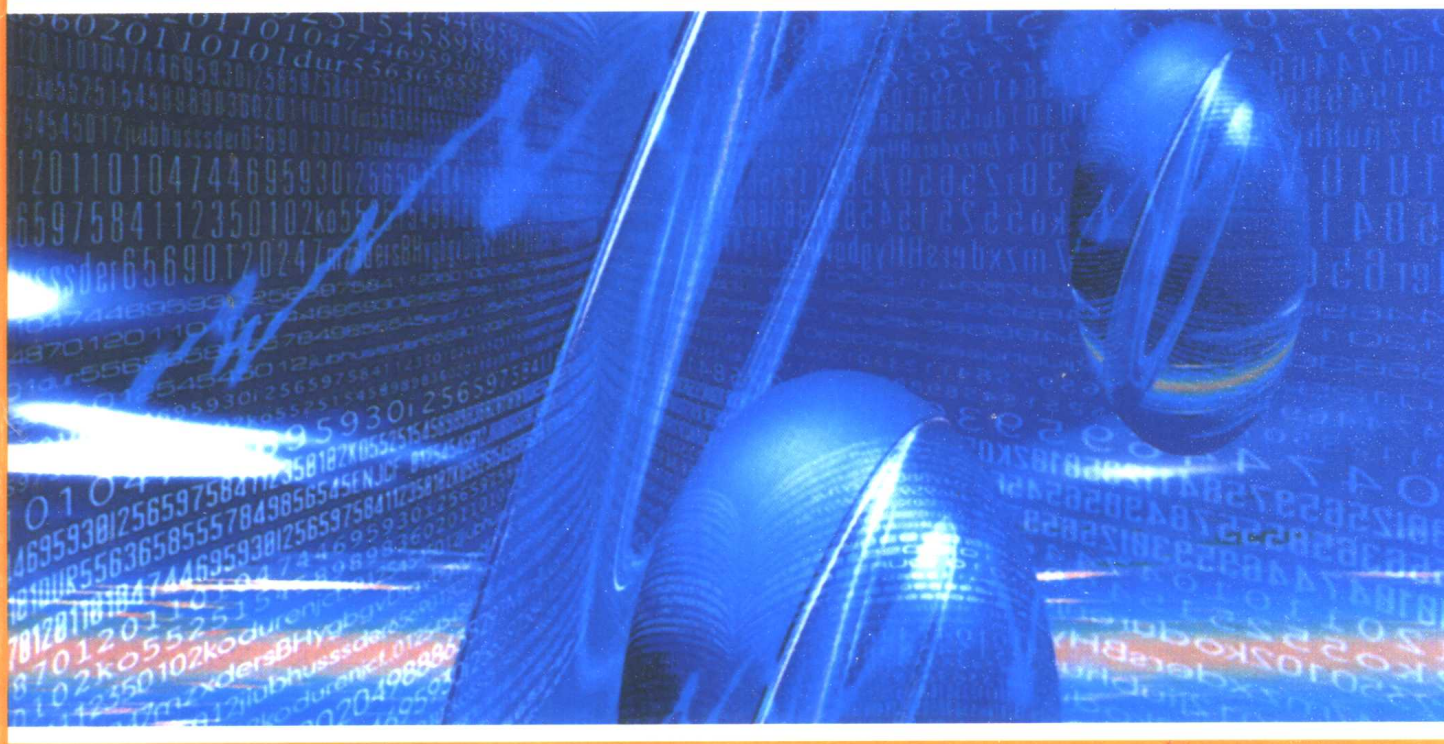
开源软件丛书

Linux Shell 实例精解



[美] Ellie Quigley 著
吴雨浓 译

PH PTR

中国电力出版
www.infopower.com

开源软件丛书

Linux Shell 实例精解

[美] Ellie Quigley 著
吴雨浓 译

中国电力出版社

内 容 提 要

本书重点讲述了当下最流行的两个 Linux Shell——Borne Again Shell（bash）和 TC Shell（tcsh），从它们的新特性、增加和内建的插件讲起直至具体的应用。通过对一个个实例的讲解，循序渐进地介绍了 Linux Shell 中的各个知识点，使读者逐步具备读写 Shell 程序的能力。

本书适合初、中级专业技术人员，以及对本技术感兴趣的读者阅读。

图书在版编目（CIP）数据

Linux Shell 实例精解/（美）埃莉著；吴雨浓译. —北京：中国电力出版社，2002.11

ISBN 7-5083-1306-2

Ⅰ.L… Ⅱ.①埃…②吴… Ⅲ.Linux 操作系统 Ⅳ. TP316.89

中国版本图书馆 CIP 数据核字（2002）第 084650 号

著作权合同登记号 图字：01-2001-2231 号

中国电力出版社出版、发行

（北京三里河路 6 号 100044 http://www.infopower.com.cn）

汇鑫印务有限公司印刷

各地新华书店经售

*

2003 年 2 月第一版 2003 年 2 月北京第一次印刷

787 毫米×1092 毫米 16 开本 34 印张 766 千字

定价 **59.00** 元

前　言

在 UNIX 下进行 Shell 编程是一件非常有趣的事情，现在在 Linux 下更是这样。在出版了《UNIX Shell by Example》以后，Mark Toud 建议我写一本关于在 Linux 下进行 Shell 编程的书。开始我们认为这很简单，至少我是这样认为，因为毕竟 Bourne 和 Bourne Again Shell 以及 C 和 TC Shell 之间没有太大的不同。你也许认为它只是添加了一些简单的新特性，但事实上并非如此。这就像是一本崭新的书跟草稿不同一样。

虽然 UNIX、Gnu 工具，以及 Shell 都提供了很多扩展和新特性，但 Linux 提供的不仅是 Gnu 的工具，还有很多功能完善的 Shell。我已经在《UNIX Shell by Example》中详细地讲述过 Korn Shell 了，所以在本书中将把注意力集中在最流行的两个 Linux Shell 上——Borne Again Shell（bash）和 TC Shell（tcsh）。

由于新特性、增加和内建的插件等等因素，讲解 Shell 的章节将不得不被剖开。在以前需要两章讲解的问题，现在则需要 4 章。那是一个漫长枯燥的过程，当我差不多完成 bash 这一章的时候，却发现自己没有使用最新的版本，于是又被迫重写了稿子。因为读者未必使用相同版本的 bash，所以本书的内容将覆盖新的版本和老的版本。

书中介绍了成功编写 Shell 程序所必不可少的 Gnu 工具——gawk、grep 和 sed。它们是模板、操作、文本编辑以及从文件和管道中提取数据的理想工具。

在学习过程中，Shell 首先是一个交互程序，所有的事情都是由命令行实现，其次作为一种程序语言，程序结构在 Shell 脚本中得到描述和示范。

事实证明，简单的例子更利于理解。随着每一行程序的解释和输出，在小例子当中理解每一个概念。学习过我的第一本书《Perl By Example》和《UNIX Shell By Example》的人会发现，这种方法被广泛应用。本书就是要使你在完全了解 Shell 前就开始逐渐具备读写和理解 Shell 程序的能力。

Shell 彼此是类似的。例如你想了解在某一种 Shell 中重新定向是如何实现的，完全可以参考其他 Shell 的相关部分的讨论，在本书的附录 B 中有快速比较图表供参考。

在编写程序时，若为了某个特定命令的用法而不得不去查找另外一本书或者 Linux 的 MAN 页，确实是一件令人厌烦的事情。为了节约时间，附录 A 中包含了一个命令清单，其中包括了所有命令的语法和定义，对于常用的和特别强大的命令还附有例子和相应解释。

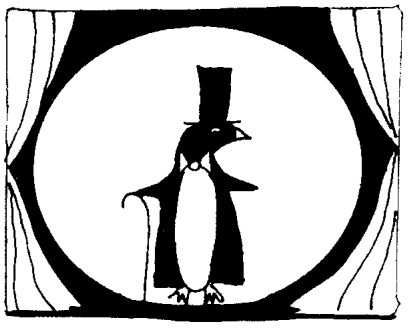
附录 B 中的比较图表有助于更直观地了解不同 Shell 间的差别，特别是在你把一种 Shell 脚本向另一种 Shell 移植或者快速查找语法的时候，它们会提示给你不同的结构是如何工作的。这些图表比较了 Korn、Borne、bash、tcsh 和 C Shell。

Shell 编程中最大的障碍是如何正确地使用“引用”。附录 C 中的“引用规则”部分详细而循序渐进地讲解了如何成功地在一些非常复杂的命令行中实现“引用”。这些内容可以帮助程序员在做琐碎的“引用”匹配尝试时，更加快捷。

我想你最终会发现这是一本非常有价值的自学指导和参考书。本书的目的就是通过例子对相关知识进行讲解，用尽量简单的方式使你保持学习的兴趣并节省时间。我确信你可以在很短的时间内成为一个具备很强编程能力的 Shell 程序员。你需要的所有内容都在这里，你会发现，Linux Shell 是非常有趣的事情！

Ellie Quigley
www.ellieq.com

目　　录



第 1 章

Linux Shell 介绍

1.1 为什么要使用 Linux

1991 年，刚刚大学毕业的 Linux Torvalds 在芬兰的赫尔辛基大学开发了一种类 UNIX 的操作系统内核，这种操作系统被设计为在 PC 上运行的 UNIX 系统，从一开始作为个人的兴趣爱好到现在发展为被安装到全世界大约 1 千万计算机上的功能完善的 32 位操作系统，其使用者的数量还在显著增长。首先，Linux 向黑客提供了一种可以自由下载的，并在内核层面上同样可以自由修改和测试代码的操作系统，它被 20 世纪 80 年代早期的 UNIX 黑客狂热推崇。现在，跟 UNIX 相比，Linux 不仅为大学黑客和“geek”们所喜爱的，在更广泛的范围内被个人和专业级用户所使用，通常为拥有大量计算机的网络系统提供服务。在很多情况下，Linux 都作为 Windows 的替代选择，在 PC 世界中得到了参与反对微软霸权的新革命团体、会议和出版物的支持。

在许多程序员和开发人员的帮助下，Linux 迅速成长为今天类 UNIX 的与 POSIX 兼容的 32 位的功能完善的操作系统。1992 年，自由软件基金会把 Gnu 软件加入到 Linux 内核中，使它成为一个意义上完整的操作系统，并把 Linux 内核源代码置于 GPL（General Public License）许可证之下。自由软件基金会提供上百个 Gnu 实用软件，它们包括 UNIX 下的标准 Bourne Shell 的加强。Linux 下的默认 Shell——Bourne Again Shell 是 Bourne Shell 的加强版，不仅在编程方式方面，而且在交互方式方面，它都允许用户通过裁减他们的工作环境和建立快捷方式来提高其工作效率。其他的 Gnu 工具，如 grep、sed 和 gawk 都跟 UNIX 下的同名工具功能类似，但是也都被加强并设计为与 POSIX[1] 兼容的了。内核和 Gnu 工具的结合，以及 Linux 可以在 PC 上运行这样一个事实，使 Linux 成为可以替代以前的 UNIX 和微软操作系统的一个良好选择。何况 Linux 对任何想得到它的人来说，包括源代码、办公套件和软件包在内都是自由的，无论你从互连网上下载还是购买发行套件的 CD，Linux 都是可移植的、稳定的和安全的。它可以使你的 PC 强大得像一个工作站。

1.1.1 POSIX 是什么

为了给不同的程序和操作系统提供相同的软件标准，IEEE 和 ISO（国际标准化组织）

1. Shell 功能的标准定义是 Posix1003.2。

共同提出了 POSIX 标准[可查阅开放系统标准（Open Systems Standards）]。其作用就是为应用软件在不同的平台之间移植提供与 UNIX 类似的标准。也就是在一台机器上写的软件在另外一台硬件配置不同的计算机上同样可以编译和运行。例如，在 FreeBSD 上写的软件可以运行在 Solaris、Linux 和 HPUX 机器上。1988 年第一个标准出炉了，被称为 POSIX1003.1。它的目的是提供 C 语言标准库。1992 年，POSIX 组为 Shell 和实用程序建立了标准，为开发可移植的 Shell 脚本提供了一组定义，被称为 IEEE1003.2 POSIX Shell 标准。虽然没有强制要求，但是多数 UNIX 系统买主都遵守 POSIX 标准。在讨论 Shell 和它们标准的 UNIX 实用程序的时候，遵守 POSIX 也就意味着在写新的实用程序或者强化旧的实用程序的时候，尽量遵守由 POSIX 委员会提出的标准。例如，Bourne Again Shell 就是几乎 100%兼容的 Shell、gawk 是一个可以在严格的 POSIX 模式下运行的用户实用程序。

1.2 Shell 的定义和功能

Shell 是提供操作系统核心（称为 kernel）与用户之间交互的特殊程序，参见图 1.1。这个 kernel 在启动时被装入内存，并管理系统直到关机为止。它负责建立和控制进程，管理内存、文件系统、通信等等。其他的实用程序，包括 Shell 在内都存储在硬盘上。kernel 把程序从硬盘中装入内存，运行它们，并在程序运行结束后回收被程序占用的系统资源。Shell 是从你登录就开始运行的实用程序，它允许用户通过 Shell 脚本或者命令行的方式输入命令，并通过翻译这些命令完成用户与 kernel 的交互。

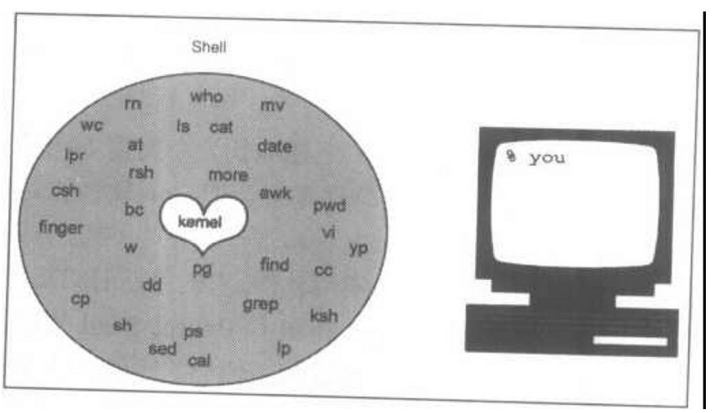


图 1.1 kernel、Shell 以及用户的关系示意图

当你登录到计算机上的时候，一个交互 Shell 会被启动并提示你输入。当你输入一条命令以后，Shell 会以如下的方式响应：①解析命令行；②处理通配符、重新定向、管道和作业控制；③搜索外部命令，如果找到就执行它。若第一次接触 Linux，你将不得不花费大量的时间用于学习如何从提示符执行命令，因而你将非常喜欢使用这种交互式的 Shell。

如果经常需要输入相同的一组命令，你就会希望系统能够自动执行这些任务。为了实现这个目的，将所有的命令放在一个可执行文件中，该文件称为 Shell 脚本。Shell 脚本跟批处理文件相似，复杂的 Shell 脚本包括判断、循环、文件操作等等程序结构。编写脚本不仅需要学习程序结构和编写技巧，还需要很好地理解 Linux 实用程序以及它们是如何工作的。有一些实用程序，例如 grep、sed 和 gawk，被用于在脚本中控制文件命令输出和文件

的时候极为强大。当熟悉这些工具和一些特定的 Shell 程序结构以后，你就可以开始编写有用的脚本了。当在脚本内执行命令时，Shell 被当作一种编程语言使用。

1.2.1 三种主要的 UNIX Shell

大多数 UNIX 系统都支持的优秀的 Shell 有 Bourne Shell（AT&T Shell）、C Shell（Berkeley Shell）和 Korn Shell（超级 Bourne Shell）。这三种 Shell 在交互方式下的行为极为相似，只是在作为脚本语言的时候在语法和执行效率上有差别。

Bourne Shell 是 UNIX 下的标准 Shell，被用于系统管理。大多数系统管理脚本，例如 rc start 和 stop 脚本，还有 shutdown 脚本都是 Borne Shell 脚本。在单用户模式下，它通常被管理员在 root 权限下运行。它是在 AT&T 被写成的，并以简洁、紧凑和迅速著称。默认的 Bourne Shell 提示符是美元符号$。

C Shell 在伯克利（Berkeley）开发成功，加入了大量的特性，如命令行历史、别名、内建算法、文件名补全和作业控制。跟 Bourne Shell 相比，使用交互 Shell 的用户更喜欢 C Shell，但系统管理员更喜欢用 Bourne Shell 来写脚本，因为在相同情况下，Bourne Shell 比 C Shell 更简单，执行更快。默认的 C Shell 的提示符是%。

Korn Shell 是 David Korn 在 AT&T 完成的 Bourne Shell 的超级版本，它比 C Shell 更强大，有很多新特性被加入。Korn Shell 的新特性包括：可编辑的历史、别名、函数、内建算法、正则表达式通配符、作业控制、协处理和指定排错。Bourne Shell 几乎完全向上兼容 Korn Shell，所以老版本的 Bourne Shell 在 Korn Shell 下仍可以正常地运行。Korn Shell 的默认提示符是美元符号$。

1.2.2 主要的 Linux Shell

Linux 下的 Shell 并不是专属于 Linux 操作系统。它们在 UNIX 系统下也可以自由地使用和编译。在你安装 Linux 的时候就已经初步认识了 Gnu 的 Shell 和工具，它们不是 UNIX 下的标准 Shell 和工具。虽然 Linux 支持很多 Shell，但是 Bourne Again Shell（bash）和 TC Shell（tcsh）是最流行的。Z Shell 是从 Bourne Again Shell、TC Shell 及 Korn Shell 发展来的集合了很多新特性的 Shell。Public Domain Korn Shell（pdksh）是 Korn Shell 的克隆，你可以买到合法的 AT&T（它是很多无名的小 Shell 的所有者）的 Korn Shell。

通过查看文件/etc/shell，能够得知在你的 Linux 版本下可以运行哪个 Shell，如实例 1.1 所示。

实例 1.1

```
$ cat /etc/shell
/bin/bash
/bin/sh
/bin/ash
/bin/bsn
/bin/tcsh
/bin/csh
/bin/ksh
/bin/zsh
```

说明

1 /bin/shell 文件包含了在你的 Linux 版本下可以运行的 Shell 程序的列表。最常用的版本是 bash（Bourne Again Shell）、tcsh（TC Shell）和 ksh（Korn Shell）

切换到一个/bin/shell 列表中的 Shell，可以使用 chsh 命令和 Shell 名字。例如，若想把 Shell 切换为 TC Shell，可以在命令行输入：

```
chsh /bin/tcsh
```

1.2.3 回顾 Shell 的历史

第一个有意义的标准 UNIX Shell 出现在 1979 年年末的 V7（AT&T 第七版）UNIX 中，并以其作者的名字 Stephon Bourne 命名。Bourne Shell 基于 Algol 编程语言，主要用于系统管理任务的自动执行。虽然它的简单和快速使得它很流行，但是它缺少在交互环境下使用所需的许多特性，例如历史、别名和作业控制。到了 bash，就是 Bourne Again Shell，它是由自由软件基金会的 Brian Fox 在 Copyleft 的许可证下开发的，现在已经成为流行的 Linux 操作系统的默认 Shell。它被特意设计成遵守 IEEE POSIX 1003.2/ISO9945.2 Shell 和工具标准。bash 给交互方式和脚本方式都提供了一些 Bourne Shell 没有的新特性，但是 Bourne Shell 的脚本依然可以不加修改就在 bash 下运行，它还合并了 C Shell 和 Korn Shell 许多有用的特性。它很大，且加强了 Bourne Shell 如下的方面：命令行历史和编辑、目录堆栈、作业控制、别名、函数、阵列、基于 2～64 的整数计算以及 Korn Shell 的特性，例如扩展元字符、菜单选择循环、let 命令等等。

C Shell 是 20 世纪 70 年代末期由美国加州（California）大学伯克利分校开发的，并作为 2BSD UNIX 的一部分发布，主要作者是 Bill Joy。C Shell 提供大量标准 Bourne Shell 没有提供的额外特性。C Shell 基于 C 语言，在用作编程语言时，使用跟 C 语言类似的语法。它也为交互使用而强化过，例如命令行历史、别名和作业控制。由于被设计为在大型机上使用并加入了大量的特性，所以 C Shell 在小型机器上运行显得有些慢，即使在大型机器上跟 Bourne Shell 相比也显得迟缓。

TC Shell 是 C Shell 的扩展。它的新特性是：命令行编辑（emacs 和 vi）、滚动历史列表、高级文件名、变量、命令补全、拼写检查、作业规划、自动锁定和退出登录、历史列表中的时间印章等等，所以它的尺寸也很大。

Bourne Shell 和 C Shell 的同时存在给 UNIX 的用户提供了选择，同时也引起了关于哪个 Shell 更好的争论。AT&T 的 David Kourne 在 20 世纪 80 年代中期，发明了 Kourne Shell。它在 1986 年发布，并且在 1988 年正式成为 SRV4 UNIX 的一部分。Kourne Shell 作为 Bourne Shell 的超级版本不仅运行在 UNIX 上，也运行在 OS/2、VMS 和 DOS 上。它向 Bourne Shell 提供向上兼容的能力，加入了很多 C Shell 的流行特性，迅速而高效。Korn Shell 先后有过多次修订。应用最广泛的版本是 1988 年的版本，虽然 1993 版本更受欢迎。Linux 的用户可能会发现他们使用的是 Korn Shell 的免费版本，被称为 Public Domain Korn Shell，或者简称 pdksh，是 David Korn 1988 年版本的克隆。它现在是免费和可移植的，并逐渐完全兼容在 UNIX 下的同名 Shell 和 POSIX 标准中。还有一个 Z Shell，是 Korn Shell 带有 TC Shell 特性的另一个克隆，作者是 Paul Falsted，在很多的站点上都可以免费得到它。

1.2.4 本书介绍什么 Shell?

由于 Bourne Again Shell 和 TC Shell 提供了最多的新特性，所以本书将集中介绍这两种流行的 Shell。在《UNIX Shell by Example》[2]中介绍过的 Korn Shell、Bourne Shell 和 C Shell 在这里还将被讲到。Bourne Again Shell 2.0 版本在作为程序语言时跟 Korn Shell 非常的相似，在交互使用时有很多类似 C Shell 的特性。同样，在作为程序语言的时候，TC Shell 跟 C Shell 基本相同，但是在交互使用的时候加入了许多新特性。

1.2.5 Shell 的使用

Shell 的一个主要用途是翻译提示符后面的命令。Shell 解析命令行，把它拆成由空格分隔的单词（称为“token”）。所谓的空格是指制表符、空格或者新的一行。如果命令行包含特殊的元字符，Shell 就给它们赋值。Shell 控制文件 I/O 和后台运行。在命令行完成处理以后，就开始搜寻命令并开始它的执行。

Shell 的另一个重要功能是通过设置 Shell 初始化文件，使用户的工作环境个性化。这些文件包括终端键盘设置和窗口字符的定义，设置终端类型、权限、提示和搜索路径变量的值，设置特定的应用所必需的变量，例如：windows、字处理程序和编程语言库。Bourne Shell 和 TC Shell 提供更多的特性进一步使用户环境个性化，这些特性包括历史和别名、文件名和命令补全、拼写检查、帮助、内建变量防止用户数据丢失、意外退出，以及登录，当作业完成以后通知用户等等。

Shell 也可以作为解释型的程序语言。Shell 程序也称为脚本，由文件中的命令行组成，在编辑器或在命令行中创建。这些命令通过程序结构组织在一起，这些结构包括：变量赋值、环境检测、循环等等。然而你并不需要编译脚本，因为它们在从键盘输入的时候就被逐行解释了。因为 Shell 负责脚本命令解释，所以对于用户来说理解这些命令还是相当必要的。在本书的附录 A 中列出了可用的命令以及它们的用法。

1.2.6 Shell 的响应

Shell 负责最终保证所有通过命令行输入的命令都被正确执行，这个过程包括：

（1）读取输入并解析命令行。

（2）给特殊字符赋值。

（3）建立管道、重新定向和后台进程。

（4）处理信号。

（5）建立可执行程序。

在后面，我们将针对每一个不同的 Shell 讨论以上每个项目的细节。

1.3 系统启动和登录 Shell

当你启动系统时，第一个进程称为 init。每一个进程都有其专用的进程 ID 号码，称为 PID。

2. Quigley, Ellie. *UNIX Shells by Example, 2nd Edition.* Upper Saddle River, NJ: Prentice Hall, 1999.

因为 init 是第一个进程，所以它的 PID 是 1。init 进程初始化系统，并启动另外一个进程打开命令行终端，建立命令行终端所必需的标准输入（stdin）、标准输出（stdout）和标准错误（stderr）。标准输入通常来自键盘，标准输出和标准错误显示在监视器上。从这个角度来看，一个登录提示符出现在你的控制台上。在你输入用户名以后，提示你输入密码（你应该有一个不超过 10 个字符的密码）。/bin/login 程序将通过核对/etc/passwd 的第一个域验证你的 ID。如果你的用户名确实存在，下一步就是对你的密码运行加密程序来验证你的密码是否正确。一旦密码得到验证，login 程序就把环境变量传送给 Shell，以定义一个工作环境。并在/etc/passwd 文件中提取 HOME、SHELL、USER、LOGNAME 等变量值。HOME 值表示的是你的主目录（Home Directory）。SHELL 变量中的值是你登录 Shell 的名字，被记录在 passwd 文件的最后一项里。USER 和（或）LOGNAME 表示你的登录名。PATH 变量帮助 Shell 找到常用的实用程序所在的目录。它是一个独立列表的克隆，这个列表的初始值是/usr/local/bin:/bin:/usr/bin。当登录完成时，在/etc/passwd 文件最后一项所记录的程序将被执行。通常情况下，这个程序是一个 Shell。如果/etc/passwd 文件的最后一项是/bin/tcsh 或者/bin/csh，TC Shell 程序就将被启动。如果/etc/passwd 文件的最后一项是/bin/bash、/bin/sh 或者什么都不是则 Bourne Again Shell 程序就被启动。如果/etc/passwd 文件的最后一项是/bin/pdksh，Public Domain Korn Shell 程序就被启动。这个 Shell 就称为 Login Shell。

当 Shell 启动以后，Shell 就寻找系统的初始化文件，并在你的主目录下寻找特定的 Shell 初始化文件。如果这样的文件存在，就执行它们。这些初始化文件的作用就是进一步地定制用户的环境。文件中的命令执行完以后，一个 Shell 提示符将出现在你的控制台上。如果你使用的是窗口系统，这个时候出现的将是一些 xterm（X 终端）或者可视化的 Shell 窗口。当你看见 Shell 提示符时，不论你在控制台上、xterm 还是其他的桌面窗口中，都表示 Shell 程序在等待你的输入了。

1.3.1 解析命令行

当你在提示符下输入一条命令后，Shell 读取行输入并解析命令行，把命令行拆成单个的字，称为令牌。令牌之间依靠空格和制表符分隔，以换行符作为命令行结束的标志。[3]Shell 首先判断命令行的第一个字是内建命令还是存储在磁盘上的。如果是内建命令，Shell 就执行该命令。否则，Shell 就在 PATH 变量指定的目录列表中查找这个程序。如果找到，就启动一个新的进程来执行这个程序，而 Shell 就休眠（或者等待）直到程序执行完毕，如果必要，还将报告程序的退出状态。跟着出现提示符，而整个进程都将被重新启动。命令行的处理顺序如下所示：

（1）历史记录替换（需要设置）。
（2）命令行拆分为 token（单词）。
（3）历史更新。
（4）处理引用。
（5）定义别名替换和函数（如果需要）。
（6）建立重新定向、后台和管道。
（7）变量替换（$user、$name 等等）。

3. 把命令行拆分为单词的过程称为“词法分析”（lexical analysis）。

（8）命令替换（echo for today is 'date'）。
（9）文件名替换。
（10）程序执行。

1.3.2 输入命令

别名、函数、内建命令和磁盘上的可执行程序都是可执行的命令。别名是现存命令的缩写，要依靠 C、TC、bash 和 Korn Shell 支持，它们都支持函数。所谓函数就是由一组组织在一起的命令组成的一个例程。别名和函数定义在 Shell 内存里。内建命令是 Shell 内部的例程，而可执行程序则存贮在磁盘上。Shell 使用 path 变量定位存储在磁盘上的可执行文件的位置，并在命令执行以前创建子过程。这需要花费一点时间。当 Shell 准备好执行命令的时候，按照如下的命令类型顺序执行：[4]

（1）别名。
（2）关键字。
（3）函数（bash）。
（4）内建命令。
（5）可执行程序。

例如，如果你输入的命令是“xyz”，Shell 则首先检查它不是一个别名。如果不是，再检查是不是一个内建命令或者函数，如果仍不是，那么就可肯定它是一个存储在磁盘上的可执行程序。接着，Shell 就搜索该命令所在的目录。

1.4 Shell 和进程

1.4.1 什么是进程

进程是处在执行状态下，并可以用惟一的 PID（进程 ID）标识的程序。进程由核心控制和管理。一个进程由可执行的程序、数据和堆栈、程序和指针、寄存器和其他程序运行所必需的信息组成。当你登录时，标准 Shell（bash）[5] 就启动一个进程，称为登录 Shell（Login Shell）。它属于一个用组 PID 标识的进程组。当只有一个进程组控制终端时，它被称为在前台运行。在 Linux 中，当你登录到系统时，Shell 控制终端会等待你输入命令，通常这时会启动另外一个进程——xinit，以启动 X Windows 系统。当 X Windows 系统启动后，窗口管理器进程（twm、fvmw 等）被执行，提供一个虚拟桌面。[6] 接着你可以从下拉菜单中启动其他进程，例如 xterm（一个终端）、xman（提供帮助手册）、emacs（一个文本编辑器）。多个进程同时被 Linux kernel 监视和运行，且每个进程都分配有一些 CPU 时间片，这个过程通常是不引人注目的。

1.4.2 什么是系统调用

Shell 有启动（创建）其他进程的能力。事实上，当你在提示符下或者通过脚本输入命令

4. 第 3 项和第 4 项是为 Bourne 和 Korn 保留的，C Shell 和 TC Shell 不支持第 3 项。
5. Linux 的默认 Shell 是 bash，即 Bourne Again Shell。
6. Linux 下有很多的窗口系统，包括 Gnome、KDE 和 X 等。

的时候，Shell 就以寻找内建命令或者外部可执行程序来响应，接着安排命令运行。这是依靠呼叫系统（system call）来完成的，称为系统调用。系统调用需要核心服务，这是进程访问硬件的惟一方法。有很多系统调用允许建立、执行和终止进程（当 Shell 提供重新定向和管道、命令替换、用户命令的时候，它还可以通过 kernel 提供其他的服务）。在后面的章节中我们还将讨论 Shell 利用系统调用产生新进程的问题。参见图 1.2。

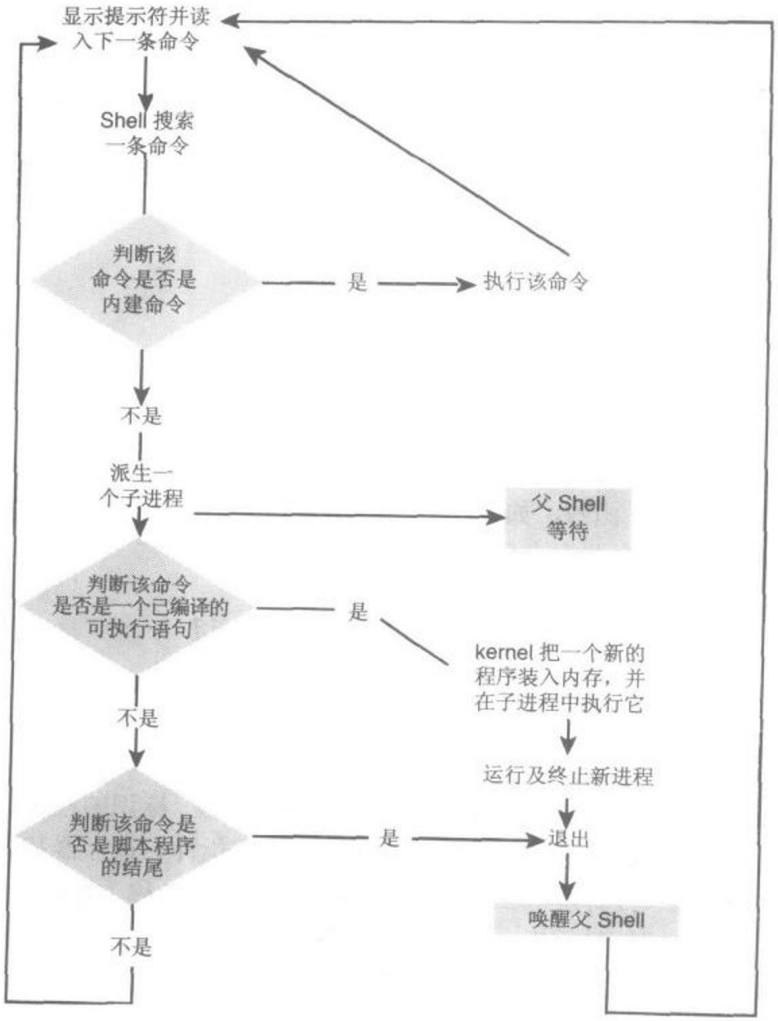


图 1.2 Shell 及命令的执行过程

1.4.3 哪些进程正在运行

ps 命令。ps 有很多的参数选项，以不同的格式列出正在运行的进程清单。实例 1.2 显示的是在用户的 Linux 系统上运行的进程列表（参考附录 A，能获得有关 ps 的更详细的用法）。

实例 1.2

```
$ ps au  (Linux ps)
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
```