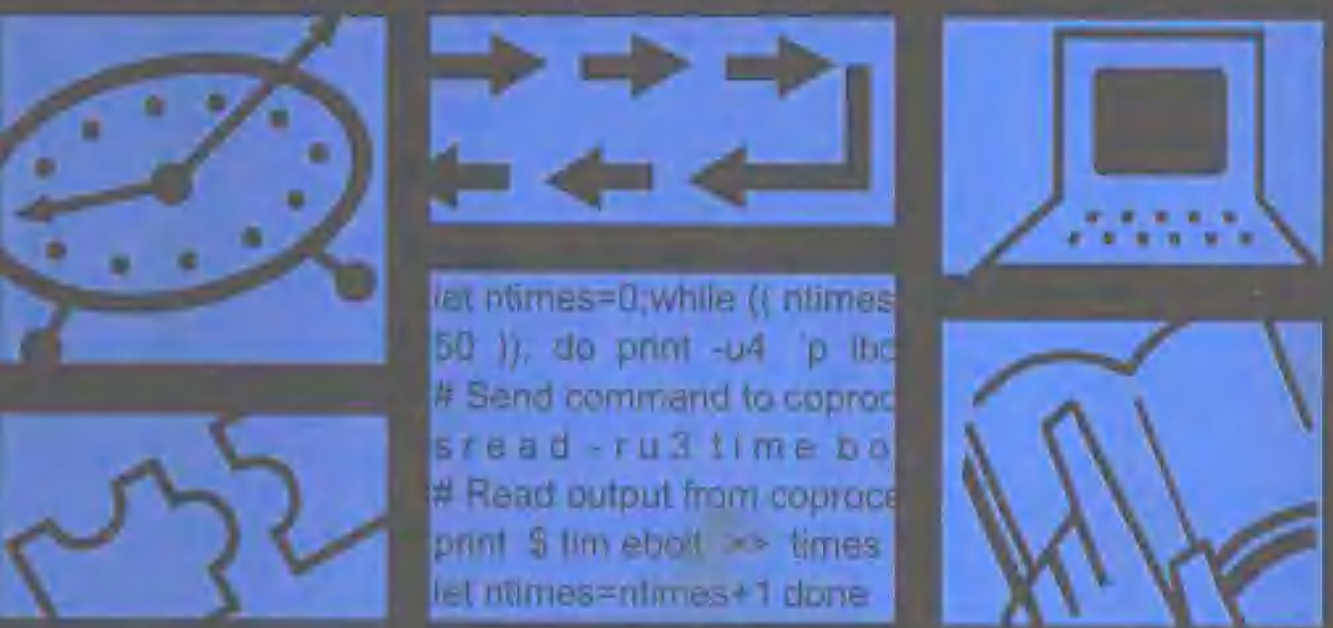
（美）Dennis O' Brien David Pitts 著

Korn Shell

实例编程指南

Korn Shell Programming By Example



中国青年出版社

que®

Korn Shell

实 例 编 程 指 南

Korn Shell Programming By Example

（美）Dennis O’Brien David Pitts 著
杨天庆 译

中国青年出版社
CHINA YOUTH PRESS

（京）新登字 083 号

版权贸易合同登记号：01-2001-1024

策　　划：胡守文
　　王修文
　　郭　光
责任编辑：江　颖
　　黄　渲
责任校对：肖新民

书　　名：《Korn Shell 实例编程指南》
编　　著：（美）Dennis O' Brien　David Pitts
出版发行：中国青年出版社
　　地址：北京市东四十二条 21 号　邮政编码：100708
　　电话：（010）84015588　传真：（010）64053266
印　　刷：山东高唐印刷有限责任公司
开　　本：787mm × 1092mm　1/16
版　　次：2001 年 10 月北京第 1 版
印　　次：2001 年 10 月第 1 次印刷
印　　数：1-5000
书　　号：ISBN 7-5006-4561-9/TP・217
定　　价：36.00 元

作者简介

Dennis O'Brien 70年代初开始从事计算机专业工作，当时是一名COBOL程序员。他参与了业界许多大胆的举措。

这是他第一本出版的书籍，但是他已经发表了许多专业论文，并且在个人软件中有许多他公开的代码。在他的公司Bruden Corp（www.bruden.com）里，他还为技术培训班编写了一些关于C语言，shell程序设计，UNIX用户，UNIX管理，UNIX内核以及开放式VMS课程的教材。

Dennis是Bruden公司的合伙人。他早在1972年就开始在计算机领域从事工作，1983年与Digital Equipment公司合作做开放式VMS的培训，1989年与Vastek公司和Bruden公司合作提供UNIX的培训。他是Vastek公司的奠基人之一。在为Digital Equipment公司工作期间，他获得了五次Instructor Excellence奖。

他培训了许多Compaq Computer公司的用户。许多Compaq Computer公司的工程师和用户都认为他是一名业界最好的技术指导老师，他能够清楚的了解初学者和学员的需要，并且使技术学起来轻松而又不失有趣。

Dennis与妻子和继子生活在麻萨诸塞州。在dennis.obrien@bruden.com上可以与Dennis联系。

David Pitts(dpitts@mk.net)曾与别人合作出版过多部书籍。他是Red Hat Linux Unleashed（第二版和第三版）、Red Hat Linux 6 Unleashed和Linux Unleashed（第四版）的主要作者，他是Que出版的《Using UNIX》、《使用Linux》、《Programming CGI in Perl》、《Visual, Basic》和《C语言》的主要作者。

David是Pitts Technical Resources,Inc（http://www.dpitts.com）的首席执行官．这家公司主要是进行世界各地程序设计和发展的咨询。

David与他挚爱的妻子生活在Kentucky（http://www.visitlex.com），高中毕业直到从Asbury大学（www.asbury.edu）毕业David一直从事程序设计。他毕生的目标就是出现在所有媒体之上，他一直想出现在一个全国性质的电视节目上和出演一部电影的角色。

前　言

你手里拿的这本书能做些什么呢？本书将教会你，一个系统管理员或用户学习怎样使用 Korn shell 编写程序。我的一个使用 Linux 的朋友问我“什么，你这本书不适用于 bash？”这里我要说明两件事情。第一，ksh 是一种非常好的 shell 语言（ksh 代表 Korn shell program,bash 代表 Bash shell，通常用于 Linux 系统中）。Korn shell 中的许多特性使它成为系统管理员创建脚本的一个极好的选择；第二，Korn shell 成为 UNIX 系统中的标准脚本语言已经很多年了。我写这本书的目的就是教你怎样简化你日常的工作，从而使你成为一个优秀的系统管理员。而且我力图以通俗易懂的语言，诙谐幽默的风格实现我的这个目的，尽量做到不会让你在学习本书的时候感到生涩又枯燥乏味。

一些专业人员所需要的、所正在寻找的就是一本能够帮助他简化工作任务，而又摆脱乏味单调的日常工作的好书。那我向你保证，你现在拿到的这本书就可以为你提供这样的帮助。

说了这么多，所要表达的意思是很明显的，这本书不是一本传统的 UNIX/Linux 专业书籍。它是一本实用性很强的包含诸多实例的手册性质的书。

对读者的要求

对本书的读者我做了两个假定，而且本书的内容是基于在这两个假设之上的。第一，假定读者已经阅读过 UNIX/Linux 的有关书籍，或者对 Korn Shell 有一定的了解。第二，假定读者熟悉 UNIX/Linux 的命令和命令行。在本书后面的附录 A 中列出了许多有用的命令，及对这些命令的说明，其中的许多命令在本书中的实例里都会用到。另外本书的实例都是基于 korn Shell 93 版本的，其中在有的实例或所介绍的技术不适用于早期 Korn Shell 版本的地方，我都给予了提示说明。Korn Shell 93 版本的编程语言在网上是很容易得到的，当然这仅限于个人使用。在搜索引擎中做一个对“Korn Shell”的查询，很容易就可以得到相关的站点，在那里你可以下载 Korn Shell 93 程序语言。

本书可以为读者提供：

本书将为你提供许多有用的、优秀的、测试过的脚本实例。这些脚本实例都是易学、

易用的，对于你编写脚本将有很大的帮助，就是说我为你提供了许多你所需要的信息，不过我认为，实例和信息我提供给你了，要想尽数掌握，关键还是在于你自己的努力和学习。

目 录

第1章 环 境

本书，特别是本章是建立在假设读者熟悉 shell 的基础上的。你可能已经使用过一些命令，比如更改密码之类的，对环境有一定的了解。本章开头将介绍几个脚本中常见的概念，其中包括程序中的注释，以及大多数 shell 脚本开头第一行中出现的“#!/bin/ksh”。之后将解释几个与环境有关的概念。

本章主要内容如下：

- #! /bin/ksh 的含义
- 怎样理解文件和目录权限
- 怎样使用 chmod 和 umask 命令
- 怎样在你的脚本中添加注释
- 怎样使用.profile 文件
- 什么是 Korn shell 环境选项
- 怎样使用 shell 变量和环境变量
- 怎样使用历史文件和命令行复检

1.1 什么是shell

shell 是一个允许用户输入命令的命令解释程序，它检查并翻译用户键入的命令。shell 会产生一个提示符（一般是$），等待用户键入命令。

命令的执行可能是包含在一个新进程的上下文中（例如 ls），也可能是调用 shell 本身的一些函数，例如调用 print 函数，或者也可能引起错误提示，例如“non existent_command”。

如果你对 shell 环境完全陌生，你可以想象它为覆盖中心物质的外壳，比如花生壳。也可以把 shell 看作是通往市中心你所经过的街坊邻居。这个比喻中，市中心就是操作系统的内核。任何程序，包括 shell 在内，为了最基本的操作，都需要与 UNIX 内核进行交流，比如文件存取。

本书讨论的 shell 是由贝尔实验室的 David Korn 创立的 Korn Shell。我们也会包含一些 ksh93 版本中的特性。

交互式情况下，shell 提示用户输入。输入采用命令字符串的形式。shell 翻译每一个交互

式命令，在交互式命令实际被执行前，译码检查命令语法的正确性并执行其他 shell 级进程。

例如，当你键入 ls -l obrien 命令，shell 翻译这个命令行为如下符号：ls、-l 和 obrien，shell 必须定位命令文件，检查用户是否有操作权限，并在运行 ls 命令前执行其他必要的检查。当 ls 程序运行时，检查-l 和 obrien 并且执行程序，或者在必要时提示错误。

ls 程序在一个相关的进程中运行。所有程序的运行都有其相关的进程，我喜欢把进程看作是漂浮着程序的海洋，之所以这样是为了支持程序的运行。（在后面的章节中你将学到更多有关进程的知识。）

1.2 什么是shell脚本

当好莱坞电影演员拿到剧本阅读时，他们以剧本中对话的先后出现为顺序。同样 shell 脚本是一个预定义的对话序列，其实就是是写满了要执行命令的文件。

1.3 对 #!/bin/ksh的解释

如前所述，shell 脚本无非就是一个存储命令的文件。当执行脚本时，文件中的整套命令被传递给进程解释程序。现在有不止一种 shell 脚本语言，怎样声明你的脚本是 Korn Shell 脚本，并由 ksh 解译，而不是 csh，sh，bash，tcsh 或者其他别的 shell 程序？

在 shell 脚本中，区别是何种脚本语言的方法就是在脚本文件的第一行使用特殊的字符，它位于整个程序的前端。

通常行首有“#”，表明此行为注释行。在 shell 脚本中，注释以“#”号开头。因此，如果“#”出现在一行之首，那么整个这行都被看作是注释，#和行尾之间的任何字符都被认为是注释的一部分而被 shell 忽略掉。注释就是嵌入脚本中的语句，不会被 shell 执行，但它通常提供对脚本的描述，以便于脚本的维护。不过，如果在#号后紧跟着！，即文件第一行的头两个字母是#和！，那么当前运行的 shell 会翻译列在#! 后面的文件说明。事实上，ksh 程序运行时对程序进行分别处理，第一行以外的其余脚本文件被提交给 ksh 程序，如果没有错误发生，这些输入会被翻译和执行。

下面是一个简单的 shell 脚本：

```
#!/bin/ksh
echo "Hello World!"
```

这个脚本将产生如下输出：

```
Hello World!
```

注意：

你系统的 Korn shell 程序可能不在/bin 目录下，如果这样的话，改变脚本的第一行，使脚本与你的 ksh 程序的路径和名字相匹配。在交互式提示符下键入 echo Hello World!结果是一样的。

如果你的文件没有#!/bin/ksh，而从第二行开始，会怎么样呢？

如果当前运行的 Shell 能够识别脚本文件中的命令，那么文件就会被执行，否则，就会提示错误信息并且从脚本中退出。最简单的例子就是要求显示一些文本，任何一种 shell 脚本语言都能识别并执行。

只要脚本的第一行是#!/bin/ksh，就可以确保脚本按照你所要求的执行命令，而不必考虑环境，因为你已经明确的要求使用 Korn shell 翻译脚本文件的其余内容。这就意味着以“#!”开头的这一行是特殊的一行，不同于普通的注释行，而且#! 必须位于脚本文件第一行的开头两个字母。

现在，你可能在执行简单的程序时遇到了一些问题，比如你编辑了一个文件并准备运行它，你可能遇到错误“Permission denied”，这是因为操作必须被赋予权限。本章将会详细讨论有关权限的问题，大体上讲权限决定谁可以执行文件。你必须告诉计算机，你所创建的脚本文件是可执行的。

下面是一个显示输出到屏幕的简单脚本。文件是不可执行的（权限中没有 x），因此试图运行文件以失败告终。然后，例子中使用 chmod 命令赋给所有者(u)和组(g)执行权（ug+x——后面会介绍）。改变许可权限后，脚本可以成功的运行。

```
$ cat hello-k
#! /bin/ksh       # 使用Korn Shell执行文件的剩余部分
print hello
$
$ ls -l hello-k  #  没有执行权限
-rw-r--r--   1 obrien   users         27 Sep 28 17:36 hello-k
$
$ hello-k
ksh: hello-k: cannot execute
$
$ chmod ug+x hello-k   #  添加执行权限
$
$ hello-k              #  添加执行脚本
hello
```

如果试图从交互式 C shell 运行你的 Korn Shell 脚本，会怎样呢？因为脚本以#!/bin/kah 开头，脚本可以在任何环境中运行。

下面的例子通过运行 csh 程序进入了 C shell 交互式环境。注意提示符由$变为%。Korn shell 脚本仍然正确运行，exit 命令从 csh 程序退出返回到 ksh 交互式 shell。

```
$
$ csh                  #  启动C shell
%
% hello-k              #  在C shell中运行hello-K
hello
%
% exit
$
```

接下来的这个例子使用 sed 命令（将在第九章“文件和目录操作”中讨论）创建一个新文件（hello-g），脚本中没有#!，这个例子提供了一个机会可以检验在非 ksh 环境中没有#!/bin/ksh，脚本是否可以成功运行。结果是改变了的脚本在 C shell 中不能运行，因为它试图翻译好像满足 C shell 脚本语法而非 Korn shell 脚本语法的脚本内容。后面例子中使用了 chmod ug+x 命令给了用户或所有者和组执行权限：

```
$ sed 's/^#.*//' hello-k > hello-g
                                    #  创建脚本时第一行没使用#!
$
$ cat hello-g
print hello
$                                   #  文件只有一行
$
$ ls -l hello-g
-rw-r--r--   1 obrien   users        16 Sep 28 17:41 hello_g
$
$ chmod ug+x hello-g                #  添加执行权限
$
$ hello-g                           #  在Korn shell中顺利运行
hello
$
$ csh
%
% hello-g
hello-g: print: not found
%                                   #  在C shell中运行出错
% exit
$
```

1.3.1 文件属性

如果创建了一个文件，比如前面讲到的 echo hello 程序，被赋予的文件执行权限基于用户的 umask 而定。多数系统中缺省的 umask 是 022。这个缺省值可以由系统管理员做变动。通常这个值的改变是在用户的.profile 文件中。比如：

```
umask 002
```

假定你创建了一个临时文件，并存于你的起始目录下，文件名为 david，使用 ls 命令可以列出文件的列表：

```
$ ls -l david
-rw-rw-r--   1 Pitts    users          24 May 13 16:57 david
```

注意：

"$" 是缺省的 Korn shell 提示符，运行 ls 命令时，是不必键入的，你所输入的是 ls -l david。

执行 ls 命令时可以得到九项有关文件（或目录）的信息：

- 文件类型
- 权限
- 硬链接数
- 所有者
- 组
- 大小
- 最后修改月份
- 最后修改日
- 最后修改时间
- 文件名

目前权限、所有者、组和文件名是我们所关心的。

文件的权限是 rw-rw-r--。这一行的第一个字符指文件类型，"-" 意思是普通文件；"d" 指目录文件；其他的列于表 1.1 中。

- 所有者是 pitts
- 组是 users

- 文件名是 david

文件权限分为四部分：

- 文件类型（一个字符）
- 所有者权限（三个字符）
- 相关组的权限（三个字符）
- 所有其他用户的权限（三个字符）

每一部分都可以被提供三个文件权限——读、写、执行。这些权限被称做文件模式，文件模式可以用 chmod 命令设置。

规定权限可以采用两种方法——数字代码和字母代码。使用字母代码这三部分分别是指：

- u 指用户
- g 指组
- o 指其他用户
- a 指所有用户

基本权限是：

- r 指读
- w 指写
- x 指执行

组合 r、w 和 x 就可为三组用户提供完整的权限设置。下面的例子中文件所有者有读、写和执行权限，其他用户只有读权限：

```
$ ls -l test
-rwxr--r--   1 obrien    users          24 May 13 16:57 test
```

命令 ls -l test 告诉计算机检索文件（test）的长(-l)列表(ls)。第二行是命令执行的结果。

因此权限由 10 个字符组成，破折号“-”表示没有赋予用户（user，group，others）权限（read，write，execute），其中 user 用 owner 更合适，但是这样容易引起混淆:无法区分 o—其他用户(others)和 o—所有者（owner），其中第一个字符的含义是不同的。参照表 1.1 查看第一个字符所代表的文件类型。

表 1.1 区别文件类型的特殊字符

字 符	文件类型
	普通文件
b	块特殊文件

（续）

字 符	文件类型
c	字符特殊文件
d	目录
l	符号链
p	命名管道
s	Socket

表示文件类型的字符后是权限的设置：rwx(user)，r--(group)，r--(other)。

注意：

这里需要对读、写和执行权限的实际含义做一些解释。对于文件，有读权限的用户可以浏览文件内容；有写权限的用户可以修改文件；有执行权限的用户可以执行文件。如果要执行的文件是一个脚本文件，用户必须还有读权限才能执行文件。如果文件是二进制文件（通过编译和链接创建）只有执行权限就可以执行文件了。

当一个高级语言（C,COBOL,FORTRAN,Pascal,Ada,C++等等）程序被提交给编译程序时，就会创建一个编译和链接的二进制文件。当程序执行时，编译程序会翻译C语言代码为机器语言，编译程序把它的输出传递给链接程序，它会产生一个二进制可执行文件。

编译和链接的二进制文件的执行速度比shell的执行快4到5倍。相比之下，这种文件不容易编写，执行速度快，脚本文件易写但是执行速度慢。

交互式shell通过检查文件的头两个字节，可以区分执行的是脚本文件还是二进制文件。如果文件有#!，就是执行脚本文件，否则就认为是二进制文件。

1.3.2 目录

目录权限与文件权限一样有读，写，执行三种。但是实质上是不同的。对于目录，读权限是指列出目录中文件的名字，但不允许查看文件的其他属性（例如所有者、组和大小等）。

下面这个例子就是没有执行权限的目录设置，不允许显示文件属性，不过可以显示包含在目录中的文件名。命令ls -ld ob的执行显示了ob目录文件的权限信息，而非包含在目录中的文件。例子如下：

```
$
$ mkdir ob    # 建立一个名为ob的目录
$
$ ls -ld ob
drwxr-xr-x   2 obrien   users       8192 Sep 27 19:00 ob #  注意r和x权限

$
$ cd ob
$
$ cat > tmp1  # 在ob目录下创建一个文件
junk in file
$
$ ls -l       # 可以获取新文件的属性
total 1
-rw-r--r--   1 obrien   users         13 Sep 27 19:01 tmp1
$
$ cd ..
$
$ ls -ld ob   # 列示目录权限
drwxr-xr-x   2 obrien   users       8192 Sep 27 19:01 ob
$
$ chmod ugo-x ob  # 取消可执行权限
$
$ ls -ld ob   # 目录无执行权限
drw-r--r--   2 obrien   users       8192 Sep 27 19:01 ob
$
$ ls -l ob    # 目录无执行权限不显示属性信息
              # execute permission on the directory
ls: ob/tmp1: No permission
total 0
$
$ ls ob       # 可以浏览文件名
tmp1
```

目录可写就是可以改变目录内容，就是说用户可以创建和删除目录中的文件。

下面的例子试图在一个没有可写权限的目录中创建文件，因而失败。添加写权限后，文件创建操作成功实现。cat 命令用来创建一个小文件。要终止由 cat >命令开始的创建操作，需要在一个空行按 Ctrl+D。

```
$ pwd
/usr/users/obrien/ob
$
$ ls -ld              # 注意无写权限
dr-xr--r--   2 obrien   users       8192 Sep 27 19:01 .
```