Mastering UNIX Shell Scripting

精通UNIX Shell脚本编程

[美] Randal K. Michael 著

詹文军 邓 波 等译

WILEY

電子工業出版社
Publishing House of Electronics Industry
http://www.phei.com.cn

精通 UNIX Shell 脚本编程

Mastering UNIX Shell Scripting

[美] Randal K. Michael 著

詹文军 邓 波 等译

電子工業出版社
Publishing House of Electronics Industry
北京 · BEIJING

内容简介

本书详细介绍如何编写shell脚本来解决实际生活中遇到的UNIX问题和任务。本书面向所有的UNIX版本，重点覆盖对象包括AIX、Linux、HP-UX和Solaris操作系统。本书的每一章都以一个经常遇到的UNIX问题作为开始。对于每个问题，都定义了一个明确的目标。在了解了目标和命令语法之后，读者可以根据命令来创建shell脚本。本书的特点是从基础开始，然后在解决方案中加入越来越多的判断逻辑。针对复杂程度不同的各种问题，本书分别用不同的章节加以介绍。

本书面向那些通过命令行方式使用UNIX的用户。在本书中讨论的主题主要针对UNIX专业人员——程序员、系统分析员、系统操作员、系统管理员，以及任何希望在技术支持领域取得进步的人。

Randal K. Michael: Mastering UNIX Shell Scripting. ISBN 0-471-21821-9.

版权贸易合同登记号　图字：01-2002-6452

图书在版编目（CIP）数据

精通UNIX Shell脚本编程 /（美）迈克尔（Michael, R. K.）著；詹文军等译.
北京：电子工业出版社，2005.1
书名原文：Mastering UNIX Shell Scripting
ISBN 7-121-00846-7

Ⅰ.精… Ⅱ.①迈… ②詹… Ⅲ.UNIX操作系统－程序设计 Ⅳ.TP316.81

中国版本图书馆CIP数据核字（2004）第143124号

责任编辑：赵红燕　　特约编辑：赵宏英
印　　刷：北京天竺颖华印刷厂
出版发行：电子工业出版社
　　北京市海淀区万寿路173信箱　邮编：100036
经　　销：各地新华书店
开　　本：787 × 1092　1/16　　印张：32.25　　字数：826千字
印　　次：2005年1月第1次印刷
定　　价：52.00元

凡购买电子工业出版社的图书，如有缺损问题，请向购买书店调换；若书店售缺，请与本社发行部联系。联系电话：（010）68279077。质量投诉请发邮件至zlts@phei.com.cn，盗版侵权举报请发邮件至dbqq@phei.com.cn。

前　言

在UNIX中，有许多方法可以完成某个给定任务。对于一个要解决的问题，我们也许能够以多种方式来得到解决方案。当然，一些解决方案将更有效率、更具有可读性，使用更少的磁盘空间或内存，可能提供、也可能不提供用户反馈，或者是针对结果给出更准确的细节或更高的精度。在本书中，我们打算详细地介绍如何编写一个shell脚本来解决实际的UNIX问题和任务。shell脚本涉及的范围包括使用一个伪随机数生成器来创建伪随机密码，检查UNIX计算机上的完整文件系统，以及发送弹出消息到Windows桌面。编写这些shell脚本所需的细节包括使用良好的风格，并在shell脚本中提供注释来描述每个步骤。其他的细节包括在命令的可读性和概念的可理解性要求不高时，将位于多行的各个命令组合到一条命令语句中，使得脚本更加可读和可维护。我们将了解使用变量和文件来保存数据的好处，并说明从命令行输出中剔除不希望或不需要的数据的方法，同时对数据进行格式化以满足特定的用途。此外，我们还打算说明如何编写函数并在我们的shell脚本中包括它们，并举例说明带函数的shell脚本优于不带函数脚本的地方。

本书面向所有的UNIX版本，但其重点覆盖对象包括AIX、Linux、HP-UX和Solaris操作系统。本书相关的Web站点（www.wiley.com/compbooks/michael）包含了本书中的大部分脚本。在必要时，本书的许多shell脚本针对不同的操作系统都重新进行了编写。其他的一些shell脚本则适用于所有的平台。有些脚本之所以需要重新编写，是因为UNIX不同版本之间的命令语法和输出方式不同，而且有时候差异很大。平台的差异导致了针对不同操作系统的shell脚本的差异。这种差异小的可以是将数据输出到不同的列或是使用不同的命令开关项，大的可以是同时使用多个命令来完成同一任务，以便在不同的UNIX版本上得到类似的输出或结果。

在本书的每一章中，我们首先介绍基本概念，然后再引出一些非常复杂的概念。shell脚本的基本目的是自动完成一些重复而且复杂的功能。这样可以减少击键错误，并按照计划定时执行shell脚本。让系统自己告诉我们它出现了问题，比起我们自己后来才发现问题总是要更好一些。本书可以帮助我们在使用系统时更具有主动性。我们将获得更多的知识来使自己游刃有余地处理更复杂的问题。本书将说明解决实际的示例任务的不同方式。解决一个问题不会只有一种正确的方式，我们将了解以多种方式解决问题的不同层面。我们的目标是成为自信而灵活的问题解决者。当完成本书的学习之后，在给你一个任务时，你将可以按多种方式来解决它，而且解决方案直观明了。

本书和所涉及技术的概述

本书的目的是作为一个学习工具和指导，使读者学习如何通过确定一个明确的目标来编写shell脚本以解决问题。在学习本书时，我们会分七次介绍大部分shell脚本编程技术，每次都是从一个不同的角度来解决不同的问题。我们发现这种学习技巧对于内容的记忆非常有效。

我们鼓励读者从头到尾阅读本书以便从中获得最大的收益。书中的每个脚本都是使用Korn shell编写的，它是UNIX中用于脚本编程的工业标准，尽管有些人可能会对此有所争议。根据UNIX操作系统的不同类型以及版本，UNIX具有不同版本的Korn shell。本书中的shell脚本可以在任何Korn shell版本上运行而不必做任何修改。

本书所涉及的内容从一些简单的任务解决方案到一些相当高级的概念应有尽有，因此无论是系统管理员还是普通用户都可以从中受益。针对复杂程度不同的各种问题，本书分别用不同的章节加以介绍。在本书中给出的shell脚本是完整的shell脚本，这也是本书区别于市面上其他脚本编程图书的特点之一。本书对这些解决方案进行了解释，对于shell脚本的每个部分，其解释程度细致到了作者的思维方式。

本书的组织方式

本书的每一章都以一个在计算机世界中每天会遇到的UNIX问题作为开始。对于每个问题，我们定义了一个明确的目标，并且通过定义正确的命令语法作为shell脚本的开始，以此来解决问题。在具有了目标和命令语法之后，便可以根据命令来创建shell脚本。下一步是过滤命令输出来剔除不需要的数据，或者我们可以决定从输出中提取所需要的数据。如果命令语法对于不同的UNIX操作系统有所差异，则我们将说明正确的语法以得到相同或类似的结果。此时，我们将在shell脚本中创建选项，以便为终端用户在命令行上提供更多的灵活性。

当需要为各种操作系统重新编写shell脚本时，我们将在每一章的结尾处展示一个组合式shell脚本，该脚本将不同风格UNIX操作系统的差异结合在一个脚本中。这个脚本可以在所有的UNIX操作系统上运行。为完成这一最后的步骤，我们可以使用uname命令查询系统来了解当前的UNIX操作系统是什么风格的。通过了解UNIX操作系统的风格，我们可以使用一个简单的case语句来针对不同风格的操作系统执行相应的命令。如果读者对此感到有点陌生，不要着急，我们将在本书中详细解释所有的内容。

本书中的每一章都针对一个不同的实际问题。一些问题非常复杂，而另一些问题则比较简单。有一些章节从多个不同的角度处理问题，而其他一些章节则是让读者自己去解决问题——当然，书中提供了一些提示来作为开始。每一章都解决一个问题，可以作为一个单独的单元来阅读而不必参考书中的其他章。不过，有些内容在一章中进行了详细的讨论，在其他章中也稍有涉及。由于这一情况，我们建议读者从本书开始阅读每一章直到结束，因为本书的确是一个良好的学习体验。

本书的读者对象

本书面向那些通过命令行方式使用UNIX的用户。本书中讨论的主题主要针对UNIX专业人员——程序员、系统分析员、系统操作员、系统管理员，以及任何希望在技术支持领域获得进步的人。初学者能够从本书中获得许多知识，但是有些内容对他们而言可能有点难，因此对于初学者而言，可能需要阅读一本基本的UNIX图书来了解某些问题的答案。在开始本书之前，读者应当对常用的UNIX命令有很好的了解，因为在书中我们不对常用的UNIX命令进行解释。

笔者在学习如何成为一名系统管理员的过程中开始了自己的UNIX职业生涯。当初开始学习时曾希望有一本像这样的书来提供指导。因此，在有过这一段经历后，笔者希望本书能够帮助其他人在职业生涯上有一个良好的开始。在编写本书时笔者考虑到自己当年曾处于读者现在的状态，而且记得自己不得不从man页面学习所有的内容，一次学习一条命令。而使用本书作为一个学习指导，读者可以有一个良好的开端，可以迅速地在UNIX世界中获得成功。

需要的工具

为了从本书汲取最多的知识，需要访问一台UNIX计算机，最好是安装了AIX、HP-UX、Linux或Solaris的计算机。可以在标准的PC硬件上运行Linux和Solaris，PC机相对便宜一些。将默认的shell环境设置为Korn Shell（ksh）是个好主意；Linux上的标准shell是Bourne Again shell（bash），而其他一些操作系统则使用Bourne shell（sh）作为默认shell。可以从命令行输入echo $SHELL来了解自己所用的默认shell。本书中的shell脚本都不需要一个图形终端，但是运行GNOME、CDE、KDE2或X-Windows对此并不妨碍。这样读者可以同时使用多个窗口，并在窗口之间剪切和粘贴代码。

读者还需要一个经常使用的文本编辑器。大多数UNIX操作系统都带有vi编辑器，而且许多UNIX操作系统还带有emacs。记住，编辑器必须是一个以标准的ANSII格式来保存文件的文本编辑器。CDE和其他X编辑器也适合使用。此外，还需要一些时间、耐心以及一个开放、富有创造力的头脑。

另外需要注意的事情是在本书的shell脚本和函数中使用的所有变量都是大写的。这样做是因为能够很快地找到代码中的变量，更容易学习shell脚本。当编写自己的脚本时，请对所有的shell脚本和函数变量使用小写。这样做很重要，原因是操作系统包括应用程序使用大写的环境变量（environment variable）。如果不够仔细，则有可能会使得自己的变量值覆盖了某个关键的系统或应用程序变量而导致系统出现问题。不过，这取决于变量在代码中的可见范围。在使用大写的变量时应当谨慎！

Web站点上的内容

在本书相关的Web站点www.wiley.com/compbooks/michael上，可以找到本书包括的所有shell脚本和大多数函数。这些函数可以很容易地直接剪切和粘贴到读者自己的shell脚本中，以便使脚本编程过程更容易。此外，还可以复制一个shell脚本stub到其他的文件名。这个脚本stub提供了快速编写脚本的一种方式。读者只需填充以下字段：Script Name、Author、Date、Version、Platform、Purpose；当进行修订时，还需要填充Rev List字段，其中用户可以定义变量和函数，而且具有一个“BEGINNING OF MAIN”部分来开始shell脚本的主体部分。

小结

本书可以指导读者成为一个具有创造性、主动性的专业问题解决者。在学完本书后，读者将可以为某个给定的任务找到直观明了的解决方案。本书将帮助读者以逻辑理性的方式解决问题，并提供建立在已知基础上的问题解决技术。对于书中给出的每个问题，读者将了解到如何

采用基本的语法，并将它转换为一个脚本编程解决方案的基础。我们总是从基础开始，然后在解决方案中加入越来越多的逻辑，最后加入其他选项来使终端用户能够更灵活地使用。

谈到终端用户时，我们必须向他们提示处理过程的进展情况。给用户提供一个空白的屏幕是不好的做法，为此我们可以创建进度指示器。读者将了解如何通过创建工具来掌握主动权，这些工具可以监视表明问题开始阶段的特定情况。这也正是了解如何查询系统来使自己在同事当中脱颖而出的机会。

通过本书提供的技术，读者将学习到很多知识。将了解问题的解决方案，如何从自己已知的情况开始并有效地创建一个解决方案，如何使得某个shell脚本能够在其他平台上工作而不必进行进一步的修改，如何做到积极主动，如何编写一个易于维护的 shell 脚本，如何在 shell 脚本中使用大量的注释，如何编写一个易于阅读且遵循逻辑的shell脚本。从根本上说，读者将了解如何成为一个能够为任何问题找到解决方案的问题解决者。

目　录

第1章　脚本编程快速入门和概述

本章首先以一个非常具有针对性的最新课程作为开始，随后对本书中所用技术进行简短说明。这些说明涉及的内容非常广泛，从逐行处理文件的最快方式到UNIX和shell脚本的大小写区分。其中虽然没有包括脚本编程主题的所有方面，但却是一个很好的开端，而且提出了本书所覆盖主题的例子。对于在本章中列出的每个主题，在本书后面的章节中都会有一个详细的说明。

笔者力劝读者完整地学习本书。书中每一章都使用不同方式讨论了不同的主题。之所以这样写，是为了强调绝不会只有一种技术解决UNIX中的某个问题。本书中的所有shell脚本都是解决某个问题的例子。读者在阅读本书时，会发现本书讨论了UNIX中的大多数常见任务（也有一些不太常见任务）。所有的shell脚本都是一个问题思考过程的良好解释，而且我们总是以正确的命令语法作为开始来编写针对某个特定目标的shell脚本。希望读者在阅读本书时能够和笔者编写本书时一样感到惬意。下面让我们开始吧！

1.1　区分大小写

UNIX是区分大小写的，因此，我们编写的shell脚本也是区分大小的。

1.2　UNIX特殊字符

以下的所有字符都具有特殊的含义或功能。如果它们以一种不需要其特殊含义的方式来使用，则必须进行转义（escaped）。为进行转义或去除特殊功能，这些字符必须在前面带有一个反斜杠\，或者是使用单引号括起来。

```
\ ( ; # $ ? & * ( ) [ ] ` ' " +
```

1.3　shell

shell（命令行解释器）是一个用来运行命令、程序和shell脚本的环境。shell具有不同的风格，就像UNIX操作系统具有不同的风格一样。每种风格的shell都具有各自可以识别的命令和功能，本书使用的是Korn shell。

```
Korn Shell        /bin/ksh   OR   /usr/bin/ksh
```

1.4　shell脚本

shell脚本的基本概念是一组命令，这些命令按照执行的顺序列出。一个良好的shell脚本应当具有注释，注释前面以一个#作为开始，用于描述执行步骤。在shell脚本中可能具有条件测试（例如值A大于值B），允许我们处理大量数据的循环，读取和存储数据的文件和变量，还可能包括函数（function）。

我们将在接下来的内容中编写许多脚本，因此从一开始就应该在脑子里有个明确的目标。通过确定明确的目标，我们可以为脚本明确具体的用途，可以确定预期的结果。在解决问题的过程中，还会发现一些技巧、窍门，当然还有思路的转折，即采用一种可能相反的方法来获得相同的结果。各种技术的选择是灵活多变的。

shell 脚本和函数都是解释型的（interpreted）。这意味着它们不进行编译。shell 脚本和函数都是 ASCII 文本，可以由 Korn shell 命令解释器读取。当我们执行一个 shell 脚本或函数时，命令解释器将逐行、逐个循环或逐个测试地读取所有的 ASCII 文本，并从头到尾执行每条语句。

1.5 函数

函数和 shell 脚本的编写方法类似，其不同之处在于，在大多数情况下，函数是在 shell 脚本中定义或编写的，而且在脚本中调用。通过函数的这种方式，我们可以编写一个需要多次使用的代码段，这样就可以在一次编写后重复使用，而不必每次都重新编写该代码，只需调用函数即可。

我们还可以在系统一级定义函数，这样就可以在系统环境中一直使用这个函数，但这是以后要讨论的主题。

1.5.1 函数的形式

函数具有以下形式：

```
function function_name
{

     commands to execute

}
```

或

```
function_name ()
{

     commands to execute

}
```

当我们在脚本中编写函数时，必须记住要在使用它之前对它进行声明或编写。函数部分必须出现在调用函数的命令语句之前。不能使用尚不存在的内容。

1.6 运行 shell 脚本

shell 脚本可以按照以下的方式来执行：

```
ksh  shell_script_name
```

以上命令将创建一个 Korn shell 并在新创建的 Korn shell 环境中执行 shell_script_name。

```
shell_script_name
```

以上命令将执行 shell_script_name（前提条件是在文件上设置了执行位（execution bit），参见 chmod 命令的 man 页面）。脚本将在 shell 脚本的第一行所声明的 shell 中执行。如果在 shell 脚

本的第一行没有声明 shell，则脚本将在默认的 shell 中执行，该默认 shell 是由用户所在系统定义的 shell。在一个非预期的 shell 中执行脚本可能会导致失效或得到无法预料的结果。

1.6.1　在 shell 脚本中声明 shell

注意一定要声明 shell。如果希望对脚本的运行方式以及在哪个 shell 下运行进行完全的控制，则必须在 shell 脚本的第一行对 shell 进行声明。如果没有声明 shell，则脚本将在默认 shell 中执行，默认 shell 是由用户所在的系统定义为执行 shell 脚本的 shell。例如，如果脚本被编写为在 Korn shell ksh 中运行，而默认运行 shell 脚本的 shell 为 C shell csh，则脚本在执行过程中很可能会失效。要声明一个 shell，必须在 shell 脚本的第一行出现表 1.1 所列出的某个声明语句。

表 1.1　声明不同类型的 shell

#!/usr/bin/sh 或 #!/bin/sh	声明一个 Bourne shell
#!/usr/bin/ksh 或 #!/bin/ksh	声明一个 Korn shell
#!/usr/bin/csh 或 #!/bin/csh	声明一个 C shell
#!/usr/bin/bash 或 #!/bin/bash	声明一个 Bourne-Again shell

注意：本书只使用了 Korn shell，即 #!/usr/bin/ksh 或 #!/bin/ksh。

1.7　shell 脚本中的注释和风格

本书强调要在脚本中编写良好的注释。对编写脚本的作者看来直观明了的内容对于其他用户可能是晦涩难懂的。我们必须编写易于阅读而且逻辑清晰的代码。这涉及到编写一个易于阅读和维护的脚本，意味着必须加入大量的注释来描述步骤。在许多情况下，编写 shell 脚本的人并不负责维护脚本。对于维护脚本的人来说，最糟糕的事情莫过于去琢磨其他人所编写、毫无注释、根本不清楚每一步骤都要干什么的代码。修改他人的代码是一件很困难的事情，由于不得不费劲地揣摩脚本作者的思路，有时我们会考虑干脆重新编写整个 shell 脚本。通过编写一个清晰可读的脚本，插入大量的注释来描述作者的思路以及如何使用输入、输出、变量和文件，我们可以避免以上的困境。

为了在命令注释中体现良好的风格，我们需要使注释便于阅读，因此，有时最好是将某个命令注释与命令分隔开，将它们放置在三个不同的行，而不是将所有的命令注释与命令排在同一行。在某些时候，最好是创建一个长的管道。而在另外一些时候，使用管道及确定预期的结果对一个脚本编写新手而言可能会有困难。无论如何，脚本中应当具有逐步描述编写者思路的注释。这样，以后需要读取这些代码的人在看到注释而理解了作者思路时将能够会心地说“嘿，这真是个不错的方法！”。

命令的可读性和逐步骤的注释是编写良好脚本的基础。如果在代码中使用了大量的注释，当在六个月之后重新阅读我们所编写的代码时，我们会感觉轻松很多。好的做法是注释所有的内容，这包括（但不限于）描述变量和文件的用途、描述循环的作用、描述每个测试，还可能包括预期的结果，以及如何操作数据和数据字段的方式。

在每行注释之前应具有一个散列符号 #。

以下给出的脚本 stub（script stub）位于本书相关站点 www.wiley.com/compbooks/michael 上，其名称为 script.stub。它具有编写一个 shell 脚本之前所需要的所有注释。script.stub 文件可以复制到一个新的文件名中。编辑新的文件名，然后在其中编写代码。程序清单 1.1 中列出了 script.stub 文件。

程序清单 1.1 script.stub 脚本启动清单

```
#!/usr/bin/ksh
#
# SCRIPT: NAME_of_SCRIPT
# AUTHOR: AUTHORS_NAME
# DATE:   DATE_of_CREATION
# REV:    1.1.A (Valid are A, B, D, T and P)
#               (For Alpha, Beta, Dev, Test and Production)
#
# PLATFORM: (SPECIFY: AIX, HP-UX, Linux, Solaris
#                     or Not platform dependent)
#
# PURPOSE: Give a clear, and if necessary, long, description of the
#          purpose of the shell script. This will also help you stay
#          focused on the task at hand.
#
# REV LIST:
#        DATE: DATE_of_REVISION
#        BY:   AUTHOR_of_MODIFICATION
#        MODIFICATION: Describe what was modified, new features, etc--
#
#
# set -n   # Uncomment to check your syntax, without execution.
#          # NOTE: Do not forget to put the comment back in or
#          #       the shell script will not execute!
# set -x   # Uncomment to debug this shell script (Korn shell only)
#
###############################################################
############ DEFINE FILES AND VARIABLES HERE ##############
###############################################################



###############################################################
############### DEFINE FUNCTIONS HERE #####################
###############################################################



###############################################################
################ BEGINNING OF MAIN ########################
###############################################################



# End of script
```

在程序清单1.1中显示的shell脚本为我们提供了一个开始编写shell脚本的框架，其中包含了声明变量和文件、创建函数的部分，还包含了编写shell脚本主体的最后一个部分，即BEGINNING OF MAIN。

1.8 控制结构

以下是广泛使用的控制结构。