UNIX实用工具译丛

UNIX Shell 编程工具

Unix Shell Programming Tools

(美) David Medinets 著／孟庆昌 刘振英 胡菱 等译

机械工业出版社
China Machine Press
McGraw-Hill

UNIX 实用工具译丛

UNIX Shell 编程工具

（美）David Medinets 著

孟庆昌 刘振英 胡 菱 等译

孟庆昌 审校

机械工业出版社
China Machine Press

本书能够帮助你掌握 UNIX 操作系统的 Bash、Perl、Tcl Shell 编程。

全书分 12 章，涵盖了有关 Shell 编程技巧和技术的全部内容。在书中，还可找到有关如何创建程序的简单建议，有助于读者早日加入优秀程序员之行列。

本书版权登记号：图字：01－1999－3674

图书在版编目（CIP）数据

UNIX Shell 编程工具／（美）麦迪那斯（Medinets, D.）著；孟庆昌等译. —北京：机械工业出版社，2000.1

（UNIX 实用工具译丛）

ISBN 7－111－07742－3

书名原文：UNIX Shell Programming Tools

Ⅰ.U... Ⅱ.①麦 ...②孟 ... Ⅲ.操作系统，UNIX－程序设计 Ⅳ.TP316.81

中国版本图书馆 CIP 数据核字（1999）第 55436 号

机械工业出版社（北京市西城区百万庄大街22号 邮政编码 100037）

责任编辑：李云静

北京忠信诚胶印厂印刷 · 新华书店北京发行所发行

2000年1月第1版 · 2000年4月第2次印刷

787mm × 1092 mm 1/16 · 23印张

印数：6 001-10 000册

定价：49.00元（附光盘）

译 者 序

UNIX 操作系统自 1969 年在 AT&T Bell 实验室诞生以来，迄今已有 30 年的历史。UNIX以其简洁且功能强大的优点成为当前应用领域最为广泛的主流操作系统之一。实践证明，UNIX 系统一直是当前重点行业和关键事务领域的可靠平台。它作为高端的解决方案，正与其他操作系统协同工作，处理着大大小小的 IT 事务。

在 UNIX 环境下如何进行程序设计？这是系统开发人员、系统管理人员和所有程序设计人员非常关心的问题。UNIX 系统不仅支持 C、C++ 等程序设计语言，而且还提供了丰富的 Shell 程序设计工具，包括 Bash、Perl 和 Tcl 等。很多有经验的程序员感受到，利用 Shell 程序设计语言进行程序开发会大大提高编程效率。尤其对系统管理和系统开发人员来说，掌握 Shell 程序设计语言，就如虎添翼。

Bash 是 Linux 系统上广为采用的 Shell、Perl 和 Tcl 是功能更强的程序设计语言。三者各有所长。为使众多在 UNIX 环境下从事工作的人员尽快学会利用这些工具进行编程的技能，我们组织翻译了这本《UNIX Shell 编程工具》。

该书可分为两大部分：前 8 章为基础篇，后 4 章为利用工具集篇。前 8 章介绍 Shell、Perl 和 Tcl 程序设计的基础知识。其中前 5 章介绍 Bash Shell 的知识，包括变量和运算符、Shell 过程、脚本执行命令以及对 Shell 的控制等。此后的 3 章分别讲解 Perl 和 Tcl 的基础知识以及匹配正文模式的细节。

第二部分探讨随 Bash Shell 而来的实用程序，分别介绍了众多的检测工具（命令），三者在可移植性上的异同、脚本调试技术以及用三种语言定制工具的实例。附录中还给出了从网上获取相关信息的方法。

本书是作者多年经验的结晶。内容丰富，前后连贯，由浅入深，实用性强。特别是书中给出了大量有价值的代码示例，读者可在自己的机器上试着运行它们，这样不仅能加深对知识的理解，而且对提高编程技巧很有帮助。本书适于自学，不同编程背景的人都可以从中受益。

我们在尊重原著的基础上，力求准确、严谨地翻译本书。但由于时间仓促，加上译者水平所限，书中难免有错误或欠妥之处，敬请读者批评指正。

参加本书翻译的人员有孟庆昌、刘振英、胡菱、周国梁、王金柱、马超和董潇潇，进行录入整理工作的有王梅玲、何满祥。全书由孟庆昌统一审校。在此，译者还要感谢中国 UNIX 用户协会（CUUG）给予的支持和帮助。

译 者

1999 年 11 月

于北京信息工程学院

前　言

欢迎你阅读《UNIX Shell 编程工具》。本书与其他 UNIX 书籍有什么差别呢？很简单，它集中在程序设计方面。在书中你不会找到有关最好的文本编辑器或者为什么一种 UNIX 版本比另一种版本更好的论述，也不会找到有关 UNIX 系统管理的信息，而只能找到有关如何创建程序的简单建议，这些程序执行你要它们完成的任务。

在其他方面本书与其他 UNIX 书籍也有差别。其中最大的一点是 Bash、Perl 和 Tcl 如何解释程序设计概念。当然，Bash 一般被认为是 Shell，而 Perl 和 Tcl 被称做程序设计语言。三者都可用来创建程序；且各有所长。能为特定任务挑选特定语言，会使你成为更优秀的程序员。

本书有很多的代码示例，并对示例添加了足够多的注释。当你在一个月或者一年内再翻看本书时，不必阅读正文来回忆示例是干什么的。简单地阅读注释，就足以唤醒你的记忆，回想起其细节。

本书使用对象

本书可供寻求在 UNIX 环境中学习如何进行程序设计的任何人使用。本书并未介绍诸如列目录内容和编辑文件之类的基本技能。

如果你已经熟悉 Perl 或者 Tcl，那么阅读本书就能让你把很多编程技巧转到建立 Bash 程序中来。

本书使用条件

为了有效地利用本书，你需要具备两件东西：运行 Bash 的计算机和像 vi 或者 emacs 一样的文本编辑器。那就够了！你甚至不必有运行 Bash 的本地计算机。如果你有办法远程登录（telnet）进入 ISP 计算机，而且那台计算机使用 Bash，那么也可以使用本书。

如何使用本书

使用本书的方式有很多种。如果你不太熟悉程序设计，那么就从头到尾阅读本书。

经验较多的程序员可以选读，跳到第 5 章，然后读第 8 章～第 12 章。

代码示例

很多读者喜欢手工键入多数示例代码，这可帮助他们每次关注一行代码。如果要这样做，请记住不必拷贝注释！不拷贝注释就使大多数示例精简了许多。

所有代码示例在网上的地址是 http：//www.codebits.com/spt。

在每个示例之后，试着做一下，看看会发生什么情况。修改几行或者加上两行。不必担心会故意引起语法错误。

编程应该是充满乐趣的。如果你在一个题目上受到挫折，就休息一下，与别人聊聊天，

然后再回到那个程序上。

约定

本书采用了以下约定：

- 在 Perl 和 Tcl 程序设计中大小写很重要。对变量和函数名始终要注意大小写。
- // (这种风格用于作者的代码注释，它们不应键入你的脚本中)。

概述

第一部分是“基础篇”，包括本书的前 8 章。这些章节介绍 Shell、Perl 和 Tcl 程序设计的基础知识。第 1 章是“Shell 简介”，它介绍 Shell 和进程，为本书其余部分打下基础。第 2 章是“变量和运算符”，它介绍如何用 Bash 来存放和管理数据。接下去是第 3 章——“过程”，它介绍如何创建可以反反复复使用的小而精的功能模块。在过程之后进入第 4 章——“脚本执行命令”，它提供了脚本一级的信息。在控制脚本之后，也应知道如何控制 Shell 本身——哪一个是脚本运行的环境。第 5 章——“控制 Shell”就弥补了这个需要。

在读过前面 5 章之后，会对 Shell 程序设计的一般性质有了很好的理解。第 6 章和第 7 章分别介绍 Perl 和 Tcl 的基础知识。第 8 章是“模式匹配”，它深入研究如何匹配正文模式的细节。

第二部分是“使用工具集篇”，探讨随 Bash Shell 而来的实用程序。就像人不是孤立的一样，Shell 也不是与世隔绝的。它是随数百个实用程序一起装入的，正等待你使用。第 9 章——“检查工具”，它考察很多较重要的实用程序，并提供某些有关如何使用它们的简短示例。第 10 章是“移植性问题”，考察 Bash、Perl 和 Tcl 彼此有什么差别以及相似之处。第 11 章——“调试概念”，介绍某些调试技术，当你的脚本不能正确工作时，这些技术很有用。最后，第 12 章是“定制工具”，用所有三种语言——Bash、Perl 和 Tcl 展示更长的示例。

第三部分为附录。附录 A——“Internet 资源”，讲述从万维网（World Wide Web）取得有关 Shell 程序设计信息的某些方法。附录 B——“ASCII 码表”，它是 ASCII 代码的列表，以便于你使用。

目　录

第二部分 使用工具集篇

第一部分 基 础 篇

第 1 章 Shell 简 介

本章介绍 Shell 和进程。包括可供使用的不同类型的 Shell，尤其是有关 Bash 的内容。本章还将介绍 Bash、Perl 和 Tcl 脚本，并说明它们的差别。最后，介绍多数 Shell 的共同性质。

首先让我们介绍一些基本知识。UNIX 是一个操作系统，它在 CPU、磁盘驱动器、内存、监视器、键盘和其他硬件部件之间起调节作用。当你看 UNIX 屏幕时，实际上并没有见到“UNIX”，只是见到由该操作系统运行的一个程序，它对键盘进行监控并做出响应。这个程序就称做注册 Shell（login Shell）。注册 Shell 是 UNIX 操作系统的一个用户界面。图 1-1是描述这种情况的示意图。

不必深究 Shell 术语的由来。如果进行猜测，我会说，它是有关你在键盘上所做的一切是如何被监控程序（注册 Shell）包装起来的，这是一个要理解的重要概念。当你在键盘上输入命令时，Shell 程序就解释它们。

例如，你正运行一个显示进程统计数据的程序——ps 程序（见图 1-2）。该程序在 Shell 程序内部运行。假设把 Shell 想象成一个大球，当运行这个计算程序时，这个球就膨胀，把它包容进来（因为从本质上说，计算程序是在 Shell 内部运行）。当程序完成时，这个球就收缩。

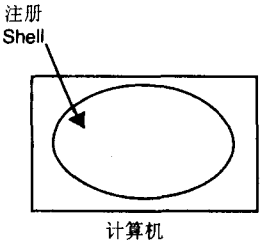

图 1-1 注册 Shell 监控命令行

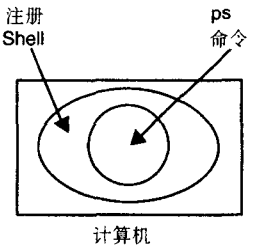

图 1-2 Shell 程序可以运行其他程序

注意 Shell 程序也称做命令解释程序。

ps 程序是作为计算机内的一个程序来执行的。简单地讲，进程就是由计算机运行的单个的命令。每一个进程有自己的内存和存储权限。

你甚至可以在另一个 Shell 程序内部运行一个 Shell 程序。在大球内部的 Shell 程序称做子进程。下面的命令序列告诉你子进程的活动情况：

```
$ echo $ SHLVL
1
```

```
$ bash
$ echo $ SHLVL
2
$ exit
$ echo $ SHLVL
1
```

注意，echo 命令显示的数（对应 Shell 嵌套的层数）为何增加了？这表示所显示的值是子 Shell 显示的值。exit 命令返回到父 Shell。

1.1　存取权限问题

如果你是首次看到程序设计（与简单地运行程序不同），那么可能不熟悉存取权限的概念。存取权限（access permission）用来控制对文件的存取。存取限制基于是文件属主（owner）、文件组用户还是其他用户。对于每一类用户有三种存取方式：读、写和执行。

存取权限可以由 chmod 命令进行修改。

由于在所有的 UNIX 系统上存取权限都是文件系统结构的一部分，并且是 Shell 程序设计的必备部分，所以我们先看两个例子。

```
$ echo " A Test " > test .dat
$ mkdir test _ dir
$ ls -l test .dat test _ dir
total 2
-rw-rw-r--   1 medined medined     7 [date] test.dat
drwxrwxr x 2 medined medined 1024 [date] test _ dir2
$ rm -r test _ dir
$ rm test .dat
```

ls -l 命令执行结果的第一列包含所列出的每个文件的存取权限。如图 1-3 所示，把第一列分为四个部分，即：d、rwx、rwx 和 r x，对它进行解释。

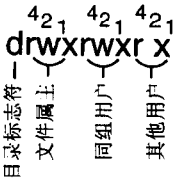

图 1-3　对文件属主和同组用户给予读、写、执行权限，对其他用户给予读和执行权限

注意　对某些特别文件来说，第一个字可以是短划线“-”或字母“d”以外的其他字符，那些文件很可能不被系统管理员使用，本书并未包含它们。

虽然从直观上看它像是计数器，但目录信息存放在文件中。上述第一部分表明该文件是否包含目录信息。如果它包含了目录信息，则第一部分就显示一个“d”；对于普通文件，它

是空白。

下面三个部分都采用相同的简单表示形式，指明文件属主、同组用户或者其他用户对该文件是否拥有读、写或执行的权力。

本章的下一节创建并运行一个 Shell 脚本。其目的主要是说明如何利用 chmod 命令来控制存取权限。对图 1-3 中的数字“421”也予以解释。

1.2 运行 Shell 脚本

本书的第一个 Shell 脚本说明如何显示一行正文。第一步是利用任何一个文本编辑器创建脚本文件，然后更改文件权限，使之有可执行权限。最后可以看到如何运行这种新式的脚本。

脚本（script）文件由一个普通文本文件组成，它包含一系列 Shell 命令。Shell 命令的格式很自由，只有少量的语法规则。在本书的后面部分会根据需要介绍更复杂的规则，当前只介绍基本内容：

- 开头的空格被忽略。因而，脚本中各行采用缩进形式会帮助你进一步理解代码意义。在以下示例中清楚地表明我喜欢用脚本缩进形式。
- 有时命令以分号终止，但始终带有一个换行符。如果你想在下行中接续一个命令，那么在本行末尾要放一个反斜线字符（\）。\ 字符让 Shell 忽略换行符。
- 在字符串之外的空格、制表符和空白行没有关系，但在一条命令或选项之后的一个空格是十分重要的。
- 在字符 # 之后的任何东西都被忽略，除非 # 字符出现在一个字符串内部。这样，就很容易把注释加到脚本中。

提示 字符串是括在双引号中的一系列字符，这些字符可以是字母“A”和数字“3”之类的字符。附录 B“ASCII 表”列出所有可用的字符。

使用任何文本编辑器建立如例 1-1 所示的文件。

注意 如果你不熟悉 UNIX 编辑器，可以按下述步骤利用 vi 编辑器建立 hello 文件：

1）利用命令 vi hello 启动 vi 编辑器。

2）使用字母 i 进入插入方式。

3）键入脚本的第一行。如果你打错了，就使用退格键删除出错字符，然后重新键入字符。

4）使用 Enter 键，启动第二行。

5）键入该脚本的第二行。如果你打错了，就使用退格键抹掉出错字符，然后重新键入字符。

6）使用 Escape 键退出插入方式。

7）使用：wq 命令写文件并退出编辑器。

要注意 vi 命令的用法可使用 man vi 命令。

也有许多 Internet 站点专门从事 vi 编辑器的工作。

例 1-1　hello——第一个 Shell 脚本

```
# ! / bin / bash
# The first line of " Congo " by Michael Crichton
echo " Dawn came to the Congo rain forest."
```

可以用以下命令运行这个脚本：

```
% chmod +x hello
% ./ hello
Dawn came to the Congo rain forest.
```

hello 脚本的第一行告诉 Shell，当运行这个脚本时应使用哪个 Shell 程序。在本例中，它就是文件/bin/bash（在我所用的系统中就是 Bash 程序）。

警告　对#字符的解释是不规则的。通常，#字符用于注释开头——Shell 一般忽略在它之后的任何字符。仅当#字符出现在脚本的第一行上时，它有特殊意义，可以规定由哪个程序执行该脚本。

chmod 命令修改了 hello 文件的存取权限，所以在命令行上简单地输入文件名，Shell 就能执行它。

echo 命令把它的参数送到监视器。实际上它的动作比这个更复杂一点，因为参数被送到称做 STDOUT 的某个目标上，通常是显示器。在第 5 章中会讨论 STDOUT。

1.3　Shell 程序设计

与建立一个记帐或记事板应用程序相比，Shell 程序设计差不多总是用来使任务自动化。Shell 与用户交互的工具非常少，至多可以问用户一个问题和接收一个文本答复。

因为我认为当需要用户交互作用时，应使用性能更全面的语言，所以本书并未包括如何读取、分析输入的内容。

Shell 脚本也用于制作“常规”Shell 命令或者程序。例如，本书中使用最多的一个脚本就非常简单，它所做的全部工作就是在软盘驱动器上运行 mount 命令。

Shell 脚本对以下方面也很有用：

- 核查磁盘的使用情况。
- 维护系统日志。
- 监视用户的活动。
- 大量其他任务。

1.4　Shell 计数

正如计数器程序一样（有许许多多），Shell 的类型也超过一种。事实上，Shell 有一个家族历史！因此，有大量的 Web 网站，它们详细介绍各种 Shell，在此就不必深入研究其历史

了。

http: //www. mit.edu/afs/athena/user/w/c/wchuang/www/unix/Unix/Shell _ history.txt 是带有这种信息的主页之一。

Bourne Shell 和 C Shell 是 Shell 家族的祖先。Bourne Shell 系列在若干方面不同于 C Shell 系列。

为了避免丢失具体 Shell 如何工作的细节，建议从 http: //www.gnu .org 下载 Bash 的自由拷贝（如果你尚未用它），并且使用它。这就保证你使用的 Shell 是现代的、坚固的。

在我看来，使用 Bash 的主要优点是：它是在“随便拷贝（copyleft）”的许可证之下提供的。可以在你所有的计算机上安装 Bash，不管是哪个厂生产的机器。这样就可以有一个一致的编程环境。

警告 如果你想看某些注释，了解为什么 csh 对程序设计来说并不很好，可试一试下面的 URL: http: //www. perl. org/CPAN/doc/FMTEYEWTK/versus/csh. whynot。

1.5 Shell 特性

1.5.1 别名

别名创建简化命令，它们表示完全的命令行。例如，很多人利用别名 ll 来表示 ls -l，或者用 la 表示 ls -a。别名只是与 Shell 程序设计的外在形式有关。它们主要用来减少在命令行上击键的数量。

提示 当我开始使用新的 Web 服务器时，经常创建一个别名，用来显示服务器日志文件最后的 10 行。

alias 命令全部归档在 Bash 的手册页中。

当创建一个 Shell 脚本时，有时你想保证命令是“基本的” Shell 命令，而不是别名（有人会让别名 ls 表示 ls -a)。使用命令的全路径名可以避开别名，就 ls 而论，其全路径名是/bin/ls 。

1.5.2 命令替换

Shell 的一个优秀特性是执行命令替换的能力。利用倒引号（`）表示法可以把一个命令的输出合并到另一个命令中。乍听起来这好像是荒唐的想法，但是随着你逐渐熟悉，就会发现它是强有力的技术。

在第 12 章中有一些示例使用了命令替换，在此只给出一个简单示例。比如你创建了一个名为 ls _ info 的文件，其中只有一行：

```
*.c
```

你可以把 ls _ info 文件的内容合并到 ls 命令行中，如下所示：

```
% ls `cat ls _ info`
one .c two .c
```

Bash 执行的命令实际上是 ls * .c 。cat ls _ info 命令的输出代替了原命令行中 cat ls _ info 这一部分。

你或许要问，为何把它称做强有力的技术？答案是，命令替换允许你由命令来设置参数。假设 ls `cat ls _ info`命令在一个脚本内部，而且用户修改了 ls _ info 文件的内容。当运行该脚本时会出现什么情况？ls 命令就自动利用修改后的 ls _ info 文件的内容来执行。

1.5.3　后台处理

Shell 另一个有用的特性是能够同时运行多个程序或脚本。在这种情况下，由键盘控制的程序称做在前台运行，而其他程序则在后台运行。

当你知道一个程序要花费很长时间才能完成，并且不需要提供输入——它静静地穿过自己向下运行，这种特性迟早会有用。find 命令往往要花费很长时间运行，它就很适合于后台处理。看一下以下示例：

```
% find / -name " hello " -print > find . log &
% date
Wed Apr 15 22 : 43 : 59 EDT 1998
% date
Wed Apr 15 22 : 46 : 23 EDT 1998
[ 1 ] + Exit 1 find / -name " hello " -print > find . log
% cat find . log
/data/tk8.0/library/demos/hello
/home/medined/hello
/usr/lib/tk8.0/demo/hello
```

两个 date 命令显示 find 命令运行时所经历的时间。它们也显示注册 Shell 保留的键盘控制。find 命令会显示出错信息；然而，它不能访问键盘。当后台进程完成时，Shell 就显示一个信息，给出该进程的退出码。由于 find 命令把它的输出送到 find .log 文件（第 5 章将介绍操作符>），因此利用 cat 命令很容易读取所找到文件的列表（见图 1-4）。

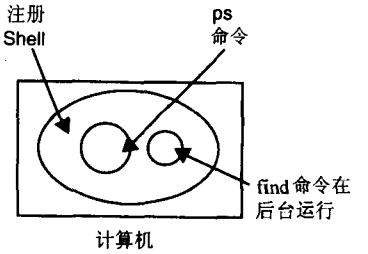

图 1-4　注册 Shell 可以在前台（控制键盘）运行一个命令，
并且运行一个或多个后台命令——如 find 命令

1.5.4　变量

所有 Shell 都能在变量中存放信息。变量是命名的信息部分，详见第 2 章。

1.5.5 管道

UNIX 利用简单命令建造复杂命令的原理或许是最著名的。利用管道可以构造进程。采用几乎无缝连接的方式，管道（pipe）把一个命令的输出连接到另一个命令的输入。图 1-5 示出 ls 命令的输出通过管道传给 sort 命令。

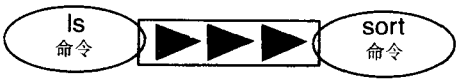

图 1-5 管道把一个命令的输出连到另一个命令的输入

试运行以下命令：

```
% ls | sort -r
```

输出会以反向排序顺序显示文件，每行一个文件：

```
unmount _ floppy
mount _ floppy
hello
find. log
```

1.5.6 重定向

上一节主要讨论了管道概念。重定向（redirection）是与之相关的概念。重定向可以让你改变程序的输入来源和程序的输出地点。

使用 sort 命令来查看重定向。当运行一个命令行上的 sort 命令时，可能有如下情况：

```
% sort
10
9
^D    <-同时按下 Control 和 D 键，结束输入 .
9
10
```

sort 命令从键盘取得输入，并将输出送往监视器。如果有几百个数字要排序，这种办法并不是很好的。但是可以利用重定向让 sort 命令从一个数据文件得到其输入。

```
% sort < unsorted .dat
9
10
```

注意，字符 < 表示对 sort 命令输入的重定向。按照这种方式，unsorted .dat 文件应包含以下的行：

```
10
9
```

字符 > 表示对 sort 命令的输出重定向：

```
% sort < unsorted .dat > sorted .dat
% cat sorted .dat
9
10
```

取得输入的“地方”称做 STDIN，输出送达的“地方”称做 STDOUT。第 5 章将详细介绍 STDIN 和 STDOUT。

图 1-6 示出重定向之前的情况，图 1-7 示出重定向之后的情况。

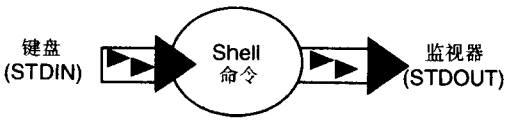

图 1-6　输入和输出的“平常”状况

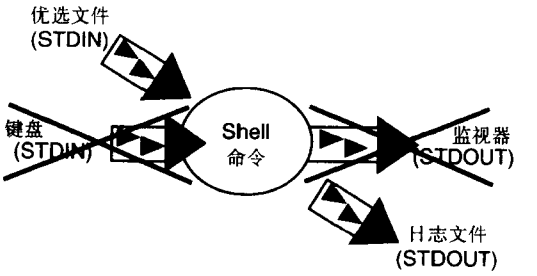

图 1-7　重定向之后的输入和输出

1.5.7　模式匹配

所有的 Shell 都能显示有 txt 扩展名的文件或者显示以字母 a 开头的文件。这种能力就称做“模式匹配”。本书将在第 8 章专门介绍模式匹配问题，所以在此不再给出示例。

1.5.8　特殊字符

你或许已经注意到有很多字符对 Shell 有特殊含义。表 1-1 列出这些特殊字符并对每一个给出简短说明。然而，为了在 Shell 程序设计中真正熟练地应用它们，还必须掌握其中多数字符的用法，这没有捷径。

表 1-1　Shell 中有特殊含义的字符

字　符	说　明
"	双引号用来使 Shell 无法认出空格、制表符和其他大多数特殊字符。这样，"David Medinets" 就被看作是一个值，而不是两个值。同样，"David < Medinets"也被视为一个值，并且忽略其中< 字符的重定向含义
'	单引号使 Shell 无法认出其中所有的特殊字符，除结尾的单引号以外
`	倒引号用于命令替换。在倒引号中间的命令产生的输出将代替原命令中的倒引号字符串。在本章前面的 1.5.2 节和第 12 章中会看到倒引号的使用示例
\	反斜线字符使 Shell 无法认出其后随的字符。换句话说，跟在反斜线之后的字符就失去其特殊含义。作为很平常的例子，利用 David \ Medinets 表示一个值，而不是两个值
;	分号允许你在一行上放多个命令。然而，每行放一个命令其可读性更好