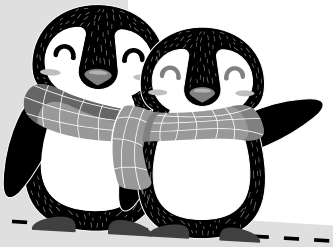

Linux
典藏大系

Linux Shell

命令行及脚本编程实例详解

（第2版）

刘艳涛◎编著

清華大學出版社
北 京

内 容 简 介

本书是获得大量读者好评的“Linux 典藏大系”中的经典畅销书《Linux Shell 命令行及脚本编程实例详解》的第 2 版。**本书第 1 版累计 13 次印刷，销量超过 2 万册，被 ChinaUnix 技术社区大力推荐**。本书理论结合实践，全面、系统地介绍 Linux Shell（Bash）脚本编程的语法、命令和技巧等内容。本书偏重于实践，在讲解理论知识时结合大量典型实例让读者了解理论知识在实际环境中的应用，并对易混淆和较难理解的知识点做了重点分析，以加深读者对知识的理解。**本书提供教学视频、实例源程序、思维导图、教学 PPT 和习题参考答案等超值配套资源，以帮助读者高效、直观地学习**。

本书共 15 章，分为 2 篇。第 1 篇“Linux Shell 基础知识与命令”，主要内容包括 Linux 和 Linux Shell 简介、初识 Linux Shell、常用的 Shell（Bash）命令、Shell 命令进阶；第 2 篇“Shell 脚本编程”，主要内容包括 Shell 编程基础、Shell 的条件执行、Bash 循环、Shell 函数、正则表达式、脚本输入处理、Shell 重定向、管道和过滤器、捕获、sed 和 awk、其他 Linux Shell 概述。

本书非常适合初次接触 Linux Shell 命令行和脚本编程的入门读者阅读，也适合有一定基础而想进一步提升的进阶读者阅读，还适合作为高等院校和 Linux 培训机构的教材。对于基于 Linux 平台的开发人员而言，本书还是一本不可多得的案头查询手册。

图书在版编目（CIP）数据

Linux Shell 命令行及脚本编程实例详解 / 刘艳涛编著. —2 版. —北京：清华大学出版社，2024.4
（Linux 典藏大系）
ISBN 978-7-302-66019-4

Ⅰ. ①L… Ⅱ. ①刘… Ⅲ. ①Linux 操作系统－程序设计 Ⅳ. ①TP316.89

中国国家版本馆 CIP 数据核字（2024）第 070104 号

责任编辑：王中英
封面设计：欧振旭
责任校对：胡伟民
责任印制：杨 艳

出版发行：清华大学出版社
网 址：https://www.tup.com.cn，https://www.wqxuetang.com
地 址：北京清华大学学研大厦 A 座 邮 编：100084
社 总 机：010-83470000 邮 购：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 装 者：北京联兴盛业印刷股份有限公司
经 销：全国新华书店
开 本：185mm×260mm **印 张**：24.75 **字 数**：625 千字
版 次：2010 年 1 月第 1 版 2024 年 4 月第 2 版 **印 次**：2024 年 4 月第 1 次印刷
定 价：99.80 元

产品编号：101195-01

前言

截至 2022 年，全球云服务市场规模达到 30 000 亿元。其中，知名的云服务提供商包括亚马逊的 AWS、微软的 Azure、谷歌的云服务和阿里巴巴的云服务等。这些云服务广泛地使用 Linux 操作系统，这需要人们在 Linux 服务器上进行大量的数据处理和管理，以及应用部署和监测等工作。

为了实现批量、自动化地完成工作，人们必须使用命令行和 Shell 脚本。在 Linux 系统中，技术人员通常基于命令行完成大量的管理和配置任务，并通过 Shell 脚本将一些重复和需要定期执行的任务自动完成，通过短短的几行脚本就能自动将手头的大部分工作完成，从而节省大量的时间。另外，理解 Shell 脚本也可以让技术人员更全面地了解操作系统。

Shell 脚本还可以和很多外部命令行工具结合起来完成信息查询、文本处理、任务定时自动化和监测系统等工作。当然，这些便利在带来效率提升的同时也隐藏着不小的风险。例如，稍不留神可能就会将整个根目录全部毁掉，或者错误地处理重要的配置文件。这时，了解 Linux 命令行和 Shell 脚本的相关细节，以及 Linux 的使用规范就显得格外重要。

本书是获得大量读者好评的“Linux 典藏大系”中的经典畅销书《Linux Shell 命令行及脚本编程实例详解》的第 2 版。截至本书完稿时，本书第 1 版累计 13 次印刷，销量超过 2 万册。本书结合大量实例全面、系统地介绍 Linux Shell（Bash）的命令、语法和使用技巧等知识，面向系统管理员、基于 Linux 系统的软件开发和测试人员，以及所有想高效使用 Linux 系统的爱好者。

关于“Linux 典藏大系”

“Linux 典藏大系”是专门为 Linux 技术爱好者推出的系列图书，涵盖 Linux 技术的方方面面，可以满足不同层次和各个领域的读者学习 Linux 的需求。该系列图书自 2010 年 1 月陆续出版，上市后深受广大读者的好评。2014 年 1 月，创作者对该系列图书进行了全面改版并增加了新品种。新版图书一上市就大受欢迎，各分册长期位居 Linux 图书销售排行榜前列。截至 2023 年 10 月底，该系列图书累计印数超过 30 万册。可以说，“Linux 典藏大系”是图书市场上的明星品牌，该系列中的一些图书多次被评为清华大学出版社“年度畅销书”，还曾获得“51CTO 读书频道”颁发的“最受读者喜爱的原创 IT 技术图书奖”，另有部分图书的中文繁体字版在中国台湾出版发行。该系列图书的出版得到了国内 Linux 知名技术社区 ChinaUnix（简称 CU）的大力支持和帮助，读者与 CU 社区中的 Linux 技术爱好者进行了广泛的交流，取得了良好的学习效果。另外，该系列图书还被国内上百所高校和培训机构选为教材，得到了广大师生的一致好评。

关于第 2 版

随着技术的发展，本书第 1 版与当前 Linux 的新版本有所脱节，这给读者的学习带来了不便。应广大读者的要求，笔者结合 Linux 技术的发展对第 1 版图书进行全面的升级改版，推出第 2 版。相比第 1 版图书，第 2 版在内容上的变化主要体现在以下几个方面：

- Bash 的版本从 4.0 更新到 5.1；
- 增加对 Z Shell 相关内容的讲解；
- 增加助记提示，帮助读者记忆繁杂的选项和参数；
- 修订第 1 版中的一些疏漏，并修正一些表述不够准确的内容；
- 增加思维导图和课后习题，以方便读者梳理和巩固所学知识。

本书特色

1．视频教学，高效、直观

本书涉及很多命令的具体用法，为此笔者专门录制了对应的配套教学视频，以便读者更加高效和直观地学习书中的重要知识点和操作，从而取得更好的学习效果。

2．内容全面，涵盖面广

本书理论结合实践，全面介绍 Linux 系统常用命令的用法，并对 Linux Shell 脚本编程的基本概念、语法、命令、技巧和难点都结合实例和示意图进行详细讲解。

3. 由浅入深，循序渐进

本书从最基本的 Shell 命令开始讲解，逐步深入 Shell 脚本编程，让读者可以循序渐进地掌握 Shell 的各种特性，并在实际开发中加以使用。

4．实例丰富，实用性强

本书结合 700 多个典型实例，对 Linux 的常用命令、Linux Shell 的相关概念与脚本编程的相关知识进行详细的讲解，从而帮助供读者了解这些知识在实际环境中的应用。

5．经验传授，避坑提示

本书总结笔者多年的 Linux 系统管理和程序设计经验，讲解时穿插大量的经验和技巧，并给出多个避坑提示，尽可能帮读者扫清 Linux Shell 编程学习中容易遇到的各种障碍。

6．提供习题、源代码、思维导图和教学 PPT

本书特意在每章后提供多道习题，用以帮助读者巩固和自测该章的重要知识点，另外还提供源代码、思维导图、习题答案和教学 PPT 等配套资源，以方便读者学习和老师教学。

本书内容

第 1 篇　Linux Shell 基础知识与命令

本篇涵盖第 1～4 章，主要介绍 Linux 命令行和 Linux Shell 基础知识，包括 Linux 和 Linux Shell 简介、Bash 简介、Shell 在 Linux 环境中的角色、Shell 变量、Shell 环境进阶、常用的 Shell（Bash）命令、Shell 命令进阶等。通过学习本篇内容，读者可以快速了解 Linux Shell 的基础知识，并掌握 Linux Shell 常用命令的用法，为后续章节的学习打好基础。

第 2 篇　Shell 脚本编程

本篇涵盖第 5～15 章，全面介绍 Shell 脚本编程的相关知识，包括 Shell 编程基础、Shell 的条件执行、Bash 循环、Shell 函数、正则表达式、脚本输入处理、Shell 重定向、管道和过滤器、捕获、sed 和 awk、其他 Linux Shell 概述等。通过学习本篇内容，读者可以全面、系统地掌握 Shell 脚本编程方方面面的知识。

读者对象

- Linux Shell 编程入门人员；
- Linux Shell 编程进阶人员；
- Linux Shell 编程爱好者；
- 基于 Linux 系统的程序员；
- Linux 系统管理员；
- 网络管理人员；
- Linux 培训机构的学员；
- 相关院校的学生与老师；
- 需要 Linux Shell 查询手册的技术人员。

配书资源获取方式

为了便于读者学习，本书提供以下配书资源：

- 配套教学视频；
- 实例源程序；
- 高清思维导图；
- 习题参考答案；
- 配套教学 PPT。

上述配书资源有 3 种获取方式：一是关注微信公众号“方大卓越”，然后回复数字“21”，即可自动获取下载链接；二是到清华大学出版社网站（www.tup.com.cn）上搜索到本书，然后在本书页面上找到“资源下载”栏目，单击“网络资源”按钮进行下载；三是在本书技术论坛（www.wanjuanchina.net）上的 Linux 模块进行下载。

技术支持

虽然笔者对本书所述内容都尽量核对，并多次进行文字校对，但因时间所限，可能还存在疏漏和不足之处，恳请广大读者批评与指正。读者在阅读本书时若有疑问，可以通过以下方式获得帮助：

- 加入本书 QQ 交流群（群号：302742131）进行提问；
- 在本书技术论坛（见上文）上留言，会有专人负责答疑；
- 发送电子邮件到 book@ wanjuanchina.net 或 bookservice2008@163.com 获得帮助。

刘艳涛

2024 年 3 月

目录

第 1 篇　Linux Shell 基础知识与命令

第 1 篇
Linux Shell 基础知识与命令

第 1 章　Linux 和 Linux Shell 简介

欢迎来到 Linux Shell 的世界，在我们开始 Linux Shell 之旅前，先简单地了解关于 Linux 和 Linux Shell 的历史及其相关的基本概念，以便为接下来的学习打好基础。希望通过本章的学习，读者能对 Linux Shell 有一个初步的了解。

1.1　关于 Linux

1.1.1　什么是 Linux

Linux 是自由开源的类 UNIX 操作系统。该操作系统的内核由林纳斯·本纳第克特·托瓦兹（Linus Benedict Torvalds）在 1991 年 10 月 5 日首次公开发布。

狭义来讲，Linux 只表示操作系统的内核本身，但通常采用“Linux 内核”来表达该意思。广义来讲，Linux 常用来指基于 Linux 内核的完整操作系统，包括 GUI 组件和许多其他实用工具。

Linux 最初是作为支持 Intel x86 架构的个人计算机的一个自由操作系统开发的，目前，Linux 已经被移植到更多的计算机硬件平台上。世界上绝大部分的超级计算机都在运行 Linux 发行版或变种，包括最快的前 10 名超级计算机运行的都是基于 Linux 内核的操作系统。Linux 也广泛应用在嵌入式系统上，如手机、平板计算机、路由器、电视和电子游戏机等。在移动设备上广泛使用的 Android 操作系统就是基于 Linux 内核开发的。

Linux 的发展是自由软件和开源软件联盟的结果。只要遵循 GNU 通用公共许可证，任何个人和机构都可以使用、修改和发布 Linux 的底层源代码。通常情况下，Linux 被打包成供桌面应用和服务器应用的 Linux 发行版。一些主流的 Linux 发行版包括 Debian 及其派生版本，（如 Ubuntu 和 Kali Linux）、Red Hat Enterprise Linux 及其派生版本，（如 Fedora 和 CentOS）、openSUSE 及其商业版 SUSE Linux Enterprise Server，以及 Arch Linux。Linux 发行版包含 Linux 内核、配套的实用程序和库，以及发行版使用的大量应用软件。

通常情况下，面向桌面应用的 Linux 发行版包括 X Windows 系统和一个相应的桌面环境，如 GNOME 或 KDE。一些 Linux 发行版也会包含使用较少资源的桌面系统和桌面环境，以适配较老或低性能的计算机。一个用于服务器应用的发行版可能会从标准安装中忽略所有的图形环境，而包含一些其他软件，如 Apache HTTP 服务和一个 SSH 服务。因为 Linux 是一个自由软件，所以任何人都可以创建一个符合自己应用需求的发行版。

1.1.2　谁创建了 Linux

1991 年，林纳斯・本纳第克特・托瓦兹开始了之后被称为 Linux 内核的项目开发。它最初是用于访问大学里的 UNIX 服务器的一个终端模拟器。托瓦兹专门为当时正在使用的硬件写了一个独立于操作系统的程序，因为他想使用 80386 处理器的新功能。这个程序的开发是在使用 GNU C 编译器的 MINIX 操作系统上完成的，即 Linux 的前身。

正如托瓦兹在他的书 *Just for Fun* 中所写，他最终意识到他编写了一个操作系统内核。1991 年 8 月 25 日他在 Usenet 上对外公布这件事情。

1.1.3　Linux 在日常生活中的应用

可以把 Linux 作为一个服务器操作系统来使用，或者作为个人计算机上的独立操作系统使用。作为一个服务器操作系统，Linux 为客户端提供不同的服务和网络资源。一个服务器操作系统必须具有以下特性：

- 稳定的；
- 强壮的；
- 安全的；
- 高性能的。

Linux 不仅具备以上特性，而且它是自由和开源的。它作为一个杰出的操作系统，可以应用于：

- 台式计算机；
- 网站服务器；
- 软件开发工作站；
- 网络监控工作站；
- 工作组服务器；
- 杀手级网络服务，如 DHCP、防火墙、路由、FTP、SSH、邮件、代理和代理缓存服务器等。

1.1.4　Linux 内核是什么

正如前面所说，Linux 内核，即 Linux 操作系统的核心。它主要由以下模块组成：

- 进程管理；
- 定时器；
- 中断管理；
- 内存管理；
- 模块管理；
- 虚拟文件系统接口；
- 文件系统；
- 设备驱动程序；

- 进程间通信；
- 网络管理；
- 系统引导。

Linux 内核决定谁将使用这些资源，可以使用多长时间，以及什么时候可以使用这些资源。它在计算机硬件和各种应用程序之间起到了媒介的作用，如图 1.1 所示。

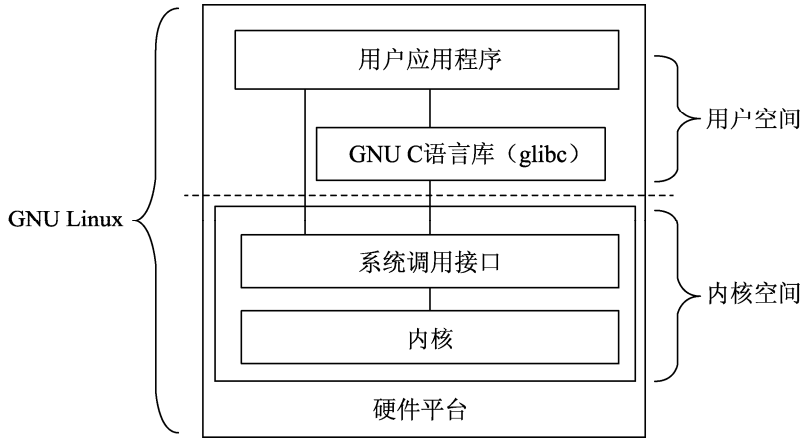

图 1.1　Linux 内核示意

1.1.5　Linux 的理念

如前面所述，Linux 是类 UNIX 的操作系统，UNIX 的理念是一套基于 UNIX 操作系统顶级开发者们的经验提出的软件开发的准则和哲学。因此这些理念也同样适用于 Linux 操作系统：

- 小即是美；
- 让程序只做好一件事；
- 可移植性比效率更重要；
- 一切即文件——使用方便而且把硬件作为文件处理是安全的；
- 使用 Shell 脚本来提高效率和可移植性；
- 避免使用可定制性低下的用户界面；
- 所有程序都是数据的过滤器。

1.2　什么是 Linux Shell

Linux Shell 是用户和 Linux 内核之间的接口程序，为用户提供使用操作系统的接口。当从 Shell 向 Linux 传递命令时，内核会做出相应的反应。

- Shell 是一个用户程序，或是一个为用户与系统交互提供的环境。
- Shell 是一个执行从标准输入设备（如键盘或文件）读入的命令的语言解释程序，它拥有自己内建的 Shell 命令集，Shell 也能被系统中其他应用程序所调用。

- 当用户登录或打开控制台时 Shell 就会运行。
- Shell 不是系统内核的一部分，但是它使用系统内核执行程序及创建文件等。

可以通过多种方式来访问和使用 Shell。

- 终端：Linux 桌面提供基于 GUI 的登录系统。一旦登录，就可以通过运行 X 终端（XTerm）、GNOME 终端（GTerm）或 KDE 终端（KTerm）应用程序来访问 Shell。
- 安全 Shell 连接（SSH）：可以通过它远程登录服务器或工作站来访问其 Shell。
- 使用控制台：一些 Linux 系统同样提供基于文本的登录系统。通常情况下，登录系统后就可以直接访问 Shell。

当普通用户成功登录时，系统将执行一个 Shell 程序，Shell 进程会提供一个命令行提示符。作为默认值，普通用户用“$”作为提示符，超级用户（root）用“#”作为提示符。一旦出现了 Shell 提示符，就可以输入命令名称及命令所需的参数，按 Enter 键后，Shell 将执行这些命令。

当 Shell 执行命令时，首先检查命令是否内部命令，若不是，再检查是否为一个应用程序（这里的应用程序可以是 Linux 本身的实用程序，如 date 和 cat，也可以是购买的商业程序，如 rtds，或者是自由软件，如 Emacs），Shell 在搜索路径里寻找这些应用程序（搜索路径是一个存放可执行程序的目录列表）。如果输入的命令不是一个内部命令并且在搜索路径里没有找到这个可执行文件，则 Shell 会显示一条错误信息。如果能够成功找到命令，则该命令将被分解为系统调用并传给 Linux 内核。

在 Shell 中，可以使用如下按键组合来编辑和回调命令。

- Ctrl + W：删除光标位置前的单词。
- Ctrl + U：清空行。
- ↑和↓方向键：查看命令历史。
- Tab：自动补全文件名、目录名和命令等。
- Ctrl + R：搜索之前使用的命令。
- Ctrl + C：中止当前命令。
- Ctrl + D：退出登录 Shell。
- Esc + T：调换光标前的两个单词。

当用户准备结束登录对话进程时，可以输入 logout 命令、exit 命令或按 Ctrl+D 组合键，结束登录。

Shell 的另一个重要特性是它自身就是一个解释型的程序设计语言，Shell 程序设计语言支持绝大多数在高级语言中能见到的程序元素，如函数、变量、数组和程序控制结构等。Shell 编程语言简单、易学，任何在提示符中输入的命令都可以放到一个可执行的 Shell 脚本中。

1.3　Shell 的种类

Linux（UNIX 或类 UNIX）中的 Shell 有多种类型，最常用的种类有 Bourne Shell（sh）、C Shell、Korn Shell 和 Z Shell。这 4 种 Shell 各有优缺点。

Bourne Shell 是 UNIX 最初使用的 Shell，并且在每种 UNIX 上都可以使用。Bourne Shell

在 Shell 编程方面相当优秀，但是在处理与用户的交互方面不如其他几种 Shell。

Bourne-Again Shell（Bash）是 Linux 系统中最常用的 Shell。它是 Bourne Shell 的扩展，与 Bourne Shell 完全向后兼容，并且在 Bourne Shell 的基础上增加、增强了很多特性，可以提供如命令补全、命令编辑和命令历史等功能，它还包含很多 C Shell 和 Korn Shell 的优点，有灵活和强大的编程接口，同时又有很友好的用户界面。

C Shell（Csh）是一种比 Bourne Shell 更适合编程的 Shell，它的语法和用法与 C 语言很相似，Linux 为喜欢使用 C Shell 编程的人提供了 TCSH。

TCSH 是与 C Shell 兼容的增强版本。它包括命令行编辑、可编程单词补全、拼写校正、历史命令替换、作业控制和类似 C 语言的语法。

Korn Shell（Ksh）集合了 C Shell 和 Bourne Shell 的优点，并和 Bourne Shell 完全兼容。Linux 系统提供了 Ksh 的扩展，它支持任务控制，可以在命令行上挂起在后台执行、唤醒或终止程序。

Z Shell 是一款用于交互式使用的 Shell，包含 Bash、Ksh 和 Tcsh 等其他 Shell 的许多优秀功能，也拥有诸多自身的特色。

Linux 中还包括一些其他的 Shell 类型，比较流行的 Ash 等。但无论哪一种 Shell，它最主要的功能都是解译使用者在命令行提示符中输入的命令。在 MS-DOS 中也有一种 Shell，它的名字是 COMMAND.COM，也用于解释用户输入的命令，只是没有 Linux Shell 的功能强大。每种 Shell 都有它的用途及命令语法，提供不同的内建功能。有些 Shell 是有专利的，有些 Shell 则可从互联网上直接免费获得。

可以使用如下命令查看系统中所有可用的 Shell：

```
-bash-5.1$ cat /etc/shells
/bin/sh
/bin/bash
/sbin/nologin
/bin/tcsh
/bin/csh
/bin/ksh
```

可以看到，系统文件内容有多行，每行都是一种 Shell，表示系统支持多种 Shell。

用户登录 Linux 系统时由/etc/passwd 文件决定将要使用哪种 Shell，例如，查看 root 账号在/etc/passwd 文件中的定义：

```
-bash-5.1$ grep root /etc/passwd
root:x:0:0:System Admin:/root:/bin/bash
```

可以看到，在输出结果中，以冒号“:”分隔的最后一个字段就是定义此账号在登录后所使用的 Shell，由此可知在此实例中，root 账号所使用的 Shell 是 Bash。

还可以使用如下命令查看账号当前使用的 Shell 的类型。

```
-bash-5.1$ echo $SHELL
/bin/bash
```

或者：

```
-bash-5.1$ ps -p $$
  PID TTY          TIME  CMD
23579 pts/0    00:00:00  bash
```

1.4　怎样使用 Shell

要使用 Shell，只需要输入简单的命令即可，命令是一个用于执行特定任务的计算机程序。

如果系统启动后进入的是文本模式，那么在登录系统后就可以直接使用 Shell，可以在登录后的 Shell 中输入命令并执行（命令是为执行特定任务而构建的计算机程序）。如果系统是以图形界面模式启动的，例如 GNOME 桌面或 KDE 桌面，那么可以在图形界面中选择“应用程序”|“系统工具”|“终端”命令打开一个 Shell；或者按 Ctrl+Alt+F3 组合键切换到虚拟控制台并使用用户名和密码登录。如果想切换回图形界面模式，可以按 Ctrl+Alt+F2 组合键。

Linux 终端提供了一个让用户简单地与 Shell（如 Bash）交互的手段。Shell 是一个解释并执行在命令行提示符中输入的命令的程序。当启动 GNOME、KDE 或 X Window 终端时，这些应用程序会启动系统账号中所指定的默认 Shell。可以随时切换到不同的 Shell。接下来简单了解 GNOME 终端的使用和配置。

GNOME 终端程序是完全可以配置的，可以通过设置如下选项来定义一些属性。

- 前景和背景色。
- 窗口标题。
- 字体大小和类型。
- 回滚缓冲区等。

1.5　Shell 脚本是什么

Shell 脚本就像早期的 DOS 时期的.bat 一样，主要的功能就是将许多命令汇总在一起，让使用者能够通过一个操作执行多个命令，方便管理员进行设置或者管理。但是 Shell 脚本比 Windows 中的批处理功能更强大，它提供了数组、循环、条件及逻辑判断等重要的功能，让使用者可以直接以 Shell 来写程序，比用其他编程语言编写的程序效率更高，毕竟它使用了 Linux/UNIX 中的命令。

Shell 脚本是利用 Shell 功能所写的一个程序，这个程序是纯文本文件格式，将一些 Shell 的语法与命令写在里面，然后使用正则表达式、管道命令及数据流重定向等功能实现需要的功能。

Shell 脚本是 Linux/UNIX 编程环境的基本组成部分。Shell 脚本一般由以下几部分构成：

- Shell 关键字，如 if…else 和 for do…done。
- Shell 命令，如 export、echo、exit、pwd 和 return。
- Linux 命令，如 date、rm 和 mkdir。
- 文本处理功能，如 awk、cut、sed 和 grep。
- 函数，通过函数把一些常用的功能放在一起。例如，/etc/init.d 目录下的大部分或全部系统 Shell 脚本所使用的函数，都在文件/etc/init.d/functions 中。

- 控制流语句，如 if…then…else 或执行重复操作的 Shell 循环。

每个 Shell 脚本都有它的用途。例如，备份文件系统和数据库到网络存储服务器上，Shell 脚本可以像 Linux 命令一样被执行。

1.6　为什么使用 Shell 脚本

Shell 脚本的应用知识对于每个想熟练地管理 Linux 操作系统的人是必须要掌握的，即使你可能从来不必写脚本。例如，当 Linux 启动时，它会执行/etc/rc.d 目录下的 Shell 脚本来加载系统配置和运行服务，那么详细地理解这些启动脚本对于分析系统的行为或者修改这些脚本都是很重要的。

学习编写 Shell 脚本并不难，因为它的语法简单、易懂，类似于直接调用命令行并将其串联在一起，并且只有几种规则需要学习。大部分简短的脚本第一次使用就可以正确地执行，即使要调试长的脚本也是简单的。

总之，使用 Shell 的原因如下：

- 使用简单；
- 节省时间：可以把冗长、重复的一连串命令合并成一条简单的命令；
- 可以创建自己的自动化工具和应用程序；
- 使系统管理任务自动化；
- 因为脚本已经过很好的测试，所以在使用脚本做类似配置服务或系统管理任务时，发生错误的概率将大大减少。

常用的 Shell 脚本的实例如下：

- 监控你的 Linux 系统；
- 备份数据和创建快照；
- 创建邮件告警系统；
- 查找耗尽系统资源的进程；
- 查找是否所有的网络服务都正常运行等。

1.7　实例：创建第一个 Shell 脚本

要想成功地写一个 Shell 脚本，需要做以下 3 件事情：

- 写一个脚本。
- 允许 Shell 执行该脚本。
- 把该脚本放到 Shell 可以找到的地方。

一个 Shell 脚本即是一个包含 ASCII 文本的文件，因此可以使用一个文本编辑器来创建一个脚本。文本编辑器是一个类似于读写 ASCII 文本文件的文字处理机程序。有很多种文本编辑器可以在 Linux 系统中使用，它们既可以用于命令行环境也可以用于用户图形界面环境，可以根据自己的喜好选择适合的文本编辑器。

打开文本编辑器并编写包含如下内容的第一个 Shell 脚本：

```
#!/bin/bash
#My first script

ls -l .*
```

保存文件，这里将其命名为 my_first。

脚本的第一行是重要的。它告诉 Shell 使用什么程序解释脚本。在本例中使用的是/bin/bash。其他脚本语言如 Perl、Awk、Python 等也同样使用这个机制。

脚本的第二行是一个注释。每一行中出现在“#”符号后的任何内容都将被 Bash 忽略。一旦脚本变得很大且很复杂，注释将是极其重要的。程序员用注释来解释代码的用途，以方便其他程序员查看代码。

脚本的最后一行是 ls 命令，它将列出当前目录下所有以点开头的文件和目录（即所有隐藏文件和目录）。

默认情况下，Linux 是不允许文件执行的（从安全性来说，这是一件非常好的事情）。下一步要做的就是允许 Shell 执行脚本。我们使用 chmod 命令来完成此操作，命令如下：

```
-bash-5.1$ chmod 755 my_script
```

755 表示给予用户读写和执行的权限，其他用户只有读和执行的权限。如果用户希望自己的脚本是私有的（即只有用户本人可以读写和执行），则使用 700 来替代。现在就可以运行脚本了，只需要在命令行提示符中调用脚本的文件名即可，命令如下：

```
-bash-5.1$ ./my_script
```

之后将会看到脚本的运行结果，如果脚本没有成功运行，请检查实际保存脚本的路径，然后切换到正确的目录并尝试重新运行脚本。

1.8　小　　结

下面总结本章的知识点：

- Linux 是自由开源的类 UNIX 操作系统。该操作系统的内核是由林纳斯·本纳第克特·托瓦兹在 1991 年 10 月 5 日首次公开发布。
- Linux 既可以作为服务器操作系统使用，又可以作为个人计算机的独立操作系统使用。
- Linux 内核即 Linux 操作系统的核心。它的主要模块分为：存储管理、CPU 和进程管理、文件系统、设备管理和驱动、网络通信、系统的初始化（引导）和系统调用等。
- Linux 的主要理念：一个程序只做一件事并做好、一切皆文件、小即是美、在文本文件中存储和配置数据、可移植性高于效率、用户界面简单美观。
- Linux Shell 是用户和 Linux 内核之间的接口程序，为用户提供使用操作系统的接口。
- Linux 中常用的 Shell 有 Bourne Shell（Sh）、C Shell（Csh）、Korn Shell（Ksh）。
- 如果系统启动后进入的是文本模式，那么登录系统后可以直接使用 Shell。
- Shell 脚本是使用纯文本文件，集合一些 Shell 的语法和命令，并用正则表示法或管

道命令以及数据流重导向等功能，达到处理效果的程序。

- Shell 脚本具有使用简单、节省时间、使系统管理自动化等特点。

1.9　习　　题

一、填空题

1．Linux 是类 UNIX 操作系统，该操作系统内核是由_____创建的。

2．Linux Shell 是_____的接口程序，为用户提供使用操作系统的接口。

3．Shell 脚本一般由_____、_____、_____、_____、_____和_____构成。

二、选择题

1．Linux 操作系统具有的特性是（　　）。

A．稳定的　　B．自由的　　C．安全的　　D．高性能

2．当普通用户成功登录系统时，Shell 命令行提示符为（　　）。

A．#　　B．$　　C．%　　D．&

3．使用 Shell 脚本的好处是（　　）。

A．简单　　B．节省时间

C．使系统管理自动化　　D．减少出错概率

三、判断题

1．如果用户的系统启动后进入的是文本模式，则用户登录系统后就可以直接使用 Shell。（　　）

2．创建一个 Shell 脚本后，必须添加执行权限才可以执行。（　　）

四、操作题

1．创建一个名为 test.sh 的 Shell 脚本，输出 Hello World!字符串。

第 2 章　初识 Linux Shell

你是否想过，当在 Shell 的任何目录下运行一个标准的命令行功能（如 cp）时，Shell 是怎么知道这个命令存放在哪里并且是否存在的呢？

这个问题可能会使刚接触 Linux Shell 环境的人感到困惑。因此，本章将介绍 Linux Shell 及 Shell 环境的相关概念。

注意：我们已经知道 Linux Shell 的类型有多种，但本书中介绍的所有 Linux Shell 相关的概念和实例都是以最常用的 Bash 为基础进行讲解的。

2.1　Bash 概述

2.1.1　Bash 简介

Bash 是一个与 Bourne Shell 兼容的命令语言解释器，可以执行从标准输入设备或文件中读取的命令。Bash 与原来的 Bourne Shell（简写为 sh）向后兼容，并且融合了一些有用的 Korn Shell 和 C Shell 的特性。相对于 sh，Bash 在编程和交互式使用两方面都进行了功能改进。另外，大部分的 sh 脚本可以在不修改的情况下由 Bash 直接运行。

Bash 具有很好的移植性。它使用在构建时发现编译平台特征的配置系统，因此可以构建在几乎任何一种 UNIX 版本上。

2.1.2　Bash 的改进

Bash 的语法是 Bourne Shell 语法的一个改进版本。在大多数情况下，Bourne Shell 脚本可以被 Bash 正常运行。下面列出了 Bash 提供的部分改进功能。

- 支持命令行编辑；
- 支持命令行补全；
- 不限制命令行大小；
- 不限制数组的大小；
- 支持 Bash 启动文件：可以运行 Bash 作为一个交互式登录 Shell 或交互式非登录 Shell；
- 支持条件表达式；
- 支持目录堆栈：访问目录的历史记录；

- 支持限制性 Shell：提供了更多的 Shell 执行的控制模式；
- 提供 Bash POSIX 模式：使 Bash 行为更接近 POSIX 标准的规定。

2.2　Shell 在 Linux 环境中的角色

Linux 环境由以下几部分构成。

- 内核：Linux 操作系统的核心。
- Shell：为用户和内核提供一个交互的接口。
- 终端模拟器：允许用户输入命令并在屏幕上回显命令的运行结果。
- Linux 桌面和窗口管理器：Linux 桌面是各种软件应用程序的集合，包括文件管理器、窗口管理器和终端模拟器等。

由此可见，Shell 在 Linux 环境中扮演非常重要的角色，包括读取命令行、解释命令行的含义并执行、通过输出返回执行结果等。

2.2.1　与登录 Shell 相关的文件

当 Linux 系统的运行目标为 multi-user.target 时，用户可以在本地登录系统控制台，或在系统运行目标为 graphical.target 时，直接以图形界面方式登录。这两种情况都需要在登录时输入用户名和密码。这时，Bash 将使用以下初始化文件和启动脚本。

- /etc/profile：系统级的初始化文件，定义了一些环境变量，由登录 Shell 调用执行。
- /etc/bash.bashrc 或/etc/bashrc：其文件名因不同的 Linux 发行版而异，每个交互式 Shell 的系统级的启动脚本都定义了一些函数和别名。
- /etc/bash.logout：系统级的登录 Shell 清理脚本，当登录 Shell 退出时执行。部分 Linux 发行版默认没有此文件。
- $HOME/.bash_profile、$HOME/.bash_login 和$HOME/.profile：用户个人初始化脚本，由登录 Shell 调用执行。这 3 个脚本只有一个会被执行，按照此顺序查找，第一个存在的脚本将被执行。
- $HOME/.bashrc：用户个人的每个交互式 Shell 的启动脚本。
- $HOME/.bash_logout：用户个人的登录 Shell 清理脚本，当登录 Shell 退出时执行。
- $HOME/.inputrc：用户个人的由 Readline 使用的启动脚本，用于处理某些情况下的键盘映射。

2.2.2　Bash 启动脚本

通过 2.2.1 节的介绍可以知道，用户登录时自动执行的脚本主要用于设置一些环境，如设置 JAVA_HOME 的路径。其中的一些脚本被登录 Shell 调用，登录 Shell 是用户登录系统时最先执行的 Shell。登录 Shell 设置一些环境，然后把这些环境授予非登录 Shell。

当用户登录时，登录 Shell 会调用如下脚本。

- /etc/profile：当用户在运行目标 multi-user.target 登录系统时首先运行。

- /etc/profile.d：当/etc/profile 运行时，会调用该目录下的一些脚本。
- $HOME/.bash_profile、$HOME/.bash_login 和$HOME/.profile：在/etc/profile 运行后，登录 Shell 会按照顺序检查这 3 个文件，如果文件存在，则执行并跳过后续的检查。
- $HOME/.bashrc：上述脚本中的一个脚本运行后即调用此脚本。
- /etc/bashrc 或/etc/bash.bashrc：由$HOME/.bashrc 调用运行。

当一个交互式的非登录 Shell 启动时，Bash 将读取并运行如下脚本。

- $HOME/.bashrc：如果此文件存在即运行。
- /etc/bashrc：将被$HOME/.bashrc 调用运行。
- /etc/profile.d：此目录下的脚本将被/etc/bashrc 或/etc/bash.bashrc 调用运行。

Bash 启动脚本主要设置的环境如下。

- 环境变量 PATH 和 PS1（将在 2.3.1 节中介绍这两个变量）。
- 通过变量 EDITOR 设置默认的文本编辑器。
- 设置默认的 umask（文件或目录的权限属性）。
- 覆盖或移除不想要的变量或别名。
- 设置别名。
- 加载函数。

2.2.3　实例：定制自己的 Bash 登录脚本

本节以一个实际的.bash_profile 脚本为例，来学习如何定制一个自己的 Bash 登录脚本。首先在自己的 Linux 账号的 home 目录下创建一个名称为.bash_profile 的文件，然后使用文本编辑器打开并编辑此文件，我们以 Vi 文本编辑器为例，其内容如下：

```
$ vi ~/.bash_profile
# .bash_profile
# Custom Command Prompt
export PS1="\n\e[1;32m[\e[0;31m\u\e[0;34m@\e[0;31m\h\e[1;32m]\e[1;32m[\e
[0;34m\w\e[1;32m]$ "

# Get the aliases and functions
if [ -f ~/.bashrc ]; then
       . ~/.bashrc
fi

# User specific environment and startup programs
PATH=$PATH:$HOME/bin:/usr/bin:/usr/local/bin:/usr/sbin:/usr/local/sbin:
/sbin
export PATH
unset USERNAME
umask 022

# Custom DJRavine Modification
login_pwd=`pwd`;
login_date=`date`;
login_users=`users`;
login_uptime=`uptime`;
server_ip=`/sbin/ifconfig | grep 'inet addr:'| grep -v '127.0.0.1' | head
-1 | cut -d: -f2 | awk '{ print $1}'`;
disk_available=$(df -h --block-size=1024 | awk '{sum += $4;} END {print
sum;}');
```

```
disk_used=$(df -lh --block-size=1024 | awk '{sum += $3;} END {print sum;}');
disk_size=$(df -lh --block-size=1024 | awk '{sum += $2;} END {print sum;}');
disk_available_gb=$(echo "scale=2; $disk_available/(1024^2)" | bc)
disk_used_gb=$(echo "scale=2; $disk_used/(1024^2)" | bc)
disk_size_gb=$(echo "scale=2; $disk_size/(1024^2)" | bc)
red="\033[31m"
blue="\033[34m"
green="\033[32m"
echo -e " "
echo -e "${blue}+-------------------------------------------------------------"
echo -e " ${green}                    Welcome!                                 "
echo -e "${blue}+-------------------------------------------------------------"
echo -e " ${green}Server IP: ${red}"$server_ip
echo -e " ${green}Date: ${red}"$login_date
echo -e " ${green}Users: ${red}"$login_users
echo -e " ${green}Directory: ${red}"$login_pwd
echo -e " ${green}Uptime: ${red}"$login_uptime
echo -e "${blue}+-------------------------------------------------------------"
df -lh  | column -c 6 | awk '{ printf " \033[22;32m%s\t%s\t\033[22;31m%s\
t%s\t%s\n", $1, $6, $2, $3, $4,$5 }'
echo -e " ${green}Total Disk Space: ${red}${disk_size_gb} GB"
echo -e " ${green}Total Free Space: ${red}${disk_available_gb} GB"
echo -e " ${green}Total Used Space: ${red}${disk_used_gb} GB"
echo -e "${blue}+-------------------------------------------------------------"
```

编辑完成后保存并退出文本编辑器。

再创建一个名称为.bashrc 的文件，使用文本编辑器打开并编辑此文件，内容如下：

```
$ vi ~/.bashrc
alias l.='ls -d .* --color=tty'
alias ll='ls -l --color=tty'
alias ls='ls --color=tty'
alias vi='vim'
```

编辑完成后保存并退出文本编辑器。重新登录系统，将会看到如下输出结果：

```
+------------------------------------------------------------------------
                           Welcome!
+------------------------------------------------------------------------
 Server IP: X.X.X.X
 Date: Tue Jun 6 11:16:52 AM CST 2023
 Users: root yantaol
 Directory: /home/yantaol
 Uptime:  11:16:52 up 2 days, 17:10, 3 users, load average: 0.03, 0.03, 0.00
+------------------------------------------------------------------------
 Filesystem    Mounted Size    Used    Avail
 /dev/hde1     /        15G    8.0G    5.8G
 /dev/hde3     /local  208G    151G     46G
 tmpfs  /dev/shm       1.9G    0       1.9G
 Total Disk Space: 223.54 GB
 Total Free Space: 53.21 GB
 Total Used Space: 158.90 GB
+------------------------------------------------------------------------

 yantaol@hostname][~]$ which l.
alias l.='ls -d .* --color=tty'
       /bin/ls
yantaol@hostname][~]$ which vi
alias vi='vim'
       /usr/bin/vim
```

2.2.4　Bash 退出脚本

当登录 Shell 之后需要退出时，如果$HOEM/.bash_logout 脚本存在，则 Bash 会读取并执行此脚本的内容。$HOEM/.bash_logout 脚本的作用如下：

- 使用 clear 命令清理终端屏幕输出；
- 移除一些临时文件；
- 自动运行一些命令或脚本等。

2.2.5　实例：定制自己的 Bash 退出脚本

编辑文件~/.bash_logout，其内容如下：

```
# .bash_logout

#clear the termimal screen
clear

# clear mysql command history
echo "Clear mysql command history"
/bin/rm $HOME/.mysql_history

echo "Backup files to NAS server"
# backup files to NAS server
~/bin/backup.sh
```

编辑完成后保存并退出文本编辑器。此时在命令行提示下运行 exit 命令，将会得到如下输出结果：

```
yantaol@hostname][~]$ exit
logout

Clear mysql command history
Backup files to NAS server
```

如果文件~/.mysql_history 和~/bin/backup.sh 在 home 目录下不存在，当运行 exit 命令时，会看到如下的输出结果：

```
$ exit
logout

Clear mysql command history
/bin/rm: cannot remove `/home/yantaol/.mysql_history': No such file or
directory
Backup files to NAS server
-bash: /home/yantaol/bin/backup.sh: No such file or directory
```

2.2.6　有效地登录 Shell 的路径

/etc/shells 是一个包含有效的登录 Shell 全路径名的文本文件，这个文件既可以被 chsh 命令（变更登录 Shell）使用，又可以被其他程序查询使用，如 FTP 服务。查看/etc/shells 的内容，命令如下：

```
$ cat /etc/shells
```

输出结果如下：

```
/bin/sh
/bin/bash
/sbin/nologin
/bin/tcsh
/bin/csh
/bin/ksh
```

也可以使用 which 命令显示 Shell 的全路径：

```
$ which bash
/bin/bash
```

2.3　Shell 变量

变量是程序或脚本的重要组成部分。变量为程序或脚本访问内存中可被修改的一块数据提供了简单的方式。

Shell 中的变量可以被指定为任意的数据类型，如文本字符串或数值。也可以通过修改 Shell 中的变量来改变 Shell 的样式。

接下来学习 Shell 中的变量。

2.3.1　变量的类型

Shell 中有两种变量类型，即系统变量（环境变量）和用户自定义的变量（本地变量或 Shell 变量）。

系统变量是由 Shell 创建和维护的变量。可以通过修改系统变量如 PS1、PATH、LANG、HISTSIZE 和 DISPLAY 等，配置 Shell 的样式。

常用的系统变量（环境变量）如表 2.1 所示。

表 2.1　常用的系统变量

系 统 变 量	含　义
BASH_VERSION	保存Bash实例的版本
DISPLAY	设置X display的名字
EDITOR	设置默认的文本编辑器
HISTFILE	保存历史命令的文件名
HISTFILESIZE	历史命令文件所包含的最大行数
HISTSIZE	记录在历史命令中的命令数
HOME	当前用户的主目录
HOSTNAME	计算机的主机名
IFS	定义Shell的内部字段分隔符，一般是空格符、制表符和换行符
PATH	搜索命令的路径。它是以冒号分隔的目录列表。Linux中的标准命令之所以能在Shell命令行的任何路径下直接使用，是因为这些标准命令所在的目录的路径定义在了PATH变量中，Shell会在PATH环境变量指定的全部路径中搜索任何匹配的可执行文件

续表

系 统 变 量	含　义
PS1	提示符设定
PWD	当前工作目录，由cd命令设置
SHELL	设置登录Shell的路径
TERM	设置登录终端的类型
TMOUT	用于Shell内建命令read的默认超时时间。单位为s。在交互式的Shell中，此变量的值作为发出命令后等待用户输入的秒数，如果用户没有输入，则会自动退出

当然，用户可以添加上述变量到用户账号的 home 目录下的初始化文件中，如~/.bash_profile 文件。这样在用户每次登录系统时，这些变量会被自动设置为用户需要的值。

如果要查看当前 Shell 的所有系统变量，则在控制台或终端输入如下命令：

```
$ env
```

或者：

```
$ printenv
```

输出结果如下：

```
HOSTNAME=hostname
SHELL=/bin/bash
TERM=vt100
HISTSIZE=1000
USER=yantaol
LS_COLORS=no=00:fi=00:di=01;34:ln=01;36:pi=40;33:so=01;35:bd=40;33;01:
cd=40;33;01:or=01;05;37;41:mi=01;05;37;41:ex=01;32:*.cmd=01;32:*.exe=
01;32:*.com=01;32:*.btm=01;32:*.bat=01;32:*.sh=01;32:*.csh=01;32:*.tar=
01;31:*.tgz=01;31:*.arj=01;31:*.taz=01;31:*.lzh=01;31:*.zip=01;31:*.z=
01;31:*.Z=01;31:*.gz=01;31:*.bz2=01;31:*.bz=01;31:*.tz=01;31:*.rpm=01;
31:*.cpio=01;31:*.jpg=01;35:*.gif=01;35:*.bmp=01;35:*.xbm=01;35:*.xpm=
01;35:*.png=01;35:*.tif=01;35:
MAIL=/var/spool/mail/yantaol
PATH=/usr/local/bin:/bin:/usr/bin
INPUTRC=/etc/inputrc
PWD=/home/yantaol
LANG=en_US.iso88591
KDE_IS_PRELINKED=1
KDEDIRS=/usr
HOME=/home/yantaol
LOGNAME=yantaol
CVS_RSH=ssh
_=/usr/bin/printenv
```

用户自定义的变量即由用户创建和维护的变量。这种类型的变量可以使用任何有效的变量名来定义。

如果要查看当前 Shell 中的所有用户自定义的变量和系统变量，那么可以在控制台或终端上使用 set 命令查看，命令如下：

```
$ env
```

输出结果如下：

```
BASH=/bin/bash
BASH_ARGC=()
BASH_ARGV=()
BASH_LINENO=()
```

```
BASH_SOURCE=()
BASH_VERSINFO=([0]="3" [1]="2" [2]="25" [3]="1" [4]="release" [5]=
"i686-redhat-linux-gnu")
BASH_VERSION='3.2.25(1)-release'
COLUMNS=126
CVS_RSH=ssh
DIRSTACK=()
HISTFILE=/home/yantaol/.bash_history
HISTFILESIZE=1000
HISTSIZE=1000
HOME=/home/yantaol
HOSTNAME=hostname
HOSTTYPE=i686
IFS=$' \t\n'
INPUTRC=/etc/inputrc
LANG=en_US.iso88591
LESSOPEN='|/usr/bin/lesspipe.sh %s'
LINES=40
LOGNAME=yantaol
LS_COLORS='no=00:fi=00:di=01;34:ln=01;36:pi=40;33:so=01;35:bd=40;33;01:
cd=40;33;01:or=01;05;37;41:mi=01;05;37;41:ex=01;32:*.cmd=01;32:*.exe=01;
32:*.com=01;32:*.btm=01;32:*.bat=01;32:*.sh=01;32:*.csh=01;32:*.tar=01;
31:*.tgz=01;31:*.arj=01;31:*.taz=01;31:*.lzh=01;31:*.zip=01;31:*.z=01;
31:*.Z=01;31:*.gz=01;31:*.bz2=01;31:*.bz=01;31:*.tz=01;31:*.rpm=01;31:
*.cpio=01;31:*.jpg=01;35:*.gif=01;35:*.bmp=01;35:*.xbm=01;35:*.xpm=01;
35:*.png=01;35:*.tif=01;35:'
PATH=/usr/local/bin:/bin:/usr/bin
PIPESTATUS=([0]="0")
PS1='\s-\v\$ '
PS2='> '
PS4='+ '
PWD=/home/yantaol
SHELL=/bin/bash
SHLVL=1
SUPPORTED=en_US.UTF-8:en_US:en:zh_CN.UTF-8
TERM=vt100
USER=yantaol
_=set
consoletype=pty
myvar=showset
```

2.3.2　实例：如何定义变量并给变量赋值

在 Shell 中，当第一次使用某个变量名时，实际上就是定义了这个变量。在 Shell 中创建和设置变量是很简单的，其语法如下：

```
varName=varValue
```

在上例中，我们看到可以使用赋值操作符“=”给变量赋值。输入的次序是：变量名、赋值操作符和赋予的值，varName 即变量名，varValue 是赋予 varName 的值。如果没有给出 varValue，则变量 varName 会被赋予一个空字符串。

在赋值操作符“=”的前后，不要有任何空格。例如，下面的变量定义将会得到 command not found 的错误：

```
varName  =  varValue
varName  =varValue
varName=  varValue
```

可以把任意字符集合赋值给一个变量。例如，将字符串 yantaol 赋值给 username：

```
$ username=yantaol
```

或者：

```
$ username=”yantaol”
```

将一个数字赋值给变量：

```
$ var=1
```

需要注意，shell 的默认赋值是字符串赋值，例如接下来的操作：

```
$ var=$var+1
$ echo $var
1+1
```

此时变量的值是“1+1”，而不是我们预想中的值“2”，在 Bash 中，如果将算术表达式的数值赋值给一个变量，则可以使用 let 命令：

```
$ let var=2+1
$ echo $var
3
```

将一个变量的值直接赋值给另一个变量：

```
$ a=3
$ b=$a
$ echo $b
3
```

将命令的执行结果赋值给变量：

```
$ var=`pwd`
$ echo $var
/home/yantaol
```

也可以使用$(…)来实现同样的功能：

```
$ var=$(pwd)
$ echo $var
/home/yantaol
```

将 Bash 的内置命令 read 读入的内容赋值给变量：

```
$ echo -n "Enter var:"; read var
Enter var:123
$ echo $var
123
```

在上例中，echo -n "Enter var:"语句是打印内容 Enter var:并且不换行，紧接着从标准输入读入内容，这里输入“123”后按 Enter 键，read 命令将读入的内容赋值给 var，即在输出结果中看到的 var 的值是 123。

2.3.3　变量的命名规则

变量名可以包含字母、数字和下画线“_”，但首字符必须是字母或下画线。不要使用“？”“*”和其他特殊字符命名变量。

有效的 Shell 变量名示例如下：

```
USERNAME
LD_LIBRARY_PATH
```

```
_var
var1
```

注意，以下变量名是无效的：

```
?var=123
user*name=yantaol
```

变量名是大小写敏感的，例如，定义如下几个变量：

```
$ var=123
$ Var=1
$ vAR=2
$ VAR=3
```

它们都是不同的变量：

```
$ echo $var
123
$ echo $Var
1
$ echo $vAR
2
$ echo $VAR
3
```

2.3.4　实例：使用 echo 和 printf 命令打印变量的值

通过前面几节的学习，我们已经知道可以使用 echo 命令显示变量的值，此外，还有一种显示变量值的方法就是使用 printf 命令，例如：

```
$ var=123
$ printf "%s\n" $var
123
```

printf 命令提供了一个类似于 printf()系统接口（C 函数）的打印格式化文本的方法。它作为 echo 命令的替代，提供了更多的特性。

printf 命令的语法格式如下：

```
printf <FORMAT> <ARGUMENTS…>
```

printf 命令就是根据指定的格式<FORMAT>打印<ARGUMENTS…>的内容。文本格式在<FORMAT>中指定，紧接着是需要格式化的所有参数。

由此，一个典型的 printf 命令调用如下：

```
printf "FirstName: %s\nLastName: %s" "$FIRSTNAME" "LASTNAME"
```

其中，FirstName: %s\nLastName: %s 和$FIRSTNAME 是格式规范，而后面的两个变量则是作为参数传入。格式字符串中的%s 打印参数的格式类型的分类符，这些分类符有不同的名称，如表 2.2 所示。

表 2.2　分类符

分　类　符	描　　述
%b	打印相关参数并解释其中带有反斜杠“\”的特殊字符
%q	以Shell引用的格式打印相关参数，使其可以在标准输入中重用
%d	以带符号十进制数的格式打印相关参数

续表

分　类　符	描　　述
%i	与%d相同
%o	以无符号八进制数的格式打印相关参数
%u	以无符号十进制数的格式打印相关参数
%x	以无符号小写十六进制数的格式打印相关参数
%X	与%x相同，只是十六进制数为大写
%f	以浮点数的格式解析并打印相关参数
%e	以双精度浮点数<N>±e<N>的格式打印相关参数
%E	与%e相同，只是用大写字母E
%g	以%f或%e的格式打印相关参数
%G	以%f或%E的格式打印相关参数
%c	以字符的格式打印相关参数，并且只打印参数中的第一个字符
%s	以字符串的格式打印相关参数
%n	指定打印的字符个数
%%	表示打印一个字符“%”

在 printf 命令的格式用字符串<FORMAT>中还可以使用一些转义字符，如表 2.3 所示。

表 2.3　转义字符

转 义 字 符	描　　述
\"	打印双引号
\NNN	用八进制的值表示一个ASCII字符，例如\101，即65，表示字符A
\\	打印一个反斜杠\
\a	发出告警音
\b	删除前一个字符
\f	换页符，在某些实现中会清屏或换行
\n	换行
\r	从行首开始，和换行不一样，仍在本行
\t	Tab键
\v	竖直制表，和\f相似，不同计算机的显示有所不同，通常会引起换行VERTICAL TAB or CTRL-K
\xHH	用十六进制的值表示一个ASCII字符，例如/x41，即65，表示字符A

接下来通过实例来了解如何使用 printf 命令格式化打印变量的值，代码如下：

```
$ var=shell
$ printf "%s\n" $var
shell
$ printf "%1s\n" $var        #如果指定的长度小于参数的实际长度，则打印完整的字符串
shell
$ printf "%1.1s\n" $var      #原点符右边的数字表示打印参数中的字符个数
s
$ printf "%1.2s\n" $var
sh
$ printf "%2.6s\n" $var
```

```
shell
$ printf "%5.6s\n" $var
shell
$ printf "%6.6s\n" $var
 shell
#如果指定打印的字符串长度超过输出的实际长度，则左边用空格补全
$ printf "%10.4s\n" $var
     shel
$ printf "%10.2s\n" $var
       sh
$ printf "%c\n" $var
s
$ var=100
$ printf "%d\n" $var
100
$ printf "%i\n" $var
100
$ printf "%u\n" $var
100
$ printf "%o\n" $var
144
bash-5.1$ printf "%s%%\n" $var
100%
$ var=123
$ printf "%x\n" $var
7b
$ printf "%X\n" $var
7B
$ var=12345678
$ printf "%e\n" $var
1.234568e+07
$ printf "%E\n" $var
1.234568E+07
$ printf "%g\n" $var
1.23457e+07
$ printf "%G\n" $var
1.23457E+07
$ var=123.45678
$ printf "%5.1f\n" $var         #打印的数值长度为 5，保留小数点后 1 位的数字
123.5
$ printf "%6.2f\n" $var
123.46
$ printf "%7.3f\n" $var
123.457
#打印的数值长度为 9，数值的实际长度为 7，保留小数点后 3 位，左面用空格补全
$ printf "%9.3f\n" $var
123.457
$ printf "%10.3f\n" $var
   123.457
$ var="abc def ghi 'l\mn"
$ printf "%q\n" "$var"
abc\ def\ ghi\ \'l\\mn
```

与 printf 命令不同，echo 命令没有提供格式化选项，因此 echo 命令比 printf 命令更简单易用。当知道所要显示的变量的内容不会引起问题时，echo 命令是一个很好的显示简单输出的命令。

echo 命令也提供了转义字符的功能，可以使用的转义字符与 printf 命令中的转义字符基本相同，但需要使用-e 选项激活转义字符功能。

下面通过实例来了解如何使用 echo 命令打印变量的值。

```
$ var=10
$ echo "The number is $var"
The number is 10
$ echo -e "Username: $USER\tHome directory: $HOME\n"
Username: yantaol     Home directory: /home/yantaol
```

有时，需要使用${}避免一些歧义。例如：

```
$ LOGDIR="/var/log/"
$ echo "The log file is $LOGDIRmessages"
```

输出结果如下：

```
The log file is
```

Bash 尝试查找一个叫作 LOGDIRmessages 的变量，而不是$LOGDIR。为了避免这种歧义，这里需要使用${}语法，命令如下：

```
$ echo "The log file is ${LOGDIR}messages"
The log file is /var/log/messages
```

2.3.5　变量的引用

当引用一个变量时，通常是用双引号（" "）将变量名括起来，如"$variable"，这样可以防止被引用的变量值中的特殊字符（除$、`和\）被解释为错误的含义。

使用双引号可以防止变量值中由多个单词组成的字符串被分离。一个用双引号括起来的变量变为一个单一的词组，即使它的值中包含空格。下面通过一个实例来了解双引号在引用变量中的作用，代码如下：

```
$ LIST="one two three"
$ for var in $LIST          #将变量 LIST 的值分成 3 个参数传入 for 循环
> do
>    echo "$var"
> done
one
two
three
$ for var in "$LIST"        #将变量 LIST 的值作为一个整体传入 for 循环
> do
>    echo "$var"
> done
one two three
```

再看一个复杂一点的例子，代码如下：

```
var1="a variable containing five words"
COMMAND This is $var1       #执行带有 7 个参数的 COMMAND 命令："This" "is" "a"
                            #"variable" "containing" "five" "words"

COMMAND "This is $var1"     #执行带有一个参数的 COMMAND 命令："This is a
                            #variable containing five words"


var2=""                             #赋予一个空字符串

COMMAND $var2 $var2 $var2           #相当于无参数执行 COMMAND 命令
```

```
COMMAND "$var2" "$var2" "$var2"     #执行带有 3 个空参数的 COMMAND 命令
COMMAND "$var2 $var2 $var2 "        #执行带有 1 个参数（两个空格）的 COMMAND 命令
```

注意：只有在变量的值中包含空格或要保留其中的空格时，将变量用双引号括起来才是必要的。

在下面的例子中，使用 echo 命令打印出了一些奇怪的变量值：

```
$ var="'(]\\{}\$\""
$ echo $var
'(]\{}$"
$ echo "$var"            #在这里，使用双引号和不使用双引号打印的值没有区别
'(]\{}$"
#在 2.3.1 节中讲过，IFS 是 Bash 的内部变量，此变量定义 Shell 的内部字段分隔符
$ IFS='\'
$ echo $var              #不加双引号的情况下，在其打印的值中，反斜杠'\'被转换为空格
'(] {}$"
$ echo "$var"
'(]\{}$"
$
$ var2="\\\\\""
$ echo $var2
  "
$ echo "$var2"
\\"
$ var3='\\\\'
$ echo "$var3"
\\\\
$
$ echo "$(echo '"')"    #$(echo '"')相当于`echo '"'`
"
$ var4="Two words"
$ echo "\$var4 = "$var4""
$var4 = Two words
```

单引号的操作类似于双引号，但是它不允许引用变量，因为在单引号中，字符'$'的特殊含义将会失效。每个特殊的字符，除了字符“'”，都将按字面含义解释。

2.3.6　实例：export 语句的使用

在前面章节（主要参见 2.2.2 节）的学习中，我们了解到用户登录系统后，系统将会启动一个登录 Shell。在这个 Shell 中，可以声明一些变量，也可以创建并运行 Shell 脚本程序。运行 Shell 脚本时，系统将创建一个子 Shell，当此 Shell 脚本运行完毕时，这个子 Shell 将终止，环境将返回到执行该脚本运行之前的 Shell。默认情况下，所有用户定义的变量只在当前 Shell 内有效，它们不能被后续的 Shell 引用。要使某个变量可以被子 Shell 引用，可以使用 export 命令将变量进行输出。

Bash 的内置命令 export 会将指定给它的变量或函数自动输出到后续命令的执行环境中。export 命令的语法如下：

```
export [-fnp] [变量或函数名称]=[变量设置值]
```

-f 选项表示 export 为一个函数；-n 选项表示将 export 属性从指定变量或函数中移除；-p 选项表示打印当前 Shell 所有输出的变量，与单独执行 export 命令结果相同。

接下来通过实例来了解 export 命令的用途。例如，创建一个变量 JAVA_HOME，然后给它赋予一个值“/usr/local/jdk”，命令如下：

```
$ JAVA_HOME=/usr/local/jdk
```

使用 echo 命令显示变量的值，命令如下：

```
$ echo $JAVA_HOME
/usr/local/jdk
```

现在运行一个新的子 Shell，命令如下：

```
$ bash
```

再使用 echo 命令显示变量 JAVA_HOME 的值，命令如下：

```
bash-5.1$ echo $JAVA_HOME
```

结果将会得到一个空行，因为变量 JAVA_HOME 没有被输出到新的子 Shell 中。下面看看使用 export 命令的结果。先退出子 Shell，命令如下：

```
bash-5.1$ exit
exit
```

然后运行 export 命令，将变量 JAVA_HOME 输出到后续命令的执行环境中，命令如下：

```
$ export JAVA_HOME
```

现在运行一个新的子 Shell，再使用 echo 命令显示变量 JAVA_HOME 的值，命令如下：

```
bash-5.1$ bash
bash-5.1$ echo $$
30798
bash-5.1$ echo $JAVA_HOME
/usr/local/jdk
```

再运行一个新的子 Shell，变量 JAVA_HOME 仍然有效：

```
bash-5.1$ bash
bash-5.1$ echo $$
31812
bash-5.1$ echo $JAVA_HOME
/usr/local/jdk
```

注意：系统变量会自动输出到后续命令的执行环境中。

2.3.7　实例：如何删除变量

在 Bash 中使用 unset 命令可以删除相应的变量或函数。unset 命令会把变量从当前 Shell 和后续命令的环境中删除，其命令语法如下：

```
unset [-fv] [变量或函数名称]
```

-f 选项表示删除一个已定义的函数；-v 选项表示删除一个变量。

下面学习 unset 命令的使用：

```
bash-5.1$ export JAVA_HOME=/usr/local/jdk
bash-5.1$ unset JAVA_HOME
bash-5.1$ echo $JAVA_HOME
```

注意：使用 unset 命令不能删除一个只读的变量，否则将会出现如下错误。

```
$ readonly JAVA_HOME=/usr/local/jdk
$ echo $JAVA_HOME
/usr/local/jdk
$ unset JAVA_HOME
-bash: unset: JAVA_HOME: cannot unset: readonly variable
```

2.3.8　实例：如何检查变量是否存在

可以使用以下语法来检查一个变量是否存在：

```
${varName? Error: The variable is not defined}
```

上述语句的含义是，如果变量 varName 已定义且不为空，则此语句就相当于$varName；如果变量 varName 的值是空，则此语句的值也是空；如果 varName 是未定义的，则此语句将返回一个错误，并显示问号“？”定义的错误信息 Error: The variable is not defined。

此外，也可以使用下面这个语句：

```
${varName:? Error: The variable is not defined}
```

此语句和上一条语句的唯一区别是：如果变量 varName 的值是空的，则此语句也返回一个错误。

这两个语句对于脚本的完整性检查是有用的，如果使用这两个语句检查的变量未定义，则脚本将会停止执行。

下面用一个实例来演示，如何使用上述格式的语句检查一个变量是否存在：

```
$ JAVA_HOME=/usr/local/jdk #定义一个变量 JAVA_HOME 并赋予值“/usr/local/jdk”
#检查变量是否存在并打印变量的值
$ echo ${JAVA_HOME? ERROR: The variable is not defined}
/usr/local/jdk
$ unset JAVA_HOME           #删除变量 JAVA_HOME
#检查变量是否存在并打印变量的值
$ echo ${JAVA_HOME:? ERROR: The variable is not defined}
bash: JAVA_HOME: ERROR: The variable is not defined
```

Bash 中还有一种更常用的检查变量是否存在的方法，即使用 if 语句判断变量是否存在。我们将在条件测试章节中对使用 if 语句进行详细介绍。

2.4　Shell 环境进阶

通过前几节的学习，想必读者对 Linux Shell 环境已经有了基本的了解，应该对本章开始提出的关于标准命令为什么从 Shell 中的任何路径都可以访问的问题有了答案。没错，这是因为 Shell 会在 PATH 环境变量指定的全部路径中搜索任何可执行的文件，一旦找到与输入相匹配的命令就执行。

接下来详细介绍 Shell 环境相关的知识。

2.4.1　实例：回调历史命令

Bash 将历史命令保存在缓冲区或默认文件~/.bash_history 中。在历史命令缓冲区中可

以保存很多命令，保存命令的多少由环境变量 HISTSIZE 定义。

可以使用 history 命令显示在命令行提示符状态下输入的命令列表，例如：

```
$ history
    6  exit
    7  rm -f .tcshrc
    8  ln -sf /opt/swe/tools/in/env/tcshrc .tcshrc
    9  ls -al. tcs
   10  ls -al .tcshrc
   11  exit
   12  exit
   13  hostname
   14  history
   15  ps -ef|grep sgd
   16  id
   17  exit
   18  pwd
   19  pwd
   20  cd /svn/
   21  ls
……
```

从例子中可以看到，history 命令可以显示历史命令列表的行号。

可以在命令行提示符状态下使用“↑”和“↓”方向键找到之前执行过的命令。在命令行提示符状态下，按 Ctrl+R 键后输入相应的关键字，可以搜索历史命令。例如，按 Ctrl+R 键后输入 export，将会看到如下结果：

```
(reverse-i-search)`export': export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:
/usr/local/sasl2/lib
```

在 Shell 命令行提示符状态下，可以简单地输入!!来重复执行上一条执行过的命令，例如：

```
$ uptime
 13:00:28 up 2 days, 13:00,  4 users,  load average: 0.00, 0.01, 0.00
$ !!
uptime
 13:00:32 up 2 days, 13:00,  4 users,  load average: 0.00, 0.01, 0.00
```

此外，还可以回调最近一次执行的以指定字符开头的命令，例如：

```
$ !up
uptime
 13:24:08 up 2 days, 13:23,  4 users,  load average: 0.00, 0.01, 0.00
```

甚至可以使用由 history 命令列出的列表的行号来重新调用相应的历史命令，例如：

```
$ history
……
  990  uptime
  991  uptime
  992  man svn
  993  history
  994  pwd
  995  history
……
$ !991
uptime
 13:32:39 up 2 days, 13:32,  4 users,  load average: 0.09, 0.06, 0.02
$ !994
```

```
pwd
/home/yantaol
```

2.4.2　实例：Shell 的扩展部分

Shell 中的扩展方式有 8 种，分别是（按扩展的先后顺序排序）：花括号扩展、波浪号扩展、参数和变量扩展、命令替换、算术扩展、进程替换、单词拆分和文件名扩展。

参数和变量扩展我们将在 5.4.2 节详细介绍，算术扩展将在 5.5.3 节详细介绍，进程替换依赖于操作系统的支持，这里不进行详细介绍，本节主要介绍花括号扩展、波浪号扩展、命令替换和文件名扩展的相关内容。

花括号扩展是一种能够生成任意字符串的机制。进行花括号扩展的模式在形式上有一个可选的前缀，其后是一组包含在花括号内的用逗号分隔的字符串或序列表达式，最后是一个可选的后缀。例如：

```
$ echo a{b,c,d}e
abe ace ade
```

从上面的例子中可以看出，前缀添加在生成的每个字符串的前面，而后缀将会添加在生成的每个字符串的尾部，整个扩展将从左向右进行。

再看一个花括号内是序列表达式的例子：

```
$ echo {a..z}                    #按字母表顺序显示 a 到 z 的字母
a b c d e f g h i j k l m n o p q r s t u v w x y z
$ echo {0..10}                   #显示 0～10 的数字
0 1 2 3 4 5 6 7 8 9 10
$ echo {5..-3}                   #由大到小地显示-3～5 之间的数字
5 4 3 2 1 0 -1 -2 -3
$ echo {g..a}
g f e d c b a
$ echo {1..3}{a..c}
1a 1b 1c 2a 2b 2c 3a 3b 3c
```

花括号扩展也可以是嵌套的。每个扩展字符串的结果是不排序的，依然按照从左到右的顺序依次扩展，例如：

```
$ echo a{{b,c,d}a,{e,f,g}b,h}i
abai acai adai aebi afbi agbi ahi
$ echo {a,b{1..3},c}
a b1 b2 b3 c
```

花括号扩展可以和许多命令配合使用，可以使命令更简化。例如，在主目录下创建 3 个目录，可以使用如下语句：

```
$ mkdir ~/{dir1,dir2,dir3}       #在 home 下创建 dir1、dir2 和 dir3 这 3 个子目录
```

关于 mkdir 命令，会在 3.2 节详细介绍，读者也可以使用 man 参考手册查看此命令。

在 Bash 4.0 中还提供了一些花括号扩展的新功能。例如，在序列表达式中指定一个增量<INCR>，格式如下：

```
{<START>..<END>..<INCR>}
```

<STSRT>和<END>是整数或者单个字符，增量<INCR>是一个整数。扩展后每个量之间的差值就是指定的增量，例如：

```
$ echo {1..10..2}
1 3 5 7 9
$ echo {10..1..-2}
10 8 6 4 2
$ echo {a..h..3}
a d g
$ echo {h..a..-3}
h e b
```

如果<START>和<END>是整数并且有前导 0，则 Bash 4.0 会试图让每个生成的量都含有同样多的位数，如果位数不同就在前面补 0，例如：

```
$ echo {0001..10..3}
0001 0004 0007 0010
```

花括号扩展在其他扩展之前进行，在其他扩展中的特殊字符都被保留了下来。如果其他扩展包含“{”或者“,”可以用反斜线转义。为了避免与参数扩展发生冲突，花括号扩展不会识别字符串中的“${”。

波浪号扩展可用来指代你自己的主目录或其他人的主目录，例如：

```
$ cd ~                              #进入自己的主目录
$ pwd
/home/yantaol
$ cd ~fred                          #进入用户 fred 的主目录
$ pwd
/home/fred
```

如果一个单词以未被引用的波浪号“~”开头，那么直到第一个未被引用的斜线之前的所有字符都会被看作波浪号前缀。如果波浪号前缀里面的字符都没有被引用，则波浪号后面的所有字符就会被当成一个可能存在的登录用户名。如果这个登录用户名是一个空字符串，则波浪号会被替换成 Shell 变量 HOME 的值，如果没有设置 HOME 变量，则将波浪号替换成执行该 Shell 的用户的主目录；否则波浪号前缀就被替换成指定的登录用户名的主目录。

如果波浪号前缀是“~+”，则它会被 Shell 变量 PWD 的值代替，例如：

```
$ echo ~+
/tmp
$ echo $PWD
/tmp
```

如果波浪号前缀是“~-”，则它会被 Shell 变量 LODPWD 的替代（如果这个变量已经设置），例如：

```
$ echo ~-
/home/yantaol
$ echo $OLDPWD
/home/yantaol
```

命令替换是用命令的输出替换命令本身，命令替换有如下两种形式：

```
$(COMMAND)
```

或者：

```
`COMMAND`
```

Bash 进行这个扩展时，先执行命令，然后用命令的标准输出结果取代命令替换，在命令的标准输出结果中，最后面的换行符会被删除。例如：

```
$ echo $(uptime)
17:10:35 up 2 days, 11:07, 0 users, load average: 0.00, 0.00, 0.00
$ echo `uptime`
17:10:40 up 2 days, 11:07, 0 users, load average: 0.00, 0.00, 0.00
```

命令替换可以嵌套。使用反引号形式“``”进行嵌套时，里面的反引号需要用反斜杠“\”转义。

如果在 Bash 中没有设置-f 选项，则支持文件名扩展。Bash 支持以下 3 种通配符来实现文件名扩展。

- *：匹配任何字符串，包括空字符串。
- ?：匹配任意单个字符。
- […]：匹配方括号内的任意字符。

例如，显示/etc 目录下的所有配置文件：

```
$ ls /etc/*.conf
```

或者列出所有以字母 a 或 b 开头的配置文件：

```
$ ls /etc/[ab]*.conf
```

显示所有 JPG 格式的文件（image1.jpg、image2.jpg..image9.jpg）：

```
$ls image?.jpg
```

2.4.3　实例：创建和使用别名

在 Linux 系统环境下，通常需要使用命令行处理一些任务，并且会很频繁地使用某些命令语句。为了节省时间，可以在文件~/.bashrc 中为这些命令语句创建一些别名。

注意：一旦修改~/.bashrc 文件，必须重新登录 Shell，新的设置才会生效。

Bash 的内置命令 alias 用于创建一个别名，创建别名的语法如下：

```
alias name='command'
```

- name：用户定义的用于别名的任意简短的名字。
- command：任意的 Linux 命令。

下面来看一个比较常用的别名的实例：

```
alias ll='ls -l'
```

在命令行输入上面的命令后，接下来每次输入 ll 后按 Enter 键，Bash 会自动将其替换为 ls −l 来执行。

注意：当在命令行执行上面的命令时，会创建一个临时的别名，这个别名只在你退出此 Shell 之前有效。

接下来看一些比较有用的别名的实例，可以将下面这些别名放到自己的~/.bashrc 文件中。

打开当前目录下最后被修改的文件：

```
alias Vim='vim `ls -t | head -1`'
```

找出当前目录下 5 个最大的文件：

```
alias findbig='find . -type f -exec ls -s {} \; | sort -n -r | head -5'
```

列出当前目录下的所有文件，包括隐藏文件，并附加指示符和颜色标识：

```
alias ls='ls -aF --color=always'
```

清除全部历史命令记录和屏幕：

```
alias hcl='history -c; clear'
```

使用 cp、mv 等命令交互式地执行并解释执行了哪些操作：

```
alias cp='cp -iv'
alias mv='mv -iv'
alias rm='rm -iv'
```

简化经常执行的命令：

```
alias x='exit'
```

查看磁盘空间使用情况：

```
alias dus='df -h'
```

切换到不同的目录下：

```
alias ..= 'cd ..'
alias …='cd ../..'
```

上面介绍了如何创建一个别名，那么能否查看当前环境下所有别名呢？答案是肯定的，直接使用命令 alias 不带任何参数即可列出所有别名，示例如下：

```
$ alias
alias ..='cd ..'
alias ...='cd ../..'
alias Vim='vi `ls -at|head -1`'
alias findbig='find . -type f -exec ls -s {} \; | sort -n -r | head -5'
```

如何查看一个特定的别名呢？示例如下：

```
$ alias dus
alias dus='df -h'
```

如果想调用实际的命令而暂时停止使用别名，则可以使用如下方法：

```
$ \aliasname
```

例如，别名 alias cp='cp -iv'，将会请你确认是否要覆盖一个已存在的文件。当要复制很多已经确认要覆盖的文件时，这些询问是很麻烦的。这时你可能会想临时地使用正常的 cp 命令替代这个 cp 别名，可以使用如下的方法：

```
$ \cp * ~/backup/files/
```

如果想删除一个别名，应该怎么办呢？Bash 的内建命令 unalias 提供了这个功能，它可以删除一个特定的别名，示例如下：

```
$ unalias dus
$ alias dus
bash: alias: dus: not found
```

使用命令 unalias 加-a 选项将删除当前环境下的所有别名：

```
$ unalias -a
$ alias
```

注意：虽然别名的使用简单又方便，但是要非常谨慎地使用别名替换标准命令。

2.4.4　实例：修改 Bash 提示符

我们在前面的章节中提到过环境变量 PS1，设置此变量即可定制自己的 Bash 提示符。显示当前提示符的设置，命令如下：

```
$ echo $PS1
\u@\h \w\n$
```

在本例中，变量 PS1 的值显示的信息如下。

- \u：当前用户的用户名；
- \h：主机名；
- \w：当前工作目录的全路径；
- \n：新的一行；
- \$：如果当前用户的 UID 是 0，则显示符号“#”，否则显示符号“$”。

下面来看看还有哪些带有反斜杠的特殊字符可以用于定制 Bash 提示符。

- \a：一个 ASCII 码报警符（07）；
- \d：“星期 月 日”格式的日期（如“Wed Aug 07”）；
- \e：一个 ASCII 码转义符（033）；
- \H：长主机名；
- \j：由当前 Shell 管理的任务数；
- \l：Shell 的终端设备的文件名；
- \r：回车；
- \s：Shell 的名称（如 bash）；
- \t：24 时制“HH:MM:SS”格式的当前时间（如 22:59:25）；
- \T：12 时制“HH:MM:SS”格式的当前时间（如 11:01:32）；
- \@：12 时制“HH:MM AM/PM”格式的当前时间（如 11:00 PM）；
- \A：24 时制“HH:MM”格式的当前时间（如 23:02）；
- \v：Bash 的版本（如 4.1）；
- \V：Bash 的发行号，版本+补丁级别（如 4.1.10）；
- \W：当前工作目录去掉前导目录后的目录名，如果是变量$HOME 所指定的目录，则用符号“~”代替；
- \!：上一个被执行命令的历史编号；
- \#：该命令的命令编号；
- \nnn：与八进制数 nnn 相对应的 ASCII 码；
- \\：一个反斜杠字符；
- \[：开始一个非打印字符序列，可用于将终端控制序列嵌入提示符中；
- \]：结束非打印字符序列。

知道了上述特殊字符的含义，接下来就可以使用这些特殊字符的组合来定制自己的提示符了。

例如，下面的命令用于定制一个可以显示当前时间的提示符：

```
$ export PS1="[\t] \u@\h\n\$ "
[23:42:31] yantaol@server
$
```

也可以在提示符中显示一个命令的输出。下面来看一个在提示符中显示内核版本的例子：

```
$ export PS1="`uname -r`|\u@\h\n\$ "
5.14.0-162.6.1.el9_1.x86_64|yantaol@server
$
```

让 Bash 提示符显示当前用户的进程数，命令如下：

```
$ export PS1="\u@\h [$(ps -ef | grep `whoami` | grep -v grep | wc -l)]> "
```

同样可以将上述语句加入所用账号的~/.bashrc 中，以便登录后自动生效。

此外，还可以让 Bash 提示符带有颜色。下面看一个设置 Bash 提示符带有前景色的例子：

```
$ export PS1="\e[0;34m\u@\h \w\n\$\e[m "
```

上面定义的几个特殊字符的含义如下。

- \e[：指示颜色提示符的开始；
- 0;34m：颜色代码，此代码代表蓝色，编码格式是 x;ym；
- \e[m：指示颜色提示符的结束。

部分颜色代码如下。

- 0;30：黑色；
- 0;34：蓝色；
- 0;32：绿色；
- 0;36：青色；
- 0;31：红色；
- 0;35：紫色；
- 0;33：褐色。

例如，想让使用 root 登录时提示符显示为红色，可以将如下语句加入文件/root/.bashrc 的底部：

```
export PS1="\e[0;31m[\u@\h \W]\\$ \e[m"
```

注意：如果想要每次登录时自动设置 Shell 提示符，则需要将环境变量 PS1 放在~/.bashrc 文件中，并使用 export 命令将其输出到其他子命令中。

同样可以使用 tput 命令修改提示符的设置。例如，使用 tput 命令显示红色的提示符：

```
export PS1="\[`tput setaf 1`\]\u@\h \W]\\$ \[`tput sgr0`\]"
```

下面是部分简单易用的 tput 命令行选项。

- tput bold：设置粗体模式。
- tput rev：显示反转颜色。
- tput sgr0：关闭所有属性。
- tput setaf：设置前景色，颜色代码参见表 2.4。
- tput setab：设置背景色，颜色代码参见表 2.4。

用于 tput 命令的各种颜色代码如表 2.4 所示。

表 2.4　转义字符

颜 色 代 码	颜　　色
0	黑色
1	红色
2	绿色
3	黄色
4	蓝色
5	洋红色
6	青色
7	白色

下面再看一个使用 tput 命令改变 Bash 提示符颜色的例子：

```
$ export PS1="\[`tput bold``tput setb 2``tput setaf 7`\]\u@\h:\w $ \
[`tput sgr0`\]"
```

上面的命令是将 Bash 提示符中的所有字体加粗，背景设为绿色，前景设为白色。

我们再看一个复杂一些的变量 PS1 的设置，让 Bash 提示符看上去更专业，命令如下：

```
$ export PS1="`tput setaf 7`\342\224\214\342\224\200\$([[ \$? != 0 ]] &&
echo \"[`tput setaf 1`X`tput setaf 7`]\342\224\200\")[$(if [[ ${EUID} ==
0 ]]; then echo '`tput setaf 1`\h'; else echo '`tput setaf 3`\u`tput setaf
7`@`tput setaf 2`\h'; fi)`tput setaf 7`]\342\224\200[`tput setaf 2`\w`tput
setaf 7`]\n`tput setaf 7`\342\224\224\342\224\200\342\224\200\342\225\274
`tput sgr0`"
```

注意：本例中的 Bash 提示符，你需要将你所用终端的字符集编码设为 UTF-8 才能正常显示，否则可能显示的是乱码。

2.4.5　实例：设置 Shell 选项

本节主要介绍如何使用 Bash 的内置命令 set 和 shopt 控制 Bash 的行为。

你可以通过开启或关闭 Bash 的相关选项控制 Bash 的行为，不同的选项使用不同的开启和关闭的方法。Bash 内置命令 set 控制一组选项，而 Bash 内置命令 shopt 控制另一组选项。

下面查看 set 命令可以设置哪些 Bash 选项：

```
$ set -o
allexport      off
braceexpand    on
emacs          on
errexit        off
errtrace       off
functrace      off
hashall        on
histexpand     on
history        on
ignoreeof      off
interactive-comments    on
```

```
keyword        off
monitor        on
noclobber      off
noexec         off
noglob         off
nolog          off
notify         off
nounset        off
onecmd         off
physical       off
pipefail       off
posix          off
privileged     off
verbose        off
vi             off
xtrace         off
```

关于 Bash 选项的详细信息，请参考 set 命令内置的帮助信息。

从命令的输出中可以看出，只使用命令 set -o 不跟任何选项将列出由 set 命令控制的每个 Bash 选项及其当前的状态（开启或关闭）。

如果要开启一个 Bash 选项，则输入如下命令：

```
$ set -o freature-name
```

关闭刚才开启的选项的命令如下：

```
$ set +o freature-name
```

通过上述实例可以知道，使用命令 set -o 或 set +o 后跟选项名可以开启和关闭特定的 Bash 选项。

例如，开启 ignoreeof 选项，此选项可以使用于退出登录 Shell 的 Ctrl+D 组合键失效，命令如下：

```
$ set -o ignoreeof
```

现在尝试按 Ctrl+D 组合键，类似如下：

```
$ Use "exit" to leave the shell.
```

再将此选项关闭，输入命令如下：

```
$ set +o ignoreeof
```

现在尝试按 Ctrl+D 组合键，就可以正常退出 Shell 了。

查看由 Bash 内置命令 shopt 控制的 Bash 选项及其状态，命令如下：

```
$ shopt
cdable_vars    off
cdspell        off
checkhash      off
checkwinsize   off
cmdhist        on
dotglob        off
execfail       off
expand_aliases  on
extdebug       off
extglob        off
extquote       on
failglob       off
```

```
force_fignore   on
gnu_errfmt      off
histappend      off
histreedit      off
histverify      off
hostcomplete    on
huponexit       off
interactive_comments    on
lithist         off
login_shell     on
mailwarn        off
no_empty_cmd_completion off
nocaseglob      off
nocasematch     off
nullglob        off
progcomp        on
promptvars      on
restricted_shell        off
shift_verbose   off
sourcepath      on
xpg_echo        off
```

关于各选项的详细信息，请参考 shopt 命令内置的帮助信息。

使用 shopt 命令开启和关闭 Bash 选项的语法如下：

```
shopt -s feature-name               #开启一个 Bash 选项
shopt -u feature-name               #关闭一个 Bash 选项
```

shopt 命令中有一个 cdspell 选项，用于监测 cd 命令中目录名的拼写错误情况并予以纠正。错误检查包括调换的字符、缺少的字符和重复的字符。例如，输入一个命令如下：

```
$ cd /var/lid
```

将会得到如下输出：

```
-bash: cd: /var/lid: No such file or directory
```

现在开启 cdspell 选项，然后再输入一次同样的命令：

```
$ shopt -s cdspell
$ cd /var/lid
/var/lib
$ pwd
/var/lib
```

注意：选项 cdspell 只在交互式 Shell 中有效。

当然，可以使用命令 shopt 和 set 为自己定制一个 Bash 环境，编辑~/.bashrc 文件，可以添加如下命令：

```
#纠正目录拼写
shopt -q -s cdspell

#当终端窗口大小改变时，确保显示得到更新
shopt -q -s checkwinsize

#开启扩展模式匹配特性
shopt -q -s extglob
```

```
#退出时追加而不是重启命令历史
shopt -s histappend

#使 Bash 尝试保存历史记录中多行命令的所有行
shopt -q -s cmdhist

#及时得到后台任务结束的通知
set -o notify
```

此外，还可以定制系统范围的 Bash 环境。默认情况下，文件/etc/profile 作为 Bash 的系统范围用户参数文件。可以强制设置对所有用户使用这个文件，但是，在 CentOS、Fedora 和 Red Hat 企业级 Linux 中推荐的方法是使用目录/etc/profile.d 中的文件。

2.5　小　　结

下面总结本章所学的主要知识：

- Bash 是大多数 Linux 系统的默认 Shell，它与原来的 Bourne Shell 向后兼容，并且融合了一些有用的 Korn Shell 和 C Shell 的特性。
- 用户登录时，登录 Shell 调用的初始化文件和脚本的次序依次是：/etc/profile、/etc/profile.d 目录下的脚本、$HOME/.bash_profile、$HOME/.bashrc 和/etc/bashrc。
- 一旦修改~/.bashrc 文件，必须重新登录 Shell，新的设置才会生效。
- 当登录 Shell 退出时，如果$HOEM/.bash_logout 脚本存在，则 Bash 会读取并执行此脚本的内容。
- Shell 中有两种变量类型：系统变量（环境变量）和用户自定义的变量（本地变量或 Shell 变量）。
- 在 Bash 中定义变量和赋值的方法是：varName=varValue，在赋值操作符“=”的前后，不要有空格。
- Bash 变量名可以包含字母、数字和下画线“_”，但首字符必须是字母或下画线。
- 当引用一个变量时，通常是用双引号将变量名括起来，这样可以防止被引用的变量值中的特殊字符（除$、`和\）被解释为其他错误含义。
- Bash 的内置命令 export 会将指定给它的变量或函数自动输出到后续命令的执行环境中。
- 在 Bash 中使用 unset 命令可以删除相应的变量或函数。
- 使用 history 命令可以显示在命令行提示符状态下输入的命令列表。
- Shell 中的扩展方式有 8 种，分别是（按扩展的先后顺序排序）：花括号扩展、波浪号扩展、参数和变量扩展、命令替换、算术扩展、进程替换、单词拆分和文件名扩展。
- Bash 的内置命令 alias 用于创建一个别名。
- 如果想每次登录时自动设置 Shell 提示符，则需要将环境变量 PS1 放在~/.bashrc 文件中，并使用 export 命令将其输出到其他子命令中。
- Bash 的内置命令 set 和 shopt 可以用于设置 Shell 选项。

2.6　习　　题

一、填空题

1．Bash 是_____的缩写，它是一个与_____兼容，执行从标准输入设备或文件中读取命令的_____。

2．Linux 环境由四部分构成，分别为_____、_____、_____和_____。

3．Shell 有两种变量类型，分别为_____和_____。

二、选择题

1．下面用来搜索命令的路径的系统变量是（　　）。

A．HISTFILE　　B．PWD　　C．PATH　　D．SHELL

2．下面（　　）变量名是有效的。

A．?name=bob　　B．name=bob　　C．_name=bob　　D．*name=bob

3．当引用一个变量时，最好使用（　　）符号将变量名括起来。

A．" "　　B．' '　　C．{}　　D．()

三、判断题

1．当用户定义变量时，赋值操作符“=”的前后不要有空格。（　　）

2．定义变量时，变量名不区分大小写。（　　）

3．用户在命令行创建的别名为临时别名。当用户退出 Shell 后，其别名失效。（　　）

四、操作题

1．定义一个名为 username 的变量，其值为 test，然后使用 echo 命令输出变量 username 的值。

2．查看历史命令列表，并通过行号重新调用相应的历史命令。