UNIX Shell 实例精解

Ellie Quigley 著

张茹 译

中国电力出版社

内 容 提 要

第一本覆盖了所有三种 UNIX shell 加 awk、sed 和 grep 的书！一本书就是你学习 UNIX shell 编程所需要的全部！本书内容包括：UNIX shell 是什么，它们做什么，它们怎样与其他 UNIX 实用程序和进程相结合；创建、运行、并调试 shell 命令表；使用 grep、egrep 和 fgrep；用 sed、流编辑器操作等。

本书适合程序员和系统管理员阅读，也可作为相关人员的培训教材使用。

图书在版编目（CIP）数据

UNIX Shell 实例精解/（美）奎克力 著；张茹译. —北京：中国电力出版社，2002.7

ISBN 7-5083-1089-6

Ⅰ. U… Ⅱ. ①奎…②张… Ⅲ. UNIX 操作系统 Ⅳ. TP316.81

中国版本图书馆 CIP 数据核字（2002）第 036837 号

著作权合同登记号 图字：01-2001-2222 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）

北京市地矿印刷厂印刷

各地新华书店经售

*

2002 年 9 月第一版 2002 年 9 月北京第一次印刷

787 毫米×1092 毫米 16 开本 36 印张 820 千字

定价 69.00 元

前　言

玩“shell”游戏非常有趣。写作本书的目的是要使读者学习一些既有趣又有用的经验。自从第一版发行以来，我从许多人那里听说，他们在我的书的帮助下认识到 shell 编程根本不难！通过例子来学习就会变得容易而且有趣。实际上，正是由于这些积极的反馈，Prentice Hall 要求我写这本更新版的新书。

17 年来，我一直教授和研究有关各种 shell 和那些 shell 程序员最常用的 UNIX 实用程序的课程，本书是我这些工作的顶点。我为教课所写的讲义曾被加利福尼亚大学 Santa Cruz 学院和加利福尼亚大学 Davis UNIX 程序学院、Sun 微系统公司教育部门、Pyramid 公司教育部门、DeAnza 学院以及遍及世界的许多出版商使用过。我通常根据我的客户们的需要一次教授一种 shell 而不是三种都教。为了满足这么多客户的需要，我分别为每一种 UNIX shell 及其工具提供了单独的资料。

不论我是在教授“grep、sed 和 awk”、“针对系统管理员的 Bourne Shell”还是“交互式 Korn Shell”，总有学生问：“什么书能涵盖所有三种 shell 和重要的实用程序如 grep、sed 和 awk 呢？我应该找这样一本书呢，还是应该找一本关于 grep 和 sed 的书呢？有能真正涵盖所有这些内容的书吗？我不想为了成为一名 shell 程序员而买三四本书。”

作为对这些问题的回答，我能推荐出许多分别涵盖这些主题的好书，而且有些 UNIX 书籍试图涵盖所有内容，但学生想要一本包含一切的书而不仅仅是一个简要概述。他们想让 UNIX 工具、正则表达式、所有三种 shell、引用规则、三种 shell 的比较、练习等都包含在一本书里。本书正是如此。当我写这本书时，我回顾自己是怎样教课的，并用同样的形式组织章节。在 shell 编程课上，第一个主题总是对什么是 shell 以及它是如何工作的一个介绍。然后我们讨论 UNIX 中如 grep、sed 和 awk 这样的实用程序，shell 程序员工具箱里的最重要的工具。当学习 shell 时，它首先是作为一种交互式编程出现的，任何事都可以在命令行里完成，其次是作为一种编程语言，编程结构在 shell 命令表里描述和示范。不论 shell 编程课是持续两天，还是一周，又甚或是一个学期，当课程结束时，学生都会精通于写命令表并对此感到兴奋。他们学会了如何玩 shell 游戏。不管你是当一门课程来学还是只愁自己玩玩，本书都将教你如何玩同一种游戏。

由于人们一般总是发现简单的例子更易于迅速地理解，所以每个概念都放在一个小例子里，例子后面有输出结果和对各程序行的注释。这种方法已经被证明是非常受他们欢迎的，而且对于那些需要写、读和维护 shell 程序的人来说，本书现在已经被充分接受了。

并列介绍三种 shell 以便如果你想知道重定向是如何在一种 shell 中实现的，那么在各个不同的 shell 章节中都有对这个主题的讨论，而且为了快速比较，在本书的附录 B 中有一个表。

当你只想要关于一条特定命令的足够信息来使自己慢慢想起这条命令是如何工作的

时，不得不去找别的书或者 UNIX 手册是非常烦人的。为了节省时间，附录 A 包含了一个有用命令及其语法和定义的表。给更强大且常用的命令提供了例子和注释。

附录 B 中的比较表将帮助你始终都能正确区别各种 shell，尤其当你把命令表从一种 shell 改成另一种 shell 时，而且当你所需要的只是想起来命令表怎么工作时，它也可以用作一种快速语法检查。

对 shell 程序员来说，最大的障碍之一是正确使用引号。附录 C 中的引用规则部分给出了在某些最复杂的命令行中成功地引用的每一步过程。这个过程大大减少了程序员在调试命令表时，徒劳地试图正确地匹配引号所浪费的大量时间。

我想你将发现这本书是一本有价值的辅导书和参考书。本书的目标是用例子解释事情并使事情简单化，以便你有兴趣学习并能节省时间。因为本书重复了我在课上所讲的内容，所以我相信你能够在短期内成为一位多产的 shell 程序员。本书包含了你所需要的一切。玩 shell 游戏很有趣。你将会发现这一点！

Ellie Quigley(ellieq@ellieq.com)

目　录

前　言

第 1 章　关于 UNIX shell 的介绍 ····· 1

1.1　定义和功能 ····· 1
1.2　系统启动和注册 shell ····· 3
1.3　进程和 shell ····· 5
1.4　环境和继承 ····· 9
1.5　执行命令表中的命令 ····· 18

第 2 章　UNIX 工具箱 ····· 25

2.1　正则表达式 ····· 25
2.2　组合正则表达式元字符 ····· 33

第 3 章　grep 家族 ····· 39

3.1　grep 命令 ····· 39
3.2　带正则表达式的 grep 例子 ····· 42
3.3　用管道的 grep ····· 48
3.4　带选项的 grep ····· 48
3.5　egrep（扩展的 grep） ····· 51
3.6　固定 grep 或快捷 grep ····· 55

第 4 章　流编辑器 ····· 57

4.1　什么是 sed ····· 57
4.2　sed 如何工作 ····· 57
4.3　寻址 ····· 57
4.4　命令和选项 ····· 58
4.5　出错消息和退出状态 ····· 59
4.6　sed 的实例 ····· 61
4.7　sed 编写命令表 ····· 75

第 5 章　awk 实用程序：awk 作为一种 UNIX 工具 ····· 80

5.1　什么是 awk ····· 80
5.2　awk 的格式 ····· 80
5.3　格式化输出 ····· 83
5.4　在一个文件里的 awk 命令 ····· 87

5.5 记录和域……88
5.6 模式和操作……92
5.7 正则表达式……94
5.8 在一个命令表文件里的 awk 命令……96
5.9 回顾……97
第 6 章 awk 实用程序：awk 编程结构……106
6.1 比较表达式……106
6.2 回顾……111
6.3 UNIX 工具实验 4……118
第 7 章 awk 实用程序：awk 编程……119
7.1 变量……119
7.2 重定向和管道……123
7.3 管道……126
7.4 关闭文件和管道……127
7.5 回顾……128
7.6 条件语句……137
7.7 循环……140
7.8 程序控制语句……141
7.9 数组……142
7.10 awk 内置函数……151
7.11 内置算术运算函数……154
7.12 用户自定义函数（nawk）……156
7.13 回顾……158
7.14 特殊情况和结束……163
7.15 回顾……171
第 8 章 交互式 Bourne shell……178
8.1 启动……178
8.2 用 Bourne shell 编程……213
第 9 章 C shell……286
9.1 交互式 C shell……286
9.2 用 C shell 编程……334
第 10 章 Korn shell……378
10.1 交互式 Korn shell……378
10.2 用 Korn shell 编程……438
附录 A 对 shell 程序员有用的 UNIX 实用程序……528

附录 B　比较三种 shell……560
附录 C　正确使用引用的步骤……565

第 1 章　关于 UNIX shell 的介绍

1.1　定义和功能

shell 是一种特殊程序，它被用作用户和一个叫做内核（kernel）的 UNIX 操作系统核心之间的界面，如图 1.1 所示。内核在系统引导时装入内存并一直管理系统直到关机为止。它创建并控制进程，并且管理内存、文件系统、通信等等。所有其他程序，包括 shell 程序，都放在外部磁盘上。内核把这些程序装入内存，执行程序，并在它们终止时清理系统。shell 是一种当你注册时启动的实用程序。它通过解释敲入到命令行或一个命令表文件里的命令来允许用户与内核交互。

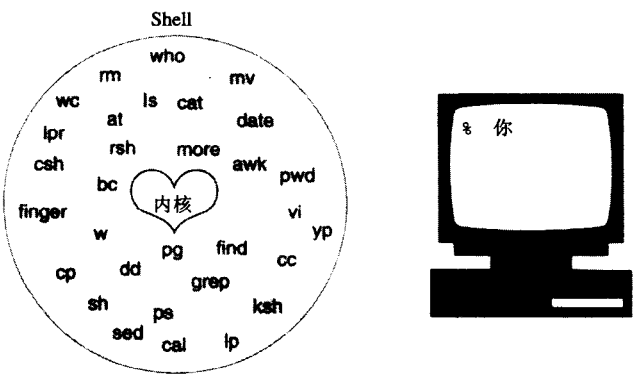

图 1.1　内核、shell 和你

当你注册时，一种交互式 shell 启动并提示你输入。当你敲入一个命令后，shell 的任务是：（a）分析命令行；（b）处理通配符、重定向、管道和作业控制；（c）搜索命令，而且如果找到了，就执行该命令。当你初次学习 UNIX 时，会把大部分时间花在根据提示执行命令上。你要交互式使用 shell。

如果经常敲入同样的命令集，你可能希望能自动执行这些任务。命令表文件允许你把命令放在一个叫做命令表文件（script file）的文件里，然后执行该文件。shell 命令表非常像批文件：它是敲入文件的 UNIX 命令的一个表，然后该文件会被执行。更高级的命令表包含判断、循环、文件测试等编程结构。写命令表不仅要求学习编程结构和技巧，而且假定你对 UNIX 实用程序及它们如何工作有很好的认识。有些实用程序，如 grep、sed 和 awk，是在命令表中用来操作命令输出结果和文件的强大工具。在你熟悉了这些特定 shell 的工具和编程结构后，你就准备好开始写有用的命令表了。当执行命令表里的命令时，是在把命令表用作一种编程语言来使有的。

1.1.1 三种主要的 shell

大多数 UNIX 系统支持的三种性能卓越的 shell 是 Bourne shell（AT&T shell）、C shell（Berkley shell）和 Korn shell（Bourne shell 的超集）。所有这三种 shell 在交互式运行时性能很类似，但是当用作编写命令表的语言时在语法和效率方面有些不同。

Bourne shell 是标准 UNIX shell，且这种 shell 用于管理系统。大多数系统管理命令表，例如 rc start 和 stop 命令表以及 shutdown 都是 Bourne shell 命令表，而且当处于单用户模式时，这是管理员作为超级用户运行程序时常用的 shell。这种 shell 是在 AT&T 写的，而且以简明、精悍和快速而著称。默认的 Bourne shell 提示符是美元符号（$）。

C shell 是在 Berkeley 研制的，而且被加上了许多特征，如命令行历史、别名、内置算术运算、文件名完成和作业控制。用户交互式运行 shell 时，C shell 比 Bourne shell 更受欢迎，但是管理员更愿意使用 Bourne shell 来写命令表，因为 Bourne shell 命令表比用 C shell 写的同样的命令表更简单，且更快捷。默认的 C shell 提示符是百分号（%）。

Korn shell 是 David Korn 在 AT&T 写的 Bourne shell 的超集。基于 C shell 增强版给该 shell 加入了许多特征，并且它的特征超过了 C shell 增强版。Korn shell 的特征包括可编辑的历史、别名、函数、正则表达式通配符、内置算术运算、作业控制、协处理和特殊调试特征。Bourne shell 几乎与 Korn shell 完全向上兼容，所以旧的 Bourne shell 程序将会在这种 shell 中运行得很好。默认的 Korn shell 提示符是美元符号（$）。

1.1.2 shell 的历史

第一个重要的标准 UNIX shell 是 1979 年底在 V7 UNIX（AT&T 第 7 版）中引入的，并且以它的创造者，Stephen Bourne 的名字命名。Bourne shell 作为编程语言是基于 Algol 语言的，而且主要用于使系统管理任务自动化。虽然它由于简单和速度而流行，却缺乏交互式使用的许多特性，如历史、别名和作业控制。

20 世纪 70 年代末在加利福尼亚大学 Berkeley 研制的 C shell 是作为 2BSD UNIX 的一部分发布的。这种主要由 Bill Joy 所写的 shell 提供了许多标准 Bourne shell 里不提供的附加特征。C shell 是基于 C 编程语言的，而且当用作编程语言时，它使用一种类似的语法。它也提供交互使用的增强功能，如命令行历史、别名和作业控制。因为这种 shell 是在大型机上设计的，而且加入了大量的附加特征，所以跟 Bourne shell 相比，C shell 在小型机上有运行缓慢的趋势，甚至在大型机上也运行迟缓。

由于 Bourne shell 和 C shell 都可用，所以现在 UNIX 用户有了选择余地，同时也在哪一种 shell 更好的问题上产生了矛盾。来自 AT&T 的 David Korn 于 20 世纪 80 年代中期发明了 Korn shell。它发布于 1986 年，并且在 1988 年正式成为 UNIX 的 SVR4 分支的组成部分。其实是 Bourne shell 的超集的 Korn shell，不仅能在 UNIX 系统上运行，而且能在 OS/2、VMS 和 DOS 上运行。它提供与 Bourne shell 的向上兼容性，增加了许多 C shell 的受欢迎的特征，而且快捷有效。Korn shell 经历了许多修订版。要找出你正在运行的是哪个版本，可以在 ksh

提示符下按^v（Control 键和 v 键）[①]。本书描述了 Korn shell 的 1988 年版本，它是最广为使用的版本。对 1993 年加入的新特征也做了描述。

1.1.3 shell 的使用

shell 的主要功能之一是解释交互式运行时，在命令行提示符下敲入的命令。shell 分析命令行，把它划分成由空白符分隔的词（叫做标记 tokens），空白符包括跳格符、空格符或换行符。如果词包含特殊的元字符，则 shell 就计算它。shell 处理文件的输入/输出和后台处理。在处理完命令行之后，shell 寻找命令并开始执行它。

shell 的另一个重要功能是定制用户环境，通常在 shell 初始化文件中做这种工作。这些文件包含下列定义：设置终端键及窗口特征；设置定义搜索路径、权限、提示和终端类型的变量；并设置特定应用程序如窗口、文本处理程序和编程语言库所要用的变量。Korn shell 和 C shell 也通过附加内部变量集来防止用户因注销时疏忽而破坏文件，以及为了在作业完成时通知用户而设置的历史和别名，内置变量，来提供进一步的定制。

shell 也可以用作解释编程语言。shell 程序也叫命令表，由在文件中列出的命令组成。程序在编辑器中创建（虽然联机编写命令表也是允许的）。它们包括散布着基本编程结构如变量赋值、条件测试和循环的 UNIX 命令。你不必编译 shell 命令表。shell 解释每行命令表就像它是从键盘输入的一样。因为 shell 负责解释命令，所以用户有必要理解那些命令是干什么用的。本书的附录 A 里包含了有用的命令和它们如何工作的列表。

1.1.4 shell 的任务

shell 最终的任务是确保在提示符下敲入的任何命令都能被正确地执行。那些任务包括：

1. 读取并分析命令行。
2. 给特殊字符求值。
3. 设置管道、重定向和后台处理。
4. 处理信号。
5. 设置执行程序。

每个主题都在它所属的特定 shell 中进行详细的讨论。

1.2 系统启动和注册 shell

当你启动系统时，第一个进程叫做 init。每个进程都有与之相关联的进程标识符，叫做 PID。因为 init 是第一个进程，所以它的 PID 值是 1。init 进程初始化系统，然后启动另一个

① Korn shell 必须设置交互式编辑器来用^v。你也可以敲入：strings/bin/ksh/grepVersion。

进程打开终端线，并设置所有与终端相关联的标准输入（stdin），标准输出（stdout），和标准出错输出（stderr）。标准输入通常来自键盘；标准输出和标准出错输出送到屏幕。这时，一个注册提示符将出现在终端上。

你敲入注册名后，将会被提示输入口令。然后/bin/login 程序通过检查 passwd 文件的第一个域来验证你的身份。如果那里有你的用户名，则下一步就是用一个解密程序来操作你敲入的口令，以判断是否的确是正确的口令。一旦口令被确认，login 程序就设置一个包含变量的初始环境，这些变量定义了将会传递给 shell 的工作环境。从 passwd 文件中提取信息赋给 HOME、SHELL、USER 和 LOGNAME 变量。给 HOME 变量赋你的起始目录；给 SHELL 变量赋注册的 shell 名，passwd 文件中输入的最后一项。给 USER 和/或 LOGNAME 变量赋你的注册名。设置一个搜索路径变量以便能够在指定目录下找到常用实用程序。当 login 完成后，将执行在 passwd 文件最后一项中，找到的程序。通常这个程序就是一种 shell。如果 passwd 文件中最后一项是/bin/csh，则 C shell 程序将被执行。如果最后一项是/bin/sh 或是空，则 Bourne shell 将会启动。如果最后一项是/bin/ksh，则 Korn shell 程序将被执行。这个 shell 叫做注册 shell（login shell）。

shell 启动之后，它查找由系统管理员安装的任何全系统初始化文件，然后检查起始目录看看那里是否有指定 shell 的初始化文件。如果有任何这种文件存在，就执行它们。初始化文件用来进一步定制用户环境。那些文件中的命令被执行后，屏幕上出现一个提示符。shell 现在正等你输入。

1.2.1 分析命令行

当你在提示下敲入一个命令时，shell 读取输入行并分析该命令行，把行划分成叫做标记（tokens）的词。标记用空格符和跳格符划分，而且命令行用换行符来结束[②]。然后 shell 查看第一个词是一个内置命令还是位于外部磁盘上某处的一个可执行程序。如果是内置的，则 shell 将在内部执行该命令。否则，shell 将搜索路径变量中列出的目录，来找出程序存放的位置。如果命令找到了，shell 将派生一个新进程，然后执行该程序。shell 将一直睡眠（或等待），直到程序执行完毕，然后，如果必要的话，将报告退出程序的状态。将会出现一个提示符，且整个进程将再次启动。处理命令行的顺序如下：

1. 历史置换（如果可用的话）。
2. 命令行划分成标记，即词。
3. 更新历史（如果可用的话）。
4. 处理引号。
5. 定义别名替换和函数（如果可用的话）。
6. 设置重定向、后台和管道。
7. 执行变量替换（$user、$name 等）。

② 把行划分成标记的过程叫做词法分析（lexical analysis）。

8．执行命令替换（today 的回送是‘date’）。

9．执行叫做 globbing 的文件名替换（cat abc.??、rm*.c 等）。

10．执行程序。

1.2.2 命令类型

当执行一个命令时，它是一个别名、一个函数、一个内置命令、一个程序或一个在磁盘上的可执行程序。别名是对现有命令的简称（绰号），且只在 C 和 Korn shll 中使用。函数只用在 Bourne（与 AT&T System V，第 2 版一起引入的）和 Korn shell 中。它们是组织成独立的例行程序的命令组。别名和函数在 shell 的内存中定义。内置命令是 shell 中的内部例行程序，而且可执行程序驻留在磁盘上。shell 用路径变量来定位磁盘上的可执行程序，并在命令可执行前派生一个子进程。这占用了时间。当 shell 准备好了执行命令时，它按下列次序估计命令类型：

1．别名。

2．内置命令。

3．函数（Korn 和 Bourne shell）。

4．可执行程序。

例如，命令是‘xyz’，shell 将查看‘xyz’是不是一个别名。如果不是，那么它是一个内置命令还是一个函数？如果都不是，那么它一定是一个存放在磁盘上的可执行程序。然后 shell 必须搜索该命令的路径。

1.3 进程和 shell

进程是正在执行的程序，并可以用它惟一的 PID 号（进程标识符）来标识。内核控制并管理进程。一个进程由可执行程序、它的数据和堆栈、程序和栈指针、寄存器以及所有程序运行所需要的信息组成。当你启动 shell 时，它就是一个进程。shell 属于由组 PID 标识的一个进程组。一次只有一个进程组控制终端并被认为是在前台运行。当你注册时，shell 控制终端并等你在提示下敲入一个命令。

shell 可以产生其他进程。其实，当你在提示下或从一个 shell 命令表输入一个命令时，shell 就负责在它的内部代码（内置）或外部磁盘上找到该命令，然后安排执行该命令。这通过调用内核来实现，叫做系统调用（system calls）。系统调用是对内核服务的一种请求和进程可以对系统硬件进行访问的惟一方式。有许多系统调用允许创建、执行和终止进程。（当 shell 执行重定向和管道传送、命令替换及用户命令的执行时，提供来自内核的其他服务。）

shell 用来引起新进程运行的系统调用在以后章节中讨论。过程如图 1.2 所示。

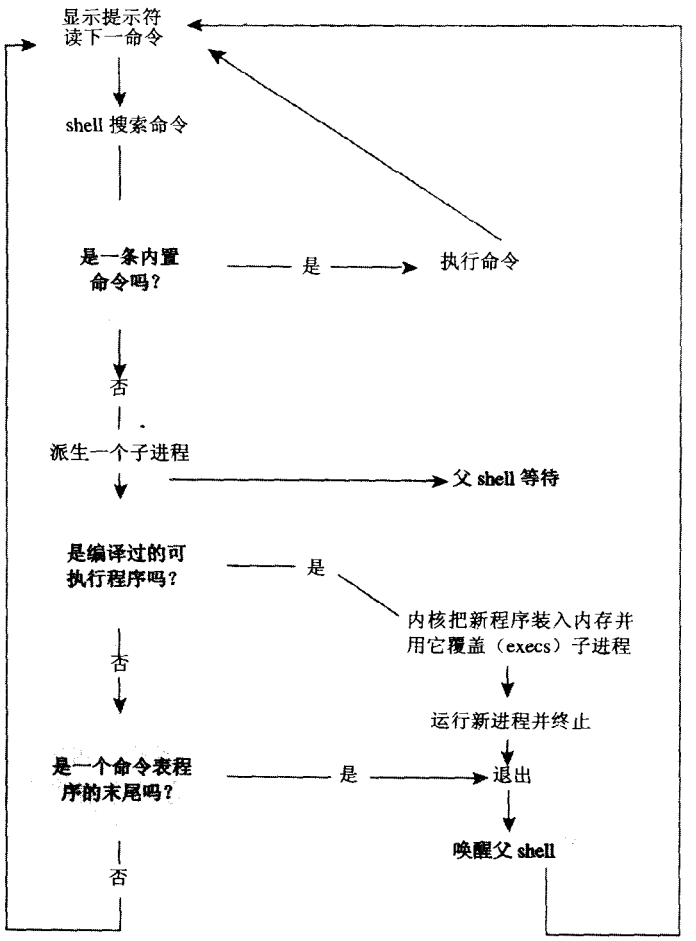

图 1.2　shell 和命令执行

1.3.1　创建进程

fork 系统调用。UNIX 中用 fork 系统调用（fork system call）创建一个进程。fork 系统调用创建调用进程的一个副本。新进程称为子（child）进程，而创建它的进程叫做父（parent）进程。子进程在调用 fork 后立刻启动运行，而且两个进程起初共享 CPU。子进程有父进程的环境、打开的文件、真实的用户标识符、通用掩码、当前工作目录和信号的一个拷贝。

当你敲入一个命令时，shell 分析命令行并判断第一个词是一个内置命令还是一个存放在外部磁盘上的可执行命令。如果命令是内置的，则 shell 处理它，但是如果是在磁盘上，则 shell 调用 fork 系统调用来复制自己（图 1.3）。它的子进程将搜索路径找命令，同时给重定向、管道、命令替换和后台处理设置文件描述符。当子进程 shell 工作时，父进程一般进入睡眠（见下面的 wait 系统调用）。

wait 系统调用。在子进程管理细节如处理重定向、管道和后台处理时，给父进程 shell 编程以进入睡眠（等待）。wait 系统调用使父进程挂起，直到它的子进程之一终止。如果 wait 成功，则它返回死的子进程的 PID 和它的退出状态。如果父进程不等待且退出子进程，则子进程被放置在僵状态（动态挂起）并将保持这个状态直到父进程调用 wait 或者父进程死亡[③]。如果父进程在子进程之前死亡，则 init 进程选用任意孤立的僵进程。然后，wait 系统调用不只用于使一个父进程进入睡眠，而且要确保进程正确地终止。

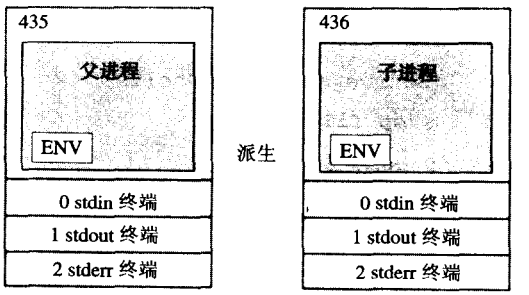

图 1.3　fork 系统调用

exec 系统调用。你在终端输入一个命令后，shell 通常派生一个新 shell 进程：子进程。就像早先提到的，子进程 shell 负责执行你敲入的命令。通过调用 exec 系统调用来实现它。记住，用户命令其实只是一个可执行程序。shell 搜索新程序的路径。如果找到了，则 shell 调用 exec 系统调用，把命令名当作它的变元。内核把新程序装入内存取代调用它的 shell。然后子 shell 被新程序覆盖。新程序成为子进程并开始执行。虽然新进程有自己的局域变量，但是所有的环境变量、打开的文件、信号和当前工作目录都被传递给新进程。这个进程在完成后退出，并且唤醒父进程 shell。

exit 系统调用。新程序可以在任何时候通过执行 exit 系统调用来终止。当一个子进程终止时，它发送一个信号（sigchild）并等待父进程接收它的退出状态。退出状态是一个 0 到 255 之间的数。0 退出状态表明程序执行成功，而非 0 退出状态意味着程序在某些方面执行失败。

例如，如果在命令行中敲入命令 ls，父进程 shell 将派生（fork）一个子进程并进入睡眠。然后子进程 shell 将取代它来执行 exec（覆盖）ls 程序。Ls 程序将取代子进程运行，继承所有的环境变量、打开的文件、用户信息和状态信息。当新进程执行完毕时，它将退出并唤醒父进程 shell。屏幕上将出现一个提示符，而且 shell 将等待另一个命令。如果你有兴趣知道一个命令是怎么退出的，每个 shell 都有一个专用内置变量包含终止的最后一个命令的退出状态。（所有这些将在各种 shell 章节中详细解释。）图 1.4 是一个进程创建和终止的例子。

实例 1.1

```
    (C Shell)
1   % cp filex filey
    % echo $status
    0
2   % cp xyz
```

③ 要删除僵进程，系统必须重新启动。

```
Usage: cp [-ip] f1 f2; or: cp [-ipr] f1 ... fn d2
% echo $status
1
(Bourne and Korn Shells)
$ cp filex filey
$ echo $?
0
$ cp xyz
Usage: cp [-ip] f1 f2; or: cp [-ipr] f1 ... fn d2
$ echo $?
1
```

解释

1. 在 C shell 命令行提示符下输入 cp（复制）命令。命令复制了 filex 的一个叫做 filey 的拷贝后，程序退出并出现提示符。Csh 的 status 变量包含执行的最后一个命令的退出状态。如果状态为 0，则 cp 程序成功地退出。如果退出状态非 0，则 cp 程序在某些方面执行失败。

2. 当输入 cp 命令时，用户未能提供两个文件名：源文件和目标文件。cp 程序向屏幕发送一条出错信息，并退出，状态为 1。这个数存在 csh 的 status 变量中。任何不是 0 的数都表示程序执行失败。

3. Bourne 和 Korn shell 对 cp 命令所做的处理跟 C shell 在头两个例子里做法一样。仅有的差别是 Bourne 和 Korn shell 都把退出状态存在？变量中，而不是 status 变量中。

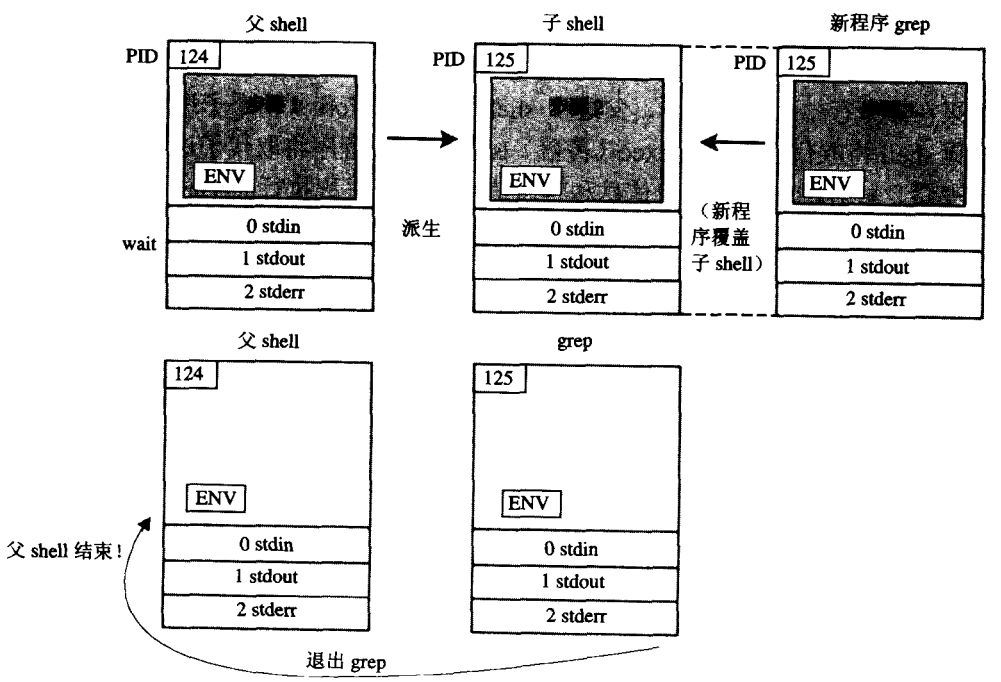

图 1.4　pork、exec、wait 和 exit 系统调用

注释

1．父进程 shell 用 pork 系统调用创建自己的一个拷贝。该拷贝叫做子进程 shell。

2．子进程 shell 有新的 PID 而且是它的父进程 shell 的一个拷贝。它将与父进程 shell 共享 CPU。

3．内核把 grep 装入内存并取代 shell 来执行（exec）它。grep 程序从子进程 shell 里继承打开的文件和环境。

4．grep 程序退出，内核清理干净，并唤醒父进程 shell。

1.4 环境和继承

当你注册时，shell 启动并从启动它的/bin/login 程序里继承许多变量、输入/输出流和进程特性。依此类推，如果从注册 shell 或父进程 shell 里生成（派生）另一个 shell，则子进程 shell(subshell)将从父进程 shell 里继承某些特性。一个 subshell 可能因为许多原因而被启动：因为处理后台进程；因为处理命令组；或者因为执行命令表。子进程 shell 从父进程 shell 里继承一个环境。环境包括进程权限（谁拥有该进程）、工作目录、文件创建掩码、专用变量、打开的文件和信号。

1.4.1 所有权

当你注册时，给 shell 一个身份。它有一个实用户标识符（UID）、一个或几个实组标识符（GID）和一个有效用户标识符及有效组标识符（EUID 和 EGID）。EUID 和 EGID 开始与实 UID 和 GID 相同。这些 ID 号能在 passwd 文件中找到且被系统用来标识用户和组。EUID 和 EGID 判断一个进程在读、写或执行文件时有什么访问权限。如果一个进程的 EUID 和文件属主（owner）的实 UID 相同，则进程对这个文件有属主访问权限。如果 EGID 和一个进程的实 GID 相同，则进程有属主组特权。

来自/etc/passwd 文件的 UID 叫做实 UID，是一个与注册名相关联的正整数。实 UID 是口令文件的第三个域。当你注册时，给注册 shell 赋实 UID 且所有从注册 shell 生成的进程都继承它的权限。任何有一个零 UID 值的正在运行的进程都属于 root（超级用户）且有 root 特权。实组标识符 GID，把你的注册名与一个组相关联。可以在口令文件的第四个域中找到它。

EUID 和 EGID 可以变成赋给别的属主的数。通过把 EUID（或 EGID[④]）改成另一个属主，你可以成为一个属于其他人的进程的属主。把 EUID 或 EGID 改成另一个属主的程序叫做 setuid 或 setgid 程序。/bin/passwd 程序是一个 setuid 程序的例子，它授予用户超级用户

④ Setgrid 权限在使用时是依赖系统的。在某些系统中，目录的 setgrid 可能导致在该目录中创建的文件属于该目录所属的同一个组。在另一些系统中，进程的 EGID 决定可以使用该文件的组。

（root）特权。setuid 程序常常是安全漏洞的来源。shell 允许你创建 setuid 命令表，且 shell 自身可以是一个 setuid 程序。

1.4.2 文件创建掩码

创建一个文件时，给文件一组默认权限。这些权限由创建文件的程序决定。子进程从它们的父进程继承一个默认的掩码。用户可以通过在提示符后发布 umask 命令，或者在 shell 初始化文件中进行设置，来给 shell 改变掩码。Umask 命令用于从现存掩码中删除权限。

开始，umask 是 000，给一个目录 777（rwxrwxrwx）权限并给一个文件 666（rw-rw-rw）权限作为默认值。在大多数系统中，由/bin/login 程序或/etc/profile 初始化文件给 umask 赋一个值 022。

从目录和文件的权限默认设置中减去 umask 值，如下所示：

777（目录）	666（文件）
- 022（umask 值）	- 022（umask 值）
---	---
755	644
结果：**drwxr-xr-x**	**-rw-r--r--**

设置了 umask 后，这个进程创建的所有目录和文件被赋成新的默认权限。在这个例子里，给属主对目录的读、写和执行权限；给组读和执行权限；而给所有其余的人（其他人）读和执行权限。对属主来说，给创建的任何文件都赋读和写权限，并给组和其他人读权限。要改变个别目录的权限和权限，就用 chmod 命令。

1.4.3 用 chmod 改变权限

每个 UNIX 文件都有一个属主。只有属主或超级用户可以通过发布 chmod 命令来改变一个文件或目录的权限。下面的例子描述了权限模式。一个组可能有许多成员，且文件属主可能会改变文件的组权限，以便组可以享受专门的特权。

chown 命令改变文件和目录的属主和组。只有属主和超级用户能调用它。在 UNIX 的 BSD 版中，只有超级用户 root 能改变所有权。

每个 UNIX 文件都有一组与之相关的权限来控制谁能读、写或执行该文件。共有 9 位构成一个文件的权限。第一个三位组控制文件属主权限，第二个三位组控制组权限，最后一个三位组控制其他组或其余所有人的权限。权限存在文件索引节点的 mode 域里。

chmod 命令改变文件和目录的权限。用户要改变文件的权限就必须拥有它们[5]。

表 1.1 描述了用来改变权限的数字的八种可能组合。

⑤ 调用者的 EUID 必须与文件的属主的 UID 相匹配，否则属主必须是超级用户。