# cat testdata2           //显示文件 testdata2 的内容
aaaaaaaaa
cccccccc
dddddddd
wwwwwwww
# gawk -f gline4 < testdata2 //使用 gawk 命令执行程序 gline4，且将文件 testdata2 作为
数据输入
AAAAAAA
CCCCCCC
DDDDDD
WWWWWWW
```

13.1.19 从网络获取数据

在协进程的基础上，gawk 还可以通过网络交换数据。gawk 可以通过 IP 网络连接到另一个系统上的某个进程进行信息交换。当用户指定了某个以“/inet/”开头的特殊文件名时，gawk 将使用网络连接来处理用户的请求。以“/inet/”开头的特殊文件的格式如下。

```
/inet/protocol/local-port/remote-host/remote-port
```



其中“protocol”通常是 TCP，也可以是 UDP。“local-port”表示本地端口，如果用户希望让 gawk 自动选择一个端口，可以将 local-port 的值设置为 0。“remote-host”为远程主机的 IP 地址或域名。“remote-host”表示远端主机的地址，“remote-port”表示远端端口号，可以指定为 http 或 ftp 地址。

例如，下面的示例程序从 www.mywebsite.com 读取文件 example，文件 example 保存在 www.mywebsite.com 的/glxt/student 目录中。程序代码如下所示（gline4）。

```
1 BEGIN { //第1~6行为 BEGIN 模式
2     server = "/inet/tcp/0/www.mywebsite.com/80" //定义变量 server，其值为
// "/inet/tcp/0/www.mywebsite.com/80"，表示使用 TCP 协议连接服务器 www.mywebsite.com 的 80
端口。使用
//的本地端口没有特别指定
3     print "GET /glxt/student/example" |& server //通过双向管道“|&”向远端服务
//器发送 GET 请求，获取/glxt/student/example 文件
4     while ( server |& getline ) //第 4、5 行为 while 循环，该循环
//连续读取文件 example 中的数据，每次读取一行且保存到变量$0 中
5         print $0 //输出变量$0 的值
6     } // BEGIN 模式结束
```

该程序的执行结果如下所示。

```
#gawk -f gline5
Zhangsan      1977      male      computer      83
lisi          1989      male      math           99
Wanglijiang   1990      female    chinese        78
Xuliang       1977      male      economic       89
Xuxin         1986      female    english        99
Wangxuebing   1978      male      math           89
lichang       1989      male      math           99
Wanglijiang   1990      female    chinese        78
Zhangsansan   1977      male      computer       83
langxuebing   1978      male      math           89
lisibao       1989      male      math           99
xiaobao       1990      female    chinese        78
```

13.1.20 gawk 的应用示例

(1) 例如，下面示例程序中输出文件 example 中的第 5 列（成绩）和第 1 列（姓名），执行结果如下所示。

```
# gawk '{ print $5, $1}' example
83 Zhangsan
99 lisi
78 Wanglijiang
89 Xuliang
99 Xuxin
89 Wangxuebing
99 lichang
78 Wanglijiang
83 Zhangsansan
89 langxuebing
99 lisibao
78 xiaobao
```

从输出结果可以看到，首先输出了文件 example 第 5 列内容，然后是第 1 列，且列之间以空格为分隔符。

(2) 例如, 下面示例中输出 `example` 文件中姓徐 (“Xu”) 的同学的姓名及出生年份、性别, 执行结果如下所示。

```
# gawk '/Xu/{ print $1, $2, $3}' example
Xuliang 1977 male
Xuxin 1986 female
```

(3) 使用正则表达式中的 “~” 和 “^” 符号可以进一步实现匹配操作。例如, 下面示例中输出 `example` 文件中所有包含字符 “l” 的行, 执行结果如下所示。

```
# gawk '/l/' example
Zhangsan      1977    male    computer    83
lisi          1989    male    math        99
Wanglijiang   1990    female  chinese     78
Xuliang       1977    male    economic    89
Xuxin         1986    female  english     99
Wangxuebing   1978    male    math        89
lichang       1989    male    math        99
Wanglijiang   1990    female  chinese     78
Zhangsansan   1977    male    computer    83
langxuebing   1978    male    math        89
lisibao       1989    male    math        99
xiaobao       1990    female  chinese     78
```

使用 “^”, 可以设定强制在行首进行匹配。例如, 下面示例中使用 “^” 和 “~”, 来选择第一个字段中第一个字符为 “l” 的行, 执行结果如下所。

```
# gawk '$1~/^l/' example
lisi          1989    male    math        99
lichang       1989    male    math        99
langxuebing   1978    male    math        89
lisibao       1989    male    math        99
```

(4) 使用正则表达式中的字符集 “[]”, 可以进一步实现匹配操作。例如, 下面示例中搜索 `example` 文件中第一个字段的首字符是 W 或 X 的行, 且输出第 1 列、第 5 列, 执行结果如下所示。

```
# gawk '$1~/^[WX]/{print $1"的成绩为:" $5}' example
Wanglijiang 的成绩为:78
Xuliang 的成绩为:89
Xuxin 的成绩为:99
Wangxuebing 的成绩为:89
Wanglijiang 的成绩为:78
```

(5) 使用正则表达式的元字符 “\$”, 可以强制在行末尾或字段末尾进行匹配。例如, 下面示例中搜索 `example` 文件中第二个字段的末尾是字符是 8 的行, 且输出第 1 列、第 5 列, 执行结果如下所示。

```
# gawk '$2~/8$/{print "姓名: "$1,"出生于: "$2,"成绩为:" $5}' example
姓名: Wangxuebing 出生于: 1978 成绩为:89
姓名: langxuebing 出生于: 1978 成绩为:89
```

13.2 流编辑器 sed

`sed` (stream editor, 流编辑器) 是一个批处理 (非交互式) 编辑器, 可以对来自文件或标准输入的输入流进行转换。`sed` 通常被用作管道中的过滤器。由于 `sed` 仅仅对其输入进行一遍扫描, 因此比其他交互式编辑器 (如 `ed`) 更加高效。



sed 一次处理一行文本并把处理结果输出送往标准输出设备（屏幕）。sed 把当前处理的行存储在时称为模式空间（pattern space）的临时缓冲区中。一旦 sed 完成对模式空间中行的处理（也就是执行完这一行中的 sed 命令），模式空间中的行就被送往屏幕（除非命令是删除行或者打印到打印机）。行被处理完成以后，就被移出模式空间，程序接着读入下一行，处理、显示、移出……文件输入的最后一行被处理完以后，sed 结束。通过在临时缓冲区存储每一行，然后在缓冲区中操作该行，保证了原始文件不会被破坏。

本节在介绍 sed 命令时，很多示例都使用了 example2 数据文件。该文件内容如下所示。

```
# cat example2
这是第一行。
这是第二行。
第三行。
第四行，文件的中部。
这是第五行。
六行
第七行。
这是第八行，文件的最后一行。
```

13.2.1 sed 的基本语法

sed 允许用户在命令行输入命令和文件名，然后在标准输出上输出命令的执行结果。sed 是非破坏性的，在使用重定向操作存储输出以前，sed 不会修改原文件的内容。默认情况下，所有行都输出到屏幕。

sed 的命令行语法格式如下。

```
sed [-n] program [file-list]
sed [-n]-f program-file[file-list]
```

其中，“program”表示命令行中的 sed 程序。该格式允许直接编写简单短小的 sed 程序而不需要创建一个单独的程序文件。

“program-file”表示一个 sed 程序文件。

“file-list”表示 sed 将要处理的普通数据文件，即 sed 的输入文件。如果没有指定任何输入文件，那么 sed 将标准输入作为它的输入。sed 命令的选项说明如表 13-7 所示。

表 13-7 sed 命令的选项说明

选 项	说 明
--file program-file, -f program-file	使 sed 从 program-file 指定的文件中而不是命令行中读取程序。该选项可以在命令行中多次使用
--in-place[=suffix] -i [suffix]	就地编辑文件。如果没有该选项，sed 将输出送入标准输出。该选项使 sed 将输出送入原来的输入文件中。一旦指定了后缀名 suffix，sed 将备份原来的输入文件。该备份文件名以原文件名后接后缀 suffix 命名。如果想在原文件名和后缀 suffix 之间有一个句点，那么必须要在后缀 suffix 中包含一个句点
--help	显示 sed 的帮助信息
--quiet, --slient -n	指示 sed 除非使用了 p 指令或标志，否则不会将文本复制到标准输出

通常，为了让 sed 执行更长或更为复杂的指令，可以将 sed 程序保存到文件中。在需要调用该程序文件时，配合使用 sed 的“-f”选项调用执行。例如，下面示例中编写 sed 程序文

件 show3-8, 用于显示输出文件 example2 的第 3~8 行, 代码如下所示 (代码 show3-8)。

```
3, 8 p
```

其中 “p” 为 sed 指令, 表示打印输出。通过 sed 命令执行该 sed 程序, 如下所示。

```
# sed -n -f show3-8 example2
```

```
第三行。
```

```
第四行, 文件的中部。
```

```
这是第五行。
```

```
六行
```

```
第七行。
```

```
这是第八行, 文件的最后一行。
```

```
#
```

可以看到, 该程序将第 3~8 行打印输出。

除了在命令行使用 sed 命令的 “-f” 选项调用 sed 程序外, 还可以编写独立脚本程序。独立脚本文件需要具备读权限和执行权限, 且在程序首行添加 “#!/bin/sed -f”, 该行代码告诉 Shell 使用 “/bin/sed -f” 运行该脚本。例如, 编写独立程序 del35 删除文件 example2 的第 3~5 行 (代码 del35), 代码如下所示。

```
#!/bin/sed -f
```

```
3,5 d
```

该程序的执行过程如下所示。

```
# chmod u+x del35
```

```
//赋予文件 del35 执行权限
```

```
# ./del35 example2
```

```
//执行程序 del35, 且指定处理的文件为 example2
```

```
这是第一行。
```

```
这是第二行。
```

```
六行
```

```
第七行。
```

```
这是第八行, 文件的最后一行
```

```
#
```



sed 从命令行所指定的文件或标准输入中获取输入流。除非明确指定输出目标, 否则 sed 将把结果输出到标准输出。

13.2.2 sed 编辑器基础

sed 程序由符合如下语法的一行或多行命令构成。

```
[address][, address ] instruction [ argument-list ]
```

其中的 “address” 是可选的。如果省略 address, 那么 sed 将对输入中的所有行进行处理。“instruction” 是一些用于改变文本的编辑指令。“address” 用于定位指令 instruction 进行操作的文本行。“argument-list” 中参数的数目和类型取决于指令 instruction。如果想把多个 sed 指令放在同一行, 需要使用分号 “;” 来分隔这些指令。

sed 按照如下的流程处理输入数据。

(1) 从文件列表 file-list 或标准输入中读取一行文本。

(2) 从命令行中的 program 或命令文件 program-file 中读取第一条指令。如果命令行中的地址 address 选定了该读入的文本行, 那么对该文本行执行指令 instruction 所指定的操作。



(3) 从 `program` 或 `program-file` 中读取下一条命令。如果命令行中的地址 `address` 选择了该读入的文本行，那么该文本行（可能已经被前面的指令修改过）执行指令 `instruction` 所指定的操作。

(4) 重复第 (3) 步直到 `program` 或 `program-file` 中的指令被全部执行。

(5) 如果输入中还有未被处理的文本行，跳转到第 (1) 步，否则结束。

13.2.3 地址

可以通过地址 (`address`) 来确定需要处理的数据行。地址可以以数字、正则表达式或二者结合的方式表示。

当地址由数字构成时，数字表示的是行号，而行号作为地址可以用来选择某一行。作为一个特例，符号 “\$” 表示输入中的最后一行。用逗号分隔的两个行号表示以这两行为起止的行的范围（包括行数表示的那两行）。正则表达式作为地址可以用来选择那些包含与正则表达式相匹配的字符串的行。

指令之前可以有一个、两个地址，或者没有地址（通过行号或正则表达式选定）。如果没有指定任何地址，那么 `sed` 将选择所有的行，使得指令对输入的所有行进行处理。如果只有一个地址将使得指令作用于由该地址选择的每一行输入。如果指定了两个地址，将使得指令作用于成组的数据行。其中组是这样划分的：第 1 个地址选择第 1 组中的首行，第 2 个地址选择后续与之匹配的下一行，该行就是第 1 组的尾行；如果没有找到与第 2 个地址匹配的行，那么第 2 个地址指向文件的末尾；在选出每组的尾地址后，`sed` 开始重复选择过程，寻找与第 1 个地址匹配的下一行，该行就是下一组的首行；`sed` 循环执行这一过程直到将整个文件处理完毕。

例如，将文件 `example2` 中的一部分（第 3~6 行）从屏幕显示输出。执行命令如下所示。

```
# sed -n '3,6 p' example2
第三行。
第四行，文件的中部。
这是第五行。
六行
#
```

例如，将文件 `example2` 中的一部分（第 3~6 行）复制到另一个文件 `example2_36` 中。执行命令如下所示。

```
# sed -n '3,6 w example2_36' example2
# cat example2_36
第三行。
第四行，文件的中部。
这是第五行。
六行
#
```

可以看到，`sed` 程序中的行号 3 和 6 指定了 `sed` 进行复制的范围。

13.2.4 指令

`sed` 指令告诉 `sed` 对通过地址被选择的行进行什么样的处理，如打印、删除或修改等。如果没有给定地址，`sed` 将处理所有输入的行。如果一个 `sed` 指令以 “!” 开头，则不论是

什么操作，都将对所有的行发生作用，而不仅仅是被选定的行。sed 的常用指令如表 13-8 所示。

表 13-8 sed 的常用指令

指 令	名 称	说 明
d	删除指令 (delete)	删除指令导致 sed 不输出被选择的行，并且不继续完成对该行的后续处理过程
n	下一条指令 (next)	下一条指令输出当前选择的行 (如果可以)，然后从输入中读入下一行，并且从 program 或 program-file 中的下一条指令开始对新读入的行进行处理
a	追加指令 (append)	追加指令在当前选择的行之后插入一行或多行文本。如果在 a 指令前有两个地址，那么它将在选定的每一行之后添加文本。早期版本的 sed 并不支持具有两个地址的 a 指令。如果 a 指令之前没有地址，它将在每行之后添加文本
i	插入指令 (insert)	插入指令与 a 指令相同，只不过 i 指令是将文本添加到选定的行之前
c	修改指令 (change)	修改指令与 a 指令和 i 指令类似。不同的是它将选定的行修改为新的文本。如果指定的是一个地址范围，c 指令将把整个范围内的行替换成新文本
s	替换指令 (substitute)	sed 中的替换指令与 vim 中的类似
p	打印指令 (print)	打印指令将选定的行写入标准输出
w file	写指令 (write)	写指令与 p 指令类似，不同的是它将输出写入由 file 指定的文件中。w 指令后接一个空格和作为输出的文件名
r file	读指令 (read)	读指令读出指定文件的内容并添加选定的行之后。r 指令后接一个空格和作为输入的文件名
q	退出指令 (quit)	退出指令使得 sed 立即结束执行

13.2.5 打印：p 指令

打印指令 p (print) 将选定的行写入标准输出。写入动作是立即发生的，因此并不反映后续指令的执行结果。p 指令会覆盖 sed 命令行中的 “-n” 选项，而 “-n” 选项阻止 sed 将选定的行复制到标准输出上。

```
例如，下面示例中输出文件 example2 中的第 3~6 行，执行命令如下所示。  
# sed -n '3,6 p' example2  
第三行。  
第四行，文件的中部。  
这是第五行。  
六行
```

```
例如，下面示例中搜索文件 example2 中包含汉字“六”的行，找到后显示输出。执行过程如下所示。  
# sed '/六/ p' example2  
这是第一行。  
这是第二行。  
第三行。  
第四行，文件的中部。  
这是第五行。  
六行  
六行  
第七行。  
这是第八行，文件的最后一行。
```



其中“/六/”是一个简单的正则表达式，sed 将选定那些包含与该模式相匹配的文本行。但从输出结果可以看到，由于没有使用 sed 命令的“-n”选项，所有的行均被显示输出。而下面示例添加了“-n”选项后，即仅输出了选定的行，如下所示。

```
# sed -n '/六/ p' example2
六行
```



除非指定选项“-n”，否则 sed 将在标准输出上输出所有的行，而无论其是否被选定。当在命令行中使用“-n”选项时，sed 仅仅在标准输出上输出选定的行，例如那些被 p 指令选定的行。

13.2.6 删除：d 指令

删除指令 d (delete) 告诉 sed 不输出被选择的行，并且不继续完成对该行的后续处理过程。例如，下面示例在输出结果中删除了 example2 文件中的第 3~6 行，如下所示。

```
# sed '3,6 d' example2
这是第一行。
这是第二行。
第七行。
这是第八行，文件的最后一行。
```

13.2.7 追加：a 指令

a 指令 (append) 在当前选择的行之后插入一行或多行文本。如果在 a 指令前有两个地址，那么它将在选定的每一行之后添加文本。如果 a 指令之前没有地址，它将在每行之后添加文本。a 指令的具体格式如下。

```
[address [, address ]] a\
text\
text\
...
text
```

除最后一行外，所添加文本的每一行都必须以反斜杠结尾（反斜杠用于指代行末的换行符）。没有以反斜杠结尾的行被视作要添加的文本的尾行。

例如，下面的 sed 程序 append48 分别在第 4 行及最后一行添加一行分隔符，如下所示（代码 append48）。

```
4 a\
-----
```

```
$ a\
```

其中“\$”表示文件的尾行。该程序的执行过程如下所示。

```
# sed -f append38 example2
这是第一行。
这是第二行。
第三行。
第四行，文件的中部。
-----
这是第五行。
六行
第七行。
```


这是第八行，文件的最后一行。

```
-----
#
```

从输出结果可以看到，在第 4 行及第 8 行后添加了分隔符行。由于需要将所有内容显示输出，所以 sed 命令没有使用“-n”选项。

13.2.8 插入：i 指令

插入指令 i (insert) 与 a 指令类似，只不过 i 指令是将文本添加到选定的行之前。例如，下面示例中使用 i 指令，在每一个包含“第”字的行前添加一行文字“---下面一行有"第"字出现-----”，程序如下所示（代码 insertd）。

```
/第/ i\
```

```
---下面一行有"第"字出现-----
```

该程序的执行结果如下所示：

```
# sed -f insertd example2
```

```
---下面一行有"第"字出现-----
```

```
这是第一行。
```

```
---下面一行有"第"字出现-----
```

```
这是第二行。
```

```
---下面一行有"第"字出现-----
```

```
第三行。
```

```
---下面一行有"第"字出现-----
```

```
第四行，文件的中部。
```

```
---下面一行有"第"字出现-----
```

```
这是第五行。
```

```
六行
```

```
---下面一行有"第"字出现-----
```

```
第七行。
```

```
---下面一行有"第"字出现-----
```

```
这是第八行，文件的最后一行
```

可以看到，除了第 6 行外，所有包含“第”字的行前均被添加了指定的文字。

13.2.9 修改：c 指令

修改指令 c (change) 与 a 指令或 i 指令十分类似。不同的是，a 指令将选定的行修改为新的文本。如果指定的是一个地址范围，c 指令将把整个范围内的行替换成新文本。

例如，下面示例中使用 c 指令修改了文件 example2 中第 3~5 行的内容，第 3 行修改为“aaaaaaaaaaaaaaaa”，第 4 行修改为“bbbbbbbbbbbbbbbb”，第 5 行修改为“cccccccccccccccc”。程序如下所示（代码 change35）。

```
3,5 c\
```

```
aaaaaaaaaaaaaaaa\
```

```
bbbbbbbbbbbbbbbb\
```

```
cccccccccccccccc
```

使用 sed 命令调用程序 change35，修改文件 example2 的第 3~5 行，执行结果如下所示。

```
# sed -f change35 example2
```

```
这是第一行。
```

```
这是第二行。
```



```
aaaaaaaaaaaaaaaa
bbbbbbbbbbbbbbbb
cccccccccccccccc
六行
第七行。
这是第八行，文件的最后一行
```

13.2.10 退出：q 指令

退出指令 q (quit) 使得 sed 立即结束执行。例如，下面使用了 q 指令，使得 sed 仅仅显示文件 example2 开头的三行，效果等同于 “head -3 example2”，如下所示。

```
# sed '3 q' example2
这是第一行。
这是第二行。
第三行。
#
```

13.2.11 替换：s 指令

sed 中的替换指令 s (substitute) 与 vim 中的替换指令类似。s 指令的具体格式如下所示。

```
[address[, address] ] s/pattern/replacement-string/[g][p][w file]
```

其中，“pattern” 是一个正则表达式，可以由空格和换行符以外的任何字符作为分隔符，默认是斜杠 (“/”)。“replacement-string” 紧跟在第 2 个分隔符后面，同时必须以与前面相同的分隔符结束，即最后一个 (第 3 个) 分隔符是必需的。replacement-string 可以包含与符号 “&”，sed 将用 pattern 匹配的字符串来替换 “&” 符号。除非使用 g 标志，否则 s 指令将只对每个选定行中的第 1 个与 pattern 匹配的字符串进行替换。

- 全局 (global) 标志 g: 使得 sed 的 s 指令对选定行中匹配 pattern 的所有字符串进行替换。
- 打印 (print) 标志 p: 将所有应用了替换操作的行输出到标准输出中。
- 写 (write) 标志 w: 与 p 标志类似，但它将输出送入 file 指定的文件中。w 标志后接一个空格和作为输出的文件名。

例如，下面示例中使用 s 指令将文件 example2 中每行首次出现的 “这是” 替换为 “This is”，代码如下所示 (代码 substitutes)。

```
s/这是/This is /p
```

使用 sed 命令调用程序 substitutes 对文件 example2 执行替换操作，并将发生替换操作的每一行显示输出。执行结果如下所示。

```
# sed -n -f substitutes example2
This is 第一行。
This is 第二行。
This is 第五行。
This is 第八行，文件的最后一行
```

例如，下面示例中使用 s 和 w 指令将文件 example2 中每行首次出现的 “这是” 替换为 “This is”，并将改变后的行输出到文件 temp 中，代码如下所示 (代码 substitutes2)。

```
s/这是/This is /w temp
```


使用 sed 命令调用程序 substitute2 对文件 example2 执行替换操作，并将发生替换操作的行保存到文件 temp 中。执行结果如下所示。

```
# sed -f substitutes2 example2
This is 第一行。
This is 第二行。
第三行。
第四行，文件的中部。
This is 第五行。
六行
第七行。
This is 第八行，文件的最后一行。
# cat temp                                //显示文件 temp 的内容，其中保存了发生替换操作的行
This is 第一行。
This is 第二行。
This is 第五行。
This is 第八行，文件的最后一行
```

例如，下面程序中对指定的多个文件进行处理，将文件中出现的所有“IS”替换为“is”，将“MY”替换为“my”，将“PAPER”替换为“paper”。该程序的具体代码如下所示（代码 substitute3）。

```
1 #!/bin/bash                                //程序第1行的“#!/bin/bash”表示调用 bash 来执行脚本程序
2 #filename:substitute3                      //程序第2行的“#filename: substitute3”表示该程序的名称为
substitute3
3 for file                                  //第3~10行为 for 循环结构。每次循环读取该脚本的一个位置参数并
赋值
//给变量 file，直至读取完全部参数
4 do                                        // for 结构的 do 语句
5   echo $file                             //显示变量 file 的内容
6   mv -f $file ftemp                      //将变量 file 指定的文件强制更名为 ftemp
7   sed 's/IS/is/g'                        //第7~9行为 sed 程序。将文件 ftemp 中出现的所有“IS”替换为“is”，
//将“MY”替换为“my”，将“PAPER”替换为“paper”
8       s/MY/my/g                          // for 循环结束
9       s/PAPER/paper/g ' ftemp > $file    //强制删除文件 ftemp
10 done                                    //表示 for 循环体结束
11 rm -f ftemp                             //强制删除文件 ftemp
```

该程序的执行过程如下所示。

```
# cat file1                                //显示文件 file1 的内容
This IS a PAPER. It IS John's PAPER.
John IS a boy,he likes to read MY PAPER.
# cat file2                                //显示文件 file2 的内容
This PAPER belongs to John. It IS written by John.
John IS a school student,he is MY good friend.
# cat file3                                //显示文件 file3 的内容
Apple IS red. Orange IS green.
MY PAPER IS on the top of desk.
#chmod u+x substitute3                    //赋予文件 substitutes3 执行权限
# ./substitute3 file1 file2 file3         //执行程序 substitutes3，且带参数 file1、
file2、file3
file1
file2
file3
# cat file1                                //显示文件 file1 的内容
This is a paper. It is John's paper.
```




```
John is a boy,he likes to read my paper.
# cat file2 //显示文件 file2 的内容
This paper belongs to John. It is written by John.
John is a school student,he is my good friend.
# cat file3 //显示文件 file3 的内容
Apple is red. Orange is green.
my paper is on the top of desk.
```

从输出结果可以看到，文件 file1、file2 和 file3 均执行了替换操作。

例如，文件 multiline 中有 10 行文本，通过 sed 程序将其中第 1~3 行中出现的所有“apple”替换为“banana”，将第 4~6 行中首次出现的“book”替换为“cup”，将第 7、8 行删除，将第 9、10 行中首次出现的“Linux”替换为“windows”，编写程序如下所示（代码 substitute4）。

```
1,3 s/apple/banana/g
4,6 s/book/cup/
7,8 d
9,10 s/Linux/windows/
```

数据文件 multiline 的内容如下所示。

```
# cat multiline
1.This is an apple and the apple is red.
2.This is an apple and the apple is red.
3.This is an apple and the apple is red.
4.It is a book on the table.
5.It is a book on the table.
6.It is a book on the table.
7.The school is my middle school.
8.The school is my middle school.
9.Linux is an operate system.
10.Linux is an operate system.
```

使用 sed 命令调用程序 substitute4，对 multiline 进行处理，执行结果如下所示。

```
# sed -f substitute4 multiline
1.This is an banana and the banana is red.
2.This is an banana and the banana is red.
3.This is an banana and the banana is red.
4.It is a cup on the table.
5.It is a cup on the table.
6.It is a cup on the table.
9.windows is an operate system.
10.windows is an operate system.
```

13.2.12 下一条：n 指令

下一条指令 n (next) 输出当前选择的行（如果可以），然后从输入中读入下一行，并且从 sed 程序中的下一条指令开始对新读入的行进行处理。

例如，下面示例使用 n 指令跳过文件 exmaple2 中的第 3 行，程序如下所示（代码 next1）。

```
3 n
p
```

该程序的执行过程如下所示。

```
# sed -n -f next1 example2
这是第一行。
这是第二行。
第四行，文件的中部。
```


这是第五行。
六行
第七行。
这是第八行，文件的最后一行。
#

例如，下面示例使用 `n` 指令跳过文件 `file1` 中的包含“`This`”的行，程序如下所示(代码 `next2`)。

```
/This/ n
p
```

该程序的执行过程如下所示。

```
# cat file1                                //显示文件 file1 的内容
This is a paper. It is John's paper.
John is a boy,he likes to read my paper.
# sed -n -f next2 file1                    //使用 sed 命令执行程序 next2 对文件 file1 进行处理
John is a boy,he likes to read my paper.//已跳过包含 “This” 的行
#
```

13.2.13 从文件读入：r 指令

读指令 `r` (`read`) 读出指定文件的内容并将其添加在选定的行之后。`r` 指令后接一个空格和作为输入的文件名。

例如，在文件 `example2` 中出现“六”字的行的下方读入文件 `example`，命令如下所示。

```
# sed '/六/ r example' example2
这是第一行。
这是第二行。
第三行。
第四行，文件的中部。
这是第五行。
六行
Zhangsan      1977      male      computer      83
lisi          1989      male      math          99
Wanglijiang   1990      female    chinese       78
Xuliang       1977      male      economic      89
Xuxin         1986      female    english       99
Wangxuebing   1978      male      math          89
lichang       1989      male      math          99
Wanglijiang   1990      female    chinese       78
Zhangsansan   1977      male      computer      83
langxuebing   1978      male      math          89
lisibao       1989      male      math          99
xiaobao       1990      female    chinese       78
第七行。
这是第八行，文件的最后一行。
#
```

13.2.14 写入文件：w 指令

写指令 `w` (`write`) 与 `p` 指令类似，不同的是 `w` 指令将输出写入指定的文件中，而 `p` 指令输出到标准输出设备。`w` 指令后接一个空格和作为输出的文件名。

例如，将文件 `example2` 中第 3~6 行写入文件 `line36` 中，命令如下所示。

```
# sed '3,6 w line36' example2
这是第一行。
这是第二行。
```



```
第三行。  
第四行，文件的中部。  
这是第五行。  
六行  
第七行。  
这是第八行，文件的最后一行。  
# cat line36 //显示文件 line36 的内容  
第三行。  
第四行，文件的中部。  
这是第五行。  
六行  
#
```

13.2.15 sed 的控制结构

sed 具有一些控制结构，利用控制结构可以控制程序的流程走向。表 13-9 中列出了常用的控制结构。

表 13-9 sed 的控制结构

格 式	说 明
!	取反操作，使得 sed 后面与其同一行的指令作用于没有被该指令选择的每一行地址上。例如，“3!d”将删除除第 3 行外的所有行，而“\$!p”将显示除最后一行外的所有行
{ }	大括号将一组指令括起来，用于定义指令组。该组指令将作用于它前面的地址所选定的行。使用分号可将一行中的多条指令分隔开
分支指令	sed 的 info 页中指出，sed 不适合使用分支指令，并且建议如果需要使用它们，最好使用 awk 或 Perl 来编写程序
:label	标识 sed 程序中的一个位置。标号 label 将作为跳转目标，适用于分支指令 b 或 t
b [label]	无条件转移到 label 标识的命令。如果没有指出分支目标 label，则跳过当前行中剩下的指令并从输入中读取下一行
t [label]	如果从输入中读取的行使得 s 指令匹配成功，则转移到 label 标识的命令。如果没有指出分支目标 label，则跳过当前行中剩下的指令并从输入中读取下一行

例如，使用取非运算符“!”将文件 example2 中除了第 3~6 行之外的其他行复制到另一个文件 example2_x36 中，命令如下所示。

```
# sed -n '3,6 !w example2_x36' example2  
# cat example2_x36  
这是第一行。  
这是第二行。  
第七行。  
这是第八行，文件的最后一行。  
#
```

13.2.16 Pattern 区和 Hold 区

sed 中有两个缓冲区：Pattern 区和 Hold 区。所有的命令都工作在 Pattern 区上。Pattern 区中保存着 sed 刚刚从输入中读取的行。Hold 区作为临时缓冲区，可以在操作 Pattern 区中的数据时用来暂存数据。在将数据放入 Hold 区之前，其内容为空。用于 Pattern 区和 Hold 区之间进行数据传送的指令如表 13-10 所示。

表 13-10 Pattern 区和 Hold 区的指令说明

指 令	说 明
g	将 Hold 区中的内容复制到 Pattern 区中。Pattern 区中原来的内容将会丢失
G	将一个换行符和 Hold 区中的内容追加到 Pattern 区中的内容之后
h	将 Pattern 区中的内容复制到 Hold 区中。Hold 区中原来的内容将会丢失
H	将一个换行符和 Pattern 区中的内容附加到 Hold 区中的内容之后
x	交换 Pattern 区和 Hold 区中的内容

例如，下面的 sed 程序使用 G 指令在输入文件中的每一行后面添加一空行，如下所示。

```
# sed 'G' example2
这是第一行。

这是第二行。

第三行。

第四行，文件的中部。

这是第五行。

六行

第七行。
这是第八行，文件的最后一行。

#
```

例如，下面示例中通过 Pattern 区和 Hold 区的配合使用，反序输出文件 example2 的内容，程序如下所示（代码 pandh）。

```
2,$G
h
$!d
```

其中，“\$”表示最后一行，“2,G”表示第 2 行至最后一行，“!”表示取反，“\$!”表示除最后一行外的其他行。该代码的执行结果如下所示。

```
# sed -f pandh example2
这是第八行，文件的最后一行。
第七行。
六行
这是第五行。
第四行，文件的中部。
第三行。
这是第二行。
这是第一行。
```

sed 程序 pandh 对文件 example2 的具体处理流程如下所示。

- (1) sed 将输入的第 1 行（“这是第一行。”）读入到 Pattern 区中。
- 指令“2,\$G”不处理输入的第 1 行，因为其地址使得指令 G 只处理第 2 行至最后一行。
- 指令 h 将第 1 行从 Pattern 区复制到 Hold 区。
- 指令“\$!d”删除了 Pattern 区中的内容。因为此后 Pattern 区中没有任何内容，所以 sed 什么也不显示。



- (2) sed 将输入中的第 2 行 (“这是第二行。”) 读入到 Pattern 区中。
- 指令 “2, \$G” 将 Hold 区中的内容 “这是第一行。” 添加到 Pattern 区中。此时 Pattern 区中的内容是 “这是第二行。\\n 这是第一行。”。
 - 指令 h 将 Pattern 区中的内容复制到 Hold 区中。
 - 指令 “\$!d” 删除输入中的第 2 行。因为第 2 行被删除，所以 sed 什么也不显示。
- (3) sed 将输入中的第 3 行 (“第三行。”) 读入到 Pattern 区中。
- 指令 “2, \$G” 将 Hold 区中的内容 “这是第二行。\\n 这是第一行。” 添加到 Pattern 区中。此时 Pattern 区中的内容是 “第三行。\\n 这是第二行。\\n 这是第一行。”。
 - 指令 h 将 Pattern 区中的内容复制到 Hold 区中。
 - 指令 “\$!d” 删除输入中的第 3 行。因为第 3 行被删除，所以 sed 什么也不显示。
- (4) sed 将输入中的第 4 行 (“第四行，文件的中部。”) 读入到 Pattern 区中。
- 指令 “2, \$G” 将 Hold 区中的内容 “第三行。\\n 这是第二行。\\n 这是第一行。” 添加到 Pattern 区中。此时 Pattern 区中的内容是 “第四行，文件的中部。\\n 第三行。\\n 这是第二行。\\n 这是第一行。”。
 - 指令 h 将 Pattern 区中的内容复制到 Hold 区中。
 - 指令 “\$!d” 删除输入中的第 4 行。因为第 4 行被删除，所以 sed 什么也不显示。
- (5) sed 将输入中的第 5 行 (“这是第五行。”) 读入到 Pattern 区中。
- 指令 “2, \$G” 将 Hold 区中的内容 “第四行，文件的中部。\\n 第三行。\\n 这是第二行。\\n 这是第一行。” 添加到 Pattern 区中。此时 Pattern 区中的内容是 “这是第五行。\\n 第四行，文件的中部。\\n 第三行。\\n 这是第二行。\\n 这是第一行。”。
 - 指令 h 将 Pattern 区中的内容复制到 Hold 区中。
 - 指令 “\$!d” 删除输入中的第 5 行。因为第 5 行被删除，所以 sed 什么也不显示。
- (6) sed 将输入中的第 6 行 (“六行”) 读入到 Pattern 区中。
- 指令 “2, \$G” 将 Hold 区中的内容 “这是第五行。\\n 第四行，文件的中部。\\n 第三行。\\n 这是第二行。\\n 这是第一行。” 添加到 Pattern 区中。此时 Pattern 区中的内容是 “六行\\n 这是第五行。\\n 第四行，文件的中部。\\n 第三行。\\n 这是第二行。\\n 这是第一行。”。
 - 指令 h 将 Pattern 区中的内容复制到 Hold 区中。
 - 指令 “\$!d” 删除输入中的第 6 行。因为第 6 行被删除，所以 sed 什么也不显示。
- (7) sed 将输入中的第 7 行 (“第七行。”) 读入到 Pattern 区中。
- 指令 “2, \$G” 将 Hold 区中的内容 “六行\\n 这是第五行。\\n 第四行，文件的中部。\\n 第三行。\\n 这是第二行。\\n 这是第一行。” 添加到 Pattern 区中。此时 Pattern 区中的内容是 “第七行。\\n 六行\\n 这是第五行。\\n 第四行，文件的中部。\\n 第三行。\\n 这是第二行。\\n 这是第一行。”。
 - 指令 h 将 Pattern 区中的内容复制到 Hold 区中。
 - 指令 “\$!d” 删除输入中的第 7 行。因为第 7 行被删除，所以 sed 什么也不显示。
- (8) sed 将输入中的第 8 行 (“这是第八行，文件的最后一行。”) 读入到 Pattern 区中。
- 指令 “2, \$G” 将 Hold 区中的内容 “第七行。\\n 六行\\n 这是第五行。\\n 第四行，文

件的中部。\\n 第三行。\\n 这是第二行。\\n 这是第一行。”添加到 Pattern 区中。此时 Pattern 区中的内容是“这是第八行，文件的最后一行。\\n 第七行。\\n 六行\\n 这是第五行。\\n 第四行，文件的中部。\\n 第三行。\\n 这是第二行。\\n 这是第一行。”。

- 指令 h 将 Pattern 区中的内容复制到 Hold 区中。
- 指令 “\$!d” 不对输入中的最后一行进行处理，所以最后一行没有被删除，输出“这是第八行，文件的最后一行。”
- 显示 Pattern 区中的内容。

13.2.17 sed 的应用示例

(1) 例如，下面程序中在文件 example2 的第 2 行后面添加两行新文本，然后使用 p 指令输出第 8 行文本，如下所示（代码 append2）。

```
2 a\
<-----这是新添加的行 \
<-----这是新添加的行
8 p
```

使用 sed 命令执行程序 append2，对文件 example2 进行处理，如下所示。

```
# sed -f append2 example2
这是第一行。
这是第二行。
<-----这是新添加的行
<-----这是新添加的行
第三行。
第四行，文件的中部。
这是第五行。
六行
第七行。
这是第八行，文件的最后一行。
这是第八行，文件的最后一行
#
```

可以看到，sed 命令行中没有使用“-n”选项，因此 sed 将文件中的所有行显示一次，而由于程序末尾使用了 p 指令，所以多显示一次第 8 行。

(2) 例如，在文件 example2 的第 2 行后添加两行新文本，然后使用 d 指令删除第 2 行，程序如下所示（代码 append3）。

```
2 a\
<-----这是新添加的行 \
<-----这是新添加的行
2 d
```

使用 sed 命令执行程序 append3，对文件 example2 进行处理，如下所示。

```
# sed -f append3 example2
这是第一行。
<-----这是新添加的行
<-----这是新添加的行
第三行。
第四行，文件的中部。
这是第五行。
六行
第七行。
```



这是第八行，文件的最后一行

(3) sed 程序中指令的编写顺序会直接导致输出结果的不同。例如，数据文件 multiline2 中有 5 行文本，程序 substitute5 将先执行第 1~3 行的替换，再执行第 2~5 行的替换，而程序 substitute6 将先执行第 2~5 行的替换，再执行第 1~3 行的替换。由于指令的执行顺序不同，输出结果也不同。程序 substitute5 如下所示（代码 substitute5）。

```
1,3 s/book/cup/
2,5 s/book/mirror/
```

程序 substitute6 如下所示（代码 substitute6）。

```
2,5 s/book/mirror/
1,3 s/book/cup/
```

执行结果如下所示。

```
# cat multiline2                                //显示文件 multiline2 的内容
1.The book is on the top of the box.
2.The book is on the top of the box.
3.The book is on the top of the box.
4.The book is on the top of the box.
5.The book is on the top of the box.
# sed -f substitute5 multiline2                  //执行 sed 程序 substitute5
1.The cup is on the top of the box.
2.The cup is on the top of the box.
3.The cup is on the top of the box.
4.The mirror is on the top of the box.
5.The mirror is on the top of the box.
# sed -f substitute6 multiline2                  //执行 sed 程序 substitute6
1.The cup is on the top of the box.
2.The mirror is on the top of the box.
3.The mirror is on the top of the box.
4.The mirror is on the top of the box.
5.The mirror is on the top of the box.
```

(4) 例如，下面示例中使用正则表达式作为地址，将文件 example3 中每行文本前添加一个 Tab。程序执行过程如下所示。

```
# cat example2
这是第一行。
这是第二行。
第三行。
第四行，文件的中部。
这是第五行。
六行
第七行。
这是第八行，文件的最后一行。
# sed 's/^./\t&/' example3
1.This is line one.
2.This is line two.
3.This is the third line.
4.The fourth.
5.This is line five.
#
```

可以看到，经过 sed 程序处理后的文本行，向右侧移动了一个 Tab。其中，“/^./”为正则表达式，匹配首字符为任意字符的行，“\t”表示 Tab，“&”代表正则表达式“/^./”所匹配到的值。该 sed 程序通过使用 s 指令在每一行文本前添加了 Tab。

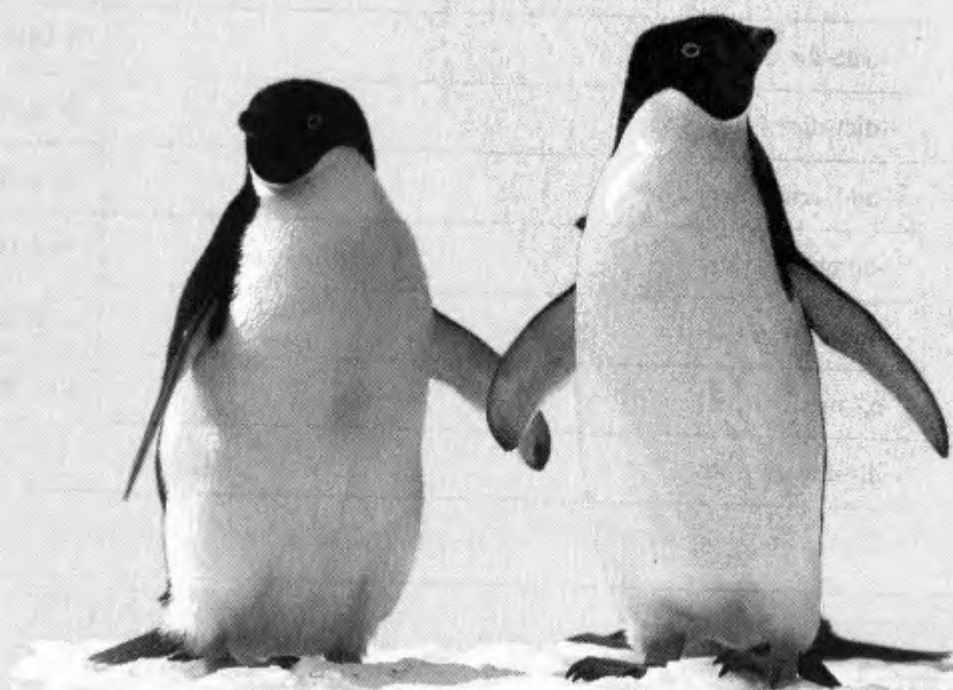
LINUX

第14章

其他常用的文本编辑命令

除了 vi(vim)、emacs 等功能强大的文本编辑工具外, Red Hat Linux 9 还提供了一些具有针对性的文本处理工具, 如 `wc` 命令可以计算文本的字数, `aspell` 命令可以对拼写进行检查, `join` 命令可以连接两个文本, `sort` 命令可以对文件排序, `more` 命令可以分屏显示文本内容等。

熟练掌握并综合利用这些工具可以有效提高文本处理效率, 加快文本编辑速度。





14.1 aspell 命令：检查文件的拼写错误

aspell 命令用于检查文件的拼写。该命令有许多选项参数，是一个功能强大的拼写检查工具，其语法格式如下所示。

```
aspell [-c, check <file>] [-a, pipe] [-l, list] [[dump] confi] [filter] [-?, help] [-v, version] [ --conf=<str>]
      [--conf-dir=<str>] [--data-dir=<str>] [--dict-dir=<str>] [--add, rem-filter=<str>] [--home-dir=<str>]
      [--mode=<str>] [-e, --mode=email] [ -d, --master=<str>] [-H, --mode=sgml] [ -t, --mode=tex] [--[dont-]reverse]
      [ --per-conf=<str>] [--prefix=<str>] [--repl=<str>] [-b, -x, --[dont-]backup]
      [ --[dont-]tex-check-comments]
      [ --add, rem-tex-command=<str>] [--add, rem-sgml-extension=<str>] [--add, rem-sgml-check=<str>]
      [ --sug-mode=<str>]
```

其中各选项的具体说明如表 14-1 所示。

表 14-1 aspell 命令的选项说明

选 项	说 明
-c, check <file>	检查指定的文件
-a, pipe	"ispell -a"兼容模式
-l, list	从标准输入产生一个拼写错误的单词的列表
[dump] confi	从标准输出显示当前配置
filter	通过过滤器过滤掉某些单词
-?, help	显示帮助信息
-v, version	显示版本信息
--conf=<str>	主配置文件
--conf-dir=<str>	本地的主配置文件
--data-dir=<str>	本地的语言数据文件
--dict-dir=<str>	本地的主单词列表
--add, rem-filter=<str>	增加或移除一个过滤器
--home-dir=<str>	个人目录
--mode=<str>	过滤器模式，包括 none、url、email、sgml、tex
-e, --mode=email	电子邮件模式
-d, --master=<str>	主目录的主名称

续表

选 项	说 明
-H,--mode=sgml	HTML/SGML 模式
-t,--mode=tex	tex 模式
--per-conf=<str>	个人配置文件
--prefix=<str>	前缀目录
--repl=<str>	替换列表文件名
-b, -x,--[dont-]backup	创建.bak 备份文件
--[dont-]tex-check-comments	检测 tex 注释
--add, rem-tex-command=<str>	tex 注释
--add, rem-sgml-extension=<str>	sgml 文件扩展
--add, rem-sgml-check=<str>	sgml 标签检测
--[dont-]reverse	反转列表
--sug-mode=<str>	建议模式, 包括 ultra、fast、normal、bad-spellers

例如，检查文件 mypaper 中的拼写错误，命令如下所。

```
# cat mypaper //显示 mypaper 文件的内容
hello
This is a test paper.
I liko Linux.
It im a very perfeer operating system.
we are Linux lovers.
#aspell -c mypaper //对 mypaper 进行拼写检查
```

执行该命令后，系统进入交互模式，如图 14-1 所示。



图 14-1 aspell 命令的交互模式

图 14-1 所示窗口的上部显示的是文件 mypaper 的文本内容，其中高亮显示的是有拼写错误的单词。窗口下部给出了针对错误单词的修改意见，还包括 Ignore（忽略）、Ignore all（全部忽略）、Replace（替换）、Replace all（替换全部）、Add（添加）、Exit（退出）等命令。



例如，将文件中拼错的单词输出到标准输出，命令如下所示。

```
# cat mypaper //显示 mypaper 文件的内容
hello
This is a test paper.
I liko Linux.
It im a very perfecr operating system.
we are Linux lovers.
# aspell -l < mypaper //将 mypaper 中拼错的单词输出到标准输出
liko
im
perfecr
```

例如，输出使用的检错词典，命令如下所示。

```
# aspell dump master //输出检错词典
Hauser's
refinances
catchall
subtrahend
Brandeas's
rarebits
petrologist's
... ..
```



aspell 命令还有一个兼容命令 ispell，功能及用法与 aspell 相似，但 ispell 不包含“-m”、“-n”、“-P”、“-S”、“-w ARG”及“-T ARG”选项。

14.2 cat 命令：显示文本内容

cat 是 Concatenate 的缩写，用于把一个文件发送到标准输出设备，与 MS-DOS 或 Windows 下的 type 命令相似。cat 命令可以对任意一个文件使用，屏幕会一次显示文件的所有内容，中间不停顿，不分屏。除了显示文件内容外，cat 还具有由键盘读取数据、将多个文件合并的功能。cat 命令的语法格式如下所示。

```
cat [-A,--show-all] [-b,--number-nonblank] [-e] [-E,--show-ends] [-n,--number] [-S,--squeeze-blank] [-t]
[ -T,--show-tabs] [-v,--show-nonprinting] [ --help] [ ] [ ] [ ] [ ]
```

其中各选项的说明如表 14-2 所示。

表 14-2 cat 命令选项说明

选 项	说 明
-A,--show-all	等价于-vET
-b,--number-nonblank	对非空行输出行编号
-e	等价于-vE
-E,--show-ends	在每行结束处显示\$
-n,--number	对输出的所有行编号
-S,--squeeze-blank	不输出多行空行

续表

选 项	说 明
-t	与-vT 等价
-T,--show-tabs	将跳格字符 Tab 显示为^I
-v,--show-nonprinting	使用^和 M 符号显示除 LFD 和 Tab 外的
--help	显示帮助信息并离开

例如，显示 `hello.c` 文件内容，命令如下所示。

```
$ cat hello.c
hello world!
```

该命令可配合重定向 “>” 建立小型文本。例如，将键盘键入内容输出重定向到文件 `example1` 中，按 `Ctrl+d` 键存盘退出，命令如下所示。

```
# cat > example1
aa bb cc dd
bb cc dd ee
cc dd ee ff
# cat example1
aa bb cc dd
bb cc dd ee
cc dd ee ff
```

`cat` 命令可以联合输出多个文件的内容，如下所示。

```
# cat example1 //显示文件 example1 的内容
aa bb cc dd
bb cc dd ee
cc dd ee ff
# cat example2 //显示文件 example2 的内容
dd ee ff gg
ee ff gg hh
ff gg hh ii
# cat example1 example2 //联合显示文件 example1、example2 的内容
aa bb cc dd
bb cc dd ee
cc dd ee ff
dd ee ff gg
ee ff gg hh
ff gg hh ii
```

如果将上例中的 `example1` 和 `example2` 合并后放入新文件 `example3`，命令如下所示。

```
# cat example1 example2 > example3 //合并后输出到 example3 文件
# cat example3
aa bb cc dd
bb cc dd ee
cc dd ee ff
dd ee ff gg
ee ff gg hh
ff gg hh ii
```

如果在显示输出的每一行前自动添加行号（空白行除外），可使用选项 “-b”，命令如下所示。

```
# cat -b example3
 1 aa bb cc dd
 2 bb cc dd ee
 3 cc dd ee ff
```



```
4 dd ee ff gg
5 ee ff gg hh
6 ff gg hh ii
```



使用选项 “-n” 会为所有行加上行号，即使空行也不例外。

例如，显示文件 mylist 的内容到标准输出设备，命令如下所。

```
#cat mylist //显示文件 mylist 的内容到标准输出
Hello , I am a student.
I am from China.
Today is 2008-08-08.
I am at DESHENG MENG.
Welcome to China.
```

例如，对文件 mylist 的输出内容进行编号，命令如下所示。

```
# cat -n mylist //显示文件 mylist 的内容到标准输出，并对输出进行编号
1 Hello , I am a student.
2 I am from China.
3 Today is 2008-08-08.
4 I am at DESHENG MENG.
5 Welcome to China.
```

例如，显示文件 mylist 的内容到标准输出设备，并在每一行的末尾添加 “\$” 符号，命令如下所示。

```
# cat -E mylist //显示文件 mylist 的内容到标准输出，并在每一行的末尾以$结束
Hello , I am a student.$
I am from China.$
Today is 2008-08-08.$
I am at DESHENG MENG.$
Welcome to China.$
```

14.3 cmp 命令：比较文件差异

cmp 命令用于对任意类型的两个文件进行比较，并且把比较结果输出到标准输出。默认情况下，如果文件是相同的，那么它们将不进行任何输出。如果两个文件不同，那么第一次出现不同的字符和行数将被报告给用户。字符和行数从 1 开始计数。cmp 命令的语法格式如下所示。

```
cmp [-l] [-i SKIP --ignore-initial=SKIP] [-i SKIP1:SKIP2] [-s --quiet --silent] [-n LIMIT --bytes=LIMIT]
[ -v] [ --help] [ -b --print-bytes] FILE1 [FILE2]
```

其中各选项的具体说明如表 14-3 所示。

表 14-3 cmp 命令的选项说明

选 项	说 明
-l	对于出现的差异显示字节数（十进制）和差异（八进制）
-i SKIP --ignore-initial=SKIP	跳过起始输入的 SKIP 个字节
-i SKIP1:SKIP2	跳过文件 1 的 SKIP1 个字节，文件 2 的 SKIP2 个字节

续表

选 项	说 明
-s --quiet --silent	不对差异信息进行显示，仅返回状态值。状态值有如下几个。 0：文件相同 1：文件存在差异 >1：存在错误
-n LIMIT --bytes=LIMIT	比较至多 LIMIT 个字节
-v	显示版本信息
--help	显示帮助信息
-b --print-bytes	显示差异字节
FILE1, FILE2	指定比较的文件

例如，对文件 mypaper.doc 和 mypaper.bak 两个文件进行比较，命令如下所示。

```
# cat mypaper.doc //显示文件mypaper.doc的内容
hello
This is a test paper.
I like Linux.
It is a very perfect operating system.
we are Linux lovers.
# cat mypaper.bak //显示文件mypaper.bak的内容
hello
This is a test paper.
I liko Linux.
It im a very perfecr operating system.
we are Linux lovers.
# cmp mypaper.doc mypaper.bak //比较文件mypaper.doc和mypaper.bak
mypaper.doc mypaper.bak differ: byte 34, line 3
```

例如，显示两个文件所有不同的信息，对于出现的差异以十进制显示字节数，以八进制显示差异，命令如下所示。

```
# cat mypaper.doc //显示文件mypaper.doc的内容
hello
This is a test paper.
I like Linux.
It is a very perfect operating system.
we are Linux lovers.
# cat mypaper.bak //显示文件mypaper.bak的内容
hello
This is a test paper.
I liko Linux.
It im a very perfecr operating system.
we are Linux lovers.
# cmp -l mypaper.doc mypaper.bak //显示两文件所有的不同
34 145 157
47 163 155
62 164 162
```

14.4

csplit 命令：按指定行数分割文件

使用 csplit 命令后，文件的输出将按指定的行数进行分割，并将结果输出到标准输出设



备。该命令的语法格如下所示。

```
csplit [-k, --keep-files][ -s, --quiet, --silent][ -z, --elide-empty-files][ -b,
--suffix-format=FORMA] [ --help][ --version]
[ -f, --prefix=PREFIX][ -f, --prefix=PREFIX][ -n, --digits=DIGITS]
```

其中各选项的具体说明如表 14-4 所示。

表 14-4 csplit 命令的选项说明

选 项	说 明
-k, --keep-files	当出现错误时，不删除输出文件
-z, --elide-empty-files	删除空输出文件
-b, --suffix-format=FORMA	使用 sprintf 的输出格式，而不再使用 %d
-f, --prefix=PREFIX	使用 PREFIX 前缀，而不是使用'xx'
-n, --digits=DIGITS	指定使用数字的位数，默认为 2
-s, --quiet, --silent	静默模式，不显示输出文件的大小
--help	显示帮助信息
--version	显示版本信息

例如，通过命令 csplit ，分割文件 dict，前 50 行为一个文件，剩下的为另一个文件。执行命令如下所示。

```
# ls -l //显示文件列表
总用量 1444
-rw-r--r-- 1 root root 1451172 8月 7 20:14 dict
# csplit dict 50 //分割文件
438
1450734
# ls -l //显示文件列表
总用量 2872
-rw-r--r-- 1 root root 1451172 8月 7 20:14 dict
-rw-r--r-- 1 root root 438 8月 7 23:23 xx00
-rw-r--r-- 1 root root 1450734 8月 7 23:23 xx01
```

14.5 cut 命令：文件剪切

cut 命令用来从指定文件中选择指定的行，并输出到标准输出，其语法格式如下所示。

```
cut [-n][ --only-delimited][ -b, --bytes=LIST][ -c, --characters=LIST][ -d,
--delimiter=DELIM]
[ --output-delimiter=STRING][ -s, --only-delimited][ -f, --fields=LIST]
```

其中各选项的具体说明如表 14-5 所示。

表 14-5 cut 命令的选项说明

选 项	说 明
-n	与-b 选项同时使用，不分割多字节字符
--only-delimited	不输出不包含定界符的行
-b, --bytes=LIST	只输出指定的字节数
-c, --characters=LIST	只输出指定的字符

续表

选 项	说 明
-d, --delimiter=DELIM	指定分隔符，而不是默认的 Tab
--output-delimiter=STRING	指定输出分隔符，默认输出分隔符与输入分隔符相同
-s, --only-delimited	仅输出包含分隔符的行
-f, --fields=LIST	只输出指定的域，同时输出任何不包含分隔符的行，除非指定了-s 选项

例如，通过命令 `cut`，显示文件 `abc.c` 文件中的每一行的第二个字符，命令如下所示。

```
# cat abc.doc           //显示文件的内容
kdjflkajsdklfjjjjjjjjj
sajklfdjdlskjflkjflksd
jdsklfjlkdsjflkjsdlkfjsdlkjf
love apple
ls
ksdjflksdjlkf
ja
# cut -c 2 a.c           //显示文件每行第二个字符
d
a
d
o
s
s
a
```

例如，通过命令 `cut`，显示文件 `abc.doc` 的每一行第一个字节的内容以及每一行第二个字节的内容，如下所示。

```
# cat abc.doc           //显示文件的内容
kdjflkajsdklfjjjjjjjjj
sajklfdjdlskjflkjflksd
jdsklfjlkdsjflkj
love apple
ls
ksdjflksdjlkf
ja
# cut -b 1 abc.doc       //显示每一行的第一个字节
k
s
j
l
l
k
j
# cut -b 2 abc.doc       //显示每一行的第二个字节
d
a
d
o
s
s
a
```

例如，显示 `student1` 文件中每行第 1~10 个字符，命令如下所示。

```
# cat student1           //显示文件 student1 的内容
Tom is a tall man
Mike is Tom's brother
Mary is a beautiful girl
```



```
Jean is Mary's sister
# cut -c1-10 student1           //显示每行第 1~10 个字符
Tom is a t
Mike is To
Mary is a
Jean is Ma
```

cut 命令并没有改变源文件的内容，只是改变了显示方式，如下所示。

```
# cat student1
Tom is a tall man
Mike is Tom's brother
Mary is a beautiful girl
Jean is Mary's sister
```

14.6 col 命令：过滤控制字符

col 命令可以过滤掉反向换行符，使输出按正确的格式显示。通常使用 col 命令将空格符替换为 Tab，这对于处理 nroff 和 tbl 的输出是非常有用的。col 命令的语法格式如下所示。

```
col [-b][-f][-x][-p][-l num]
```

col 命令的选项说明如表 14-6 所示。

表 14-6 col 命令的选项说明

选 项	说 明
-b	过滤掉控制字符 BACKSPACE
-f	fine 模式，允许向前的半换行符
-x	将 Tab 替换为多个空格
-p	强制对未知的控制序列不进行更改
-l num	指定缓冲区最少行数，默认为 128 行

例如，将 ls 的 man 信息重定向到文件 ls.man.txt，并通过命令 col 过滤掉 RLF、HRLF 以及 BACKSPACE 等控制字符，如下所示。

```
# man ls|col -b >ls.man.txt           //将 ls 的帮助信息重定向输出，并且过滤掉 RLF、HRLF 等字符
```

14.7 colrm 命令：删除指定的列

colrm 命令从指定文件中过滤掉指定的列。若在命令行中只指定一个列数值，那么将删除指定列之后的所有列；若指定两个列数值，将删除两个值之间的所有列。colrm 的输入为标准输入设备，输出被发送到标准输出设备。该命令的语法格式如下所示。

```
colrm [startcol [endcol]]
```

常用的 colrm 命令选项如表 14-1 所示。

例如，通过命令“colrm 2 5”，删除指定的第 2~5 列，当从标准输入输入“abcdefghijklmn”时，输出的结果为“afghijklmn”，当从标准输入输入的结果为“1234567890”时，输出的结果为“167890”，如下所示。

表 14-7 常用的 colrm 命令选项

选 项	说 明
starcol	指定 colrm 命令起始过滤的列
endcol	指定 colrm 命令结束过滤的列

```
# colrm 2 5 //删除指定的第 2 列和第 5 列
abcdefgijklmn //输入
afghijklmn //输出
1234567890 //输入
167890 //输出
```

例如，首先使用命令“cat abc”，显示文件的内容：第一行为“I am a Linux learner”，第二行为“I am from Beijing.”。通过命令“colrm 5<abc”，删除文件的每一行第 5 列之后的内容，输出的结果：第一行为“I am”，第二行为“I am”。通过命令“colrm 5 8 <abc”，删除文件第 5~8 列之间的内容，输出的结果：第一行为“I aminux learner.”，第二行为“I amm Beijing.”。执行命令如下所示。

```
# cat abc //显示文件 abc 的内容
I am a Linux learner.
I am from Beijing.
# colrm 5 <abc //删除文件的每一行第 3 列之后的内容
I am
I am
# colrm 5 8 <abc //删除文件第 5~ 8 列之间的内容
I aminux learner.
I amm Beijing.
```

14.8 comm 命令：对已排序文件逐行进行比较

comm 命令对两个已排序文件逐行进行比较，输出结果由 3 列组成，其中第 1 列表示仅在第 1 个文件中出现的行，第 2 列表示仅在第 2 个文件出现的行，第 3 列表示在 2 个文件中都存在的行。comm 命令的语法格式如下：

```
comm [-[1][2][3]] file1 file2
```

其中各选项的具体说明如表 14-8 所示。

表 14-8 comm 命令的选项说明

选 项	说 明
-1	显示输出第 1 列，即仅在第 1 个文件中出现的行
-2	显示输出第 2 列，即仅在第 2 个文件中出现的行
-3	显示输出第 3 列，即仅在第 3 个文件中出现的行
--help	显示此帮助信息并离开
--version	显示版本信息并离开

例如，对文件 student1 和 student2 进行比较，并显示两者的异同，命令行如下。

```
# cat student1 //显示 student1 内容
Tom is a tall man
Mike is Tom's brother
Mary is a beautiful girl
```



```
Jean is Mary's sister
# cat student2                                //显示 student2 内容
Tom is Mary's brother
Mike is Tom's brother
Mary is a beautiful girl
Jean is Tom's sister
# sort student1 > student1_1                  //对 student1 排序, 保存到 student1_1
# sort student2 > student2_2                  //对 student2 排序, 保存到 student2_2
# cat student1_1 student2_2                  //显示 student1_1、student2_2 的内容
Jean is Mary's sister
Mary is a beautiful girl
Mike is Tom's brother
Tom is a tall man
Jean is Tom's sister
Mary is a beautiful girl
Mike is Tom's brother
Tom is Mary's brother
# comm student1_1 student2_2                  //对 student1 和对 student2 进行比较
Jean is Mary's sister
    Jean is Tom's sister
        Mary is a beautiful girl
        Mike is Tom's brother
Tom is a tall man
    Tom is Mary's brother
```

14.9 diff 命令：文件比较

diff 命令用于比较两个文本文件，并显示它们的不同，其语法格式如下所示。

```
diff [-a --text][ -b --ignore-space-change][ -B --ignore-blank-lines][ -i
--ignore-case][ -q --brief]
[ -c -C NUM --context[=NUM]][ -d --minimal][ -e --ed] [-r --recursive][ -n
--rcs][ -N --new-file]
[ -p --show-c-function][ --paginate][ -s --report-identical-files][ -t
--expand-tabs] [ -v --version][ --help]
[ -u -U NUM --unified[=NUM]][ -y --side-by-side][ -D NAME --ifdef=NAME]
[ --new-group-format=<模式>][-S FILE --starting-file=FILE][ --from-file=FILE1]
```

其中各选项的具体说明如表 14-9 所示。

表 14-9 diff 命令的选项说明

选 项	说 明
-a, --text	把所有文件看成文本形式，并对它们进行逐行比较，即使其不是文本文件
-b, --ignore-space-change	忽略空格
-B, --ignore-blank-lines	忽略空白行
-i, --ignore-case	忽略大小写
-q, --brief	只报告两个文件是否相同，不报告详细的不同信息
-c, -C NUM, --context[=NUM]	输出复制内容的行数
-d, --minimal	改变算法，使得 diff 命令可能找到一个较小的文件差异集
-e, --ed	产生 ed 脚本
-r, --recursive	比较目录时，递归比较子目录
-n, --rcs	RCS 格式

续表

选 项	说 明
-N, --new-file	在目录比较中, 如果一个文件只出现在第一个目录中, 那么认为它在另外一个目录中也出现, 并且为空
-p, --show-c-function	显示每次变化所在的 C 函数
--paginate	通过 pr 输出, 并且标记页数
-s, --report-identical-files	当比较的两个文件相同时, 报告信息
-t, --expand-tabs	输出时, Tab 键采用空格键来输出
-u, -U NUM --unified[=NUM]	输出统一文件的行数
-y, --side-by-side	输出时使用对比格式, 即输出两列
-D NAME, --ifdef=NAME	输出合并的文件, 显示文件中出现的 “#ifdef NAME” 差异
--new-group-format=<模式>	用按照 if-then-else 方式输出第二个文件中的若干行内容
-S FILE, --starting-file=FILE	当比较文件时, 从指定的文件进行比较
--from-file=FILE1	同所有的操作数比较指定的文件, 可以是目录
-v, --version	显示版本信息
--help	显示帮助信息



diff 命令比较两个指定文件的内容。如果指定的最后两个参数都是目录, 那么 diff 命令将比较两目录下的同名文件; 如果同时指定 “-r” 选项, 那么对这两个目录下的子目录也将进行递归比较。

diff 命令的输出结果有特定的格式, 如表 14-10 所示。

表 14-10 diff 文件的输出结果说明

输出结果形式	说 明
n ₁ a n ₂	表示第一个文件的 n ₁ 行添加了...成为第二个文件的 n ₂ 行
n ₁ ,n ₂ c n ₃ ,n ₄	表示将第一个文件的 n ₁ ~n ₂ 行 (在<号之后的内容) 改变成第二个文件的 n ₃ ~n ₄ 行 (在>号之后的内容)
n ₁ ,n ₂ d n ₃	表示将第一个文件中 n ₁ ~n ₂ 行删除成为文件 2 中的 n ₃ 行

a 表示添加, c 表示改变, d 表示删除

例如, 对文件 student1、student2 进行比较, 并输出文件的不同之处, 命令如下所示。

```
# cat student1 //student1 文件内容
Tom is a tall man
Mike is Tom's brother
Mary is a beautiful girl
Jean is Mary's sister
# cat student2 //student2 文件内容
Tom is Mary's brother
Mike is Tom's brother
Mary is a beautiful girl
Jean is Tom's sister
# diff student1 student2 //比较结果
1c1
< Tom is a tall man
```



```

---
> Tom is Mary's brother
4c4
< Jean is Mary's sister
---
> Jean is Tom's sister

```

例如，通过命令 `diff`，比较文件夹 `/etc/ppp` 和文件夹 `/etc/httpd` 的不同，命令如下所示。

```

# ls ppp
chap-secrets      ip-down           ipv6-down         pap-secrets
firewall-masq     ip-down.ipv6to4  ipv6-up           peers
firewall-standalone ip-up            mytest           plugins
ioptions          ip-up.ipv6to4    options          pppoe-server-options
# ls httpd
build conf conf.d logs modules mytest run
# diff ppp httpd           //对 ppp 目录和 httpd 目录进行比较
Only in httpd: build
Only in ppp: chap-secrets
Only in httpd: conf
Only in httpd: conf.d
Only in ppp: firewall-masq
Only in ppp: firewall-standalone
Only in ppp: ioptions
Only in ppp: ip-down
Only in ppp: ip-down.ipv6to4
Only in ppp: ip-up
Only in ppp: ip-up.ipv6to4
Only in ppp: ipv6-down
Only in ppp: ipv6-up
Only in httpd: logs
Only in httpd: modules
Only in ppp: options
Only in ppp: pap-secrets
Only in ppp: peers
Only in ppp: plugins
Only in ppp: pppoe-server-options
Only in httpd: run

```

例如，对文件 `123.doc` 和 `abc.doc` 进行比较，命令如下所示。

```

# cat 123.doc           //显示文件 123.doc 的内容
i am from Harbin
i like sports
especiallly basketball and football
my favorite football club is AC
i will love it for ever
# cat abc.doc           //显示文件 abc.doc 的内容
i am a student
i am from Harbin
i like sports
especiallly basketball and football
my favorite football club is IN
i will love it for ever
# diff 123.doc abc.doc   //比较两个文件的不同
0a1
> i am a student
4c5
< my favorite football club is AC

```



```
---
```

```
< my favorite football club is IN
```

通过命令 `diff`，比较文件 `abc.doc` 和文件 `123.doc` 的不同，并生成差异文件 `file.diff`，此差异文件可以作为将来的补丁，如下所示。

```
# diff -ruN abc.doc 123.doc >file.diff      //比较文件 abc.doc 和 123.doc，生成差异文件
# cat file.diff                             //显示差异文件的内容
--- abc 2007-05-15 10:05:28.000000000 +0800
+++ 123 2007-05-15 10:06:08.000000000 +0800
@@ -1,6 +1,5 @@
-i am a student
 i am from Harabin
 i like sports
  especially basketball and football
-my favorite football club is IN
+my favorite football club is AC
 i will love it for ever
```

通过命令 `patch`，使用差异文件 `file.diff` 对文件 `123.doc` 进行打补丁。打完补丁之后，文件 `123.doc` 的内容和文件 `abc.doc` 的内容相同，命令如下所示。

```
# patch -p1 <file.diff                      //打补丁
missing header for unified diff at line 3 of patch
can't find file to patch at input line 3
Perhaps you used the wrong -p or --strip option?
The text leading up to this was:
-----
|--- abc.doc      2007-05-15 10:05:28.000000000 +0800
|+++ 123.doc      2007-05-15 10:06:08.000000000 +0800
-----
File to patch: 123.doc                      //123.doc 为输入的文件
patching file 123.doc
Reversed (or previously applied) patch detected! Assume -R? [n] y
# cat abc.doc                               //显示文件 abc.doc 的内容
i am a student
i am from Harbin
i like sports
  especially basketball and football
my favorite football club is IN
i will love it for ever
# cat 123.doc                               //显示打完补丁后的文件 123.doc 的内容
i am a student
i am from Harabin
i like sports
  especially basketball and football
my favorite football club is IN
i will love it for ever
```

14.10

diffstat 命令：diff 结果的统计信息

`diffstat` 命令读取 `diff` 命令的输出，然后显示插入、删除、修改等操作的统计信息。该命令的语法格式如下所示：

```
diffstat [-c] [-f NUM] [-p NUM] [-u] [-w NUM] [-v]
```



其中各选项的具体说明如表 14-11 所示。

表 14-11 diffstat 命令的选项说明

选 项	说 明
-c	在每一行前添加注释符号“#”
-f NUM	格式 (0=简洁, 1=常规)
-n NUM	指定文件名的最少长度
-p NUM	指定路径被剥离的层次
-u	不对输入列表进行排序
-w NUM	指定输出的最大宽度
-V	显示版本号

例如，利用 diffstat 命令对 diff 命令的输出结果进行统计，如下所示。

```
# diff mypaper.bak mypaper.doc //使用diff命令比较mypaper.bak与mypaper.doc
3,4c3,4
< I liko Linux.
< It im a very perfecr operating system.
---
> I like Linux.
> It is a very perfect operating system.
# diff mypaper.bak mypaper.doc | diffstat //统计diff信息
0 files changed //未改变
```

14.11 ed 命令：行文本编辑器

ed 命令用来编辑文本，是一个比较简单的基于行的文本编辑器。该命令的语法格式如下所示。

```
ed [-G][-s][--help] [--version][-p string] [file]
```

其中各选项的具体说明如表 14-12 所示。

表 14-12 常用的 ed 命令选项

选 项	说 明
-G	支持向后兼容
-s	压缩调试模式
--help	显示 ed 命令的帮助信息
file	指定读入的文件
--version	显示 ed 命令的版本信息
-p string	指定提示符

ed 有两种模式：命令模式和输入模式。当首次启动 ed 时，ed 运行在命令模式下，命令从标准输入设备读入并被执行。常用的 ed 命令选项如表 14-13 所示。

表 14-13 常用的 ed 命令选项

选 项	说 明
a	将文本追加到文件
c	修改当前行文本
i	在当前行处插入文本
e file	编辑指定文件
f file	设置默认文件名
q	退出 ed
Q	无条件退出 ed，即强制退出

例如，在命令模式下执行命令查找替换操作，如下所示。

```
,s/old/new/g
```

该命令表示用字符串“new”替换所有字符串“old”。

如果在启动 ed 时指定了文件名，则在编辑器缓存中会创建指定文件的副文，对副本所做的所有修改不会直接应用于文件本身。如果在退出 ed 时，没有明确地使用“w”命令执行写入操作，对副本的修改将会丢失，不会被存入文件。



对文本所做的所有编辑只有明确地使用“w”命令执行写入操作，才会被存入文件。

例如，通过命令 ed，对文本进行编辑，并将编辑的文本保存为 myfile，执行过程如下所示。

```
# ed //进入 ed 编辑器
a //追加文本，进入编辑模式
aaaaaaaaaaaaaaaa //编辑文件
bbbbbbbbbbbbbbbb
cccccccccccccccc
. //切换到命令模式
w file //写入文件 file
32
q //退出 ed 编辑器
# cat file //查看文件 file 的内容
aaaaaaaaaaaaaaaa
bbbbbbbbbbbbbbbb
cccccccccccccccc
#
```



red 是更为严格的 ed。red 只能在当前目录中编辑文件，不支持 Shell 命令。

14.12 fmt 命令：编排文本文件

fmt 命令用于重排指定文件的每一个段落，并输出到标准输出设备。如果不指定文件或指定为“-”，将从标准输入设备读入。该命令的语法格式如下所示。



```
fmt      [-c, --crown-margin][ -s, --split-only][ -t, --tagged-paragraph][ -u,
--uniform-spacing][ -p, --prefix=STRING]
        [ -w, --width=NUMBER][ --help][ --version][FILE]
```

其中各选项的具体说明如表 14-14 所示。

表 14-14 mt 命令选项说明

选 项	说 明
-c, --crown-margin	每段的前两行缩进
-s, --split-only	拆开过长行
-t, --tagged-paragraph	对第一行缩进，第一行和第二行的缩排格式不同
-u, --uniform-spacing	句子间空两个空格，字符间空一个空格
-p, --prefix=STRING	仅合并起始字符为指定字符的列
-w, --width=NUMBER	指定每行的最大行宽
FILE	指定编排的文件
--help	显示帮助信息
--version	显示版本信息

例如，通过 `fmt` 命令重排文件 `abc.doc`，如下所示。

```
# cat abc.doc                //显示文件 abc 的内容
a
ab
abc
abcd
abcde
abcdef
abcdefg
abcdefgh
abcdefghi
abcdefghij
abcdefghijk
abcdefghijkl
# fmt abc.doc                //重排文件 abc
a ab abc abcd abcde abcdef abcdefg abcdefgh abcdefghi abcdefghij
abcdefghijkl abcdefghijkl abcdefghijklm
#
```

14.13 gedit 命令：gnome 的文本编辑器

`gedit` 是一款图形化的文本编辑器，使用方便快捷。在开始菜单中选择【附件】，然后选择【文本编辑器】可以启动 `gedit`，也可以通过在命令行下执行 `gedit` 命令来启动 `gedit`。该命令的语法格式如下所示。

```
gedit    [--new-document][ --new-window][ --quit][ --debug][ filename][ --help][ --version]
```

其中各选项的具体说明如表 14-15 所示。

例如，在命令行下直接输入 `gedit` 命令，进行文本的编辑操作，如下所示。

```
# gedit abc                //编辑文件
```

执行该命令后，将启动 `gedit`，进入图形化文本编辑窗口，如图 14-2 所示。

表 14-15 gedit 命令选项说明

选 项	说 明
--new-document	创建一个新的文档
--new-window	创建一个新的顶级窗口
--quit	退出 gedit
--debug	使用 debug 模式
filename	指定打开的文件名
--help	显示帮助信息
--version	显示版本信息

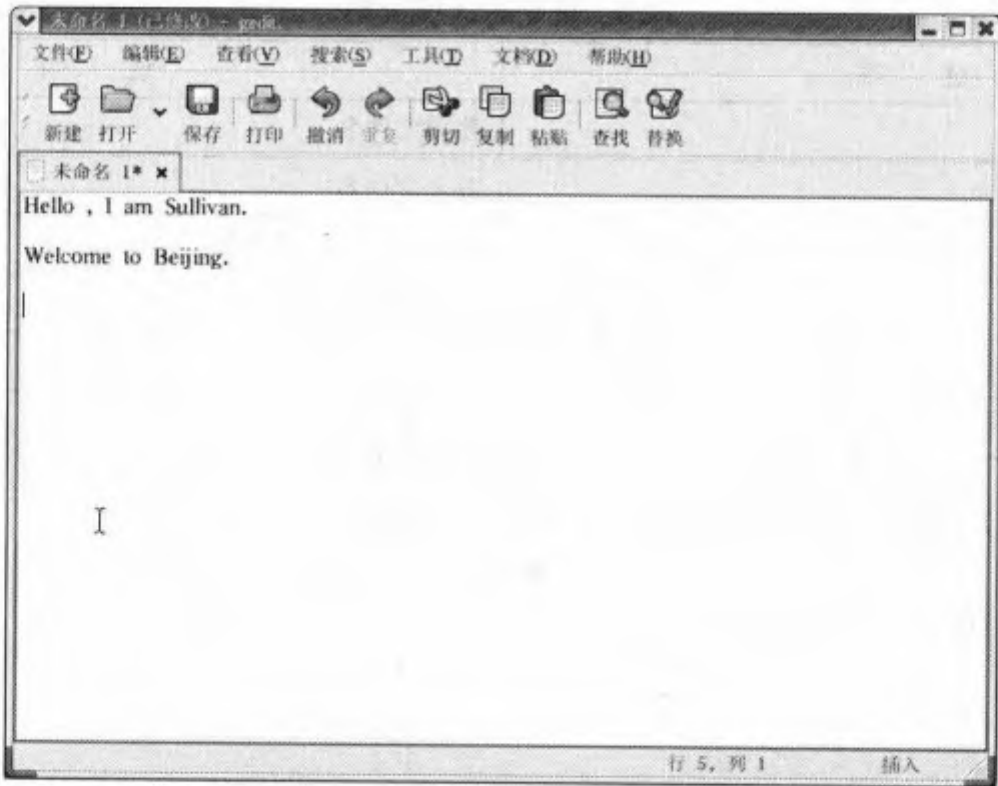


图 14-2 gedit 主窗口

窗口最上端显示当前编辑文件的文件名，默认为“未命令 1”。此界面中还包括菜单栏、工具栏和编辑区。对于同时打开多个文件的情况，gedit 使用标签形式显示文件内容，点击相应的文件名标签，选择需要编辑的文件。若单击“X”则关闭该文件。

在 gedit 中，除了具有文件编辑功能外，还具有打印设置、搜索、拼写检查、文档统计等功能，在相应的菜单中进行选择即可。在【编辑】菜单中，选择【首选项】，能够对 gedit 进行全局设置，如图 14-3 所示。gedit 首选项设置包括编辑器、字体颜色、打印设置、插件设置等。设置完毕后，单击【关闭】按钮保存设置。

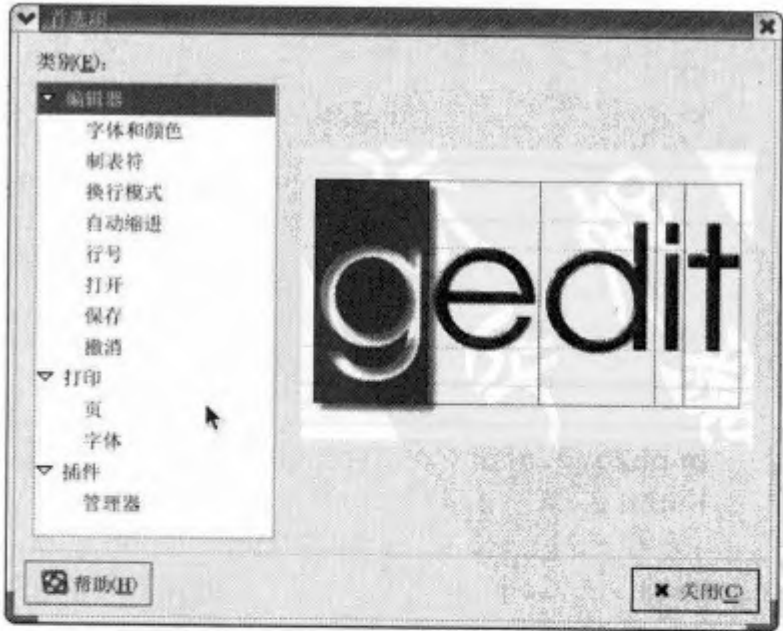


图 14-3 gedit 首选项



14.14 head 命令：从开始显示指定行

cat 命令一次会输出文件的全部内容，而 head 命令则用于查看文件前面的部分内容。head 命令默认将显示输出指定文件的前 10 行到标准输出。如果不指定文件，将从标准输入设备中读入。head 命令的语法格式如下所示。

```
head [-q, --quiet, --silent][ -v, --verbos][ -c, --bytes=SIZE][ -n, --lines=NUMBER][ --help][ --version]
```

其中各选项的具体说明如表 14-16 所示。

表 14-16 head 命令的选项说明

选 项	说 明
-q, --quiet, --silent	隐藏文件名
-v, --verbos	显示文件名
-c, --bytes=SIZE	仅显示指定的字节数
-n, --lines=NUMBER	显示指定的行数
--help	显示帮助信息
--version	显示版本信息

例如，显示指定文件 list 的前 10 行，命令如下所示。

```
# cat list                                     //显示文件 list 的内容
anaconda-ks.cfg
bubble
bubble.bb
bubble.bbg
bubble.c
bubble.c.gcov
bubble.da
--help
install.log
install.log.syslog
kk
kk2
kk3

#head list                                     //显示文件的前 10 行
anaconda-ks.cfg
bubble
bubble.bb
bubble.bbg
bubble.c
bubble.c.gcov
bubble.da
--help
install.log
install.log.syslog
```

例如，显示文件的前 5 行，命令如下所示。


```
#head -5 list //显示文件的前 5 行
anaconda-ks.cfg
bubble
bubble.bb
bubble.bbg
bubble.c
```

例如, 显示文件 mylist 的前 30 个字节, 命令如下所示。

```
# cat mylist //显示文件 mylist 的内容
*#
anaconda-ks.cfg
b
bubble
bubble.bb
bubble.bbg
bubble.c
bubble.c.gcov
bubble.da
--help
install.log
install.log.syslog
# head -c 30 mylist //显示文件的前 30 个字节
*#
anaconda-ks.cfg
b
bubble
```

14.15 indent 命令: 调整 c 原始代码文件的格式

indent 命令通过插入或删除空格, 可以调整 C 代码的格式, 使得代码更容易读懂。indent 命令还可以将一种类型的源代码转化到另一种类型。该命令的语法格式如下所示。

```
indent [-bad, --blank-lines-after-declarations] [-bap, --blank-lines-after-procedures]
[-bbb, --blank-lines-before-block-comments] [-bbo, --break-before-boolean-
operator]
[-bc, --blank-lines-after-commas] [-bl, --braces-after-if-line] [-blin,
--brace-indentn]
[-bls, --braces-after-struct-decl-line] [-brs, --braces-on-struct-decl-
line] [-br, --braces-on-if-line]
[-cdw, --cuddle-do-while] [-ce, --cuddle-else] [-dn, --line-comments-
indentationn]
[-bfda, --break-function-decl-args] [-bfde, --break-function-decl-args]
[-din, --declaration-indentationn]
[-fcl, --format-first-column-comments] [-fca, --format-all-comments] [-gnu,
--gnu-style]
[-hnl, --honour-newlines] [-in, --indent-leveln] [-ipn, --parameter-
indentationn] [-kr, --k-and-r-style]
[-ln, --line-lengthn] [-lcn, --comment-line-lengthn] [-lp, --continue-at-
parentheses]
[-lps, --leave-preprocessor-space] [-nbad, --no-blank-lines-after-declarations]
[-nbap, --no-blank-lines-after-procedures] [-nbbo, --break-after-boolean-
operator]
```



```
[ -nbc, --no-blank-lines-after-commas][ -nbfa, --dont-break-function-decl-args]
[ -nbp, --no-blank-lines-after-procedures][ -nbl, --dont-break-line-up-parentheses]
[ -cn, --comment-indentationn]
[ -npcs, --no-space-after-function-call-names][ -nprs, --no-space-after-parentheses]
[ -npsl, --dont-break-procedure-type][ -nsaf, --no-space-after-fo][ -nsai, --no-space-after-if]
[ -nsaw, --no-space-after-while][ -nsc, --dont-star-comments][ -nsob, --leave-optional-blank-lines]
[ -nss, --dont-space-special-semicolon][ -nut, --no-tabs][ -nv, --no-verbosity][ -orig, --original]
[ -npro, --ignore-profile][ -pcs, --space-after-procedure-calls][ -pin, --paren-indentationn]
[ -pmt, --preserve-mtime][ -prs, --space-after-parentheses][ -psl, --procnames- start-lines]
[ -saf, --space-after-for][ -sai, --space-after-if][ -saw, --space-after-while]
[ -sbin, --struct-brace-indentationn][ -ncdb, --no-comment-delimiters-on-blank-lines]
[ -sc, --start-left-side-of-comments][ -sob, --swallow-optional-blank-lines]
[ -ss, --space-special-semicolon][ -nfcl, --dont-format-first-column-comments]
[ -st, --standard-output][ -tsn, --tab-sizen][ -ut, --use-tabs][ -v, --verbose][ -version]
[ -ncdw, --dont-cuddle-do-while][ -nce, --dont-cuddle-else][ -cbin, --case-brace-indentationn]
[ -nfca, --dont-format-comments][ -nhnl, --ignore-newlines][ -bs, --Bill-Shannon, --blank-before-sizeof
```

其中各选项的具体说明如表 14-17 所示。

表 14-17 indent 命令的选项说明

选 项	说 明
-bad, --blank-lines-after-declarations	声明后强制添加空白行
-bap, --blank-lines-after-procedures	在程序后加空白行
-bbb, --blank-lines-before-block-comments	在块注释之前添加空白行
-bbo, --break-before-boolean-operato	在布尔操作之前的长行进行分行处理
-bc, --blank-lines-after-commas	在声明的逗号之后换行
-bl, --braces-after-if-line	在 if 之后加括号
-blin, --brace-indentn	括号缩进 n 个空格
-bls, --braces-after-struct-decl-line	在结构声明之后加括号
-brs, --braces-on-struct-decl-line	在结构声明行加括号
-br, --braces-on-if-line	在 if 行加括号
-bs, --Bill-Shannon, --blank-before-sizeof	sizeof 后空一格
-cn, --comment-indentationn	在代码右侧第 n 列设置注释
-cbin, --case-brace-indentationn	在空白行设置注释分隔符
-cdw, --cuddle-do-while	使用 “}” 关闭 while 循环
-ce, --cuddle-else	使用 “}” 关闭 else 循环

续表

选 项	说 明
-dn, --line-comments-indentationn	设置注释缩进不产生在代码右侧 n 列
-bfda, --break-function-decl-args	声明时在所有参数之前分行
-bfde, --break-function-decl-args	在声明的最后一个参数之后分行
-din, --declaration-indentationn	在第 n 列中设置变量.
-fcl, --format-first-column-comments	在第一列格式化注释
-fca, --format-all-comments	注释格式化
-gnu, --gnu-style	使用 GNU 代码风格, 默认值
-hnl, --honour-newlines	在输入的新行前, 将长行进行分行处理
-in, --indent-leveln	设置缩进为 n 个空折
-ipn, --parameter-indentationn	在旧函数定义中的缩进参数类型
-kr, --k-and-r-style	使用 Kernighan & Ritchie 代码风格
-ln, --line-lengthn	设置非注释的最大行长度
-lcn, --comment-line-lengthn	设置注释的最大行长度
-lp, --continue-at-parentheses	将括号行连接起来
-lps, --leave-preprocessor-space	在 “#” 和预处理指令之间留有空格
-nbad, --no-blank-lines-after-declarations	在声明之后不强制空白行
-nbap, --no-blank-lines-after-procedures	在过程体之后不强制空行
-nbbo, --break-after-boolean-operator	在布尔操作之前, 不对长行进行分行处理
-nbc, --no-blank-lines-after-commas	在声明中不在逗号之后强制空行
-nbfda, --dont-break-function-decl-args	函数声明中不允许每一参数占用一行
-ncdb, --no-comment-delimiters-on-blank-lines	不在空白行中放置注释分隔符
-ncdw, --dont-cuddle-do-while	不在 while 循环中闭合大括号
-nce, --dont-cuddle-else	不在 else 语句中闭合大括号
-nfcl, --dont-format-first-column-comments	不在第一列格式化注释
-nfca, --dont-format-comments	不格式化任何注释
-nhnl, --ignore-newlines	在输入的新行处不将长行进行分行处理
-nip, --no-parameter-indentation	参数不进行缩进
-nlp, --dont-line-up-parentheses	不对括号行进行连接
-npcs, --no-space-after-function-call-names	不在函数调用之后放置空格
-nprs, --no-space-after-parentheses	不在 “(” 之后和 “)” 之前加入空格
-npsl, --dont-break-procedure-type	将过程类型名称放同一行
-nsaf, --no-space-after-fo	不在每一个 for 语句之后放置空格
-nsai, --no-space-after-if	不在第一个 if 语句之后放置空格
-nsaw, --no-space-after-while	不在每一个 while 循环之后放置空格



续表

选 项	说 明
-nsc, --dont-star-comments	不在注释的左侧放置 “*”
-nsob, --leave-optional-blank-lines	不去除选项空白行
-nss, --dont-space-special-semicolon	不在分号前强制空格
-nut, --no-tabs	使用空格而不是 Tab
-nv, --no-verbosity	关闭冗余模式
-orig, --original	使用原始的 Berkeley 代码风格
-npro, --ignore-profile	不读入 indent.pro 文件
-pcs, --space-after-procedure-calls	在被调用的过程名与 “(” 之间插入空格
-pin, --paren-indentationn	当一个表达式被分割时，对 “(” 指定额外的缩进
-pmt, --preserve-mtime	在输出文件中保留访问和修改时间
-prs, --space-after-parentheses	在每一个 “(” 之后和 “)” 之前放置空格
-psl, --procnames-start-lines	在名称之前放置过程类型
-saf, --space-after-for	在每一个 for 语句之后加空格
-sai, --space-after-if	在每个 if 语句之后加空格
-saw, --space-after-while	在每一个 while 循环之后加空格
-sbin, --struct-brace-indentationn	对结构、联合体或枚举类型的括号进行缩进
-sc, --start-left-side-of-comments	在左侧注释放置 “*” 符号
-sob, --swallow-optional-blank-lines	去掉可选的空白行
-ss, --space-special-semicolon	对单行的 for 语句和 while 语句，在分号之前强制空行
-st, --standard-output	写到标准输出
-tsn, --tab-sizen	设置 Tab 跳过的空格
-ut, --use-tabs	使用 Tab，该项默认
-v, --verbose	冗余模式
-version	显示帮助信息

例如，以 GNU 的格式处理 C 文件 myproc.c，命令如下所。

```
# cat myproc.c //显示文件 myproc.c 的内容
#include <stdio.h>
int main ()
{
    FILE *fp;
    int a, b;
    fp = fopen ("abc.doc", "rb");
    a = ftell (fp);
    printf ("#####d\n", a);
    fseek (fp, 10, 0);
    b = ftell (fp);
    printf ("#####d\n", b);
    fclose (fp);}
# indent myproc.c //以 GNU 的格式来处理 C 文件
# cat myproc.c
#include <stdio.h>
```



```

int
main ()
{
    FILE *fp;
    int a, b;
    fp = fopen ("abc.doc", "rb");
    a = ftell (fp);
    printf ("#####d\n", a);
    fseek (fp, 10, 0);
    b = ftell (fp);
    printf ("#####d\n", b);
    fclose (fp);
}

```

例如，以 Berkeley 的格式来处理 C 文件 myproc.c，命令如下所示

```

# cat myproc.c                                     //显示文件 myproc.c 文件的内容
#include <stdio.h>
int main ()
{
    FILE *fp;
    int a, b;
    fp = fopen ("abc.doc", "rb");
    a = ftell (fp);
    printf ("#####d\n", a);
    fseek (fp, 10, 0);
    b = ftell (fp);
    printf ("#####d\n", b);
    fclose (fp);}
# indent -st -orig -v myproc.c                     //以 Berkeley 的格式处理 C 文件;myproc.c
#include <stdio.h>
int
main()
{
    FILE          *fp;
indent: file.c:5: Warning:Line broken 2
    int          a,
                b;

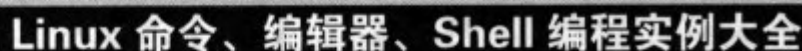
    fp = fopen("abc.doc", "rb");
    a = ftell(fp);
    printf("#####d\n", a);
    fseek(fp, 10, 0);
    b = ftell(fp);
    printf("#####d\n", b);
indent: myproc.c:12: Warning:Line broken 2
    fclose(fp);
}
There were 15 non-blank output lines and 0 comments

```

14.16 jed 命令：编辑文本文件

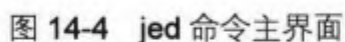
jed 命令用来编辑文本文件，其语法格式如下所示。

```
jed [-2] [-batch] [-n] [-g n] [-i file] [-s string] [-f function]
```

表 14-18 jed 命令选项说明

例如，通过 `jed` 命令进入编辑主界面，如下所示。

执行该命令后，将进入 `jed` 主界面，如图 14-4 所示。此界面上方为菜单栏，中部为编辑区，底部为状态栏。按 `Ctrl+X+C` 组合键可以退出 `jed`。



```
# jed -2 /etc/initlog.conf
```

610

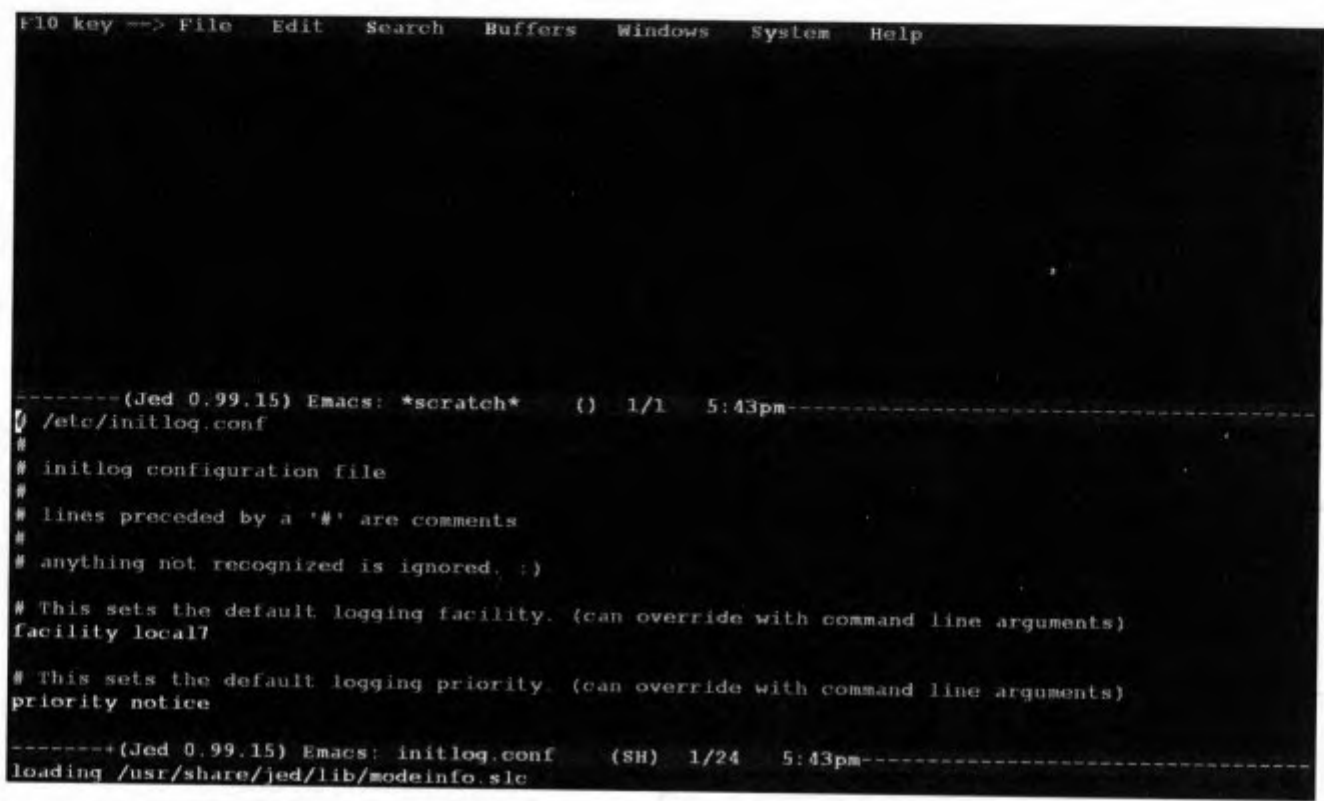


图 14-5 jed 窗口被拆分

14.17 joe 命令：编辑文本文件

joe 命令是一个功能强大的 ASCII 文本编辑器，其操作比较复杂，但是功能非常强大。joe 可以模拟多种工作环境，如表 14-19 所示。

表 14-19 joe 命令模拟的工作环境

选 项	说 明
jstar	模拟 WordStar 环境
jmacs	模拟 Emacs 环境
jpico	模拟 Pico 环境
rmjoe	简约版 joe

该命令的语法格式如下所示。

```
joe [-asis][ -beep][ -csmode][ -force][ -help][ -keepup][ -lightoff][ -marking]
[ -mid][ -nobackups][ -nonotice]
[ -nosta][ -autoindent][ -crlf][ -linum][ -overwrite][ -rdonly][ -backpath
path][ -columns nnn][ -lines nnn]
```

其中各选项的具体说明如表 14-20 所示。

表 14-20 joe 命令选项说明

选 项	说 明
-asis	只处理小于 127 的字符
-beep	出错则发声
-csmode	连续检查
-force	最后一行加换行符



续表

选 项	说 明
-help	帮助菜单
-keepup	状态菜单
-lightoff	不反白显示
-marking	反白显示
-mid	卷页时，光标移到中央
-nobackups	不备份
-nonotice	不显示版本信息
-nosta	不显示状态列
-autoindent	自动缩进
-crlf	兼容性设置，适于编辑 MS-DOS 或 VMS 文件
-linum	显示行号
-overwrite	覆盖模式
-rdonly	只读方式
-backpath path	备份目录
-columns nnn	设置栏数
-lines nnn	设置屏幕的行数

进入命令 joe 后，通过快捷键可以完成相关操作。一些常用的快捷键定义如表 14-21 所示。

表 14-21 常用的 joe 快捷键定义

选 项	说 明
PgUp	上一页
PgDn	下一页
Ctrl+a	该行开始
Ctrl+e	该行结尾
Ctrl+k+u	文件起始处
Ctrl+k+v	文件结尾处
Ctrl+k+h	帮助菜单
Ctrl+k+f	查找字符串
Ctrl+k+c	复制
Ctrl+k+y	删除
Ctrl+k+d	保存
Ctrl+c	关闭文件

例如，通过命令 joe ，编辑文件/etc/initlog.conf，执行命令如下所示。

```
# joe /etc/initlog.conf //编辑文件 abc
Processing '/etc/joe/joerc'...done
Processing '/etc/joe/joerc'...done
```


执行该命令后，进入 joe 命令的主界面，如图 14-6 所示。

```

IW      /etc/initlog.conf (Modified)      Row 1      Col 1      6:04 Ctrl-K H for help
# /etc/initlog.conf
#
# initlog configuration file
#
# lines preceded by a '#' are comments
#
# anything not recognized is ignored. :)
#
# This sets the default logging facility. (can override with command line arguments)
facility local7
#
# This sets the default logging priority. (can override with command line arguments)
priority notice
#
# ignore foo
#
# means to discard any output lines that match the regular expression foo
#
# This regexp is useful if you use fsck's -C option.
ignore [^:]*: \[[=]+
#
# This regexp is useful for fsck's feature of setting file types
ignore [^:]*: Setting filetype
#
# This regexp is useful for quotacheck
ignore ^[-\\W\\N/]$

```

图 14-6 joe 命令的主界面

通过命令“joe +5 /etc/initlog.conf”，打开文件/etc/initlog.conf，并将光标移动到第 5 行，然后进行编辑，执行命令如下所示。

```
# joe +5 /etc/initlog.conf //移到指定行数
Processing '/etc/joe/joerc'...done
Processing '/etc/joe/joerc'...done
```

执行该命令后，进入 joe 命令的主界面，如图 14-7 所示。可以看到，光标定位到文件 /etc/initlog.conf 的第 5 行。

```

IW /etc/initlog.conf Row 5 Col 1 6:19 Ctrl-K H for help
# /etc/initlog.conf
#
# initlog configuration file
#
# lines preceded by a '#' are comments
#
# anything not recognized is ignored. :)
#
# This sets the default logging facility. (can override with command line arguments)
facility local7
#
# This sets the default logging priority. (can override with command line arguments)
priority notice
#
# ignore foo
#
# means to discard any output lines that match the regular expression foo
#
# This regexp is useful if you use fsck's -C option.
ignore [^:]+: \[[-]+\
#
# This regexp is useful for fsck's feature of setting file types
ignore [^:]+: Setting filetype
#
# This regexp is useful for quotacheck
ignore ^[-\\\\\\|/]$

```

图 14-7 定位到文件指定行



在如图 14-7 所示界面使用快捷键 Ctrl+K+H，屏幕的上方将出现帮助信息，以方便进一步操作，如图 14-8 所示。

```
Help Screen      turn off with ^KH      more help with ESC . (^[.)
CURSOR          GO TO          BLOCK          DELETE          MISC          EXIT
^B left ^F right ^U prev. screen ^KB begin ^D char. ^KJ reformat ^KK save
^P up ^N down ^V next screen ^KK end ^Y line ^T options ^C abort
^Z previous word ^A beg. of line ^KM move ^W >word ^R refresh ^KZ shell
^X next word ^E end of line ^KC copy ^O word< ^@ insert FILE
SEARCH          ^KU top of file ^KW file ^J >line SPELL ^KE edit
^KF find text ^KV end of file ^KY delete ^ undo ^[N word ^KR insert
^L find next ^KL to line No. ^K/ filter ^ redo ^[L file ^KD save

IW /etc/initlog.conf Row 5 Col 1 6:51 Ctrl-K H for help
# /etc/initlog.conf
#
# initlog configuration file
#
# lines preceded by a '#' are comments
#
# anything not recognized is ignored. :)
#
# This sets the default logging facility. (can override with command line arguments)
facility local7
#
# This sets the default logging priority. (can override with command line arguments)
priority notice
#
# ignore foo
# means to discard any output lines that match the regular expression foo
#
# This regexp is useful if you use fsck's -C option.
ignore [^:]*: \[[-]*
# This regexp is useful for fsck's feature of setting file types
ignore [^:]*: Setting filetype
# This regexp is useful for quotacheck
ignore ^[-\\|/|S
```

图 14-8 显示帮助快捷信息

例如，通过命令 joe ，以只读方式打开文件/etc/initlog.conf，执行命令如下所示。

```
# joe -ronly /etc/initlog.conf //以只读的方式打开文件
Processing '/etc/joe/joerc'...done
Processing '/etc/joe/joerc'...done
```

执行该命令后进入 joe 命令的主界面，如图 14-9 所示。从上部标题栏可以看到，该文件以只读方式打开（Read only）。

```
IW /etc/initlog.conf (Read only) Row 1 Col 1 6:55 Ctrl-K H for help
# /etc/initlog.conf
#
# initlog configuration file
#
# lines preceded by a '#' are comments
#
# anything not recognized is ignored. :)
#
# This sets the default logging facility. (can override with command line arguments)
facility local7
#
# This sets the default logging priority. (can override with command line arguments)
priority notice
#
# ignore foo
# means to discard any output lines that match the regular expression foo
#
# This regexp is useful if you use fsck's -C option.
ignore [^:]*: \[[-]*
# This regexp is useful for fsck's feature of setting file types
ignore [^:]*: Setting filetype
# This regexp is useful for quotacheck
ignore ^[-\\|/|S

** Joe's Own Editor v2.9.7 ** Copyright (C) 2001 **
```

图 14-9 以只读方式打开文件

14.18 join 命令：基于相同字段连接两个文件

join 命令用于连接两个指定的文件，其语法格式如下所示。

```
join [-i, --ignore-case] [-e] [--help] [--version] [-a 1] [-a 2] [-j] [-j1] [-j2] [-1 FIELD] [-2 FIELD] [-o FORMAT] [-t CHAR] [FILE1 ][FILE2]
```

其中各选项的说明如表 14-22 所示。

表 14-22 join 命令选项说明

选 项	说 明
-i, --ignore-case	不区别大小写
-e	用空白代替丢失的输入字段
--help	显示帮助信息
--version	显示版本信息
-a 1	显示文件 1 中不匹配的行
-a 2	显示文件 2 中不匹配的行
-j	等同于“-1 FIELD -2 FIELD”
-j1	等于“-1 FIELD”
-j2	等于“-2 FIELD”
-1 FIELD	基于文件 1 字段
-2 FIELD	基于文件 2 字段
-t CHAR	指定输入/输出的域分隔符
-o FORMAT	指定格式输出
FILE1	进行比较的第一个文件
FILE2	进行比较的第二个文件

例如，通过命令 join，将文件 123 和文件 abc 相同的部分连接起来，如下所示。

```
# cat 123 //显示文件 123 信息
A 001
B 002
C 003
D 004
E 005
F 006
G 007
H 008
I 009
# cat abc //显示文件 abc 信息
A xy101
B xy102
C xy103
D xy104
E xy105
F xy106
G xy107
H xy108
I xy109
# join 123 abc //按先文件 123 后文件 abc 的顺序连接两个文件
A 001 xy101
```



```
B 002 xy102
C 003 xy103
D 004 xy104
E 005 xy105
F 006 xy106
G 007 xy107
H 008 xy108
I 009 xy109
#
```

如果按先文件 abc 后文件 123 的顺序进行连接，将输出如下结果。

```
# join abc 123 //按先文件 abc 后文件 123 的顺序连接两个文件
A xy101 001
B xy102 002
C xy103 003
D xy104 004
E xy105 005
F xy106 006
G xy107 007
H xy108 008
I xy109 009
#
```



同样对于这两个文件，如果文件 1 和文件 2 的连接顺序不同，输出的结果也可能不同。

14.19 less 命令：前后翻阅文本内容

less 命令和 more 命令功能相似，显示文件时允许用户既可以向前也可以向后翻阅文件。向前翻阅按 pageup 键，向后翻阅按 pagedown 键，退出按 q 键。less 命令的语法格式如下所示。

```
less [-c][ -f][ -i][ -I][ -i][ -m][ -M][ -N][ -p pattern][ -S][ -Q]
```

less 命令的选项说明见表 14-23。

表 14-23 less 命令选项说明

选 项	说 明
-c	从顶部（从上到下）刷新屏幕，并显示文件内容，而不是通过底部滚动完成刷新
-f	强制打开文件，如果是二进制文件也不提示警告
-i	搜索时忽略大小写，但搜索串中包含大写字母除外
-I	搜索时忽略大小写，但搜索串中包含小写字母除外
-m	显示读取文件的百分比
-M	显示读取文件的百分比、行号及总行数
-N	在每行前输出行号
-p pattern	例如，在/etc/ftpuser 中搜索单词 student，可以使用“less -p student /etc/ftpuser”
-S	把连续多个空白行作为一个空白行显示
-Q	在终端下不响铃

在用 less 命令查看文件时，使用一些常用的操作命令可以加快查找、定位的速度。常用的操作命令见表 14-24。

例如，查看当前目录下 test 文件的内容，命令如下所示。

```
#less test
```


如果在显示文件 `example1` 内容的同时，加上行号，命令如下所示。

```
#less -N example1
1 aa bb cc dd
2 bb cc dd ee
3 cc dd ee ff
4 dd ee ff gg
5 ee ff gg hh
6 ff gg hh ii
```

表 14-24 less 常用的操作命令

操 作 命 令	说 明
回车键	向下移动一行
y	向上移动一行
空格键	向下滚动一屏
b	向上滚动一屏
d	向下滚动半屏
h	less 的帮助
u	向上滚动半屏
w	从指定行数的下一行显示，如指定的值是 9，则从第 10 行开始显示
g	跳到第一行
G	跳到最后一行
p n%	跳到 n%，如 50%表示从整个文档的 50%处开始显示
/pattern	搜索 pattern，如/ftpuser 表示从文件中搜索单词 ftpuser
v	调用 vi 编辑器
q	退出 less
!command	调用 Shell 命令，如使用 “! ls” 表示列出当前目录下的所有文件

14.20 more 命令：分屏显示文本内容

`more` 命令用于一次显示一屏。若信息未能完全显示，屏幕底部出现 “-More-(xx%)”，按 `Space` 键，显示下一屏内容；按回车键，显示下一行内容；按 `B` 键，显示上一屏；按 `Q` 键，退出 `more` 命令。`more` 命令的语法格式如下所示。

```
more [+n][-n][ +/pattern][ -c][ -d][ -l][ -p][ -s][ -u]
```

其中各选项的具体说明如表 14-25 所示。

表 14-25 more 命令选项说明

选 项	说 明
+n	从第 <i>n</i> 行开始显示
-n	定义屏幕大小为 <i>n</i> 行
+/pattern	从 <i>pattern</i> 前两行开始显示。
-c	从顶部清屏，然后显示
-d	提示 “Press space to continue, 'q' to quit（按空格键继续，按 <code>q</code> 键退出）”，禁用响铃功能



续表

选 项	说 明
-l	忽略 Ctrl+l（换页）字符
-p	通过清除窗口而不是滚屏来对文件进行换页，与-c 选项相似
-s	把连续的多个空行显示为一行
-u	把文件内容中的下划线去掉

在查看一个内容较多而无法在一屏内显示的文件时，经常要用到 `more` 的操作命令，常用的操作命令及说明见表 14-26。

表 14-26 常用的 `more` 操作命令及说明

操 作 命 令	说 明
Enter	向下 n 行，需要定义。默认为 1 行
Ctrl+F	向下滚动一屏
空格键	向下滚动一屏
Ctrl+B	返回上一屏
=	输出当前行的行号
: f	输出文件名和当前行的行号
V	调用 vi 编辑器
!命令	调用 Shell，并执行命令
q	退出 more

例如，显示文件 `test` 中从第 3 行起的内容，命令如下所示。

```
#more +3 test
```

例如，使用 “+/pattern” 选项，从文件 `test` 中查找第一个出现 “teacher” 字符串的行，并从该处前两行开始显示输出，命令如下所示。

```
#more +/teacher test
```

例如，显示文件 `test`，且每屏显示 8 行，命令如下所示。

```
#more -8 test
```

例如，从终端顶部开始显示文件内容，并给出提示信息，命令如下所示。

```
#more -dc test
```

14.21 od 命令：输出文件内容

`od` 命令用于按八进制或其他格式显示输出指定的文件，其语法格式如下所示。

```
od [-A, --address-radix=RADIX] [-j, --skip-bytes=BYTES] [-N, --read-bytes=BYTES] [-s, --strings[=BYTES]]  
  [-t, --format=TYPE] [-v, --output-duplicates] [-w, --width[=BYTES]] [--traditional]  
  [--help] [--version]
```

其中各选项的具体说明如表 14-27 所示。

例如，八进制的形式显示文件 `mypaper` 的内容，命令如下所示。

表 14-27

od 命令的选项说明

选 项	说 明
-A, --address-radix=RADIX	指定文件偏移的输出方式
-j, --skip-bytes=BYTES	跳过输入的起始指定字节数
-N, --read-bytes=BYTES	根据指定的输入字节限制输出
-s, --strings[=BYTES]	输出的字符串至少包含指定字节的图形字符
-t, --format=TYPE	指定输出的格式
-v, --output-duplicates	不使用*标识行压缩
-w, --width[=BYTES]	每行输出的字节数
--traditional	以传统方式接受参数
--help	显示此帮助信息并离开
--version	显示版本信息并离开

```
# cat mypaper //显示文件内容
hello
This is a test paper.
I liko Linux.
It im a very perfecr operating system.
we are Linux lovers.
# od abc //以八进制的形式显示文件内容
# od mypaper
0000000 062550 066154 005157 064124 071551 064440 020163 020141
0000020 062564 072163 070040 070141 071145 005056 020111 064554
0000040 067553 046040 067151 074165 005056 072111 064440 020155
0000060 020141 062566 074562 070040 071145 062546 071143 067440
0000100 062560 060562 064564 063556 071440 071571 062564 027155
0000120 073412 020145 071141 020145 064514 072556 020170 067554
0000140 062566 071562 005056
0000146
```

例如，以字符形式显示文件 mypaper 的内容，命令如下所示。

```
# od -t c mypaper //以字符方式显示文件内容
0000000 h e l l o \n T h i s i s a
0000020 t e s t p a p e r . \n I l i
0000040 k o L i n u x . \n I t i m
0000060 a v e r y p e r f e c r o
0000100 p e r a t i n g s y s t e m .
0000120 \n w e a r e L i n u x l o
0000140 v e r s . \n
0000146
```

例如，以 ASCII 码的形式显示文件 mypaper 的内容，命令如下所示。

```
#od -t a mypaper //以 ASCII 的形式显示文件内容
0000000 h e l l o nl T h i s sp i s sp a sp
0000020 t e s t sp p a p e r . nl I sp l i
0000040 k o sp L i n u x . nl I t sp i m sp
0000060 a sp v e r y sp p e r f e c r sp o
0000100 p e r a t i n g sp s y s t e m .
0000120 nl w e sp a r e sp L i n u x sp l o
0000140 v e r s . nl
0000146
```

例如，以十六进制的形式显示文件 mypaper 的内容，命令如下所示。



```
# od -t x mypaper //以十六进制的形式显示文件内容
0000000 6c6c6568 68540a6f 69207369 20612073
0000020 74736574 70617020 0a2e7265 696c2049
0000040 4c206f6b 78756e69 74490a2e 206d6920
0000060 65762061 70207972 65667265 6f207263
0000100 61726570 676e6974 73797320 2e6d6574
0000120 2065770a 20657261 756e694c 6f6c2078
0000140 73726576 00000a2e
```

例如，以十进制的形式显示文件 mypaper 的内容，命令如下所示。

```
# od -t d mypaper //以十进制的形式显示文件内容
0000000 1819043176 1750338159 1763734377 543236211
0000020 1953719668 1885433888 170816101 1768693833
0000040 1277194091 2020961897 1950943790 544041248
0000060 1702240353 1881176434 1701212773 1864397411
0000100 1634887024 1735289204 1937339168 778921332
0000120 543520522 543519329 1970170188 1869357176
0000140 1936876918 2606
0000146
```

例如，以浮点数的形式显示文件 mypaper 的内容，命令如下所示。

```
# od -t f mypaper //以浮点的形式显示文件内容
0000000 3.657391587857248e+194 1.021897754876849e-152
0000020 2.165824246080534e+233 6.727859028816127e+199
0000040 1.811525684238524e+272 1.754837277864890e-152
0000060 1.278840342679751e+232 1.948089587261170e+227
0000100 1.693757741359845e+190 4.728724660372878e-85
120 1.279660009783787e-152 5.330496210725272e+228
0000140 5.530877975600906e-311
0000146
```

14.22 pr 命令：编排文件格式

pr 命令用于编排指定的文件，然后将结果输出到标准输出，其语法格式如下所示。

```
pr [-COLUMN, --columns=COLUMN] [-c, --show-control-chars] [-d, --double-space]
[ --help] [ --version]
[ -D, --date-format=FORMAT] [-e[CHAR[WIDTH]], --expand-tabs[=CHAR[WIDTH]]]
[ -h HEADER, --header=HEADER] [-i[CHAR[WIDTH]], --output-tabs[=CHAR[WIDTH]]]
[ -l PAGE_LENGTH, --length=PAGE_LENGTH] [-m, --merge]
[ -n[SEP[DIGITS]], --number-lines[=SEP[DIGITS]]] [-N NUMBER, --first-line-number= NUMBER]
[ -o MARGIN, --indent=MARGIN] [-r, --no-file-warnings] [-s[CHAR], --separator[=CHAR]]
[ -SSTRING, --sep-string[=STRING]] [-t, --omit-header] [-v, --show-nonprinting]
[ -w PAGE_WIDTH, --width=PAGE_WIDTH] [-W PAGE_WIDTH, --page-width=PAGE_WIDTH]
```

其中各选项的具体说明如表 14-28 所示。

表 14-28 pr 命令的选项说明

选 项	说 明
-COLUMN, --columns=COLUMN	按指定分栏数输出
-c, --show-control-chars	使用控制字符
-d, --double-space	双倍空格输出

续表

选 项	说 明
-D, --date-format=FORMAT	使用指定的格式作为标题日期
-e[CHAR[WIDTH]], --expand-tabs[=CHAR[WIDTH]]	扩展输入字符为 Tab 宽度
-h HEADER, --header=HEADER	使用指定的题标, 替换文件名
-i[CHAR[WIDTH]], --output-tabs[=CHAR[WIDTH]]	将空格替换为 Tab
-l PAGE_LENGTH, --length=PAGE_LENGTH	设置页面长度
-m, --merge	并行输出所有文件, 一个文件占一行
-n[SEP[DIGITS]], --number-lines[=SEP[DIGITS]]	对行编号
-N NUMBER, --first-line-number=NUMBER	指定第一页第一行的起始编号
-o MARGIN, --indent=MARGIN	每一行偏移指定的页边距
-s[CHAR],--separator[=CHAR]	指定每一列的分隔符, 默认为 Tab
-SSTRING, --sep-string[=STRING]	将指定的字符串作为列分隔符
-t, --omit-header	忽略页眉、页脚
-v, --show-nonprinting	使用八进制反斜杠符号
-w PAGE_WIDTH, --width=PAGE_WIDTH	对多列文本设置页面宽度
-W PAGE_WIDTH, --page-width=PAGE_WIDTH	设置页面宽度, 对长行进行分行处理
--help	显示帮助信息
--version	显示版本信息

例如，通过命令 `pr`，编排文件 123，并将结果重定向到文件 456；通过命令 `cat`，显示文件 456 的内容，命令执行过程如下所示。

```
# cat 123                                //显示文件 123 的内容
000000      00000000
111111      11111111
222222      22222222
333333      33333333
444444      44444444
555555      55555555

# pr 123 > 456                            //编排文件 123
# cat 456                                  //显示文件 456 的内容
2008-08-08 23:04                          123                      Page 1
000000      00000000
111111      11111111
222222      22222222
333333      33333333
444444      44444444
555555      55555555
```

例如，过命令 `pr`，编排文件 123，分 3 栏显示文件 123 的内容，并将结果重定向到文件 456，通过命令 `cat`，显示文件 456 的内容，命令执行过程如下所示。

```
# cat 123                                //显示文件内容
000000      00000000
111111      11111111
222222      22222222
```



```
333333      33333333
444444      44444444
555555      55555555
666666      66666666
777777      77777777
888888      88888888
999999      99999999
# pr -3 123 > 456           //进行 3 分栏
# cat 456                   //显示文件内容
2008-08-08 23:13           123           Page 1
000000      00000000  444444      44444444  777777      77777777
111111      11111111  555555      55555555  888888      88888888
222222      22222222  666666      66666666  999999      99999999
333333      33333333
```

14.23 patch 命令：修补文件

patch 命令用来为文件打补丁，可以一次打多个补丁，通常可以用来为系统升级。该命令的语法格式如下所示。

```
patch [-b,--backup][--backup-if-mismatch][--no-backup-if-mismatch][--binary,--binary]
[-v, --version][--help]
[-d dir, --directory=dir][--dry-run][ -E, --remove-empty-files][ -e, --ed][ -f,
--force] [ -N, --forward]
[ -F num, --fuzz=num][ -l, --ignore-whitespace][ -n, --normal] [ -l, --ignore-
hitespace] [ --verbose]
[ -o outfile, --output=outfile][ --quoting-style=word][ -s, --silent, --quiet]
```

其中各选项的具体说明如表 14-29 所示。

表 14-29 patch 命令的选项说明

选 项	说 明
-b, --backup	备份
--backup-if-mismatch	如果补丁不匹配文件则进行备份
--no-backup-if-mismatch	补丁不匹配文件，也不进行备份
--binary --binary	以二进制模式读写文件，除了标准输出设备和/dev/tty 之外
-d dir, --directory=dir	指定目录
--dry-run	显示应用补丁的结果，但并不实际修改任何文件
-E, --remove-empty-files	在应用补丁之后，删除空的输出文件
-e, --ed	按 ed 脚本解析补丁文件
-f, --force	强制执行，不再进行询问
-F num, --fuzz=num	设置最大模糊因子
-l, --ignore-whitespace	忽略空格和 Tab
-n, --normal	将补丁文件视为常规的差异文件
-N, --forward	忽略已经应用的补丁
-o outfile, --output=outfile	将需要输出的内容输出到指定的输出文件，而不是输出到补丁文件

续表

选 项	说 明
--quoting-style=word	使用指定的风格单词来引用输出名称
-s, --silent, --quiet	静默模式，不显示处理信息
--verbose	运行时，显示详细的处理信息
-l, --ignore-whitespace	松散匹配模式
-v, --version	显示版本信息
-pnum or --strip=num	去除指定数量的斜杠前缀，连续的斜杠将被识别为一个斜杠
--help	显示帮助信息

例如，通过命令 diff，比较文件 mybook 和文件 mylist 的不同，并生成差异文件 paper.diff，此差异文件可以作为补丁，执行过程如下所示。

```
# cat mybook                                     //显示文件 mybook 的内容
Hello , I am a student.
I am from English.
Today is 2008-08-08.
I am at ERPO.
Welcome to China.
# cat mylist                                     //显示文件 mylist 的内容
Hello , I am a student.
I am from China.
Today is 2008-08-08.
I am at DESHENG MENG.
Welcome to China.
# diff -ruN mybook mylist >paper.diff             //比较文件 mylist 和 mybook，生成差异文件
# cat paper.diff                                  //显示文件差异文件的内容
--- mybook      2008-08-08 17:09:59.000000000 +0800
+++ mylist      2008-08-08 17:08:15.000000000 +0800
@@ -1,5 +1,5 @@
  Hello , I am a student.
-I am from English.
+I am from China.
  Today is 2008-08-08.
-I am at ERPO.
+I am at DESHENG MENG.
  Welcome to China.
```

通过命令 patch，进行打补丁，此处选择的被打补丁的文件是 mybook。打完补丁之后，文件 mybook 的内容和文件 mylist 的内容相同。执行过程如下所示。

```
# cat mybook                                     //显示打补丁之前的文件 mybook 的内容
Hello , I am a student.
I am from English.
Today is 2008-08-08.
I am at ERPO.
Welcome to China.
# patch -p1 <paper.diff                          //打补丁
missing header for unified diff at line 3 of patch
can't find file to patch at input line 3
Perhaps you used the wrong -p or --strip option?
The text leading up to this was:
-----
```



```
|--- mybook      2008-08-08 17:09:59.000000000 +0800
|+++ mylist      2008-08-08 17:08:15.000000000 +0800
-----
File to patch: mybook           //选择要打补丁的文件
patching file mybook
# cat mybook                    //显示打补丁之后的文件 mybook 的内容
Hello , I am a student.
I am from China.
Today is 2008-08-08.
I am at DESHENG MENG.
Welcome to China.
```

例如，通过命令 patch，为系统打补丁，补丁文件为/home/src/patch-2.6.8.如下所示。

```
#patch -p0 </home/src/patch-2.6.8.1 //为系统打补丁
```

14.24 pico 命令：编辑文字文件

pico 命令是一个编辑文本文件的工具，它提供交互式的操作界面，使用灵活方便。该命令的语法格式如下所示。

```
pico [-a][ -b][ -e][ -f][ -h][ -j][ -g][ -k][ -m][ -t][ -s speller][ -w][ -x][ -z][ -v][ +n][ -o dir][ --help][ --version][ file]
```

其中各选项的具体说明如表 14-30 所示。

表 14-30 pico 命令选项说明

选 项	说 明
-a	显示所有文件，包括隐藏文件
-b	文本替换
-d	重新指定删除键
-e	使用文件名完成
-f	使用功能键
-h	列出有效的命令行选项
-j	在文件浏览器中使用“goto”命令
-g	大文件浏览器中使用显示光标模式
-k	“Cut Text”命令将删除从当前光标到行尾的所有字符，而不是整个行
-m	支持鼠标
-nn	允许收到新邮件时进行通知，n 指定检测邮箱的间隔，单位为秒
-t	工具模式
-s speller	指定拼写检查工具
-w	关闭单词自动换行
-x	关闭屏幕底部主菜单
-z	支持快捷键 Ctrl+z 挂起 pico
-v	只读，无法编辑
+n	打开时跳到指定行数
-o dir	指定工作目录

续表

选 项	说 明
--help	帮助信息
--version	版本信息
file	指定打开的文件

进入 `pico` 主界面后，通过快捷键可以完成对文本的基本操作。一些常用的快捷键定义如表 14-31 所示。

表 14-31 `pico` 命令的快捷键定义

快 捷 键	说 明
Ctrl+G	帮助菜单
Ctrl+X	关闭
Ctrl+O	保存
Ctrl+R	插入文件
Ctrl+W	搜索
Ctrl+K	删除行
Ctrl+U	回到上一步
Ctrl+A	该行的最前面
Ctrl+E	该行的最后面
Ctrl+D	删除行
Ctrl+\	替换

例如，通过命令 `pic`，编辑文件 `/etc/initlog.conf`，如下所示。

```
# pico /etc/initlog.conf //编辑文件/etc/initlog.conf
执行该命令后，将出现图 14-10 所示的编辑界面。
```



图 14-10 使用 `pico` 编辑文件



例如，通过命令 `pico` 编辑文件 `/etc/initlog.conf`，且关闭底部的主菜单，如下所示。

```
# pico -x /etc/initlog.conf //编辑文件/etc/initlog.conf
执行该命令后，将出现图 14-11 所示的编辑界面。
```



图 14-11 关闭底部主菜单的编辑窗口

14.25 sort 命令：文件排序

`sort` 命令用于对文件中的所有行进行排序，并将结果显示在屏幕上。`sort` 命令的语法格式如下。

```
sort [-d, --dictionary-order] [-g, --general-numeric-sort] [-b, --ignore-leading-blanks] [-c, --check] [-r, --reverse]
[-d, --dictionary-order] [-f, --ignore-case] [-m, --merge] [-n, --numeric-sort] [-o, --output=FILE]
[-t, --field-separator=SEP] [--help] [--version]
```

其中各选项的具体说明如表 14-32 所示。

表 14-32 sort 命令的选项说明

选 项	说 明
-d, --dictionary-order	按字典顺序排序，可比较的字符仅包含字母、数字、空格和制表符
-g, --general-numeric-sort	按数字进行排序
-b, --ignore-leading-blanks	忽略行前的空格
-c, --check	检查是否已排过序
-d, --dictionary-order	排序时只识别英文数字及空格
-f, --ignore-case	把小写字母看作大写字母
-m, --merge	把已经排过序的文件列表合并成一个文件，并送往标准输出。对合并排序好的文件，不再进行排序

续表

选 项	说 明
-n, --numeric-sort	按照数值排序
-r, --reverse	逆序排列
-o, --output=FILE	将结果输出到指定文件，而不是标准输出设备
-t, --field-separator=SEP	指定分隔符
--help	显示帮助信息
--version	显示版本信息
+	指定排序的起始比较位置

例如，对文件 student 按字典进行排序，命令如下所示。

```
$ cat student //显示文件 student 的内容
Tom      A      B      C      D
Mike     B      C      D      E
Mary     C      D      E      F
Jean     D      E      F      G

$ sort student //对 student 排序
Jean     D      E      F      G
Mary     C      D      E      F
Mike     B      C      D      E
Tom      A      B      C      D
```

sort 命令常和管道 (|) 联合使用，例如，对当前目录的文件按字典顺序进行排列并显示命令如下所示。

```
# ls
anaconda-ks.cfg Desktop install.log man_chage
chage_li file1.doc install.log.syslog
# ls | sort -d //按字典顺序排列
anaconda-ks.cfg
chage_li
Desktop
file1.doc
install.log
install.log.syslog
man_chage
# ls | sort -r //按逆序排列文件
man_chage
install.log.syslog
install.log
file1.doc
Desktop
chage_li
anaconda-ks.cfg
```

通过命令 sort，对文件进行排序，默认的情况是依据第 1 个字段进行排序。例如，通过命令 “sort +6 temp”，从文件 temp 的第 7 个字段开始进行比较排序，然后将结果输出到标准输出设备，如下所示。

```
# cat temp //显示文件内容
1 I am a linux learner 3
3 I am a linux learner 7
5 I am a linux learner 8
8 I am a linux learner 9
```



```
2 I am a linux learner 2
4 I am a linux learner 6
9 I am a linux learner 1
7 I am a linux learner 4
6 I am a linux learner 0
0 I am a linux learner 5
# sort temp //文件排序
0 I am a linux learner 5
1 I am a linux learner 3
2 I am a linux learner 2
3 I am a linux learner 7
4 I am a linux learner 6
5 I am a linux learner 8
6 I am a linux learner 0
7 I am a linux learner 4
8 I am a linux learner 9
9 I am a linux learner 1
# sort +6 temp //按照第 7 字段对文件进行排序
6 I am a linux learner 0
9 I am a linux learner 1
2 I am a linux learner 2
1 I am a linux learner 3
7 I am a linux learner 4
0 I am a linux learner 5
4 I am a linux learner 6
3 I am a linux learner 7
5 I am a linux learner 8
8 I am a linux learner 9
```

例如，对多个文件 temp1、temp2、temp3 进行排序，并将结果输出到标准输出，命令如下所示。

```
# cat temp1 //显示文件内容
3 I am a linux learner 7
5 I am a linux learner 8
4 I am a linux learner 6
6 I am a linux learner 0
0 I am a linux learner 5
# cat temp2 //显示文件内容
1 I am a linux learner 3
8 I am a linux learner 9
7 I am a linux learner 4
cat temp3 //显示文件内容
2 I am a linux learner 2
9 I am a linux learner 1
# sort temp1 temp2 temp3 //多个文件排序
0 I am a linux learner 5
1 I am a linux learner 3
2 I am a linux learner 2
3 I am a linux learner 7
4 I am a linux learner 6
5 I am a linux learner 8
6 I am a linux learner 0
7 I am a linux learner 4
8 I am a linux learner 9
9 I am a linux learner 1
```


14.26 spell 命令：简单的拼写检查程序

spell 命令用于检查输入的文本是否有拼写错误，并列出拼写出错的单词。
例如，通过命令 spell，检查输入的文本是否有拼写错误，如下所示。

```
# spell                                     //输入文本
helle , Linux is a vwry perfett system.
I likk Linux.
But I am still a learnte.
Please tell me your interestind.           //按 Ctrl+d 组合键结束输入
helle                                     //显示可能拼写不正确的单词
interestind
learnte
likk
perfett
vwry
#
```

14.27 tail 命令：显示文件的末尾部分

tail 命令与 head 命令类似，用于读取文件的内容。只是 tail 命令将读取指定文件的末尾部分，然后输出到标准输出设备，而不像 head 命令从文件开始部分读取。如果不给 tail 命令指定读取的文件，默认把标准输入设备作为输入信息进行处理。tail 命令的的语法格式如下所示。

```
tail [--retry] [-c, --bytes=N] [-f, --follow[={name|descriptor}]] [-F] [-n, --lines=N]
[-max-unchanged-stats=N]
      [--pid=PID] [-q, --quiet, --silent] [-s, --sleep-interval=S] [-v,
--verbose] [--help] [--version]
```

其中各选项的具体说明如表 14-33 所示。

表 14-33 tail 命令的选项说明

选 项	说 明
--retry	仅与选项“-f”一同使用，对无法访问的文件也尝试打开
-c, --bytes=N	不输出最后的 N 字节
-f, --follow[={name descriptor}]	向文件追加数据
-F	与 “--follow=name --retry” 作用相同
-n, --lines=N	输出指定的后面 N 行，而不是默认的后 10 行
-max-unchanged-stats=N	与（--follow=name）一起使用，对于查看 N 次都没有改变大小的文件重新打开
--pid=PID	与选项“-f”一同使用，在指定进程结束后退出
-q, --quiet, --silent	静默模式，不输出标题
-s, --sleep-interval=S	与选项“-f”一同使用，休眠 S 秒
-v, --verbose	总是输出文件的标题
--help	显示帮助信息并退出
--version	输出版本信息并退出



例如，显示文件 123 后 10 行的内容，命令如下所示。

```
# cat 123 //显示文件 123 的内容
000000 00000000
111111 11111111
222222 22222222
333333 33333333
444444 44444444
555555 55555555
666666 66666666
777777 77777777
888888 88888888
999999 99999999
111111 11111111
111111 11111111

# tail 123 //显示文件 123 的后 10 行
222222 22222222
333333 33333333
444444 44444444
555555 55555555
666666 66666666
777777 77777777
888888 88888888
999999 99999999
111111 11111111
111111 11111111
```

例如，显示文件 123 的后两行，可以有如下两种方法。具体命令如下所示。

```
# tail 123 //显示文件末尾信息
000000 00000000
111111 11111111
222222 22222222
333333 33333333
444444 44444444
555555 55555555
666666 66666666
777777 77777777
888888 88888888
999999 99999999
111111 11111111
111111 11111111

# tail -2 123 //显示文件 123 的后 2 行
111111 11111111
111111 11111111

# tail +11 123 //从文件 123 的第 11 行开始显示，显示到文件结束
111111 11111111
111111 11111111
```

tail 命令还可以配合其他命令完成一些更为复杂的操作，例如对网络流量进行实时监测。首先，通过命令 tcpdump 记录网络 tcp 流量，并将数据重定向到文件 mynet.log，如下所示。

```
# tcpdump >mynet.log& //记录 tcp 流量
```

同时，通过命令 tail 显示文件 mynet.log 的末尾 10 行的内容。因为文件 mynet.log 是始终更新的，通过 tail 命令的选项“-f”，tail 会不断读取文件 mynet.log 文件的内容，不停地将 mynet.log 文件更新的后 10 行内容发送到标准输出设备，从而可以实现监测网络。具体命令

如下所示。

```
# tail -f mynet.log                                     //显示文件后 10 行内容
00:23:19.977572 192.168.20.148 > 192.168.20.122: icmp: echo request
00:23:19.977766 192.168.20.122 > 192.168.20.148: icmp: echo reply
00:23:21.049860 192.168.20.148 > 192.168.20.122: icmp: echo request
00:23:21.049931 192.168.20.122 > 192.168.20.148: icmp: echo reply
00:23:22.185189 192.168.20.148 > 192.168.20.122: icmp: echo request
00:23:22.186075 192.168.20.122 > 192.168.20.148: icmp: echo reply
00:23:23.583675 192.168.20.148 > 192.168.20.122: icmp: echo request
00:23:23.584751 192.168.20.122 > 192.168.20.148: icmp: echo reply
```

14.28 tac 命令：反序输出文件

tac 是 cat 的字母反序排列，tac 命令用来显示文件的内容，但与 cat 命令相反，该命令是以反序的方式显示文本内容。该命令的语法格式如下所示。

```
tac [-b, --before][ -r, --regex][ -s, --separator=STRING][ --help][ --version]
```

其中各选项的具体说明如表 14-34 所示。

表 14-34 tac 命令的选项说明

选 项	说 明
-b, --before	在前面加分隔符，而不是在后面
-r, --regex	将分隔符解析成正则表达式
-s, --separator=STRING	指定分隔符
--help	显示帮助信息
--version	显示版本信息

例如，通过命令 tac，反序显示文件 123 的内容，如下所示。

```
# cat 123                                               //显示文件内容
000000      00000000
111111      11111111
222222      22222222
333333      33333333
444444      44444444
555555      55555555
666666      66666666
777777      77777777
888888      88888888
999999      99999999

# tac 123                                               //反序显示文件内容
999999      99999999
888888      88888888
777777      77777777
666666      66666666
555555      55555555
444444      44444444
```



```
333333      33333333
222222      22222222
111111      11111111
00          00000000
```

14.29 tee 命令：读取标准输入到标准输出或文件

tee 命令用来将标准输入的内容，输出到标准输出设备或保存到指定的文件。该命令的语法格式如下所示。

```
tee [-a, --append][ -i, --ignore-interrupts][ --help][ --version]
```

其中各选项的具体说明如表 14-35 所示。

表 14-35 sum 命令的选项说明

选 项	说 明
-a, --append	附加到指定的文件，不进行覆盖
-i, --ignore-interrupts	忽略中断信号
--help	显示帮助信息
--version	显示版本信息

例如，通过命令 tee，将标准输入保存到文件 info，如下所示

```
# tee info //写文件并显示
I am Sullivan.
I am Sullivan.
I come from Beigin.
I come from Beigin.
I like Linux.
I like Linux.
Welcome to Linux world.
Welcome to Linux world.
# cat info //显示文件内容
I am Sullivan.
I come from Beigin.
I like Linux.
Welcome to Linux world.
```

例如，通过命令 cat，显示文件 info 的内容，并通过管道将文件 info 的内容输出并保存到文件 backup1、backup2 和 backup3 中，如下所示。

```
# cat info | tee backup1 backup2 backup3 //备份文件
I am Sullivan.
I come from begin
I like Linux
Welcome to Linux world
# cat backup1 //显示文件内容
I am Sullivan.
I come from begin
I like Linux
Welcome to Linux world
```



```
# cat backup2 //显示文件内容
I am Sullivan.
I come from begin
I like Linux
Welcome to Linux world
# cat backup3 //显示文件内容
I am Sullivan.
I come from begin
I like Linux
Welcome to Linux world
```

14.30 tr 命令：转换文件中的字符

tr 命令将从标准输入设备读入的字符串转换为指定的字符串，然后输出到标准输出设备。该命令的语法格式如下所示。

```
tr [-c, --complement][ -d, --delete][ -s, --squeeze-repeats][ -t, --truncate-set1]
[ --help][ --version] SET1 [SET2]
```

其中各选项的具体说明如表 14-36 所示。

表 14-36 tr 命令选项说明

选 项	说 明
-c, --complement	首先对 SET1 进行补齐
-d, --delete	删除 SET1 中的字符，不进行转换
-s, --squeeze-repeats	多个同样的字符只显示一个
-t, --truncate-set1	根据 SET2 的长度截取 SET1，使得二者长度相同
SET1, SET2	指定的字符串
--help	帮助信息
--version	版本信息

例如，通过命令 tr，将文件 abc.doc 中的小写字母转换为大写字母，并输出到标准输出设备，执行命令如下所示。

```
# cat abc.doc //显示文件内容
a
ab
abc
abcd
abcde
abcdef
abcdefg
abcdefgh
# cat abc.doc | tr a-z A-Z //将小写字母转换为大写字母
A
AB
ABC
ABCD
ABCDE
```



```
ABCDEF
ABCDEF
ABCDEF
#
```

例如，通过命令 `tr`，将文件 `abc` 中所有的字符“c”都用数字“3”替换，如下所示。

```
#cat| abc //显示文件内容
a
ab
abc
abcd
abcde
abcdef
abcdefg
abcdefg
# cat abc.doc | tr c 3 //将文件中的“c”替换为“3”
a
ab
ab3
ab3d
ab3de
ab3def
ab3defg
ab3defg
#
```

14.31 uniq 命令：检查文件中重复出现的行

`uniq` 命令用来检查文件中重复出现的行，可以将重复的部分删除，还可以显示重复的部分。该命令的语法格式如下所示。

```
uniq [-c, --count] [-d, --repeated] [-f, --skip-fields=N] [-i, --ignore-case] [-s,
--skip-chars=N] [-u, --unique]
[-w, --check-chars=N] [--help] [--version]
```

其中各选项的具体说明如表 14-37 所示。

表 14-37 `uniq` 命令选项说明

选 项	说 明
-c, --count	每一行前，显示重复次数
-d, --repeated	显示重复的行
-f, --skip-fields=N	忽略指定的字段
-i, --ignore-case	忽略大小写
-s, --skip-chars=N	忽略指定的字符
-u, --unique	仅输出不重复的行
-w, --check-chars=N	指定每行要比较的字符数目
--help	显示帮助信息
--version	显示版本信息

例如，通过命令 `uniq`，不显示文件 `test.doc` 中重复的行，如下所示。


```
# cat test.doc //显示文件内容
aaaaaaaaaaaaaaaaa
bbbbbbbbbbbbbbbbbb
bbbbbbbbbbbbbbbbbb
cccccccccccccccccc
cccccccccccccccccc
dddddddddddddddddd
# uniq test .doc //不显示重复的行
aaaaaaaaaaaaaaaaa
bbbbbbbbbbbbbbbbbb
cccccccccccccccccc
dddddddddddddddddd
#
```

例如，通过命令“`uniq -d test.doc`”，显示文件 `test.doc` 中重复的行，如下所示。

```
# cat test.doc //显示文件内容
aaaaaaaaaaaaaaaaa
bbbbbbbbbbbbbbbbbb
bbbbbbbbbbbbbbbbbb
cccccccccccccccccc
cccccccccccccccccc
dddddddddddddddddd
# uniq -d test.doc //仅显示重复的行
bbbbbbbbbbbbbbbbbb
cccccccccccccccccc
#
```

14.32 view 命令：查看文件内容

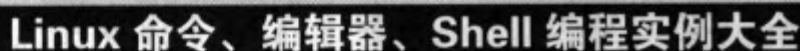
`view` 命令用来查看文本文件内容。实际上，`view` 命令是以只读方式连接到命令 `vi`。该命令的语法格式如下所示。

```
view [-b] [-d] [-m] [-n] [-r] [-R] [-s] [-V] [+num] [-C command] [--help] [-o N] [--version]
```

其中各选项的具体说明如表 14-38 所示。

表 14-38 view 命令选项说明

选 项	说 明
-b	二进制模式
-d	差异模式
-m	只读模式
-n	缓存不使用
-r	显示缓存信息
-R	只读模式
-s	运行时，不显示处理信息
-V	运行时，显示详细的处理信息

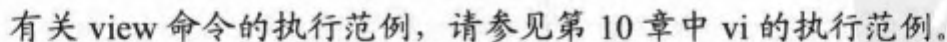


选 项	说 明
+ <code>[num]</code>	从指定行数开始显示
<code>-C command</code>	读取第一个文件之后执行指定的命令
<code>--help</code>	显示帮助信息
<code>-o N</code>	指定同时打开的窗口数
<code>--version</code>	显示版本信息

```
view /etc/initlog.conf
```

[illegible]

图 14-12 使用 view 命令打开文件



wc 命令：计算文件的字数、字节数、行数

636

所示。

```
wc [-c, --bytes][ -m, --chars][ -l, --lines][ -L, --max-line-length][ -w, --words][ --help][ --version]
```

其中各选项的具体说明如表 14-39 所示。

表 14-39 wc 命令选项说明

选 项	说 明
-c, --bytes	显示字节数
-m, --chars	显示字符数
-l, --lines	显示新行数
-L, --max-line-length	显示最长行的长度
-w, --words	显示单词数
--help	显示帮助信息
--version	显示版本信息

例如，计算文件 mytest 中的字符数，命令如下所示。

```
# cat mytest //显示文件 mytest 内容
initlog configuration file
lines preceded by comments
anything not recognized is ignored.
This sets the default logging priority.
means to discard any output lines that match the regular expression foo.
# wc -c mytest //计算字符数
206 mytest //包含 206 个字符
```

例如，计算文件 mytest 的行数，命令如下所示。

```
# cat mytest //显示文件 mytest 内容
initlog configuration file
lines preceded by comments
anything not recognized is ignored.
This sets the default logging priority.
means to discard any output lines that match the regular expression foo.
# wc -l mytest //计算文件的行数
5 mytest //共有 5 行
```

例如，显示文件 mytest 中包含的单词数，命令如下所示。

```
# cat mytest //显示文件 mytest 内容
initlog configuration file
lines preceded by comments
anything not recognized is ignored.
This sets the default logging priority.
means to discard any output lines that match the regular expression foo.
# wc -w mytest //显示单词数
30 mytest //单词数为 30
```

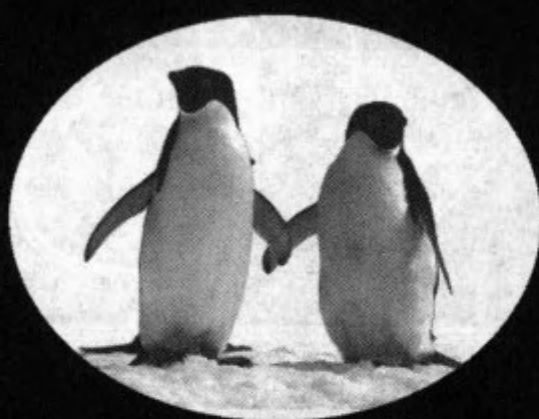
例如，通过命令 wc ，依次显示文件 mytest 文件的行数、单词数和字符数，如下所示。

```
# cat mytest //显示文件 mytest 内容
initlog configuration file
```



```
lines preceded by comments
anything not recognized is ignored.
This sets the default logging priority.
means to discard any output lines that match the regular expression foo.
# wc mytest //显示文件的行数、单词数和字符数
5 30 206 mytest //5 行、30 个单词、206 字符
```

新华书店
PDFG



LINUX

下篇 Shell 编程

Linux 系统下提供了大量可供使用的命令,通过 Shell 编程可以将这些命令进行组合,从而完成更为复杂的操作。

本篇详细阐述了 Linux 下 Shell 编程的相关内容,主要包括 Shell 编程基础知识、Shell 变量、条件测试表达式、Shell 的控制结构、函数的使用及 Shell 编程的综合实例。

Linux

第15章

Shell 编程基础

Shell 是操作系统的外壳，是用户使用操作系统的命令接口，是一个介于用户和系统内核之间的桥梁。Shell 接受用户以命令行方式输入的命令（包括系统提供的内部命令、独立存在于某个目录下的程序），也能解释执行由 Shell 命令编写的 Shell 程序。Shell、用户和内核的关系如图 15-1 所示。

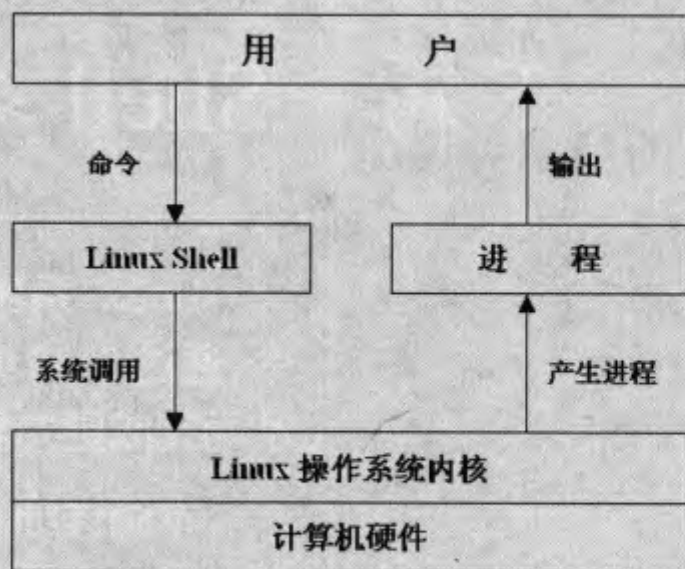
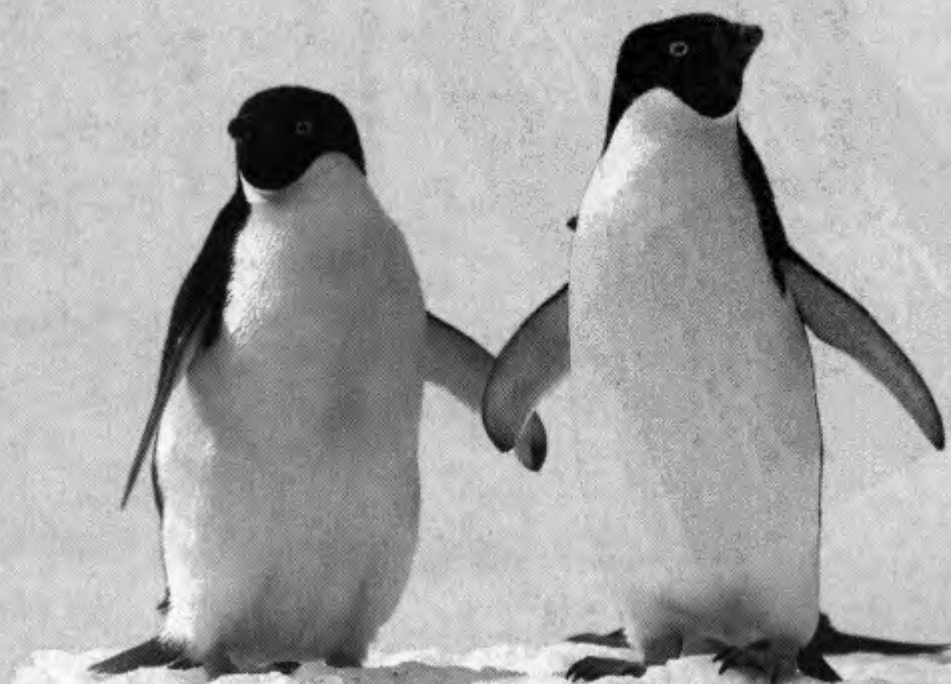


图 15-1 Shell、用户及内核的关系



15.1 Shell 概述

Shell 语言编写的程序仅能在使用 UNIX 和 Linux 的计算机上使用,与之相比,Java 程序则具有跨平台的特点,可以在装有 Java 虚拟机的任意平台上运行。

15.1.1 Shell 的功能与定义

Shell 是一个功能强大的计算环境,既可以将其作为命令解释器,也可以将其作为编程语言。Shell 的功能主要包括以下几方面。

- 接受来自键盘的命令。
- 检查命令的正确性。
- 命令错误则给出报错信息。
- 命令正确则使用相应的系统调用执行命令(产生进程)。
- 命令执行完毕,给出命令执行的结果。

15.1.2 Shell 的启动和退出

用户在成功登录进入系统后,系统将产生一个特定的 Shell (这是用户的第一个进程)负责解释执行用户的命令或 Shell 脚本。

在系统提示符后键入命令或 Shell 脚本名,可以执行指定的命令或 Shell 脚本。如果用户希望中止命令或脚本的执行,可以直接按 Ctrl+c 键。

如果用户结束工作希望退出系统,可以按 Ctrl+d 键,或者使用命令 `logout` 或 `exit`。

15.1.3 Shell 脚本介绍

用户可以将 Linux 的命令以编程的方式组合起来提交给 Shell,Shell 负责解释执行这种程序设计语言所编写的代码。Linux 把由这种程序设计语言构成的文件称为 Shell Script 或 Shell 脚本。

Shell 脚本是包含一个或多个 Shell 命令的集合,这些命令会按照编写的顺序而被依次执行。Shell 提供了一种利用简单命令组合实现复杂功能的机制。Shell 脚本中支持绝大多数编程语言中所能见到的程序控制结构,可以使用注释(“#”)对脚本的功能进行说明,可以使用条件测试表达式进行分支判断,可以使用循环语句处理大量数据,可以定义使用变量、数组等。Shell 脚本也提供对函数的支持。

Shell 脚本是解释执行而不是编译执行。脚本由 ASCII 文本组成,由 Shell 命令解释器读取并执行。当执行一个 Shell 脚本时,命令解释器会逐行、逐个循环地读取所有的 ASCII 文本,并从头到尾依次执行每条语句。

Shell 编程的实质是连续自动执行一系列命令。本书上篇中对 Linux 中的命令进行了详细介绍。表 15-1 中列出了 Shell 编程经常使用的命令及相关约定。



Shell 编程常用命令

命 令	说 明
pwd	显示当前目录
passwd	修改用户的密码
ls	列出目录中的内容
cd	改变当前的工作目录
*	匹配任意长度的任意字符
?	匹配任意一个字符
[]	任一属于字符组的字符
cat	一次性显示输出文件的全部内容
file	显示文件的类型
more	分屏显示文件的内容
clear	清屏，提示符回到屏幕左上方
cp	复制文件
mv	移动文件
chown	修改文件或目录的所有者
chgrp	修改文件或目录的用户组
chmod	修改文件或目录的权限
rm	删除文件
mkdir	创建目录
rmdir	删除目录，要求目录为空
grep	查找指定的字符串
find	查找指定的文件
egrep	支持扩展正则表达式的 grep 命令
>>	附加到指定文件的结尾
>	输出重定向
<	输入重定向
	管道符，用于连接多个命令，前一命令的输出作为后一命令的输入
	逻辑或

续表

命 令	说 明
&&	逻辑与
&	在后台执行
date	显示当前的日期和时间
echo	回显，即将字符串输出到标准输出设备
sleep	暂停指定的时间间隔
wc	计算文件的单词数、行数、字符数
head	查看文件的开始部分，默认为前 10 行
tail	查看文件的结尾部分，默认为后 10 行
diff	比较两个文件的差异
spell	拼写检查
lp	打印文件
cal	显示日历
who	显示当前登录用户的信息
talk	与其他用户对话
write	给指定用户发消息
df	对文件系统的磁盘空间使用情况进行统计
ps	显示进程相关信息
netstat	显示网络状态
basename	提取基本文件名
man	显示联机参考手册
su	临时切换到另一用户
cut	剪切文件
vi	启动 vi 编辑器
emacs	启动 emacs 编辑器
awk	过滤器
sed	流编辑器
pr	编排文本格式



Shell 脚本具有如下优点。

- 一次性完成目标；
- 不用重复键入一系列命令；
- 简化用户的操作。



建立 Shell 脚本文件同建立普通文件的方式一样，可以使用 vi、emacs 等来建立。

15.2 Shell 分类

Shell 具有不同的风格，就像 UNIX 操作系统有不同的风格一样。每种不同的风格都具有各自的命令和功能，最常用的 Shell 包括 Bourne Shell、Bourne Again Shell、C Shell、Korn Shell 和 Perl Shell 等。



本书编程部分所涉及的脚本语言如不是特别说明，均以流行的 Bourne Again Shell 为例。

用户可以选择自己喜欢的 Shell（在系统管理员为用户创建账号时指定），在/etc/passwd 文件最后一个字段中可以看到用户当前使用的 Shell 的名称。



可以使用 grep 命令查看/etc/passwd 文件，以显示当前登录用户使用的 Shell，命令格式如下。

```
grep $LOGNAME /etc/passwd
```

15.2.1 Bourne Shell

Bourne Shell（简称 bsh）是 UNIX 最初使用的 Shell，由 Stephen Bourne 在 1979 年创建，具有简单、编程能力强等特点。但其操作使用不够方便，在同用户交互方面不如其他几种 Shell 强大。



要启动 Bourne Shell，只需在命令行上键入“sh”并按回车键。

15.2.2 Bourne Again Shell

Bourne Again Shell（简称 Bash）是 Linux 默认的 Shell，也是最流行的一种 Shell，各个版本的 UNIX 和 Linux 几乎都支持 Bash。Bash 继承了 Bourne Shell 的标准并扩充了人机交互

的特性。Brian Fox 是 Bash 的创始人，被称为 Bash 之父。Bash 提供了命令历史、命令补全、命令编辑等功能。



要启动 Bas，只需在命令行上键入“bash”并按回车键。

15.2.3 C Shell

Bourne Shell 不支持在后台运行作业，它总是接管屏幕和键盘的控制，因此程序必须一个一个地依次执行，前一程序没执行完毕，后一程序就不能开始运行。

加利福尼亚大学伯克利分校的 Bill Joy 认识到 Bourne Shell 的上述不足，决定用 C 语言中的关键字来设计一种新的 Shell——C Shell。C Shell 提供了方便的用户界面设计，以及命令行编辑、拼写校正、命令历史替换、作业控制等功能。由于 C Shell 采用 C 语言风格，因而受到许多 C 程序员的青睐。此外，C Shell 还有一个扩充版本 TC Shell。



要启动 C Shell，只需在命令行上键入“csh”并按回车键。要启动 TC Shell，只需在命令行上键入“tcsh”并按回车键。

15.2.4 Korn Shell

贝尔实验室的 David Korn 设计了 Korn Shell（简称 ksh）。Korn Shell 是 Bourne Shell 的继承和发展，使用 Bourne Shell 编写的脚本不必修改就可以直接使用 Korn Shell 运行。同时，Korn Shell 包含 C Shell 的特性，具有较强的作业控制能力，如挂起、后台执行、唤醒、终止等，还提供了窗口功能。

15.2.5 Perl Shell

Perl Shell 是另一个比较流行的 Linux Shell。Perl 是 Practical Extraction 和 Report Language 的首字母缩写。Perl Shell 通常用来编写 Web 应用程序的通用网关接口（GCI）。

15.3 Shell 基础知识

与大多数编程语言相比，Shell 编程语言的语法简单，易于掌握。在学习 Shell 编程之前，首先要了解 Shell 的一些基本语法及约定，包括大小写约定、Shell 注释、输入输出，以及如何运行一个 Shell 程序等。

15.3.1 区分大小写

UNIX、Linux 都是区分大小写的，Shell 脚本同样也是区分大小写的。例如，Shell 脚本中的变量“var”与“Var”会被识别为不同的两个变量。



15.3.2 Shell 脚本中的注释

各种编程语言中都普遍使用注释。注释只是为阅读代码提供便利，相当于代码的解释和说明，通常用于记录代码的用途、代码编写人、修改时间或代码编写方法等。在代码执行时，注释会被 Shell 解释器自动忽略。一个优秀的程序员应该养成在代码中书写注释的习惯，良好的注释将有利于日后的维护和更新。

Bourne Again shell 和 TC Shell 的注释语法相同，使用符号“#”标识注释行。如果符号“#”在脚本的第一行出现并且其后没有感叹号“!”，或者“#”出现在脚本中其他任意位置上，那么 Shell 将其视为注释的开始。Shell 会忽略从符号“#”开始到行末（即下一个换行符）之间的所有内容。

例如，下面是/etc/inittab 文件的部分内容，代码中使用了大量注释。

```
# cat /etc/inittab //使用 cat 命令显示 inittab 文件内容
# //以下注释对该文件的功能进行描述
# inittab This file describes how the INIT process should set up
# the system in a certain run-level.
# //以下注释说明代码的作者
# Author: Miquel van Smoorenburg, <miquels@drinkel.nl.mugnet.org>
# Modified for RHS Linux by Marc Ewing and Donnie Barnes
# //以下注释对系统的运行等级进行描述
# Default runlevel. The runlevels used by RHS are:
# 0 - halt (Do NOT set initdefault to this)
# 1 - Single user mode
# 2 - Multiuser, without NFS (The same as 3, if you do not have networking)
# 3 - Full multiuser mode
# 4 - unused
# 5 - X11
# 6 - reboot (Do NOT set initdefault to this)
#
id:5:initdefault: //真正被执行的语句
... ..
```

15.3.3 I/O 重定向

在 Linux 系统中，数据流可分为 3 类：数据输入、数据输出和错误输出。相应地，每一个进程也都有 3 个特殊的文件描述指针：标准输入（standard input，文件描述指针为 0）、标准输出（standard output，文件描述指针为 1）和标准错误输出（standard error，文件描述指针为 2）。这 3 个特殊的文件描述指针使进程在通常情况下接收标准输入设备的输入，由标准输出设备显示输出。

标准输入通常指传给一个命令的键盘输入。例如，运行命令“ls -al”，其中“-al”是传给 ls 命令的标准输入。标准输出是一个命令的执行结果，例如运行“cat /file1.doc”命令后所看到的文件内容就是标准输出。通常标准输出被定向到显示器。如果在执行命令过程中发生错误，可能会存在一条错误消息，这是标准错误数据流，通常也被定向到显示器。例如在上例中，如果 file1.doc 文件不存在，系统会提示“cat: /file1.doc: 没有此文件或目录”，并从显示器输出。

有两种基本的方法可以用来重定向标准输入（stdin）、标准输出（stdout）和标准错误

(stderr)。可以利用管道把这些数据流之一输送给另外一条命令，也可以利用 I/O 重定向把这些数据流之一重定向到一个文件。管道和 I/O 重新定向是 Linux 系统中特有的概念。所谓管道是指将某个命令的输出结果传送到另一个命令，当成另外一个命令的输入，其代表符号是“|”。所谓 I/O 重定向是指将命令执行的结果重新导出到其他设备或文件（以“>”或“>>”来表示），或是重新导入到其他设备或文件（以“<”或“<<”来表示）。常用的输入输出重定向命令格式与执行结果如表 15-2 所示。

表 15-2 常用输入输出重定向命令格式说明

命 令 行	说 明
命令 1 > 文件 1	将命令 1 的输出结果重定向到文件 1
命令 1 &> 文件 1	将命令 1 的输出结果和标准错误输出一起重定向到文件 1
命令 1 >> 文件 1	将命令 1 的输出结果追加到文件 1 中
命令 1 2> 文件 1	将命令 1 的标准错误输出的结果重定向到文件 1 中
命令 1 < 文件 1	将文件 1 做为命令 1 的标准输入
命令 1 << 字符串 1	允许连续输入数据，直到接收到字符串 1

1. 输出重定向 (>)

一般在 Linux 下执行任何命令或程序的过程中，默认都是将结果输出到屏幕。若希望将标准输出保存到一个文件中，可以使用输出重定向的功能。例如，对/etc 目录下的所有文件建立一个清单，并保存到根目录的 etc_list 文件中，命令如下所示。

```
# ls /etc > /etc_list
# cat /etc_list                //查看建立的列表文件
a2ps.cfg
a2ps-site.cfg
acpi
adjtime
... ..
```

在执行上例中的“ls”命令时，由于所有原来会出现的输出内容都已重新定向到 etc_list 文件中，屏幕不会再显示任何输出结果。重定向的文件，如本例中的 etc_list，不需预先创建，系统在执行此命令中若发现重新定向的文件不存在，则会自动产生命令中指定的文；若文件已经存在，则会进行写覆盖。可以使用附加输出重定向(>>)把新添加的数据追加到已存在文件的尾部，来避免写覆盖的发生。

利用前面介绍过的“cat”命令，配合输出重定向操作，可以将数据直接从键盘录入文件中。例如，创建 file1.doc 文件并录入“Hello world !!!”，按 Ctrl+c 或 Ctrl+d 组合键可以结束文件的编辑，命令如下所示。

```
# cat > file1.doc
Hello World!!!                //按 Enter 键
//按 Ctrl+c 或 Ctrl+d
#
```

输入完毕后先按 Enter 键（否则最后一行不会存储），再按 Ctrl+C 或 Ctrl+d 组合键就可结束文件的编辑，系统会将此内容以 file1.doc 文件名存储。

例如，利用 cat 命令与输出重定向，将标准输入设备的输入输出到文件 temp 中保存，执



行命令如下所示。

```
# cat > temp //将标准输入重定向到 temp 文件中
Hello, everyone!
I am sullivan,I like Linux.
This is a test demo. //按 Enter 键
//按 Ctrl+d 或 Ctrl+c 结束输入
#
# cat temp //显示 temp 文件的内容
Hello, everyone!
I am sullivan,I like Linux.
This is a test demo.
#
```



Ctrl+c 组合键用于结束一个程序，Ctrl+d 组合键用于结束终端输入。建议使用 Ctrl+d 组合键结束终端输入。

2. 附加输出重定向(>>)

如果重新定向的文件原来已经存在，那么在使用输出重定向(>)给文件输入新的数据后，新数据将覆盖所有旧的数据。在硬盘空间足够的条件下，为了避免覆盖旧数据，可以采用 Linux 提供的“附加输出重定向”(>>)。附加输出重定向操作可以将新输入的数据附加(Append。在原有旧数据之后，表示符号为“>>”。从下例中可以看出输出重定向与附加输出重定向的区别。

```
# cat file1.doc
Hello World!!! //源文件内容
# cat > file1.doc
This is a new world!
# cat file1.doc
This is a new world! //源文件内容被覆盖
# cat >> file1.doc
I like this world.
# cat file1.doc //附加在源文件后
This is a new world!
I like this world.
```

3. 输入重定向(<)

输出重定向与输入重定向相反，前者是将命令或程序的执行结果通过屏幕或文件来输出，但是后者却是将输入设备(如键盘或文件)提供给命令来执行。例如，下面的 Shell 程序实现对/etc 目录的备份，其中使用输入重定向来读入 file 文件的内容。

```
#!/bin/sh
i=0;
find / -name etc > file;
while read LINE
do
    DIRS=$LINE
    BACKUP="/tmp/backup${i}.tar.gz"
    tar -zcvf $BACKUP $DIRS
    i=$((i+1))
done < file
```


4. 附加输入重定向 (<<字符串)

附加输入重定向经常用在电子邮件系统中，可以让用户自行定义一个字符串，如 `exit`，系统在收到此字符串前，会持续地将数据输入文件。下面是一个编写电子邮件的范例，定义的结束字符串是“`exit`”，用户可以连续输入邮件内容，直到输入“`exit`”。

```
# mail yang_mh@bit.edu.cn << exit
> This mail is a test.
> exit
```

5. 错误输出重定向 (2>)

输出重定向 (>) 操作在命令执行发生错误时，会将误信息直接显示到屏幕，并不记录到文件中。而错误输出重定向 (2>) 会在命令执行发生错误时，将错误信息直接输入到文件中，而不会再将信息结果显示在屏幕上，如下所示。

```
# ls
cal_txt    finger2_txt  groups_txt  last_txt  newgrp_txt  suple_txt  who.txt
file1.doc  finger3_txt  groups_txt2 ll        suple3_txt  suple_txt~ w.txt
file2.doc  finger_txt   id_txt      ln        suple3_txt~ w
# ls exec
ls: exec: 没有那个文件或目录
# ls exec 2> example
# cat example
ls: exec: 没有那个文件或目录
```

6. 标准输出与错误输出重定向 (&>)

标准输出与错误输出重定向 (&>) 可以将标准输出和错误输出信息一并重新定向到文件，屏幕上不会显示任何信息，如下所示。

```
# ls
cal_txt    file2.doc  finger_txt  id_txt    ln        suple3_txt~ w
example    finger2_txt  groups_txt  last_txt  newgrp_txt  suple_txt  who.txt
file1.doc  finger3_txt  groups_txt2 ll        suple3_txt  suple_txt~ w.txt
# ls id_txt file
ls: file: 没有那个文件或目录
id_txt
# ls id_txt file 2> doc //将错误输出重定向到文件 doc
id_txt
# cat doc
ls: file: 没有那个文件或目录
# ls id_txt file &> doc //将标准输出与错误输出一并重定向到文件 doc
# cat doc
ls: file: 没有那个文件或目录
id_txt
```

15.3.4 管道 (Pipe)

在 Linux 系统中，管道的主要功能是将其他程序的输出结果直接导出到另一个程序来作为输入数据，即将前一个程序的输出作为后一个程序的输入，管道操作符是“`|`”。管道的语法格式如下所示。

```
命令1 | 命令2 [ | 命令3 ... ]
```

可以将标准错误输出一一起送入管道，命令格式如下。

```
命令1 |& 命令2 [ |& 命令3 ... ]
```



管道操作符的左边一定是一条送往标准输出设备的命令，管道操作符的右边一定是一条从标准输入设备读入的命令。

例如，联合使用 `cat`、`sort`、`wc` 命令，统计文件 `student` 中没有重复的行数，执行命令如下所示（文件 `student.exe`）。

```
# cat student.exe
1 I am a linux learner
3 I am a linux learner
5 I am a linux learner
8 I am a linux learner
2 I am a linux learner
4 I am a linux learner
8 I am a linux learner
9 I am a linux learner
8 I am a linux learner
7 I am a linux learner
6 I am a linux learner
0 I am a linux learner
# cat student | sort -u | wc -l
10
```

可以看到，该命令将“`cat student`”的输出结果输入给命令 `sort`，由 `sort` 命令处理之后再输出给 `wc` 命令，最终显示没有重复的行数为 10。

例如，将 `ps` 命令的输出结果作为 `more` 命令的输入，以实现分页查看进程信息，命令如下所示。

```
#ps -aux | more
```

例如，以长格式查看 `/etc` 目录下的所有文件。由于 `/etc` 目录下的文件有很多，直接使用“`ls -al`”命令显示的内容会很快卷过屏幕，无法仔细查看。利用管道将“`ls -al`”命令的执行结果输入 `more` 或 `less` 命令，可以实现分页显示，命令如下所示。

```
#ls -al /etc | less
```

15.3.5 前台与后台

一个程序在运行时可以采用两种运行方式，一种是在前台运行，另一种是在后台运行。两者的主要区别如下所示。

- 前台运行：Shell 要等待命令（进程）结束，才能恢复自身的运行，即回到操作系统的提示符。即下一命令只有等待当前命令执行完毕才能输入。
- 后台运行：Shell 不必等待命令（进程）结束，就恢复自身的运行，即回到操作系统的提示符。用户可以直接输入执行下一命令。

有时需要执行的程序要花费很多时间（如从打印机输出、远程下载或上传、对源代码进行编译等），如果在前台运行，用户将无法继续进行其他工作。可将这些耗时程序放到后台运行。只需在程序名后添加符号“&”，格式如下所示。

```
command &
```

例如，在后台执行 `gcc` 命令，对 `myprog.c` 源程序进行编译，如下所示。

```
$ gcc myprog.c&
```


由于后台进程是 Shell 的一个子进程，所以当用户退出 Shell 时（即退出登录），后台进程将被自动结束。因此，在用户退出系统时，一定要注意是否有还未结束的后台进程正在运行。



在父进程结束时所有子进程都将被结束。

15.3.6 命令执行顺序

当用户交互地向 Shell 输入多个命令，或者编写 Shell 脚本时，必须确定命令的执行顺序。命令执行顺序可分为顺序执行和有条件执行。顺序执行是指多个命令依次执行，前后命令无因果关系。有条件执行是指命令之间存在条件关系，只有满足一定条件才能执行或不被执行。

(1) 顺序执行多个命令是指在命令行一条一条地输入命令，命令之间用分号“;”分隔，且分号前后的命令无因果关系。

例如，将命令 `echo`、`pwd` 和 `ls` 组合在一起，并依次顺序执行，如下所示。

```
# echo Hello World! ; pwd ; ls
Hello World!           //命令 echo 的输出结果
/root                  //命令 pwd 的输出结果
*                      dump.tmp index.html  install.log      ls.doc              //命令 ls
的输出结果
anaconda-ks.cfg hello  index.html.1 install.log.syslog mail
#
```

从输出结果可以看到，首先 `echo` 命令被执行，显示输出“Hello World!”；之后执行 `pwd` 命令显示当前的工作目录为“/root”；最后执行 `ls` 命令，列出当前目录内容。这行命令产生结果与下面示例中分别执行这三个命令相同，只是下面示例中每个命令执行完毕均产生一个提示符“#”，而本示例中仅在执行完三个命令之后才产生提示符。分别执行这三个命令的结果如下所示。

```
# echo Hello World!
Hello World!           //命令 echo 的输出结果

# pwd
/root                  //命令 pwd 的输出结果
# ls
*                      dump.tmp index.html  install.log      ls.doc              //命令 ls 的输出结果
anaconda-ks.cfg hello  index.html.1 install.log.syslog mail
#
```



分号“;”也可以被看作是一种命令分隔符，但是该分隔符不能立即启动命令执行，也不能改变命令的任何功能。用户可以在单独一个命令行中连续地输入一串命令，并用分号将其彼此分开，在命令行的最后输入回车键即可执行这一串命令。

(2) 有条件执行分为“&&”和“||”，区别如下所示。

- &&: 连接两个命令，前一命令执行成功后才能执行后一个命令，相当于逻辑与。
- ||: 连接两个命令，前一命令执行不成功才执行后一个命令，相当于逻辑或。



例如，只有在/home/sullivan 目录存在的情况下，才切换到该目录下，执行命令如下所示。

```
[root@localhost bin]# ls /home/sullivan && cd /home/sullivan
[root@localhost sullivan]#
```

例如，只有在目录/home/sullivan 下不存在文件 20.1 时，才将该文件复制到/home/sullivan 目录下：如果目录/home/sullivan 下该文件已经存在，则不再进行复制。利用“||”可以方便地完成该操作，执行命令如下所示。

```
# ls /home/sullivan/20.1 || cp 20.1 /home/sullivan
/home/sullivan/20.1
# ls /home/sullivan
20.1
```

15.3.7 Shell 通配符

Shell 中支持使用通配符，常用的通配符如下所示。

- `?`: 表示任意的单个字符。
- `*`: 表示任意长度的任意字符串。
- `[]`: 表示匹配放在`[]`中的字符集中的任意一个字符。
- `{ }`: 将大括号中的字符串以及前导字符串和后继字符串作为匹配条件。

例如，显示当前目录下的所有文件，命令如下所示。

```
# ls *
20.1 20.11 20.13 20.15.1 20.16.1 20.2 20.4 20.6 20.8 21.16.1 my.txt
20.10 20.12 20.14 20.15.2 20.16.2 20.3 20.5 20.7 20.9 21.16.2 prog.txt
```

例如，显示当前目录下以字符串“20.1”开头的文件，命令如下所示。

```
# ls 20.1*
20.1 20.10 20.11 20.12 20.13 20.14 20.15.1 20.15.2 20.16.1 20.16.2
```

例如，显示当前目录下所有以.txt 结尾的文件，命令如下所示。

```
# ls *.txt
my.txt prog.txt
```

例如，显示当前目录下所有名称包含“16”的文件，命令如下所示。

```
# ls *16*
20.16.1 20.16.2 21.16.1 21.16.2
```

例如，显示当前目录下所有文件名以“20”或“21”开头的文件，命令如下所示。

```
# ls 2[0,1]*
20.1 20.11 20.13 20.15.1 20.16.1 20.2 20.4 20.6 20.8 21.16.1
20.10 20.12 20.14 20.15.2 20.16.2 20.3 20.5 20.7 20.9 21.16.2
```

例如，显示所有由 4 个字符组成的文件名，且前 3 个字符必须是“20.”，命令如下所示。

```
# ls 20.?
20.2 20.3 20.4 20.5 20.6 20.7 20.8 20.9
```

例如，显示所有由 4 个字符组成的文件，且前 3 个字符必须是“20.”，第 4 个字符必须是 1、2、3 或 4，命令如下所示。

```
# ls 20.[1234]
20.2 20.3 20.4
```

通配符“`[]`”在表示一个连续的数字或字母范围时可以使用“-”。例如，`[1-9]`表示数字 1~9 之间任意一个数字，`[A-Z]`表示大写字母 A 到大写字母 Z 之间的任意一个字母，`[1-389]`表示数字 1、2、3、8、9 中的任意一个数字。可以使用逗号来表示多个集合，例如，`[A-H,w-z]`

表示大写字母 A 到 H, 小写字母 w 到 z 之间的任意一个字母。

在一个集合前使用前缀 “!” 表示取反, 即由指定集合之外的所有字符组成的集合。例如 [aoeiu] 表示所有元音字母中的任意一个字母, 而 [!aoeiu] 表示所有辅音字母中的任意一个字母。

例如, 显示所有由 4 个字符组成的文件, 且前 3 个字符必须是 “20.”, 第 4 个字符必须是 1、2、3 或 4, 命令如下所示。

```
# ls 20.[1-4]
20.2 20.3 20.4
```

例如, 显示所有由 4 个字符组成的文件, 且前 3 个字符必须是 “20.”, 第 4 个字符必须是 1、2、3、8 或 9, 命令如下所示。

```
# ls 20.[1-389]
20.1 20.2 20.3 20.8 20.9
```

在查找文件时, 大括号是经常使用的通配符, 其基本格式如下所示。

前导字符串{字符串 1, 字符串 2,}后继字符串

例如, 查找文件 cake.doc 和 cars.doc, 命令如下所示。

```
# ls
cake.doc cars.doc clock.doc cock.doc cool.doc cop.doc copy.doc cows.doc
# ll ca{ke,rs}.doc
-rw-r--r-- 1 root root 0 8月 16 17:27 cake.doc
-rw-r--r-- 1 root root 0 8月 16 17:27 cars.doc
```

例如, 通过 echo 命令显示输出多种匹配结果如下所示。

```
# echo m{oon,op,ud}s
moons mops muds
```

大括号也可以嵌套使用, 如下所示。

```
# echo c{a{r,n,to}on}s
carons canons catoons
```

15.3.8 单引号、双引号及反引号

Shell 中使用三种引号: 双引号、单引号和反引号。其中, 双引号对字符串中出现的 “\$”、“”、“\” 进行替换; 单引号不进行替换, 将字符串作为普通字符输出; 反引号会将字符串作为命令执行, 并返回执行的结果。其具体含义如下所示。

- 符双引号 (“ ”): 在双引号中的字符, 除了 “\$”、“”、“\” 和 “\” 以外的所有字符都被解释成字符本身。
- 单引号 (‘ ’): 在单引号中的所有特殊字符 (如 “\$”、“”、“\” 和 “\”) 都失去特殊意义, 而成为普通字符。
- 反引号 (` `): 在反引号中的字符被解释成命令, 例如, `cmd` 会被解释成命令 cmd 执行的结果。

例如, 在双引号中输出变量 PATH, 命令如下所示。

```
# echo "$PATH"
/usr/kerberos/sbin:/usr/kerberos/bin:/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin:/usr/X11R6/bin:/root/bin
```

可以看到, 在双引号中, 变量 PATH 被其值替换。例如, 在同样的环境下, 在单引号中引用变量 PATH, 其结果如下所示。



```
# echo '$PATH'
$PATH
```

可以看到，单引号将\$PATH 视为普通字符串显示输出。

例如，使用双引号显示输出 ls，其结果如下所示。

```
# echo "ls"
ls
```

例如，使用反引号显示输出 ls，其结果如下所示。

```
# echo `ls`
cake.doc cars.doc clock.doc cock.doc cool.doc cop.doc copy.doc cows.doc
```

可以看到，反引号将 ls 视为命令，并将 ls 命令的执行结果返回。下面示例中使用双引号，结果如下所示。

```
# echo "I am $USER"
I am root
```

可以看到，双引号中的\$USER 被替换。下面示例中使用单引号，结果如下所示。

```
# echo 'I am $USER'
I am $USER
```

可以看到，单引号中的\$USER 按原样输出，没有进行替换。

例如，下面示例中使用反引号获取命令“date +%y”的执行结果如下所示。

```
# echo Today `date +%y`
Today ' year is 08
```

15.3.9 指定使用的 Shell

Shell 脚本文件的第一行通常会放置一行特殊的字符串，告诉操作系统使用哪个 Shell 来执行这个文件。因为操作系统在试图执行文件之前将检查该程序的起始字符串，这些字符可以避免让操作系统进行不必要的失败尝试。

如果脚本的前两个字符是#!，那么系统会将这两个字符后面的那些字符作为执行该脚本的命令解释器的绝对路径名。该路径可以指定到任何程序的路径名，而并不仅仅局限于 Shell。

例如，下面的示例中指定由 bash 来运行脚本（代码 bashscript）。

```
1 #!/bin/bash
2 #filename:bashscript
3 echo "这是一个 bash 脚本程序。"
4 echo "-----"
5 ps -f
```

其中，“ps -f”可以用来显示正在执行该脚本的 Shell 的名字。该脚本执行结果如下所示。

```
# chmod u+x bashscript //赋予文件 bashscript 可执行权限
# ./bashscript //执行程序 bashscript
这是一个 bash 脚本程序。
```

UID	PID	PPID	C	STIME	TTY	TIME	CMD
root	30474	30472	0	11:46	pts/1	00:00:00	-bash
root	30535	30474	0	11:50	pts/1	00:00:00	/bin/bash ./bashscript
root	30542	30535	0	11:50	pts/1	00:00:00	ps -f
#							

从显示结果可以看到，当前使用的 Shell 为 bash，执行脚本 bashscript 的 Shell 也为 bash。

如果用户希望使用不同于当前使用的 Shell 的其他 Shell 运行脚本，可以在#!字符后指定使用其他 Shell。例如，下面的示例中使用 TC Shell 执行脚本（代码 tcshscript）。


```

1 #!/bin/tcsh
2 #filename:tcshscript
3 echo "这是一个 TCsh 脚本程序。"
4 echo "-----"
5 ps -f

```

该脚本执行结果如下所示。

```

#chmod u+x tcshscript          //赋予文件 tcshscript 可执行权限
# ./tcshscript                 //执行程序 tcshscript
这是一个 TCsh 脚本程序。
-----
UID      PID  PPID  C  STIME TTY          TIME CMD
root    30474 30472  0  11:46 pts/1      00:00:00 -bash
root    30601 30474 30  12:26 pts/1      00:00:00 /bin/tcsh ./tcshscript
root    30621 30601  0  12:26 pts/1      00:00:00 ps -f

```

从显示结果可以看到,当前正在使用的 Shell 为 bash,执行脚本 tcscript 的 Shell 为 TC Shell。因为在脚本中有“#!/bin/tcsh”一行字符,所以操作系统可以确定由 tcsh 来执行这个脚本,而不管用户当前正在运行的 Shell。

如果没有使用“#!”指定可执行程序名,将调用当前正在使用的 Shell 执行脚本。

15.3.10 运行脚本

运行 Shell 脚本有三种方法,如下所示。

(1) 通过 chmod 命令把文件的权限设成可读、可执行,然后直接执行该可执行文件,格式如下所示。

```

$ chmod u+x 脚本文件名
$ ./脚本文件名及其参数

```

其中,chmod 命令中的选项“u”表示 user,表示该文件的所有者。而“u+x”表示为文件的所有者添加执行权限。如果其他用户也要执行这个文件,就必须改变该文件的组访问权限(“g”)或其他用户访问权限(“o”)。



任何用户希望把文件名作为命令执行时,都必须具有执行权限。如果该文件是一个 Shell 脚本,用户尝试执行这个文件时,还必须具备读权限。而在执行一个二进制可执行文件(已编译程序)时,不需要读权限。

(2) 直接使用 Shell 的启动命令来执行脚本。例如,使用 bash 命令执行脚本的格式如下所示。

```

$ bash 脚本文件名及其参数

```

由于 bash 命令默认读取一个包含命令的文件,所以不需要为脚本赋予执行权限。如果希望使用 TC Shell 来执行脚本,格式如下所示。

```

$ tcsh 脚本文件名及其参数

```

同样,使用 sh 命令也可以执行脚本,格式如下所。

```

$ sh 脚本文件名及其参数

```

值得注意的是,用命令 sh 不能调用原 Bourne Shell,因为 sh 命令实质上是/bin/bash 的符号链接文件,其仅仅是 bash 命令的另一个名称而已。使用 ll 命令可以查看这种链接关系,如下所示。



```
# ll /bin/sh
lrwxrwxrwx 1 root root 4 8月 7 14:33 /bin/sh -> bash
```

在以下示例中分别使用“tcsh”、“bash”、“sh”命令执行脚本 tcshscript、bashscript，结果如下所示。

```
# tcsh tcshscript
这是一个 TCsh 脚本程序。
-----
UID      PID  PPID  C  STIME TTY      TIME CMD
root    30474 30472  0  11:46 pts/1    00:00:00 -bash
root    30731 30474  0  13:00 pts/1    00:00:00 tcsh tcshscript
root    30751 30731  0  13:00 pts/1    00:00:00 ps -f
```

从显示结果可以看到，当前使用的 Shell 为 bash，执行脚本 tcshscript 的 Shell 为 tcsh。下面使用命令 bash 执行脚本 bashscript。

```
# bash bashscript
这是一个 bash 脚本程序。
-----
UID      PID  PPID  C  STIME TTY      TIME CMD
root    30474 30472  0  11:46 pts/1    00:00:00 -bash
root    30752 30474  0  13:00 pts/1    00:00:00 bash bashscript
root    30759 30752  0  13:00 pts/1    00:00:00 ps -f
```

从显示结果可以看到，当前使用的 Shell 为 bash，执行脚本 bashscript 的 Shell 也为 bash。下面使用命令 sh 执行脚本 bashscript。

```
# sh bashscript
这是一个 bash 脚本程序。
-----
UID      PID  PPID  C  STIME TTY      TIME CMD
root    30474 30472  0  11:46 pts/1    00:00:00 -bash
root    30760 30474  0  13:00 pts/1    00:00:00 sh bashscript
root    30761 30760  0  13:00 pts/1    00:00:00 ps -f
```

从显示结果可以看到，当前使用的 Shell 为 bash，执行脚本 bashscript 的 Shell 为 sh。



尽管可以使用 Shell 的启动命令来执行脚本，但该方法将会使脚本的执行速度比具有执行权限之后直接调用脚本慢一些。所以大多数用户更喜欢将文件改为可执行，在命令行上直接输入文件名来运行脚本。

(3) 使用 bash 内部命令“source”或“.”运用 Shell 脚本，格式如下所示。

```
$ source 脚本文件名及其参数
$ . 脚本文件名及其参数
```



“.”和后面的程序名之间要留有空格。

例如，编写一个简单的 Shell 程序 hello，内容如下所示。

```
$ cat hello
#Filename:hello
echo "Hello world!"
```

采用以上三种方法执行程序 hello，如下所示。

第一种：将 hello 这个文件的权限设定为可执行如下所示。


```
$ chmod 755 hello //或 chmod u+x hello
$ ./hello
hello world !
```

第二种：直接使用 Shell 命令 sh 或 bash 来执行如下所示。

```
$ sh hello
hello world!
```

或

```
$ bash hello
hello world!
```

第三种：使用 bash 内部命令 “source” 或 “.”，如下所示。

```
$ source hello
hello world
```

或

```
$ . hello
hello world
```



一个命令或 Shell 程序执行完毕之后，将返回给父进程一个退出状态。退出状态是一个 0~255 之间的整数。通常，如果退出状态为 0，表示命令或 Shell 程序被成功执行；如果退出状态不是 0，则表示命令或 Shell 程序在某些方面失败了。变量 “\$?” 中保存了最近一次命令或 Shell 程序执行的退出状态。

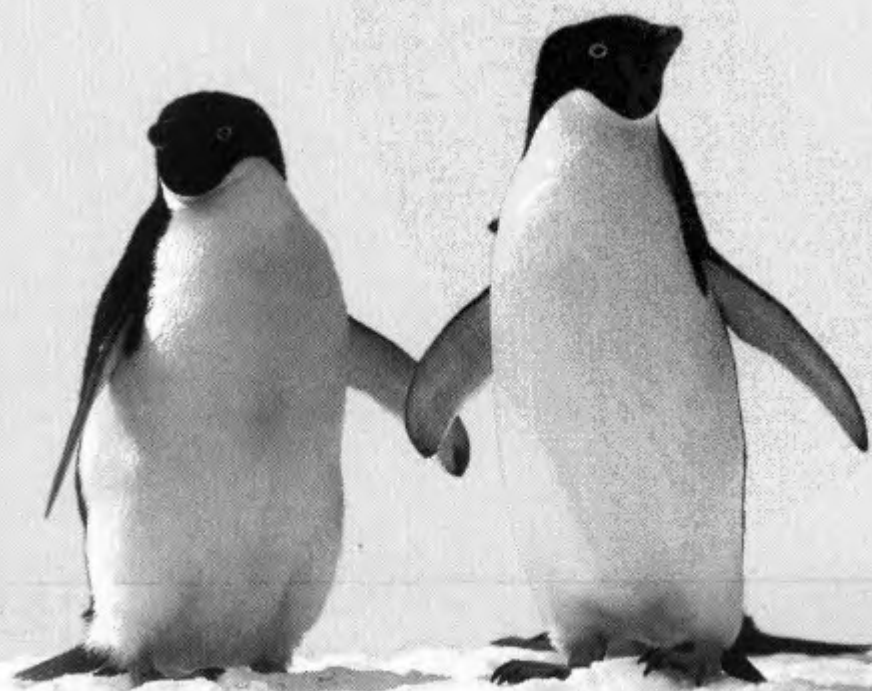
UNIX

第16章

Shell 变量及相关操作

任何一种编程语言都有变量的定义。顾名思义变量 (variable) 是指在程序执行过程中其值可以改变的量。变量是计算机内存中被命名的存储位置, 其中存放数字或字符串等数据。事实上, 变量是一个指向数据的指针。

变量指向的数据称为变量的值, 变量的值可以是数字、文本、文件名、设备或其他类型的数据。变量为用户提供了一种存储、检索和操作数据的途径。



16.1 Shell 变量类型

Shell 中变量的类型分为环境变量、位置变量、预定义的特殊变量以及用户自定义变量。每种变量都有其特殊的功能。

16.1.1 环境变量

Shell 的环境变量是一类 Shell 预定义变量，是用于设置系统运行环境的变量。环境变量由系统统一命名，部分环境变量的值由系统设定，部分环境变量的值可由用户给定。

环境变量的名称由大写字母组成，常用的 Shell 环境变量如下所示。

(1) HOME：用户主目录的全路径名，一般为“/home/用户登录的用户名”。不带参数的 `cd` 命令默认进入环境变量 HOME 指定的目录，即 `cd` 与“`cd $HOME`”等价。

```
# echo $HOME           //显示变量 HOME 当前值
/root
# cd                   //将当前目录切换到登录用户的主目录
# pwd                 //显示当前的工作目录
/root
# cd $HOME            //进入变量 HOME 指定的目录
# pwd                 //显示当前的工作目录
/root
```

(2) PATH：变量 PATH 中定义了一些目录路径，路径由冒号分隔。在执行命令或 Shell 脚本时，Shell 会按 PATH 变量中设定的顺序搜索这些目录，找到的第一个匹配的命令或 Shell 脚本将被执行。为操作方便至少要设置 `PATH=/bin:/usr/bin:/usr/local/bin` 路径。例如，显示当前的搜索路径，命令如下所示。

```
# echo $PATH           //显示变量 PATH 当前值
/usr/kerberos/sbin:/usr/kerberos/bin:/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr
/sbin:/usr/bin:/usr/X11R6/bin:/root/bin
```

可以采用下面的命令格式将新的路径添加到当前路径中。

```
$PATH=$PATH:新路径名
```

(3) TERM：终端类型。例如，通过变量 TERM 显示当前的终端类型，如下所示。

```
# echo $TERM
xterm
```

(4) PWD：当前工作目录的绝对路径。该变量的取值会随 `cd` 命令的使用而改变。例如，通过变量 PWD 显示当前目录的绝对路径，如下所示。

```
# echo $PWD
/root
```

(5) PS1：主提示符。对于根用户，默认的主提示符为“#”，对于普通用户，默认的主提示符为“\$”。例如，下面示例通过重新设置变量 PS1 的值，改变当前使用的主提示符。

```
# echo $PS1           //显示变量 PS1 的值
[\u@\h \W]\$
# PS1="Please enter command : " //重新给变量 PS1 赋值 "Please enter command :"
```



```
Please enter command : //提示符已修改
```

```
Please enter command :
```

(6) PS2: 辅助提示符。在 Shell 接收用户输入命令过程中, 如果用户在输入行的末尾输入 “\” 然后回车, 或者 Shell 判断用户虽然按回车键, 但输入的命令未结束时, 将显示辅助提示符。用户可以在辅助提示符下继续输入命令, 默认的辅助提示符为 “>”。例如, 下面示例中通过输入 “\”, 主动使用辅助提示符。

```
$echo $PS2 //显示变量 PS2 的值, 默认为 “>”
```

```
>
```

```
$echo "Hello ,\ //输入\, 并按回车
```

```
> I am from Beijing"
```

```
Hello ,I am from Beijing
```

下面示例中, Shell 判断用户虽然按回车键但输入的命令未结束, 自动显示辅助提示符, 如下所示。

```
$echo "Ha ha, Learning Linux is a very interesting work.
```

```
> I like Linux."
```

```
Ha ha, Learning Linux is a very interesting work.
```

```
I like Linux.
```

(7) SHELL: 定义 Shell 的解释器路径。例如, 显示当前使用的 Shell 解释器的绝对路径, 命令如下所示。

```
$echo $SHELL
```

```
/bin/bash
```

(8) MAIL: 定义系统信箱的路径。变量 MAIL 中保存了邮箱的绝对路径。例如, 使用 echo 命令显示当前邮箱的路径如下所示。

```
$echo $MAIL
```

```
/var/spool/mail/root
```

(9) LOGNAME: 登录用户的用户名。例如, 通过显示变量 LOGNAME 的值, 查看当前登录用户的用户名, 如下所示。

```
$echo $LOGNAME
```

```
root
```

(10) UID: 当前用户的 UID。例如, 通过显示变量 UID 的值, 查看当前登录用户的 UID, 如下所示。

```
$echo $UID
```

```
0
```



用 env 或 set 命令可以显示当前环境变量的名称和它们的值, 也可用 env 或 set 命令来设置环境变量。

16.1.2 位置变量

位置变量是依据出现在命令行上的参数的位置来确定的变量。在调用 Shell 程序的命令行中, 参数的位置定义如下所示。

```
$命令 参数1 参数2 参数3 ...
```


当 Shell 解释一个命令时，会将位置变量与命令行中的每一参数关联，依次如下。

\$0: 对应当前执行的命令名（或 Shell 程序的文件名）。

\$1: 对应第一个位置参数。

\$2: 对应第二个位置参数。

\$3: 对应第三个位置参数。

... ..

\$9: 对应第 9 个位置参数。

这 10 个位置变量对应命令行上各项共 10 个位置参数，命令名（或 Shell 脚本名）对应 \$0，命令的第一个参数对应 \$1，命令的第二个参数对应 \$2，依此类推。

当命令行上的参数多于 9 个时，需要用 shift 命令移动位置参数。每执行一次 shift 命令，变量 0 不变，从 1 开始位置参数左移一位。例如，使用 shift 命令左移一次，即把 2、3、4 ... 9 的值分别赋值给 1、2、3 ... 9，其中第 9 位由于数据已被移出，变为空。

例如，编写 shifttest 程序，内容如下所示（代码 shifttest）。

```
$cat shifttest
echo $0 $1 $2 $3...$9
shift                                //左移一位
echo $0 $1 $2 $3...$9               //再次输出参数 0~9
shift 2                             //再次左移 2 位
echo $0 $1 $2 $3...$9               //再次输出参数 0~9
```

带位置参数执行 shifttest 程序，如下所示。

```
$chmod u+x shifttest
$./shifttest 1 2 3 4 5 6 7 8 9      //带参数 1、2...9
shifttest 1 2 3 4 5 6 7 8 9         //显示输出
shifttest 2 3 4 5 6 7 8 9           //左移一位，最右侧为空
shifttest 4 5 6 7 8 9               //再次左移 2 位，最右侧 3 位为空
```

例如，编写 whileshift 程序，利用 while 循环和 shift 命令依次输出位置参数，具体代码如下所示（代码 whileshift）。

```
1 #!/bin/bash                        //程序第 1 行的“#!/bin/bash”表示调用 bash 来执行脚本程序
2 #filename: whileshift              //程序第 2 行的“#filename: whileshift”表示该程序的名称为
whileshift
3 echo "----- 循环开始 -----" //显示输出“----- 循环开始 -----”
4 while [ $# -gt 0 ]                 //第 4~8 行为 while 循环，其中控制条件为“$# -gt 0”，
                                     //即表示在循环变量$#的值等于 1 时，继续循环，否则退出循环
5 do                                 // while 循环中的 do 语句
6     echo "当前的位置参数为: $*" //显示输出“当前的位置参数为: $*”
7     shift                          //使用 shift 命令左移参数一位
8 done                               // while 循环中的 done 语句
9 echo "-----循环结束 -----" //显示输出“-----循环结束 -----”
```

该代码的执行过程如下所示。

```
#chmod u+x whileshift                //赋予文件 whileshift 执行权限
# ./whileshift aa bb cc dd ee ff gg //带参数执行程序 whileshift.
----- 循环开始 -----
当前的位置参数为: aa bb cc dd ee ff gg
当前的位置参数为: bb cc dd ee ff gg
当前的位置参数为: cc dd ee ff gg
当前的位置参数为: dd ee ff gg
当前的位置参数为: ee ff gg
```



当前的位置参数为: ff gg
当前的位置参数为: gg
-----循环结束 -----

16.1.3 预定义的特殊变量

预定义的特殊变量与环境变量（环境变量也是一类预定义变量）类似，由 Shell 根据实际情况来设置。所不同的是，该类变量具有特殊的含义，其值不能由用户重新设置。所有的预定义特殊变量都由“\$”符号与另一个符号组成，常用的预定义特殊变量如下所示。

- **\$#**: 实际位置参数个数（不包括 Shell 脚本名）。
- **\$***: 命令行中的所有位置参数组成的字符串。
- **!**: 上一个后台命令对应的进程号。
- **\$?**: 表示最近一条命令执行后的退出状态（返回值），为十进制数。一般命令执行成功返回值为 0。
- **\$\$**: 当前进程号 PID。

例如，编写 Shell 程序 myprg，myprg 文件内容如下所示（代码 myprg）。

```
$ cat myprg
echo $#           //显示参数个数
echo $*           //显示所有参数
echo $1 $2 $3     //显示第 1、2、3 参数
echo $1           //显示第 1 个参数
```

带参数执行 myprg 程序，如下所示。

```
$ chmod u+x myprg           //赋予文件 myprg 执行权限
$ ./myprg A B C D           //带参数执行程序 myprg
4
A B C D
A B C
A
```

“\$?” 用于检查上次命令是否被正确执行。在 Linux 中，命令退出状态为 0 通常表示该命令被正确执行，任何非 0 值都表示命令出错。例如，下面示例（代码 lastresult）用于对指定的目录进行备份，其中综合使用了多个特殊变量。

```
1 #!/bin/bash           //程序第 1 行的“#!/bin/bash”表示调用 bash 来执行脚本程序
2 #filename: lastresult //程序第 2 行的“#filename: lastresult”表示该程序的名称为 lastresult
3 if [ $# -eq 1 ];then  //第 3~14 行为 if...then...else...fi 结构，用于判断命令行参数数量是否为 1。
                        //如果为 1 则执行第 4~11 行；如果不为 1 则执行第 13 行，给出错误提示
4   DIR=$1              //定义变量 DIR，且赋值为位置变量 $1 的值
5   FILE='basename $0' //“FILE='basename $0'”用于提取该程序的程序名。其中“$0”对应该程序
                        //的路径及程序名，命令“basename $0”表示去除变量 $0 中的路径，仅提
                        //取程序名，反引号表示将“basename $0”的执行结果返回，赋值给变量
                        //FILE
6   DIR2='basename $1'  //“DIR2='basename $1'”用于提取 $1 中去除路径的目录名。其中“$1”对
                        //应该程序的第 1 个参数，命令“basename $1”表示去除路径的目录名，
                        //反引号表示将“basename $1”的执行结果返回，赋值给变量 DIR2
7   if [ -x $DIR ];then //第 7~11 行是 if...then...else...fi 结构，用于判断指定的目录是否存在，
                        //如果存在则进行备份，否则显示目录不存在的提示信息
                        //“$DIR is not exist,please try again!”。其中“-x $DIR”表
```

示判断目录


```

//DIR 是否存在。
8   tar zcvf ${FILE}${DIR2}.tar.gz $DIR //使用 tar 命令对变量 DIR 指定的目录进行备份, 备
份名为
// "${FILE}${DIR2}.tar.gz"
9   else // if 结构中的 else 语句, 匹配第 7 行的 if
10      echo "$DIR is not exist,please try again!" //“echo "$DIR is not exist,please
try again!” 用于显示 “$DIR
// is not exist,please try again!”
11   fi // if 结构中的 fi 语句, 匹配第 7 行中的 if, 表示 if 结
构结束
12 else // if 结构中的 else 语句, 匹配第 3 行中的 if
13   echo "Please enter an argument." //用于显示输出 “Please enter an argument.”
14 fi // if 结构中的 fi 语句, 匹配第 3 行中的 if, 表示 if 结
构结束
```

该代码执行过程如下所示。

```

#chmod u+x lastresult //赋予文件 lastresult 可执行权限
#./lastresult /prog/student1 //带参数执行程序 lastresult
tar: Removing leading '/' from member names
prog/student1/
# ls //列出目录
16.1.3student1.tar.gz student1
```

16.1.4 用户自定义的变量

用户可以根据需要自行定义变量。变量名称区分大小写，例如 Shell 会认 name、Name、NAME 是不同的变量名。变量的名称由字母或下划线开头，后面是任意数量的字母、数字、下划线。表 16-1 列出了一些合法和非法的变量名。

表 16-1 变量名示例

合法的变量名	非法的变量名
_name	7sentence
Size	!pp
my_define	error&warn



为了使变量名和命令名相区别，建议将所有的变量名用大写字母来表示。

16.2 变量的操作

变量在 Shell 的功能和灵活性等方面起到了非常重要的作用，它也是所有 Shell 程序的重要组成部分。Shell 变量的基本操作包括变量的创建、删除等。

16.2.1 创建变量

有两个内置命令 declare 和 typeset 可用于创建变量。通过命令的选项设置还可以设定变



量的创建方。例如，使用 `declare` 命令的“-r”选项可以创建只读变量，只读变量是不能被撤销，不带任何选项执行 `declare` 命令，将列出所有设置的变量。

除了使用 `declare` 或 `typeset` 命令来创建和设置变量之外，通过将值直接赋予一个变量名也可以创建一个变量。为变量赋值时，变量名前不应加美元符号“\$”，格式如下所示。

```
变量名=变量值
```

其中，等号“=”前后不可有空格。在 Shell 中，不需要用显式的语法来声明变量（有些高级语言，如 C，要求变量必须先声明再使用）。变量在第一次被赋值时就被声明了，即设置变量初值的同时也就对变量进行了声明，用户可以在任何需要的时候，直接让 Shell 建立一个新变量。

例如，定义变量 `CITY` 的值为“Beijing”，命令如下所示。

```
CITY = Beijing
```

对于包含空格的变量，则整个字符串都要用双引号括起来。例如，变量 `USAGE` 的值为“usage: [-d : According to /home directory]”，定义变量如下所示。

```
USAGE="usage: [ -d : According to /home directory] "
```

使用变量时，需要在变量名之前加美元符号“\$”，表示引用变量的值。例如，显示变量 `USAGE` 的值，如下所示。

```
# USAGE="usage: [ -d : According to /home directory] "
# echo $USAGE
usage: [ -d : According to /home directory]
# CITY=Beijing
# echo $CITY
Beijing
```

16.2.2 区分变量名

有时变量名会与其他文字混在一起使用，很容易引起混淆。例如，下面示例中定义了变量 `order`，赋值为 22，命令执行如下所示。

```
# order=22
# echo "Tom is $ordernd."
Tom is
```

可以看到，显示的结果并不是“Tom is 22nd”，而仅仅是“Tom is”。因为 Shell 会将“ordernd”视为变量进行搜索，由于没有定义 `ordernd` 变量，所以显示为空。此时，可以使用大括号来区别变量与其他字符，如下所示。

```
# echo "This is ${order}nd"
Tom is 22nd.
```

可以看到，正确输出了“Tom is 22nd.”。

16.2.3 删除变量

删除变量的方法是使用 Shell 内部的 `unset` 命令。如果变量未被设置成只读属性，则用户自定义变量和环境变量都可以通过 `unset` 命令删除。

例如以下示例中，通过使用 `unset` 命令将已经定义的 `name` 数组变量删除，执行命令如下所示。

```
#typeset //显示当前已定义的变量
... ..
```



```

UID=0
USER=user1
USERNAME=root
XPVM_ROOT=/usr/share/pvm3/xpvm
name=([0]="李小平" [1]="李四" [2]="王小五" [3]="赵勤" [4]="李小平")
#
# unset name                //删除变量 name
#typeset                    //显示当前已定义的变量
... ..
UID=0
USER=user1
USERNAME=root
XPVM_ROOT=/usr/share/pvm3/xpvm //name 数组变量已删除
#

```

可见，unset 命令把变量 name 从 Shell 内存中删除。

16.3 变量的赋值

Shell 中变量值的输入有多种方法，如直接赋值、通过 read 命令赋值、使用命令行参数赋值和利用命令的输出结果赋值等。

16.3.1 使用 read 命令赋值

read 命令是一个内置命令，可以从标准输入设备或从一个文件读取数据。read 命令读取一个输入行直到遇到一个换行符为止。当读取时，行末尾的换行符将被转换成一个空字节。read 命令的语法格式如下所示。

```
read 变量1 变量2 ...
```

Shell 脚本在执行到 read 命令时，将暂停脚本的执行并等待键盘的输入。当用户输入数据并按回车键后，将完成赋值操作，脚本会继续执行。该命令在给变量赋值时遵循如下规则。

- 多个数据或变量之间用空格分隔。
- 若变量个数与数据个数相等，则对应赋值。
- 若变量个数大于输入数据个数，则没有输入数据的变量取空值。
- 若变量个数小于输入数据个数，则将多余的数据赋值给最后一个变量。

如果在 read 命令等待输入时，用户按下了组合键 Ctrl+d，则会终止脚本的执行。通过使用 read 命令，脚本可以方便地接受用户的输入并将输入信息保存到变量中。

例如，下面程序 read1 中使用 read 命令将用户的输入保存到变量 name 中，然后通过 echo 命令显示输出，程序代码如下所示（代码 read1）。

```

1 #!/bin/bash
2 #filename:read1
3 echo -n "请输入您的用户名: "
4 read name
5 echo "您输入的用户名为: $name "

```

其中，代码第 3 行提示用户输入用户名，选项“-n”表示用户的输入与提示信息出现在同一行上。该程序的执行过程如下所示。



```
#chmod u+x read1 //赋予文件 read1 执行权限
#./read1 //执行程序 read1
请输入您的用户名: 张三 //输入“张三”，提示信息与用户输入出现在同一行
您输入的用户名为: 张三
```

例如，下面的程序 `getfullname` 中使用 `read` 命令完成了变量的赋值操作，程序代码如下所示（代码 `getfullname`）。

```
1 #!/bin/bash
2 #filename:getfullname
3 echo "Please enter your First Middle and Last name:"
4 read XX YY ZZ
5 echo "your First name : $XX"
6 echo "your Middle name: $YY"
7 echo "your Last name : $ZZ"
8 echo "your full name : $XX $YY $ZZ"
```

其中，代码第 4 行通过 `read` 语句读入用户输入的数据，并依次赋值给 `XX`、`YY`、`ZZ` 三个变量。执行 `getfullname` 程序如下所示。

```
# ./getfullname
Please enter your First Middle and Last name:
Ming Hua Yang
your First name : Ming
your Middle name: Hua
your Last name : Yang
your full name : Ming Hua Yang
```

当用户输入的数据个数多于变量个数时，多余的数据将全部赋给最后一个变量，如下所示。

```
# ./getfullname
Please enter your First Middle and Last name:
Ming Hua Yang Sull //输入数据多于 3 个
your First name : Ming
your Middle name: Hua
your Last name : Yang Sull
your full name : Ming Hua Yang Sull
```

如果没有指定一个变量来保存 `read` 的输入内容，`bash` 会把用户的输入放在一个名为 `REPLY` 的变量中。使用 `read` 命令的选项 “-p” 可以显示提示信息（替代 `echo` 命令）并读入用户的输入，用户的输入将被保存到变量 `PEPLY` 中。例如，下面程序中没有专门指定保存用户输入的变量，程序代码如下所示（代码 `read2`）。

```
1 #!/bin/bash
2 #filename:read2
3 read -p "请输入用户名: "
4 echo "您的输入是: $REPLY"
```

其中，代码第 3 行提示用户输入用户名，选项 “-p” 用于显示提示信息，且用户的输入与提示信息出现在同一行上。代码第 4 行显示输出变量 `REPLY` 的值。该程序的执行过程如下所示。

```
#chmod u+x read2 //赋予文件 read1 执行权限
#./read2 //执行程序 read1
请输入用户名: 张小二
您的输入是: 张小二
```



如果 `read` 能够成功地读入输入的内容，将返回退出状态码 0。如果 `read` 读到 EOF，就返回一个非零值。

16.3.2 直接给变量赋值

在 Shell 程序中，在定义变量的同时可以直接给变量赋值。为变量赋值时，变量名前不应加符号美元符号“\$”，且等号前后不可有空格。

例如，下面程序 myresume 中，定义了变量 NAME、GENDER、AGE、TEL，并依次赋值“Sullivan”、“male”、“35”、“01066337823”，最后通过 echo 命令显示输出变量的值，代码如下所示（代码 myresume）。

```
1 #!/bin/bash
2 #filename:myresume
3 NAME=Sullivan
4 GENDER=male
5 AGE=35
6 TEL=01066337823
7 echo "-----Resume-----"
8 echo "name:$NAME"
9 echo "gender:$GENDER"
10 echo "age: $AGE"
11 echo "telephone:$TEL"
12 echo "-----"
```

其中，代码第 3~6 行依次对变量 NAME、GENDER、AGE、TEL 直接赋值。执行程序 myresume，显示结果如下所示。

```
# ./myresume
-----Resume-----
name:Sullivan
gender:male
age: 35
telephone:01066337823
-----
```

16.3.3 使用命令行参数赋值

Shell 提供了位置变量，因此用户可以通过使用命令行参数对位置变量赋值。当命令行上的参数多于 9 个时，可以使用 shift 命令移动位置参数。

例如，下面程序 cmdarg 将命令行参数打印输出，代码如下所示（代码 cmdarg）。

```
1 #!/bin/bash
2 #filename:cmdarg
3 echo "-----"
4 echo "Program name: $0"
5 echo "Your first argument: $1"
6 echo "Your second argument: $2"
7 echo "Your third argument: $3"
8 echo "Your forth argument:$4"
9 echo "-----"
```

其中，代码第 4~8 行分别实现了对位置参数的输出。该代码执行过程如下所示。

```
#chmod u+x cmdarg
# ./cmdarg 111 222 333 444
-----
```



```
Program name: ./cmdarg           //显示程序名
Your first argument: 111         //显示第 1 个位置参数的值
Your second argument: 222       //显示第 2 个位置参数的值
Your third argument: 333        //显示第 3 个位置参数的值
Your forth argument:444         //显示第 4 个位置参数的值
-----
```

16.3.4 利用命令的输出结果赋值

在 Shell 程序中，可以将一个可执行命令的执行结果赋值给一个变量。这需要在赋值语句中使用反引号（“`”）。反引号表示引用命令的执行结果。

例如在下面程序 `cmdresult` 中，通过反引号将命令 `pwd` 及 `find` 的执行结果赋值给变量 `CURRDIR` 和 `RESULT`，程序代码如下所示（代码 `cmdresult`）。

```
1 #!/bin/bash
2 #filename:cmdresult
3 CURRDIR=`pwd`
4 RESULT=`find /etc -name rc.local`
5 echo "-----"
6 echo "Current Directory: $CURRDIR"
7 echo "Search Result:$RESULT"
8 echo "-----"
```

其中，代码第 3 行将命令 `pwd` 的运行结果赋值给变量 `CURRDIR`，代码第 4 行将“`find /etc -name rc.local`”的执行结果赋值给变量 `RESULT`，该命令实现在 `/etc` 目录中查找名称为 `rc.local` 的文件。代码第 6 行显示输出变量 `CURRDIR` 的值，代码第 7 行显示输出变量 `RESULT` 的值。该代码的执行结果如下所示。

```
#chmod u+x chmdresult
# ./cmdresult
-----
Current Directory: /prog
Search Result:/etc/rc.d/rc.local
/etc/rc.local
-----
```

16.3.5 从文件读入数据实现赋值

在 Shell 程序中，可以从一个指定文件读入数据并赋值给变量。为了方便地处理数据，通常通过 `while` 循环一行行读入数据，即每循环一次将文件中的一行读入变量，直到读取到文件的结尾。

例如，下面程序 `readfile` 从文件 `execfile` 中读取文件名，通过 `chmod` 命令为每一文件赋予可执行权限，程序代码如下所示（代码 `readfile`）。

```
1 #!/bin/bash
2 #filename:readfile
3 ls *.exe >execfile
4 while read LINE
5 do
6   chmod +x $LINE
7 done<execfile
```

其中，第 4~7 行是 `while...do...done` 结构，第 7 行的“`done<execfile`”表示 `while` 循环从文件 `execfile` 读入数据，“`read LINE`”表示每次循环读入一行数据并赋值给变量 `LINE`。该

代码执行过程如下所示。

```
# ll *.exe //显示当前目录下所有扩展名为.exe 的文件列表及权限
-rw-r--r-- 1 root root 5 8月16 23:28 bubble.exe
-rw-r--r-- 1 root root 33 8月16 23:29 homelist.exe
-rw-r--r-- 1 root root 20 8月16 23:28 sort.exe
-rw-rw-rw- 1 root root 278 8月16 19:23 student.exe
#chmod +x readfile //为文件 readfile 赋予可执行权限
#./readfile //执行程序 readfile
# cat execfile //显示文件 execfile 的内容, 其中包含了所有扩展名
为.exe 的文件列表
bubble.exe
homelist.exe
sort.exe
student.exe
# ll *.exe //显示所有扩展名为.exe 的文件列表及权限
-rwxr-xr-x 1 root root 5 8月16 23:28 bubble.exe
-rwxr-xr-x 1 root root 33 8月16 23:29 homelist.exe
-rwxr-xr-x 1 root root 20 8月16 23:28 sort.exe
-rwxrwxrwx 1 root root 278 8月16 19:23 student.exe
```

16.4 变量的输出

输出变量最简单的方法是使用 `echo` 命令。此外, `printf` 命令可以用来实现格式化输出, 其作用是打印输出格式化的字符串, 效果类似于 C 语言的 `printf` 函数。

16.4.1 echo 命令: 直接输出变量的值

输出变量最简单的方法是使用命令 `echo`, 该命令可以向终端输出数据。该命令输出变量的语法格式如下所示。

```
echo $name1 [$name2 ...]
```

其中, `$name1`、`$name2...` 表示输出的变量, 引用变量时需要在变量名前添加 “\$” 符号, 多个变量可以一并输出。

例如, 从标准输入读入变量 `VAR1` 和 `VAR2` 的值, 通过 `echo` 命令显示输出变量的值, 命令如下所示。

```
# read VAR1 VAR2
aaa bbb //输入 aaa、bbb
# echo $VAR1 //显示输出变量 VAR1 的值
aaa
# echo $VAR2 //显示输出变量 VAR2 的值
bbb
# echo $VAR1 $VAR2 //显示输出变量 VAR1、VAR2 的值
aaa bbb
```

16.4.2 printf 命令: 格式化输出变量

`printf` 命令可以用来格式化输出, 其作用是打印输出格式化的字符串, 与 C 语言的 `printf` 函数非常类似。 `printf` 命令需要指定打印输出的字符串, 以及描述打印效果的格式说明符。格



式说明符通常是“%”后跟一个字符，例如，“%f”表示按浮点数输出，“%d”表示按整数输出。一些常用的格式说明符如表 16-2 所示。

表 16-2 printf 命令的格式说明符

格式说明符	说 明
\"	输出双引号
\oNNN	按八进制输出，N 表示 0~3 之间的数字
\\	输出反斜杠
\a	响铃
\b	退格
\c	不再做后面的输出
\f	填表
\n	换行
\r	回车
\t	Tab，水平制表符
\v	重直制表符
\xNNN	按十六进制输出，N 表示 1~3 之间的数字
%%	输出%
%b	对带有“\”的字符串进行转义



与 C 语言不同，Linux 中的 printf 命令的参数之间不需要用逗号分隔。

例如，显示输出命令 printf 的类型，如下所示。

```
# type printf
printf is a shell builtin           //printf 命令是一个内部命令
```

例如，将数字 59 作为一个小数点后面保留 4 位小数的浮点数打印输出，其中格式说明符“%.4f”表示小数点后保留 4 位。执行命令如下所示。

```
# printf "The number is %.4f \n" 59
The number is 59.0000
```

例如，输出字符串“Zhangsan”、“Wangwu”和数字 2800.6，其中“Zhangsan”、“Wangwu”按左对齐 20 个字符宽度输出，数字 2800.6 按保留 4 位小数 20 个字符宽度输出。执行命令如下所示。

```
# printf "%-20s%-15s%20.4f" "Zhangsan" "Wangwu" 2800.6
Zhangsan           Wangwu           2800.6000
#
```

其中，“%-20s”表示左对齐，20 个字符宽度；“%20.4f”表示右对齐，4 位小数，20 个字符宽。

例如，将上例中的“Zhangsan”、“Wangwu”按右对齐 20 个字符宽度输出，数字 2800.6 按左对齐输出，执行命令如下所示。


```
# printf "%20s%15s%-20.4f" "Zhangsan" "Wangwu" 2800.6
Zhangsan      Wangwu2800.6000
#
```

例如，使用 printf 命令格式化输出字符串“Zhangsan”和数学表达式的值，执行命令如下所示。

```
# printf "%s's average score is %.2f\n" "Zhangsan" $(( (80+99+66)/3 ))
Zhangsan's average score is 81.0
#
```

其中，“%s”表示按字符串进行输出，“%.2f”表示保留两位小数的浮点数，“\$(((80+99+66)/3))”表示计算 80、99、66 的平均值。

16.5 数组变量

一个变量可以理解为包含单个值的容器，而数组则正好相反，可以包含多个值。数组是数据值的集合，是一个可以存储一组或一系列数值的变量。由于数组可以同时保存一个或多个值，在对大量数据进行处理时，使用数组更为灵活方便。

一个数组可以具有许多个元素，每个元素都有一个值。存储在数组中的值称为数组元素，数组中的元素是在索引（也称下标）的基础上被引用和操作的。

16.5.1 数组的定义

大多数编程语言都支持数字索引。所谓“数字索引”是指数组元素的索引（下标）为整数。当需要通过相对位置来确定数组元素时（例如，通过简单地加 1 或减 1 直接计算出数组中的下一个或前一个有效索引值），使用数字索引是非常方便的。数组的索引通常是整数并以数字 0 作为起始，如图 16-1 即是一个数字索引与数组元素值对应关系的示例。

Sname[0]	Sname[1]	Sname[2]	Sname[3]	Sname[4]
“张明”	“李四”	“王小五”	“赵勤”	“李小平”

图 16-1 数组示例

其中数组名为 name，包括数组元素：“张明”、“李四”、“王小五”、“赵勤”和“李小平”。可以通过索引（下标）0 来引用“张明”，通过索引 1 来引用“李四”，通过索引 2 来引用“王小五”……依此类推。要引用数组中的元素，需要使用一对中括号（[]），例如，以下语句可以显示索引 2 的数组元素值。

```
echo ${name[2]}
```

其中的大括号“{}”确保在解析时，将“name[2]”视为一个变量名。

下面的格式声明了一个数组并为其赋值。

```
name=(element1 element2...)
```

例如，以下程序中创建了两个数组：name 和 score。name 数组中保存了学生名单，而 score 数组中保存了对应的学生成绩。通过 for 循环，显示学生成绩列表，具体代码如下所示（代码 scoreevaluate）。

```
1 #!/bin/bash //程序第 1 行的“#!/bin/bash”表示调用 bash 来执行脚本程序
2 #filename:scoreevaluate //程序第 2 行的“#filename: scoreevaluate”表示该程序的名称为
```



```
scoreevaluate
```

```
3 name=("张三" "李四" "王小二" "李小明" "张四宝") //定义数组 name。数组元素 0 为“张三”，
数组元素 1
//为“李四”，数组元素 2 为“王小二”，数组元素 3
//为“李小明”，数组元素 4 为“张四宝”
4 score=(65 56 67 99 89) //定义数组 score。数组元素 0 为“65”，数组元素 1 为“56”，数组元素
2 为“67”，
//数组元素 3 为“99”，数组元素 4 为“89”
5 for i in 0 1 2 3 4 //第 5~8 行为 for 循环。循环变量为 i，i 每次循环依次取值 0，1，2，3，4
6 do // for 循环中的 do 语句
7 echo "${name[$i]} 的考试成绩为 :${score[$i]} " //显示输出 "${name[$i]} 的考试成绩
为 :${score[$i]}”。
//${name[$i]} 引用 name 数组中下标为 $i 的元素，
//${score[$i]} 引用 score 数组中下标为 $i 的元素
8 done // for 循环中的 done 语句
```

执行该程序的过程如下所示。

```
#chmod u+x scoreevaluate //赋予文件 scoreevaluate 执行权限
#./scoreevaluate //执行程序 scoreevaluate
张三的考试成绩为 :65
李四的考试成绩为 :56
王小二的考试成绩为 :67
李小明的考试成绩为 :99
张四宝的考试成绩为 :89
```

例如，下面程序中创建一个名为 `name` 的数组，其中保存了人员名单，通过 `while` 循环将数组中的名单逐一输出，代码如下所示（代码 `numarray`）。

```
1 #!/bin/bash //程序第 1 行的“#!/bin/bash”表示调用 bash 来执行脚本程序
2 #filename:numarray //程序第 2 行的“#filename:numarray”表示该程序的名称为 numarray
3 name[0]="张明" //数组 name 的第 0 个元素赋值“张明”
4 name[1]="李四" //数组 name 的第 1 个元素赋值“李四”
5 name[2]="王小五" //数组 name 的第 2 个元素赋值“王小五”
6 name[3]="赵勤" //数组 name 的第 3 个元素赋值“赵勤”
7 name[4]="李小平" //数组 name 的第 4 个元素赋值“李小平”
8 echo "---人员名单---" //显示输出“---人员名单---”
9 i=0 //变量 i 赋值 0
10 while [ $i -lt 5 ] //第 10~14 行是 while 循环，循环控制条件为“$i -lt 5”，
//即变量 i 小于 5 则继续执行循环，否则退出
11 do // while 循环结构中的 do 语句
12 echo ${name[$i]} //显示输出数组变量 name[$i] 的值
13 i=$((i+1)) //变量 i 自加 1
14 done // while 循环结构中的 done 语句，表示循环体结束
```

该代码的执行结果如下所示。

```
# . numarray
---人员名单---
张明
李四
王小五
赵勤
李小平
```

16.5.2 数组的复制

数组还有两个特殊的索引——“*”和“@”，其作用都是提取出整个数组元素，但是当

它们加上双引号使用时，略有不同。符号“@”的含义是把原数组的内容复制到一个新数组中，生成的新数组和原来的是一样的；符号“*”是把原数组中的所有元素（除了用于区别元素的分隔符，通常是空格）当成一个元素复制到新数组中，生成的新数组只有一个元素。

例如，下面示例中将数组 `name` 原样复制到一个新数组 `list` 中，并将数组 `name` 中的全部元素当成一个元素复制到新数组 `all` 中，命令如下所示。

```
# declare -a                                //使用 declare 命令显示当前的所有数组
declare -ar BASH_VERSION='([0]="2" [1]="05b" [2]="0" [3]="1" [4]="release"
[5]="i386-redhat-linux-gnu")'
declare -a DIRSTACK='()'
declare -a GROUPS='()'
declare -a PIPESTATUS='([0]="0")'
declare -a name='([0]="张三" [1]="李四" [2]="王小二" [3]="李小明" [4]="张四宝")'
//数组 name
declare -a score='([0]="65" [1]="56" [2]="67" [3]="99" [4]="89")'
# all =("${name[*]}")                        //将数组 name 的全部内容作为一个元素复制到数组 all 中
# list =("${name[@]}")                      //将数组 name 的内容复制到一个新数组 list 中
# declare -a                                //使用 declare 命令显示当前的所有数组
declare -ar BASH_VERSION='([0]="2" [1]="05b" [2]="0" [3]="1" [4]="release"
[5]="i386-redhat-linux-gnu")'
declare -a DIRSTACK='()'
declare -a GROUPS='()'
declare -a PIPESTATUS='([0]="0")'
declare -a list = '([0]="张三" [1]="李四" [2]="王小二" [3]="李小明" [4]="张四宝")'
//创建的新数组 list
declare -a all = '([0]="张三 李四 王小二 李小明 张四宝")' //创建的新数组 all
declare -a name='([0]="张三" [1]="李四" [2]="王小二" [3]="李小明" [4]="张四宝")'
declare -a score='([0]="65" [1]="56" [2]="67" [3]="99" [4]="89")'
```

16.5.3 数组元素的个数

在对数组元素进行输出时，通常需要预先知道数组元素的个数。使用下面的格式可以返回数组中元素的个数，如下所示。

```
${#name[*]}
```

其中 `name` 为数组名称。例如，查看数组 `name` 和 `score` 的元素个数如下所示。

```
# declare -a                                //使用 declare 命令显示当前的所有数组
declare -ar BASH_VERSION='([0]="2" [1]="05b" [2]="0" [3]="1" [4]="release"
[5]="i386-redhat-linux-gnu")'
declare -a DIRSTACK='()'
declare -a GROUPS='()'
declare -a PIPESTATUS='([0]="0")'
declare -a name='([0]="张三" [1]="李四" [2]="王小二" [3]="李小明" [4]="张四宝")'
//数组 name
declare -a score='([0]="65" [1]="56" [2]="67" [3]="99" [4]="89")'
//数组 score
# echo ${#name[*]}                          //查看数组 name 的元素个数
5
# echo ${#score[*]}                         //查看数组 score 的元素个数
5
```

例如，通过“`${#name[*]}`”计算数组元素的个数，然后使用 `while` 循环结构输出数组 `name` 和 `score` 中的元素，程序如下所示（代码 `arraynum`）。



```

1 #!/bin/bash //程序第1行的“#!/bin/bash”表示调用bash来执行脚本程序
2 #filename:arraynum //程序第2行的“#filename: arraynum”表示该程序的名称为arraynum
3 name=("张三" "李四" "王小二" "李小明" "张四宝") //定义数组name。数组元素0为“张三”，
数组元素1
//为“李四”，数组元素2为“王小二”，数组元素3为
//“李小明”，数组元素4为“张四宝”
4 score=(65 56 67 99 89) //定义数组score。数组元素0为“65”，数组元素1为“56”，数组元
//素2为“67”，数组元素3为“99”，数组元素4为“89”
5 N=${#name[*]} //计算数组name中包含的元素个数，保存到变量N中
6 echo "----学生数: $N ----" //显示输出“----学生数: $N ----”
7 i=0 //变量i赋值0
8 while [ $i -lt $N ] //第8~12行为while循环结构，控制条件为“$i -lt $N”，即变量i
//小于变量N时继续执行循环，否则退出循环
9 do // while 循环结构中的do语句
10 echo "${name[$i]}的考试成绩为 :${score[$i]} " //显示输出“${name[$i]}的考试成绩
为 :${score[$i]}”
11 i=$((i+1)) //变量i加1
12 done // while 循环结构中的done语句，表示循环体结束

```

该代码的执行结果如下所示。

```

#chmod u+x arraynum
# . arraynum
----学生数: 5 ----
张三的考试成绩为 :65
李四的考试成绩为 :56
王小二的考试成绩为 :67
李小明的考试成绩为 :99
张四宝的考试成绩为 :89

```

16.5.4 数组元素的长度

使用下面的格式可以返回数组中指定元素的长度，如下所示。

```
${#name[ num]}
```

其中，name 为数组名称，num 为待查询的数组元素的下标。例如，查看数组 name 中每个元素的字符长度，具体代码如下所示（代码 arrayelem）。

```

1 #!/bin/bash //程序第1行的“#!/bin/bash”表示调用bash来执行脚本程序
2 #filename:arrayelem //程序第2行的“#filename: arrayelem”表示该程序的名称为
arrayelem
3 name=( Zhangsan Lisi Wangxiaoer Liliangjie Zhourunfa) //定义数组score。数组元素0为
“Zhangsan”，
//数组元素1为“Lisi”，数组元素2为
//“Wangxiaoer”，数组元素3为“Liliangjie”，
//数组元素4为“Zhourunfa”
4 N=${#name[*]} //通过“${#name[*]}”计算数组name的元素个数，且赋值给变量N
5 echo "----数组元素长度统计----" //显示输出“----数组元素长度统计----”
6 i=0 //变量i赋值0
7 while [ $i -lt $N ] //第7~11行为while...do...done循环结构。循环控制条件为
//“$i -lt $N”，即变量i小于变量N时继续执行循环，否则
退出循环
8 do // while 循环中do语句
9 echo "第 $i 个元素的长度为 :${#name[$i]} " //显示输出“第 $i 个元素的长度
为 :${#name[$i]}”
10 i=$((i+1)) //变量i加1
11 done // while 循环中done语句

```


该程序的执行结果如下所示。

```
#chmod u+x arrayelem
#./arrayelem
----数组元素长度统计----
第 0 个元素的长度为 :8
第 1 个元素的长度为 :4
第 2 个元素的长度为 :10
第 3 个元素的长度为 :10
第 4 个元素的长度为 :9
```

16.6 算术运算

Bourne Shell 没有内置的算术运算，不能直接做加、减、乘、除等算术运算，如下所示。

```
$X=10
$echo $X
10
$X=$X+1
$echo $X
10+1
```

可以看到，在输出变量 X 的值时，输出结果是“10+1”，未能输出希望看到的值“11”。如果需要完成简单的整数算术运算，可以使用 `expr` 命令或 `let` 命令。对于浮点算法运算，可以使用命令 `awk` 或 `bc`。

16.6.1 expr 命令

`expr` 命令是一个表达式处理命令。当计算算术运算表达式时，它可以用来执行简单的整数操作，有 5 种算术运算符可以使用：`+`、`-`、`\*`、`/`、`%`。在使用运算符时，运算符前后需要留有空白，且只能进行整数运算。

例如，以下示例中使用 `expr` 命令完成了简单的算术运算。

```
$ a=5; b=16 //变量 a 赋值 5，变量 b 赋值 16
$ x='expr $b / $a' //变量 b 整除变量 a
$ y='expr $b % $a' //变量 b 与变量 a 取模
$ echo x=$x y=$y //显示输出变量 x 与 y 的值
x=3 y=1
$ a='expr $a + 1' //变量 a 加 1
$ echo $a //显示变量 a 的值
6
```

16.6.2 let 命令

`let` 命令与 `expr` 可以互换使用。`let` 语句中不需要在变量前面加美元符号，但必须将单个或带有空格的表达式用双引号引起来。

例如，以下示例中使用 `let` 命令完成了简单的算术运算。

```
$ x=100 //变量 x 赋值 100
$ let x=x+1 //变量 x 加 1
$ echo $x //显示输出变量 x 的值
101
```



```
$ let y=x-1           //变量 y 等于变量 x 的值减 1
$ echo $y             //显示输出变量 y 的值
100
$ let y=x*2           //变量 x 乘以 2，并赋值给变量 y
$ echo $y             //显示输出变量 y 的值
202
$ let y=x/2           //变量 x 除以 2，并赋值给变量 y
$ echo $y             //显示输出变量 y 的值
50
```

16.6.3 浮点算术运算

如果需要完成更为复杂的计算，可以使用 `bc`、`awk` 等命令。例如，使用 `bc` 命令计算 192 除以 2 的结果，如下所示。

```
# n='echo " scale=3; 193 / 2 " | bc '
# echo $n
96.500
```

可以看到，使用 `echo` 命令将需要计算的数据及参数从管道“|”输出给 `bc` 命令，其中“`scale=3`”表示设置小数点右边的精确位数为 3，即保留 3 位小数。整行命令“`echo " scale=3; 193 / 2 " | bc`”被放在反引号里，表示返回该行命令的执行结果，且保存到变量 `n` 中。通过执行命令“`echo $n`”，显示输出变量 `n` 的值为 96.500。

例如，使用 `awk` 命令计算 3.3 乘以 4 以及 3.3 除以 4 的结果，如下所示。

```
# product='awk 'BEGIN {x=3.3;y=4 ; printf "%.2f %.4f" ,x*y,x/y;}'
# echo $product
13.20 0.8250
```

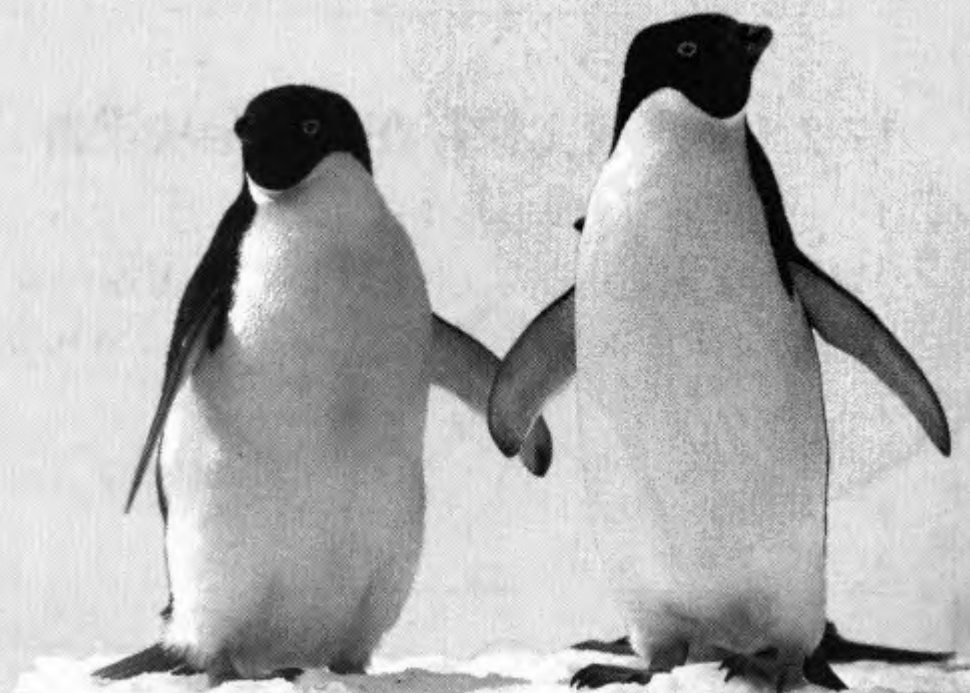
其中使用了 `awk` 命令。变量 `x` 赋值 3.3，变量 `y` 赋值 4，“`printf "%.2f %.4f" ,x*y,x/y`”表示按浮点数输出变量 `x` 与变量 `y` 的乘积且保留两位小数，输出变量 `x` 与变量 `y` 的商且保留 4 位小数。整行命令“`awk 'BEGIN {x=3.3;y=4 ; printf "%.2f %.4f" ,x*y,x/y;}'`”被放在反引号里，表示返回该行命令的执行结果，且保存到变量 `product` 中。通过执行命令“`echo $product`”，显示输出变量 `product` 的值为“13.20 0.8250”，即，3.3 乘以 4 的值为 13.20，3.3 除以 4 的值为 0.8250。

UNIX

第17章

条件测试

Shell 脚本中提供了对数据、字符和文件的条件测试，测试的结果或者为 0 或者非 0。若值为 0，表示条件为真；若值为非 0，表示条件为假。条件测试通常配合控制结构使用，可以根据不同的测试结果来控制程序的执行。





17.1 Shell 的测试命令

test 命令是 Shell 编程中条件判断最常用的测试命令，一般总是与 if、while、until 等语句一起使用。test 命令的语法格式如下所示。

```
test 表达式
test 命令也可由方括号取代，如下所示。
[ 表达式 ]
其中，在 “[” 之后及 “]” 之前要留有空格。
```

17.2 测试文件属性

文件属性测试表达式用于测试文件的属性状态，包括文件是否可写入、可执行，文件是否存在，是否是目录，等等。常用的文件测试表达式格式如表 17-1 所示。

表 17-1 文件属性测试表达式

表 达 式	说 明
-b file	如果文件 file 存在且为块设备（Block special），则值为真
-c file	如果文件 file 存在且为字符设备（Character special），则值为真
-r file	如果文件 file 存在且为只读，则值为真
-w file	如果文件 file 存在且是可写入，则值为真
-x file	如果文件 file 存在且是可执行，则值为真
-s file	如果文件 file 存在且长度大于零，则值为真
-d file	如果 file 是一个目录，则值为真
-f file	如果文件 file 是一个普通文件，则值为真
-e file	如果文件存在，则值为真

17.2.1 测试文件存在并且是块设备

测试文件存在并且是块设备，使用符号 “-b”。例如，编写 Shell 程序 fileb，测试变量指定的文件是否是块设备文件，如果是，则复制到/dev 目录下；否则，显示 “\$FILENAME is not a block device file”。测试表达式为 “test -b \$FILENAME”。该程序的具体代码如下所示（代码 fileb）。

```
1 #!/bin/bash
2 #filename:fileb
```



```

3 echo "Please enter the file name:"
4 read FILENAME
5 if test -b $FILENAME ; then
6 cp $FILENAME /dev
7 else
8 echo "$FILENAME is not a block device file"
9 fi

```

其中，代码第 4 行读入指定的文件名，且保存到变量 FILENAME 中。代码第 5~9 行是 if...then...else...fi 结构，其中“-b”对文件是否是块设备进行检测。如果是块设备，执行第 6 行进行复制，否则执行第 8 行，显示提示信息。该程序执行过程如下所示。

```

#chmod u+x fileb           //赋予文件 fileb 可执行权限
#./fileb                   //执行程序 fileb
Please enter the file name:
rc.local                   //输入“rc.local”
rc.local is not a block device file //rc.local 不是块设备

```

17.2.2 测试文件存在并且是字符设备

测试文件存在并且是字符设备使用符号“-c”。例如，编写 Shell 程序 filec，测试变量指定的文件是否是字符设备文件，如果是，以长格式显示该文件的信息；否则，显示“\$FILENAME is not a character device file”。测试表达式为“test -c \$FILENAME”。该程序的具体代码如下所示（代码 filec）。

```

1 #!/bin/bash
2 #filename:filec
3 echo "Please enter the file name:"
4 read FILENAME
5 if test -c $FILENAME ; then
6 ls -l $FILENAME
7 else
8 echo "$FILENAME is not a character device file"
9 fi

```

其中，代码第 4 行读入指定的文件名，且保存到变量 FILENAME 中。代码第 5~9 行是 if...then...else...fi 结构，其中“-c”对文件是否是字符设备进行检测。如果是字符设备则执行第 6 行，显示该字符设备的详细信息；否则执行第 8 行，显示不是字符设备的提示信息。该程序的执行过程如下所示。

```

#chmod u+x filec           //赋予文件 filec 可执行权限
#./filec                   //执行程序 filec
Please enter the file name:
/dev/ttys2                 //输入“/dev/ttys2”
crw-rw-rw-  1 root    tty      3,  50 2003-01-30 /dev/ttys2
#./filec                   //执行程序 filec
Please enter the file name:
rc.local                   //输入“rc.local”
rc.local is not a character device file //rc.local 不是字符设备

```

17.2.3 测试文件存在并且是目录

测试文件存在并且是目录使用符号“-d”。例如，编写 Shell 程序 filed，测试变量指定的文件是否是字符设备文件。如果是目录；则将当前目录切换到该目录；否则，显示“\$FILENAME



is not directory”。测试表达式为“test -d \$FILENAME”。该程序的具体代码如下所示（代码 filed）。

```
1 #!/bin/bash
2 #filename:filed
3 echo "Please enter the file name:"
4 read FILENAME
5 if test -d $FILENAME ; then
6 cd $FILENAME
7 pwd
8 else
9 echo "$FILENAME is not a directory file"
10 fi
```

其中，代码第 4 行读入指定的文件名，且保存到变量 FILENAME 中。代码第 5~10 行是 if...then...else...fi 结构，其中“-d”对文件是否是目录进行检测。如果是目录则执行第 6、7 行，切换到该目录，且执行 pwd 命令显示当前目录。如果不是目录，则执行第 9 行，显示提示信息。该程序执行的过程如下所示。

```
#chmod u+x filed           //赋予文件 filed 可执行权限
#./filed                   //执行程序 filed
Please enter the file name:
rc.local                   //输入“rc.local”
rc.local is not a directory file //rc.local 不是目录
#./filed                   //执行程序 filed
Please enter the file name:
/root                      //输入“/root”
/root                      //pwd 命令执行结果，显示当前目录
```

17.2.4 测试文件存在

测试文件是否存在使用符号“-e”。例如，编写 Shell 程序 filee，测试变量指定的文件是否存在。如果存在，则显示该文件的详细信息，否则创建该文件。测试表达式为“test -e \$FILENAME”，该程序的具体代码如下所示（代码 filee）。

```
1 #!/bin/bash
2 #filename:filee
3 echo "Please enter the file name:"
4 read FILENAME
5 if test -e $FILENAME ; then
6 ls -l $FILENAME
7 else
8 touch $FILENAME
9 fi
```

其中，代码第 4 行读入指定的文件名，且保存到变量 FILENAME 中。代码第 5~9 行是 if...then...else...fi 结构，其中“-e”对文件是否存在进行检测。如果存在则执行第 6 行，显示该文件的详细信息，否则执行第 8 行，通过 touch 命令创建该文件。该程序的执行过程如下所示。

```
#chmod u+x filee           //赋予文件 filee 可执行权限
#./filee                   //执行程序 filee
Please enter the file name:
rc.local                   //输入“rc.local”
-rw-r--r-- 1 root root 220 8月17 09:10 rc.local
```



```
# ls homework.doc           //使用 ls 命令查看当前目录下是否有 homework.doc 文件
ls: homework.doc: 没有那个文件或目录
#./filee                    //执行程序 filee
Please enter the file name:
homework.doc                //输入“homework.doc”，创建该文件
#./filee                    //执行程序 filee
Please enter the file name:
homework.doc                //输入“homework.doc”
-rw-r--r--  1 root    root          0  8月17 09:12 homework.doc
```

17.2.5 测试文件存在并且可执行

测试文件是否存在并且可执行使用符号“-x”。例如，编写 Shell 程序 filex，测试变量指定的文件是否存在且为可执行文件。如果存在且是可执行文件，则执行该文件，否则通过 chmod 命令赋予该文件可执行权限。测试表达式为“test -x \$FILENAME”，该程序的具体代码如下所示（代码 filex）。

```
1 #!/bin/bash
2 #filename:filex
3 echo "Please enter the file name:"
4 read FILENAME
5 if test -x $FILENAME ; then
6 sh $FILENAME
7 else
8 chmod +x $FILENAME
9 ls -l $FILENAME
10 fi
```

其中，代码第 4 行读入指定的文件名，且保存到变量 FILENAME 中。代码第 5~11 行是 if...then...else...fi 结构，其中“-x”对文件是否是可执行文件进行检测。如果是可执行文件则执行第 6 行，执行该程序文件，否则执行第 8、9 行，通过 chmod 命令为该文件赋予执行权限，并通过 ls 命令显示该文件的属性信息。该程序的执行过程如下所示。

```
# ls -l myhello              //查看文件 myhello 的属性
-rw-r--r--  1 root    root          36  8月21 09:41 myhello      //无可执行权限
# cat myhello                //查看文件 myhello 的内容
echo "这是 echo 命令的测试输出!!!"
#./filex                     //执行程序 filex
Please enter the file name:
myhello                      //输入文件名“myhello”
-rwxr-xr-x  1 root    root          36  8月21 09:41 myhello      //已赋予可执行权限
#./filex                     //执行程序 filex
Please enter the file name:
myhello                      //输入文件名“myhello”
这是 echo 命令的测试输出!!!  //程序 myhello 被执行
```

17.3 测试数值

数值的测试包括相等测试、不相等测试、大于测试、大于等于测试、小于测试和小于等于测试。常用的数值测试表达式如表 17-2 所示。



表 17-2 数值测试表达式

表 达 式	说 明
n1 -eq n2	n1 等于 n2，则值为真
n1 -ne n2	n1 不等于 n2，则值为真
n1 -gt n2	n1 大于 n2，则值为真
n1 -lt n2	n1 小于 n2，则值为真
n1 -ge n2	n1 大于等于 n2，则值为真
n1 -le n2	n1 小于等于 n2，则值为真

17.3.1 相等测试

数值的相等测试使用符号“-eq”。例如，编写 Shell 程序 numberequal，测试变量 N1 和 N2 的值是否相等。如果相等，显示“Equal”，否则显示“Not Equal”。测试表达式为“test \$N1 -eq \$N2”，该程序的具体代码如下所示（代码 numberequal）。

```
1 #!/bin/bash
2 #filename:numberequal
3 echo "Please enter the first number:"
4 read N1
5 echo "Please enter the second number:"
6 read N2
7 if test $N1 -eq $N2 ; then
8 echo "Equal"
9 else
10 echo "Not Equal"
11 fi
```

其中，代码第 4 行、第 6 行读入两个数值，且保存到变量 N1 和 N2 中。代码第 7 行，使用“-eq”对变量 N1 与 N2 所表示的两个数值是否相等进行判断。该程序的执行过程如下所示。

```
#chmod +x numberequal //赋予文件 numberequal 执行权限
#./numberequal //执行程序 numberequal
Please enter the first number:
8970 //输入“8970”
Please enter the second number:
8970 //输入“8970”
Equal
#./numberequal //执行程序 numberequal
Please enter the first number:
8970 //输入“8970”
Please enter the second number:
234 //输入“234”
Not Equal
```

17.3.2 不相等测试

数值的不相等测试使用符号“-ne”。例如，编写 Shell 程序 numbernoequal，测试变量 N1 和 N2 的值是否相等。如果相等，显示“Equal”，否则显示“Not equal”。测试表达式为“test \$N1 -ne \$N2”，该程序的具体代码如下所示（代码 numbernoequal）。


```

1 #!/bin/bash
2 #filename:numeroequal
3 echo "Please enter the first number:"
4 read N1
5 echo "Please enter the second number:"
6 read N2
7 if test $N1 -ne $N2 ; then
8 echo "Not equal"
9 else
10 echo "Equal"
11 fi

```

其中，代码第 4 行、第 6 行读入两个数值，且保存到变量 N1 和 N2 中。代码第 7 行，使用“-ne”对变量 N1 与 N2 所表示的两个数值是否相等进行判断。该程序的执行过程如下所示。

```

#chmod +x numeroequal           //赋予文件 numeroequal 执行权限
#./numeroequal                  //执行程序 numeroequal
#./numeroequal
Please enter the first number:
45                               //输入“45”
Please enter the second number:
67                               //输入“67”
Not equal
#./numeroequal                  //执行程序 numeroequal
Please enter the first number:
45                               //输入“45”
Please enter the second number:
45                               //输入“45”
Equal

```

17.3.3 大于测试

数值的大于测试使用符号“-gt”。例如，编写 Shell 程序 numbergt，测试变量 N1 是否大于 N2。如果大于，显示“\$N1 great than \$N2”，否则显示“\$N1 not great than \$N2”。测试表达式为“test \$N1 -gt \$N2”，该程序的具体代码如下所示（代码 numbergt）。

```

1 #!/bin/bash
2 #filename:numbergt
3 echo "Please enter the first number:"
4 read N1
5 echo "Please enter the second number:"
6 read N2
7 if test $N1 -gt $N2 ; then
8 echo "$N1 great than $N2"
9 else
10 echo "$N1 not great than $N2"
11 fi

```

其中，代码第 4 行、第 6 行读入两个数值，且保存到变量 N1 和 N2 中。代码第 7 行，使用“-gt”对变量 N1 是否大于 N2 进行判断。该程序的执行过程如下所示。

```

#chmod +x numbergt             //赋予文件 numbergt1 执行权限
#./numbergt                    //执行程序 numbergt
Please enter the first number:
67                             //输入“67”

```



```
Please enter the second number:
24                                //输入“24”
67 great than 24
#./numbergt                      //执行程序 numbergt
Please enter the first number:
56                                //输入“56”
Please enter the second number:
88                                //输入“88”
56 not great than 88
```

17.3.4 大于等于测试

数值的大于等于测试使用符号“-ge”。例如，编写 Shell 程序 numberge，测试变量 N1 是否大于等于 N2。如果大于等于，则显示“\$N1 great and equal than \$N2”，否则显示“\$N1 not great and equal than \$N2”。测试表达式为“test \$N1 -ge \$N2”，该程序的具体代码如下所示（代码 numberge）。

```
1 #!/bin/bash
2 #filename:numberge
3 echo "Please enter the first number:"
4 read N1
5 echo "Please enter the second number:"
6 read N2
7 if test $N1 -ge $N2 ; then
8 echo "$N1 great and equal than $N2"
9 else
10 echo "$N1 not great and equal than $N2"
11 fi
```

其中，代码第 4 行、第 6 行读入两个数值，且保存到变量 N1 和 N2 中。代码第 7 行，使用“-ge”对变量 N1 是否大于等于 N2 进行判断。该程序的执行过程如下所示。

```
#chmod +x numberge              //赋予文件 numberge 执行权限
#./numberge                     //执行程序 numberge
Please enter the first number:
45                                //输入“45”
Please enter the second number:
45                                //输入“45”
45 great and equal than 45
#./numberge                     //执行程序 numberge
Please enter the first number:
67                                //输入“67”
Please enter the second number:
34                                //输入“34”
67 great and equal than 34
#./numberge                     //执行程序 numberge
Please enter the first number:
45                                //输入“45”
Please enter the second number:
89                                //输入“89”
45 not great and equal than 89
```

17.3.5 小于测试

数值的小于测试使用符号“-lt”。例如，编写 Shell 程序 numberlt，测试变量 N1 是否小

于 N2。如果小于，则显示 “\$N1 less than \$N2”，否则显示 “\$N1 not less than \$N2”。测试表达式为 “test \$N1 -lt \$N2”，该程序的具体代码如下所示（代码 numberlt）。

```
1 #!/bin/bash
2 #filename:numberlt
3 echo "Please enter the first number:"
4 read N1
5 echo "Please enter the second number:"
6 read N2
7 if test $N1 -lt $N2 ; then
8 echo "$N1 less than $N2"
9 else
10 echo "$N1 not less than $N2"
11 fi
```

其中，代码第 4 行、第 6 行读入两个数值，且保存到变量 N1 和 N2 中。代码第 7 行，使用 “-ge” 对变量 N1 是否大于等于 N2 进行判断。该程序的执行过程如下所示。

```
#chmod +x numberlt           //赋予文件 numberlt 执行权限
#./numberlt                  //执行程序 numberlt
Please enter the first number:
23                           //输入 “23”
Please enter the second number:
45                           //输入 “45”
23 less than 45
#./numberlt                  //执行程序 numberlt
Please enter the first number:
56                           //输入 “56”
Please enter the second number:
23                           //输入 “23”
56 not less than 23
```

17.3.6 小于等于测试

数值的小于等于测试使用符号 “-le”。例如，编写 Shell 程序 numberle，测试变量 N1 是否小于等于 N2。如果小于等于，则显示 “\$N1 less and equal than \$N2”，否则显示 “\$N1 not less and equal than \$N2”。测试表达式为 “test \$N1 -le \$N2”，该程序的具体代码如下所示（代码 numberle）。

```
1 #!/bin/bash
2 #filename:numberle
3 echo "Please enter the first number:"
4 read N1
5 echo "Please enter the second number:"
6 read N2
7 if test $N1 -le $N2 ; then
8 echo "$N1 less and equal than $N2"
9 else
10 echo "$N1 not less and equal than $N2"
11 fi
```

其中，代码第 4 行、第 6 行读入两个数字，且保存到变量 N1 和 N2 中。代码第 7 行，使用 “-le” 对变量 N1 是否小于等于 N2 进行判断。该程序的执行过程如下所示。

```
#chmod +x numberle           //赋予文件 numberle 执行权限
#./numberle                  //执行程序 numberle
```



```
Please enter the first number:
45 //输入“45”
Please enter the second number:
45 //输入“45”
45 less and equal than 45
#./numberle //执行程序 numberle
Please enter the first number:
67 //输入“67”
Please enter the second number:
23 //输入“23”
67 not less and equal than 23
#./numberle //执行程序 numberle
Please enter the first number:
34 //输入“34”
Please enter the second number:
77 //输入“77”
34 less and equal than 77
```

17.4 测试字符串

字符串测试包括相等测试、不相等测试、长度为零测试、长度不为零测试以及非空测试。常用的字符串测试表达式如表 17-3 所示。

表 17-3 字符串测试表达式

表 达 式	说 明
-z s1	如果字符串 s1 的长度是零，则值为真
-n s1	如果字符串 s1 的长度不为零，则值为真
s1 = s2	如果字符串 s1 与字符串 s2 相等，则值为真
s1 != s2	如果字符串 s1 与字符串 s2 不等，则值为真
s1	如果字符串 s1 不是空串，则值为真

17.4.1 相等测试

字符串相等测试使用符号“=”。例如，编写 Shell 程序 stringequal，测试变量 S1 和 S2 是否相等。如果相等，显示“equal”，否则显示“not equal”。测试表达式为“test \$S1 = \$S2”，该程序的具体代码如下所示（代码 stringequal）。

```
1 #!/bin/bash
2 #filename:stringequal
3 echo "Please enter the first string:"
4 read S1
5 echo "Please enter the second string:"
6 read S2
7 if test $S1 = $S2 ; then
8 echo "equal"
9 else
10 echo "not equal"
11 fi
```


其中，代码第 4 行、第 6 行读入两个字符串，且保存到变量 S1 和 S2 中。代码第 7 行，使用 “=” 对变量 S1 与 S2 所表示的两个字符串是否相等进行判断。该程序的执行过程如下所示。

```
#chmod +x stringequal           //赋予文件 stringequal 执行权限
#./stringequal                  //执行程序 stringequal
Please enter the first string:
ababab                          //输入 “ababab”
Please enter the second string:
ababab                          //输入 “ababab”
equal
#./stringequal                  //执行程序 stringequal
Please enter the first string:
aaaaaa                          //输入 “aaaaaa”
Please enter the second string:
aaaaaa                          //输入 “aaaaaa”
equal
#./stringequal                  //执行程序 stringequal
Please enter the first string:
aaaaaa                          //输入 “aaaaaa”
Please enter the second string:
aaa                             //输入 “aaa”
not equal
#./stringequal
Please enter the first string:
aaaaaa                          //输入 “aaaaaa”
Please enter the second string:
bbbbbb                          //输入 “bbbbbb”
not equal
```

17.4.2 不相等测试

字符串不相等测试使用符号 “!=”。例如，编写 Shell 程序 stringnoequal，测试变量 S1 和 S2 是否相等。如果相等，显示 “equal”，否则显示 “not equal”。测试表达式为 “test \$S1 != \$S2”，该程序的具体代码如下所示（代码 stringnoequal）。

```
1 #!/bin/bash
2 #filename:stringnoequal
3 echo "Please enter the first string:"
4 read S1
5 echo "Please enter the second string:"
6 read S2
7 if test $S1 != $S2 ; then
8 echo "not equal"
9 else
10 echo "equal"
11 fi
```

其中，代码第 4 行、第 6 行读入两个字符串，且保存到变量 S1 和 S2 中。代码第 7 行，使用 “!=” 对变量 S1 与 S2 所表示的两个字符串是否相等进行判断。该程序的执行过程如下所示。

```
#chmod +x stringnoequal           //赋予文件 stringnoequal 执行权限
#./stringnoequal                  //执行程序 stringnoequal
Please enter the first string:
```



```
ababab //输入 "ababab"
Please enter the second string:
ababab //输入 "ababab"
equal
#./stringequal //执行程序 stringequal
Please enter the first string:
aaaaaa //输入 "aaaaaa"
Please enter the second string:
aaaaaa //输入 "aaaaaa"
equal
#./stringequal //执行程序 stringequal
Please enter the first string:
aaaaaa //输入 "aaaaaa"
Please enter the second string:
aaa //输入 "aaa"
not equal
#./stringnoequal
Please enter the first string:
aaaaaa //输入 "aaaaaa"
Please enter the second string:
bbbbbb //输入 "bbbbbb"
not equal
#
```

17.4.3 长度为零测试

字符串长度为零测试使用“-z”。例如，编写 Shell 程序 stringzero，测试变量 S1 是否为零。如果长度为零，显示“The length of string is 0”，否则显示“The length of string is not 0”。测试表达式为“-z \$S1”，该程序的具体代码如下所示（代码 stringzero）。

```
1 #!/bin/bash
2 #filename:stringzero
3 echo "Please enter a string:"
4 read S1
5 if test -z $S1 ;then
6 echo "The length of string is 0"
7 else
8 echo "The length of string is not 0"
9 fi
```

其中，代码第 4 行读入一个字符串，且保存到变量 S1 中。代码第 5 行，使用“-z”对变量 S1 所表示的字符串长度是否为零进行判断。该程序的执行过程如下所示。

```
#chmod +x stringzero //赋予文件 stringzero 执行权限
# ./stringzero //执行 stringzero
Please enter a string:
//直接回车

The length of string is 0
#./stringzero //执行 stringzero
Please enter a string:
abcdefghijkl //输入 "abcdefghijkl"
The length of string is not 0
```

17.4.4 长度不为零测试

字符串长度不为零测试使用“-n”。例如，编写 Shell 程序 stringnozero，测试变量 S1 是

否为零。如果长度不为零，显示“The length of string is not zero”，否则显示“The length of string is zero”。测试表达式为“-n \$S1”，该程序的具体代码如下所示（代码 stringnozero）。

```
1 #!/bin/bash
2 #filename:stringnozero
3 echo "String length test:"
4 S1=""
5 if test $S1 ;then
6 echo "The length of string1 is not zero"
7 else
8 echo "The length of string1 is zero"
9 fi
10 S2="Hello"
11 if test $S2 ;then
12 echo "The length of string2 is not zero"
13 else
14 echo "The length of string2 is zero"
15 fi
```

其中，代码第 4 行定义了一个字符串，内容为空，且保存到变量 S1 中。代码第 5 行，使用“-n”对变量 S1 所表示的字符串长度是否为零进行判断。代码第 10 行定义了另一个字符串，内容为“Hello”，且保存到变量 S2 中。代码第 11 行，使用“-n”对变量 S2 所表示的字符串长度是否为零进行判断。该程序的执行过程如下所示。

```
#chmod +x stringnozero           //赋予文件 stringnozero 执行权限
#./stringnozero                  //执行 stringnozero
String length test:
The length of string1 is zero
The length of string2 is not zero
```

17.4.5 非空测试

可以在条件表达式中直接对字符串是否为空进行测试。例如，编写 Shell 程序 stringnonnull，测试变量 S1 是否为零。如果长度不为零，显示“The length of string is not zero”，否则显示“The length of string is zero”。该程序的具体代码如下所示（代码 stringnonnull）。

```
1 #!/bin/bash
2 #filename:stringnonnull
3 echo "Please enter a string:"
4 read S1
5 if test $S1 ;then
6 echo "The length of string is not zero"
7 else
8 echo "The length of string is zero"
9 fi
```

其中，代码第 4 行读入一个字符串，且保存到变量 S1 中。代码第 5 行，直接对变量 S1 所表示的字符串是否为空进行判断。该程序的执行过程如下所示。

```
#chmod +x stringnonnull           //赋予文件 stringnonnull 执行权限
#./ stringnonnull                 //执行 stringnonnull
Please enter a string:

The length of string is zero
#./ stringnonnull                 //执行 stringnonnull
Please enter a string:
```



```
qwqwq
The length of string is not zero
```

17.5 测试逻辑运算符

测试命令可以由单个测试条件构成，也可以由多个测试条件构成，这些测试条件之间用逻辑运算符连接起来。Shell 提供了非（“!”）、或（“-o”）、与（“-a”）三个逻辑运算符，其优先级为：“!” 最高，“-a” 次之，“-o” 最低。逻辑运算符的具体说明如表 17-4 所示。

表 17-4 逻辑运算符

逻辑操作符	说 明
-a	二进制“与”操作符
-o	二进制“或”操作符
!	一元“非”操作符

17.5.1 逻辑与测试

逻辑与“-a”表示只有所有测试条件都为真时，表达式的值才为真；只要有一个测试条件为假，表达式的值即为假。

例如，编写 Shell 程序 myand，测试变量 X 的值是否大于等于 10 且小于等于 50。如果是，显示“10<= \$X <=50”，否则显示“not between 10 and 50”。测试表达式为“test \$X -ge 10 -a \$X -le 50”，该程序的具体代码如下所示（代码 myand）。

```
1 #!/bin/bash
2 #filename:myand
3 read X
4 if test $X -ge 10 -a $X -le 50 ;then
5 echo "10<= $X <=50"
6 else
7 echo " not between 10 and 50"
8 fi
```

其中，代码第 4 行使用“-a”对变量 X 是否大于等于 10 且小于等于 50 进行测试。该程序的执行过程如下所示。

```
#chmod +x myand           //赋予文件 myand 执行权限
#./myand                  //执行程序 myand
60                        //输入 60
60 not between 10 and 50
#./myand                  //执行程序 myand
34                        //输入 30
10<= 34 <=5
```

例如，编写 Shell 程序 myand2，测试变量 FILE 的值是否是可读的普通文件。如果是，显示“Normal and Readonly”，否则显示“Not Normal and Readonly”。测试表达式为“test -r \$FILE -a -f \$FILE”，该程序的具体代码如下所示（代码 myand2）。

```
1 #!/bin/bash
2 #filename:myand2
3 echo Please enter filename:
```



```

4 read FILE
5 if test -r $FILE -a -f $FILE ;then
6 echo "Normal and Readonly"
7 else
8 echo "Not Normal and Readonly"
9 fi

```

其中，代码第 5 行使用“-a”对变量 FILE 是否是普通文件且只读进行判断。该程序的执行过程如下所示。

```

# ls -al //以长格式显示当前目录
-rw-r--r-- 1 root root 220 8月 14 20:01 rc.local
drwxr-xr-x 2 root root 4096 8月 14 17:17 kk
-rw-r--r-- 1 root root 166 8月 17 03:21 myand2
#chmod u+x myand2
#./myand2 //执行程序
Please enter filename:
kk //输入目录名 kk
Not Normal and Readonly
#./myand2
Please enter filename:
rc.local //输入文件名 rc.local
Normal and Readonly

```

17.5.2 逻辑或测试

逻辑或“-o”表示只要有一个测试条件为真，表达式的值即为真；只有所有测试条件都为假时，表达式的值才为假。

例如，编写 Shell 程序 myor，测试变量 NAME 的值是否是 Zhang 或 Chen。如果是，显示“Your Last name is Zhang or Chen”，否则显示“Your Last name is not Zhang or Chen”。测试表达式为“\$NAME = Zhang -o \$NAME = Chen”，该程序的具体代码如下所示（代码 myor）。

```

1 #!/bin/bash
2 #filename:myor
3 echo Please enter your Last name:
4 read NAME
5 if test $NAME = Zhang -o $NAME = Chen ;then
6 echo "Your Last name is Zhang or Chen"
7 else
8 echo "Your Last name is not Zhang or Chen"
9 fi

```

其中，代码第 5 行使用“-o”对变量 NAME 是否是 Zhang 或 Chen 进行判断。该程序的执行过程如下所示。

```

#chmod +x myor //赋予文件 myor 执行权限
#./myor //执行程序 myor
Please enter your Last name:
yang
Your Last name is not Zhang or Chen
#./myor
Please enter your Last name:
Zhang
Your Last name is Zhang or Chen

```



17.5.3 逻辑非测试

逻辑非“!”表示取反，即测试条件为真时，表达式的值为假，而测试条件为假时，表达式的值为真。

例如，编写 Shell 程序 myd，测试变量 FILE 是否是目录。如果不是目录，显示“Not Directory”，否则显示“This is a directory”。测试表达式为“test !-d \$FILE”，该程序的具体代码如下所示（代码 myd）。

```
1 #!/bin/bash
2 #filename:myd
3 echo Please enter a file name:
4 read FILE
5 if test ! -d $FILE ;then
6 echo "Not Directory"
7 else
8 echo "This is a directory"
9 fi
```

其中，代码第 5 行使用“! -d”对变量 FILE 是否是目录进行判断，“!”表示取非。该程序的执行过程如下所示。

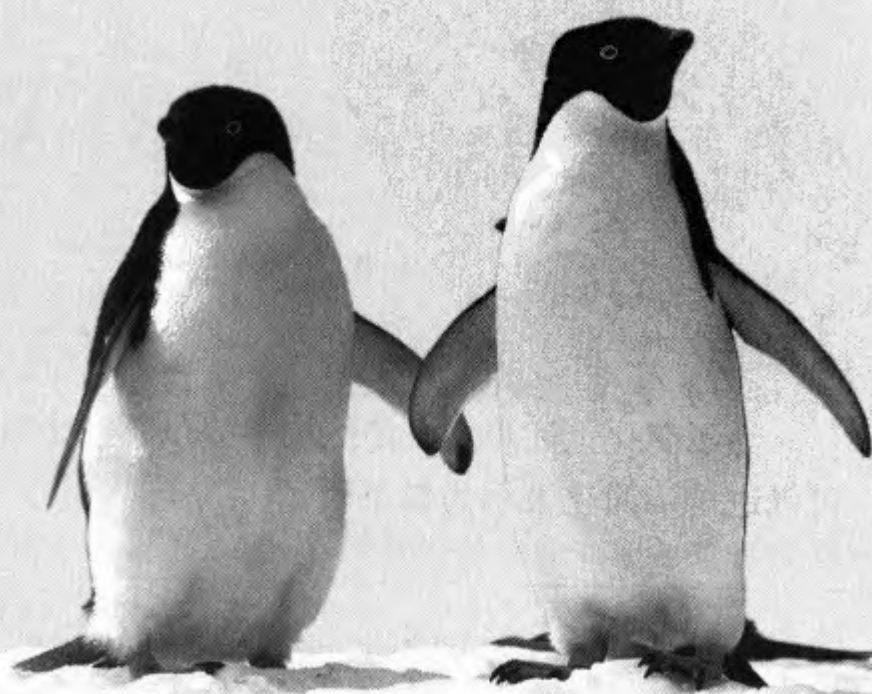
```
# ls -al //以长格式显示当前目录
-rw-r--r-- 1 root root 220 8月14 20:01 rc.local
drwxr-xr-x 2 root root 4096 8月14 17:17 kk
-rw-r--r-- 1 root root 166 8月17 03:21 myd
#chmod u+x myd
#./myd //执行程序
Please enter a file name:
kk //输入目录名 kk
This is a directory
#./myd //执行程序
Please enter a file name:
rc.local //输入文件名 rc.local
Not Directory
```


Linux

第18章

Shell的控制结构

流程控制结构是程序语言中用来控制一段脚本执行流程的结构。Shell 提供了对多种流程控制结构的支持，包括条件结构、分支结构和循环结构。条件结构、分支结构和循环结构为程序提供了基本的决策能力，组合使用这几种结构可以完成复杂的任务。





18.1 if 结构

各种主流编程语言中都包括了条件结构，条件结构允许指定一个只有在特定条件满足时才会被执行的代码，即“如果条件为真，执行这些代码，否则跳过这些代码”。Shell 中使用 if 结构实现这一功能。

18.1.1 if 结构的语法格式

最简单的判断形式就是 if。if 结构允许对表达式的值进行测试，并根据测试值执行相应的语句，if 结构包括 if、then、else、elif、fi 等语句。if 结构的语法格式如下所示。

```
if 表达式
then 命令表
[else 命令表]
fi
```

其中选项的说明如下。

- “表达式”是判断条件，当“表达式”的值为真时，执行“then”之后的命令表。如果“表达式”的值为假，执行“else”之后的命令表。“命令表”表示一条或若干条命令。
- “else”可以省略。
- “fi”表示 if 结构结束。

可以使用分号将多个命令在一行内进行组合，因此可以把 then 与 if 放在同一行中，并在 then 的前面加一个分号。由此，一些程序员出于美观和节省空间的考虑，经常将以上格式简写为如下形式。

```
if 表达式 ; then
命令表
[else 命令表]
fi
```

if 语句可以无限层地嵌套在其他 if 语句中，从而实现多重选项，如下所示。

```
if 表达式
then 命令表
else
    if 表达式
    then 命令表
    else
        if 表达式
        then 命令表
        else
            ...
        fi
    fi
fi
```

为了简化上述形式，if 结构中提供了 elif 语句，相当于“else if”。因此，上述多层嵌套可以进一步简化表示为如下所示。


```
if 表达式
then 命令表
[ elif 表达式
  then 命令表 ]
...
[ else 命令表 ]
fi
```



应避免过多地使用 elif 语句，因为过多地使用 elif 语句会使程序难于理解、调试和修改。在遇到需要多处使用分支进行判断的情况时，应尽量使用 case 语句。

18.1.2 if...then...fi 示例

if...then...fi 是最基本的 if 结构，即当条件为真时，执行一组指定的代码，否则跳过这组代码，执行 if 结构之后的代码。

例如，以下 Shell 程序中使用了 if...then...fi 结构，对用户输入的两个字符串进行比较，如果两字符串相同则显示输出“字符串相同”，否则不显示输出。程序流程如图 18-1 所示。

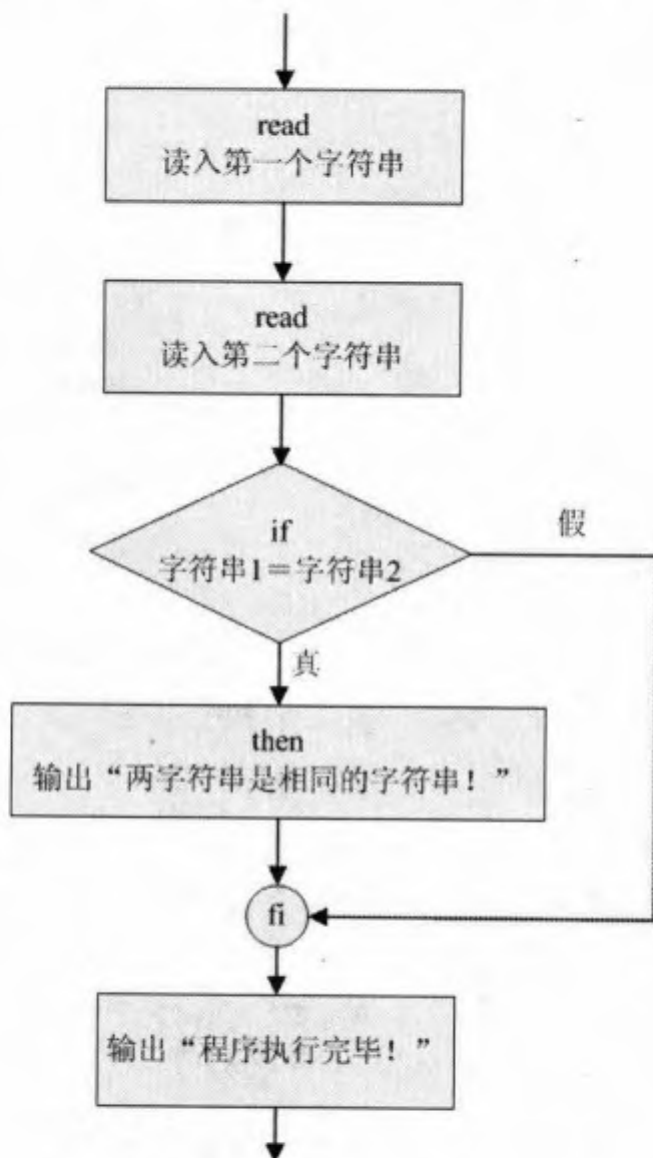


图 18-1 if...then...fi 结构示例流程图

该程序的具体代码如下所示（代码 ifsingle）。



```

1 #!/bin/bash                                //程序第1行的“#!/bin/bash”表示调用bash来执行脚本程序
2 #filename:ifsingle                          //程序第2行的“#filename:ifsingle”表示该程序的名称为
ifsingle
3 echo "请输入第一个字符串: "                //显示输出“请输入第一个字符串: ”
4 read WORD1                                 //读入变量WORD1
5 echo "请输入第二个字符串: "                //显示输出“请输入第二个字符串: ”
6 read WORD2                                 //读入变量WORD2
7 echo "-----"                             //显示输出分隔行
8 if test $WORD1 = $WORD2                    //第8~11行是if...then...fi结构。判断条件为
                                              //“$WORD1 = $WORD2”
                                              //即变量WORD1与变量WORD2是否相同
                                              //如果相同,则执行第10行
                                              //否则转到第12行
9 then                                       //if结构中的then语句
10     echo "两字符串是相同的字符串!"        //显示输出“两字符串是相同的字符串!”
11 fi                                       //if结构中的fi语句,表示if结构结束
12 echo "程序执行完毕!"                     //显示输出“程序执行完毕!”

```

该程序的执行过程如下所示。

```

#chmod u+x ifsingle                        //赋予文件ifsingle执行权限
#./ifsingle                                //执行ifsingle程序
请输入第一个字符串:
zkwjklidfj                                //输入“zkwjklidfj”
请输入第二个字符串:
zkwjklidfj                                //输入“zkwjklidfj”
-----
两字符串是相同的字符串!
程序执行完毕!

#./ifsingle                                //执行ifsingle程序
请输入第一个字符串:
kafjkl;dfj                                //输入“kafjkl;dfj”
请输入第二个字符串:
jiefdslakdf                                //输入“jiefdslakdf”
-----
程序执行完毕!

```

18.1.3 if...then...else...fi 示例

在使用 if...then...fi 结构时,只有条件为真时才执行指定的代码。但有时不仅在条件为真时需要执行一组指定的代码,在条件为假时也需要执行一组代码。在 if...then...fi 结构中引入 else 语句,可以实现该功能,这也使得 if 结构具有类似分支语句的作用。

例如,编写一个 Shell 程序,判断程序脚本是否带有位置参数。如果没有,显示“本脚本程序不带参数!”,否则显示所有位置参数。程序流程如图 18-2 所示。

该程序的具体代码如下所示(代码 ifelse)。

```

1 #!/bin/bash                                //程序第1行的“#!/bin/bash”表示调用bash来执行脚本程序
2 #filename:ifelse                          //程序第2行的“#filename: ifelse”表示该程序的名称为
ifelse
3 if [ $# -eq 0 ]                            //第3~8行为if...then...else...fi结构。判断条件为“$# -eq 0”
                                              //其中“$#”表示当前脚本的参数数目
                                              //该判断条件即表示,如果当前脚本的参数数目为0则执行第5行
                                              //否则执行第7行
4 then                                       //if结构中的then语句
5     echo "本脚本程序不带参数!"           //显示输出“本脚本程序不带参数!”

```



```
6 else // if 结构中的 else 语句
7 echo "本脚本程序带参数: $*" //显示输出“本脚本程序带参数: $*”。其中“$*”表示所有位置参数
8 fi // if 结构中的 fi 语句，表示 if 结构结束
```

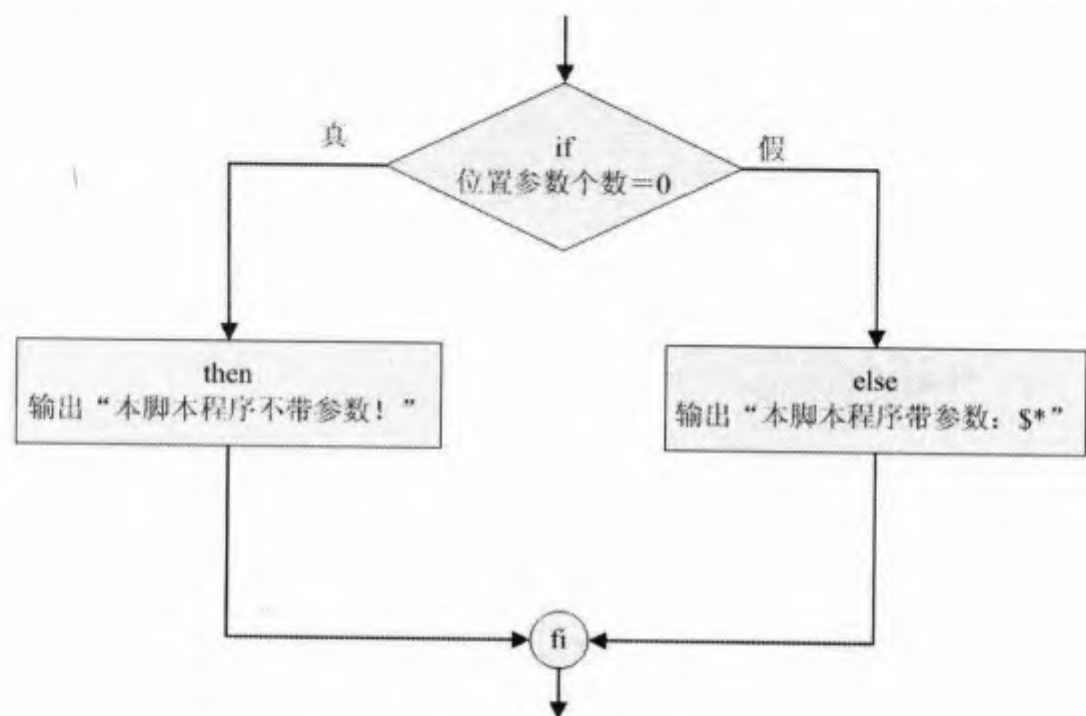


图 18-2 if...then...else...fi 示例流程图

该程序的执行过程如下所示。

```
# chmod +x ifelse //赋予文件 ifelse 执行权限
# ./ifelse //执行程序 ifelse
本脚本程序不带参数!
# ./ifelse 1111
本脚本程序带参数: 1111
# ./ifelse 1111 222 333 aaa
本脚本程序带参数: 1111 222 333 aaa
```

18.1.4 if...then...elif...示例

使用 if...then...else...fi 结构时，如果条件为假将无条件执行指定的代码段。但有时用户会希望在执行该代码段之前先进行一次判断，这时就可以使用 if...then...elif...结构。事实上，语句“elif”可以理解为“else if”的组合，用来实现多路判断。

例如，以下 Shell 程序中使用了 if...then...elif...then...else...fi 结构，对用户输入的三个字符串进行比较。如果三个字符串相同，显示输出“三个字符串相同”，如果其中两个字符串相同，则显示输出相同的字符串，否则不显示输出。程序流程如图 18-3 所示。

该程序的代码如下所示（代码 ifelif）。

```
1 #!/bin/bash //程序第 1 行的“#!/bin/bash”表示调用 bash 来执行脚本程序
2 #filename:ifelif //程序第 2 行的“#filename: ifelif”表示该程序的名称为 ifelif
3 echo "输入第一个字符串: " //显示输出“输入第一个字符串: ”
4 read STR1 //读入变量 STR1
5 echo "输入第二个字符串: " //显示输出“输入第二个字符串: ”
6 read STR2 //读入变量 STR2
7 echo "输入第三个字符串: " //显示输出“输入第三个字符串: ”
8 read STR3 //读入变量 STR3
9 if [ $STR1 = $STR2 -a $STR2 = $STR3 ] //第 9~23 行为
//if...then...elif...then...elif...then...elif...then...else...fi 结构
```

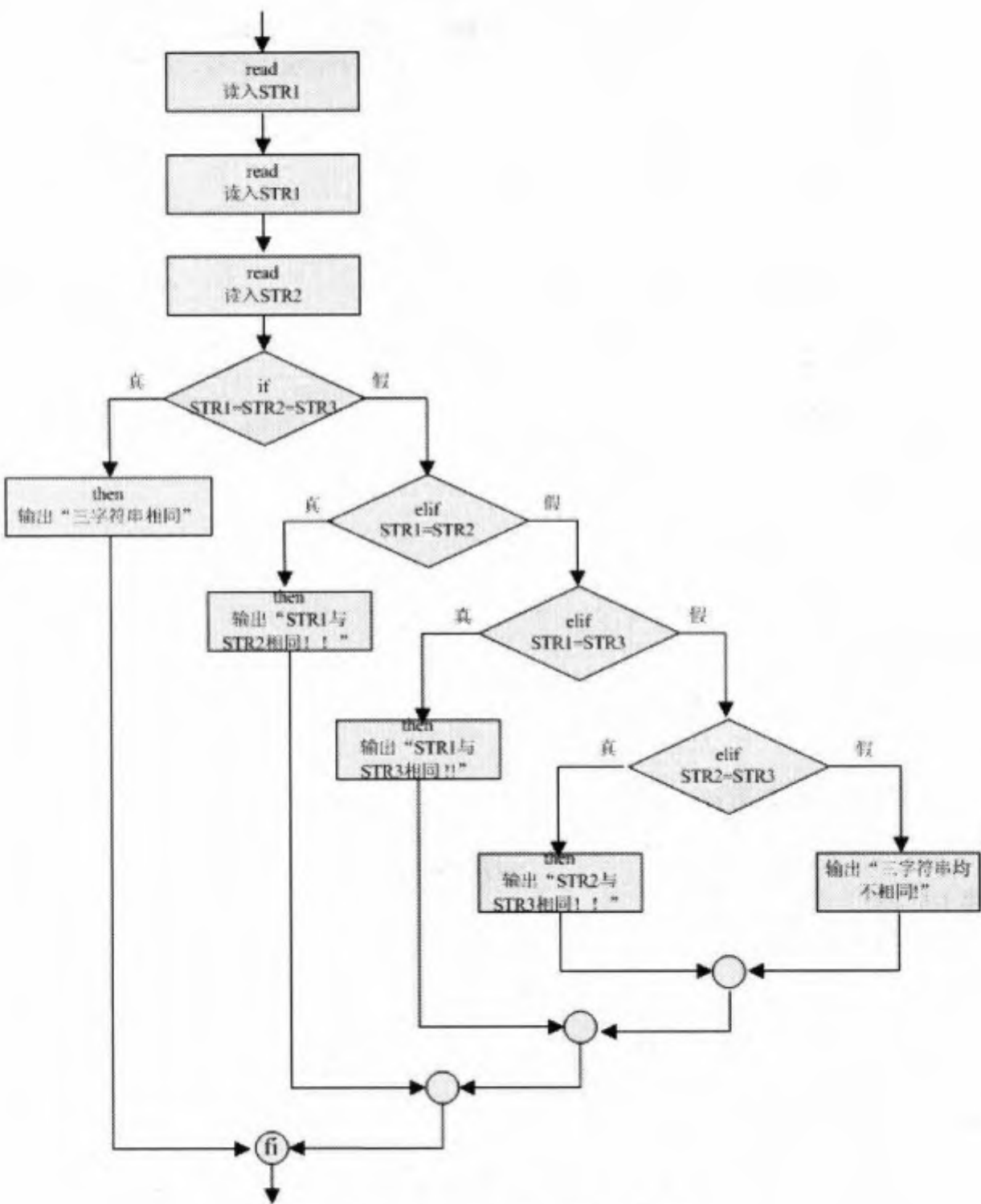


图 18-3 if...then...elif...示例流程图

```
10 then
11   echo "三字符串相同 "
12 elif [ $STR1 = $STR2 ]
13 then
14   echo "STR1 与 STR2 相同!! "
15 elif [ $STR1 = $STR3 ]
```

//本行代码“\$STR1 = \$STR2 -a \$STR2 = \$STR3”
//用于判断字符串 STR1、STR2、STR3 是否相同，其中“-a”
//表示逻辑与，即 STR1 与 STR2 相等并且 STR2 与 STR3 相
//等时，表达式“\$STR1 = \$STR2 -a \$STR2 = \$STR3”才
//为真。如果为真，则执行第 11 行，否则转到第 12 行继续进
//行判断
//if 结构中的 then 语句，匹配第 9 行的 if
//显示输出“三字符串相同”
//if 结构中的 elif 语句，其中判断条件“\$STR1 = \$STR2”用于
//判断 STR1 是否与 STR2 相同，如果相同则执行第 14 行
//否则转到第 15 行继续进行判断
//if 结构中的 then 语句，匹配第 12 行的 elif
//显示输出“STR1 与 STR2 相同!!”
//if 结构中的 elif 语句，其中判断条件“\$STR1 = \$STR3”用于判
//断 STR1 是否与 STR3 相同，如果相同则执行第 17 行，否则转到
//第 18 行继续进行判断


```

16 then                                //if 结构中的 then 语句，匹配第 15 行的 elif
17     echo "STR1 与 STR3 相同 !! "    //显示输出 "STR1 与 STR3 相同!! "
18 elif [ $STR2 = $STR3 ]              //if 结构中的 elif 语句，其中判断条件 "$STR2 = $STR3" 用于判
                                        //断 STR2 是否与 STR3 相同，如果相同则执行第 20 行，否则执行
                                        //第 22 行
19 then                                //if 结构中的 then 语句，匹配第 18 行的 elif
20     echo "STR2 与 STR3 相同!! "    //显示输出 "STR2 与 STR3 相同!! "
21 else                                //if 结构中的 else 语句，匹配第 18 行的 elif
22     echo "三字符串均不相同! "      //显示输出 "三字符串均不相同!"
23 fi                                  //if 结构中的 fi 语句，匹配第 9 行的 if，表示 if 结构结束

```

该程序的执行过程如下所示。

```

#chmod u+x ifelif                      //赋予文件 ifelif 执行权限
# ./ifelif                             //执行程序 ifelif
输入第一个字符串:
abcdefg                                //输入 "abcdefg"
输入第二个字符串:
abcdefg                                //输入 "abcdefg"
输入第三个字符串:
abcdefg                                //输入 "abcdefg"
三字符串相同
# ./ifelif                             //执行程序 ifelif
输入第一个字符串:
abcdefg                                //输入 "abcdefg"
输入第二个字符串:
abcdefg                                //输入 "abcdefg"
输入第三个字符串:
abcezzz                                //输入 "abcezzz"
STR1 与 STR2 相同!!
# ./ifelif                             //执行程序 ifelif
输入第一个字符串:
abcdefg                                //输入 "abcdefg"
输入第二个字符串:
hijklmn                                //输入 "hijklmn"
输入第三个字符串:
opqrstu                                //输入 "opqrstu"
三字符串均不相同!
# ./ifelif                             //执行程序 ifelif
输入第一个字符串:
abceefg                                //输入 "abceefg"
输入第二个字符串:
hkjklsf                                //输入 "hkjklsf"
输入第三个字符串:
abceefg                                //输入 "abceefg"
STR1 与 STR3 相同 !!

```

18.1.5 if 嵌套

if 语句可以嵌套在其他 if 语句中，并且可以通过多重嵌套实现多重选择。例如，编写一个 Shell 程序，对程序的命令行参数进行判断，当命令行参数为“YES”或“yes”时，显示输出“Your argument is Yes !!”；当命令行参数为“NO”或“no”时，显示输出“Your argument is NO !!”；如果都不是，则输出“Error argument !!”。程序流程如图 18-4 所示。

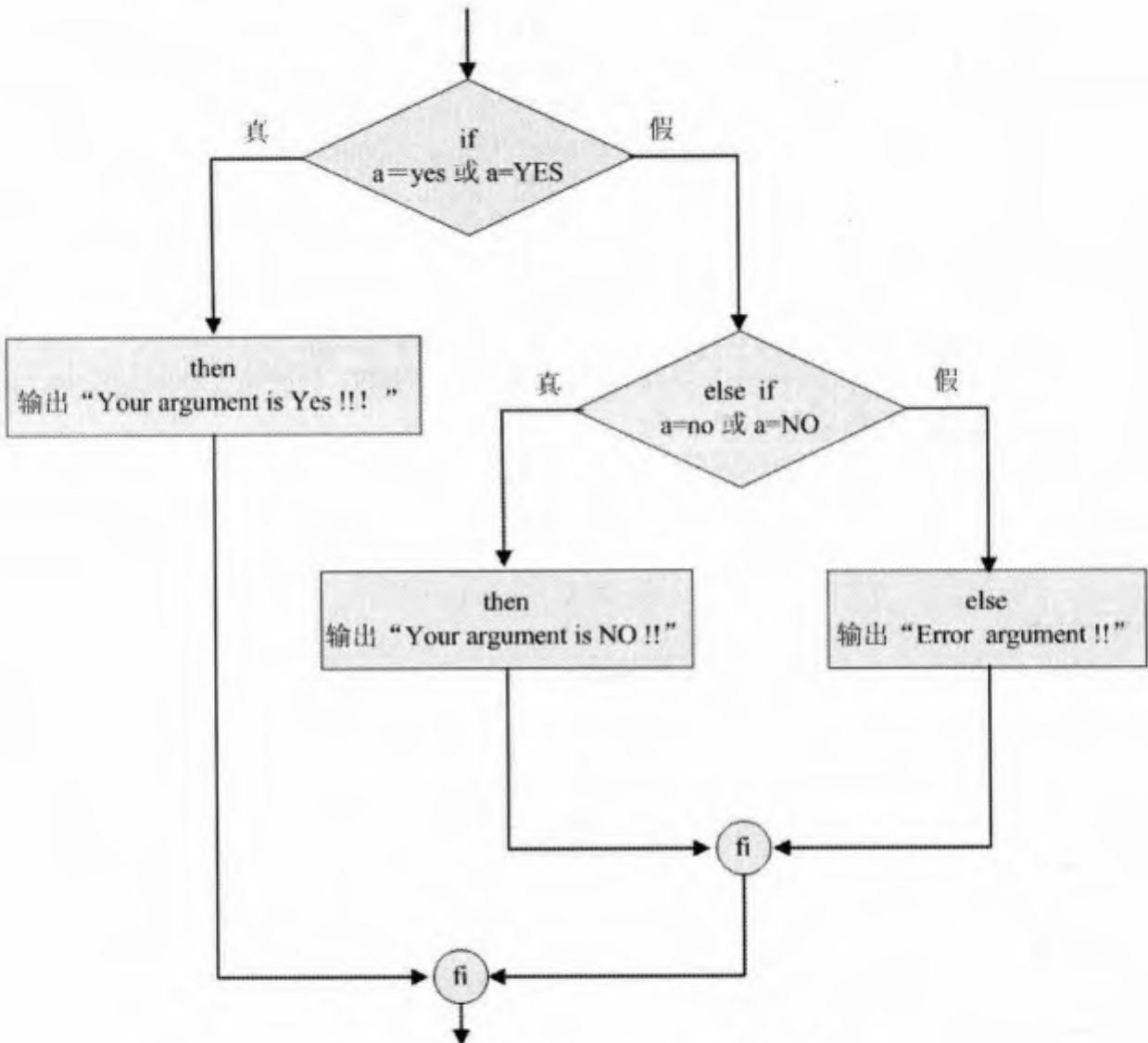


图 18-4 if...then...fi 结构嵌套示例流程图

该程序的具体代码如下所示（代码 yesno）。

<pre>1 #!/bin/bash 执行脚本程序 2 #filename:yesno 的名称为 yesno 3 a=\$1 4 if [\$a = "YES" -o \$a = "yes"] ; then a 的值是 "YES" 或运算，当变量 a 的 = "yes" 的 5 echo "Your argument is Yes !!" 6 else 7 if [\$a = "NO" -o \$a = "no"] ;then 的值是 "NO" 运算，当变量 a 的值 的值为真 8 echo "Your argument is NO !!" 9 else 10 echo "Error argument !!" 出 "Error argument !!"</pre>	<pre>//程序第 1 行的 "#!/bin/bash" 表示调用 bash 来 //程序第 2 行的 "#filename:yesno" 表示该程序 //将第一个位置变量\$1 的值，赋值给变量 a //第 4~12 行是 if...then...else...fi 结构。当变量 //或 "yes" 时，执行第 5 行，否则执行第 7~11 行 //其中 "\$a = "YES" -o \$a = "yes" 是逻辑 //值是 "YES" 或 "yes" 时，"\$a = "YES" -o \$a //值为真 //用于显示输出 "Your argument is Yes !!" // if 结构中的 else 语句 //第 7~11 行是 if...the...else...fi 结构。当变量 a //或 "no" 时，执行第 8 行，否则执行第 10 行 //其中 "\$a = "NO" -o \$a = "no" 是逻辑或 //是 "NO" 或 "no" 时，"\$a = "NO" -o \$a = "no" //用于显示输出 "Your argument is NO !!" // if 结构中的 else 语句，匹配第 7 行的 if // "echo "Error argument !!" 用于显示输</pre>
---	---


```

11 fi //if 结构中的 fi 语句，匹配第 7 行的 if，表示 if 结构结束
12 fi //if 结构中的 fi 语句，匹配第 4 行的 if，表示 if 结构结束

```

该代码的执行过程如下所示。

```

# chmod a+x yesno //赋予文件 yesno 执行权限
# ./yesno yes //带参数“yes”执行 yesno 程序
Your argument is Yes !!
# ./yesno YES //带参数“YES”执行 yesno 程序
Your argument is Yes !!
# ./yesno no //带参数“no”执行 yesno 程序
Your argument is NO !!
# ./yesno NO //带参数“NO”执行 yesno 程序
Your argument is NO !!
# ./yesno ok //带参数“ok”执行 yesno 程序
Error argument !!

```

如果使用 `elif` 语句该程序，可以进一步简化为如下所示（代码 `yesno2`）。

```

1 #!/bin/bash
2 #filename:yesno2
3 a=$1
4 if [ $a = "YES" -o $a = "yes" ] ; then
5     echo "Your argument is Yes !!"
6 elif [ $a = "NO" -o $a = "no" ] ; then
7     echo "Your argument is NO !!"
8 else
9     echo "Error argument !!"
10 fi

```

从执行结果可以看到，`yesno` 与 `yesno2` 相同。



由于多层嵌套会降低程序的可读性，建议用缩进格式编写嵌套结构。

18.2 case 结构

分支结构 `case` 是一个多路分支判断结构，可以测试一个表达式的值，并根据测试结果选择执行相应的分支程序。也就是说，`case` 控制结构提供了一种多分支选择机制，可以用来替换 `if...then...elif...结构`。

18.2.1 case 结构的语法格式

`case` 结构以“`case`”开头，以“`esac`”结尾。具体执行哪一支，取决于“表达式”与每个分支前的“模式”之间的匹配情况。`case` 结构的语法格式如下所示。

```

case 表达式 in
    模式 11[|模式 12]...) 命令表 1;;
    模式 21[|模式 22]...) 命令表 2;;
    .....
    *) 命令表 n;;
esac

```

其中各项说明如下。



- “表达式”是判断条件，通常是一个变量名称。“表达式”与每一分支前的模式进行比较，如果匹配，则执行相应分支右括号“)”后面的命令表。
 - 每一分支中的模式，可以有多个，但必须以“)”结尾。
 - 命令表可以是一个命令或多个命令。
 - 分支语句的个数没有规定，可以无限制地增加。每一分支以“;;”表示该分支结束。
 - 分支“*)”表示在前面所有可能的匹配都不满足时的处理方式，该分支不是必需的，可以省略。但如果使用该分支则必须将其放在所有其他分支的后面。
 - “esac”表示 case 结构结束。
 - 模式之间可以使用“|”，表示各模式之间是“或”的关系。如“P|p”意味着大写和小写的 p 都可匹配。
 - 模式中可以使用 Shell 通配符，如“*”、“?”和“[]”等。
- case 结构的流程如图 18-5 所示。

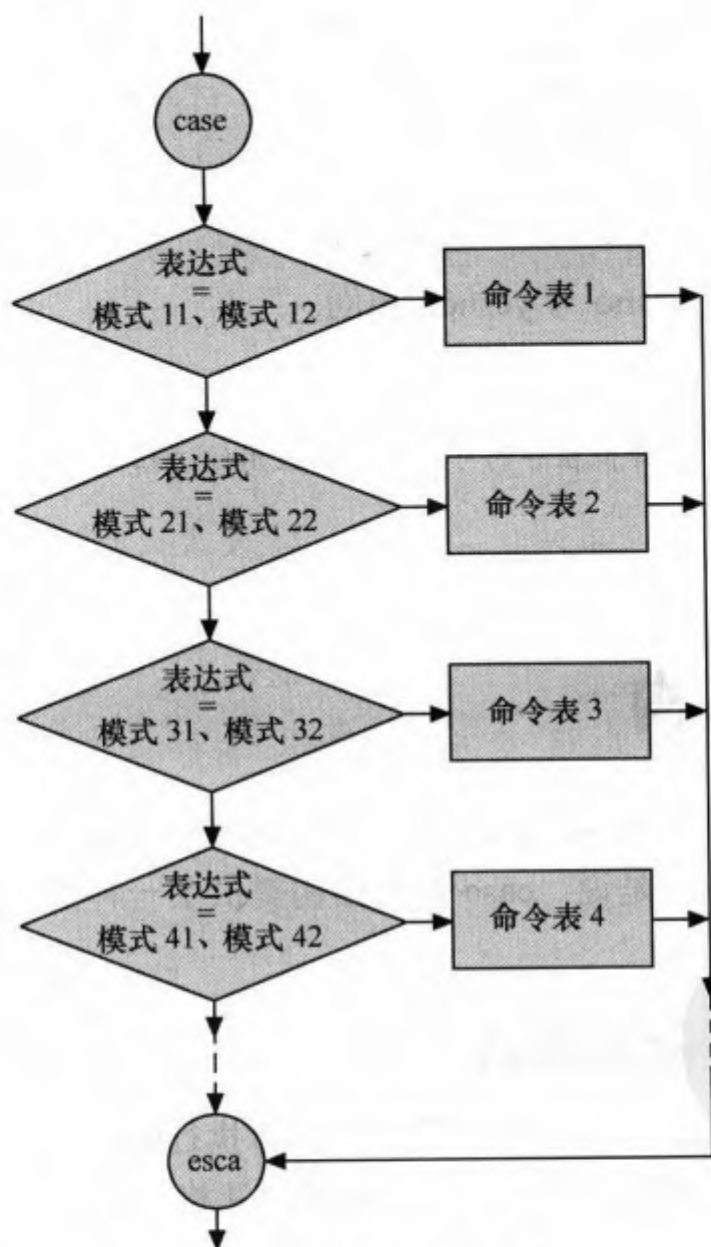


图 18-5 case 结构流程图

18.2.2 case 结构示例

- (1) 编写一个 Shell 程序，对用户输入的第一个位置参数进行判断，当参数为“dir”或

“path”时，显示当前的工作目录；当参数为“date”或“time”时，显示当前时间；若都不是，则显示提示信息。代码实现如下所示（代码 casexample）。

```

1 #!/bin/bash           //程序第1行的“#!/bin/bash”表示调用bash来执行脚本程序
2 #filename:casexample  //程序第2行的“#filename:casexample”表示该程序的名称为
casexample
3 case $1 in            //第3~8行是case...in...esac结构，对$1进行判断。如果$1的值为“dir”
或“path”，              //则执行第4行，显示当前目录。如果$1的值为“date”或“time”，则执
行第5行
                        //显示当前日期。如果是其他值，则执行第6、7行，显示提示信息
4 dir|path ) echo " Current path is 'pwd' " ;; //“dir|path”分支，即如果$1的值为
“dir”或“path”则执
                        //行该分支，其中“|”表示或
                        //“echo " Current path is 'pwd' " ”用于显示当前目录，其
                        //中命令pwd使用了反引号，表示引用pwd命令的执行结果
5 date|time ) echo " Today is 'date' " ;; //“date|time”分支，即如果$1的值为“date”
或“time”
                        //则执行该分支，其中“|”表示或
                        //“echo " Today is 'date' " ”用于显示当前日期、时间，
                        //其中命令date使用了反引号，表示引用date命令执行结果
6 * ) echo " Invalid augument!! " //“*”分支，即如果$1的值既不是“dir”或
“path”，也不
                        //是“date”或“time”，则执行该分支。该分支包括第6、
                        //7两行。本行“echo " Invalid augument!! " ”用于显示输
                        //出“Invalid augument!!”
7 echo "Enter argument: dir/path/date/time" ;; //用于显示提示信息
                        //“Enter argument: dir/path/date/time”
8 esac // case结构中的esac语句，表示该case结构结束

```

该代码的执行过程如下所示。

```

#chmod u+x casexample //赋予文件casexample执行权限
# ./casexample dir     //执行程序casexample，带参数“dir”
Current path is /prog  //显示当前目录/prog
# ./casexample path    //执行程序casexample，带参数“path”
Current path is /prog  //显示当前目录为/prog
# ./casexample time    //执行程序casexample，带参数“time”
Today is 日 8月 17 11:02:21 CST 2008 //显示当前时间
# ./casexample date    //执行程序casexample，带参数“date”
Today is 日 8月 17 11:02:26 CST 2008 //显示当前时间
# ./casexample pwd     //执行程序casexample，带参数“pwd”
Invalid augument!!    //显示提示信息
Enter argument: dir/path/date/time
$ casexample path      //执行程序casexample，带参数“path”
Current path is /home/sullivan //显示当前目录为/home/sullivan

```

(2) 编写一个Shell程序，对用户输入的参数进行判断，当参数为“Yes”、“Y”、“YES”、“y”或“yes”时，显示提示“Yes!!”；当参数为“No”、“n”、“NO”、“no”或“N”时，显示提示“No!!”；若都不是，则显示提示信息。代码实现如下所示（代码 yes_no）。

```

1 #!/bin/bash           //程序第1行的“#!/bin/bash”表示调用bash来执行脚本程序
2 #filename:yes_no      //程序第2行的“#filename:yes_no”表示该程序的名称为 yes_no
3 a=$1                 //定义变量a，将位置变量$1赋值给变量a
4 case $a in            //第3~9行是case...in...esac结构，对变量a进行判断。如果a的值为“Yes”、
a的值                  //“Y”、“YES”、“y”或“yes”，则执行第5行，显示提示“Yes!!”。如果

```



```

//为“N”、“n”、“NO”、“no”或“No”执行第6行，显示提示“No!!”
//如果是其他值，则执行第7、8行，显示提示信息
5 Yes|Y|YES|y|yes) echo "Yes!!" ;; // “Yes|Y|YES|y|yes”分支，即如果 a 的值为
“Yes”、“Y”、
// “YES”、“y”或“yes”时，则执行该分支，其中“|”表示或
// “echo “Yes!!””用于显示“Yes!!”
6 N|n|NO|no|No) echo "No!!" ;; // “N|n|NO|no|No”分支，即如果 a 的值为“N”、
“n”、“NO”、
// “no”或“No”时，则执行该分支，其中“|”表示或。“echo “No!!””
//用于显示“No!!”
7 *) echo "Error agument!!" // “*”分支，即如果变量 a 的值既不是“Yes”、
“Y”、
// “YES”、“y”或“yes”，也不是“N”、“n”、
// “NO”、“no”或“No”，则执行该分支。该分支
//包括第7、8两行。本行“echo “Error agument!!””
//用于显示输出“Error agument!!”
8 echo "Pleaese enter Yes/Y/YES/y/yes/N/n/NO/no/No " // 用
于显示输出
// “Pleaese enter Yes/Y/YES/y/yes/N/n/NO/no/No ”
9 esac // case 结构中的 esac 语句，表示 case 结构结束

```

该程序的执行过程如下所示。

```

#chmod u+x yes_no //赋予文件 yes_no 执行权限
# ./yes_no yes //执行程序 yes_no，带参数“yes”
Yes!!
# ./yes_no y //执行程序 yes_no，带参数“y”
Yes!!
# ./yes_no Y //执行程序 yes_no，带参数“Y”
Yes!!
# ./yes_no YES //执行程序 yes_no，带参数“YES”
Yes!!
# ./yes_no NO //执行程序 yes_no，带参数“NO”
No!!
# ./yes_no N //执行程序 yes_no，带参数“N”
No!!
# ./yes_no no //执行程序 yes_no，带参数“no”
No!!
# ./yes_no ok //执行程序 yes_no，带参数“ok”
Error agument!!
Pleaese enter Yes/Y/YES/y/yes/N/n/NO/no/No

```

(3) 编写一个 Shell 程序，对当天是星期几进行判断。时间的获取采用命令“date +%u”，其中“+%u”表示以数字方式显示当天是一周中的第几天。代码实现如下所示（代码 week）。

```

1 #!/bin/bash //程序第1行的“#!/bin/bash”表示调用 bash 来执行脚本程序
2 #filename:week //程序第2行的“#filename:week”表示该程序的名称为 week
3 WEEK='date +%u' // “WEEK='date +%u'”表示将命令“date +%u”的执行结果返回给变量 WEEK
//反引号表示取得命令“date +%u”的执行结果
4 case $WEEK in //第4~13行是 case...in...esac 结构，对变量 WEEK 进行判断。如果 WEEK 的
//值为“1”，则执行第5行，如果 WEEK 的值为“2”，则执行第6行，如果 WEEK
//的值为“3”，则执行第7行.....依此类推。如果 WEEK 的值不是1~7，则执行第
//12行，显示提示信息
5 1) echo "今天是星期一" ;; // “1”分支，即当变量 WEEK 的值为1时，执行该分支。“echo “今
天是星
//期一””用于显示“今天是星期一”
6 2) echo "今天是星期二" ;; // “2”分支，即当变量 WEEK 的值为2时，执行该分支。“echo “今

```



```

天是星
7 3)echo "今天是星期三";; //期二“”用于显示“今天是星期二”
天是星 //“3”分支，即当变量 WEEK 的值为 3 时，执行该分支。“echo “今
    //期三””用于显示“今天是星期三”
8 4)echo "今天是星期四";; //“4”分支，即当变量 WEEK 的值为 4 时，执行该分支。“echo “今
天是星 //期四””用于显示“今天是星期四”
9 5)echo "今天是星期五";; //“5”分支，即当变量 WEEK 的值为 5 时，执行该分支。“echo “今
天是星 //期五””用于显示“今天是星期五”
10 6)echo "今天是星期六";; //“6”分支，即当变量 WEEK 的值为 6 时，执行该分支。“echo “今
天是星 //期六””用于显示“今天是星期六”
11 7)echo "今天是星期日";; //“7”分支，即当变量 WEEK 的值为 7 时，执行该分支。“echo “今
天是星 //期日””用于显示“今天是星期日”
12 *)echo "Pleade enter 1/2/3/4/5/6/7";; //“*”分支，即当变量 WEEK 的值不是 1~7 时，执行该
分支 //“echo “Pleade enter 1/2/3/4/5/6/7””用于显示“Pleade
enter // 1/2/3/4/5/6/7”
13 esac // case 结构中的 esac 语句，表示该 case 结构结束

```

例如，执行程序 week，结果如下所示。

```

# ./week
今天是星期日

```

(4) case 结构语句的工作方式类似于 if 结构语句，但是 case 允许条件的可能值多于两个。在一个 if 语句中，判断条件只能是布尔类型的值，其值或者为真，或者为假。而在 case 语句结构中，条件值可以是整数、字符串等。case 结构语句中的判断条件只求值一次并用来和每个分支进行比较，而在 if 结构中，条件会要求再次求值。以下示例程序 score 和 score2 将分别采用 if 和 case 结构语句来实现程序的分支。程序 score 的具体代码如下所示（代码 score）。

```

1 #!/bin/bash //程序第 1 行的“#!/bin/bash”表示调用 bash 来执行脚本程序
2 #filename:score //程序第 2 行的“#filename:score”表示该程序的名称为 score
3 echo "请输入你的分数:" //“echo “请输入你的分数:””用于显示输出“请输入你的分数:”
4 read SCORE //使用 read 语句读入用户输入的数据，且保存在变量 SCORE
5 CONVERT='expr $SCORE / 10' //将变量 SCORE 的值与 10 整除，且将整除后的结果保存到变量
//CONVERT 中。其中“`expr $SCORE / 10`”使用了反引号，表示取得
//“expr $SCORE / 10”的执行结果
6 echo "-----" //显示输出分隔行
7 if [ $CONVERT -eq 9 ] //第 7~21 行是 if...then...elif...then...elif...then ...
else...fi //结构。本行判断变量 CONVERT 的值是否等于 9，如果等于 9，转到第 9
//行继续执行，如果不等于 9，则转到第 10 行继续执行
8 then // if 结构中的 then 语句，匹配第 7 行的 if
9 echo "恭喜,你的成绩为: 优秀" //显示输出“恭喜,你的成绩为: 优秀”
10 elif [ $CONVERT -eq 8 ] //判断变量 CONVERT 是否等于 8，如果等于 8，转到执行第 12 行
//如果不等于 8，转到执行第 13 行
11 then // if 结构中的 then 语句，匹配第 10 行的 elif
12 echo "你的成绩为: 良好" //显示输出“你的成绩为: 良好”
13 elif [ $CONVERT -eq 7 ] //判断变量 CONVERT 是否等于 7。如果等于 7，转到执行第 15 行。如果
//不等于 7，转到执行第 16 行
14 then // if 结构中的 then 语句，匹配第 13 行的 elif

```




```

15 echo "你的成绩为：中等" //显示输出“你的成绩为：中等”
16 elif [ $CONVERT -eq 6 ] //判断变量 CONVERT 是否等于 6。如果等于 6，转到执行第 18 行。如果
//不等于 6，转到执行第 20 行
17 then //if 结构中的 then 语句，匹配第 16 行的 elif
18 echo "你的成绩为：及格" //显示输出“你的成绩为：及格”
19 else //if 结构中的 else 语句，匹配第 16 行的 elif
20 echo "你的成绩为：不及格" //显示输出“你的成绩为：不及格”
21 fi //if 结构中的 fi 语句，匹配第 7 行的 if

```

程序 score 的执行过程如下所示。

```

# chmod +x score //赋予文件 score 执行权限
# ./score //执行程序 score
请输入你的分数：
93
-----
恭喜, 你的成绩为： 优秀
# ./score //执行程序 score
请输入你的分数：
87
-----
你的成绩为： 良好
# ./score //执行程序 score
请输入你的分数：
74
-----
你的成绩为：中等
# ./score //执行程序 score
请输入你的分数：
64
-----
你的成绩为：及格
# ./score //执行程序 score
请输入你的分数：
54
-----
你的成绩为：不及格
# ./score //执行程序 score
请输入你的分数：
23
-----
你的成绩为：不及格

```

程序 score2 中采用 case 分支对输入的成绩进行判断，具体代码如下所示（代码 score2）。

```

1 #!/bin/bash //程序第 1 行的“#!/bin/bash”表示调用 bash 来执行脚本程序
2 #filename:score2 //程序第 2 行的“#filename:score2”表示该程序的名称为 score2
3 echo "请输入您的成绩：" //显示输出“请输入您的成绩：”
4 read SCORE //读入用户输入的数据，且保存到变量 SCORE 中
5 CONVERT='expr $SCORE / 10' //将变量 SCORE 的值与 10 整除，且将整除后的结果保存到变量
CONVERT
//中。其中“'expr $SCORE / 10'”使用了反引号，表示取得
//“expr $SCORE / 10”的执行结果
6 echo "-----" //显示输出分隔行
7 case $CONVERT in //第 7~14 行是 case...in...esac 结构。当变量 CONVERT 的值为 9 时，
//执行第 8 行分支。当变量 CONVERT 的值为 8 时，执行第 9 行分支。当变
//量 CONVERT 的值为 7 时，执行第 10 行分支。当变量 CONVERT 的值为 6
//时，执行第 11 行分支。当变量 CONVERT 的值为 1、2、3、4 或 5 时，执

```



```

8 9) echo "恭喜! 优秀";; //行第 12 行分支。当是其他值时, 执行第 13 行分支
9 8) echo "良好";; //当变量 CONVERT 的值为 9 时执行该分支, 显示输出 "恭喜! 优秀"
10 7) echo "中等";; //当变量 CONVERT 的值为 8 时执行该分支, 显示输出 "良好"
11 6) echo "及格";; //当变量 CONVERT 的值为 7 时执行该分支, 显示输出 "中等"
12 1|2|3|4|5) echo "不及格";; //当变量 CONVERT 的值为 6 时执行该分支, 显示输出 "及格"
//当变量 CONVERT 的值为 1、2、3、4 或 5 时执行该分支, 显示输出 "不及格"
13 *) echo "输入错误!!" //当变量 CONVERT 的值不是 1~9 时执行该分支, 显示输出 "输入错误!!"
14 esac // case 结构中的 esac 语句, 表示 case 结构结束

```

程序 score2 的执行过程如下所示。

```

# chmod +x score2 //赋予文件 score2 执行权限
# ./score2 //执行程序 score2
请输入您的成绩:
98
-----
恭喜! 优秀
# ./score2 //执行程序 score2
请输入您的成绩:
89
-----
良好
# ./score2 //执行程序 score2
请输入您的成绩:
79
-----
中等
# ./score2 //执行程序 score2
请输入您的成绩:
66
-----
及格
# ./score2 //执行程序 score2
请输入您的成绩:
56
-----
不及格
# ./score2 //执行程序 score2
请输入您的成绩:
43
-----
不及格

```

从程序 score 与 score2 的输出结果可以看到, if 结构可以实现 case 结构的功能, 但用 case 可以使程序更清晰, 可读性更强。



case 是多路分支结构, 虽然可以通过多重嵌套的 if 语句来实现多路分支, 但嵌套越深, 程序的可读性越差。因此, 在适当的地方应该用 case 替换 if。

18.2.3 case 嵌套

case 结构同 if 结构一样, 也可以嵌套使用。为提高嵌套结构可读性, 建议采用缩进格式编写 case 结构。

例如, 以下 Shell 程序可以根据用户输入的类型 (teacher、student 或其他) 及姓名 (yang、



zhang、li 等) 显示不同的信息。其中用到了 case 嵌套，具体代码如下所示 (代码 case2)。

```

1 #!/bin/bash           //程序第 1 行的“#!/bin/bash”表示调用 bash 来执行脚本程序
2 #filename:case2       //程序第 2 行的“#filename: case2”表示该程序的名称为 case2
3 case $1 in            //第 3~22 行是 case...in...esac 结构，该结构共分为 student、teacher 和“*”
                        // (其他) 3 个分支。如果位置变量$1 的值为 student，则执行第 4~11 行的
student 分支，
                        //如果位置变量$1 的值为 teacher，则执行第 12~20 行的 teacher 分支。如果
既不是 student
                        //也不是 teacher，则执行 21 行的其他分支。其中在 student 分支及 teacher
分支内又嵌套了
                        //case...in...esac 结构
4 student)             //student 分支，即$1 的值为 student 时，执行该分支
5     case $2 in        //第 5~10 行是 case...in...esac 结构。该结构共分为 yang、zhang、li 和“*”
                        // (其他) 4 个分支。如果位置变量$2 的值为 yang 则执行第 6 行的 yang 分支，如果
                        //位置变量$2 的值为 zhang，则执行第 7 行的 zhang 分支，如果位置变量$2 的值为
                        //li，则执行第 8 行的 li 分支。如果都不是，则执行 9 行的其他分支
6     yang) echo "您好!yang 同学，您的成绩为 80 ";; //如果$2 的值为 yang，则执行该分支，
显示输
                        //出“您好!yang 同学，您的成绩为 80”
7     zhang) echo "您好!zhang 同学，您的成绩为 95 ";; //如果$2 的值为 zhang，则执行该分支，
显示
                        //输出“您好!zhang 同学，您的成绩为 95”
8     li) echo "您好!li 同学，您的成绩为 75 ";; //如果$2 的值为 li，则执行该分支，显
示输出
                        //“您好!li 同学，您的成绩为 75”
9     *) echo "对不起，还查不到您的名字! ";; //如果$2 的值不能匹配以上各分支，则执
行该
                        //分支，显示输出“对不起，还查不到您
的名字!”
10    esac             // case 结构中的 esac 语句，匹配第 5 行的 case，表示 case 结构结束
11    ;;               // “;;”表示第 3 行 case 结构中的 student 分支结束
12 teacher)           // teacher 分支，即$1 的值为 teacher 时，执行该分支
13    case $2 in        //第 13~20 行是 case...in...esac 结构。该结构共分为 tan、yu、feng、
                        //cao 和“*” (其他) 5 个分支。如果位置变量$2 的值为 tan 则执行第 14
                        //行的 tan 分支，如果位置变量$2 的值为 yu，则执行第 15 行的 yu 分支，
                        //如果位置变量$2 的值为 feng，则执行第 16 行的 feng 分支，如果位置变
                        //量$2 的值为 cao，则执行第 17 行的 cao 分支。如果都不是，则执行 18
                        //行的其他分支
14    tan) echo "您好!tan 老师，您的职称为讲师 ";; //如果$2 的值为 tan，则执行该分支，显示
                        //输出“您好!tan 老师，您的职称为讲师”
15    yu) echo "您好!yu 老师，您的职称为教授";; //如果$2 的值为 yu，则执行该分支，显示
                        //输出“您好!yu 老师，您的职称为教授”
16    feng) echo "您好!feng 老师，您的职称为副教授";; //如果$2 的值为 feng，则执行该分支，显示
                        //输出“您好!feng 老师，您的职称为副
教授”
17    cao) echo "您好!cao 老师，您的职称为助教 ";; //如果$2 的值为 cao，则执行该分支，显示
                        //输出“您好!cao 老师，您的职称为助教”
18    *) echo "对不起! 您的资料还未加入! ";; //如果$2 的值不能匹配以上各分支，则执行
                        //该分支，显示输出“对不起! 您的资料
还未加入!”
19    esac             // case 结构中的 esac 语句，匹配第 13 行的 case，表示 case 结构结束
20    ;;               // “;;”表示第 3 行 case 结构中的 teacher 分支结束
21    *) echo "参数不正确! ";; //第 3 行 case 结构中的其他分支，表示不能匹配 student 或 teacher
分支
                        //时，执行该分支，显示输出“参数不正确!”
22 esac               //匹配第 3 行 case 结构的 esac 语句，表示该 case 结构结束

```


该程序的具体执行过程如下所示。

```
#chmod u+x case2           //赋予文件 case2 可执行权限
# ./case2 student yang      //带参数“student”和“yang”执行程序 case2
您好!yang 同学, 您的成绩为 80
# ./case2 student li        //带参数“student”和“li”执行程序 case2
您好!li 同学, 您的成绩为 75
# ./case2 teacher cao       //带参数“teacher”和“cao”执行程序 case2
您好!cao 老师, 您的职称为助教
# ./case2 teacher yu        //带参数“teacher”和“yu”执行程序 case2
您好!yu 老师, 您的职称为教授
# ./case2 teacher zhang     //带参数“teacher”和“zhang”执行程序 case2
对不起! 您的资料还未加入!
# ./case2 student tan       //带参数“student”和“tan”执行程序 case2
对不起, 还查不到您的名字!
# ./case2 manager zhang     //带参数“manager”和“zhang”执行程序 case2
参数不正确!
```

18.3 select 结构

select 循环结构可以以一种更简便的方式生成菜单。生成的菜单具有编号，在输入选择的编号后，可根据用户的选择执行相应的代码。

18.3.1 select 结构的语法格式

使用 select 结构将会在屏幕上生成一个数字化的菜单，并提示用户进行选择，默认的提示符为“#”。用户只需在提示符下输入对应菜单项的数字，即可完成选择。select 结构的语法格式如下所示。

```
select 变量 in 列表
do
    命令表
done
```

其中，“列表”是一个由字符串组成的序列，在程序执行 select 时，该列表将以菜单形式进行显示，用户可以在其中进行选择。Shell 会在显示列表后，等待用户选项其中某一项。当用户选择了某一项之后，Shell 会把该选项赋值给“变量”，并执行 select 结构体中的命令。do...done 之间是 select 循环的循环体，do 表示循环开始，done 表示循环结束。

select 总是与 case 结构一起使用，允许用户在菜单中选择，并基于选择执行相应的命令。当执行到 break 语句时，将退出 select 结构。



select 结构是一种循环，可以使用 break 命令退出循环，也可以使用 exit 命令结束脚本。

18.3.2 select 结构示例

例如，编写一个 Shell 程序，根据用户的选择，显示一周的日程安排。其中，利用 select



结构生成显示菜单，提供给用户进行选择，代码如下所示（代码 selects）。

```

1 #!/bin/bash           //程序第1行的“#!/bin/bash”表示调用bash来执行脚本程序
2 #filename:selects     //程序第2行的“#filename: selects”表示该程序的名称为selects
3 echo "-----查询一周计划-----" //显示输出“-----查询一周计划-----”
4 select N in 周一 周二 周三 周四 周五 周六 周日 //第4~17行是select...in...do...done结构。执行该
//结构时将显示菜单，菜单项由“周一 周二 周三 周四 周
//五 周六 周日”组成。根据用户的选择，相应的选项将
//赋值给变量N
5 do //select结构中的do语句，表示select结构体开始
6 case $N in //第6~16行是case...in...esac结构。根据变量N的取值，跳转到相应的分支
//如果变量N的值为“周一”，则执行第7行。如果变量N的值为“周二”，则执行第
//8行。如果变量N的值为“周三”，则执行第9行。如果变量N的值为“周四”，则
//执行第10行。如果变量N的值为“周五”，则执行第11行。如果变量N的值为“周
//六”，则执行第12行。如果变量N的值为“周日”，则执行第13行。如果变量N的
//值为其他值，则执行第14行
7 周一) echo "呜呜，今天是星期一，是工作日。"; // case结构的“周一”分支。当变量N的值为
//“周一”时执行该分支，显示输入“呜呜，
//今天是星期一，是工作日。”
8 周二) echo "哈哈，今天是星期二，下午休息。"; //case结构的“周二”分支。当变量N的值为
//“周二”时执行该分支，显示输入“哈哈，今
//天是星期二，下午休息。”
9 周三) echo "呜呜，今天是星期三，是工作日。"; //case结构的“周三”分支。当变量N的值为
//“周三”时执行该分支，显示输入“呜呜，今
//天是星期三，是工作日。”
10 周四) echo "呜呜，今天是星期四，晚上还要加班。"; //case结构的“周四”分支。当变量N
的值为
//“周四”时执行该分支，显示输入“呜呜，今
//天是星期四，晚上还要加班。”
11 周五) echo "哈哈，今天是星期五，晚上有PARTY!"; //case结构的“周五”分支。当变量N的值为
//“周五”时执行该分支，显示输入“哈哈，今
//天是星期五，晚上有PARTY!”
12 周六) echo "呜呜，今天是星期六，但要加半天班!"; //case结构的“周六”分支。当变量N的值为
//“周六”时执行该分支，显示输入“呜呜，今
//天是星期六，但要加半天班!”
13 周日) echo "哈哈，今天是星期日，休息日!!!"; //case结构的“周日”分支。当变量N的值为
//“周日”时执行该分支，显示输入“哈哈，今
//天是星期日，休息日!!!”
14 *) echo "选择有误!!退出." //case结构的其他分支，即当变量N的值不是
//“周一”、“周二”...“周日”时执行该分支
//该分支包括14、15两行，本行显示输入
//“选择有误!!退出。”
15 break;; //case结构的其他分支，执行break命令，退
出select结构
16 esac // case结构中的esac语句，表示case结构
结束
17 done // select结构中的done语句，表示select结构体结束

```

该代码执行过程如下所示。

```

#chmod u+x selects //赋予文件selects可执行权限
#./selects //执行程序selects
-----查询一周计划-----
1) 周一
2) 周二
3) 周三
4) 周四

```



```

5) 周五
6) 周六
7) 周日
#? 1 //输入“1”
呜呜，今天是星期一，是工作日。
#? 2 //输入“2”
哈哈，今天是星期二，下午休息。
#? 3 //输入“3”
呜呜，今天是星期三，是工作日。
#? 4 //输入“4”
呜呜，今天是星期四，晚上还要加班。
?? 5 //输入“5”
哈哈，今天是星期五，晚上有 PARTY!
#? 6 //输入“6”
呜呜，今天是星期六，但要加半天班!
#? 7 //输入“7”
哈哈，今天是星期日，休息日!!!
#? 0 //输入“0”
选择有误!! 退出。
#

```

18.4 while 循环

循环用于反复执行一系列的语句，其中循环的次数依赖于一定的条件（或者是指定的次数，或者是符合特定的要求）。Shell 中的循环结构包括 `while`、`for`、`until` 等，其中最简单的循环形式是 `while`。`while` 循环也称当型循环。

18.4.1 while 语句的语法格式

与 `if` 语句相同，`while` 语句也需要设定一个条件。两者的区别在于，当条件为真时，`if` 语句只执行后续代码一次，而 `while` 循环只要条件为真，就会不断地重复执行后续的代码块。

通常，在无法确定循环的具体次数时，使用 `while` 循环语句；当确切知道循环需要执行的次数时，采用 `for` 循环。

`while` 循环的基本结构如下所示。

```

while 表达式
do
    命令表
done

```

其中，“表达式”为循环控制条件，在每一次循环开始前都会被检查。如果“表达式”的值为真，则执行 `do` 和 `done` 之间的代码，执行完毕后（即遇到“`done`”）会重新对“表达式”的值进行计算，如果仍为真，则再次进入循环体。如果“表达式”的值变为假，则循环结束。`do` 和 `done` 之前的命令表构成了 `while` 的循环体。

`while` 循环的具体执行步骤如下：首先执行“表达式”，若返回码为 0，则为真，进入循环体，执行“命令表”一次，然后返回执行“表达式”……直到“表达式”的返回码为非 0（即为假），循环结束，继续执行 `done` 语句之后的代码。`while` 循环结构的流程如图 18-6 所示。

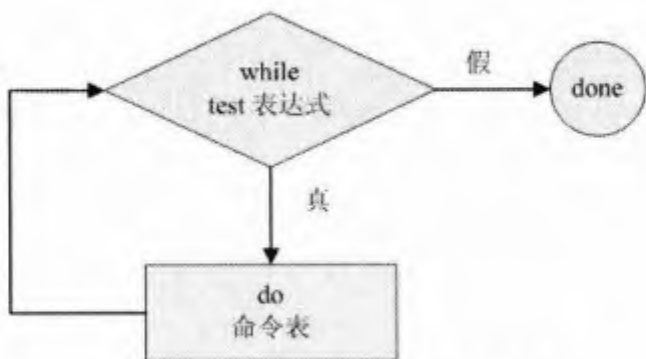


图 18-6 while 循环结构的流程图

18.4.2 while 循环示例

(1) 以下示例使用 while 结构显示输出数字 1~10（代码 1to10）。

```
1 #!/bin/bash           //程序第 1 行的“#!/bin/bash”表示调用 bash 来执行脚本程序
2 #filename:1to10       //程序第 2 行的“#filename: 1to10”表示该程序的名称为 1to10
3 i=1                   //定义变量 i，赋初值为 1
4 while [ $i -le 10 ]   //第 4~8 行为 while...do...done 结构。“$i -le 10”为循环控制条件，当
变量 i 的值小           //于等于 10 时，执行 while 循环体（第 6~7 行），否则退出 while 循环
5 do                     // while 循环结构中的 do 语句，表示循环体开始
6 echo "第 $i 次循环，输出： $i" //显示输入“第 $i 次循环，输出： $i”
7 i=$((i+1))            //变量 i 自加 1
8 done                  // while 循环结构中的 done 语句，表示循环体结束
```

该代码执行结果如下所示。

```
#chmod u+x 1to10 //赋予文件 1to10 执行权限
#./1to10         //执行程序 1to10
第 1 次循环，输出： 1
第 2 次循环，输出： 2
第 3 次循环，输出： 3
第 4 次循环，输出： 4
第 5 次循环，输出： 5
第 6 次循环，输出： 6
第 7 次循环，输出： 7
第 8 次循环，输出： 8
第 9 次循环，输出： 9
第 10 次循环，输出： 10
```

(2) 编写一个 Shell 程序，利用 while 循环计算 1~N 的平方值，N 值由用户输入。代码如下所示（代码 ii）。

```
1 #!/bin/bash           //程序第 1 行的“#!/bin/bash”表示调用 bash 来执行脚本程序
2 #filename:ii          //程序第 2 行的“#filename: ii”表示该程序的名称为 ii
3 echo "请输入一个数字： " //显示输出“请输入一个数字： ”
4 read N                //读入变量 N 的值
5 i=1                   //变量 i 赋值 1
6 while [ $i -le $N ]   //第 6~11 行是 while...do...done 循环。循环控制条件为“$i -le $N”，
即判                     //断变量 i 的值是否小于等于 N。如果变量 i 的值小于等于 N，则执行第 7~
11 行的                   //循环体，否则退出循环
7 do                     // while 循环结构中的 do 语句
8 RESULT='expr $i \* $i' //将变量 i 与变量 i 相乘，且将相乘后的结果保存到变量 RESULT 中，
```



```

//即变量 RESULT 中保存了变量 i 的平方值。其中 "expr $i \* $i" 使
//用了反引号，表示取得 "expr $i \* $i" 的执行结果
9 echo "$i-----$RESULT" //显示输出 "$i-----$RESULT"
10 i=$((i+1)) //变量 i 自加 1
11 done //while 循环结构中的 done 语句，表示循环体结束

```

执行程序 ii，过程如下所示。

```

#chmod u+x ii //赋予程序 ii 执行权限
#./ii //执行程序 ii
请输入一个数字:
5 //输入 "5"
1-----1
2-----4
3-----9
4-----16
5-----25
#./ii //执行程序 ii
请输入一个数字:
8 //输入 "8"
1-----1
2-----4
3-----9
4-----16
5-----25
6-----36
7-----49
8-----64

```

(3) 编写一个 while 无限循环程序，每隔 5s，输出“哈哈，这是一个无限循环！”，直到用户按下 Ctrl+c 组合键结束该程序为止。代码如下所示（代码 endless）。

```

1 #!/bin/bash //程序第 1 行的 "#!/bin/bash" 表示调用 bash 来执行脚本程序
2 #filename:endless //程序第 2 行的 "#filename:endless" 表示该程序的名称为 endless
3 while : //第 3~7 行是 while...do...done 循环结构。循环控制条件为空，即表示无限循环
4 do // while 结构中的 do 语句，表示循环体开始
5 echo "哈哈，这是一个无限循环！" //显示输出 "哈哈，这是一个无限循环！"
6 sleep 5 //等待 5 秒钟
7 done //while 结构中的 done 语句，表示循环体结束

```

执行该程序的过程如下所示。

```

#chmod u+x endless //设置文件 endless 执行权限
#./endless //执行程序 endless
哈哈，这是一个无限循环！
哈哈，这是一个无限循环！
哈哈，这是一个无限循环！
哈哈，这是一个无限循环！
哈哈，这是一个无限循环！
哈哈，这是一个无限循环！ //按 Ctrl+c 组合键结束 endless 程序
#

```

18.4.3 while 的嵌套

while 循环可以多层嵌套，即在 while 循环中嵌套 while 循环。例如，以下代码使用两层



while 循环，输出乘法表（代码 99）。

```

1 #!/bin/bash           //程序第 1 行的“#!/bin/bash”表示调用 bash 来执行脚本程序
2 #filename:99          //程序第 2 行的“#filename:99”表示该程序的名称为 99
3 i=1                   //变量 i 赋值 1
4 while [ $i -le 9 ]    //第 4~15 行是 while...do...done 循环。其中“$i -le 9”是循环控制条件，即表示
                        //每循环一次，判断变量 i 的值是否小于等于 9。如果变量 i 小于等于 9，则执行 6~14 行
                        //之间的循环体，否则退出 while 循环
5 do                    // while 结构中的 do 语句，与第 4 行的 while 相匹配，表示循环体的起始
6   j=1                 //变量 j 赋值 1
7   while [ $j -le $i ] //第 7~12 行是 while...do...done 循环。其中“$j -le $i”是循环控制条件，
即表示                //每循环一次，判断变量 j 的值是否小于等于变量 i 的值。如果变量 i 小于等于 j，
                        //第 9~11 行之间的循环体内容，否则退出 while 循环
                        // while 结构中的 do 语句，与第 7 行的 while 匹配，表示循环体的起始
8   do                  //
9     RESULT='expr $j \* $i' //将变量 i 与变量 j 相乘，且将相乘后的结果保存到变量 RESULT 中
                        //其中“'expr $j \* $i'”使用了反引号，表示取得“expr $j \*
$i”的执行结果
10    echo -n "$j*$i=$RESULT " //显示输出“$j*$i=$RESULT”。选项“-n”表示输出后不自动换行
11    j=$((j+1))          //变量 j 自加 1
12  done                // while 结构中的 done 语句，与第 7 行的 while 匹配，表示循环体
的结束
13  echo ""              //换行
14  i=$((i+1))           //变量 i 自加 1
15 done                 // while 结构中的 done 语句，与第 4 行的 while 匹配，表示循环体
的结束

```

该程序的执行过程如下所示。

```

# chmod u+x 99           //赋予文件 99 执行权限
# ./99                   //执行程序 99
1*1=1
1*2=2 2*2=4
1*3=3 2*3=6 3*3=9
1*4=4 2*4=8 3*4=12 4*4=16
1*5=5 2*5=10 3*5=15 4*5=20 5*5=25
1*6=6 2*6=12 3*6=18 4*6=24 5*6=30 6*6=36
1*7=7 2*7=14 3*7=21 4*7=28 5*7=35 6*7=42 7*7=49
1*8=8 2*8=16 3*8=24 4*8=32 5*8=40 6*8=48 7*8=56 8*8=64
1*9=9 2*9=18 3*9=27 4*9=36 5*9=45 6*9=54 7*9=63 8*9=72 9*9=81

```

18.4.4 避免死循环

如果 while 循环的条件一开始就为假，那么程序就会直接跳过 while 循环，不会执行循环体中的任何代码，即 while 循环中的代码一次也未被执行。例如，以下代码中的 while 循环不被执行。

```

1 j=1
2 while [ $j -le 0 ]
3 do
4   echo "$j*$i=$RESULT"
5 done
6 echo "Finish!"

```


其中，第 1 行给变量 `j` 赋值 1。由于第 2 行 `while` 循环的判断条件 “`$j -le 0`” 无法满足，程序会跳过整个 `while` 循环，继续执行第 6 行。

如果 `while` 循环的条件一开始为真，则程序会至少执行循环体中的代码一次，当条件变为假时退出循环。但如果 `while` 的循环控制条件永远为真，那么程序就一遍遍地执行，永不结束，即程序进入了死循环（endless loop）。例如，以下代码中的 `while` 循环无法自动结束，程序将进入死循环状态。

```
1 j=-1
2 while [ $j -le 0 ]
3 do
4   echo "$j*$i=$RESULT"
5 done
6 echo "Finish!"
```

其中，第 1 行给变量 `j` 赋值 -1。由于第 2 行 `while` 循环的判断条件 “`$j -le 0`” 始终满足，程序会一遍遍地执行 `while` 循环，无法跳出，更无法继续执行第 6 行代码。

在使用循环结构时，应避免死循环，因此在循环体中至少要有一行修改 `while` 循环控制条件的代码。例如，下面程序中循环读入用户的输入，直到用户输入字符串 “quit” 为止，具体代码如下所示（代码 `wquit`）。

```
1 #!/bin/bash           //程序第 1 行的 “#!/bin/bash” 表示调用 bash 来执行脚本程序
2 #filename:wquit       //程序第 2 行的 “#filename: wquit” 表示该程序的名称为 wquit
3 echo "-----"       //显示输出分隔行
4 echo "1.Step 1 "      //显示输出 “1.Step 1”
5 echo "2.Step 2 "      //显示输出 “2.Step 2”
6 echo "Please enter your choice:" //显示输出 “Please enter your choice:”
7 read INPUT            //读入变量 INPUT
8 while [ $INPUT != "quit" ] //第 8~15 行为 while 循环，其中循环控制条件为 “$INPUT != "quit"”
                           //即当变量 INPUT 的值为 “quit” 时，退出 while 循环，否则继续执行 while 循环
9 do                    // while 循环结构中的 do 语句
10  echo "-----"      //显示输出分隔行
11  echo "1.Step 1 "    //显示输出 “1.Step 1”
12  echo "2.Step 2 "    //显示输出 “2.Step 2”
13  echo "Please enter your choice:" //显示输出 “Please enter your choice:”
14  read INPUT          //读入变量 INPUT
15 done                // while 结构中的 done 语句，表示 while 结构结束
```

该程序的执行过程如下所示。

```
# chmod u+x wquit      //赋予文件 wquit 执行权限
# ./wquit              //执行程序 wquit
-----
1.Step 1
2.Step 2
Please enter your choice:
1                      //输入 “1”
-----
1.Step 1
2.Step 2
Please enter your choice:
2                      //输入 “2”
```



```
-----
1.Step 1
2.Step 2
Please enter your choice:
q                               //输入“q”
-----
1.Step 1
2.Step 2
Please enter your choice:
quit                            //输入“quit”
#
```

可以看到，当用户输入“quit”时，变量 INPUT 的值为“quit”，while 循环的控制条件也将变为假，循环结束退出。



程序进入死循环后，可以通过按 Ctrl+c 组合键强制结束程序的运行。

18.5 for 循环

有些高级语言如 Visual Basic 使用 for...next 循环，可以让程序员设定循环执行的次数。但 Bash Shell 中没有 for...next 循环，而其具有的 for...in 循环也没有该功能。Bash 的控制结构 for...in 与 TC Shell 中的 foreach 类似，可以根据列表确定执行循环的次数。

18.5.1 for 循环语法结构

事实上，所有能用 for 结构完成的循环，都可以用 while 来实现，但如果是通过一个列表来控制循环的次数，则采用 for 循环结构更简洁，代码也更紧凑。for 结构的语法格式如下所示。

```
for 变量 [ in 列表 ]
do
命令表
done
```

其中各项说明如下。

- “列表”通常由一系统以空格为分隔符的字符串组成。
- “列表”中的字符串将依次赋值给“变量”，每次赋值后将执行“命令表”中的命令。循环将在取完“列表”中的所有参数时停止。
- do...done 为 for 循环的循环体，每次循环将执行循环体中的命令。do 表示循环体的开始，done 表示循环体结束。
- 结束循环后，将继续执行 done 后的代码。
- 如果省略了“in 列表”，则 for 会将当前执行脚本中的每个位置参数作为“列表”，一次执行列表中的一个，即“for 变量”隐含表示“for 变量 \$@”。

for 循环结构的流程如图 18-7 所示。

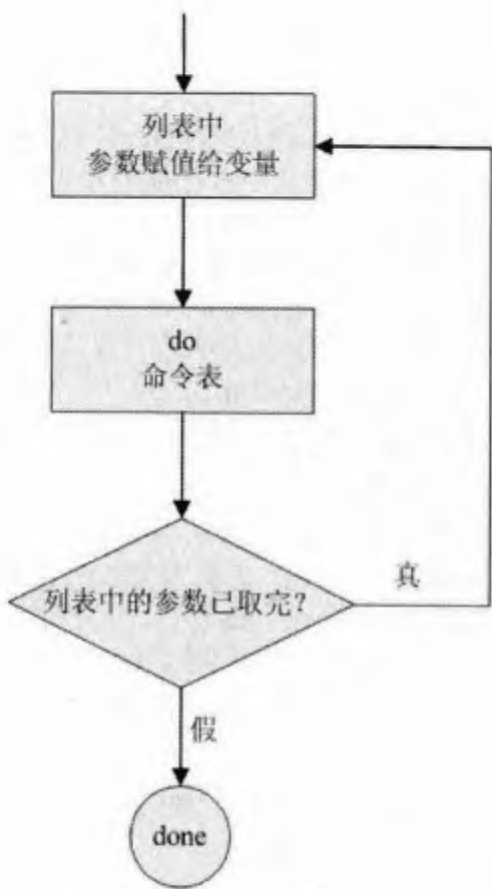


图 18-7 for 循环结构流程图

18.5.2 for 循环示例

(1) 编写一个 Shell 程序，使用 for 循环计算 1~6 的平方值，代码如下所示（代码 1-6）。

```
1 #!/bin/bash          //程序第 1 行的“#!/bin/bash”表示调用 bash 来执行脚本程序
2 #filename:1-6        //程序第 2 行的“#filename:1-6”表示该程序的名称为 1-6
3 for i in 1 2 3 4 5 6 //第 3~7 行是 for 循环结构。每次循环，循环变量 i 依次取值 1、2、3、4、5、6
4 do                  // for 循环中的 do 结构，表示循环体开始
5   RESULT='expr $i \* $i' //将变量 i 与变量 i 相乘，且将相乘后的结果保存到变量 RESULT 中
                        //即变量 RESULT 中保存了变量 i 的平方值。其中“'expr $i \* $i'”使
                        //用了反引号，表示取得“expr $i \* $i”的执行结果
6   echo "$i-----$RESULT" //显示输出“$i-----$RESULT”
7 done              //for 循环中的 done 结构，表示循环体结束
```

该程序的执行过程如下所示。

```
#chmod u+x 1-6          //赋予文件 1-6 执行权限
#./1-6                  //执行程序 1-6
1-----1
2-----4
3-----9
4-----16
5-----25
6-----36
```

(2) 编写一个 Shell 程序，将当前目录下所有后缀为 exe 的文件复制到目录/test 下，程序代码如下所示（代码 cpexe）。

```
1 #!/bin/bash          //程序第 1 行的“#!/bin/bash”表示调用 bash 来执行脚本程序
2 #filename:cpexe      //程序第 2 行的“#filename:cpexe”表示该程序的名称为 cpexe
3 for FILE in *.exe    //第 3~6 行是 for 循环结构。每次循环，循环变量 FILE 依次取得当前目录下的
                        //一个后缀为 exe 的文件名，直到全部取尽
4 do                  //for 循环中的 do 结构，表示循环体开始
```



```
5 cp $FILE /test //使用 cp 命令将变量 FILE 表示的文件复制到目录/test 中
6 done           //for 循环中的 done 结构，表示循环体结束
```

该程序代码执行过程如下所示。

```
# ls /test           //显示目录 test 的内容，目录 test 为空
# ls *.exe           //显示当前目录中所有的 exe 文件
bubble.exe homelist.exe sort.exe student.exe
#chmod u+x cpexe     //赋予文件 cpexe 可执行权限
#./cpexe             //执行程序 cpexe
# ls /test           //显示目录 test 的内容，目录中已存在 exe 文件
bubble.exe homelist.exe sort.exe student.exe
```

(3) 编写一个 Shell 程序，使用 for 循环在屏幕显示字母 A~E，代码如下所示（代码 A-E）。

```
1 #!/bin/bash        //程序第 1 行的“#!/bin/bash”表示调用 bash 来执行脚本程序
2 #filename:A-E       //程序第 2 行的“#filename:A-E”表示该程序的名称为 A-E
3 for I in A B C D E  //第 3~6 行是 for 循环结构。每次循环，循环变量 I 依次取值 A、B、C、D、E、F
4 do                 //for 循环中的 do 结构，表示循环体开始
5   echo "请选择答案$I" //显示输出“请选择答案$I”
6 done              //for 循环中的 done 结构，表示循环体结束
```

该程序代码执行过程如下所示。

```
#chmod u+x A-E       //赋予文件 A-E 执行权限
#./A-E               //执行程序 A-E
请选择答案 A
请选择答案 B
请选择答案 C
请选择答案 D
请选择答案 E
```

(4) 编写一个 Shell 程序，读入一组数字，使用 for 循环计算该组数字的乘积，具体代码如下所示（代码 multi）。

```
1 #!/bin/bash        //程序第 1 行的“#!/bin/bash”表示调用 bash 来执行脚本程序
2 #filename:multi     //程序第 2 行的“#filename: multi”表示该程序的名称为 multi
3 echo "请输入计算乘积的数字：" //显示输出“请输入计算乘积的数字：”
4 read VAL            //读入用户输入的数据，且保存到变量 VAL 中
5 RESULT=1            //变量 RESULT 赋值为 1
6 for I in $VAL        //第 6~9 行是 for 循环结构。每次循环，循环变量 I 依次取得变量
                       //RESULT 中的各个参数值
7 do                 // for 循环中的 do 结构，表示循环体开始
8   RESULT='expr $RESULT \* $I' //将变量 RESULT 与变量 I 相乘，且将相乘后的结果保存到变量 RESULT
                               //中。其中“'expr $RESULT \* $I ’”使用了反引号，表示取得
                               //“expr $RESULT \* $I”的执行结果
9 done              // for 循环中的 done 结构，表示循环体结束
10 echo "计算的结果为：$RESULT" //显示输出“计算的结果为：$RESULT”
```

该程序的执行过程如下所示。

```
#chmod u+x multi     //赋予文件 multi 可执行权限
#./multi             //执行程序 multi
请输入计算乘积的数字：
1 3 4 5 7            //输入一组数字“1 3 4 5 7”
计算的结果为：420
#./multi             //执行程序 multi
请输入计算乘积的数字：
34 12 78 45 78      //输入一组数字“34 12 78 45 78”
计算的结果为：111702240
```

(5) 下面程序中的 for 循环省略了“in 列表”，for 循环会将当前执行脚本中的每个位置参数作为“列表”，一次执行列表中的一个（代码 fornoin）。


```

1 #!/bin/bash
脚本程序
2 #filename: fornoin
名称为 fornoin
3 i=1
4 for arg
5 do
6   echo " 第 $i 个位置参数为: $arg "
7   i=$((i+1))
8 done

```

//程序第 1 行的“#!/bin/bash”表示调用 bash 来执行
 //程序第 2 行的“#filename: fornoin”表示该程序的
 //变量赋值 1
 //第 4~8 行为 for 循环。循环使用的列表来自位置参数
 //for 循环中的 do 语句
 //显示输出“第 \$i 个位置参数为: \$arg”
 //变量 i 加 1
 //for 循环中的 done 语句

该代码执行结果如下所示。

```

#chmod u+x fornoin
#./fornoin aaa bbb ccc ddd eee
第 1 个位置参数为: aaa
第 2 个位置参数为: bbb
第 3 个位置参数为: ccc
第 4 个位置参数为: ddd
第 5 个位置参数为: eee

```

//赋予文件 fornoin 执行权限
 //执行程序 fornoin

18.6 until 循环

until 循环也称为直到型循环。until 循环和 while 循环的区别在于，while 循环在条件为真时继续执行循环，而 until 则是在条件为假时继续执行循环，在条件为真时，停止执行。TC Shell 中没有 until 循环。

18.6.1 until 循环的语法结构

由于 until 结构具有特殊的执行顺序，比较适合编写菜单程序。until 循环的语法结构如下所示。

```

until
命令表 1
test 表达式
do
    命令表 2
done

```

其中各项说明如下。

- “命令表 1”在循环开始前执行，且在每一次循环结束后，再次执行。
- “表达式”为循环控制条件，在表达式为真时，结束循环，表达式为假时，继续执行循环。
- do...done 之间为 until 循环的循环体。每次循环时将执行其中的“命令表 2”。
- until 循环结束后，将执行 done 后的语句。

until 循环的具体执行步骤为：首先执行命令表 1，并检测“表达式”的值，若表达式返回值为非 0，则执行循环体命令表 2 一次，然后返回再次执行命令表 1，并再次检测“表达式”的值……如此往复，直到“表达式”的返回值为 0，循环结束。until 结构的流程如图 18-8 所示。

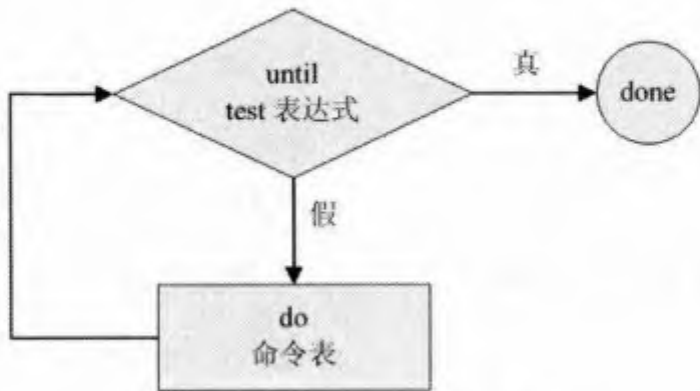


图 18-8 until 结构流程图

18.6.2 until 循环示例

例如，编写一个 Shell 程序，读入用户输入的 N 值，使用 until 循环计算 1~N 的平方值，代码如下所示（代码 untilN）。

```
1 #!/bin/bash //程序第 1 行的“#!/bin/bash”表示调用 bash 来执行脚本程序
2 #filename:untilN //程序第 2 行的“#filename: untilN”表示该程序的名称为 untilN
3 i=1 //变量 i 赋值 1
4 echo "请输入 N 值: " //显示输出“请输入 N 值: ”
5 read N //读入变量 N 的值
6 until //第 6~12 行是 until 循环结构
7 test $i -gt $N //每次循环检测变量 i 是否大于变量 N。如果变量 i 大于变量 N，则结束循环，
否则执 //行循环第 9~11 行
8 do //until 循环结构中的 do 语句
9 RESULT='expr $i \* $i' //将变量 i 与变量 i 相乘，且将相乘后的结果保存到变量 RESULT 中，
//即变量 RESULT 中保存了变量 i 的平方值。其中“'expr $i \* $i'”使
//用了反引号，表示取得“expr $i \* $i”的执行结果
10 echo "$i-----$RESULT" //显示输出“$i-----$RESULT”
11 i=$((i+1)) //变量 i 自加 1
12 done //until 结构中的 done 语句，表示循环结构体结束
```

该程序的执行过程如下所示。

```
#chmod u+x untilsquare //赋予文件 untilsquare 执行权限
#./untilsquare //执行程序 untilsquare
请输入 N 值:
7 //输入值“7”
1-----1
2-----4
3-----9
4-----16
5-----25
6-----36
7-----49
#./untilsquare //执行程序 untilsquare
请输入 N 值:
5 //输入值“5”
1-----1
2-----4
3-----9
4-----16
5-----25
```


18.7 break 和 continue 语句

在流程控制语句中，break 和 continue 是两个比较重要的语句，都可以对程序的走向实施控制。使用 break 语句可以退出循环，使用 continue 语句可以跳过本次循环中剩余的代码，继续下一次循环。

18.7.1 break 语句

使用 break 语句可以结束 while、for、until 或 select 等结构的执行，即从结构中跳出。退出循环后，将转到 done 语句后继续执行。

(1) 例如，编写一个 Shell 程序，根据用户的输入数字，使用 break 语句退出循环。该程序的具体代码如下所示（代码 breaks）。

```
1 #!/bin/bash           //程序第1行的“#!/bin/bash”表示调用bash来执行脚本程序
2 #filename:breaks      //程序第2行的“#filename:breaks”表示该程序的名称为breaks
3 echo "请输入数字N: "  //显示输出“请输入数字N: ”
4 read N                //读入用户的输入，且保存到变量N中
5 for i in 1 2 3 4 5 6 7 8 9 //第5~13行是for...in...do...done循环结构。每次循环，变量i将依次
取值1~9
6 do                    // for 循环结构中的do语句
7     if [ $i -eq $N ];then //第7~12行是if...then...else...fi结构。其中“$i -eq $N”用于
//判断变量i是否等于变量N。如果变量i等于变量N，则执行第8~9行，
//否则执行第11行
8         echo "-----退出for循环-----" //显示输出“-----退出for循环-----”
9         break //使用break语句，退出for循环
10        else //if结构中的else结构
11            echo "----当前是第$i次循环----" //显示输出“----当前是第$i次循环----”
12        fi //if结构中的fi语句，表示if结构结束
13 done //for结构中的done语句，表示for循环体结束
```

该程序的执行过程如下所示。

```
#chmod u+x breaks //赋予文件breaks执行权限
#./breaks //执行程序breaks
请输入数字N:
7 //输入“7”
----当前是第1次循环----
----当前是第2次循环----
----当前是第3次循环----
----当前是第4次循环----
----当前是第5次循环----
----当前是第6次循环----
-----退出for循环-----
#./breaks //执行程序breaks
请输入数字N:
4 //输入“4”
----当前是第1次循环----
----当前是第2次循环----
----当前是第3次循环----
-----退出for循环-----
```




(2) 例如下面程序中, 在两层 while 循环中都使用了 break 语句。内层 while 循环中的 break 语句在变量 j 等于 5 时退出内层循环, 外层 while 循环中的 break 语句在变量 i 等于 2 时退出外层循环。程序代码如下所示 (代码 wbreak)。

```

1 #!/bin/bash           //程序第 1 行的“#!/bin/bash”表示调用 bash 来执行脚本程序
2 #filename:wbreak      //程序第 2 行的“#filename: wbreak”表示该程序的名称为 wbreak
3 i=1                   //变量 i 赋值 1
4 while [ $i -lt 10 ]   //第 4~20 行是 while...do...done 结构。循环控制条件为“$i -lt 10”, 即变量
i 小于
                        //10 时, 执行 while 循环体第 6~19 行, 否则结束 while 循环
5 do                     //while 结构中的 do 语句, 匹配第 4 行的 while
6     echo "-----外层第$i 次循环 -----" //显示输出“-----外层第$i 次循环
-----”
7     j=1                //变量 j 赋值为 1
8     while [ $j -lt 10 ] //第 8~15 行为 while...do...done 结构。循环控制条
                        //件为“$j -lt 10”, 即变量 j 小于 10 时, 执行 while
循环
                        //体第 10~14 行, 否则结束 while 循环
9     do                 // while 结构中的 do 语句, 匹配第 8 行的 while
10    if [ $j -eq 5 ];then //第 10~12 行为 if...then...fi 结构。“$j -eq 5”为判断
                        //条件, 即当变量 i 等于 5 时, 执行第 11 行,
                        //否则转到第 13 行
11        break          //break 语句, 用于结束内层 while 循环
12    fi                 //if 结构中的 fi 语句, 匹配第 10 行的 if, 表示 if 结构结束
13    echo "    ---内层第$j 次循环 ----" //显示输出“    ---内层第$j 次循环 ----”
14    j=$((j+1))         //变量 j 自加 1
15    done               // while 循环结构中的 done 语句, 匹配第 8 行的 while,
表示循环体结束
16    if [ $i -eq 2 ];then //第 16~18 行为 if...then...fi 结构。“$j -eq 2”为判
断条件, 即当
                        //变量 i 等于 2 时, 执行第 17 行, 否则转到第 19 行
17        break          //break 语句, 用于结束外层 while 循环
18    fi                 //if 结构中的 fi 语句, 匹配第 16 行的 if, 表示 if 结构结束
19    i=$((i+1))         //变量 i 自加 1
20 done                 // while 循环结构中的 done 语句, 匹配第 4 行的 while, 表示循环体结束

```

该代码执行过程如下所示。

```

#chmod u+x wbreak      //赋予文件 wbreak 执行权限
#./wbreak               //执行程序 wbreak
-----外层第 1 次循环 -----
    ---内层第 1 次循环 ----
    ---内层第 2 次循环 ----
    ---内层第 3 次循环 ----
    ---内层第 4 次循环 ----
-----外层第 2 次循环 -----
    ---内层第 1 次循环 ----
    ---内层第 2 次循环 ----
    ---内层第 3 次循环 ----
    ---内层第 4 次循环 ----

```

(3) 如果有多层结构嵌套, 还可以在 break 关键字后面加上数字, 指明要跳出的结构的层数。单独使用 break 语句就是默认从当前结构中跳出, 即相当于“break 1”。例如, 下面代码修改了上例中的 wbreak 程序, 将 break 语句跳出的层数修改为 2, 代码如下所示 (代码 wbreak2)。

```

1 #!/bin/bash
2 #filename:wbreak2
3 i=1

```



```

4 while [ $i -lt 10 ]
5 do
6     echo "-----外层第$i 次循环 -----"
7     j=1
8     while [ $j -lt 10 ]
9     do
10         if [ $j -eq 5 ];then
11             break 2
12         fi
13         echo "      ---内层第$j 次循环 ----"
14         j=$((j+1))
15     done
16     if [ $i -eq 2 ];then
17         break
18     fi
19     i=$((i+1))
20 done

```

该程序的执行结果如下所示。

```

#./wbreak2
-----外层第 1 次循环 -----
      ---内层第 1 次循环 ----
      ---内层第 2 次循环 ----
      ---内层第 3 次循环 ----
      ---内层第 4 次循环 ----

```

可以看到，在变量 j 等于 5 时，两层 while 循环均被跳过。



使用 break 语句可以将深埋在嵌套循环中的语句退出到指定层数，或者直接退出到最外层。

18.7.2 continue 语句

continue 语句用来跳过本次循环中剩余的代码，即直接跳回到循环的开始位置（条件判断处）。如果条件为真则开始下一次循环，否则退出循环。continue 语句的实际作用是转到下一轮循环。

(1) 例如，编写 Shell 程序输出数字 1~9，并通过 continue 语句跳过对指定数字的输出，代码如下所示（代码 continuefor）。

1 #!/bin/bash	//程序第 1 行的“#!/bin/bash”表示调用 bash 来执行脚本程序
2 #filename:continuefor	//程序第 2 行的“#filename: continuefor”表示该程序的名称为
continuefor	
3 echo "输入跳过的数字: "	//显示输出“输入跳过的数字: ”
4 read N	//读入变量 N
5 echo "-----"	//显示分隔行
6 i=1	//定义变量 i，且赋值 1
7 for i in 1 2 3 4 5 6 7 8 9	//第 7~14 行是 for...do...done 结构。每次循环
	//变量 i 依次取值“1”、“2”、“3”...“9”
8 do	// for 结构中的 do 语句
9 if [\$i -eq \$N];then	//第 9~12 行是 if...then...fi 结构。其中“\$i -eq \$N”是判断条件
	//当变量 i 的值等于变量 N 时，执行第 10、11 行，否则跳过 if 结构，
执行第 13 行	
10 echo "-"	//显示输出“-”
11 continue	// continue 语句跳过本次循环



```
12 fi // if 结构中的 fi 语句
13 echo "$i" //显示输出变量 i 的值
14 done // for 循环中的 done 语句，表示 for 循环体结束
```

该代码执行过程如下所示。

```
#chmod u+x continuefor //赋予文件 continuefor 执行权限
#./continuefor //执行程序 continuefor
输入跳过的数字:
4 //输入“4”
-----
1
2
3
- //跳过 4
5
6
7
8
9
#./continuefor //执行程序 continuefor
输入跳过的数字:
8 //输入“8”
-----
1
2
3
4
5
6
7
- //跳过 8
9
```

(2) 例如，以下代码实现输出小于 N 的奇数，N 由用户输入，其中利用 `continue` 语句跳过输出偶数的情况，代码如下所示（代码 `continues`）。

```
1 #!/bin/bash //程序第 1 行的“#!/bin/bash”表示调用 bash 来执行脚本程序
2 #filename:continues //程序第 2 行的“#filename: continues”表示该程序的名称为 continues
3 echo "输入 N 值，显示小于 N 的奇数：" //显示输出“输入 N 值，显示小于 N 的奇数：”
4 read N //读入变量 N 的值
5 echo "-----" //显示输出分隔行
6 i=0 //变量 i 赋值 0
7 while [ $i -lt $N ] //第 7~15 行是 while...do...done 循环结构。循环控制条件为
//“$i -lt $N”，即当变量 i 的值小于变量 N 时，执行 while
循环体
8 do //第 9~14 行，否则跳出 while 循环
//while 循环结构中的 do 语句
9 RESULT='expr $i \% 2' //将变量 i 与 2 取模（取余），且将取模后的结果保存到变量
RESULT
//中，即变量 RESULT 中保存了变量 i 与 2 的模。其中
//“'expr $i \% 2'”使用了反引号，表示取得“expr $i \% 2”
//的执行结果
10 i=$((i+1)) //变量 i 自加 1
11 if [ $RESULT -ne 0 ];then //第 11~13 行是 if...then...fi 结构。判断条件为“$RESULT -ne 0”，
//即当变量 RESULT 的值不等于 0 时，执行第 12 行，跳过本次 while
//循环，否则继续执行第 14 行
12 continue //使用 continue 语句跳过 while 循环
```



```

13 fi                                //if 结构中的 fi 语句
14 echo "$i"                          //显示输出变量 i 的值
15 done                               //while 循环结构中的 done 语句

```

执行该程序，过程如下所示。

```

#chmod u+x continues                //赋予文件 continues 可执行权限
#./continues                        //执行程序 continues
输入 N 值，显示小于 N 的奇数：
10                                  //输入 "10"
-----
1
3
5
7
9
#./continues                        //执行程序 continues
输入 N 值，显示小于 N 的奇数：
7                                  //输入 "7"
-----
1
3
5
7

```

(3) 与 break 一样，continue 语句也可以接受一个数字来决定跳出几层循环。例如，以下代码（代码 3wcontinue）共有三层 while 循环，在不使用 continue 语句的情况下，循环只能逐层退出，如下所示（代码 3wcontinue）。

```

1 #!/bin/bash                      //程序第 1 行的“#!/bin/bash”表示调用 bash 来执行脚本程序
2 #filename:3wcontinue             //程序第 2 行的“#filename: 3wcontinue”表示该程序的名称为
3wcontinue
3 i=1                              //定义变量 i，赋值 1
4 while [ $i -lt 10 ]              //第 4~20 行是 while...do...done 结构。循环控制条件为“$i -lt 10”，
                                     //即如果变量 i 小于 10 则执行第 6~19 行，否则退出循环
5 do                                //while 结构中的 do 语句，匹配第 4 行的 while
6     echo "-----进入第一层循环 -----" // 显示输出 “----- 进入第一层循环 -----”
7     while [ $i -lt 10 ]           //第 7~18 行是 while...do...done 结构。循环控
                                     //制条件为“$i -lt 10”，即如果变量 i 小于 10 则执行
                                     //第 9~17 行，否则退出循环
8     do                            //while 结构中的 do 语句，匹配第 7 行的 while
9         echo "-----进入第二层循环 -----" //显示输出“-----进入第二层循环 -----”
10        while [ $i -lt 10 ]        //第 10~16 行是 while...do...done 结构。循环控
                                     //制条件为“$i -lt 10”，即如果变量 i 小于 10 则执行
                                     //第 12~15 行，否则退出循环
11        do                        //while 结构中的 do 语句，匹配第 10 行的 while
12            i=10;                 //变量 i 赋值 10
13            echo "-----进入第三层循环 -----" //显示输出“-----进入第三层循环 -----”
14            echo "-----" //显示输出分隔行
15            echo "-----离开第三层-----" // 显示输出 “----- 离开第三层 -----”
16        done                     // while 结构中的 done 语句，匹配第 10 行
                                     的 while
17        echo "-----离开第二层-----" // 显示输出 “----- 离开第二层 -----”

```




```

18     done                                // while 结构中的 done 语句，匹配第 7 行的
while
19     echo "-----离开第一层-----"      // 显示输出 "-----离开第一层
-----"
20 done                                    // while 结构中的 done 语句，匹配第 4 行的
while

```

该代码的执行结果如下所示。

```

#chmod u+x 3continue                      //赋予文件 3continue 执行权限
#./3wcontinue                             //执行程序 3wcontinue
-----进入第一层循环 -----
-----进入第二层循环 -----
-----进入第三层循环-----
-----
-----离开第三层-----
-----离开第二层-----
-----离开第一层-----

```

如果在其中第三层循环使用语句“continue 2”，则可以跳出二层循环，直接跳到第一层循环的结尾部分继续执行（代码 3wcontinue2），如下所示。

```

1 #!/bin/bash
2 #filename:3wcontinue2
3 i=1
4 while [ $i -lt 10 ]
5 do
6     echo "-----进入第一层循环 -----"
7     while [ $i -lt 10 ]
8     do
9         echo "-----进入第二层循环 -----"
10        while [ $i -lt 10 ]
11        do
12            i=10;
13            echo "-----进入第三层循环----- "
14            echo "-----"
15            continue 2
16            echo "-----离开第三层-----"
17        done
18        echo "-----离开第二层-----"
19    done
20    echo "-----离开第一层-----"
21 done

```

该代码执行结果如下所示。

```

#chmod u+x 3continue2                    //赋予文件 3continue2 执行权限
#./3continue2                             //执行程序 3continue2
-----进入第一层循环 -----
-----进入第二层循环 -----
-----进入第三层循环-----
-----
-----离开第一层-----

```

可以看到，代码第 15 行中的“continue 2”跳出了内层的两层循环，继续执行第一层循环的后半部分。如果将代码第 15 行的“continue 2”修改为“continue”，即只跳出一层循环，则代码执行结果如下所示。

```

#./3continue2                             //执行程序 3continue2
-----进入第一层循环 -----

```


-----进入第二层循环-----

-----进入第三层循环-----

-----离开第二层-----

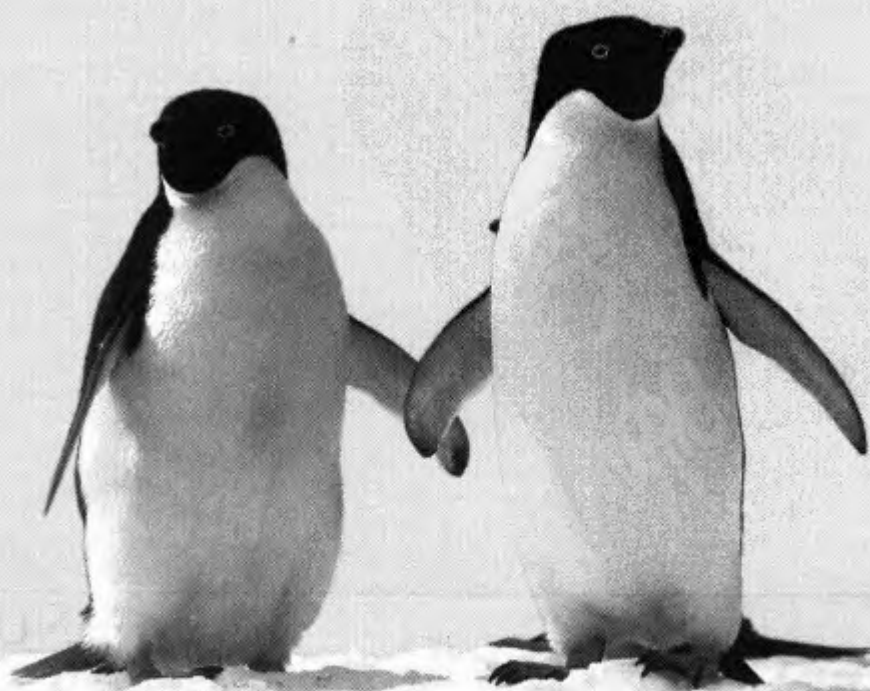
-----离开第一层-----

UNIX

第19章 函数

函数是从 ATT 的 UNIX System V R2 版本开始引入到 Bourne Shell 中的，并在 Bourne Again Shell (bash) 中得到进一步完善和加强。函数就是一个命令或一组命令的名字。函数可以使程序模块化，从而提高效率。

Shell 将函数存放在内存而不是磁盘文件中，所以访问函数的速度要比访问脚本的速度快得多。由于 Shell 还对函数进行预处理，因此其启动速度也要比脚本快得多。



19.1 函数的定义

Shell 函数类似于 Shell 脚本，里面存放了一系列的命令。在一个程序中可以创建许多函数，在使用一个函数之前，必须先程序中创建该函数。创建一个函数也称定义函数或函数的声明，定义函数的格式如下所示。

```
[function] 函数名
{
    命令表
[return]
}
```

其中各项说明如下。

- 关键字“function”表示定义一个函数，可以省略，其后是“函数名”（有时函数名称后面有一对空的括号，但非必需）。
- 符号“{”表示函数执行命令的入口，该符号也可以放在“函数名”那一行。“}”表示函数体结束，其中“{”与“}”之间的命令表为函数体。
- 命令表由调用该函数时将要执行的命令列表构成，可以是任意的 Shell 命令，包括调用其他函数。
- 函数中的“return”用于返回函数中最后一个命令的退出状态值或给定的参数值。
- 如果在函数中使用 exit 命令，则可以退出整个脚本。如果函数退出，就返回到脚本中调用该函数的地方。
- 可以在函数中使用 break 语句来中断函数的执行。
- 使用内部命令“declare -f”可以显示定义的函数清单。如果只显示函数的名字，可以用命令“declare -F”。
- 可以使用命令“export -f”将函数输出给 Shell。
- 可以使用命令“unset -f”从 Shell 内存中删除函数。

Shell 函数的定义可以放在“~/.bash_profile”文件中，也可以放在使用该函数的脚本中，还可以直接放在命令行中。可以使用内部的 unset 命令删除函数。一旦用户注销，Shell 将不再保持这些函数。

19.2 函数的调用

调用函数涉及函数的执行、函数参数传递、函数返回值、载入函数以及函数的删除等问题。函数在被执行前必须首先被定义。

19.2.1 执行函数

Shell 在执行一个别名、函数、内部命令或基于磁盘的可执行文件时，总是先执行别名，



然后是函数、内部命令，最后才是可执行程序。执行函数只需直接输入函数名即可。

例如，下面在命令行中定义了函数 `showwho`，并直接在命令行中执行该函数。

```
# function showwho ( )
> {
>   date
>   echo "当前登录的用户名: "
>   echo $LOGNAME
> }
# showwho
四  8月 21 06:19:22 CST 2008
当前登录的用户名:
root
```

通过命令行定义的函数在用户退出登录时将自动被删除。如果希望函数总是可用，而不是每次登录时都要重新输入，可以将函数的定义放在“`~/.bash_profile`”中，然后通过“`.`”命令或“`source`”执行 `.bash_profile` 文件，使修改能够立即生效。命令如下所示。

```
# cat ~/.bash_profile                                //显示输出~/.bash_profile 文件的内容
# .bash_profile

# Get the aliases and functions
if [ -f ~/.bashrc ]; then
    . ~/.bashrc
fi

# User specific environment and startup programs

PATH=$PATH:$HOME/bin
BASH_ENV=$HOME/.bashrc
USERNAME="root"

export USERNAME BASH_ENV PATH
function showwho ( )                                //添加 showwho 函数定义
{
    date
    echo "当前登录的用户名: "
    echo $LOGNAME
}
# . ~/.bash_profile                                  //重新执行~/.bash_profile 文件
# showwho                                           //执行函数 showwho
四  8月 21 06:19:22 CST 2008
当前登录的用户名:
root
```

函数通常是被保存到一个使用该函数的可执行文件中。例如，下面程序中定义了函数 `showmessage`，该函数在主程序中被执行了两次，代码如下所示（代码 `callfunc`）。

```
1 #!/bin/bash          //程序第1行的“#!/bin/bash”表示调用bash来执行脚本程序
2 #filename:callfunc    //程序第2行的“#filename: callfunc”表示该程序的名称为callfunc
3 function showmessage  //第3~7行定义了函数showmessage
4 {                    //左大括号表示函数showmessage的函数体开始
5   echo "Hello,this is a test prog." //显示输出“Hello,this is a test prog.”
6   echo "Function is a important part of program."//显示输出“Function is a important
part of program.”
7 }                    //右大括号表示函数showmessage的函数体结束
```



```

8 echo "-----第一次调用函数 showmessage -----" //显示输出 "-----第一次调用函数 showmessage
-----"
9 showmessage //调用函数 showmessage
10 echo "-----第二次调用函数 showmessage -----" //显示输出 "-----第二次调用函数 showmessage
-----"
11 showmessage //调用函数 showmessage

```

该程序执行过程如下所示。

```

#chmod u+x callfunc //赋予文件 callfunc 执行权限
#./callfunc //执行程序 callfunc
-----第一次调用函数 showmessage -----
Hello,this is a test prog.
Function is a important part of program.
-----第二次调用函数 showmessage -----
Hello,this is a test prog.
Function is a important part of program.

```

19.2.2 函数参数传递

函数可以通过位置变量传递参数。位置变量是依据出现在函数名之后的参数的位置来确定的变量。在调用函数的命令行中，参数的位置定义如下所示。

\$函数名 参数1 参数2 参数3 ...

当函数被执行时，会将位置变量与调用该函数的命令行中的每一参数关联，依次如下。

\$1: 对应第1个位置参数。

\$2: 对应第2个位置参数。

\$3: 对应第3个位置参数。

... ..

\$9: 对应第9个位置参数。

(1) 例如，下面程序中定义了函数 show。show 函数中显示输出了与主脚本共享的变量 a、b、c、d 的值，以及通过位置变量传递到函数中的参数的值，具体代码如下所示（代码 func）。

```

1 #!/bin/bash //程序第1行的“#!/bin/bash”表示调用 bash 来执行脚本程序
2 #filename:func //程序第2行的“#filename: func”表示该程序的名称为 func
3 show( ) //第3~5行定义函数 show
4 { echo $a $b $c $d //使用 echo 命令，显示输出变量 a、b、c 和 d
5 echo $1 $2 $3 $4; } //使用 echo 命令，显示输出位置变量$1、$2、$3、$4
6 a=111 //变量 a 赋值 111
7 b=222 //变量 b 赋值 222
8 c=333 //变量 c 赋值 333
9 d=444 //变量 d 赋值 444
10 echo "Function Begin" //显示输出“Function Begin”
11 show a b c d //执行函数 show，且带位置参数 a、b、c、d。调用函数 show 时，参数 a 将
赋值给 show //函数的位置变量$1，参数 b 将赋值给 show 函数的位置变量$2，参数 c 将
赋值给 show //函数的位置变量$3，参数 d 将赋值给 show 函数的位置变量$4
12 echo "Function End" //显示输出“Function End”

```



该程序的执行过程如下所示。

```
#chmod u+x func //赋予文件 func 可执行权限
#./func //执行程序 func
Function Begin
111 222 333 444
a b c d
Function End
```

(2) 例如，以下程序中定义了函数 `verify`，用于验证用户名、密码是否是“Zhangsan”和“654321”，代码如下所示（代码 `verifyuser`）。

```
1 #!/bin/bash //程序第 1 行的“#!/bin/bash”表示调用 bash 来执行脚本程序
2 #filename:verifyuser //程序第 2 行的“#filename: verifyuser”表示该程序的名称为
verifyuser
3 function verify //第 3~11 行是 verify 函数。用于验证位置参数 1、参数 2 的值是否是
“Zhangsan” //和“654321”。如果是，则显示“pass!”，否则显示“Rejected, wrong
account!”
4 { //表示函数体开始
5 if [ $1 = "Zhangsan" ] && [ $2 = "654321" ] //第 5~10 行为 if...then...else...fi 结构。用
于判断变 //量$1、$2 的值是否是“Zhangsan”和“654321”。如果
//是，则执行第 7 行，否则执行第 9 行
6 then // if 结构中的 then 语句
7 echo "pass!" //显示输出“pass!”
8 else // if 结构中的 else 语句
9 echo "Rejected, wrong account!" //显示输出“Rejected, wrong account!”
10 fi // if 结构中的 fi 语句，表示 if 结构结束
11 } //表示 verify 函数体结束
12 verify $1 $2 //执行函数 verify，且将当前脚本的位置参数 1、参数 2，作为 verify 的位置参数传
递给函数 verify
```

该代码的执行结果如下所示。

```
# chmod u+x verifyuser //赋予文件 verifyuser 执行权限
# ./verifyuser Lisi 12345 //执行程序 verifyuser
Rejected, wrong account!
# ./verifyuser Zhangsan 654321 //执行程序 verifyuser
pass!
#
```

(3) 例如，编写一个 Shell 程序，其中定义函数 `stringcat`，用于将输入的两个字符串合并输出。具体代码如下所示（代码 `strcat`）。

```
1 #!/bin/bash //程序第 1 行的“#!/bin/bash”表示调用 bash 来执行脚本程序
2 #filename:stringcat //程序第 2 行的“#filename: strcat”表示该程序的名称为 strcat
3 stringcat ( ) //第 3~6 行为 stringcat 函数
4 { //stringcat 函数的左括号，表示函数体开始
5 echo $1$2; //使用 echo 命令，在一行显示输出位置变量$1、$2
6 } //stringcat 函数的右括号，表示函数体结束
7 STR= //定义变量 STR
8 echo "请输入第一个字符串: " //显示输出“请输入第一个字符串: ”
9 read STR1 //读入变量 STR1
10 echo "请输入第二个字符串: " //显示输出“请输入第二个字符串: ”
11 read STR2 //读入变量 STR2
12 echo "连接后的字符串为: " //显示输出“连接后的字符串为: ”
13 stringcat $STR1 $STR2 //调用函数 stringcat，且将变量 STR1 作为函数 stringcat 的位置
参数 1， //变量 STR2 作为函数 stringcat 的位置参数 2
```


该程序的执行过程如下所示。

```
#chmod u+x strcat          //赋予文件 strcat 执行权限
#./strcat                  //执行程序 strcat
请输入第一个字符串:
欢迎您!
请输入第二个字符串:
Sullivan 先生, 欢迎到北京。
连接后的字符串为:
欢迎您! Sullivan 先生, 欢迎到北京。
```

(4) 例如, 编写程序计算 $1 \dots N$ 的立方和, 其中定义函数 `cube`, 用于计算一个数的立方值。具体代码如下所示 (代码 `cubetotal`)。

```
1 #!/bin/bash              //程序第 1 行的“#!/bin/sh”表示调用 bash 来执行脚本程序
2 #filename:cubetotal      //程序第 2 行的“#filename: cubetotal”表示该程序的名称为
cubetotal
3 cube ( )                //第 3~6 行是定义的 cube 函数
4 {                        //cube 函数的左大括号, 表示函数体开始
5   ITEM='expr $1 \* $1 \* $1' //表示计算 cube 函数的位置变量 $1 的立方, 并赋值给变量 ITEM。其中
// “'expr $1 \* $1 \* $1'” 采用了反引号, 表示取得 “expr $1
\* $1 \* $1”
//的计算结果
6 }                        //cube 函数的右大括号, 表示函数体结束
7 echo "请输入一个整数: " //显示输出 “请输入一个整数: ”
8 read N                  //读入变量 N
9 i=1                      //变量 i 赋值为 1
10 RESULT=0                //变量 RESULT 赋值为 0
11 ITEM=                    //定义变量 ITEM
12 while [ $i -le $N ]     //第 12~17 行是 while...do...done 循环结构。其中循环控制条件为
// “$i -le $N”, 即变量 i 小于等于变量 N 时, 执行 while 循环体
第 14~16 行
13 do                      //否则结束循环
14   cube $i                //while 循环结构中的 do 语句
15   RESULT='expr $RESULT + $ITEM ' // “RESULT='expr $RESULT + $ITEM '” 表示计算变量
//RESULT 与变量 ITEM 的和, 且赋值给变量 RESULT。其中
// “'expr $RESULT + $ITEM'” 采用了反引号, 表示取得
// “expr $RESULT + $ITEM” 的计算结果
16   i=$((i+1))             //变量 i 自加 1
17 done                    //while 循环中的 done 语句, 表示 while 结构结束
18 echo "计算 1...$N 的立方和为: $RESULT" //显示输出 “计算 1...$N 的立方和为: $RESULT”
```

该程序的执行过程如下所示。

```
#chmod u+x cubetotal      //赋予文件 cubetotal 可执行权限
#./cubetotal              //执行程序 cubetotal
请输入一个整数:
4
计算 1...4 的立方和为: 100
#./cubetotal              //执行程序 cubetotal
请输入一个整数:
2
计算 1...2 的立方和为: 9
#./cubetotal              //执行程序 cubetotal
请输入一个整数:
6
计算 1...6 的立方和为: 441
```




19.2.3 函数的返回值

函数中的关键字“return”可以放在函数体的任意位置，通常用于返回某些值。程序在执行到 return 之后，函数就停止往下执行，返回到主程序的调用行。return 的返回值只能是 0~256 之间的一个整数，返回值将保存到变量\$?中。可以给 return 指定返回值，如果没有指定 return 的返回值，则返回的函数值就是该函数中最后一个被执行的命令的退出状态值。

例如，以下程序定义了函数 verify，验证脚本位置参数输入的是否是“Zhangsan”和“123456”。在函数中，根据不同情况返回不同的值，然后通过 case 结构根据函数的返回值显示不同的提示信息，程序的具体代码如下所示（代码 funcVerify）。

```

1  #!/bin/bash           //程序第 1 行的“#!/bin/bash”表示调用 bash 来执行脚本程序
2  #filename: funcVerify //程序第 2 行的“#filename: funcVerify”表示该程序的名称为 funcVerify
3  function verify       //第 3~25 行为定义的函数，函数名为 verify
4  {                    //左大括号表示函数体开始
5      if [ $# -ne 2 ]   //第 5~8 行为 if...then...fi 结构。用于判断位置参数的数目是否为 2。如果
$#不等                //于 2，则执行第 7 行，否则跳过该 if 结构，继续执行第 9 行
6      then              //if 结构中的 then 语句，匹配第 5 行的 if
7          return 1      //返回主调用脚本，返回值为 1
8      fi                //if 结构中的 fi 语句，匹配第 5 行的 if
9      if [ $1 = "Zhangsan" -a $2 = "123456" ] //第 9~12 行为 if...then...fi 结构，用于判断位置
参数 1 的              //值是否是“Zhangsan”，位置参数 2 的值是否是“123456”
                      //判断条件为“$1 = "Zhangsan" -a $2 = "123456"”，其中“-a”
                      //表示逻辑与操作。如果变量$1 的值为“Zhangsan”，变量$2
                      //的值为“123456”，则执行第 11 行，否则跳过该 if 结构，继
                      //续执行第 13 行
10     then              //if 结构中的 then 语句，匹配第 9 行的 if
11         return 2      //返回主调用脚本，返回值为 2
12     fi                //if 结构中的 fi 语句，匹配第 9 行的 if
13     if [ $1 != "Zhangsan" -a $2 = "123456" ] //第 13~16 行为 if...then...fi 结构，用于判断位
置参数 1                //的值是否是“Zhangsan”，位置参数 2 的值是否是“123456”
                      //判断条件为“$1 != "Zhangsan" -a $2 = "123456"”，其中“-a”
                      //表示逻辑与操作。如果变量$1 的值不是“Zhangsan”，变量$2
                      //的值为“123456”，则执行第 15 行，否则跳过该 if 结构，继
                      //续执行第 17 行
14     then              //if 结构中的 then 语句，匹配第 13 行的 if
15         return 3      //返回主调用脚本，返回值为 3
16     fi                //if 结构中的 fi 语句，匹配第 13 行的 if
17     if [ $1 = "Zhangsan" -a $2 != "123456" ] //第 17~20 行为 if...then...fi 结构，用于判断位
置参数 1                //的值是否是“Zhangsan”，位置参数 2 的值是否是“123456”
                      //判断条件为“$1 = "Zhangsan" -a $2 != "123456"”，其中“-a”
                      //的值不为“123456”，则执行第 19 行，否则跳过该 if 结构，
                      //继续执行第 21 行
18     then              //if 结构中的 then 语句，匹配第 17 行的 if
19         return 4      //返回主调用脚本，返回值为 4
20     fi                // if 结构中的 fi 语句，匹配第 17 行的 if
21     if [ $1 != "Zhangsan" -a $2 != "123456" ] //第 21~24 行为 if...then...fi 结构，用于判断位
置参数 1                //的值是否是“Zhangsan”，位置参数 2 的值是否是“123456”

```



```

22  then
23      return 5
24  fi
25 }
26 verify $1 $2
27 case $? in
28  1) echo "参数数目错误! ";;
29  2) echo "通过验证! ";;
30  3) echo "用户名错误! ";;
31  4) echo "密码错误! ";;
32  5) echo "用户名、密码错误! ";;
33 esac

```

//判断条件为“\$1 != "Zhangsan" -a \$2 != "123456"”，其中“-a”
 //表示逻辑并操作。如果变量\$1的值不为“Zhangsan”，变量\$2
 //的值不为“123456”，则执行第23行，否则跳过该if结构，
 //继续执行第25行
 //if结构中的then语句，匹配第21行的if
 //返回主调用脚本，返回值为5
 //if结构中的fi语句，匹配第21行的if
 //verify函数定义结束
 //调用verify函数，且将当前主脚本的位置参数1，位置参数2作
 //为verify函数的位置参数1、位置参数2使用
 //第27~33行为case...in...esac结构，用于根据变量\$?的值来
 //执行不同的分支。如果变量\$?的值为1，则执行第28行；如果
 //变量\$?的值为2，则执行第29行；如果变量\$?的值为3，则执行
 //第30行；如果变量\$?的值为4，则执行第31行；如果变量\$?的
 //值为5，则执行第32行。其中“\$?”表示上次命令的返回值，即
 //调用函数“verify \$1 \$2”的返回值
 //如果变量\$?的值为1，则执行该分支，显示输出“参数数目错误！”
 //如果变量\$?的值为2，则执行该分支，显示输出“通过验证！”
 //如果变量\$?的值为3，则执行该分支，显示输出“用户名错误！”
 //如果变量\$?的值为4，则执行该分支，显示输出“密码错误！”
 //如果变量\$?的值为5，则执行该分支，显示输出“用户名、密码错误！”
 // case结构中的esac语句，表示case结构结束

该程序的执行过程如下所示。

```

# chmod u+x funcVerify //赋予文件funcVerify执行权限
# ./funcVerify Zhangsan 123456 //执行程序funcVerify
通过验证!
# ./funcVerify Zhangsan 654321 //执行程序funcVerify
用户名错误!
# ./funcVerify Zhangsan 654321 //执行程序funcVerify
密码错误!
# ./funcVerify Zhangsan 123456 //执行程序funcVerify
通过验证!
# ./funcVerify Zhangsan //执行程序funcVerify
参数数目错误!
# ./funcVerify Zhangming 1234 //执行程序funcVerify
用户名、密码错误!

```

19.2.4 载入函数

函数的定义可以放在“~/.bash_profile”文件中，可以直接放在命令行中，也可以放在脚本文件中。如果需要使用的函数保存在其他脚本文件中，可以通过“source”命令或“.”命令把它们装入到内存中，以供当前脚本使用。

例如，以下程序squareandcube中定义了函数square和cube。其中，square是用于计算平方的函数，cube是用于计算立方的函数，具体代码如下所示（代码squareandcube）。

```

1 #!/bin/bash //程序第1行的“#!/bin/bash”表示调用sh来执行脚本程序
2 #filename:squareandcube //程序第2行的“#filename: squareandcube”表示该程序的名称为squareandcube
3 function square //第3~9行为定义的square函数，函数名为square
4 { //左大括号表示square函数体开始
5     local temp //定义本地变量temp
6     let temp=$1*$1 //使用let命令计算位置变量$1的平方，且将计算的结果值赋予变量temp
7     echo "$1 平方和为: $temp " //显示输出“$1 平方和为: $temp”

```



```

8 } //右大括号表示 square 函数体结束
9 function cube //第 9~14 行为定义的 cube 函数，函数名为 cube
10 { //左大括号表示 cube 函数体开始
11     local temp //定义本地变量 temp
12     let temp=$1*$1*$1 //使用 let 命令计算位置变量 $1 的立方，且将计算的结果值赋予变量
temp
13     echo "$1 立方和为: $temp" //显示输出 "$1 立方和为: $temp"
14 } //右大括号表示 cube 函数体结束

```

下面程序 sourcefunc 用于计算 1^2 、 2^2 、 $3^2 \dots N^2$ ，以及 1^3 、 2^3 、 $3^3 \dots N^3$ ，其中 N 由用户输入。程序中使用 source 命令将上例中的 squareandcube 文件载入，并调用 squareandcube 文件中定义的 square 和 cube 函数完成计算。程序的具体代码如下所示（代码 sourcefunc）。

```

1 #!/bin/bash //程序第 1 行的“#!/bin/bash”表示调用 sh 来执行脚本程序
2 #filename: sourcefunc //程序第 2 行的“#filename: sourcefunc”表示该程序的名称为
sourcefunc
3 source squareandcube //使用 source 命令载入文件 squareandcube
4 echo "请输入一个整数: " //显示输出“请输入一个整数: ”
5 read N //读入变量 N 的值
6 i=1 //变量 i 赋值 1
7 while [ $i -le $N ] //第 7~11 行为 while...do...done 循环结构。循环控制条件为“$i -le $N”，
//即变量 i 的值小于等于变量 N 则继续循环，否则退出循环
8 do //while 循环中的 do 语句，表示循环开始
9     square $i //调用 square 函数，且将变量 i 的值作为参数
10     i=$((i+1)) //变量 i 自加 1
11 done //while 循环中的 done 语句，表示循环体结束
12 echo "-----" //显示分隔符
13 i=1 //变量 i 赋值 1
14 while [ $i -le $N ] //第 14~18 行为 while...do...done 循环结构。循环控制条件为“$i -le $N”，
//即变量 i 的值小于等于变量 N 则继续循环，否则退出循环
15 do // while 循环中的 do 语句，表示循环开始
16     cube $i //调用 cube 函数，且将变量 i 的值作为参数
17     i=$((i+1)) //变量 i 自加 1
18 done // while 循环中的 done 语句，表示循环体结束

```

该程序的执行过程如下所示。

```

#chmod +x sourcefunc //赋予文件 sourcefunc 可执行权限
#./sourcefunc //执行程序 sourcefunc
请输入一个整数:
4 //输入“4”
1 平方和为: 1
2 平方和为: 4
3 平方和为: 9
4 平方和为: 16
-----
1 立方和为: 1
2 立方和为: 8
3 立方和为: 27
4 立方和为: 64

```

19.2.5 删除函数

使用命令“unset -f”可以从 Shell 内存中删除函数，具体格式如下所示。

```
unset -f 函数名
```

例如以下示例中，直接在命令行中定义了函数 testvar()，然后通过 unset 命令从内存中删

除了该函数。具体执行过程如下所示。

```
# testvar ()                                //定义函数 testvar( )
> {
>     local a="这是在函数中定义的局部变量";
>     a="这是在函数中定义的全局变量";
>     echo "在函数内输出: $a"
> }
# declare -f                                //显示内存中已定义的函数
testvar ()                                  //存在 testvar 函数
{
    local a="这是在函数中定义的局部变量";
    a="这是在函数中定义的全局变量";
    echo "在函数内输出: $a"
}
# unset -f testvar                          //删除函数 testvar
# declare -f                                //显示内存中已定义的函数
#                                           //testvar 函数已删除
```

19.3 函数的作用域

每一个变量都有一个作用的范围，即作用域。作用域定义了变量的可见性和生命周期。在函数的使用过程中，通常会涉及到以下两类作用域。

- 全局作用域
- 局部作用域

19.3.1 全局作用域

函数在当前环境下运行，跟调用它的脚本分享变量，即无论在函数内外，没有使用关键字 `local` 进行特别声明的变量都具有全局作用域。全局变量可以在程序的任何地方被访问。

(1) 例如，下面程序中在主程序中定义了全局变量 `a`，函数 `testvar` 中显示输出主程序中定义的全局变量 `a`，具体代码如下所示（代码 `funcvar2`）。

```
1 #!/bin/bash                                //程序第1行的“#!/bin/bash”表示调用 bash 来执行脚本程序
2 #filename:funcvar2                          //程序第2行的“#filename: funcvar2”表示该程序的名称
为 funcvar2
3 function testvar                            //第3~6行为定义的 testvar 函数
4 {                                           //表示函数体开始
5     echo "在函数内输出: $a"                //显示输出“在函数内输出: $a”
6 }                                           //表示函数体结束
7 a="这是在主程序中定义的变量"              //定义变量 a，且赋值为“这是在主程序中定义的变量”
8 testvar                                    //执行函数 testvar
9 echo "在主程序中输出: $a"                  //显示输出“在主程序中输出: $a”
```

该程序的执行结果如下所示。

```
#chmod u+x funcvar2                          //赋予文件 funcvar2 执行权限
#./funcvar2                                  //执行程序 funcvar2
在函数内输出: 这是在主程序中定义的变量
在主程序中输出: 这是在主程序中定义的变量
```

可以看到，在函数 `testvar` 中可以直接访问全局变量 `a`。

(2) 例如下面程序中，在函数 `testvar` 中定义了全局变量 `a`，并分别在函数 `testvar` 内及主程序中访问该变量，具体代码如下所示（代码 `funcvar`）。



```
1 #!/bin/bash //程序第1行的“#!/bin/bash”表示调用 sh 来执行脚本程序
2 #filename:funcvar //程序第2行的“#filename: funcvar”表示该程序的名称为
funcvar
3 function testvar //第3~7行为定义的 testvar 函数
4 { //表示函数体开始
5     a="这是在函数中定义的变量" //定义变量 a, 且赋值“这是在函数中定义的变量”
6     echo "在函数内输出: $a" //显示输出“在函数内输出: $a”
7 } //表示函数体结束
8 testvar //调用函数 testvar
9 echo "在主程序中输出: $a" //显示输出“在主程序中输出: $a”
```

该程序的执行结果如下所示。

```
#chmod u+x funcvar //赋予文件 funcvar 执行权限
#./funcvar //执行程序 funcvar
在函数内输出: 这是在函数中定义的变量
在主程序中输出: 这是在函数中定义的变量
```

可见, 在函数内外都可以正常访问在函数内定义的全局变量。

19.3.2 局部作用域

在函数中使用关键字“local”声明的变量称为该函数的局部变量。局部变量的作用域局限在函数内, 也就是说, 局部变量只在其被声明的函数内可见(包括嵌套函数内部), 而在函数外面是不可访问的。要创建只在函数内部使用, 且在函数退出时就自动撤销的局部变量(也称本地变量), 只需在变量前添加关键字“local”。

例如, 以下程序中定义了函数 testvar, 并在该函数内定义了局部变量 a, 通过 echo 命令分别在函数内外输出局部变量 a 的值, 具体代码如下所示(代码 funcvar3)。

```
1 #!/bin/bash //程序第1行的“#!/bin/bash”表示调用 bash 来执行
脚本程序
2 #filename:funcvar3 //程序第2行的“#filename: funcvar3”表示该程序的
名称为 funcvar3
3 function testvar //第3~7行为定义 testvar 函数
4 { //表示函数体开始
5     local a="这是在函数中定义的局部变量" //定义局部变量 a, 且赋值“这是在函数中定义的局部变量”
6     echo "在函数内输出: $a" //显示输出“在函数内输出: $a”
7 } //表示函数体结束
8 testvar //执行函数 testvar
9 echo "在主程序中输出: $a" //显示输出“在主程序中输出: $a”
```

该程序的执行结果如下所示。

```
#chmod u+x funcvar3 //赋予文件 funcvar3 执行权限
#./funcvar3 //执行程序 funcvar3
在函数内输出: 这是在函数中定义的局部变量
在主程序中输出:
```

可以看到, 由于变量 a 是定义在函数 testvar 中的局部变量。当函数 testvar 执行完毕后, 该函数中定义的所有局部变量也自动被销毁。因此, 在主程序中输出变量 a 的值时, 由于 a 已被销毁, 所以输出结果为空。



当在函数内外定义了相同名称的全局变量与局部变量时, 即当全局作用域与局部作用域发生重叠时, 按就近原则处理。

例如，以下程序中在主程序中定义了全局变量 `a`，在函数 `testvar` 中定义了全局变量 `a` 及局部变量 `a`，并在函数内外分别显示输出变量 `a` 的值。具体代码如下所示（代码 `funcvar4`）。

```
1 #!/bin/bash //程序第1行的“#!/bin/bash”表示调用bash来执行脚本程序
2 #filename:funcvar4 //程序第2行的“#filename: funcvar4”表示该程序的名称
为 funcvar4
3 function testvar //第3~8行为定义的 testvar 函数
4 { //表示函数体开始
5     a="这是在函数中定义的全局变量" //定义变量 a，且赋值“这是在函数中定义的全局变量”
6     local a="这是在函数中定义的局部变量" //定义局部变量 a，且赋值“这是在函数中定义的局部变量”
7     echo "在函数内输出: $a" //显示输出“在函数内输出: $a”
8 } //表示函数体结束
9 a="这是在主程序中定义的变量" //定义变量 a，且赋值“这是在主程序中定义的变量”
10 testvar //执行函数 testvar
11 echo "在主程序中输出: $a" //显示输出“在主程序中输出: $a”
```

该程序的执行结果如下所示。

```
#chmod u+x funcvar4 //赋予文件 funcvar4 可执行权限
#./funcvar4 //执行程序 funcvar4
在函数内输出: 这是在函数中定义的局部变量
在主程序中输出: 这是在主程序中定义的变量
```

从输出结果可以看到，虽然在函数 `testvar` 内定义了全局及局部变量 `a`，但在执行第 7 行的输出操作时，匹配了最近的局部变量。

19.3.3 应用示例

使用局部变量，可以使得函数执行完毕后自动释放变量所占用的内存空间，从而减少系统资源的消耗。在运行大型程序时，定义和使用局部变量尤为必要。以下示例中综合使用了全局变量和局部变量。

(1) 例如，以下程序用于计算两个数的平均值。其中所需计算的数值由位置参数获得，为计算平均值专门定义了函数 `increment`。程序代码如下所示（代码 `funcinc`）。

```
1 #!/bin/bash //程序第1行的“#!/bin/bash”表示调用 sh 来执行脚本程序
2 #filename:funcinc //程序第2行的“#filename: funcinc”表示该程序的名称为
funcinc
3 function increment ( ) //第3~9行为定义的函数，函数名为 increment
4 { //左大括号表示函数体开始
5     local avg; //定义本地变量 avg
6     let avg=($1+$2)/2 //使用 let 命令计算函数的位置变量 $1 和 $2 的平均值，且将计
//算结果赋值给变量 avg
7     echo "在函数内输出变量 avg 的值: $avg" //显示输出“在函数内输出变量 avg 的值: $avg”
8     return $avg //函数返回，且将变量 avg 的值作为返回值返回
9 } //表示函数体结束
10 increment $1 $2 //调用函数 increment，且将当前脚本的位置参数 1，
//参数 2 作为参数传递给函数 increment
11 echo "平均值: $?" //显示输出“平均值: $?”，其中变量 $? 表示上次命令的
执行 //结果，即命令“increment $1 $2”的返回值
12 echo "在函数外输出变量 avg 的值: $avg" //显示输出“在函数外输出变量 avg 的值: $avg”
```

该程序的执行结果如下所示。

```
#chmod +x funcinc //赋予文件 funcinc 可执行权限
#./funcinc 45 66 //执行程序 funcinc，且参数为 45，66
在函数内输出变量 avg 的值: 55
```



平均值: 55

在函数外输出变量 avg 的值:

```
#./funcinc 67 123
```

//执行程序 funcinc, 且参数为 67,123

在函数内输出变量 avg 的值: 95

平均值: 95

在函数外输出变量 avg 的值

可以看到, 在“在函数外输出变量 avg 的值:”一行的输出结果始终为空。这是由于变量 avg 是函数 increment 的局部变量, 当该函数退出时, 变量 avg 也被销毁, 因此输出为空。

(2) 例如, 以下程序用于计算一个数的平方值, 其中定义了函数 square。为了不浪费内存资源, 在 square 函数中使用了本地变量用于临时保存计算结果。该程序的具体代码如下所示(代码 dosquare)。

```
1 #!/bin/bash //程序第 1 行的“#!/bin/bash”表示调用 sh 来执行脚本程序
2 #filename: dosquare //程序第 2 行的“#filename: dosquare”表示该程序的名称为 dosquare
3 function square //第 3~10 行为定义的函数, 函数名为 square
4 { //左大括号表示函数体开始
5     local sq //定义本地变量 sq
6     let sq=$1*$1 //使用 let 命令计算位置变量$1 的平方, 且将其保存到变量 sq 中
7     echo "计算的结果为: " //显示输出“计算的结果为: ”
8     echo "$1 * $1 = $sq" //显示输出“$1 * $1 = $sq”
9     return //函数返回
10 } //右大括号表示函数体结束
11 echo "请输入需要计算的数值: " //显示输出“请输入需要计算的数值: ”
12 read N //使用 read 语句, 读取用户的输入且保存到变量 N 中
13 square $N //调用函数 square, 且将变量 N 作为函数的位置参数
```

该程序的执行过程如下所示。

```
#chmod +x dosquare //赋予文件 dosquare 执行权限
#./dosquare //执行程序 dosquare
请输入需要计算的数值:
8 //输入“8”
计算的结果为:
8 * 8 = 64
#./dosquare //执行程序 dosquare
请输入需要计算的数值:
9 //输入“9”
计算的结果为:
9 * 9 = 81
```

19.4 函数的嵌套

Shell 支持函数的嵌套, 即可在函数中定义并调用其他函数。

例如以下程序中, 在主程序中定义了函数 first, 在 first 函数中又定义了 second 函数, 在 second 函数中又定义了 third 函数。程序执行时, 首行主程序开始调用 first 函数, 然后 first 函数调用 second 函数, second 函数又调用 third 函数。在 third 函数调用完毕后, 返回 second 函数; second 函数执行完毕后, 返回 first 函数; first 函数执行完毕后, 返回主程序。该程序的具体代码如下所示(代码 funcfunc)。


```

1 #!/bin/bash //程序第1行的“#!/bin/bash”表示调用 sh 来执行脚本程序
2 #filename:funcfunc //程序第2行的“#filename: funcfunc”表示该程序的名称
为 funcfunc
3 function first //第3~16行是定义的 first 函数
4 { //表示 first 函数的函数体开始
5     function second //第5~13行是定义的 second 函数
6     { //表示 second 函数的函数体开始
7         function third //第7~10行是定义的 third 函数
8         { //表示 third 函数的函数体开始
9             echo "----这是第三层----" //显示输出“----这是第三层----”
10        } //表示 third 函数的函数体结束
11        echo "----这是第二层----" //显示输出“----这是第二层----”
12        third //调用 third 函数
13    } //表示 second 函数的函数体结束
14    echo "----这是第一层----" //显示输出“----这是第一层----”
15    second //调用 second 函数
16 } //表示 first 函数的函数体结束
17 echo "---开始函数调用----" //显示输出“---开始函数调用----”
18 first //调用 first 函数

```

该程序的执行过程如下所示。

```

#chmod u+x funcfunc //赋予文件 funcfunc 执行权限
#./funcfunc //执行程序 funcfunc
---开始函数调用----
----这是第一层----
----这是第二层----
----这是第三层----

```

19.5 函数的递归

递归是指用自己的定义来说明自己。递归是程序设计中常用的一种程序设计方法。递归函数就是具有调用自身功能的函数。通常，使用递归函数可以将复杂的问题分解为许多小的简单任务，然后反复执行，直到问题解决。递归函数特别适用于浏览动态数据结构，如列表和树。但在 Shell 程序中，极少要求使用如此复杂的数据结构，所以递归函数在实际应用中很少被使用。

为了避免死循环，在设计递归函数的时候，必须有一个明确且必定会发生的条件来终止函数的递归调用。

例如，下面的程序中采用了递归调用，反序输出 1~10 共 10 个数字，代码如下（代码 reverse）。

```

1 #!/bin/bash //程序第1行的“#!/bin/bash”表示调用 bash 来执行脚本程序
2 #filename:reverse //程序第2行的“#filename: reverse”表示该程序的名称为 reverse
3 function reverse //第3~11行为定义的 reverse 函数，函数名为 reverse
4 { //表示函数体开始
5     local t //定义本地局部变量 t
6     echo $1 //显示输出位置变量 $1
7     if [ $1 -gt 0 ];then //第7~10行为 if...then...fi 结构，判断条件为“$1 -gt 0”，即当位置变量
// $1 的值大于 0 时执行第 8、9 行，否则跳过
8         t=$(( $1 - 1 )) //将变量 $1 的值减 1，再赋值给变量 t
9         reverse $t //以变量 t 为参数，调用函数 reverse
10    fi // if 结构中的 fi 语句，表示 if 结构结束

```



```
11 } //表示函数 reverse 的函数体结束
12 reverse 10 //以“10”为参数，调用函数 reverse
```

该程序的执行过程如下所示。

```
#chmod u+x reverse //赋予文件 reverse 可执行权限
#./reverse //执行程序 reverse
10
9
8
7
6
5
4
3
2
1
0
```

当主程序开始调用“reverse 10”时，程序的执行过程如下所示。

(1) 函数 reverse() 被第 1 次调用。

位置变量\$1 的值为 10，显示输出“10”。判断\$1 的值大于 0，则变量 t 赋值为 9，执行“reverse 9”。

(2) 函数 reverse() 被第 2 次调用。

位置变量\$1 的值为 9，显示输出“9”。判断\$1 的值大于 0，则变量 t 赋值为 8，执行“reverse 8”。

(3) 函数 reverse() 被第 3 次调用。

位置变量\$1 的值为 8，显示输出“8”。判断\$1 的值大于 0，则变量 t 赋值为 7，执行“reverse 7”。

(4) 函数 reverse() 被第 4 次调用。

位置变量\$1 的值为 7，显示输出“7”。判断\$1 的值大于 0，则变量 t 赋值为 6，执行“reverse 6”。

(5) 函数 reverse() 被第 5 次调用。

位置变量\$1 的值为 6，显示输出“6”。判断\$1 的值大于 0，则变量 t 赋值为 5，执行“reverse 5”。

(6) 函数 reverse() 被第 6 次调用。

位置变量\$1 的值为 5，显示输出“5”。判断\$1 的值大于 0，则变量 t 赋值为 4，执行“reverse 4”。

(7) 函数 reverse() 被第 7 次调用。

位置变量\$1 的值为 4，显示输出“4”。判断\$1 的值大于 0，则变量 t 赋值为 3，执行“reverse 3”。

(8) 函数 reverse() 被第 8 次调用。

位置变量\$1 的值为 3，显示输出“3”。判断\$1 的值大于 0，则变量 t 赋值为 2，执行“reverse 2”。

(9) 函数 reverse() 被第 9 次调用。

位置变量\$1 的值为 2，显示输出“2”。判断\$1 的值大于 0，则变量 t 赋值为 1，执行

“reverse 1”。

(10) 函数 reverse() 被第 10 次调用。

位置变量 \$1 的值为 1，显示输出 “1”。判断 \$1 的值大于 0，则变量 t 赋值为 0，执行 “reverse 0”。

(11) 函数 reverse() 被第 11 次调用。

位置变量 \$1 的值为 0，显示输出 “0”。判断 \$1 的值不大于 0，函数返回第 10 次调用。

(12) 返回函数 reverse() 第 10 次调用。

第 10 次调用执行完毕，返回第 9 次调用。

(13) 返回函数 reverse() 第 9 次调用。

第 9 次调用执行完毕，返回第 8 次调用。

(14) 返回函数 reverse() 第 8 次调用

第 8 次调用执行完毕，返回第 7 次调用。

(15) 返回函数 reverse() 第 7 次调用。

第 7 次调用执行完毕，返回第 6 次调用。

(16) 返回函数 reverse() 第 6 次调用。

第 6 次调用执行完毕，返回第 5 次调用。

(17) 返回函数 reverse() 第 5 次调用。

第 5 次调用执行完毕，返回第 4 次调用。

(18) 返回函数 reverse() 第 4 次调用。

第 4 次调用执行完毕，返回第 3 次调用。

(19) 返回函数 reverse() 第 3 次调用。

第 3 次调用执行完毕，返回第 2 次调用。

(20) 返回函数 reverse() 第 2 次调用。

第 2 次调用执行完毕，返回第 1 次调用。

(21) 返回函数 reverse() 第 1 次调用。

第 1 次调用执行完毕，返回主脚本程序。

(22) 执行程序完毕。

在很多情况下，递归可以用循环来取代，因为两者都是做一些重复的事情。递归函数比循环慢而且要占用更多的内存，所以应该尽可能多地用循环。例如，上例程序可以用循环来完成，代码如下所示（代码 whilereverse）。

```
1 #!/bin/bash
2 #filename:whilereverse
3 i=10
4 while [ $i -gt 0 ]
5 do
6     echo $i;
7     i=$((i-1))
8 done
```

该代码执行结果如下所示。

```
#chmod u+x whilereverse
#./whilereverse
```

```
//赋予文件 whilereverse 可执行权限
//执行程序 whilereverse
```



10
9
8
7
6
5
4
3
2
1



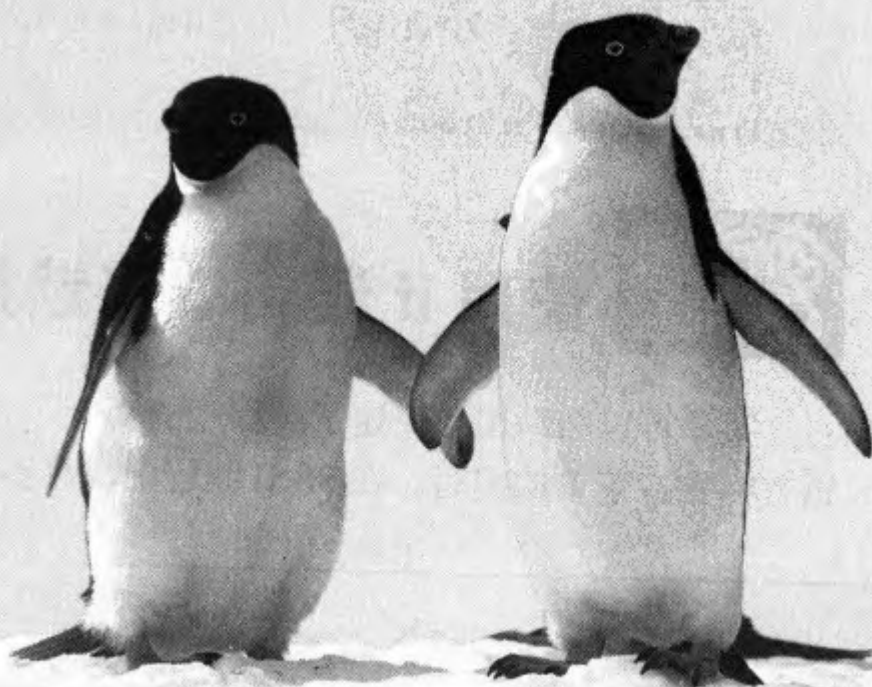
递归方法通常比循环方法的代码更简短、可读性更强。但在使用过程中，如果忘记给出递归的终止条件，函数会一直重复调用下去，直到内存耗尽为止。

UNIX

第20章

Shell 编程实例

每一个 Shell 脚本都是解决某个问题的实例，而其本身也是对一个问题的求解过程的良好描述。本章综合应用 Shell 编程中的各种控制结构、判断表达式，通过 Shell 脚本程序完成了一系列复杂的系统管理与维护任务。





20.1 判断文件类型

Linux 中的文件类型包括普通文件、目录文件、套接字文件、设备文件、链接文件等。通过 Shell 编程，可以对指定文件的文件类型进行判断，并根据判断结果进行相应的处理。

20.1.1 系统设计

例如，判断一个文件是否是符号链接文件。如果是符号链接文件，则将其移动到/etc 目录下；如果不是符号链接文件，则不进行处理。该程序需要使用 if 语句，并配合符号链接文件测试表达式“-L”对文件类型进行判断。移动文件使用 mv 命令。

20.1.2 代码实现

该程序代码主要由一个 if 语句构成。具体如下所示（代码 20.1）。

```
1 #!/bin/bash           //程序第1行的“#!/bin/bash”表示调用bash来执行脚本程序
2 #filename:20.1        //程序第2行的“#filename:20.1”表示该程序的名称为20.1
3 FILENAME=             //程序第3行的“FILENAME=”定义了变量FILENAME，且值为空
4 echo "Input file name:" //通过echo命令显示输出“Input file name”，提示用户输入文件名
5 read FILENAME         //用于从标准输入设备读入用户的输入值，并存入变量FILENAME中
6 if [ -L $FILENAME ]   //第6~9行是if...then...fi结构，用于判断变量FILENAME中指定的
                        //文件是否是链接文件，如果是链接文件则进行移动操作。本行中的“-L”用
                        //于测试文件类型是否是链接文件
7 then                 //if...then...fi结构中的语句then
8   mv $FILENAME /etc  //该行在判断条件“-L $FILENAME”为真时被执行，即将指定的链接文件
                        //移动到/etc目录中
9 fi                  //if...then...fi结构中的语句fi，表示该结构结束
```

该代码执行过程如下所示。

```
# ll lf                //查看链接文件lf的信息
lrwxrwxrwx 1 root    root      4 8月 15 04:43 lf -> file
#chmod u+x 20.1        //赋予文件20.1可执行权限
#./20.1                //执行程序20.1
Input file name:       //输入文件名称
lf
# ll lf                //再次查看链接文件lf的信息
ls: lf: 没有那个文件或目录 //已移到/etc目录下
# ll /etc/lf
lrwxrwxrwx 1 root    root      4 8月 15 04:43 /etc/lf -> file
```

20.2 使用 if 语句编写选择菜单

菜单可以帮助用户快速了解程序的功能，并协助用户方便地进行选择，因此大多数编程语言都支持菜单的编写。在 Shell 编程中，有多种方法可以实现菜单功能，例如，通过 while

循环或 until 循环。本节以最基本的 if 结构为例，讲述如何通过 if 结构的多层嵌套来实现菜单。值得注意的是，多层 if 嵌套降低了代码的可读性，并不是良好的编程习惯。

20.2.1 系统设计

例如，设计一个 Shell 程序，给用户一个提示菜单，根据用户的选择进行相应的操作。菜单的选项有以下 4 个。

- (1) 启动 vi 编辑器。
- (2) 启动 X-windows，进入图形界面。
- (3) 启动后台打印进程，根据输入的文件名打印指定的文件。
- (4) 提示按组合键 Ctrl+C 退出程序。

当用户的选择不符合以上菜单项时，重新显示菜单。

20.2.2 代码实现

该程序代码联合使用多个 if 语句，语句之间彼此嵌套，从而使用了针对菜单选择进行相应处理的功能。代码如下所示（代码 20.2）。

```
1 #!/bin/bash           //程序第 1 行的“#!/bin/bash”表示调用 bash 来执行脚本程序
2 #filename:20.2        //程序第 2 行的“#filename:20.2”表示该程序的名称为 20.2
3 echo "Please choose:(1-3,or ^c)" //第 3~7 行是显示菜单。本行“echo "Please
choose:(1-3,or ^c)"”
//使用 echo 命令显示输出“Please choose:(1-3,or ^c)”
4 echo "1 vi"           //用于显示输出“1 vi”
5 echo "2 X-window"     //用于显示输出“2 X-window”
6 echo "3 print-file"   //用于显示输出“3 print-file”
7 echo "Ctrl+c quit"    //用于显示输出“^c quit”
8 read xyz              //用于将用户的输入存入变量 xyz 中
9 if [ $xyz = 1 ]        //从第 9~26 行是 if...then...elif...then...elfi...then...else...fi 结构，
//用于对用户的输入进行判断，并根据输入值执行相应的命令。本行判
//断变量 xyz 的值是否为 1。如果值为 1，则执行第 11 行、12 行，否则
//执行第 13 行
10 then                 //if 结构中的语句 then
11 vi                   //启动 vi
12 exit 1               //退出程序
13 elif [ $xyz = 2 ]    //if 结构中的语句 elif，用于判断变量 xyz 的值是否为 2。如果值为 2，则
执行第 15 行
//16 行，否则执行第 17 行
14 then                 //if 结构中的语句 then
15 startx               //启动 X-windows
16 exit 1               //退出程序
17 elif [ $xyz = 3 ]    //if 结构中的语句 elif，用于判断变量 xyz 的值是否为 3。如果值为 3，则执行
//第 19~22 行，否则执行第 24、25 行
18 then                 //if 结构中的语句 then
19 echo "               //Please enter filename:" //用于显示输出“Please enter
filename:”
20 read filename        //用于读入用户输入的文件名，且保存到变量 filename 中
21 lp $filename &        //表示在后台打印输出指定的文件
22 exit 1               //退出程序
23 else                 //if 结构中的语句 else
24 echo "-----" //用于输出分隔线“-----”
```



```
25 /bin/sh 20.2 //调用执行程序 20.2
26 fi //if 结构中的语句 fi, 表示该结构结束
```

上述代码执行过程如下所示。

```
#chmod u+x 20.2 //设置可执行权限
#./20.2 //执行程序 20.2
Please choose:(1-3,or ^c) //显示菜单
1 vi
2 X-window
3 print-file
Ctrl+c quit
5 //输入 5 并按回车键
-----
Please choose:(1-3,or ^c) //显示菜单
1 vi
2 X-window
3 print-file
Ctrl+c quit
8 //输入 8 并按回车键
-----
Please choose:(1-3,or ^c) //显示菜单
1 vi
2 X-window
3 print-file
Ctrl+c quit //按 Ctrl+c 组合键退出
```

20.3 使用 while 循环编写选择菜单

在 Shell 编程中,有多种可以实现菜单功能的编程方法,20.2 节介绍了采用最基本的 if 结构实现菜单,本节将介绍采用 while 循环完成菜单的编程方法。与 if 结构相比,while 循环代码更清晰简洁、可读性更强。

20.3.1 系统设计

例如,设计一个 Shell 程序,实现如下所示菜单功能,即根据用户输入不同的选择执行相应的操作。

- (1) 当用户输入 start 时,在后台启动应用程序 myfiles;
- (2) 当用户输入 stop 时,关闭在后台运行的 myfiles 程序;
- (3) 当用户输入 status 时,查看 myfiles 进程的相关信息;
- (4) 当用户输入 quit 时,显示“Quit,Thanks”并退出程序。

当用户输入非 start、stop、status 及 quit 时,循环显示菜单。本例中,假定 myfiles 是保存在当前目录下的可执行程序。

20.3.2 代码实现

使用 while 循环循环显示菜单,并根据用户对菜单的选择进行相应处理,代码如下所示(代码 20.3)。


```

1 #!/bin/bash           //程序第 1 行的“#!/bin/bash”表示调用 bash 来执行脚本程序
2 #filename:20.3        //程序第 2 行的“#filename:20.3”表示该程序的名称为 20.3
3 i=1                   //定义变量 i，赋初值为 1
4 while [ "$i" -eq 1 ]   //第 4~23 行是 while...do...done 结构。“while [ "$i" -eq 1 ]”
用于判断
5 do                     //变量 i 的值是否为 1。如果值为 1 则继续循环，否则退出循环
6 RESPONSE=              // while 结构中的 do 语句
7 echo "-----"        //“RESPONSE=”定义变量 RESPONSE，且初值为空
出分隔线
8 echo "Please choice:"   //显示输出“Please choice:”
9 echo "Input -- start -- start program myfiles." //显示输出“Input -- start --
start program myfiles.”
10 echo "Input -- stop -- kill myfiles."           //显示输出“Input -- stop --
kill myfiles.”
11 echo "Input -- status-- view status of myfiles process." //显示输出“Input --
status-- view status of
// myfiles process.”
12 echo "Input -- quit -- exit."                   //显示输出“Input -- quit -- exit.”
13 echo "-----"                                   //显示输出分隔线
14 read RESPONSE                                     //读入用户的输入，并保存到变量
RESPONSE 中
15 case $RESPONSE in                                //第 15~22 行是 case...in...esac 结构。
“case $RESPONSE in”
16     start) ./myfiles&;;                          //表示对变量 RESPONSE 进行判断
//如果变量 RESPONSE 的值为“start”，则执行命
令“./myfiles&”
17     stop) killall myfiles ;;                     //如果变量 RESPONSE 的值为“stop”，则执行命令
//“killall -9 myfiles”，杀死 myfiles 进程
18     status) ps -aux | grep myfiles;;             //如果变量 RESPONSE 的值为“status”，则执行命
令
//“ps -aux | grep myfiles”，显示“myfiles”
相关进程信息
19     quit) i=0                                     //如果变量 RESPONSE 的值为“quit”，则执行第
19、20、21 行。
//本行在变量 RESPONSE 的值为“quit”时，将变量
i 的值赋值 0
20     echo "Quit,Thanks"                           //在变量 RESPONSE 的值为“quit”时，显示输出
“Quit,Thanks”
21     ;;                                           //“;;”表示 quit 分支结束
22 esac                                             //“esac”表示 case 结构结束
23 done                                             //“done”表示 while 循环结束

```

该代码执行过程如下所示。

```

#chmod u+x 20.3           //赋予文件 20.3 可执行权限
#./20.3                   //执行程序 20.3
-----                  //显示菜单
Please choice:
Input -- start -- start program myfiles.
Input -- stop -- kill myfiles.
Input -- status-- view status of myfiles process.
Input -- quit -- exit.
-----
start                     //输入 start，在后台运行 myfiles
-----                  //显示菜单

```




```
Please choice:
Input -- start -- start program myfiles.
Input -- stop -- kill myfiles.
Input -- status-- view status of myfiles process.
Input -- quit -- exit.

-----
status //输入 status, 显示进程相关信息
root 16850 0.0 0.5 5796 1508 pts/1 S 18:43 0:00 -bash
root 16859 0.0 0.2 4816 640 pts/1 S 18:43 0:00 grep 16850
----- //显示菜单

Please choice:
Input -- start -- start program myfiles.
Input -- stop -- kill myfiles.
Input -- status-- view status of myfiles process.
Input -- quit -- exit.

-----
stop //输入 stop, 杀死相关进程
----- //显示菜单

Please choice:
Input -- start -- start program myfiles.
Input -- stop -- kill myfiles.
Input -- status-- view status of myfiles process.
Input -- quit -- exit.

-----
status //输入 status, 显示进程相关信息
root 16861 0.0 0.2 4820 640 pts/1 S 18:43 0:00 grep 16850
[1]+ □ ./myfiles
----- //显示菜单

Please choice:
Input -- start -- start program myfiles.
Input -- stop -- kill myfiles.
Input -- status-- view status of myfiles process.
Input -- quit -- exit.

-----
quit //输入 quit, 退出程序
Quit,Thanks
#
```

20.4 使用 until 循环编写选择菜单

在 Shell 编程中, 由于 until 循环所具有的特殊结构非常适合菜单的编写, 因此, 大多数程序员在实现菜单功能时最常用的方法就是利用 until 循环。

20.4.1 系统设计

例如, 设计一个 Shell 程序, 给用户提供一个选择菜单, 根据用户的选择进行相应的操作。菜单的选项如下。

- (1) 按数字键“1”, 列出当前目录的内容。
- (2) 按数字键“2”, 切换到指定的目录。

- (3) 按数字键“3”，根据指定的文件名创建文件。
- (4) 按数字键“4”，根据指定的文件名编辑文件。
- (5) 按数字键“5”，根据指定的文件名删除文件。
- (6) 按数字键“6”，退出程序。

当用户的输入非 1、2、3、4、5、6 时，重新显示菜单。

20.4.2 代码实现

使用 `until` 循环，循环显示菜单，并根据用户对菜单的选择进行相应处理。除了 `until` 语句外，还需要配合使用 `case`、`read`、`echo` 等语句。代码如下所示（代码 20.4）。

```

1 #!/bin/bash           //程序第 1 行的“#!/bin/bash”表示调用 bash 来执行脚本程序
2 #filename:20.4        //程序第 2 行的“#filename:20.4”表示该程序的名称为 20.4
3 input=                //定义变量 input，且赋初值为空
4 until                 //第 4~36 行是 until...do...done 结构。本行为 until 语句
5   echo "-----"      //第 5~13 行用于显示菜单。本行显示分隔线
6   echo "Please enter your choice: (1-6)" //显示输出“Please enter your choice: (1-6)”
7   echo "(1)List you selected directory" //显示输出“(1)List you selected directory”
8   echo "(2)Change to you selected directory" //显示输出“(2)Change to you selected directory”
9   echo "(3)Creat a new file" //显示输出“(3)Creat a new file”
10  echo "(4)Edit you selected file" //显示输出“(4)Edit you selected file”
11  echo "(5)Remove you selected file" //显示输出“(5)Remove you selected file”
12  echo "(6)Exit Menu" //显示输出“(6)Exit Menu”
13  echo "-----" //显示输出分隔行
14  read input //读入用户输入，并保存到变量 input 中
15  test $input = 6 //测试变量 input 的值是否为 6。如果值为 6，则退出 until 循环，否则继续执行
16 do // until 结构中的 do 语句
17   case $input in //第 17~35 行是 case...in...esac 结构。本行“case $input in”表示对
//变量 input 的值进行判断
18     1) ls;; //如果变量 input 的值为 1，则执行命令 ls，列出当前目录的内容
19     2) echo "Enter target directory:" //如果变量 input 的值为 2，则执行第 19~22 行。
本行“echo "Enter target
//directory:"”表示显示输出“Enter target directory:”
20         read dir //当变量 input 的值为 2 时，执行命令“read dir”，即读入用户的输入，且保存到 dir 变量中
21         cd $dir //当变量 input 的值为 2 时，执行命令“cd $dir”，即改变当前目录
//为变量 dir 中指定的目录
22         ;; //“;;”表示值为 2 的分支结束
23     3) echo "Enter a file name:" //如果变量 input 的值为 3，则执行第 23~26 行。本行“echo
" Enter a file
// name:”表示显示输出“Enter a file name:”
24         read file //当变量 input 的值为 3 时，执行命令“read file”，即读入用户的输入，且保存到 file 变量中
25         touch $file //当变量 input 的值为 3 时，执行命令“touch $file”，即创建一个新
//文件，文件名由变量 file 指定
26         ;; //“;;”表示值为 3 的分支结束
27     4) echo "Enter a file name:" //如果变量 input 的值为 4，则执行第 27~30 行。本行
//“echo "Enter a file name:”表示显示输出“Enter a file name:”

```




```
28     read file           //当变量 input 的值为 4 时，执行命令“read file”，即读入用户的输
                             //入，且保存到 file 变量中
29     vi $file            //当变量 input 的值为 4 时，执行命令“vi $file”，即用 vi 打开变量
                             //file 指定的文件
30     ;;                  //“;;”表示值为 4 的分支结束
31 5) echo "Enter a file na //如果变量 input 的值为 5，则执行第 31~34 行
                             //本行“echo "Enter a filename:"”表示显示输出“Enter a filename:”
32     read file           //当变量 input 的值为 5 时，执行命令“read file”，即读入用户的输
                             //入，且保存到 file 变量中
33     rm $file            //当变量 input 的值为 5 时，执行命令“rm $file”，即删除由变量 file
                             //指定的文件
34     ;;                  //“;;”表示值为 5 的分支结束
35 esac                   //“esac”表示 case 结构结束
36 done                   //“done”表示 while 循环体结束
```

该代码的执行过程如下所示。

```
#chmod u+x 20.4           //给文件 20.4 赋予执行权限
#./20.4                   //执行程序 20.4
----- //显示菜单
Please enter your choice: (1-6)
(1)List you selected directory
(2)Change to you selected directory
(3)Creat a new file
(4)Edit you selected file
(5)Remove you selected file
(6)Exit Menu
-----
1                          //输入“1”，列出当前目录
0808.tar.gz  20.9          man1_gz_list
20.1         cpumemmesg    man1_list
20.10        cpumemmesg_noheader myfiles
20.11        cpumesg       pa
20.12        df_field      pas
----- //显示菜单
Please enter your choice: (1-6)
(1)List you selected directory
(2)Change to you selected directory
(3)Creat a new file
(4)Edit you selected file
(5)Remove you selected file
(6)Exit Menu
-----
2                          //输入“2”
Enter target directory:
/root                  //输入“/root”，将当前目录切换到/root 目录
----- //显示菜单
Please enter your choice: (1-6)
(1)List you selected directory
(2)Change to you selected directory
(3)Creat a new file
(4)Edit you selected file
(5)Remove you selected file
(6)Exit Menu
-----
```



```

3                                     //输入“3”
Enter a file name:
menutest.doc                         //输入创建文件的名字“menutest.doc”
-----                             //显示菜单
Please enter your choice: (1-6)
(1)List you selected directory
(2)Change to you selected directory
(3)Creat a new file
(4)Edit you selected file
(5)Remove you selected file
(6)Exit Menu
-----
1                                     //输入“1”，显示当前目录，文件menutest.doc已存在
*          etcbak      index.html.1      ls.doc mbox      mypaper.doc
anaconda-ks.cfg hello      install.log      mail      menutest.doc pa
dump.tmp      index.html install.log.syslog Mail      minicom.log temp
-----                             //显示菜单
Please enter your choice: (1-6)
(1)List you selected directory
(2)Change to you selected directory
(3)Creat a new file
(4)Edit you selected file
(5)Remove you selected file
(6)Exit Menu
-----
5                                     //输入“5”
Enter a file name:
menutest.doc                         //输入要删除文件的名字“menutest.doc”
rm: 是否删除一般空文件‘menutest.doc’? y
-----
Please enter your choice: (1-6)
(1)List you selected directory
(2)Change to you selected directory
(3)Creat a new file
(4)Edit you selected file
(5)Remove you selected file
(6)Exit Menu
-----
1                                     //输入“1”，显示当前目录，文件menutest.doc已被删除
*          etcbak      index.html.1      ls.doc mbox      pa
anaconda-ks.cfg hello      install.log      mail      minicom.log temp
dump.tmp      index.html install.log.syslog Mail      mypaper.doc
-----                             //显示菜单
Please enter your choice: (1-6)
(1)List you selected directory
(2)Change to you selected directory
(3)Creat a new file
(4)Edit you selected file
(5)Remove you selected file
(6)Exit Menu
-----
6                                     //输入“6”，退出程序
#

```



20.5 对指定目录定时备份

对一些特定的目录或文件进行备份是系统管理员的经常性工作。通过 Shell 编程，可以按指定的格式进行备份与压缩，配合使用 `cron` 等定时任务工具，可以进一步简化繁琐的备份工作。

20.5.1 系统设计

例如，设计一个 Shell 程序，在每月 1 日 0:00 备份并压缩 `/etc` 目录的所有文件及子目录。备份文件存放在 `/root/etcbak` 目录里，且备份文件的名称格式为 `YYMMDD_etc.tar.gz`，其中 `YY` 表示年份，`MM` 表示月份，`DD` 为日。

20.5.2 代码实现

该程序中使用 `date` 命令获得当前的日期，具体代码如下所示（代码 20.5）。

```
1 #!/bin/bash //程序第 1 行的“#!/bin/bash”表示调用 bash
来执行脚本程序
2 #filename:20.5 //程序第 2 行的“#filename:20.5”表示该程序的名称为 20.5
3 DIRNAME='ls /root | grep etcbak' //定义了变量 DIRNAME，变量的值来自命令“ls /root
| grep
etcbak'”采用 // etcbak'”的执行结果。“ls /root | grep
etcbak'”采用 //了反引号，表示命令“ls /root | grep etcbak'”的
执行结果， //其中“ls /root”用于列/root 目录下的内容，然后通过管道“|”
道“|” //将目录内容输出给 grep 命令，由 grep 命令在其中查找与
//“etcbak”匹配的记录，最终将查找结果返回给变量 DIRNAME
//该行代码实质是查找根目录下是否有与字符串“etcbak”相似的
//目录名称，如果有，则将相应目录名称返回到变量 DIRNAME，
//如果没有，变量 DIRNAME 的值为空
4 echo $DIRNAME //用于显示变量 DIRNAME 的值
5 if [ -z $DIRNAME ]; then //第 5~7 行是 if...then...fi 结构。本行“-z $DIRNAME”
//用于判断变量 DIRNAME 的值是否为空，其中“-z”用于对字符
//串长度进行判断，如果字符串为空，则返回值为真。当判断变量
//DIRNAME 为空时，表示根目录下没有“etcbak”相关目录，需
//要进行创建，继续执行第 6 行。如果变量 DIRNAME 不为空，表
//示根目录下已经存在“etcbak”相关目录，继续执行第 8 行
6 mkdir /root/etcbak //表示在根目录下创建 etcbak 子目录
7 fi // if 结构中的 fi 语句，表示 if 结构结束
8 cd /root/etcbak //表示将当前目录切换到/root/etcbak
9 YY='date +%y' //定义变量 YY，并将命令“date +%y”的执行结果返回给变量 YY。这里采用了反引号，
//取得命令“date +%y”的执行结果。其中命令“date +%y”取得当前日期的年份
10 MM='date +%m' //定义变量 MM，并将命令“date +%m”的执行结果返回给变量 MM。这里采用了反引号，
//取得命令“date +%m”的执行结果。其中命令“date +%m”取得当前日期的月份
11 DD='date +%d' //定义变量 DD，并将命令“date +%d”的执行结果返回给变量 DD。这里采用了反引号，
```



```

//取得命令“date +%d”的执行结果。其中命令“date +%d”取得当前日期的天数
12 BACKETC=$YY$MM$DD_etc.tar.gz //定义变量BACKETC,该变量将作为压缩后文件的文件名。BACKETC
//取值“$YY$MM$DD_etc.tar.gz”,即文件名的格式为:
//年月日_etc.tar.gz
13 tar zcvf $BACKETC /etc //采用tar命令对/etc目录进行压缩
14 echo "File backup finished!" //显示输出“File backup finished!”

```

该程序代码执行过程如下所示。

```

#chmod u+x 20.5 //赋予文件20.5可执行权限
# date
五 8月1 22:21:47 CST 2008
#./20.5 //执行程序20.5
tar: Removing leading '/' from member names
etc/
etc/sysconfig/
etc/sysconfig/network-scripts/
etc/sysconfig/network-scripts/ifdown-aliases
etc/sysconfig/network-scripts/ifcfg-lo
...
etc/hosts.bak
etc/.aumixrc
etc/ioctl.save
File backup finished!
# ls /root/etcbak //显示目录/root/etcbak
080701_etc.tar.gz 080801_etc.tar.gz

```

为了实现每月1日定时自动备份,需要进一步编写任务定时器。首先将程序20.5保存到/usr/bin目录下,然后使用“crontab -e”命令添加定时任务,如下所示。

```
0 0 1 * * /usr/bin/20.5
```

20.6

连续对多个目录逐个备份

在对系统备份时,通常需要根据类别对文件集中进行备份。通过 Shell 编程,可以一次对多个具有相似特征的文件或目录进行统一备份。

20.6.1 系统设计

例如编写 Shell 程序,对系统中的配置文件进行备份,并设定每 50 分钟执行一次。配置文件通常保存在 conf 目录中,在备份之前需要先建立备份目录列表,然后通过 while 循环完成备份。定时备份需要编写任务定时器。此例中将用到 while、tar、crontab 等语句。

20.6.2 代码实现

通过使用 find 命令建立备份目录列表,然后通过 while 循环对列表中记录的每一目录进行备份,代码如下所示(代码 20.6)。

```

1 #!/bin/bash //程序第1行的“#!/bin/bash”表示调用bash来执行脚本程序
2 #filename :20.6 //程序第2行的“#filename:20.6”表示该程序的名称为20.6
3 i=0 //定义变量i,赋初值为0
4 find / -name conf > file //表示从根目录开始查找名为conf的目录,且将查找结果保存到文件file中
//该行代码的意义在于确定需要压缩的目录,并将目录列表保存到文件file中
//每一目录占用一行

```



```

5 if [ -d /tmp/conf ]; then //第 5~10 行是 if...then...else...fi 结构。本行“-d /tmp/conf”
表示判
//断是否存在目录/tmp/conf。如果存在则执行第 6 行，显示
/tmp/conf 已经存
//在；如果不存在，则转到执行第 8、9 行，创建该目录
6 echo "/tmp/conf already exists." //用于显示输出“/tmp/conf already exists.”
7 else //if 结构中的 else 语句
8 mkdir /tmp/conf //创建目录/tmp/conf
9 echo "/tmp/conf is created." //用于显示输出“/tmp/conf is created.”
10 fi //if 结构中的 fi 语句，表示 if 结构结束
11 while read LINE //第 11~17 行是 while...do...done 循环结构。其中“read LINE”表示每
//次循环时，将读入的内容赋值给变量 LINE。从第 17 行可以看到，本例从文
//件 file 读入数据，即每次循环从文件 file 读入一行，赋值给变量 LINE
12 do // while 结构中的 do 语句
13 DIRS=$LINE; //将变量 LINE 的值赋值给变量 DIRS
14 BACKUP="/tmp/conf/backup${i}.tar.gz" //定义变量 BACKUP，赋值“/tmp/conf/backup${i}.
tar.gz”，
//该行代码用于指定压缩文件的名称。
15 tar -zcvf $BACKUP $DIRS //利用 tar 命令完成压缩。其中变量 DIRS 指定了被压缩目录，
//变量 BACKUP 指定了压缩文件的名称及存放路径
16 i=$((i+1)) //循环变量 i 的值增加 1。每次循环该值加 1，用于保证压缩文
//件有不同的名称，不会出重名覆盖
17 done<file //表示该 while 结构从文件 file 读入数据，同时表示 while 循
环体结束

```

该代码的执行过程如下所示。

```

#chmod u+x 20.6 //赋予文件 20.6 可执行权限
#./20.6 //执行程序 20.6
/tmp/conf is created.
tar: Removing leading '/' from member names
proc/sys/net/ipv4/conf/
proc/sys/net/ipv4/conf/eth0/
proc/sys/net/ipv4/conf/eth0/arp_filter
proc/sys/net/ipv4/conf/eth0/tag
...
usr/share/pvm3/conf/LINUXSPARC.m4
usr/share/pvm3/conf/LINUXX86_64.def
usr/share/pvm3/conf/LINUXX86_64.m4
# cat file //显示备份目录列表
/proc/sys/net/ipv4/conf
/etc/log.d/conf
/etc/httpd/conf
/usr/share/doc/mrtg-2.9.17/contrib/mrtg_php_portal/conf
/usr/share/pvm3/conf
# ls /tmp/conf //显示备份完的文件
backup0.tar.gz backup1.tar.gz backup2.tar.gz backup3.tar.gz backup4.tar.gz

```

为了实现每隔 50 分钟定时自动备份，需要进一步编写任务定时器。首先将程序 20.6 保存到/usr/bin 目录下，然后使用“crontab -e”命令添加定时任务，如下所示。

```
0 /50 * * * * /usr/bin/20.6 #设定每隔 50 分钟执行程序 20.5
```

20.7 自动下载指定文档

在使用 ftp 建立连接过程中，通常需要输入一系列连接参数，如远端主机地址、用户名、

密码等。即使登录到远端主机之后，仍需要使用若干内部命令对 ftp 连接进行配置，如设置传输模式、是否进行哈希等。如果需要多次对同一远端主机使用相同的参数进行连接，重复输入连接命令及参数将使工作变得非常繁琐。

为了简化操作，可以通过编写 Shell 程序实现连接的自动建立，甚至是文件的自动上传、下载。

20.7.1 系统设计

例如，编写一个 Shell 程序，自动登录到远端 ftp 服务器 192.168.20.122，自动下载/etc/rc.d 目录下的 rc.local 文件。登录服务器时使用的用户名为“teacher”，密码为“654321”，采用二进制传输模式，且在下载时利用哈希显示下载进度。此例中需要用到 ftp 的一些常用内部命令，如 bin、hash、get、bye 等。

20.7.2 代码实现

该程序通过 echo 命令将建立 ftp 连接所需参数借助管道“|”传递给 ftp 命令，代码如下所示（代码 20.7）。

```
1 #!/bin/bash //程序第 1 行的“#!/bin/bash”表示调用 bash 来执行脚本程序
2 #filename :20.7 //程序第 2 行的“#filename:20.7”表示该程序的名称为 20.7
3 echo "open 192.168.20.122 //第 3~10 行通过 echo 命令将下载所需的参数传递给 ftp 命令。本行
//“open 192.168.20.122”表示建立到 ftp 服务器 192.168.20.122
的连接
4 user teacher 6543 //表示建立连接时，使用的用户名为“teacher”，密码为“654321”
5 bin //表示采用二进制传输模式
6 hash //表示进行哈希，即以符号“#”表示传输的进度
7 cd /etc/rc.d //表示切换远端服务器 192.168.20.122 上的当前目录为/etc/rc.d
8 get rc.local //表示下载远端服务器 192.168.20.122 上的文件 rc.local
9 bye //“bye”表示结束本次 ftp 连接
10 "|ftp -n //其中管道“|”表示将 echo 命令的输出，输入到 ftp 命令。选项“-n”表
示不允许自动登录
```

该代码的执行过程如下所示。

```
#chmod u+x 20.7 //赋予文件 20.7 可执行权限
# ./20.7 //执行程序 20.7
Please login with USER and PASS.
Please login with USER and PASS.
KERBEROS_V4 rejected as an authentication type
Hash mark printing on (1024 bytes/hash mark).
## //显示下载进度
# ls //显示当前目录
rc.local //已将文件 rc.local 下载到本地
```

20.8 自动上传指定文档

对系统进行备份比较安全可靠的方式是双机热备份，即对比较重要的文件备份并通过网络及时传输到其他主机保存。这样，在系统崩溃完全无法恢复时，也可以从其他主机恢复重



要数据。

通过 Shell 编程实现对系统重要资源的自动备份并上传到备份服务器，可以极大地减轻系统管理员日常维护的工作量。

20.8.1 系统设计

例如，编写一个 Shell 程序对系统的重要资源进行备份，备份内容包括：

- (1) 系统配置文件，即/etc/rc.d 目录中的所有文件；
- (2) /etc 目录下的所有配置文件，即.conf 文件；
- (3) 系统日志文件，即/var/log 目录中的所有文件。

备份后，将备份文件通过 ftp 传输到远端主机 192.168.20.122 上，且登录 ftp 的用户名为“teacher”、密码为“654321”。ftp 连接采用二进制传输模式，显示上传进度及提示信息。

通过编写任务定时器，设定每周一的 0:00 自动执行编写的程序实现备份及上传操作。在此例中将用到 read、echo、tar、crontab 等语句。

20.8.2 代码实现

该程序使用 tar 命令对指定的目录进行备份，并通过 echo 命令将参数传递给 ftp 命令，实现上传操作。该代码的具体说明如下所示（代码 20.8）。

```
1 #!/bin/bash           //程序第 1 行的“#!/bin/bash”表示调用 bash 来执行脚本程序
2 #filename :20.8       //程序第 2 行的“#filename:20.8”表示该程序的名称为 20.8
3 mkdir -p /backup-sys //表示创建目录/backup-sys，其中选项“-p”表示在目录已存在时不提示
                        //错误信息
4 if [ $? -eq 0 ] ;then //第 4~22 行是 if...then...else...fi 结构。本行“$? -eq 0”表示判断上次
                        //命令
                        //（本例即指命令“mkdir -p /backup-sys”）是否正确执行，其中$?表
                        //示上次命令
                        //的返回值。该行代码即表示如果目录/backup-sys 存在，则继续执行第 5~
                        //19 行，
                        //否则执行第 21 行，显示输出无法创建目录
5 tar zcvf /backup-sys/rc.tar.gz /etc/rc. //利用 tar 命令对/etc/rc.d 目录进行压缩，
      压缩文件名
                                          //为 rc.tar.gz 且保存到/backup-sys 目录
      中
6 tar zcvf /backup-sys/conf.tar.gz /etc/*.conf //利用 tar 命令对/etc 目录下的所
      有.conf 文件进行压
                                          //缩，压缩文件名为 conf.tar.gz 且保
      存到
                                          //目录/backup-sys 中
7 tar zcvf /backup-sys/log.tar.gz /var/lo //利用 tar 命令对/etc/log 目录进行压
      缩，压缩文件名为
                                          //log.tar.gz 且保存到/backup-sys
      目录中
8 cd /backup-sys           //改变当前目录为/backup-sys
9 if [ $? -eq 0 ];then     //第 9~19 行是 if...then...else...fi 结构。本行“$? -eq 0”
      用于判断上次
                        //命令（本例即指命令“cd /backup-sys”）是否正确执行（如
      果上次命令被
                        //正确执行，则$?值为 0，否则为 1）。如果命令“cd
```


/backup-sys” 执行成

将当前目录切

```
10 echo "open 192.168.20.122
命令。本行
```

192.168.20.122 的连接

```
11 user teacher 654321
“654321”
```

```
12 binary
```

```
13 prompt
```

```
14 hash
```

```
15 mput *
```

上传到远端服

```
16 bye"| ftp -n
命令的输出，
```

```
17 else
```

```
18 echo "Could not into backup-sys directory!"
backup-sys directory!"
```

```
19 fi
```

```
20 else
```

```
21 echo "Could not create backup directory! //显示输出“Could not create backup
directory!"
```

```
22 fi
```

相匹配，表示 if 结构结束

该代码执行过程如下所示。

```
#chmod u+x 20.8
```

```
#./20.8
```

```
tar: Removing leading '/' from member names
```

```
etc/rc.d/
```

```
etc/rc.d/init.d/
```

```
etc/rc.d/init.d/microcode_ctl
```

```
etc/rc.d/init.d/kdcrotate
```

```
etc/rc.d/init.d/kudzu
```

```
... ..
```

```
var/log/vsftpd.log
```

```
var/log/vsftpd.log.1
```

```
Please login with USER and PASS.
```

```
Please login with USER and PASS.
```

```
KERBEROS_V4 rejected as an authentication type
```

```
Interactive mode off.
```

```
Hash mark printing on (1024 bytes/hash mark). //显示上传进度
```

```
#####
```

```
#####
```

```
#####
```

```
#####
```

```
#####
```

```
#####
```

```
#####
```

```
#####
```



```
#####  
#####  
#####  
#####
```


为了实现每周一 0:00 的定时自动备份及上传操作，需要进一步编写任务定时器。首先将程序 20.8 保存到 /usr/bin 目录下，然后使用 “crontab -e” 命令添加定时任务，如下所示。

```
0 0 * * mon /usr/bin/20.8 #设定每周一 0:00 执行程序 20.8
```

20.9 通过命令连续创建 50 个用户账号

管理员有时需要一次性创建大量用户账号，例如新学期开学或成立新的部门时。如果使用 useradd 命令一个一个用户地创建将浪费大量时间，而且容易出现录入错误。通常在此情况下，利用 Shell 编程可以便捷地完成批量用户的创建。

20.9.1 系统设计

例如，编写 Shell 程序实现系统自动创建 50 个用户。首先添加一个新组为 users1，然后添加属于这个组的 50 个用户。用户名的格式为 stdXX，其中 XX 从 01 到 30，即用户名为 student01、student02、student03...student10、student11 student12...student50。本例将用到 while 循环，useradd、groupadd、chgrp、chown 等命令。

20.9.2 代码实现

该程序利用 while 循环实现用户的连续创建。具体代码如下所示（代码 20.9）。

```
1 #!/bin/bash //程序第 1 行的“#!/bin/bash”表示调用 bash 来执行脚本程序  
2 #filename :20.9 //程序第 2 行的“#filename:20.9”表示该程序的名称为 20.9  
3 i=1 //定义变量 i，且赋初值为 1  
4 groupadd user1 //创建用户组“user1”  
5 while [ $i -le 50 ] //第 5~17 行是 while...do...done 循环结构。本行“$i -le 50”表示判断变  
//量 i 的当前值是否小于等于 50。如果小于等于 50 则执行 do...done 之间的循环  
//体；如果大于 50，退出 while...do...done 循环结构。其中“-le”表示小  
于等于  
6 do // while 结构中的 do 语句  
7 if [ $i -le 9 ] ;then //第 7~11 行是 if...then...else...fi 结构，用于判断变量 i 的当前值是否小  
//于等于 9，小于等于 9 时执行第 8 行，大于 9 时执行第 10 行。由于本例要  
//求在创建的用户名称中，后两位必须是数字，对于编号是个位数的用户名需  
//要使用 0 来占位，即前 9 个用户名为 student01、student02、  
//student03...student09。而这不能直接利用变量 i 逐次加 1 完成命名，所以  
//需要进行判断。对于小于等于 9 的情况，在数字前添加 0 来占位，大于 9 的  
//情况则无需 0 占位  
8 USERNAME=student0$i //定义变量 USERNAME，且取值为“student0${i}”，即用户名  
//由“student0+变量 i 的值”组成  
9 else //if 结构中的 else 语句  
10 USERNAME=student${i} //定义变量 USERNAME，且取值为“student${i}”，即用户名  
//由“student+变量 i 的值”组成  
11 fi //if 结构中的 fi 语句，表示 if 结构结束
```



```

12  useradd $USERNAME          //使用 useradd 命令创建新用户，用户名为变量 USERNAME 的值
13  mkdir /home/$USERNAME      //使用 mkdir 命令创建新用户的主目录，目录名为
"/home/$USERNAME"
14  chown -R $USERNAME /home/$USERNAM //使用 chown 命令指定主目录
                                     // "/home/$USERNAME" 的所有者为
                                     //$USERNAME。其中选项 "-R" 表示递归操作文
                                     //件及子目录
15  chgrp -R user1 /home/$USERN    //使用 chgrp 命令指定主目录 "/home/$USERNAME"
                                     //的用户组为 user1。其中选项 "-R" 表示递归操作文
                                     //件及子目录
16  i=$((i+1))                 //变量 i 自加 1
17 done                         // "done" 表示 while 循环体结束

```

代码执行过程如下所示。

```

#chmod u+x 20.9                //赋予文件 20.9 执行权限
#./20.9                         //执行程序 20.9
#ls /home                       //查看创建用户的主目录
student01 student07 student13 student19 student25 student31 student37 student43
student49
student02 student08 student14 student20 student26 student32 student38 student44
student50
student03 student09 student15 student21 student27 student33 student39 student45
sullivan
student04 student10 student16 student22 student28 student34 student40 student46
teacher
student05 student11 student17 student23 student29 student35 student41 student47
student06 student12 student18 student24 student30 student36 student42 student48

```

20.10 通过操作文件创建多个用户

在 20.9 节中，通过 while 循环连续执行 useradd 命令实现了批量用户的创建。对于系统来说，创建一个新用户实质对应以下三项操作。

- (1) 创建新用户的主目录，通常是/home 下以用户名为名称的子目录；
- (2) 在/etc/passwd 文件中添加新用户的记录；
- (3) 在/etc/shadow 文件中添加新用户的记录；

useradd 命令实质就是完成上述三项操作的命令。系统管理员完全可以通过手工操作完成以上三项工作，从而完成新用户的添加。而通过编写 Shell 编程，不仅可以自动完成以上三项操作实现新用户添加，也可以实现批量用户的自动添加。

20.10.1 系统设计

创建新用户时，系统会在/home 目录下创建以用户名命名的目录，并且在/etc/passwd 和/etc/shadow 文件中按相应文件格式写入用户信息。本例编写一个 Shell 程序，按以上步骤实现 50 个用户的自动创建。

首先使用 cat 命令查看/etc/passwd 文件的格式，如下所示。

```

# cat /etc/passwd
root:x:0:0:root:/root:/bin/bash

```



```
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
news:x:9:13:news:/etc/news:
uucp:x:10:14:uucp:/var/spool/uucp:/sbin/nologin
operator:x:11:0:operator:/root:/sbin/nologin
...
desktop:x:80:80:desktop:/var/lib/menu/kde:/sbin/nologin
radvd:x:75:75:radvd user:/:/sbin/nologin
teacher:x:500:500:./home/teacher:/bin/bash
```

使用 `cat` 命令查看 `/etc/shadow` 文件的格式，如下所示。

```
# cat /etc/shadow
root:$1$xn timers $LO46xrcRYUyXrpSZpyL./0:14098:0:99999:7:::
bin:!:14098:0:99999:7:::
daemon:!:14098:0:99999:7:::
adm:!:14098:0:99999:7:::
lp:!:14098:0:99999:7:::
sync:!:14098:0:99999:7:::
shutdown:!:14098:0:99999:7:::
halt:!:14098:0:99999:7:::
mail:!:14098:0:99999:7:::
news:!:14098:0:99999:7:::
uucp:!:14098:0:99999:7:::
operator:!:14098:0:99999:7:::
...
desktop:!!:14098:0:99999:7:::
radvd:!!:14098:0:99999:7:::
teacher:$1$6ZebarlR$iCqcdus8qCZfH9fxXUAd4l:14098:0:99999:7:::
```

在确定 `/etc/passwd` 及 `/etc/shadow` 文件的格式之后，就可以参照该格式编写新用户的用户信息。

20.10.2 代码实现

程序中主要用到了 `while` 循环，`chown`、`chgrp` 等命令以及流过滤器 `awk`。该程序的具体代码如下所示（代码 20.10）。

```
1 #!/bin/bash //程序第 1 行的“#!/bin/bash”表示调用 bash 来执行脚本程序
2 #filename:20.10 //程序第 2 行的“#filename:20.10”表示该程序的名称为 20.10
3 i=1 //定义变量 i，赋初值为 1，用于在 while 循环中控制循环的次数，通常称为循环变量
4 awk 'BEGIN { FS=":"; } { print $3 ;}' /etc/passwd >pa //表示将 passwd 文件中以冒号
分隔的第 3 列保存到 //pa 文件中。其中“BEGIN”表示从/etc/passwd 文
//件每行的首字符开始处理，“{ FS=":"; }”表示指定
//分隔符为冒号，“{ print $3 ;}”表示输出/etc/passwd
//文件中第 3 列。该行代码实质是将文件/etc/passwd
//中第 3 列提取出来保存到文件 pa 中
5 x='tail -1 pa' //将文件 pa 倒数第一行内容保存到变量 x 中，即变量 x 保存了当前最大的用户
//ID。其中“'tail -1 pa'”采用了反引号，即将命令“tail -1 pa”的执行
结果返回，
```



```

        //赋值给变量 x
6 x=$((x+1)) //将变量 x 的值加 1, 作为下一新建用户的用户 ID
7 while [ $i -le 50 ] //第 7~16 行为 while...do...done 循环结构。while 循环的控制条件为
    // “$i -le 50”, 即小于等于 50 时, 执行 while 循环体, 大于 50 时结束循环
8 do // while 结构中的 do 语句
9 mkdir -p /home/stud${i} //使用 mkdir 命令创建用户的主目录为 “/home/stud${i}”, 其中
    选项 “-p” 表示
    //当目录已存在时不显示错误信息
10 chown -R $stud${i} /home/$stud${i} //使用 chown 命令指定主目录 “/home/stud${i}”
    的所有者为
    //stud${i}。其中选项 “-R” 表示递归操作文件及子目录
11 chgrp -R user1 /home/$stud${i} //使用 chgrp 命令指定主目录 “/home/stud${i}”
    的用户组为
    //user1。其中选项 “-R” 表示递归操作文件及子目录
12 echo "stud${i}:x:${x}:100::/home/stud${i}:/bin/bash" >> /etc/passwd //将字符串
    // “stud${i}:x:${x}:100::/home/stud${i}:/bin/bash” 附加到文件 /etc/passwd 文件的尾部。
    其中字符串按 /etc/passwd
    //文件的格式编写, 第 1 个字段 “stud${i}” 表示用户名, 第 2 个字段 “x” 表示该用户有密码, 第 3 个
    字段 “${x}”
    //为用户 ID, 第 4 个字段 “100” 为用户组 ID, 第 5 个字段为用户的基本信息, 这里为空, 第 6 个字段
    “/home/stud${i}”
    //为用户主目录, 第 7 个字段 “/bin/bash” 为用户登录时默认使用的 Shell
13 echo "stud${i}::14098:0:99999:7:::" >> /etc/shadow //将字符串 “echo
    “stud${i}::14098:0:99999:7:::”
    //附加到文件 /etc/shadow 文件的末尾。字符串内容按 /etc/shadow 文件的格式编写, 共 9 个字段, 分
    别表示用户名、
    //密码、从 1970 年到上次修改密码的天数、密码必须连续使用的天数, 密码有效期、密码失效前告警的天
    数、从密
    //码过期到彻底停用的天数、账号失效日期, 最后一段作为保留字段。本例中第 1 字段为 “stud${i}”,
    第 2 字段为
    //空, 即无密码, 第 3 字段值为 14098, 第 4 字段值为 0, 第 5 字段为 99999, 第 6 字段值为 7, 其余字
    段为空
14 i=$((i+1)) //将变量 i 的值加 1
15 x=$((x+1)) //将变量 x 的值加 1
16 done // “done” 表示 while 循环体结束

```

该代码的执行过程如下所示。

```

#chmod u+x 20.10 //赋予文件 20.10 执行权限
#./20.10 //执行程序 20.10
# tail -10 /etc/passwd //显示/etc/passwd 文件后 10 行, 可以看到相关用户信息已添加
stud41:x:590:590::/home/stud41:/bin/bash
stud42:x:591:591::/home/stud42:/bin/bash
stud43:x:592:592::/home/stud43:/bin/bash
stud44:x:593:593::/home/stud44:/bin/bash
stud45:x:594:594::/home/stud45:/bin/bash
stud46:x:595:595::/home/stud46:/bin/bash
stud47:x:596:596::/home/stud47:/bin/bash
stud48:x:597:597::/home/stud48:/bin/bash
stud49:x:598:598::/home/stud49:/bin/bash
stud50:x:599:599::/home/stud50:/bin/bash
# tail -10 /etc/shadow //显示/etc/shadow 文件后 10 行, 可以看到相关用户信息已添加
stud41::14098:0:99999:7:::
stud42::14098:0:99999:7:::
stud43::14098:0:99999:7:::
stud44::14098:0:99999:7:::
stud45::14098:0:99999:7:::

```



```
stud46::14098:0:99999:7:::
stud47::14098:0:99999:7:::
stud48::14098:0:99999:7:::
stud49::14098:0:99999:7:::
stud50::14098:0:99999:7:::
# ls /home //查看/home目录下用户的主目录,可以看到相关用户主目录已创建
stud19 stud29 stud39 stud49 stud9 stud15 stud25 stud35 stud45
stud1 stud2 stud3 stud4 stud5 stud16 stud26 stud36 stud46
stud10 stud20 stud30 stud40 stud50 stud17 stud27 stud37 stud47
stud11 stud21 stud31 stud41 stud6 stud18 stud28 stud38 stud48
stud12 stud22 stud32 stud42 stud7 stud14 stud24 stud34 stud44
stud13 stud23 stud33 stud43 stud8 teacher sullivan
```

20.11 连续创建多个目录

Linux 是一个多用户、多任务操作系统,支持多用户的使用。经验表明,在多用户系统上,防止磁盘空间浪费的一种有效方法是为每一用户指定一个专门的私有目录,然后配合使用磁盘配额技术,控制每个用户使用的磁盘空间量。

当需要连续创建的目录数目比较多时,通过 Shell 编程可以简化创建目录的操作,并可以统一对目录的权限进行设定。

20.11.1 系统设计

例如,设计一个 Shell 程序,在/student 目录下建立 50 个目录,即 student1~student50,并设置每个目录的权限如下。

- 文件所有者所在组的权限为:读、执行。
- 文件所有者的权限为:读、写、执行。
- 其他用户的权限为:读。

本例将用到 while 循环,mkdir、chmod、echo 等命令。

20.11.2 代码实现

该程序需要使用 while 循环完成 50 个目录的集中创建。在创建目录之前需要首先通过 if 结构对创建目录的父目录是否存在进行判断。通过 chmod 命令赋予每个目录的权限为 754,即文件所有者的权限为“7”,文件所有者所在组的权限为“5”,其他用户的权限为“4”。该程序的具体代码实现如下所示(代码 20.11)。

```
1 #!/bin/bash //程序第1行的“#!/bin/bash”表示调用bash来执行脚本程序
2 #filename: 20.11 //程序第2行的“#filename:20.11”表示该程序的名称为20.11
3 i=1 //定义变量i,赋初值为1,该变量作为while的循环变量
4 while [ $i -le 50 //第4~18行为while...do...done循环结构。当变量i的值小于等于50时继续
//循环,当变量i的值大于50时,结束while循环
5 do //while结构中的do语句
6 if [ -d /student ];then //第6~17行为if...then...else...fi结构。本行中的“-d
/student ”
//用于判断/student目录是否存在。如果存在则执行第7~10
行,如果不存
```



```

//在则执行第 12~16 行, 即首先创建父目录/student
7   mkdir /student/student$i //使用 mkdir 命令创建目录 "/student/student$i"
8   chmod 754 /student/student$i //使用 chmod 命令为目录 "/student/student$i" 赋予
权限 754
9   echo "student$i" //用于显示输出创建的目录名
10  i=$((i+1)) //变量 i 自加 1
11  else // if 结构中的 else 语句
12  mkdir /student //使用 mkdir 命令创建目录/student
13  mkdir /student/student$i //使用 mkdir 命令创建目录 "/student/student$i"
14  chmod 754 /student/student //使用 chmod 命令为目录 "/student/student$i" 赋予
权限 754
15  echo "student$i" //用于显示输出创建的目录名
16  i=$((i+1)) //变量 i 自加 1
17  fi //if 结构中的 fi 语句, 表示 if 结构结束
18 done // "done" 表示 while 循环体结束

```

该代码的执行过程如下所示。

```

#chmod u+x 20.11 //设置文件 20.11 执行权限
#./20.11 //执行程序 20.11
student1
student2
student3
... ..
student48
student49
student50
#
# ls /student //查看/student 目录下创建的子目录
student1 student15 student20 student26 student31 student37 student42
student48 student8
student10 student16 student21 student27 student32 student38 student43
student49 student9
student11 student17 student22 student28 student33 student39 student44 student5
student12 student18 student23 student29 student34 student4 student45
student50
student13 student19 student24 student3 student35 student40 student46 student6
student14 student2 student25 student30 student36 student41 student47 student7
# ll /student //创建目录的权限为 rwxr-xr--, 即 754
总用量 200
drwxr-xr-- 2 root root 4096 8 月 14 21:01 student1
drwxr-xr-- 2 root root 4096 8 月 14 21:14 student10
drwxr-xr-- 2 root root 4096 8 月 14 21:14 student11
drwxr-xr-- 2 root root 4096 8 月 14 21:14 student12
drwxr-xr-- 2 root root 4096 8 月 14 21:14 student13
drwxr-xr-- 2 root root 4096 8 月 14 21:14 student14
... ..

```

20.12 检查系统资源使用情况

由于系统资源通常是有限的, 大量被某一用户或进程占用将影响其他用户甚至威胁到系统的稳定性。系统管理员在对系统进行维护时, 出于安全及效率等原因的考虑, 应该及时掌握系统资源的使用情况。

常用的检查系统资源使用情况的命令有 df、ps 等。其中, df 命令可以查看文件系统的使



用情况，包括已用的磁盘空间、剩余磁盘空间等。ps 命令可以查看 CPU、内存等的使用情况。通过 Shell 编程，可以方便地实现对系统资源情况的查询。

20.12.1 系统设计

例如，编写一个 Shell 程序，对 df 和 ps 命令的执行结果进行检查，进而对系统资源使用情况进行判断，检查过程如下。

(1) 使用 df 命令检查磁盘空间使用情况，即查看 df 命令执行结果的第 5 例“已用%”是否超过 50%，若超过 50%则将相关信息记录到文件 df_report 中。

(2) 使用 ps 命令检查系统各进程所占用的 CPU 和内存情况，即查看 ps 命令执行结果的第 3 例(“%CUP”)和第 4 列(“%MEM”),若其中有一项不等于 0.0 则将相关信息记录到文件 ps_report 中。

首先查看 df 命令的执行结果，如下所示。

```
# df
```

文件系统	1K-块	已用	可用	已用%	挂载点
/dev/sda2	7637400	5032868	2216564	70%	/
/dev/sda1	101089	9425	86445	10%	/boot
none	127632	0	127632	0%	/dev/shm

可以看到，在 df 命令执行结果中，首行为标题行，且标题行中有字符串“1K-”(通过该字符串可以定位 df 输出结果中的标题行)，在第 5 例显示“已用%”，表示磁盘空间使用的百分率。

查看 ps 命令的执行结果，如下所示。

```
# ps -aux
```

USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND
root	1	0.0	0.0	1372	68	?	S	Aug10	0:05	init
root	2	0.0	0.0	0	0	?	SW	Aug10	0:00	[keventd]
root	3	0.0	0.0	0	0	?	SW	Aug10	0:00	[kapmd]
...	...	...	...	...	...	...	...	...	...	...
root	2048	0.0	0.0	1440	168	?	S	Aug10	0:03	syslogd -m 0
root	2052	0.0	0.0	1368	4	?	S	Aug10	0:00	klogd -x
rpc	2070	0.0	0.0	1540	40	?	S	Aug10	0:00	[portmap]
rpcuser	2089	0.0	0.0	1524	0	?	SW	Aug10	0:00	[rpc.statd]
...	...	...	...	...	...	...	...	...	...	...

可以看到，在 ps 命令执行结果中，首行为标题行，且标题行中有字符串“USER”(通过该字符串可以定位 ps 输出结果中的标题行)，输出结果中的第 3 例(“%CUP”)显示 CUP 的使用率、第 4 列(“%MEM”)显示内存的使用率。

20.12.2 代码实现

该程序由三部分组成：第一部分对命令 df 及 ps 的输出结果进行预处理，提出需要的字段；第二部分用于检查磁盘空间使用情况；第三部分用于检查 CPU 及内存的使用情况。该程序的具体代码如下所示(代码 20.12)。

```
1 #!/bin/bash //程序第 1 行的“#!/bin/sh”表示调用 sh 来执行脚本程序
2 #filename:20.12 //程序第 2 行的“#filename:20.12”表示该程序的名称为 20.12
3 #####Preprocess##### //注释，说明以下部分是预处理
4 df>dfmesg //表示将 df 命令的执行结果保存到文件 dfmesg 中
5 pr -e dfmesg | awk '{stuff}' //表示将文件 dfmesg 中的所有 Tab 转换为空格
```



```

6 sed '/1K-/d' dfmesg > dfmesg_noheader //表示删除 dfmesg 文件中包含“1K-”的行，即将
dfmesg                                     //文件中的标题行删除，且将删除标题后的数据保存到
                                           //文件 dfmesg_noheader 中
7 ps -aux | more > cpumemmesg             //表示将系统所有进程的详细信息保存到 cpumemmesg
                                           //文件中。其中“ps -aux”用于显示系统中所有进程的
                                           //详细信息，通过管道“|”将 ps 的输出结果输出给 more
                                           //命令，并最终保存到 cpumemmesg 文件中
8 pr -e cpumemmesg | awk '{stuff}'        //表示将文件 cpumemmesg 中的所有 Tab 转换为空格
9 sed '/USER/d' cpumemmesg >cpumemmesg_noh //表示删除 cpumemmesg 文件中包含“USER”
                                           //的行，即将 cpumemmesg 文件中的标题行删除，
                                           //且将删除标题后的数据保存到文件
                                           //cpumemmesg_noheader 中
10 #####Check System HD##### //注释，表示以下部分为检测磁盘空间使用情况
11 while read LINE1                     //第 11~28 行为 while...do...done 循环结构。其中控制条件为“read
LINE1”，                                //即表示每循环一次从指定文件读入一行数据，且保存到变量 LINE1 中。
读入的文
                                           //件由第 28 行“done<dfmesg_noheader”指定，该行代码表示将
                                           //dfmesg_noheader 文件的内容输入给 while 循环
12 do                                   // while 结构中的 do 语句
13 DISK=$LINE1                         //定义变量 DISK，且赋值变量 LINE1 的值，即将 dfmesg_noheader
文件中某行的
                                           //内容赋值给变量 DISK
14 echo $DISK>df_item                 //将变量 DISK 的值保存到文件 df_item 中
15 DF='awk 'BEGIN { FS=" "; } { print $5; }' df_item' //表示将 df_item 文件中以空格
为分隔符的第 5 列内容
                                           //保存到变量 DF 中。其中“'awk 'BEGIN { FS=" "; }
                                           //{ print $5; }' df_item'”采用了反引号，表示命令“awk
                                           //'BEGIN { FS=" "; } { print $5; }' df_item”的执行结
                                           //果返回给变量 DF。反引号中的“BEGIN”表示从
                                           //df_item 文件每行的首字符开始处理，“{FS=" "; }”
                                           //表示指定分隔符为空格，“{ print $5 ;}”表示输出
                                           //df_item 文件中第 5 列。该行代码实质是将文件
                                           //df_item 中第 5 列提取出来保存到变量 DF 中
16 echo $DF>df_field                 //用于将变量 DF 的值保存到文件 df_field 中
17 DD='awk 'BEGIN { FS=" "; } {print $1}' df_field' //“DD='awk 'BEGIN { FS=" "; } {print
$1}' df_field'”
                                           //表示将 df_field 文件中以“ ”为分隔符的第 1 列
                                           //内容保存到变量 DD 中。其中“'awk 'BEGIN
                                           //{ FS=" "; } {print $1}' df_field'”采用了反引号，表
                                           //示命令“awk 'BEGIN { FS=" "; } {print $1}' df_field”
                                           //的执行结果返回给变量 DD。反引号中的“BEGIN”
                                           //表示从 df_item 文件每行的首字符开始处理，
                                           //“{FS=" "; }”表示指定列分隔符为“ ”，“{ print
                                           // $1 ;}”表示输出 df_field 文件中第 1 列。该行代码实
                                           //质是将文件 df_field 中第 1 列提取出来保存到变量
                                           //DD 中
18 DINTER=$(( $DD+0 ))               //用于将字符型变量 DD 转换为数字型变量。该行代码的实质是将变量 DD
                                           //与数字 0 相加，从而转换为数字型变量
19 if [ $? -eq 0 ] ; then             //第 19~27 行为 if...then...else...fi 结构。本行代码“$? -eq 0”用于判
断上次命令（即第 18 行代码“DINTER=$(( $DD+0 ))”）是否被正
确执行，
20 if [ $DINTER -gt 50 ] ; then //第 20~24 行为 if...then...else...fi 结构。本行代码“$DINTER

```




```

-gt 50"
//用于判断变量 DINTER 的值是否大于等于 50。当大于等于 50
时,说明磁
//盘空间的占用量超出,应进行记录,继续执行第 21 行,否则执
行 23 行,
//记录磁盘空间仍然充裕
21     echo $DISK>>df_report //用于将变量 DISK 的值附加保存到文件 df_report
22     else // if 结构的 else 语句,匹配第 20 行的 if...then
23     echo "This system have enough HD.">>df_report //用于将字符串 "This system have
enough HD."
//附加写入文件 df_report
24     fi // if 结构的 fi 语句,匹配第 20 行的 if...then,表示该 if 结构结束
25     else // if 结构的 else 语句,匹配第 19 行的 if...then
26     echo "Check system HD Error" //用于显示输出 "Check system HD Error"
27     fi //if 结构的 fi 语句,匹配第 19 行的 if...then,表示该 if 结构结束
28 done<dfmesg_noheader //其中 "done" 匹配第 11 行的 while 结构,表示循环体结束,
// "<dfmesg_noheader" 表示 while 结构从文件
dfmesg_noheader
//读入数据
29 #####Check system CPU&MEM##### //注释,说明以下部分代码用于对 CPU 及内存
//的使用情况进行检测
30 while read LINE2 //第 30~39 行为 while...do...done 循环
结构。其中控制条件为 "read LINE2",
//即表示每循环一次从指定文件读入一行
数据,且保存到变量 LINE2 中。读入的
//文件由第 39 行 "done <
cpumemmesg_noheader" 指定,该代码表示将
//cpumemmesg_noheader 文件的内容
输入给 while 循环
31 do // while 结构中的 do 语句
32 CPUMEM=$LINE2 //表示将变量 LINE2 的值赋给变量 CPUMEM
33 echo $CPUMEM>ps_item //表示将变量 CPUMEM 的值保存到文件 ps_item 中
34 CPUMESG='awk 'BEGIN { FS=" "; } { print $3; } ' ps_item' //表示将 ps_item 文
件中以空格为分隔符
//的第 3 列内容保存到
变量中。其中 "'awk 'BEGIN { FS=" "; } { print $3; } ' ps_item'" 采用了反引号,表示命令 "awk
// 'BEGIN { FS=" "; }
{ print $3; } ' ps_item" 的执行结果返回给变量 CPUMESG。反引号中的 "BEGIN" 表示从 ps_item
//文件每行的首字符开始处理, "{ FS=" "; }" 表示指定列分隔符为空格, "{ print $3; }" 表示
输出 ps_item 文件中
//第 3 列。该行代码实质是将文件 ps_item 中第 3 列提取出来保存到变量 CPUMESG 中
35 MEMMESG='awk 'BEGIN { FS=" "; } { print $4; } ' ps_item' //表示将 ps_item 文
件中以空格为分隔符
//的第 4 列内容保存到变量中。其中 "'awk 'BEGIN { FS=" "; } { print $4; } ' ps_item'"
采用了反引号,表示命令 "awk
// 'BEGIN { FS=" "; } { print $4; } ' ps_item" 的执行结果返回给变量 MEMMESG。反引号
中的 "BEGIN" 表示从 ps_item
//文件每行的首字符开始处理, "{ FS=" "; }" 表示指定列分隔符为空格, "{ print $4; }" 表示
输出 ps_item 文件中
//第 4 列。该行代码实质是将文件 ps_item 中第 4 列提取出来保存到变量 MEMMESG 中
36 if [ $CPUMESG != "0.0" -o $MEMMESG != "0.0" ] ;then //第 36~38 行为
if...then...fi 结构
//其中"$CPUMESG != "0.0" -o $MEMMESG != "0.0" 表示,或者变量 CPUMESG 的值不等于"0.0",
或者变
//量 MEMMESG 的值不等于"0.0"。该 if 结构用于判断只要变量 CPUMESG 或 MEMMESG 的值有一个不等

```


于“0.0”，

```

    //则执行第 37 行，即记录不为零的项，否则不进行记录
37     echo $CPUMEM>>ps_report           //表示将变量 CPUMEM 的值附加保存到文件 ps_report 中
38     fi                               //if 结构中的 fi 语句，匹配第 36 行的 if...then，表示
if 结构结束
39 done < cpumemmesg_noheader           //其中“done”匹配第 30 行的 while 结构，表示循环体
结束，
    //“< cpumemmesg_noheader”表示 while 结构从文件
    //cpumemmesg_noheader 读入数据

```

该代码的执行结果如下所示。

```

#chmod u+x 20.12           //赋予文件 20.12 执行权限
# ./20.12                  //运行程序 20.12
# cat dfmesg_noheader      //显示文件 dfmesg_noheader 的内容
/dev/sda2      * 7637400 5120644 2128788 71% /
/dev/sda1      101089 9425 86445 10% /boot
none          127632 0 127632 0% /dev/shm
# cat df_report           //显示文件 df_report 的内容
/dev/sda2 7637400 5120644 2128788 71% /
This system have enough HD.
This system have enough HD.
# cat cpumemmesg_noheader //显示文件 cpumemmesg_noheader 的内容
root      1 0.0 0.0 1372 84 ?      S   Aug10 0:05 init
root      2 0.0 0.0 0 0 ?        SW  Aug10 0:00 [keventd]
root      3 0.0 0.0 0 0 ?        SW  Aug10 0:00 [kapmd]
root      4 0.0 0.0 0 0 ?        SWN Aug10 0:00 [ksoftirqd_CPU0]
root      9 0.0 0.0 0 0 ?        SW  Aug10 0:30 [bdflood]
root      5 0.0 0.0 0 0 ?        SW  Aug10 0:11 [kswapd]
... ..
# cat ps_report           //显示文件 ps_report 的内容
root 2194 0.0 0.1 3504 340 ? S Aug10 0:01 /usr/sbin/sshd
root 2234 0.0 0.1 5928 436 ? S Aug10 0:01 [sendmail]
smmsp 2243 0.0 0.1 5712 324 ? S Aug10 0:00 [sendmail]
root 2254 0.0 0.5 19116 1512 ? S Aug10 0:00 /usr/bin/spamd -d -c -a
root 2307 0.0 0.2 8524 544 ? S Aug10 0:07 cupsd
xfs 2432 0.0 0.4 7580 1240 ? S Aug10 0:00 [xfs]
... ..

```

20.13 账户的一致性检查

通常，创建新用户时会在/home 目录下创建以用户名命名的用户主目录，在/etc/passwd 和/etc/shadow 文件中按相应文件格式写入用户信息。但有时系统管理员在删除用户时只删除了三项中的某一项或两项，从而造成账户不一致问题。

为了消除这种不一致的情况，可以编写 Shell 程序自动查看系统中用户主目录与 passwd 和 shadow 文件中的用户信息是否一致。如果出现不一致现象，给出相应的警告。

20.13.1 系统设计

编写一个 Shell 程序，对账户的不一致性进行检查，即检查用户主目录、/etc/passwd 和



/etc/shadow 三者中信息不一致问题。在检查时可以采用两方面策略如下所示。

- (1) 依据/home 目录中的目录名称, 对/etc/passwd 和/etc/shadow 文件进行一致性检查。
 - (2) 依据/etc/passwd 文件中的记录, 对/home 目录及/etc/shadow 文件进行一致性检查。
- 本例将使用 case...in...esac 结构分别处理这两方面的策略。

20.13.2 代码实现

该程序主框架结构为 case 分支, 在每一分支中配合使用了 while 循环, 最终将不一致信息从屏幕打印输出。该程序的具体代码如下所示, 其中代码第 8~52 行为 case...in...esac 结构, 用于判断位置参数 1 的值是“-d”、“-p”, 还是“*”, 从而执行不同的分支。“-d”表示以目录/home 下的目录名称为基准进行判断, “-p”表示以/etc/passwd 文件中的用户信息为基准进行判断, “*”表示当参数既不是“-d”也不是“-p”时, 显示提示信息。具体结构如下所示。

```
case "$1"
-d) ...
;;
-p) ...
;;
*) ...
;;
esac
```

其中“\$1”表示本脚本程序执行时所带的第一个参数。程序 20.13 的具体代码如下(代码 20.13)。

```
1 #!/bin/bash           //程序第 1 行的“#!/bin/bash”表示调用 bash 来执行脚本程序
2 #filename:20.13       //程序第 2 行的“#filename:20.13”表示该程序的名称为 20.13
3 USAGE="usage: 'basename $0' [-d : According to /home directory | -p : According
to /etc/passwd]"
//定义变量 USAGE, 将提示信息“usage: 'basename $0' [-d : According to /home directory
| -p : According
//to /etc/passwd]”赋值给该变量 USAGE。其中“'basename $0'”采用反引号, 表示该命令的执行结果,
该命令
//用于提取正在运行的脚本程序的名称, 且去除路径。“[-d : According to /home directory | -p :
According
//to /etc/passwd]”为提示信息, 提示用户在执行该脚本时使用参数“-d”或“-p”。其中“-d”表示依据
/home
目录中的目录名称, 对/etc/passwd 和/etc/shadow 文件进行一致性检查, “-p”表示依据/etc/passwd 文
件中的记录, 对/home 目录及/etc/shadow 文件进行一致性检查
4 if [ $# -ne 1 ] ; then //第 4~7 行为 if...then...fi 结构。“$# -ne 1”用于判断$#的值是否为 1。
//如果值不为 1, 则执行第 5、6 行, 显示提示信息并退出; 如果值为 1, 则跳过
//第 5、6、7 行。其中“$#”是 Shell 的特殊变量, 表示本脚本程序执行时所带
//的参数个数
5 echo $USAGE           //用于显示输出变量 USAGE 的值
6 exit 1                //退出程序
7 fi                    //if 结构中的 fi 语句, 匹配第 4 行的 if...then
8 case $1 in            //第 8~52 行为 case...in...esac 结构, 即判断参数是“-d”、“-p”, 还是“*”,
//从而执行不同的分支
9 -d) ls /home > home_list //第 9~25 行是 case 结构中的“-d”分支, 即$1 的值为“-d”时执
//行该分支。其中“ls /home > home_list”表示将目录/home 下的目
//录结构保存到文件 home_list 中
10 while read LINE1      //第 10~24 行是 while...do...done 结构, 该结构每循环一次从
//home_list 文件中读入一个目录名称, 并依此对文件/etc/passwd 和
```



```

//文件/etc/shadow进行检查。其中“read LINE1”是控制条件，即表示
//每循环一次从指定文件读入一行数据，且保存到变量LINE1中。读入的
//文件由第24行“done < home_list”指定，该代码表示将home_list文
//件的内容输入给while循环
11     do                                // while 结构中的 do 语句
12         HOME=$LINE1                  //将变量LINE1的值赋值给变量HOME
13         grep "/home/$HOME" /etc/passwd > pass //表示从/etc/passwd文件中查找包含
// “/home/$HOME”的行，并保存到文件pass中
14         if [ $? -eq 0 ] ; then //第14~23行为if...then...else...fi结构。本行“$? -eq 0”
用于判断上
//次执行的命令（即“grep "/home/$HOME" /etc/passwd > pass”）是
//否正确执行。如果$?的值等于0，则表示
// “grep "/home/$HOME" /etc/passwd > pass”被正确执行，即grep
//命令在/etc/passwd文件中找到了匹配“/home/$HOME”的记录并被正确
//存到pass文件中，需要继续执行第15~20行，对/etc/shadow文件进行
//检查：如果$?的值不等于0，则执行第22行，显示不能匹配/etc/passwd
//文件的提示信息
15         grep "$HOME" /etc/shadow > sha //表示从/etc/shadow文件中查找包含
// “$HOME”的行，并保存到文件sha中
16         if [ $? -eq 0 ] ; then //第16~20行为if...then...else...fi结构。本行“$? -eq 0”
用于判断上
//次执行的命令（即“grep "$HOME" /etc/shadow > sha”）是否正确执
//行。如果$?的值等于0，则表示“grep "$HOME" /etc/shadow > sha”
//被正确执行，即grep命令在/etc/shadow文件中找到了匹配“$HOME”
//的记录并被正确保存到sha文件中，需要继续执行第17行，显示匹配一
//致的信息；如果$?的值不等于0，则执行第19行，显示不能匹配
//文件/etc/shadow的提示信息
17         echo "The $HOME 's directory is according with /etc/passwd and
/etc/shadow." //显示输出
// “The $HOME 's directory is according with /etc/passwd and /etc/shadow.”，即
目录与/etc/passwd及/etc/shadow
//匹配一致
18         else // if 结构中的 else 语句，匹配第16行的if
19             echo "The $HOME 's directory is not according with /etc/shadow."
//显示输出
// “The $HOME 's directory is not according with /etc/shadow.”，即目录与/etc/shadow
文件不一致
20         fi // if 结构中的 fi 语句，匹配第16行的if...then
21         else // if 结构中的 else 语句，匹配第14行的if...then
22             echo "The $HOME 's directory is not according with /etc/passwd."
//显示输出
// “The $HOME 's directory is not according with /etc/passwd.”，即目录与/etc/passwd
文件不一致
23         fi // if 结构中的 fi 语句，匹配第14行的if...then
24         done < home_list //其中“done”匹配第10行的while结构，表示循环体结束，“< home_list”
//表示while结构从文件home_list读入数据
25         ;; // “;;”表示case结构中的“-d”分支结束
26         -p) grep "/home" /etc/passwd > pas //第26~48行是case结构中的“-p”分支，即$1的
//值为“-p”时执行该分支。其中
// “grep "/home" /etc/passwd > pas”表示在
//文件/etc/passwd中查找匹配“/home”的记录，并保存
//到文件pas中
27         if [ $? -eq 0 ] ; then //第27~47行为if...then...else...fi结构。本行
// “$? -eq 0”用于判断上次执行的命令
//（即“grep "/home" /etc/passwd > pas”）是否正确

```




```

//执行。如果$?的值等于 0，则表示
//“grep "/home" /etc/passwd > pas”被正确执行，即
//grep 命令在/etc/passwd 文件中找到了匹配“/home”的
//记录并被正确保存到 pas 文件中，需要继续执行
//第 28~44 行；如果$?的值不等于 0，则执行第 46 行，
//显示无法在/etc/passwd 文件中匹配“/home”的记录
28     awk 'BEGIN { FS=":" ; } { print $6 }' pas >pas_field //表示将 pas 文件中
以“:”为分隔符的第
//6 列内容保存到文件 pas_field 中。其中“BEGIN”表示从 pas 文件每行的首字符开始处理，
“{ FS=":" ; }”表示
//指定列分隔符为冒号，“{ print $6; }”表示输出 pas 文件中第 6 列。该行代码实质是将文件 pas
中第 6 列（用户
//主目录信息）提取出来保存到文件 pas_field 中
29     awk 'BEGIN { FS="/" ; } { print $3 }' pas_field>pas_field_nopath //表
示将 pas_field 文件中以
//“/”为分隔符的第 3 列内容保存到文件 pas_field_nopath 中。其中“BEGIN”表示从 pas_field
文件每行的首字
//符开始处理，“{ FS="/" ; }”表示指定列分隔符为“/”，“{ print $3; }”表示输出 pas_field 文
件中第 3 列。该行
//代码实质是将文件 pas_field 中第 3 列（用户主目录名称，不包含路径）提取出来保存到文件
pas_field_nopath 中
30     while read LINE2 //第 30~44 行是 while...do...done 结构，该结构每循
//环一次从 pas_field_nopath 文件中读入一个用户名称，并依此对其/home 下的主目录及
/etc/shadow 文件进行一致
//性检查。其中“read LINE2”是控制条件，即表示每循环一次从指定文件读入一行数据，且保存到
变量 LINE2
//中。读入的文件由第 44 行“done<pas_field_nopath”指定，该代码表示将 pas_field_nopath
文件的内容输入给
//while 循环
31     do // while 结构中的 do 语句，匹配第 30 行的 while
32         X=$LINE2 //将变量 LINE2 的值赋值给变量 X
33         grep "$X" /etc/shadow >shad //表示从/etc/shadow 文件中查找包含“$X”的
行，并保存
//到文件 shad 中
34         if [ $? -eq 0 ] ; then //第 34~43 行为 if...then...else...fi 结构。本行
//“$? -eq 0”用于判断上次执行的命令（即“grep "$X" /etc/shadow >shad”）
是否正确执行。如果$?的值等
//于 0，则表示“grep "$X" /etc/shadow >shad”被正确执行，即 grep 命令在/etc/shadow 文
件中找到了匹配“$X”
//的记录并被正确保存到 shad 文件中，需要继续执行第 35~40 行；如果$?的值不等于 0，则执行第 42
行，显示
//不能匹配/etc/shadow 文件的提示信息
35         Y="/home/$X" //定义变量 Y，且赋值为“/home/$X”
36         if [ -d $Y ] ; then //第 36~40 行为 if...then...else...fi 结构。本行“-d $Y”
//用于判断指定的目录是否存在（目录名保存在变量 Y 中）。如果目录存在，则执行第 37 行，显示输出目录和
//文件/etc/shadow、/etc/passwd 匹配一致；如果目录不存在，则执行第 39 行，显示输出指定的主目录不
存在
37         echo "The $X is according with /etc/shadow and directory." //显示输出
//“The $X is according with /etc/shadow and directory.”，即/etc/passwd 文件中的记录
与/home 下的主目录
//及文件/etc/shadow 匹配一致
38         else // if 结构中的 else 语句，匹配第 36 行
39         echo "The $X 's home directory $Y is not exist." //显示输出“The $X

```



```
's home directory
// $Y is not exist.", 即无法匹配主目录
40         fi                                // if 结构中的 fi 语句, 匹配第 36 行的 if...then
41         else                                // if 结构中的 else 语句, 匹配第 34 行的 if...then
42         echo "The $X is not according with /etc/shadow ." //显示输出
// "The $X is not according with /etc/shadow .", 即无法匹配/etc/shadow 文件
43         fi                                // if 结构中的 fi 语句, 匹配第 34 行的 if...then
44         done < pas_field_nopath            //其中 "done" 匹配第 30 行的 while 结构, 表示循环体
结束, // "< pas_field_nopath" 表示 while 结构从文件
//pas_field_nopath 读入数据
45         else                                // if 结构中的 else 语句, 匹配第 27 行的 if...then
46         echo "Can not find /home." //显示输出 "Can not find /home.", 即找不到指定的
主目录
47         fi                                // if 结构中的 fi 语句, 匹配第 27 行的 if...then
48         ;;                                // ";;" 表示 case 结构中的 "-p" 分支结束
49         *) echo $USAGE                    //第 49~51 行是 case 结构中的 "*" 分支, 即在 $1 的值无
法匹配 "-d" 和 "-p" 分支时执行该分支。其中 "echo $USAGE" 表示显示输出变量 USAGE 的值, 即显示帮助信息
50         exit 1                            //退出程序
51         ;;                                // ";;" 表示 case 结构中的 "*" 分支结束
52 esac                                     //表示 case...in...esac 结构结束, 匹配第 8 行的 case 语句
```

该程序的执行过程如下所示。

```
#chmod u+x 20.13                        //赋予文件 20.13 可执行权限
#./20.13                               //不带参数执行 20.13
usage: -bash [ -d : According to /home directory | -p : According to /etc/passwd]
#./20.13 -p                            //配合参数 "-p" 执行 20.13
The wnn 's home directory /home/wnn is not exist.
The teacher is according with /etc/shadow and directory.
The student01 is according with /etc/shadow and directory.
The student02 is according with /etc/shadow and directory.
The student03 is according with /etc/shadow and directory.
... ..
#./20.13 -d                            //配合参数 "-d" 执行 20.13
The sullivan 's directory is not according with /etc/passwd.
The teacher 's directory is according with /etc/passwd and /etc/shadow.
The stul 's directory is not according with /etc/passwd.
The studl 's directory is according with /etc/passwd and /etc/shadow.
The studl0 's directory is according with /etc/passwd and /etc/shadow.
... ..
```

20.14 文件类型转换

Red Hat Linux 9 系统提供了丰富的联机帮助手册 (man page)。手册是 Linux 中广泛使用的联机帮助形式, 其中不仅包括了常用的命令帮助说明, 还包括配置文件、设备文件、协议和库函数等多种信息。

如今随着 WWW 的发展, 人们更习惯于通过网页查询信息。使用 man2html 命令可以方便地将 man page 格式文件转换成 html 文件, 通过网站发布, 可以更方便地提供手册信息的查询服务。



20.14.1 系统设计

例如，编写一个 Shell 程序将系统 man page 文件转换成 html 文件，并保存到网页根目录下，以方便用户通过网络进行查询。设网页根目录为/var/www，需要转换的 man page 文件目录为/usr/share/man/man1，转换后的 html 文件保存在目录/var/www/man/man1 中。

20.14.2 代码实现

该程序分三个阶段完成如下所示。

(1) 对 man page 文件进行解压缩。由于系统中的 man page 文件以.gz 压缩格式存在，在转换前需要首先使用 gunzip 进行解压缩。

(2) 将解压缩后的 man page 文件转换为 html 文件，需要使用命令 man2html。

(3) 将转换后生成的 html 文件复制到指定 Web 目录，进行发布。

程序中使用了两个 while 循环，第一个 while 循环实现文件的解压缩，第二个 while 循环实现文件类型的转换，具体代码如下所示（代码 20.14）。

```
1 #!/bin/bash //程序第1行的“#!/bin/bash”表示调用bash来执行脚本程序
2 #filename:20.14 //程序第2行的“#filename:20.14”表示该程序的名称为20.14
3 mkdir -p /man1/gzip //使用mkdir命令创建目录/man1/gzip，选项“-p”表示在父目录不存在时，
//一并创建
4 cp /usr/share/man/man1/* /man1/gzip //使用cp命令将/usr/share/man /man1目录下的
所有.gz压缩文
//件复制到目录/man1/gzip中
5 ls /usr/share/man/man1 > man1_gz_list //表示通过ls命令将/usr/share/man/man1目录
中的文件列表保存
//到/man1_gz_list文件中
6 #####Extract .gz file##### //注释，表示以下是对.gz文件解压缩
7 while read LINE1 //第7~11行是while...do...done结构，该结构每循环一次从man1_gz_list文
//件中读入一个文件名，并依此对其进行解压缩操作。其中“read LINE1”是控制
//条件，即表示每循环一次从指定文件读入一行数据，且保存到变量LINE1中。读
//入的文件由第11行“done < man1_gz_list”指定，该代码表示将/man1_gz_list
//文件的内容输入给while循环
8 do // while结构中的do语句，匹配7行的while
9 GZFILE=$LINE1 //将变量LINE1赋值给变量GZFILE
10 gunzip /man1/gzip/$GZFILE //使用gunzip命令对文件“/man1/gzip/$GZFILE”进行解压
缩操作
11 done< man1_gz_list //其中“done”匹配第7行的while结构，表示循环体结束
//“<man1_gz_list”表示while结构从文件man1_gz_list
读入数据
12 #####Convert##### //注释，表示以下代码用于文件类型的转换
13 ls /man1/gzip > man1_list //表示将/man1/gzip目录下的文件列表信息保存到文件
man1_list
14 mkdir /man1/html //使用mkdir命令创建目录/man1/html
15 while read LINE2 //第15~19行是while...do...done结构，该结构每循环一次从
//man1_list文件中读入一个文件名，并依此对其进行类型转换操作。其中“read LINE2”是控制条件，即表
示每
//循环一次从指定文件读入一行数据，且保存到变量LINE2中。读入的文件由第19行“done < man1_list”
指定，
//该代码表示将man1_list文件的内容输入给while循环
16 do // while结构中的do语句，匹配第15行的while
```



```

17 FILE=$LINE2 //将变量 LINE2 赋值给变量 FILE
18 man2html $FILE > /man1/html/$FILE.html //表示使用 man2html 命令将 man 类型文件转换
为 html 文件,
//并保存到目录/man1/html 中
19 done < man1_list //其中“done”匹配第 15 行的 while 结构,表示循环体结
//束,“< man1_list”表示 while 结构从文件 man1_list 读入数据
20 #####Copy#####//注释,表示以下部分代码实现复制功能
21 mkdir -p /var/www/man1 //使用 mkdir 命令创建目录/var/www/man1。其中选项
“-p”表
//示如果指定的父目录不存在,则自动创建,如果指定的目录已存在,不报错
22 cp -rf /man1/html /var/www/man/man1 //使用 cp 命令将目录
/man1/html 中的文件
//复制到目录/var/www/man/man1 中。其中选项“-f”表示强制执行复制,“-r”表示复制子目录
该程序的执行过程如下所示。

```

```

#chmod u+x 20.14 //设置文件 20.14 可执行权限
#./20.14 //执行程序 20.14
# ls /var/www/man/man1 //显示转换后,复制到 Web 目录下的 html 文件
:~.1.html ldapmodify.1.html pnmtopng.1.html
[.1.html ldapmodrdn.1.html pnmtops.1.html
4l1toppm.1.html ldappasswd.1.html pnmtorast.1.html
a2p.1.html ldapsearch.1.html pnmtorle.1.html
a2ps.1.html ldd.1.html pnmtosgi.1.html
ab.1.html leaftoppm.1.html pnmtosir.1.html
ac.1.html less.1.html pnmtotiff.1.html
access.1.html lesskey.1.html pnmtotiffcmyk.1.html
achfile.1.html let.1.html pnmtowd.1.html
acinclude.1.html lex.1.html pod2html.1.html

```

20.15 对输入的 3 个数字进行排序

对输入的多个数字按指定顺序进行排序输出是学习各种编程语言的基本训练之一。本例用 if 语句编写一个 Shell 程序,对输入的 3 个数按从小到大的顺序进行排序,并显示排序过程及最终排序结果。

20.15.1 系统设计

在读入排序数据时可以采用以下两种方法。

(1) 使用位置参数读入排序数据。其中“\$#”表示参数的数目,“\$1”表示参数 1,“\$2”表示参数 2,“\$3”表示参数 3。

(2) 通过 read 语句读入数据。

本例将对应两种数据输入方法分别编写 Shell 程序。

20.15.2 代码实现

以下代码使用 read 语句读入待排序的数字,通过 3 个 if 结构完成了数字的排序,具体代码如下所示(代码 20.15.1)。

```

1 #!/bin/bash //程序第 1 行的“#!/bin/bash”表示调用 bash 来执行脚本程序
2 #filename:20.15.1 //程序第 2 行的“#filename:20.15.1”表示该程序的名称为 20.15.1

```



```

3 echo "Please enter three digit to compare://"用于显示输出 "Please enter three digit to
compare:",
//提示输入 3 个数字
4 read XX YY ZZ //用于读入用户的输入, 将用户输入的第 1 个数字赋值
//给变量 XX, 第 2 个数字赋值给变量 YY, 第 3 个数字
//赋值给变量 ZZ
5 echo "Your enter is $XX $YY $ZZ" //用于显示输出 "Your enter is $XX $YY $ZZ", 即
//显示用户输入的 3 个数字
6 if [ $XX -gt $YY ];then //第 6~11 行是 if...then...fi 结构, 用于对变量 XX,
//YY 进行比较, 如果变量 XX 的值大于变量 YY, 则进行交换。本行 "$XX -gt $YY" 表示判断变量 XX 是否大于
//变量 YY, 如果大于则执行第 7~11 行, 否则跳过第 7~11 行
7 KK=$YY //将变量 YY 的值赋值给变量 KK
8 YY=$XX //将变量 XX 的值赋值给变量 YY
9 XX=$KK //将变量 KK 的值赋值给变量 XX
10 echo " Current order : $XX $YY $ZZ " //显示输出 "Current order : $XX $YY $ZZ", 即显
//示当前的数字序列
11 fi // if 结构中的 fi 语句, 匹配第 6 行的 if, 用于表示该
if 结构结束
12 if [ $YY -gt $ZZ ] ; then //第 12~17 行是 if...then...fi 结构, 用于对变量 YY, ZZ 进
//行比较, 如果变量 YY 的值大于变量 ZZ, 则进行交换。本行 "$YY -gt $ZZ" 表示判断变量 YY 是否大于变量
//ZZ, 如果大于则执行第 13~17 行, 否则跳过第 13~17 行
13 KK=$YY //将变量 YY 的值赋值给变量 KK
14 YY=$ZZ //将变量 ZZ 的值赋值给变量 YY
15 ZZ=$KK //将变量 KK 的值赋值给变量 ZZ
16 echo " Current order : $XX $YY $ZZ " //显示输出 "Current order : $XX $YY $ZZ",
//即显示当前的数字序列
17 fi // if 结构中的 fi 语句, 匹配第 12 行的 if, 用于表示该 if
结构结束
18 if [ $XX -gt $YY ] ; then //第 18~23 行是 if...then...fi 结构, 用于对变量 XX, YY 进
//行比较, 如果变量 XX 的值大于变量 YY, 则进行交换。本行 "$XX -gt $YY" 表示判断变量 XX 是否大于变量
//YY, 如果大于则执行第 19~23 行, 否则跳过第 19~23 行
19 KK=$YY //将变量 YY 的值赋值给变量 KK
20 YY=$XX //将变量 XX 的值赋值给变量 YY
21 XX=$KK //将变量 KK 的值赋值给变量 XX
22 echo " Current order : $XX $YY $ZZ " //用于显示输出 "Current order : $XX $YY $ZZ",
即显示当
//前的数字序列
23 fi //if 结构中的 fi 语句, 匹配第 18 行的 if, 用于表示该
if 结构结束
24 echo " The result order is : $XX $YY $ZZ " //显示输出 "The result order is : $XX
$YY $ZZ", 即显示
//最终的排序结果

```

以下代码通过位置参数读入待排序的数字, 并通过三个 if 结构完成了数字的排序, 具体如下所示 (代码 20.15.2)。

```

1 #!/bin/bash //程序第 1 行的 "#!/bin/bash" 表示调用 bash 来执行脚本程序
2 #filename:20.15.2 //程序第 2 行的 "#filename:20.15.2" 表示该程序的名称为 20.15.2
3 if [ $# -lt 3 ];then //第 3~29 行为 if...then...else...fi 结构。本行 "$# -lt 3" 表示该程序脚
本的运行
//参数数目如果小于 3 则执行第 4 行, 否则执行第 6~29 行。其中 "$#" 表示该程序
//的运行参数数目
4 echo "Warning!Need more argument!" //显示输出 "Warning!Need more argument!", 即提
示用户输入的
//参数过少
5 else //if 结构中的 else 语句

```



```

6  echo "Your digits to compare is $1 $2 $3"    //显示输出 "Your digits to compare is
$1 $2 $3",
                                           //即显示位置参数的值
7  XX=$1    //将位置参数 1 的值赋值给变量 XX
8  YY=$2    //将位置参数 2 的值赋值给变量 YY
9  ZZ=$3    //将位置参数 3 的值赋值给变量 ZZ
10 if [ $XX -gt $YY ];then                //第 10~15 行是 if...then...fi 结构, 用于对变量 XX, YY 进行
比较,
    //如果变量 XX 的值大于变量 YY, 则进行交换。本行 "$XX -gt $YY" 表示判断变量 XX 是否大于变量
YY, 如果
    //大于则执行第 11~15 行, 否则跳过第 11~15 行
11  KK=$YY    //将变量 YY 的值赋值给变量 KK
12  YY=$XX    //将变量 XX 的值赋值给变量 YY
13  XX=$KK    //将变量 KK 的值赋值给变量 XX
14  echo "Current order is $XX $YY $ZZ"    //用于显示输出 "Current order : $XX $YY
$ZZ", 即
                                           //显示当前的数字序列
15  fi                //if 结构中的 fi 语句, 匹配第 10 行的 if, 用于表示该 if
结构结束
16  if [ $YY -gt $ZZ ] ; then            //第 16~21 行是 if...then...fi 结构, 用于对变量 YY, ZZ 进
//行比较, 如果变量 YY 的值大于变量 ZZ, 则进行交换。本行 "$YY -gt $ZZ" 表示判断变量 YY 是
否大于变量
    //ZZ, 如果大于则执行第 17~21 行, 否则跳过第 17~21 行
17  KK=$YY    //将变量 YY 的值赋值给变量 KK
18  YY=$ZZ    //将变量 ZZ 的值赋值给变量 YY
19  ZZ=$KK    //将变量 KK 的值赋值给变量 ZZ
20  echo "Current order is $XX $YY $ZZ"    //显示输出 "Current order : $XX $YY $ZZ",
即显示当前
                                           //的数字序列
21  fi                //if 结构中的 fi 语句, 匹配第 16 行的 if, 用于表示该
if 结构结束
22  if [ $XX -gt $YY ]; then            //第 22~27 行是 if...then...fi 结构, 用于对变量 XX, YY 进
//行比较, 如果变量 XX 的值大于变量 YY, 则进行交换。本行 "$XX -gt $YY" 表示判断变量 XX 是否大于变量
//YY, 如果大于则执行第 23~27 行, 否则跳过第 23~27 行
23  KK=$YY    //将变量 YY 的值赋值给变量 KK
24  YY=$XX    //将变量 XX 的值赋值给变量 YY
25  XX=$KK    //将变量 KK 的值赋值给变量 XX
26  echo "Current order is $XX $YY $ZZ"    //显示输出 "Current order : $XX $YY $ZZ",
即显示当前
                                           //的数字序列
27  fi                //if 结构中的 fi 语句, 匹配第 22 行的 if, 用于表示该 if 结
构结束
28  echo "The result order is $XX $YY $ZZ"    //用于显示输出 "The result order is $XX
$YY $ZZ",
                                           //即输出最后的排序结果
29  fi                //if 结构中的 fi 语句, 匹配第 3 行中的 if, 表示 if 结构
结束

```

程序 20.15.1 执行过程如下所示。

```

#chmod u+x 20.15.1                //赋予文件 20.15.1 可执行权限
#./20.15.1                        //执行程序 20.15.1
Please enter three digit to compare:
78 56 34                          //输入待排序的数字
Your enter is 78 56 34
Current order : 56 78 34
Current order : 56 34 78
Current order : 34 56 78
The result order is : 34 56 78    //最终排序结果

```




```
# ./20.15.1 //执行程序 20.15.1
Please enter three digit to compare:
56 12 42 //输入待排序的数字
Your enter is 56 12 42
Current order : 12 56 42
Current order : 12 42 56
The result order is : 12 42 56 //最终排序结果
```

程序 20.15.2 执行过程如下所示。

```
#chmod u+x 20.15.2 //赋予文件 20.15.2 可执行权限
#./20.15.2 87 56 34 //执行程序 20.15.2, 参数为 87、56、34
Your digits to compare is 87 56 34
Current order is 56 87 34
Current order is 56 34 87
Current order is 34 56 87
The result order is 34 56 87 //最终排序结果
#./20.15.2 82 12 78 //执行程序 20.15.2, 参数为 82、12、78
Your digits to compare is 82 12 78
Current order is 12 82 78
Current order is 12 78 82
The result order is 12 78 82 //最终排序结果
#./20.15.2 82 12 //执行程序 20.15.2, 参数为 82、12, 缺少 1 个参数
Warning!Need more argument! //显示警告信息
#./20.15.2 82 //执行程序 20.15.2, 参数为 82、12, 缺少 2 个参数
Warning!Need more argument! //显示警告信息
#./20.15.2 //执行程序 20.15.2, 参数为 82、12, 缺少 3 个参数
Warning!Need more argument! //显示警告信息
#./20.15.2 88 45 23 12 //执行程序 20.15.2, 参数为 88、45、23、12, 多 1 个参数
Your digits to compare is 88 45 23
Current order is 45 88 23
Current order is 45 23 88
Current order is 23 45 88
The result order is 23 45 88 //最终排序结果, 仅对前 3 个参数进行排序
```

20.16 计算奇数之和

编写程序计算数字序列之和、差、积、阶乘、平方和、方根和等，是学习程序语言，尤其是掌握 while、until 等循环控制语句经常进行的基本训练。本例编写一个 Shell 脚本，计算 1~50 之间的奇数之和并显示计算结果。

20.16.1 系统设计

编写 Shell 程序计算 1~50 之间所有奇数之和。本例采用 while 和 until 两种结构实现。

20.16.2 代码实现

程序 20.16.1 中使用 while 循环实现了对 1~50 之间所有奇数之和的计算。循环变量初值为 1，每循环一次，自加 2，依次取值 1、3、5、7、9...，当超出 50 时，结束循环，从而实现了对所有奇数的累加。该程序具体代码如下所示（代码 20.16.1）。


```

1 #!/bin/bash           //程序第 1 行的“#!/bin/bash”表示调用 bash 来执行脚本程序
2 #filename:20.16.1     //程序第 2 行的“#filename:20.16.1”表示该程序的名称为 20.16.1
3 i=1                   //定义循环变量 i，赋初值为 1
4 SUM=0                 //定义变量 SUM，用于保存奇数和，赋初值为 0
5 while [ $i -le 50 ]   //第 5~9 行是 while...do...done 结构，用于循环计算奇数和，每循环一次变
量 i 取
    //得一个奇数，并与部分和 SUM 相加。本行“$i -le 50”是 while 循环的控制条件，即小于等于 50 时，执行
while
    //循环体，大于 50 时结束循环
6 do                     // while 结构中的 do 语句
7     SUM=$((SUM+$i))    //将变量 SUM 与变量 i 相加，赋值给变量 SUM
8     i=$((i+2))         //将变量 i 自加 2
9 done                   //while 结构中的 done 语句，表示 while 循环体结束
10 echo "The sum of odd number is:$SUM" //显示输出“The sum of odd number is:$SUM”。其中
    //$SUM 为变量 SUM 的值

```

程序 20.16.2 中使用 until 循环实现了对 1~50 之间所有奇数之和的计算。循环变量初值为 1，每循环一次，自加 2，且每循环一次检验其值是否为 51，若取值为 51，结束循环。该程序具体代码如下所示（代码 20.16.2）。

```

1 #!/bin/bash           //程序第 1 行的“#!/bin/bash”表示调用 bash 来执行脚本程序
2 #filename:20.16.2     //程序第 2 行的“#filename:20.16.1”表示该程序的名称为 20.16.1
3 i=1                   //定义循环变量 i，赋初值为 1
4 SUM=0                 //定义变量 SUM，用于保存奇数和，赋初值为 0
5 until                 //第 5~11 行是 until...do...done 结构。本行“until”表示该结构开始
6 test $i = 51          //用于检测变量 i 的值是否等于 51，如果等于 51 则结束 until...do...done
循环，否则继
    //续执行循环体第 8~9 行
7 do                     // until 结构中的 do 语句，表示循环体开始
8     SUM=$((SUM+$i))    //将变量 SUM 与变量 i 相加，赋值给变量 SUM
9     i=$((i+2))         //将变量 i 自加 2
10 done                  //until 结构中的 done 语句，表示循环体结束
11 echo "The sum of odd number is:$SUM" //显示输出“The sum of odd number is:$SUM”。其中
    //$SUM 为变量 SUM 的值

```

执行程序 20.16.1 及 20.16.2，如下所示。

```

#chmod u+x 20.16.1      //赋予文件 20.16.1 执行权限
#chmod u+x 20.16.2      //赋予文件 20.16.2 执行权限
#./20.16.1              //执行程序 20.16.1
The sum of odd number is : 625
#./20.16.2              //执行程序 20.16.2
The sum of odd number is : 625

```

20.17

使用 select 命令生成菜单

编写 Shell 程序使用 select 命令生成选择菜单，允许用户在菜单中选择，并基于选择执行相应的命令。

20.17.1 系统设计

select 循环的主要作用是生成菜单，菜单中每一选项前都有数字和右括号。在菜单底部显示提示符，等待用户输入选择。通过环境变量 PS3 可以设定提示符，默认为“#?”。



本例中提示符设定为“请选择需要执行的命令:”，菜单中显示的命令分别如下。

- “ls -l”: 以长格式显示当前的目录。
- “date +%y”: 显示当前的年份。
- pwd: 显示当前工作目录。
- who: 显示当前登录系统的用户。
- “cd /root”: 将当前目录切换到/root。
- ps: 显示进程。
- df: 显示当前系统磁盘的使用情况。

按下 Ctrl+c 键，将结束 select 循环。

20.17.2 代码实现

在 select 循环列表中放置需要执行的命令，对于带选项的命令需要使用双引号，具体代码如下所示。

```
1 #!/bin/bash          //程序第1行的“#!/bin/bash”表示调用bash来执行脚本程序
2 #filename:20.17      //程序第2行的“#filename:20.17”表示该程序的名称为20.17
3 PS3="请选择需要执行的命令: " //环境变量PS3赋值“请选择需要执行的命令:”
4 select cmd in "ls -l" "date +%y" pwd who "cd /root" ps df //第4~7行为select循环。循环列表为
//“ls -l”、“date +%y”、pwd、who、“cd /root”、ps和df。列表中的每一项会在菜单中显示，并等待用户选
//项其中某一项。当用户选择了某一项之后，Shell会把该选项赋值给变量cmd，并执行select结构体中的命令
5 do                  //do表示循环体开始
6 $cmd                //执行变量cmd表示的命令
7 done               //done表示循环体结束
```

执行程序20.17，如下所示。

```
#chmod u+x 20.17          //赋予程序20.17可执行权限
#./20.17                  //执行程序20.17
1) ls -l
2) date +%y
3) pwd
4) who
5) cd /root
6) ps
7) df
请选择需要执行的命令: 7 //输入“7”
文件系统      1K-块      已用      可用 已用% 挂载点
/dev/sda2      7637400  5129808  2119624  71% /
/dev/sda1      101089   9425     86445   10% /boot
none           127632   0        127632  0% /dev/shm
请选择需要执行的命令: 6 //输入“6”
  PID TTY          TIME CMD
22500 pts/5    00:00:00 bash
24910 pts/5    00:00:00 ps
请选择需要执行的命令: 5 //输入“5”
请选择需要执行的命令: 4 //输入“4”
root      :0                Aug 10 09:08
root      pts/2            Aug 11 16:40 (:0.0)
root      pts/4            Aug 20 12:50 (192.168.20.148)
root      pts/5            Aug 20 13:00 (192.168.20.148)
```



```

请选择需要执行的命令: 3           //输入“3”
/root
请选择需要执行的命令: 2           //输入“2”
08
请选择需要执行的命令: 1           //输入“1”
总用量 228
-rw-r--r--  1 root    root          0  8月 11 04:40 *
-rw-r--r--  1 root    root        960  8月  7 15:43 anaconda-ks.cfg
-rw-r--r--  1 root    root        478  8月 10 18:43 dump.tmp
drwxr-xr-x  2 root    root       4096  8月 14 19:17 etcbak
... ..
请选择需要执行的命令:             //按 Ctrl+c 组合键
#

```

20.18 打印图形

通过灵活地使用 `while` 循环及 `echo` 或 `printf` 命令，可以输出比较复杂的格式，从而使 Shell 程序的输出更美观，可读性更强。

20.18.1 系统设计

编写 Shell 程序输出由数字组成的图形（如下所示），其中数字图形共 10 行，每行为从 0~N-1 个数字。

```

0
0 1
0 1 2
0 1 2 3
0 1 2 3 4
0 1 2 3 4 5
0 1 2 3 4 5 6
0 1 2 3 4 5 6 7
0 1 2 3 4 5 6 7 8
0 1 2 3 4 5 6 7 8 9

```

20.18.2 代码实现

通过两层 `while` 循环完成数字图形的输出，其中内层 `while` 循环控制行内的输出，外层 `while` 循环控制所有行的输出，代码如下所示。

```

1 #!/bin/bash           //表示调用 bash 来执行该脚本
2 #filename:20.18       //注释，表示当前脚本文件名为 20.18
3 x=0                  //变量 x 赋值为 0
4 while [ $x -lt 10 ]   //第 4~14 行为 while 循环，循环控制条件为 "$x -lt 10"
5 do                   //表示循环体开始，匹配第 4 行的 while
6   x=$((x+1))         //变量 x 自加 1
7   y=0               //变量 y 赋值为 0
8   while [ $y -lt $x ] //第 8~12 行为 while 循环，循环控制条件为 "$y -lt $x"
9   do                //表示循环体开始，匹配第 8 行的 while
10    echo -n " $y"    //不换行显示输出变量 y 的值
11    y=$((y+1))      //变量 y 自加 1

```



```
12 done //表示 while 循环体结束，匹配第 8 行的 while
13 echo //输出回车
14 done //表示 while 循环体结束，匹配第 4 行的 while
```

代码执行过程如下所示。

```
#chmod u+x 20.18 //赋予文件 20.18 可执行权限
# ./20.18 //执行程序 20.18
0
0 1
0 1 2
0 1 2 3
0 1 2 3 4
0 1 2 3 4 5
0 1 2 3 4 5 6
0 1 2 3 4 5 6 7
0 1 2 3 4 5 6 7 8
0 1 2 3 4 5 6 7 8 9
```

20.19 处理压缩文件

压缩文件的处理主要是指对文件进行压缩、解压缩，以及对压缩文件进行检测，通常是由压缩命令的不同选项进行控制。通过编写 Shell 程序可以实现按需对压缩文件进行相关处理。

20.19.1 系统设计

例如，编写 Shell 程序，根据用户输入的位置参数及文件或目录名进行相应的处理。当用户输入的位置参数 1 的值为“-c”时，使用 tar 命令及“-zcvf”选项对指定的文件或目录进行压缩；当用户输入的位置参数 1 的值为“-t”时，对指定的文件进行检测，输出 tar 包中被压缩文件的相关信息；当用户输入的位置参数 1 的值为“-x”时，对指定的文件进行解压缩。位置参数 2 为待处理的文件或目录。

20.19.2 代码实现

该程序采用 case 结构对用户输入的不同参数分别执行不同的处理。对于指定多个文件的情况，使用 for 循环逐一提取每个文件名，代码如下所示（代码 20.19）。

```
1 #!/bin/bash //表示调用 bash 执行该脚本
2 #filename:20.19 //注释，表示该脚本的名为 20.19
3 USAGE="Usage: 'basename $0' [-c|-t|-x][files|directories]" // 将
"Usage: 'basename
//"$0' [-c|-t|-x][files|directories]" 赋值给变量 USAGE，其中
"'basename $0'" 表示提取当前脚本的脚本名
4 if [ $# -lt 2 ] //第 4~7 行为 if...then...fi 结构，判断条件为 "$# -lt 2"
5 then //if 结构中的 then 语句
6 echo "$USAGE" //显示输出变量 USAGE
7 fi //if 结构中的 fi 语句
8 case "$1" in //第 8~36 行为 case 结构，该结构对变量 $1 的值进行判断，并根据结果值执行不
同的
```


//分支。定义了“-t|-x”分支（第 9~28 行），定义了“-c”分支（第 29~33 行），定义了“*”分支（第 34、

//35 行）

9 -t|-x) TARGS=\${1}zvf //第 9~28 行为 case 分支，当变量\$1 的值为“-t”或“-x”时执行该分支。本行为变量

//TARGS 赋值\${1}zvf，即-tzvf 或-xzvf

10 shift

//位置参数左移一位

11 for i in "\$@"

文件名)

//第 11~27 行为 for 循环，变量 i 在每次循环依次取得一个位置参数（一个

12 do

//for 循环的 do 语句

13 if [-f "\$i"]

所表示

//第 13~26 行为 if...then...else...fi 结构，判断条件为“-f "\$i"”，即变量 i

//的文件名是否是普通文件

14 then

//if 结构中的 then 语句

15 FILES='tar \$TARGS "\$i" 2>/dev/null'

\$i”的执行结

//将命令“tar -tzvf \$i”或“tar -xzvf

//果赋值给变量 FILES，且将命令的错误输出定向到/dev/null，即不在屏幕显示错误输出信息

16 if [\$? -eq 0]

0”，即

//第 16~23 行为 if...then...else...fi 结构。判断条件为“\$? -eq

//对上一命令是否正确执行进行判断

17 then

//if 结构中的 then 语句，匹配第 16 行的 if

18 echo

//输出回车

19 echo "\$i"

//输出变量 i

20 echo "\$FILES"

//输出变量 FILES

21 else

//if 结构中的 else 语句，匹配第 16 行的 if

22 echo "ERROR: \$i no ta tarfile."

//显示输出“ERROR: \$i no ta tarfile.”

23 fi

//if 结构中的 else 语句，匹配第 16 行的 if

24 else

//if 结构中的 else 语句，匹配第 13 行的 if

25 echo "ERROR: \$i not a file."

//显示输出“ERROR: \$i not a file.”

26 fi

//if 结构中的 else 语句，匹配第 13 行的 if

27 done

//for 结构中的 done 语句，表示 for 结构体结束

28 ;;

//“-t|-x”分支结束

29 -c) shift

//第 29~33 行为“-c”分支，即变量\$1 的值为“-c”

执行该分支。本行表示位置参数左移一位

30 TARGS="-zcvf"

//变量 TARGS 赋值“-zcvf”

31 echo \$@

//显示输出所有的位置参数

32 tar \$TARGS archive.tar.gz \$@

//执行命令“tar -zcvf archive.tar.gz \$@”

33 ;;

//“-c”分支结束

34 *) echo "\$USAGE"

//显示输出变量 USAGE

35 ;;

//“*”分支结束

36 esac

//case 结构结束

执行该程序，如下所示。

ls //显示当前目录

20.1 20.11 20.13 20.15.1 20.16.1 20.17 20.19 20.3 20.5 20.7 20.9

20.10 20.12 20.14 20.15.2 20.16.2 20.18 20.2 20.4 20.6 20.8

#chmod u+x 20.19 //赋予文件 20.19 可执行权限

./20.19 kk //带参数“kk”执行程序 20.19

Usage: 20.19[-c|-t|-x][files|directories]

./20.19 -c 20.1 20.2 //带参数“-c”、“20.1”、“20.2”执行程序 20.19

20.1 20.2

20.1

20.2

ls //显示当前目录

20.1 20.12 20.15.1 20.16.2 20.19 20.4 20.7 archive.tar.gz



```
20.10 20.13 20.15.2 20.17 20.2 20.5 20.8
20.11 20.14 20.16.1 20.18 20.3 20.6 20.9
# ./20.19 -t archive.tar.gz //带参数“-t”、“archive.tar.gz”执行程序 20.19

archive.tar.gz
-rwxr-xr-x root/root      128 2008-08-15 04:44:58 20.1
-rwxr-xr-x root/root      343 2008-08-15 06:24:34 20.2
# ./20.19 -x archive.tar.gz //带参数“-x”、“archive.tar.gz”执行程序 20.19

archive.tar.gz
20.1
20.2
#
```

20.20 数值进制转换

printf 命令可以实现数据的格式化输出，其使用的格式说明符通常是“%”后跟一个字符。例如，“%f”表示按浮点数输出，“%d”表示按整数输出，“%o”表示按八进制输出，“%x”表示按十六进制输出，“%e”表示按科学计数法进行输出。

20.20.1 系统设计

例如编写 Shell 程序，当用户输出的位置参数 1 为“-o”时，对位置参数 2 指定的数值按八进制进行输出；当位置参数 1 为“-x”时，对位置参数 2 指定的数值按十六进制进行输出；当位置参数 1 为“-e”时，对位置参数 2 指定的数值按科学计数法进行输出。

20.20.2 代码实现

该程序采用 case 结构对用户输入的不同参数分别执行不同的处理，代码如下所示（代码 20.20）。

```
1 #!/bin/bash //表示调用 bash 执行该程序
2 #filename:20.20 //注释，表示该脚本名称为 20.20
3 if [ $# -lt 2 ] //第 3~6 行为 if...then...fi 结构，判断条件为“$# -lt 2”，即变量$#是
否小于 2，其
//中$#表示位置参数数目
4 then //if 结构中的 then 语句
5 echo "ERROR: Insufficient arguments" //显示输出“ERROR: Insufficient arguments”
6 exit 1 //退出
7 fi //if 结构中的 fi 语句，匹配第 3 行的 if
8 case $1 in //第 8~17 行为 case 结构，根据变量$1 的值转向到不同的分支
9 -o) printf "%o \n" "$2" //case 结构中的“-o”分支，使用 printf 命令按十六进制输出
位置参数 2
10 ;; //表示“-o”分支结束
11 -x) printf "%x \n" "$2" //case 结构中的“-x”分支，使用 printf 命令按十六进制输出
位置参数 2
12 ;; //表示“-x”分支结束
13 -e) printf "%e \n" "$2" //case 结构中的“-e”分支，使用 printf 命令按十六进制输出
位置参数 2
14 ;; //表示“-e”分支结束
15 *) echo "Error!Please enter -o,-x or -e" //显示输出“Error!Please enter -o,-x or -e”
```



```
16 ;; //表示“*”分支结束
17 esac //表示case结构结束
```

该程序的执行过程如下所示。

```
#chmod u+x 20.20 //赋予文件20.20执行权限
# ./20.20 //执行程序20.20
ERROR: Insufficient arguments
# ./20.20 -o //带参数“-o”执行程序20.20
ERROR: Insufficient arguments
# ./20.20 -o 64 //带参数“-o”、“64”执行程序20.20
100
# ./20.20 -x 64 //带参数“-x”、“64”执行程序20.20
40
# ./20.20 -e 64 //带参数“-e”、“64”执行程序20.20
6.400000e+01
```

20.21 使用 sed 打印指定行

sed 是一个流编辑器（非交互式），可以对来自文件或标准输入的输入流进行转换。本例中，通过 sed 命令将 df 命令输出结果中那些没有以 “/” 开头的行过滤掉。

20.21.1 系统设计

df 命令可以用来显示文件系统占用的磁盘空间情况，如下所示。

```
# df
文件系统      1K-块    已用    可用    已用%    挂载点
/dev/sda2      7637400 5032868 2216564   70%     /
/dev/sda1      101089   9425    86445    10%     /boot
fd              0         0         0         0%     /dev/fd
none           127632    0    127632    0%     /dev/shm
```

可以看到，输出结果中有一些没有以 “/” 开始的行，使用 sed 命令可以过滤掉这些行。

20.21.2 代码实现

可以采用两种方法过滤掉没有以 “/” 开始的行，如下所示。

- 显示输出以 “/” 开始的行；
- 输出时，删除不是以 “/” 开始的行。

采用第一种方法，执行命令如下所示。

```
# df | sed -n '/^\//p'
/dev/sda2      7637400 5207348 2042084   72%     /
/dev/sda1      101089   9425    86445    10%     /boot
```

从输出结果可以看到，没有以 “/” 开始的行没有被显示。命令中使用了正则表达式 “^/”，该表达式表示匹配首字符为 “/” 的行，“p” 指令表示显示输出。sed 命令默认显示输出所有行，使用选项 “-n” 表示关闭默认（即不显示输出任何行），但对匹配 p 指令的行，允许显示输出。

采用第二种方法，执行命令如下所示。

```
# df | sed '/^[^\//]/d'
/dev/sda2      7637400 5207348 2042084   72%     /
/dev/sda1      101089   9425    86445    10%     /boot
```



从输出结果可以看到，没有以“/”开始的行从显示输出中被删除了。命令中使用了正则表达式“`^[^/]`”，其中第一个“`^`”表示匹配行首字符，第二个“`^`”表示取非，“`d`”指令表示删除匹配所行。

20.22 使用 sed 筛选普通文件

使用“`ls -al`”命令会将所有文件、目录显示输出，但如果用户想查看某一类文件，例如链接文件或设备文件等，可以借助 `sed` 对 `ls` 命令的输出结果进行筛选。

20.22.1 系统设计

直接使用“`ls -al`”命令会显示指定目录下的所有文件，如下所示。

```
# ls -al /bin
总用量 5520
drwxr-xr-x  2 root    root      4096  8月  7 15:37 .
drwxr-xr-x 26 root    root      4096  8月 17 22:11 ..
-rwxr-xr-x  1 root    root      2644 2003-02-25 arch
-rwxr-xr-x  1 root    root     92444 2003-02-06 ash
-rwxr-xr-x  1 root    root    492968 2003-02-06 ash.static
-rwxr-xr-x  1 root    root    10456 2003-01-25 aumix-minimal
lrwxrwxrwx  1 root    root         4  8月  7 14:34 awk -> gawk
-rwxr-xr-x  1 root    root    10848 2003-02-19 basename
... ..
```

可以看到，输出结果中既包括目录，也包括普通文件、链接文件等。如果用户仅希望显示普通文件（文件类型位为“-”），可以使用 `sed` 命令过滤掉输出结果中行首字符不是“-”的行。

20.22.2 代码实现

为了将“`ls -al`”命令的输出结果输入 `sed`，使用管道“`|`”执行命令如下所示。

```
# ls -al /bin | sed '/^[^\/]/d'
-rwxr-xr-x  1 root    root      2644 2003-02-25 arch
-rwxr-xr-x  1 root    root     92444 2003-02-06 ash
-rwxr-xr-x  1 root    root    492968 2003-02-06 ash.static
-rwxr-xr-x  1 root    root    10456 2003-01-25 aumix-minimal
-rwxr-xr-x  1 root    root    10848 2003-02-19 basename
```

20.23 使用 gawk 反序输出各列

`gawk` 是模式扫描和处理语言，又称过滤器，是 Linux/UNIX 下用来处理数据的程序语言。其处理的数据可以来自标准输入或文件，也可以是其他命令的输出。使用 `gawk` 可以方便地操作字段（列），对字段进行编辑或输出指定的字段。

20.23.1 系统设计

例如,编写一个 Shell 程序反序输出指定文件各字段的内容。其中使用了 gawk 变量 NF, NF 中的当前记录包括的字段数。需要反序输出的文本文件如下,其中各列之间以 Tab 作为分隔符。

```
# cat funds
实验材料费      10      实验板
仪器设备费      1.2    试验所需的实验箱,功能模块
实验室改装费    0      网线、交换机
协作费          0      无
项目津贴费      1.2    3000 元/人/年×4 人=1.2 万元
管理费          0.6    按北京市和学校相关规定
```

20.23.2 代码实现

利用 for 循环,在输出每一行数据时,按字段号从大到小的顺序输出,即可实现反序输出。

```
1 #!/bin/bash                                //表示调用 bash 执行该程序
2 #filename:20.23                            //注释,表示该脚本名称为 20.23
3 if [ $# -lt 1 ]                            //第 3~7 行为 if...then...fi 结构,用于判断位置参数
是否小于 1,如果小于 1 则退出
4 then                                       //if 结构中的 then 语句
5   echo "USAGE: 'basename $0' filename "   //显示输出 "USAGE: 'basename $0' filename "
6   exit 1                                  //退出
7 fi                                         //if 结构中的 fi 语句,表示 if 结构结束
8 awk ' BEGIN{FS="\t"}                     //指定变量 FS 的值为 "\t",即分隔符为 Tab
9   {                                        //语句块开始
10     for (i=NF; i>=1 ;i--)                //第 10~13 行为 for 循环,循环变量赋初值为 NF,
才结束循环                                //第次循环完毕变量 i 自减 1,直至变量 i 值为 1 时
11     {                                    //for 循环体开始
12       printf ("%s\t", $i)                //按字符串格式输出第 i 个字段
13     }                                    //for 循环体结束
14     printf("\n")                          //输出回车
15   }' $1                                  //使用 gawk 对位置参数 1 指定的文件进行处理
```

该程序的执行过程如下所示。

```
#chmod u+x 20.23                            //赋予文件 20.23 执行权限
# ./20.23 funds                             //执行程序 20.23
实验板 10      实验材料费
试验所需的实验箱,功能模块      1.2      仪器设备费
网线、交换机      0      实验室改装费
无      0      协作费
3000 元/人/年×4 人=1.2 万元      1.2      项目津贴费
按北京市和学校相关规定 0.6      管理费
```

20.24 对文件进行大小写转换

使用 tr 命令可以将读入的字符串转换为指定的字符串格式,例如对字符串的大小写进行



转换。在 Shell 程序中，通过灵活地使用 `tr` 命令可以对整个文件进行大小写转换处理。

20.24.1 系统设计

编写一个 Shell 程序，对用户指定的文件进行大小写转换处理。如果用户在命令行指定了参数“-L”，则将文件中的所有字符转换为小写；如果用户在命令行中指定的参数是“-U”，则将文件中所有字符转换为大写。

20.24.2 代码实现

程序中使用 `case` 分支对用户输入的参数“-L”或“-U”分别进行处理。在每一分支中使用了 `while` 循环，`while` 在每次循环时从指定的文件依次读入一行字符串，输出给 `tr` 命令进行处理，处理结果保存在“_low”或“_upper”文件中。代码如下所示（代码 20.24）。

```

1 #!/bin/bash           //表示调用 bash 执行该程序
2 #filename:20.24       //注释，表示该脚本名称为 20.24
3 if [ $# -lt 2 ]       // 第 3~7 行为 if...then...fi 结构，用于判断位置参数是否小于 2，如果小于 2 则
退出
4 then                 //if 结构中的 then 语句
5   echo "Error:insufficient argument! -U,-L and filename" // 显示 输出
"Error:insufficient argument!
                        //-U,-L and filename"
6   exit 1             //退出
7 fi                  //if 结构中的 fi 语句
8 case $1 in          //第 8~21 行为 case 结构，对用户输入的位置参数 1 进行判断，根据其值执行不同的
分支
9  -L)while read LINE //第 9~13 行为 case 结构的“-L”分支，即变量$1 的值为“-L”
                        //时执行该分支。第 9~12 行为 while 结构，每次循环从变量$2
                        //指定的文件读入一行数据，赋值给变量 LINE
10  do                 //while 循环的 do 语句
11    echo $LINE | tr '[A-Z]' '[a-z]' >> ${2}_low //使用 tr 命令将变量 LINE 中保存
的字符串转换为小
                        //写，并附加保存到文件${2}_low 中
12  done< $2           //表示 while 循环体结束，“< $2”表示 while 循环从$2 获得输入
的数据
13  ;;                //表法“-L”分支结束
14  -U)while read LINE //第 14~18 行为 case 结构的“-U”分支，即变量$1 的值为“-U”时
                        //执行该分支。第 14~17 行为 while 结构，每次循环从变量$2 指定
                        //的文件读入一行数据，赋值给变量 LINE
15  do                 //while 循环的 do 语句
16    echo $LINE | tr '[a-z]' '[A-Z]' >> ${2}_upper //使用 tr 命令将变量 LINE 中保存
的字符串转
                        //换为大写，并附加保存到文件
${2}_upper 中
17  done < $2          //表示 while 循环体结束，“< $2”表示 while 循环从$2 获得输入
的数据
18  ;;                //表法“-U”分支结束
19  *) echo "Error, Please enter -U or -L " //case 结构的“其他”分支，显示输出
"Error,
                        //Please enter -U or -L"
20  ;;                //分支结束
21 esac               //case 结构结束

```


该程序的执行过程如下所示。

```
# cat datafile //显示文件 datafile 的内容
aaaaaa bbbbbb
cccccc DDDDDD
eeeeee ffffff
#chmod u+x 20.24 //赋予文件 20.24 执行权限
# ./20.24 datafile //执行程序 20.24
Error:insufficient argument! -U,-L and filename
Error, Please enter -U or -L
# ./20.24 -L datafile //执行程序 20.24
# cat datafile_low //显示文件 datafile_low 的内容
aaaaaa bbbbbb
cccccc dddddd
eeeeee ffffff
# ./20.24 -U datafile //执行程序 20.24
# cat datafile_upper //显示文件 datafile_upper 的内容
AAAAAA BBBB
CCCCC DDDDD
EEEEEE FFFFFF
```

20.25 统计文件中的文本信息

wc 命令用来计算指定文件的字数、字节数以及行数，并可以按不同的形式显示统计结果。编写一个 Shell 程序，根据用户的需要对指定文件的文本信息进行统计。

20.25.1 系统设计

程序中使用了 case 分支结构，根据用户的输入执行不同的处理程序。分支如下所示。

- 用户输入参数“-l”：统计文件中文本的行数。
- 用户输入参数“-c”：统计文件中字符数目。
- 用户输入参数“-w”：统计文件中的字数。

20.25.2 代码实现

程序中使用了 wc 对文件的信息进行统计。为了直接提取到 wc 命令执行结果中的统计数字，使用 gawk 命令将包含数字的字段取出。具体代码如下所示（代码 20.25）。

```
1 #!/bin/bash //表示调用 bash 执行该程序
2 #filename:20.25 //注释，表示该脚本名称为 20.25
3 if [ $# -lt 2 ] //第 3~7 行为 if...then...fi 结构，用于判断位置参数是否小于 2，如果小于 2 则退出
4 then //if 结构中的 then 语句
5     echo "Error,Please enter -l ,-c or -w and filename" //显示输出“Error,Please
enter -l ,-c or
// -w and filename”
6     exit 1 //退出
7 fi //if 结构中的 fi 语句
8 case $1 in //第 8~20 行为 case 结构，对用户
//case 结构中的“-l”分支，其中
9 -l)LINE=' wc -l $2 | gawk '{print $1}''
```



“wc -l \$2”表示统计变量\$2

“|”输出给 gawk 命令，由 gawk 提取出第 1 个字段的内容，最后保存到

```
10 echo "文件 $2 中共 $LINE 行"
11 ;;
12 -c)Char=' wc -c $2 | gawk '{print $1}''
```

\$2”表示统计变量

管道“|”输出给 gawk 命令，由 gawk 提取出第 1 个字段的内容，最后保

```
13 echo "文件 $2 中共 $Char 个字符"
14 ;;
15 -w)WORD=' wc -w $2 | awk '{print $1}''
```

\$2”表示统计变量

道“|”输出给 gawk 命令，由 gawk 提取出第 1 个字段的内容，最后保存

```
16 echo "文件 $2 中共 $WORD 个单词"
17 ;;
18 *) echo "Please enter -l,-c or -w"
```

“Please enter -l,-c or -w”

```
19 ;;
20 esac
```

该程序的执行过程如下所示。

```
# chmod u+x 20.25 //赋予文件 20.25 执行权限
# ./20.25 kk //执行程序 20.25
Error,Please enter -l ,-c or -w and filename
# cat backup.cnf //显示文件 backup.cnf 的内容
#
# inittab This file describes how the INIT process should set up
# the system in a certain run-level.
#
# Author: Miquel van Smoorenburg, <miquels@drinkel.nl.mugnet.org>
# ./20.25 -w backup.cnf //执行程序 20.25
文件 backup.cnf 中共 27 个单词
# ./20.25 -l backup.cnf //执行程序 20.25
文件 backup.cnf 中共 5 行
# ./20.25 -c backup.cnf //执行程序 20.25
文件 backup.cnf 中共 198 个字符
```

//指定文件的行数，将统计结果通过管道

//变量 LINE 中

//显示输出“文件 \$2 中共 \$LINE 行”

//“-l”分支结束

//case 结构中的“-c”分支，其中“wc -c

//指定文件的字符数，将统计结果通过

//存到变量 Char 中

//显示输出“文件 \$2 中共 \$Char 个字符”

//“-c”分支结束

//case 结构中的“-w”分支，其中“wc -w

//指定文件的字数，将统计结果通过管

//到变量 WORD 中

//显示输出“文件 \$2 中共 \$WORD 个单词”

//“-w”分支结束

//case 结构的“其他”分支，显示输出

//“其他”分支结束

//case 结构结束

[G e n e r a l I n f o r m a t i o n]

书名= L i n u x命令、编辑器、S h e l l编程实例大全

作者= 杨明华, 谭励, 于重重编著

页码= 7 9 0

I S B N= 7 9 0

S S号= 1 2 2 1 7 4 3 0

d x N u m b e r =

出版时间= 2 0 0 9 . 0 4

出版社= 该引擎未能查询到

定价：8 9 . 0 0

试读地址= h t t p : / / b o o k . s z d n e t . o r g . c n / b o o k D e t a i l . j
s p ? d x N u m b e r = 0 0 0 0 0 6 6 8 0 7 2 3 & d = 6 9 9 4 A 2 7 5 2 4 4 8 5 E 7 E
3 2 C C A 5 4 3 3 B D 9 9 B 5 F & f e n l e i = 1 8 1 7 0 4 & s w = % B 1 % E 0 % B 3
% C C

全文地址= h t t p : / / i m g 8 . 5 r e a d . c o m / i m a g e / s s 2 j p g . d l l
? d i d = n 2 3 & p i d = 1 0 5 1 1 E 9 F 3 1 5 A 5 7 7 8 B 4 D 6 2 5 4 2 E 0 F 8 D 9
F C D 6 F 8 C D 3 6 C C 7 8 5 E 1 A E 0 D 9 C A E 4 1 F 2 8 C 2 B 4 9 D 7 C 4 2 2 3 F
4 B B 2 A 8 A 9 5 4 4 6 8 B E A 6 1 E 3 A 7 A 1 C 3 9 A 3 B C B 1 2 6 4 0 5 8 2 0 4 4
E 7 B C E 2 E 0 E 3 E 9 2 2 F 5 9 5 2 D 5 7 1 5 F E 6 3 1 F 7 A 2 8 6 0 A 8 6 0 8 9 7
C 6 4 3 8 7 6 C 1 8 A 7 6 7 8 A 3 0 0 B D 2 5 7 B 7 0 F 9 F C 0 0 9 C E 9 1 7 A 6 3 E
B 0 E 8 5 7 2 E 9 E 1 B E 1 3 2 B 0 F F C 2 5 7 3 F & j i d = /