国外经典教材·计算机科学与技术

PEARSON Prentice Hall

UNIX Shells by Example, 3rd Edition

UNIX Shell范例精解（第3版）

(美) Ellie Quigley 著

刘洪涛 译

清华大学出版社

国外经典教材·计算机科学与技术

UNIX Shell 范例精解

（第 3 版）

（美）Ellie Quigley 著

刘洪涛 译

清华大学出版社

北 京

内 容 简 介

本书全面介绍了 UNIX 和 Linux 上各种流行的 shell。本书在内容的组织上颇有特色，作者不是面面俱到，而是选择基本而实用的知识点进行讲解。对于每个知识点，先简明扼要地介绍，然后给出若干个精心设计的实例，对比着进行讲解。这种教学方式深入浅出、引人入胜，使学习过程变得轻松而充满乐趣。

本书是系统管理人员、程序设计人员的首选读物。

图书在版编目（CIP）数据

UNIX Shell 范例精解（第 3 版）/（美）奎[illegible]著；刘[illegible]译.—北京：清华大学出版社，2004.10
（国外经典教材·计算机科学与技术）
书名原文：UNIX Shells by Example, 3rd Edition
ISBN 7-302-09396-2
Ⅰ.U… Ⅱ.①奎…②刘… Ⅲ.UNIX 操作系统[illegible]设计—教材 Ⅳ.TP316.81

中国版本图书馆 CIP 数据核字（2004）第 089350 号

出 版 者：清华大学出版社　　**地　　址**：北京清华大学学研大厦
http://www.tup.com.cn　　**邮　　编**：100084
社 总 机：010-62770175　　**客户服务**：010-62776969
文稿编辑：徐刚
封面设计：久久度文化
印 刷 者：北京密云胶印厂
装 订 者：北京市密云县京文制本装订厂
发 行 者：新华书店总店北京发行所
开　　本：185×260　**印张**：45.25　**字数**：1096 千字
版　　次：2004 年 10 月第 1 版　2004 年 10 月第1 次印刷
书　　号：ISBN 7-302-09396-2/TP·6561
印　　数：1～4500
定　　价：68.00 元

出版说明

近年来，我国的高等教育特别是计算机学科教育，进行了一系列大的调整和改革，急需一批门类齐全、具有国际先进水平的计算机经典教材，以适应当前我国计算机科学的教学需要。通过使用国外先进的经典教材，可以了解并吸收国际先进的教学思想和教学方法，使我国的计算机科学教育能够跟上国际计算机教育发展的步伐，从而培育出更多具有国际水准的计算机专业人才，增强我国计算机产业的核心竞争力。为此，我们从国外知名的出版集团Pearson引进这套“国外经典教材·计算机科学与技术”教材。

作为全球最大的图书出版机构，Pearson在高等教育领域有着不凡的表现，其下属的Prentice Hall和Addison Wesley出版社是全球计算机高等教育的龙头出版机构。清华大学出版社与Pearson出版集团长期保持着紧密友好的合作关系，这次引进的“国外经典教材·计算机科学与技术”教材大部分出自Prentice Hall和Addison Wesley两家出版社。为了组织该套教材的出版，我们在国内聘请了一批知名的专家和教授，成立了一个专门的教材编审委员会。

教材编审委员会的运作从教材的选题阶段即开始启动，各位委员根据国内外高等院校计算机科学及相关专业的现有课程体系，并结合各个专业的培养方向，从Pearson出版的计算机系列教材中精心挑选针对性强的题材，以保证该套教材的优秀性和领先性，避免出现“低质重复引进”或“高质消化不良”的现象。

为了保证出版质量，我们为该套教材配备了一批经验丰富的编辑、排版、校对人员，制定了更加严格的出版流程。本套教材的译者，全部来自于对应专业的高校教师或拥有相关经验的IT专家。每本教材的责编在翻译伊始，就定期不间断地与该书的译者进行交流与反馈。为了尽可能地保留与发扬教材原著的精华，在经过翻译、排版和传统的三审三校之后，我们还请编审委员或相关的专家教授对文稿进行审读，以最大程度地弥补和修正在前面一系列加工过程中对教材造成的误差和瑕疵。

由于时间紧迫和受全体制作人员自身能力所限，该套教材在出版过程中很可能还存在一些遗憾，欢迎广大师生来电来信批评指正。同时，也欢迎读者朋友积极向我们推荐各类优秀的国外计算机教材，共同为我国高等院校计算机教育事业贡献力量。

清华大学出版社

2004.03.20

国外经典教材·计算机科学与技术

编审委员会

译 者 序

工欲善其事，必先利其器，对于 UNIX/Linux 系统管理员或开发人员而言，shell 就是提高工作效率的利器，而这本书则是如何使用这一利器的秘籍。作为一名专门从事 UNIX/Linux 系统软件开发的软件工程师，我对 shell 的功能和妙趣深有体会。shell 就像瑞士军刀，灵巧而实用。

翻译这本书同时也是一个学习的过程。翻译过程中，我很自然地跟随作者的思路，结合使用经验，系统地学习了一遍 shell。我认为这本书最突出的优点在于内容的组织。在内容的选择上，作者不是面面俱到，而是选择那些基本而实用的知识点进行讲解。对于每个知识点，作者先简明扼要地介绍，然后给出若干个精心设计的示例，对比着进行讲解。这种教学方式深入浅出、引人入胜，使学习 shell 变得轻松而充满乐趣。我认为这确实是一本值得推荐的好书。

为了提高译文的质量，我邀请同事李安欣、沈婕、顾巧云和尹战文参与了翻译。本书介绍了 5 种常用的 shell，我对 Bourne 和 Bash 比较熟悉，对另外 3 种稍为生疏。4 位同事的加入解决了这个问题。我们在翻译过程中始终保持认真和谨慎的态度。我们经常对原文进行讨论，一些复杂的例子都在机器上进行了实验。我们还发现了作者的一些小纰漏，在译文中做出了纠正。

译 者

2003 年 9 月

前　言

shell 游戏充满了乐趣。写这本书的目的就是让读者的学习过程变得有趣而又有收获。本书的第一版推出后，很多读者来信说：他们从我的书中得到帮助，认识到 shell 编程根本就不难！实例让 shell 编程容易而有趣。正是因为读者的肯定，Prentice Hall 才邀请我编写这个新的修订本。新版本中增加了 Bash 和 TC shell 这两个流行的 shell。提起 Bash 和 TC shell，人们总会联想到 Linux 系统，其实，UNIX 的用户也都能免费使用它们。事实上，现在很多 UNIX 用户更喜欢用这两种 shell，而不是那些传统的 UNIX shell，因为这两种 shell 提供了功能更强、使用更灵活的交互环境以及更完善的编程功能。

本书是我 19 年教学生涯的顶点。这些年来，我针对各种 shell 和程序员常用的工具设计了多门课程。我为这些课程编写的讲义被用于加州大学圣克鲁兹分校和戴维斯分校的 UNIX 教学、Sun 公司的培训，还被 Apple 公司、DeAnZa 大学以及全球众多厂商采用。根据客户的需求，我通常每次只讲授一种 shell，而不是一次讲授全部 shell。为了满足众多客户的需要，我还为每种 shell 和工具单独编写了培训教材。

无论我讲授的课程是“Grep，Sed 和 Awk”、“系统管理员 Bourne shell 教程”还是“交互式的 Korn shell”，总有学员会问：“有没有一本书能涵盖所有的 shell，以及 grep、sed 和 awk 这些重要的实用程序？我该买一本关于 awk 的书好呢，还是买一本关于 grep 和 sed 的书好？有没有一本真正覆盖所有内容的书？我可不想为了做一个 shell 程序员而买上三、四本书”。

遇到这类问题时，我可以向学生们推荐一大堆好书，但这些书只是单独讲述某个主题。也有一些 UNIX 参考书试图囊括所有的内容，但都只是蜻蜓点水的介绍，学员们需要的却是详细讲解。学员们希望有一本书能包含他们需要的全部内容：UNIX 工具、正则表达式、引用规则、主要的几种 shell、不同 shell 的对比、练习题等等。这本就是他们想要的书。写此书时，我仔细回想了自己是如何讲授课程，然后按同样的结构来组织书中各章。shell 编程课的第一个主题始终是介绍 shell 是什么，它如何工作；接下来则是讲述 grep、sed 和 awk 等 UNIX 实用程序，这些是 shell 程序员工具箱中最重要的工具。我将分两个步骤讲授 shell：首先将它作为一个交互式程序，所有的工作都可以在命令行上完成；然后再把它作为编程语言，在脚本中讲解并示范其编程结构。（如果被视为编程语言，C shell 和 TC shell 几乎完全相同，因此，我用两章分别介绍它们的交互使用，但只在其中一章讨论编程结构。）无论课程长度是两天、一周还是一个学期，shell 编程课程结束后，学员们都能精通并且喜欢编写脚本。因为他们学会了如何玩 shell 游戏。不管你参加培训或自学，这本书都能教会你如何玩 shell 游戏。

我发现简单的例子便于快速理解，所以，讲述每个概念时，我都先给出一个小例子和它的输出结果，然后对例子中每一行进行解释。这个方法已经被证明是行之有效的：我的第一本书《Perl by Example》的读者很喜欢这种讲解方式；本书也同样受到了需要编写、阅读和维护 shell 程序的人们的欢迎。

本书对五种 shell 的讲述是平行铺开的。举个例子就能说明这样安排的好处：如果你想知道如何在某个 shell 执行重定向，其他每一种 shell 的对应章节都有同一主题的讨论。本书的附录 B 中给出了五种 shell 的快速对比表。

当你只是想获得足够的信息来唤起你对某个命令如何工作的记忆，却要去查阅另一本书或 UNIX 参考手册，你肯定会觉得麻烦。为了节约读者的时间，附录 A 列出了一些有用的命令，包括命令的语法和定义。对于其中功能更为强大和更常用的命令还给出了例子和注释。

附录 B 中的对比表能够帮助你分清不同的 shell，尤其是当你将脚本从某种 shell 移植到另一种时。你也可以把它作为一个语法结构的提示表，用于快速语法查询。

shell 程序遇到的一个最大障碍就是不能正确使用引用。附录 C 中的引用规则给出了一组详细的步骤，教你如何在复杂的命令行上正确执行引用。这组步骤能显著减少 shell 程序员调试脚本时无法正确匹配引号而浪费的时间。

读者将发现这是一本很有用的辅导书和参考手册。本书的目的就是通过例子来讲解 shell，让它变得简单，这样就能节约读者的时间，并且让他们体会到学习的乐趣。这本书包含我在课堂上所讲的全部内容，因此我确信读者将在很短的时间成为高效的 shell 程序员。你所需的一切都在你手中的这本书内。shell 游戏充满乐趣，你很快就会感受到！

作　者

目　录

第 1 章　UNIX shell 简介

1.1　定义与功能

shell 是一种特殊的程序。如图 1.1 所示，它是用户与 UNIX 系统的“心脏”(一个称为内核的程序)之间的接口。内核(kernel)在系统引导时被载入内存，管理系统直至关机。它创建和控制进程、管理内存、文件系统以及通信等。内核以外的所有程序(包括 shell 程序)都保存在磁盘上。内核将这些程序载入内存运行，在它们终止后清理系统。shell 是一个工具程序，登录后系统启动。它解释运行由命令行或脚本文件输入的命令，从而实现用户与内核间的交互。

当登录成功后，系统会启动一个交互式的 shell，提示进行输入。键入命令后，shell 开始执行任务：(a)解析命令行；(b)处理通配符、重定向、管道和作业控制；(c)查找命令，找到后开始执行。UNIX 的初学者大都通过交互方式使用 shell，即在命令提示符后逐条输入、执行命令。

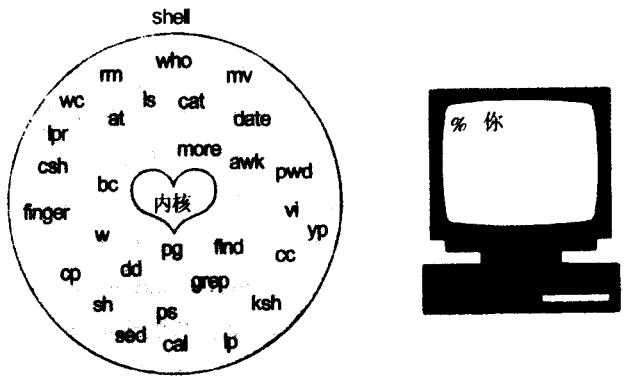

图 1.1　Kernel、shell 和你

如果总要键入一组大致相同的命令，你自然会希望将这些工作自动化。把命令写到一个文件里，然后执行这个文件就可以了。保存命令的文件称作脚本文件。shell 脚本跟批处理文件很相似，都是把一组 UNIX 命令输入文件，然后执行该文件。更复杂的脚本还包括用

于实现判断、循环、文件测试等功能的程序结构。编写脚本不仅要掌握编程结构和编程技巧，还需要对 UNIX 工具集及其运行机制有较深的理解。有一些工具，如 grep、sed 和 awk，在处理命令输出和文件时，功能很强大。熟悉了这些工具和所用 shell 的程序结构后，你就可以编写有用的脚本了。当从脚本中执行命令时，shell 被视作一种编程语言。

1.1.1 三种主要的 UNIX shell

UNIX 系统大都支持三种主流的 shell，它们是 Bourne shell(也叫 AT&T shell)、C shell(也叫 Berkeley shell)和 Korn shell(Bourne shell 的一个扩展集)。交互方式运行时，这三种 shell 非常相似；但作为脚本语言，它们在语法和效率上有一定差别。

Bourne shell 是标准的 UNIX shell，用于系统管理。大部分系统管理脚本，如 rc start 和 stop 脚本、shutdown，都是 Bourne shell 脚本。单用户模式下，管理员以 root 身份运行的通常是 Bourne shell。Bourne shell 是 AT&T 开发的，以简练、紧凑和快速著称。缺省的 Bourne shell 命令提示符是美元符号。

C shell 由美国加州大学 Berkeley 分校开发，增加了很多新的功能，比如命令行历史、别名、嵌入算术运算、文件名自动补全和作业控制。交互式用户更喜欢 C shell，但管理员们还是更喜欢用 Bourne shell 编写脚本。因为同样的脚本，用 Bourne shell 写更简单，运行更快。缺省的 C shell 命令提示符是百分号。

Korn shell 的作者是 AT&T 的 David Korn，Korn shell 是 Bourne shell 的一个扩展集。和 C shell 相比，Korn shell 在 Bourne shell 基础上进行了更多改进，增加了很多新的功能。Korn shell 的功能特点包括：可编辑的命令历史、别名、函数、正则表达式通配符、嵌入算术运算、作业控制、协同处理以及特殊的调试功能。Korn shell 几乎完全向上兼容 Bourne shell，所以老的 Bourne shell 程序在 Korn shell 中运行良好。缺省的 Korn shell 命令提示符是美元符号。

1.1.2 Linux 的 shelll

Bash 和 TC 这两个 shell 经常被称作“Linux” shell，它们是免费软件，并且可以在任何 UNIX 系统上编译；实际上，Solaris 8 和 Sun 的 UNIX 操作系统上现在已经捆绑了这两种 shell。安装 Linux 系统后，使用的将是 GNU 的 shell 和工具，而非标准的 UNIX 的 shell 和工具。尽管支持多种 shell，Linux 上最流行的还是 Bourne Again shell(bash)和 TC shell。Z shell 也是一个 Linux 的 shell，它综合了 Bourne Again shell、TC shell 和 Korn shell 的大量功能。此外，Linux 上还有克隆自 Korn shell 的 Public Domain Korn shell (pdksh)，需要向 AT&T 付费的 Korn shell，更不用说那些大量不知名的更小的 shell。

要知道你用的 Linux 有哪些 shell，可以查看/etc/shell 目录下的文件。

想改用/etc/shell 列出的某种 shell，只要键入 chsh 命令和 shell 名。例如，想以后一直用 TC shell，那就在命令提示符后键入：chsh /bin/tcsh。

1.1.3 shell 的历史

第一个重要的标准 UNIX 的 shell 于 1979 年底在 V7(AT&T 的第 7 版)UNIX 上推出，

以作者 Stephen Bourne 命名。作为编程语言，Bourne shell 基于另一种叫 Algol 的语言。Bourne shell 当时主要用于系统管理任务的自动化，Bourne shell 以简单和高速而深受欢迎，却缺少了很多用于交互的功能，如命令历史、别名和作业控制。现在出现了 bash，即 Bourne Again Shell。自由软件基金会的 Brian Fox 取得 GNU 版权许可后开发出 bash。bash 是 Linux 操作系统上缺省的 shell。它的设计符合了 IEEE POSIX P1003.2/ISO 9945.2 Shell 和工具标准。在交互和编程两方面，bash 都提供了很多 Bourne shell 没有的功能(但 Bourne shell 脚本无需修改还能在 bash 下运行)。bash 还结合了 C shell 和 Korn shell 的最有用的功能，它很优秀。bash 对 Bourne shell 的改进包括：命令行历史记忆与编辑、目录栈、作业控制、函数、别名、数组、整数运算(底数可以是 2 到 64)以及 Korn shell 的一些功能，如扩展的元字符、用于生成菜单的 select 循环和 let 命令等。

C shell 由加州大学 Berkeley 分校于上世纪七十年代末开发，作为 2BSD UNIX 系统的一部分发布。它的主要作者是 Bill Joy。C shell 提供了很多标准的 Bourne shell 不具备的功能。C shell 基于 C 语言，作为编程语言使用时，语法也类似于 C。C shell 增强了交互功能，如命令行历史、别名和作业控制。由于专为大型机设计并增加了很多新功能，C shell 在小型机上运行可能比较慢，即使在大型机上，它的速度也不如 Bourne shell。

TC shell 是 C shell 的扩展版本。新增的功能包括：命令行编辑(emacs 和 vi)、命令历史清单的滚动、高级的文件名功能、变量和命令补全、拼写纠错、作业调度、账户自动上锁和注销以及历史清单中增加时间戳等。

Bourne shell 和 C shell 的共存，使 UNIX 用户有了选择余地，也导致了关于哪个 shell 更好的争论。二十世纪中期，AT&T 的 David Korn 推出了 Korn shell。Korn shell 于 1986 年发布，并在 1988 年 UNIX 的 SVR4 版本发布时正式成为其一部分。Korn shell 其实是 Bourne shell 的一个扩展集，它不仅能运行于 UNIX 系统，还能在 OS/2、VMS 和 DOS 上运行。Korn Shell 提供了对 Bourne shell 的向上兼容，加入了许多 C shell 中受欢迎的功能，而且快速和高效。Korn shell 历经许多版本，虽然 1993 版正逐渐流行，目前用得最广的还是 1988 版。Linux 用户可能会发现自己正在使用 Korn shell 的免费版本，称为 Public Domain Korn Shell，简称 pdksh。pdksh 是 1988 版 Korn shell 的克隆。pdksh 是免费和可移植的。对它的改进正在进行中，以使其能够完全兼容 Korn shell，且符合 POSIX 标准。此外还有 Z shell(zsh)，这也是一个 Korn shell 的克隆，集成了 TC shell 的一些功能。Z shell 的作者是 Paul Falsted，可以从很多网站免费得到 Z shell。

1.1.4　shell 的作用

shell 的一项主要功能是：解释交互方式下从命令行输入的命令。shell 解析命令行，将其分解为词(也称为 token)；词之间由空白分隔，空白由制表符，空格键或换行组成。如果词中有特殊的元字符，shell 会对其进行替换。shell 处理文件 I/O 和后台运行。对命令行的处理结束后，shell 搜索命令并执行。

shell 的另一项重要功能是定制用户的环境，这通常在 shell 的初始化文件中完成。初始化文件中有很多定义，包括设置终端按键和窗口属性；用来定义搜索路径、权限、提示符和终端类型的变量；设置特定应用程序所需的变量，如窗口、字处理程序和编程语言的库等。

Korn shell 和 C shell 提供更多的定制功能：引入命令历史、别名、设置内置变量防止用户破坏文件或无意中退出，通知用户作业完成。

shell 还能被当成解释性的编程语言。shell 程序（也叫 shell 脚本）由文件中的一列命令组成。shell 程序用编辑器生成（也可以在命令行上直接输入脚本）。它们由 UNIX 命令组成，命令之间插入了一些程序结构，如变量赋值、条件测试和循环。shell 脚本不需要编译。shell 会逐行解释脚本，就好像它是从键盘输入一样。shell 负责解释命令，如果用户需要了解有哪些命令，可参考附录 A，其中列出了一些有用的命令。

1.1.5 shell 的任务

shell 负责确保用户在命令提示符后键入的命令被正确执行。这里的"责任"包含以下内容：

1. 读输入并解析命令行。
2. 替换特殊字符。
3. 设置管道，重定向和后台运行。
4. 处理信号。
5. 程序执行的相关设置。

对于上面的每个问题，在与具体的 shell 相关时，都会详细讨论。

1.2 系统启动与登录 shell

系统启动时运行的第一个进程是 init。每个进程都有一个称为 PID 的进程标识号。init 是第一个进程，所以它的 PID 是 1。init 进程初始化系统，并启动另一进程来打开终端线路并设置标准输入（stdin）、标准输出（stdout）和标准错误输出（stderr），三者都与终端关联。标准输入通常来自键盘；标准输出和标准错误输出则显示在屏幕上。做完这些设置，终端上就会出现登录提示。

系统会在键入用户名后提示输入口令。程序/bin/login 通过检查 passwd 文件的第一列来确认用户的身份。如果用户名在其中，它会运行一个加密程序来加密键入的口令以验证其正确性。口令验证通过后，login 程序设置初始环境。初始环境是一组定义工作环境的变量，这组变量将传给 shell。变量 HOME、SHELL、USER 和 LOGNAME 根据 passwd 文件中的信息被赋值。HOME 被设为主目录；SHELL 则被设为登录 shell 的名字，即 passwd 文件中的最后一列。USER 和 LOGNAME 被赋值为登录名。login 还设置了变量 search path，常用的工具程序可以在该变量指定的目录中找到。login 程序结束时执行它在 passwd 文件的最后一列中找到的程序。这个程序通常是一个 shell。如果 passwd 文件的最后一列是/bin/csh，执行的就是 C shell；如果是/bin/sh 或为空，执行 Bourne shell；如果是/bin/ksh，则执行 Korn shell。被执行的 shell 称为登录 shell。

shell 启动后，先查找由系统管理员设置的系统级的初始化文件，然后在主目录中查找有没有登录 shell 对应的初始化文件。找到的文件都被执行。初始化文件用于对用户环境的进一步定制。初始化文件中的命令执行后，屏幕上会出现一个提示符。现在，shell 等着

用户的输入。

1.2.1　解析命令行

命令提示符后键入一条命令后，shell 读入这个命令行并解析，将其分解为词，词称为 token。token 之间用空格或制表符分隔，命令行由换行符终结。将行截成 token 的过程叫词法分析。接下来，shell 检查第一个词是否为内置命令或磁盘上的可执行程序。如果是内置命令，shell 就在内部执行它。否则，shell 将在路径变量所指定的目录中查找这个程序。如果找到了命令的程序，shell 就创建一个进程来执行。之后，shell 进入睡眠（或等待）状态，直至程序执行完毕。shell 会根据需要，报告程序的退出状态。此时，屏幕上又出现命令提示符，整个过程从头开始。命令行的处理顺序如下：

1. 执行历史命令替代（视情况）。
2. 命令行被分解为 token（或称"词"）。
3. 更新历史命令（视情况）。
4. 引用的处理。
5. 别名替代和函数定义（视情况）。
6. 设置重定向，后台任务和管道。
7. 执行变量替换（如 $ name，$ user 等）。
8. 执行命令替换（如 date）。
9. 执行称为 globbing 的文件名替换（如 cat abc. ??，rm * . c 等）。
10. 执行程序。

1.2.2　命令类型

命令被执行时，可以是别名、函数、内置命令或磁盘上的可执行程序。别名是原有命令的缩写（绰号），可用于 C、TC、Bash 和 Korn shell。函数可用于 Bourne shell（在 AT&T System V 的第 2 版引入），Bash 和 Korn shell。函数就是一组命令，并像独立程序那样组织起来。别名和函数都在 shell 的内存空间中定义。内置命令是 shell 的内部程序，而可执行程序则在磁盘上。shell 用路径变量在磁盘上定位可执行程序。执行命令前，shell 需要创建一子进程。程序定位和子进程创建都要花一定的时间。执行命令前，shell 按如下顺序判定程序类型（其中，3 和 4 是为 Bourne 和 Korn shell 保留的，3 不能用于 C shell 和 TC shell）：

1. 别名。
2. 关键字。
3. 函数（bash）。
4. 内置命令。
5. 可执行程序。

举个例子，如果命令是 xyz，shell 先检查 xyz 是不是一个别名。如果不是，那它是不是内置命令或函数？如果都不是，那它肯定是磁盘上的一个可执行程序。接下来 shell 就要开始查找这个命令的路径。

1.3 进程与 shell

进程是正在运行的程序，可以用它惟一的 PID 号（进程标识符）来标识。内核负责控制和管理进程。进程由可执行程序、数据、堆栈、程序指针、堆栈指针、寄存器以及程序运行时需要的所有信息组成。shell 被启动后，就成为一个进程，并且属于某个进程组。进程组使用组 PID 进行标识。任何时候，只能有一个进程组拥有终端的控制权，即所谓的“前台运行”。登录系统后，用户的 shell 便控制了终端，等待命令提示符后输入命令。

shell 可以派生其他进程。实际上，当用户从命令提示符或 shell 脚本输入命令后，shell 就要负责在自己的内部代码中（内置）和磁盘上查找该命令，并且安排其运行。这是通过对内核的调用（即系统调用）完成的。系统调用是对内核服务的请求，它是进程访问系统硬件的惟一途径。UNIX 提供了很多系统调用，用于创建、执行和终止进程（shell 执行重定向和管道、命令替换，执行用户命令时，提供的是来自内核的其他服务）。

后续各节将讨论 shell 运行新进程使用到的那些系统调用。见图 1.2。

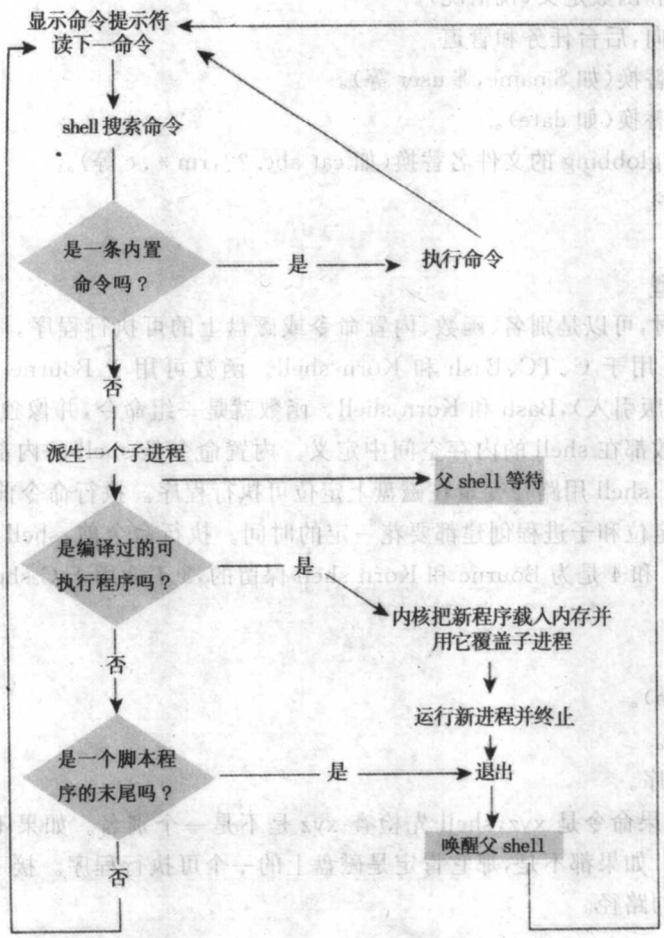

图 1.2 shell 与命令执行

1.3.1 哪些进程正在运行

ps 命令。ps 命令和它的众多选项能够以多种格式列出当前正在运行的进程。例 1.1 显示的是一个 Linux 系统上所有由用户运行的进程(关于 ps 和选项,参见附录 A)。

例 1.1

```
$ ps au  (BSD/Linux ps)  (use ps -ef for SVR4)
USER   PID %CPU %MEM  SIZE   RSS TTY STAT START  TIME COMMAND
ellie  456  0.0  1.3  1268   840   1 S    13:23  0:00 -bash
ellie  476  0.0  1.0  1200   648   1 S    13:23  0:00 sh /usr/X11R6/bin/sta
ellie  478  0.0  1.0  2028   676   1 S    13:23  0:00 xinit /home/ellie/.xi
ellie  480  0.0  1.6  1852  1068   1 S    13:23  0:00 fvwm2
ellie  483  0.0  1.3  1660   856   1 S    13:23  0:00 /usr/X11R6/lib/X11/fv
ellie  484  0.0  1.3  1696   868   1 S    13:23  0:00 /usr/X11R6/lib/X11/fv
ellie  487  0.0  2.0  2348  1304   1 S    13:23  0:00 xclock -bg #c0c0c0 -p
ellie  488  0.0  1.1  1620   724   1 S    13:23  0:00 /usr/X11R6/lib/X11/fv
ellie  489  0.0  2.0  2364  1344   1 S    13:23  0:00 xload -nolabel -bg gr
ellie  495  0.0  1.3  1272   848  p0 S    13:24  0:00 -bash
ellie  797  0.0  0.7   852   484  p0 R    14:03  0:00 ps au
root   457  0.0  0.4   724   296   2 S    13:23  0:00 /sbin/mingetty tty2
root   458  0.0  0.4   724   296   3 S    13:23  0:00 /sbin/mingetty tty3
root   459  0.0  0.4   724   296   4 S    13:23  0:00 /sbin/mingetty tty4
root   460  0.0  0.4   724   296   5 S    13:23  0:00 /sbin/mingetty tty5
root   461  0.0  0.4   724   296   6 S    13:23  0:00 /sbin/mingetty tty6
root   479  0.0  4.5 12092  2896   1 S    13:23  0:01 X :0
root   494  0.0  2.5  2768  1632   1 S    13:24  0:00 nxterm -ls -sb -fn
```

1.3.2 创建进程

系统调用 fork。在 UNIX 系统中,进程是通过系统调用 fork 创建的。fork 生成调用进程的一个副本。新生成的这个进程称为子进程,创建它的进程称为父进程。fork 调用完成后,子进程立即开始运行,最初阶段,父子进程共享 CPU。子进程还得到父进程的部分进程信息的副本,包括环境、打开的文件、真实和有效用户标识、掩码、当前工作目录和信号。

输入命令后,shell 开始解析命令行,判断第一个单词是内置命令还是磁盘上的可执行命令。如果是内置命令,就由 shell 来处理;如果是磁盘上的命令,shell 就调用 fork 来生成自己的一个拷贝(图 1.3)。子进程将搜索路径以找到这个命令,并设置用于重定向的文件描述符、管道、命令替换和后台处理。子 shell 运行时,父 shell 通常处于睡眠状态(参见后面介绍的 wait)。

系统调用 wait。当子进程处理重定向、管道和后台处理等细节时,程序安排父 shell 进入睡眠状态。系统调用 wait,导致父进程挂起,直至它的一个子进程终止。如果 wait 调用成功,它返回死去子进程的 PID 和退出状态。如果父进程没有等待,子进程死亡时就进入僵尸(假死)状态,并且保持这个状态,直至父进程调用 wait 或死亡。欲移除僵尸进程,系统必须重启。如果父进程先于子进程死亡,init 进程会收养孤儿僵尸进程。因此,系统调用 wait 不仅仅用于让父进程进入睡眠状态,还可用于保证进程正常终止。