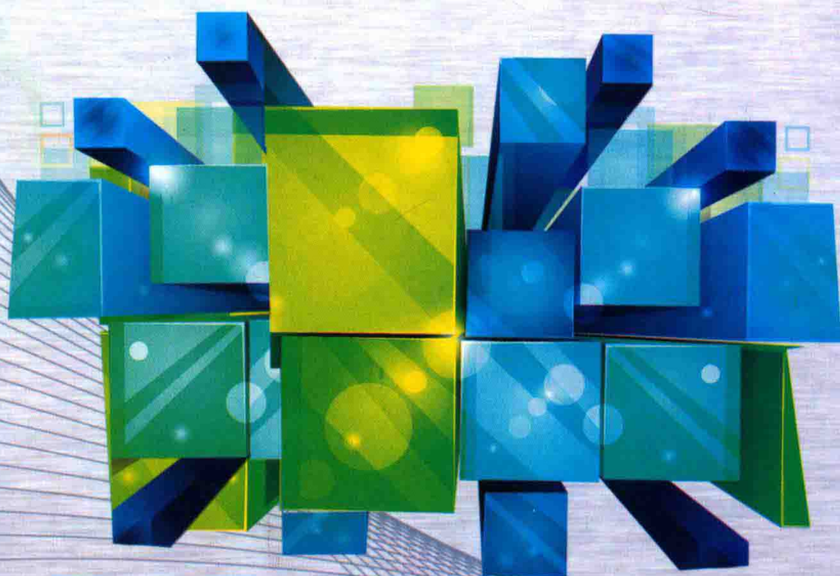
电子信息类"十三五"规划教材

程教育计算机专业实战系列教材

Linux编程实战

Linux Programming in Practice

主　编　王铁军
副主编　杨　昊
参　编　徐　虹



西安电子科技大学出版社
http://www.xduph.com

高等学校电子信息类“十三五”规划教材

CDIO 工程教育计算机专业实战系列教材

Linux 编程实战

主　编　王铁军

副主编　杨　昊

参　编　徐　虹

西安电子科技大学出版社

内 容 简 介

本书共三篇，内容涵盖 Linux 操作系统下 C 语言编程的三大板块：基础编程、内核编程和并行编程。全书具体分 11 章，包括 Linux 下 Shell 脚本编程、Linux 下 C 语言编程基础、多进程编程、内核模块编程、字符设备编程、块设备编程、并行计算与并行程序设计、OpenMP 程序设计基础、OpenMP 程序设计进阶、MPI 程序设计基础、MPI 程序设计进阶。

本书从基础编程开始，循序渐进地让读者逐渐掌握 Linux 下 C 语言编程的各个方面，达到实战训练的目的。全书采用案例教学法编写，书中提供了大量相关实践案例及源代码，以帮助读者增强对所学知识的融会贯通，有效提高其工程实践能力。

本书适合普通高等院校计算机类专业作为 Linux 下 C 语言编程、操作系统实验、嵌入式开发、并行程序设计等课程的教材，同时也可以作为自学 Linux 下 C 语言编程爱好者的参考书。

图书在版编目(CIP)数据

Linux 编程实战 / 王铁军主编. —西安：西安电子科技大学出版社，2017.10

ISBN 978-7-5606-4706-7

Ⅰ. ① L…　Ⅱ. ① 王…　Ⅲ. ① Linux 操作系统—程序设计　Ⅳ. ① TP316.85

中国版本图书馆 CIP 数据核字(2017)第 226890 号

策　　划　李惠萍
责任编辑　杜　萍　雷鸿俊
出版发行　西安电子科技大学出版社(西安市太白南路 2 号)
电　　话　(029)88242885　88201467　　邮　　编　710071
网　　址　www.xduph.com　　电子邮箱　xdupfxb001@163.com
经　　销　新华书店
印刷单位　陕西利达印务有限责任公司
版　　次　2017 年 10 月第 1 版　2017 年 10 月第 1 次印刷
开　　本　787 毫米×1092 毫米　1/16　印　张　10
字　　数　227 千字
印　　数　1～2000 册
定　　价　19.00 元

ISBN 978-7-5606-4706-7 / TP

XDUP 4998001-1

*****如有印装问题可调换*****

中国电子教育学会高教分会推荐
高等学校电子信息类“十三五”规划教材
CDIO 工程教育计算机专业实战系列教材

编审专家委员会名单

主　任　何　嘉

副主任　魏　维

编审人员（排名不分先后）：

方　睿	吴　锡	王铁军	邹茂扬	李莉丽
廖德钦	鄢田云	黄　敏	杨　昊	陈海宁
张　欢	徐　虹	李　庆	余贞侠	叶　斌
卿　静	文　武			

前 言

Linux 是一种自由的、开放源代码的类 UNIX 操作系统。目前，Linux 操作系统已经被广泛地移植到各种硬件平台，成为应用最为广泛、使用人数最多的操作系统。同时，当下诸多新技术，如云计算、物联网、大数据、并行计算、人工智能，其底层几乎无一例外地选用了 Linux 操作系统。因此，掌握 Linux 操作系统的基本操作并能够在其上进行编程开发，已经成为计算机类专业学生以及相关方向科研人员必备的技能之一。

本书旨在通过循序渐进的方式，向读者介绍 Linux 操作系统下的 C 语言编程。本书由三篇组成，内容涵盖 Linux 操作系统下 C 语言编程的三大板块：基础编程、内核编程和并行编程。本书结构规范，系统性强，实例丰富，理论与实践相结合。作者采用案例教学法编写，书中提供了大量相关实践案例及其源代码，以帮助读者增强对所学知识的融会贯通，有效提高其工程实践能力。在阅读本书之前，要求读者具有最基本的 Linux 操作系统使用能力和 C 语言编程基础。

本书具有如下特点：

➢ 实战教学。本书在介绍基础知识的基础上，提供了丰富的案例、源代码和参考结果，方便读者在实战中学习。

➢ 通俗易懂。本书在编写过程中充分考虑到各层次的读者水平，以浅显的语言描述了相对深奥的计算机专业知识，通俗易懂，适合各层次学生和专业人士选用。

➢ 循序渐进。本书从三个方面由浅入深地介绍 Linux 下的 C 语言编程，读者既可以从第 1 章开始阅读学习，也可以根据自己的实际情况从不同篇章入手开始阅读学习，同时也可以将本书作为参考书进行查阅。

➢ 启发式教学。在大部分章节后面，作者根据书中所介绍的内容，给读者提供了一些习题，从这些习题入手，读者可以对书中知识点进行拓展学习，提高学习的深度。

本书由王铁军、杨昊、徐虹和郭龚编写，其中，王铁军负责编写第 2 章的

试读结束：需要全本请在线购买： www.ertongbook.com

部分内容、第4章的部分内容以及第5章、第6章，并负责全书的组织和统稿；杨昊负责编写第7～11章的内容；徐虹负责编写第3章、第4章的部分内容；郭奕负责编写第1章、第2章的部分内容，并负责全书审稿工作。特别感谢西安电子科技大学出版社李惠萍编辑对本书编写所提出的宝贵意见，使得本书得以不断改进和完善。

按照编写目标，编者进行了许多思考和努力。由于编者水平有限，书中难免仍有疏漏和不妥之处，恳请读者批评指正，以便我们不断改进。

作者联系信箱 tjw@cuit.edu.cn。

编　者
2017年6月

目　　录

上篇　Linux 编程实战之基础编程实战

第 1 章　Linux 下 Shell 脚本编程 2

1.1　Linux Shell 简介 2

1.2　Bash 编程基础 4

1.2.1　术语定义 4

1.2.2　环境变量 4

1.2.3　命令替换 6

1.2.4　${ }变量替换 6

1.2.5　数组 array 8

1.2.6　if 语句 8

1.2.7　位置参数 9

1.2.8　条件表达式 10

1.2.9　Bash 命令 12

1.2.10　命令组合 13

1.2.11　循环结构 13

1.2.12　算术运算 16

1.2.13　case 语句 16

1.2.14　函数与名称空间 17

1.3　Bash 编程实例 18

1.3.1　#!符号 18

1.3.2　赋可执行权限 19

1.3.3　脚本实例 19

习题 20

第 2 章　Linux 下 C 语言编程基础 21

2.1　C 语言的编译和执行 21

2.1.1　预处理阶段 22

2.1.2　编译阶段 22

2.1.3　汇编阶段 23

2.1.4　链接阶段 23

2.2　GCC 及主要运行参数介绍 24

2.2.1　预处理 25

2.2.2　编译 25

2.2.3 汇编 27
2.2.4 链接 29
2.2.5 其他参数 30
2.3 Makefile 文件语法及示例 30
2.3.1 概述 30
2.3.2 目标 31
2.3.3 前置条件 32
2.3.4 命令 32
2.3.5 Makefile 举例 33
2.4 调试及 gdb 的使用 34
2.4.1 启动 gdb 34
2.4.2 gdb 交互命令 35
习题 37

第 3 章 多进程编程 38
3.1 进程管理基础 38
3.1.1 进程在 Linux 内核中的表示 38
3.1.2 进程在 Linux 内存中的表示 39
3.2 进程间通信相关函数简介 40
3.3 进程间通信编程实例 44
3.3.1 信号 44
3.3.2 管道 47
习题 50

中篇 Linux 编程实战之内核编程实战

第 4 章 内核模块编程 52
4.1 内核模块简介 52
4.2 内核模块管理及相关函数 53
4.2.1 模块的组织结构 53
4.2.2 模块的加载 54
4.2.3 模块的卸载 54
4.3 Linux 2.6 版本内核模块的编译 55
4.4 内核模块编程示例 56
4.4.1 模块程序设计思想 56
4.4.2 模块程序结构分析 57
4.4.3 Makefile 文件的设计 62
4.4.4 程序执行过程 63
4.4.5 程序执行结果分析 64

习题 65

第 5 章　字符设备编程 66
5.1　字符设备简介 66
5.2　字符设备管理及相关函数 66
5.2.1　设备编号 66
5.2.2　处理 dev_t 类型 67
5.2.3　分配和释放设备编号 67
5.3　字符设备编程实践 68
5.3.1　入口函数流程图 68
5.3.2　字符设备的结构 71
5.3.3　字符设备驱动程序入口点 71
5.3.4　字符设备驱动程序的安装 73
5.3.5　测试程序 74
习题 75

第 6 章　块设备编程 76
6.1　块设备简介 76
6.2　块设备管理及相关函数 77
6.2.1　块设备的表示 77
6.2.2　块设备的基本入口点 79
6.2.3　自旋锁 80
6.2.4　块设备的注册 81
6.3　块设备编程实践 82
6.3.1　块设备的结构 82
6.3.2　块设备的注册 83
6.3.3　块设备的操作 85
6.3.4　请求处理 87
6.3.5　编译并安装设备 90
6.3.6　测试块设备 91
习题 94

下篇　Linux 编程实战之并行编程实战

第 7 章　并行计算与并行程序设计 96
7.1　并行计算简介 96
7.2　串行程序与并行程序 96
7.3　并行程序设计简介 97
7.4　并行计算前沿技术简介 98

第 8 章　OpenMP 程序设计基础 100
8.1　OpenMP 概述 100
8.2　一个基于 OpenMP 的并行程序 102
8.3　OenpMP 兼容性检查 105
8.4　OpenMP 最常用的三个库函数简介 106
8.5　parallel 语句 107
8.6　critical 语句 110
8.7　变量作用域 111
8.8　reduction 语句 113
8.9　parallel for 语句 114
第 9 章　OpenMP 程序设计进阶 116
9.1　single 和 master 语句 116
9.2　barrier 语句 117
9.3　atomic、锁和 critical 进阶 117
9.4　schedule 子句 120
9.5　循环依赖 120
习题 123
第 10 章　MPI 程序设计基础 126
10.1　分布式内存模型 126
10.2　MPI 简介 126
10.3　环境安装 126
10.4　环境测试 127
10.5　典型的 MPI 程序 128
10.6　MPI 消息 131
10.7　MPI_ANY_SOURCE 和 MPI_ANY_TAG 135
10.8　消息状态 138
第 11 章　MPI 程序设计进阶 140
11.1　集合通信 140
11.2　广播 140
11.3　归约 141
11.4　全局归约 142
11.5　散射 143
11.6　聚集 145
11.7　全局聚集 147
习题 148

上篇

Linux 编程实战之基础编程实战

Linux 操作系统经过二十多年的发展，已逐步形成了一个非常完善的生态系统。为了更加高效地使用和管理 Linux 操作系统，来自全世界的开发者不断开发出多个版本的 Shell。使用 Shell 编写脚本程序，可以帮助 Linux 操作系统的管理员和使用者高效开发出功能强大的程序，实现特定的功能。能够理解和编写 Shell 脚本，已经成为熟练掌握 Linux 操作系统、进行操作系统运维的一个重要指标。此外，作为服务器端或嵌入式方面的开发人员，能否在 Linux 操作系统上进行 C 语言的开发和调试，同样是衡量一个开发人员的重要依据。

本篇由三章组成，分别从 Shell 脚本编程、C 语言编程和多进程编程三个方面向读者介绍 Linux 操作系统下的基础编程方法。其中，第 1 章首先介绍了 Linux 操作系统下 Shell 的常识性概念，重点介绍了 Bash 的语法，最后通过实例介绍如何在 Linux 操作系统下编写一个简单的脚本程序；第 2 章主要介绍了 Linux 操作系统下进行 C 语言编程所需要掌握的基本理论知识和所使用的工具，通过实例讲解 C 语言编译执行的过程、gcc 编译器的使用、Makefile 文件的编写以及调试工具 gdb 的使用方法；第 3 章首先介绍了 Linux 操作系统中进程的表示，其次给出了进程间通信的相关函数，最后通过信号和管道两个实例，向读者介绍了多进程编程的方法。

学习本篇内容之前，读者应该掌握基本的 Linux 命令和 C 语言语法。通过本篇内容的学习，读者可以对 Linux 操作系统下 Shell 编程和 C 语言编程具有一定的认识，能够编写出简单的 Shell 脚本。

试读结束：需要全本请在线购买： www.ertongbook.com

第 1 章　Linux 下 Shell 脚本编程

1.1　Linux Shell 简介

Shell 是操作系统提供的用户界面之一，如图 1-1 所示，它是用户与 Linux 操作系统内核之间进行交互操作的一种接口。除了 Shell 之外，用户还可以通过 X Window 或其他应用程序与操作系统内核交互，实现对计算机硬件的控制。

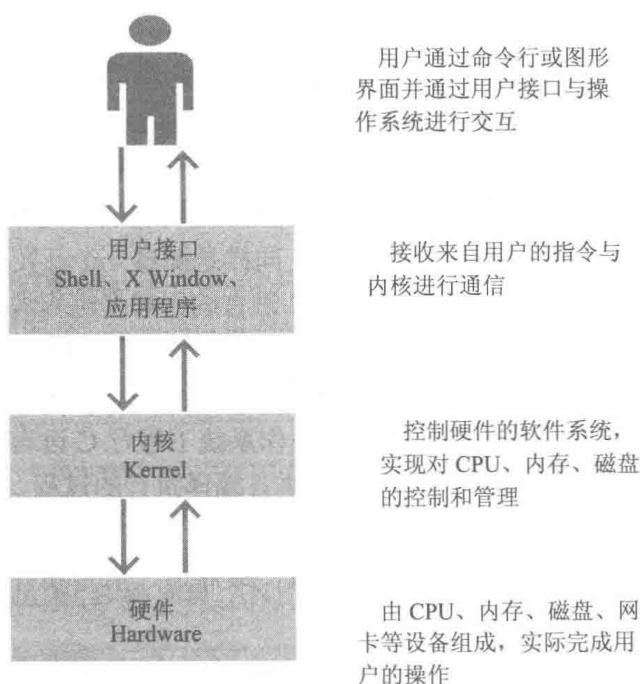


图 1-1　用户与操作系统相关性示意图

虽然有多种方式可以实现与 Linux 操作系统内核的交互，但是有以下几点原因使得我们需要使用 Shell：

(1) Linux 操作系统存在很多不同的发行版本，包括 Redhat、Ubuntu、CentOS、Slackware、Debian、Fedora、openSUSE、红旗、中标麒麟等。不同的发行版本间都存在或多或少的差异，其提供的 X Window 和应用程序集合都略有不同，给用户的使用和管理带来障碍。而所有的发行版本均提供了标准的 Shell 接口，可以实现轻松切换。

(2) 如果需要远程使用和管理 Linux 系统，特别是 Linux 服务器，使用 Shell 是最为便捷的一种途径。而且，通过 Shell 远程访问 Linux 系统，可以降低对网络带宽和延迟的需求。

(3) 如果需要让 Linux 系统自动或定期实现某些功能，Shell 脚本编程是 Linux 操作系

统管理的一大利器。此外，Shell 脚本编程还可以实现诸多复杂的功能，相对于 C、Java 等语言的编程而言，Shell 编程代码编写效率高。

目前，有多种 Shell 环境可供用户选择，图 1-2 给出了 1977 年以来出现在 Linux 操作系统中 Shell 环境的主要序列。

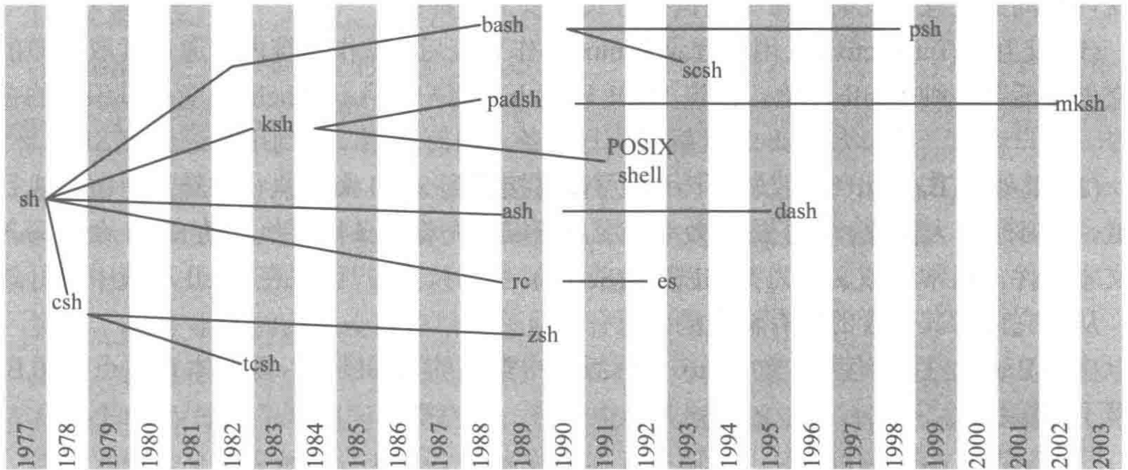


图 1-2　1977 年以来出现的 Linux Shell 环境

以下是 CentOS 发行版本自带的 Shell 环境：

(1) /bin/sh：已经被 /bin/bash 所取代。

(2) /bin/bash：是目前 Linux 操作系统发行版本中默认的 Shell 环境。

(3) /bin/ksh：Kornshell 是由 AT&T 贝尔实验室发展而来的，兼容 Bash。

(4) /bin/tcsh：整合 C Shell，提供更多的功能。

(5) /bin/csh：即 C Shell，已经被/bin/tcsh 所取代。

(6) /bin/zsh：基于 ksh 发展而来的，提供功能更强大的 Shell 环境。

一个 Shell 的系统架构如图 1-3 所示。在用户空间，Shell 环境接受用户的输入，通过对输入内容(一个字符串，包括命令及参数)进行词法分析和解析，判断用户输入的命令是否符合规定。如果没有问题则对命令进行扩展，最后执行用户输入的命令；否则输出错误信息反馈给用户。命令在执行过程中通过系统调用实现与内核的交互，完成相应功能。

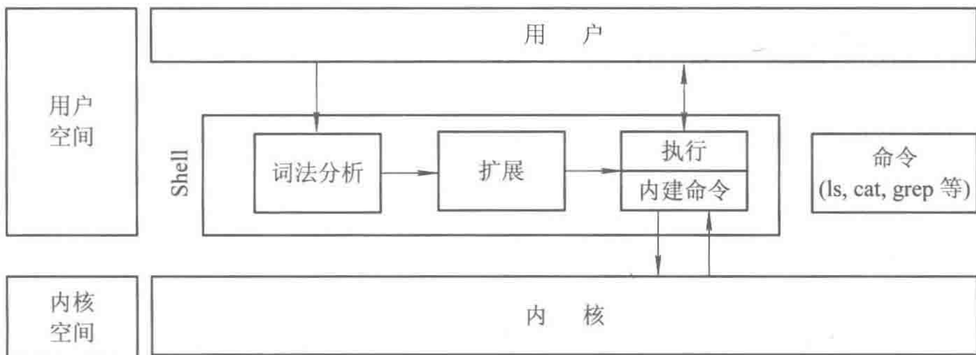


图 1-3　Shell 的系统架构

在 Shell 环境下，用户可以输入并执行的命令(Command)分为内建命令(Built-ins)和外部命令。常见的内建命令包括 cd、source、true、false、continue、break、exit、logout 等，这

些内建命令集成在某一个 Shell 环境中，不同的 Shell 环境包含不同的内建命令。外部命令则是以可执行文件的形式存在于文件系统中，如在/bin 目录下存在 pwd、ls、date、mv 等，这些外部命令独立于 Shell 环境单独存在。

在 Shell 环境下，可以通过执行 Shell 命令(包括内建命令和外部命令)实现用户与操作系统内核间的交互。此时，命令的执行方式有以下两种：

(1) 交互式(Interactive)：用户登录 Linux 操作系统后，操作系统内核通过读取用户的配置文件激活某一种 Shell 环境。在该 Shell 环境下，用户输入该 Shell 环境可以识别的合法命令。用户输入一条命令，Shell 就解释执行一条，周而复始，直到用户退出该 Shell 环境。

(2) 批处理(Batch)：交互方式下用户可以直接看到 Shell 命令执行的结果，但是执行效率低，且必须有人值守。为了提高效率，用户可以事先将多条需要执行的 Shell 命令写入一个文本文件，通常称该文件为 Shell 脚本(Script)。此时，用户可以在 Shell 环境中运行该脚本，从而完成对其包含的所有命令的执行。

由于Bash是目前绝大多数Linux发行版本中默认的Shell环境，因此本章重点介绍Bash环境下的编程。

1.2 Bash 编程基础

1.2.1 术语定义

- 空白符：一个空格或者制表符。
- 内建命令：在 Bash 内部集成(即安装了 Bash 环境就自带的命令)，而不是文件系统中由某个可执行文件实现的一些命令。
- 控制运算符：实现控制功能的一些符号，包括换行符“\n”以及“||”、“&&”、“&”、“;”、“;;”、“|”、“|&”、“(”或“)”。
- 返回值(退出状态)：命令返回给调用者的一个值。因为这个值(二进制)不得超过八位，所以其最大值是 255。在 Linux 操作系统中，如果一个命令成功执行，则结果返回值为 0；否则返回非 0 值。
- 保留字：保留字是计算机编程语言中的一个术语，指在高级编程语言本身已经定义过的字(字符或字符串)，编程人员不能再将这些字作为变量名或过程名使用。保留字包括：!、case、coproc、do、done、elif、else、esac、fi、for、function、if、in、select、then、until、while、{、}、time、[[、]]。

1.2.2 环境变量

在几乎所有的 Shell 环境中都支持环境变量。环境变量被命令、Shell 脚本和可执行程序使用。用户通过定义或修改环境变量的值可对操作系统、应用程序、脚本或命令的行为进行干预或改变。

环境变量根据其作用域可分为系统级环境变量和用户级环境变量。其中，系统级环境

变量可被系统中所有用户访问和共享；用户级环境变量只在定义该环境变量的用户空间使用。用户级环境变量的优先级高于系统级环境变量，即如果用户定义了与系统级相同名称的环境变量，则该用户空间的应用程序、脚本和命令只会访问用户级环境变量的值，而此时其他用户依然访问的是系统级环境变量。

1. 变量定义

在 Bash 中定义环境变量的标准方法如下：

```
$ myvar="This is my environment variable!"
```

以上命令定义了一个名为“myvar”的环境变量，该环境变量的值为字符串 "This is my environment variable!"。注：“myvar”左侧的美元符号“$”是命令提示符，由 Shell 环境自动输出，用户不必自行键入。定义环境变量有两点注意事项：

第一，在等号(=)的两边没有空格，任何空格将导致错误；

第二，虽然在定义一个单词(中间没有空白符)时可以省略引号，但是当定义的环境变量值有多个单词(即多个单词间有空白符，包含空格或制表键)时，双引号是必需的。

在脚本中定义变量的语法和定义环境变量的方法相同，只不过两者的作用域不同。默认情况下，在脚本中定义的变量只在脚本执行过程中有效。如果希望退出脚本后依然有效，需要使用如下语法将脚本中定义的变量导出为环境变量：

```
export JAVA_HOME = "/usr/local/java"
```

或者

```
JAVA_HOME = "/usr/local/java"
export JAVA_HOME
```

2. 变量使用

可以用如下方法使用定义好的环境变量：

```
$ echo $myvar
This is my environment variable!
```

echo 是 Linux 操作系统下的一个命令，用于输出指定的字符串。通过在环境变量 myvar 的前面加上一个 $ 符号，可以使 Bash 用 myvar 的值“This is my environment variable!”替换它，这在 Bash 术语中叫做“变量扩展”。

定义变量时，通常情况下双引号和单引号没有区分。但是在变量扩展时，如果定义的变量包含对其他变量的引用或特殊符号，则两者有所区别。例如：

```
$ a = "this is a string."
$ b = "$a is a var."
$ echo $b
this is a string. is a var.
```

此时，在输出变量 b 时，会用变量 a 的值进行变量扩展。但是，如果在定义变量 c 时使用单引号，如：

```
$ c = '$a is a var.'
$ echo $c
$a is a var.
```

则此时不进行变量替换，$a 被作为一个字符串输出。

但是，不能将变量引用与其他字符串进行拼接使用，例如：

```
$ echo foo$myvarbar
foo
```

我们希望会显示“fooThis is my environment variable! bar”，但却不是这样。此时，Bash 变量扩展陷入了困惑，因为它无法识别要扩展的变量是 $m、$my、$myvar 还是 $myvarbar。为了解决这一问题，可以通过在变量名称外侧加入一对花括号进行解决，例如：

```
$ echo foo${myvar}bar
fooThis is my environment variable!bar
```

即：当变量两侧没有用空白符(空格或制表键)与周围文本分开时，需要使用更明确的花括号形式。

1.2.3 命令替换

命令替换可获取一个命令执行的输出结果。例如，将一个命令执行的输出结果赋值给一个变量，如下所示：

```
$ MYDIR = `dirname /usr/local/share/doc/foo/foo.txt`
$ echo $MYDIR
/usr/local/share/doc/foo
```

命令替换的语法是将要执行的命令以反引号(`)括起。注意：不是单引号，而是键盘中通常位于 Tab 键上的反引号。此外，还可以用 Bash 备用命令替换语法来做同样的事，如下所示：

```
$ MYDIR = $(dirname /usr/local/share/doc/foo/foo.txt)
$ echo $MYDIR
/usr/local/share/doc/foo
```

使用 $( )进行命令替换，可以方便查看代码的用户(因为在某些字体下，单引号和反引号很难区分)，同时还可以方便进行任意深度的命令替换嵌套。

1.2.4 ${ }变量替换

${ }用来作变量替换。一般情况下，$var 与 ${var}并无区别，但是用 ${ } 会比较精确地界定变量名称的范围，例如：

```
$ A = B
$ echo $AB
```

原本是打算先将$A 的结果替换出来，然后再补一个字母 B 于其后，但在命令行上却不会输出任何内容(即此时 Bash 只会认为需要输出变量 AB 的值，但是因为没有定义变量 AB，所以输出为空)。若此时通过 ${ }进行变量替换，就不会出现上述问题。例如：

```
$ echo ${A}B
BB
```

${ } 在进行变量替换的同时，还可以支持一些特殊功能。例如，我们定义了一个变量 file 如下：

```
file = /dir1/dir2/dir3/my.file.txt
```

1. 变量内容截取[①]

变量内容截取说明见表 1-1。

表 1-1　变量内容截取

变　量	取　值	说　　明
${file#*/}	dir1/dir2/dir3/my.file.txt	拿掉第一个“/”及其左边的字符串
${file##*/}	my.file.txt	拿掉最后一个“/”及其左边的字符串
${file#*.}	file.txt	拿掉第一个“.”及其左边的字符串
${file##*.}	txt	拿掉最后一个“.”及其左边的字符串
${file%/*}	/dir1/dir2/dir3	拿掉最后一个“/”及其右边的字符串
${file%%/*}	(空值)	拿掉第一个“/”及其右边的字符串
${file%.*}	/dir1/dir2/dir3/my.file	拿掉最后一个“.”及其右边的字符串
${file%%.*}	/dir1/dir2/dir3/my	拿掉第一个“.”及其右边的字符串

2. 变量长度截取

变量长度截取说明见表 1-2。字符从左到右进行编号，编号从 0 开始。

表 1-2　变量长度截取

变　量	值	说　　明
${file:0:5}	/dir1	提取最左边的 5 个字节
${file:5:5}	/dir2	提取第 5 个字节右边的连续 5 个字节

3. 变量值的字符串替换

变量值的字符串替换说明见表 1-3。

表 1-3　变量值的字符串替换

变　量	值	说　　明
${file/dir/path}	/path1/dir2/dir3/my.file.txt	将第一个 dir 替换为 path
${file//dir/path}	/path1/path2/path3/my.file.txt	将全部 dir 替换为 path

---

① 记忆方法：# 是去掉左边(在键盘上 # 在 $ 之左边)；% 是去掉右边(在键盘上 % 在 $ 之右边)；单一符号是最小匹配；两个符号是最大匹配。

试读结束：需要全本请在线购买： www.ertongbook.com