Linux操作系统基础教程

任课教师 刘昊霖
联系方式： liuhaolin@xtu.edu.cn

1

目 录 CONTENTS

2021/12/2

2

课程安排

课程安排：

理论课：2～7周（24个学时）

实验课： （8个学时）

考核形式：

总成绩=期末考试*60% +实验课*30% +考勤*10%

先导课

计算机组成原理、操作系统、C语言

数据结构、网络原理、程序设计基础

2021/12/2

3

教学目的

- 理解：Linux “一切都是文件”的特点
- 通过分析Linux系统，加深对操作系统的理解
- 熟悉并掌握利用Linux内核的基本功能，为以后在内核逻辑上解决问题打下基础（例如：高性能双机热备）
- 提高专业素养、锻炼实践动手能力

2021/12/2

4

教学目的

- 深化对操作系统的理解
  - 承上启下，认识对底层硬件的使用和驱动机制
  - 保护机制、中断机制、虚拟存储
- 潜移默化的专业修养
  - 专业的方法和技巧，效率＋可维护性
- Linux为我所用
  - 使用管理：服务器配置、网络安全管理
  - 开发工作：用作专用功能服务器、设备驱动

2021/12/2

5

推荐书籍

- Linux高级程序设计第三版 杨宗德 编著 ，人民邮电出版社
- 《鸟哥的Linux的私房菜 基础学习篇》 鸟哥、许伟、林彩娥等编著
- 《linux体系与编程—基于linux0.01版本》卢军、曾茂城编著
- 《Linux编程从入门到精通》 宫虎波编著
- 《Linux体系与编程：基于Linux0.01版本》荣智涛 中国水利水电出版社

2021/12/2

6

第1章 Linux概述

《Linux操作系统基础教程》

7

第1章 Linux概述

8

1.1.1 Linux的含义

Linux 是一款足以和微软公司的 **Windows** 相抗衡的开源操作系统， 在学习它之前，需要简单了解其含义、产生以及发展等知识

严格意义上讲，**Linux** 是在 **GPL** 协议（**GNU General Public License**，**GNU** 通用公共许可协议）下发行的遵循 **POSIX**（**Portable Operating System Interface of UNIX**，可移植性操作系统）标准的操作系统内核。 但通常所说的 **Linux** 是基于 **Linux** 内核，并且使用 **GNU**（**GNU's not UNIX**）工程各种工具和数据库的操作系统。即通常所说的 **Linux** 包含 内核（**Kernel**）和建立在内核基础上的各种系统工具程序（**Utilities**） 与应用软件（**Applications**），而不是仅指 **Linux** 系统内核。

Linux 是一套免费使用和自由传播的类 **UNIX** 操作系统，它是由世 界各地成千上万的程序员设计实现的，其目的是建立不受任何商品化 软件的版权制约及全世界都能自由使用的 **UNIX** 兼容产品。**Linux** 以它 的高效性和灵活性著称，它是 **UNIX** 的“克隆”。在源代码级上，它兼 容绝大部分的 **UNIX** 标准（如 **IEEE POSIX**、**System V**、**BSD** 等），具 有支持多任务、多用户、多线程和多 **CPU** 的能力。

9

1.1.2 Linux的出现

Linux 诞生于 **1991** 年 **10** 月 **5** 日，是由芬兰人 **Linus Benedict Torvalds**（李纳斯·托 沃兹，如图所示）创造的一款开源、免费的操作系统。**Linux** 的诞生、发展和成长 过程始终依赖 **5** 个重要支柱：**UNIX** 操作系统、**MINIX** 操作系统、**GNU** 计划、**POSIX** 标准和 **Internet**。

10

1.1.2 Linux的出现

20 世纪 **80** 年代，**IBM** 公司推出了享誉世界的微型计算机（**IBM PC**）。随着 **PC** 的 出现，在 **PC** 上建立 **UNIX** 系统成为可能。**Linux** 的出现早开始于 **Linus Benedict Torvalds**，当时他的目的是设计一个代替 **Minix**（一位名叫 **Andrew Tannebaum** 的计算机 教授在 **1987** 年为了方便教学而自行设计的一个简化的 **UNIX** 系统）的操作系统，由此 开始了 **Linux** 的雏形设计。**1991** 年 **10** 月 **5** 日，**Linus Torvalds** 在 **comp.os.minix** 新闻组 上对外宣布 **Linux** 内核诞生。

Linux 的历史是和 **GNU** 紧密联系在一起的。**GNU** 从 **1983** 年开始计划致力于开发一 个自由并且完整的类 **UNIX** 操作系统，包括软件开发工具和各种应用程序。到 **1991** 年 **Linux** 内核发布的时候，**GNU** 几乎已经完成了除系统内核之外的各种必备软件的开发。 在 **Linus Torvalds** 和其他开发人员的共同努力下，**GNU** 组件可以运行于 **Linux** 内核之上。 即 **Linux** 整个内核是基于 **GNU** 通用公共许可，也就是 **GPL** 的，但是 **Linux** 内核并不是 **GNU** 计划的一部分。

1994 年，在北卡罗莱州的一个小组程序员开始发布 **Red Hat**，它**GNU/Linux** 为核 心，集成了 **400** 多个源代码开放的程序模块。**1996** 年，**Linux** 系统开始在世界范围内广泛应用，其创始人 **Linus** 选择企鹅图案作为 **Linux** 的**logo**，如下图所示

11

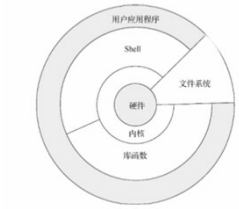
1.1.2 Linux的出现

其含意为： 开源的 **Linux** 为全人类共同所有，任何公司无权将其私有。**1998** 年，小红帽高级研发 实验室成立。目前很多门户网站，如新浪、搜狐、腾讯等均在 **Linux** 下运行，同时 **Android** 和 **IOS** 也是基于 **Linux** 内核的操作系统。

12

1.2.1 Linux体系结构

Linux 系统一般由 **4** 个主要部分组成：内核、**Shell**、文件系统和应用程序。内核、 **Shell** 和文件系统一起构成基本的操作系统结构，用户可以运行程序、管理文件和使用 系统。部分层次结构如图所示。

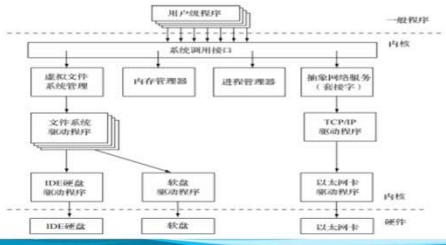


13

1.2.1 Linux体系结构

1. Linux 内核

Linux 内核是操作系统的核心，具有很多基本功能，它负责管理系统的进程、内存、设备驱动程序、文件和网络系统，决定系统的性能和稳定性。**Linux** 内核由以下几部分 组成：内存管理器、进程管理器、设备驱动程序、虚拟文件系统（**Virtual File System**， **VFS**）和网络管理等，如图所示。



14

1.2.1 Linux体系结构

（**1**）内存管理器：内存管理器主要提供对内存资源的访问控制。**Linux** 系统会在硬 件物理内存和进程使用的内存（称作虚拟内存）之间建立一种映射关系。这种映射是以 进程为单位，因而不同的进程可以使用相同的虚拟内存，而这些相同的虚拟内存，可以 映射到不同的物理内存上。

（**2**）进程管理器：进程实际上是某特定应用程序的一个运行实体。在 **Linux** 系统中， 能够同时运行多个进程，**Linux** 通过在短时间间隔内轮流运行这些进程而实现“多任务”。其中短时间间隔称为“时间片”，让进程轮流运行的方法称为“进程调度”，完成调度的 程序称为调度程序。进程调度主要提供对 **CPU** 的访问控制。在计算机中，**CPU** 资源是 有限的，而众多的应用程序都要使用 **CPU** 资源，因此需要“进程调度子系统”对 **CPU** 进行调度管理。进程管理的重点是创建进程和停止进程，并控制它们之间的通信（**signal** 或者 **POSIX** 机制）

（**3**）设备驱动程序：设备驱动程序是 **Linux** 内核的主要部分。设备驱动程序实际控 制操作系统和硬件设备之间的交互，并且提供一组操作系统可理解的抽象接口，完成和 操作系统之间的交互，与硬件相关的具体操作细节也由设备驱动程序完成。

（**4**）虚拟文件系统：**VFS** 隐藏各种文件系统的具体细节，为文件操作提供统一的接口。

15

1.2.1 Linux体系结构

（**5**）网络管理：网络管理在 **Linux** 内核中主要负责管理各种网络设备，并实现各种 网络协议栈，终实现通过网络连接其他系统的功能。

2. Linux Shell

Shell 是系统的用户界面，提供了用户与内核进行交互操作的一种接口。**Shell** 也是 一个程序，它接收键盘输入的命令，并传递给操作系统执行。用户输入命令后交由 **Shell** 处理，**Shell** 再与操作系统内核取得通信，调用完成用户命令所需执行的程序。由 **Shell** 的字面意思很自然地把它和操作系统内核 **Kernel** 联系在一起。如果把操作系统想象成 一个坚果，内核就是坚果的种子部分，**Shell** 就是坚果的外壳部分，用户命令需要通过 **Shell** 的传递才能达到内核调用相应的程序。由此可以把 **Shell** 视为一种 命令解析器，它解释由用户输入的命令并将它们送到内核，在用户命令和系统内核之间 建立了桥梁。这种在用户和系统之间建立交互的方式与图形用户界面（**Graphical User Interface**，**GUI**）非常相似，实际上 **GUI** 就是一种 **Shell**，只是习惯上我们仅把命令行称 为 **Shell**。另外，**Shell** 编程语言具有普通编程语言的很多特点，用这种编程语言编写的 **Shell** 程序与其他应用程序具有相同的效果。在没有特别说明的情况下，本书提到的 **Shell** 特指命令行形式的 **Shell**。

16

1.2.1 Linux体系结构

Shell 分为两种，一种是命令行界面（**Command Line Interface**，**CLI**），它是指可在 用户提示符下键入可执行指令，通常不支持鼠标。用户通过键盘输入命令，然后由 **Shell** 解释用户输入的命令并将它们送到内核。另一种是 **GUI**，它是指采用图形方式显示的操作用户界面。

在 **Linux** 下可用的 **Shell** 有很多种，常见的几种是 **Bourne Shell**、**bash**、**C Shell** 和 **K Shell**。

（**1**）**Bourne Shell**：初的 **UNIX Shell** 是由 **Stephen R·Bourne** 于**20**世纪**70**年代中期在新泽西的 **AT&T** 贝尔实验室编写的，即 **Bourne Shell**。**Bourne Shell** 是一个交换式的命令解释器和命令编程语言。**Bourne Shell** 可以运行为 **login shell** 或者 **login shell** 的子 **Shell**（**Subshell**）。只有 **login** 命令可以调用 **Bourne Shell** 作为一个 **login shell**。此时，**Shell** 先读取**/etc/profile** 文件和**$HOME/.profile** 文件。**/etc/profile** 文件为所有的用户定制环境，**$HOME/.profile** 文件为本用户定制环境。最后，**Shell** 会等待读取输入。

（**2**）**bash**：**bash Shell**（**Bourne Again Shell**）是 **Bourne Shell** 的一个免费版本。**bash** 是大多数 **Linux** 系统默认使用的 **Shell**，也是 **GNU** 计划为了改善 **Bourne Shell** 用户交互方面的不足而创建的。它包含很多优秀的功能，如命令补全、命令历史记录、可用 **help** 命令查看帮助等。本书使用的 **Shell** 就是 **bash**。注意：若书中未声明，那么提到 **Shell** 即为 **bash**。

17

1.2.1 Linux体系结构

（**3**）**C Shell**（包括 **csh**、**tcsh**）：是一种非常适合于编程的 **Shell**，由 **Bill Joy**于**20** 世纪**80** 年代早期开发的，其语法与**C**语言风格接近。后来出现的**Tc Shell** 是 **C Shell** 的一个增强版本，与**C Shell** 完全兼容。**Tc Shell**功能十分强大，具有命令行编辑、可编 程单词补全、拼写校正、作业控制等功能。

（**4**）**K Shell**（**Korn Shell**）：由 **AT&T** 贝尔实验室的**David Korn**开发。**K Shell**继承了**C Shell**和**Bourne Shell**的优点。与**bash**一样，它不仅是命令解析器，还是一种命令编程语言。**K Shell**具有支持任务控制、进程协作、行内编辑、后台执行等功能。

（**5**）其他**Shell**：**z Shell**（简称 **zsh**），它是 **Korn Shell**的一个增强版本，具备**bash Shell**的许多功能及特色，同时也是大的**Shell**之一，由**Paul Falstad**完成，共有**84**个内部命令。如果只是一般的用途，没有必要安装这样的**Shell**。**POSIX Shell**是**Korn Shell**的 一个变种，当前提供**POSIX Shell**的大卖主是**Hewlett-Packard**（**HP**）公司。

3. Linux 文件系统

Linux 具有“一切皆文件”的特点。文件系统是文件存放在磁盘等存储设备上的组织方法。**Linux** 系统支持目前流行的多种文件系统，如 **EXT2**、**EXT3**、**EXT4**、**FAT**、**FAT32**、**VFAT** 和 **ISO9660** 等，但不支持 **Windows** 的主流文件系统 **NTFS**。

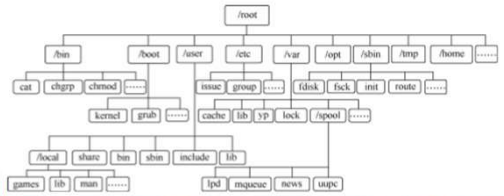
18

1.2.1 Linux体系结构

（1）Linux 文件结构

文件结构是指文件在存储设备中的组织方式，主要体现在对文件和目录的组织上，目录提供了管理文件的有效而方便的途径。

Linux 使用倒立的树形目录结构，在安装系统时，安装程序已经为用户创建了文件系统和完整而固定的目录组成形式，并指定每个目录的作用和其中的文件类型，如图下所示。每个目录的详细功能在第 3 章介绍。



19

1.2.1 Linux体系结构

（2）Linux 文件系统

文件系统是指文件存在的物理空间。使用文件系统，用户可以管理各项文件及目录资源。Linux 系统中的每个分区都是一个文件系统，都有自己的目录层次。Linux 将这些属于不同分区的单独文件系统按照一定的方式，形成一个系统的总的目录层次结构。一个操作系统的运行离不开对文件的操作，因此必然要拥有并维护自己的文件系统。

4. Linux 应用程序

经过 20 余年的发展和积累，在自由软件世界中不断努力的软件开发人员为开发源 码领域贡献了无数优秀的应用程序。Linux 操作系统下的应用软件已经非常丰富，不仅 功能全面，而且性能卓越。Windows 操作系统中的大多数常用应用程序，在 Linux 平台 中都可以找到对应的软件，而且 Linux 部分软件的功能和性能甚至已经超越了 Windows 平台的同类产品。

标准的 Linux 系统一般都有一套称为应用程序的程序集，它包括文本编辑器、编程 语言、X Window、办公套件、Internet 工具和数据库等。

20

1.2.2 Linux系统特点

Linux 源于 UNIX，从一开始就继承了 UNIX 的先进性，但其是一个真正免费的、开源的操作系统。它充分利用了现行 CPU 的任务切换功能，创造了多任务、多用户环境，允许多个用户同时使用一个计算机系统。同时，多个用户可以从相同或不同的终端上用同一个应用程序的副本工作，真正实现了多用户的并行操作。与以往操作系统的不同之处在于，它采用了抢先式多任务的机制，保证每一个程序都有机会运行，每个程序一直执行到操作系统抢占 CPU 让其他程序执行为止，这种机制让 CPU 的功能发挥出最大的作用。Linux 的每个进程都运行在自己的虚拟地址空间中，并且不会损坏其他进程或内核使用的地址空间。

Linux 系统是单内核，这种内核比微内核复杂。在这种内核中，大量的功能放在内核中直接实现。而在微内核系统中，许多功能是采用服务进程的形式放在内核外实现的。Linux 的任务与内核之间也是相互隔离的，即使行为不良或程序编写不良也不会损坏系统。

Linux具有严密的文件及目录结构。文件都是按照作用或者性质来存放的，其目录结构是标准的倒立的树状结构。此外，Linux将所有硬件设备都作为文件来处理。这样，要使用某一设备时，只需要简单读写该设备文件即可，极大方便了设备的使用。

21

1.2.2 Linux系统特点

Linux 采取了许多安全技术，包括对读写的控制、带保护的子系统、审计跟踪、核心授权等，这为网络多用户环境中的用户提供了必要的安全保障。

Linux 系统具有很强的适应性。Windows 操作系统只能运行在 Intel 处理器上，各厂商的 UNIX 只能运行在各自的处理器上，但是 Linux 系统几乎能运行在所有常见的处理器上。Linux 还支持广泛的外部设备，因此在 Linux 中几乎可以找到所有的设备驱动程序。

归结起来，Linux 操作系统主要具有以下特点：开放性、多任务和多用户、支持多种硬件平台、可靠的安全系统、良好的用户界面、强大的网络功能、设备独立、支持多种文件系统、良好的可移植性。Linux 的价格优势也是毋庸置疑的，但是其稳定性、可靠性才是其得到广泛使用的主要原因。

22

1.3 Linux的发行版本

Linux 从创立至今 20 余年，一直倡导开放与自由，因此拥有众多的发行版本（Linux Distributions）。下面简单介绍常见的 Linux 发行版本。

1. Red Hat

Red Hat Linux 由 Red Hat 公司发行，是著名的 Linux 发行版，诞生于 1994 年。 Red Hat 系列的包管理方式采用基于 RPM 包的 YUM 包管理方式。包分发方式是编译好的二进制文件，此管理系统长期以来都是业界的事实标准（事实标准是指非由标准化组织制定，由处于技术领先地位的企业、企业集团制定，由市场实际接纳的技术标准）。Red Hat 可以说是国内使用多的 Linux 版本，这个版本的特点就是使用人数多，资源多，而且网上的许多 Linux 教程也都以 Red Hat 为例进行讲解。2003 年 9 月 22 日，原来合并在一起的 Fedora 和 Red Hat 开始分开发行，并形成两个分支：开源免费的 Fedora 和商业版本的 RHEL（Red Hat Enterprise Linux）。

2. CentOS

CentOS（Community Enterprise Operating System）在 2003 年底推出，它由 Red Hat Enterprise Linux 依照开放源代码规定释出的源代码编译而成。由于出自同样的源代码，因此有些要求高度稳定性的服务器以 CentOS 替代商业版的 RHEL 使用。两个发行版之间唯一的区别是品牌，CentOS 是一个基于 Red

23

1.3 Linux的发行版本

提供的可自由使用源代码的企业级 Linux 发行版本，不需要付任何服务费用。RHEL 是很多企业采用的 Linux 发行版本， 需要向 Red Hat 付费才可以使用，并能得到相应的服务、技术支持和版本升级。CentOS 能提供及时的、安全更新的所有套装软件升级为目标的社区项目。

3. Debian Debian

首次公布于 1993 年，它的目标是提供一个稳定容错的 Linux 版本。其创始人为美国普渡大学的一名学生 Ian Murdock。Ian Murdock 初把他的系统称为“Debian Linux Release”。Debian 基于 Linux Kernel，并且大部分基础的操作系统工具来自于 GNU 工程，因此又称为 GNU/Linux。Debian GNU/Linux 附带了29000多个软件包，因此获得了开源社区的普遍支持。目前采用的 deb 包和 Red Hat Linux 的 RPM 包是 Linux 中最为重要的两个软件包管理系统。

4. Ubuntu

Ubuntu 于 2004 年 9 月首次公布，是以桌面应用为主的 Linux 操作系统。Ubuntu 基于 Debian 的 unstable 版本加强而来，形成了完善的、近乎完美的 Linux 桌面系统。其运作主要依靠 Canonical 有限公司的支持。根据选择的桌面系统不同，有 3 个版本可供选择：基于 Gnome 的 Ubuntu、基于 KDE 的 Kubuntu 和基于 Xfc 的 Xubuntu。Ubuntu 的特点是界面友好、容易上手、对硬件的支持非常全面，是适合做桌面系统的 Linux 发行版本。

24

1.3 Linux的发行版本

5．Gentoo

Gentoo 是 **Linux** 较为年轻的发行版本，因此吸取了之前发行版本的优点，这也是 **Gentoo** 被称为完美的 **Linux** 发行版本的原因之一。**Gentoo** 初由 **DanielRobbins**（**Stampede Linux**和 **FreeBSD**的开发者之一）创建。由于开发者熟识**FreeBSD**，所以**Gentoo** 拥有媲美 **FreeBSD** 的广受美誉的 **ports** 系统——**Portage** 包管理系统。**Gentoo** 是一个十分特殊的 **Linux** 发行版，因为它是一种基于源代码的发行版，虽然可以使用编译好的二进制软件，但是大部分使用 **Gentoo** 的用户都选择自己动手编译软件管理系统。**Gentoo** 是所有 **Linux** 发行版本中安装复杂的，但又是安装完成后便于管理的版本，也是在相同硬件环境下运行快的版本。

6．Slackware

Slackware Linux 由 **Patrick Volkerding** 创建于 **1993** 年，是现存古老的 **Linux** 发行版之一。**Slackware Linux** 是一个高度技术性的、干净的发行版，只有少量非常有限的个人设置。它使用简单，基于文本的系统安装和比较原始的包管理系统，没有解决软件的依赖关系（**Linux** 中的软件依赖关系是拓扑树结构，如 **A** 直接或间接依赖 **B**，**B** 就不可能直接或间接依赖 **A**。试想从时间上，**A**、**B** 必然有一个先出现，而先出现的不可能依赖于后出现的，因此必然有一个先出现而另一个依赖于先者）。因此，**Slackware** 被认为 是纯净和不稳定的发行版。

25

1.3 Linux的发行版本

7．Mandriva

Mandriva Linux 由 **Gaël Duval** 于 **1998** 年 **7** 月在 **Mandrake Linux** 下发起。起初，它只是重新优化了包含更友好 **KDE** 桌面的 **Red Hat Linux** 版本，但后续版本增加了更友 的体验，如一个新的安装程序、改进的硬件检测等。由于这些改进的结果，**Mandriva Linux** 得以蓬勃发展。**Mandriva Linux** 主要偏向于桌面版本。其大特点在于它是高级软件， 拥有一流的系统管理套件（**DrakConf**）和优秀的 **64** 位版本支持，并且得到广泛的国际支持。

26

1.4 Linux的主要应用领域

由于 **Linux** 开放源代码，降低了对封闭源代码软件潜在安全性的忧虑，这使得 **Linux** 操作系统拥有广泛的应用领域。目前，**Linux** 的应用领域主要包括以下几个方面。

1．桌面应用领域

目前，**Windows** 操作系统在桌面应用中一直占据绝对的优势，但是随着 **Linux** 操作系统在图形用户界面和桌面应用软件方面的发展，**Linux** 在桌面应用方面也得到了显著的提高，越来越多的桌面用户转而使用 **Linux**。不过，**Linux** 在桌面应用市场上的占有率不高。如今新版本的 **Linux** 系统特别在桌面应用方面进行了改进，达到了更高的水平，完全可以作为一种集办公应用、多媒体应用、网络应用等多方面功能于一体的图形界面操作系统。

2．高端服务器领域

Linux 在服务器领域扮演领军者角色，这在很大程度上得益于它具有稳定性、安全性、开放源代码、总体拥有成本较低等优点。根据调查，**Linux** 操作系统在服务器市场上的占有率已超过 **50%**。由于 **Linux** 可以提供企业网络环境所需的各种网络服务，加上 **Linux** 服务器可以提供虚拟专用网络（**VPN**）或充当路由器（**Router**）与网关（**Gateway**）， 因此在不同操作系统相互竞争的情况下，企业只需要掌握 **Linux** 技术并配合系统整合与网络等技术，便能够享有低成本、高可靠性的网络环境。

27

1.4 Linux的主要应用领域

3．嵌入式应用领域

在通常情况下，嵌入式及信息家电的操作系统支持所有的运算功能，但是需要根据实际应用对其内核进行定制和裁剪，以便为专用的硬件提供驱动程序，并在此基础上开发应用。目前，能够支持嵌入式的常见操作系统有 **Palm OS**、嵌入式 **Linux**、**Android** 和 **Windows CE** 等。虽然 **Linux** 在嵌入式领域刚刚起步，但是 **Linux** 的特性正好符合 **IA**（基于 **Intel** 架构）产品的操作系统、稳定、实时与多任务等需求，而且 **Linux** 开放源 码，不必支付许可证费用。许多世界知名厂商包括 **IBM**、索尼等纷纷在其 **IA** 中采用 **Linux** 开发视频电话和数字监控系统等。

4．文件服务器系统

网络文件系统（**Network File System**，**NFS**）是由 **SUN** 公司制订的一种文件服务标准，它能实现基于 **Linux/UNIX** 的网络文件共享服务。应用 **Linux** 的 **NFS** 服务，可以很好地解决企业的 **Linux/UNIX** 环境文件共享问题。

28

1.4 Linux的主要应用领域

5．企业门户网站

所谓企业门户网站，就是为企业提供全面信息资讯和服务的行业性网站。在 **Linux** 下组建企业的门户网站，可以选择的方案很多，如著名的 **LAMP**（**Linux-Apache-MySQL- PHP**）方案。**LAMP** 网站架构是目前国际流行的 **Web** 框架，该框架包括：**Linux** 操作系统、**Apache** 网络服务器、**MySQL** 数据库、**Perl**、**PHP** 或者 **Python** 编程语言，其所有组成产品均是开源软件，是国际上成熟的架构框架，很多流行的商业应用都是采用此架构， 与 **Java/J2EE** 架构相比，**LAMP** 具有 **Web** 资源丰富、轻量、快速开发等特点；与微软公 司的**.NET** 架构相比，**LAMP** 具有通用、跨平台、高性能、低价格的优势。因此 **LAMP** 无论是性能、质量还是价格都是企业搭建网站的首选平台。

6．数据备份

对于企业来说，数据就是它的财产，因此数据备份的重要性不言而喻。 **Linux** 是非常安全的操作系统。在 **Linux** 新版本中，广泛采用日志文件系统，如 **EXT3**。它可以有效降低服务器在突然断电、死机等情况下，对数据可能造成的损失。在 **Linux** 下，也支持高性能的冗余磁盘阵列（**Redundant Arrays of Independent Disks**， **RAID**）、磁盘阵列等物理设备。应用 **RAID** 或者磁盘阵列，可以有效降低因物理存储介质失效带来的数据损失。

29

Linux操作系统基础教程

任课教师 刘昊霖

联系方式： liuhaolin@xtu.edu.cn

30

目 录 CONTENTS

2021/12/2

31

第2章 Linux的基本操作

《Linux操作系统基础教程》

32

第2章 Linux的基本操作

2021/12/2

33

第2章 Linux的基本操作

2021/12/2

34

2.1.1 虚拟机内安装Linux

1. 认识虚拟机

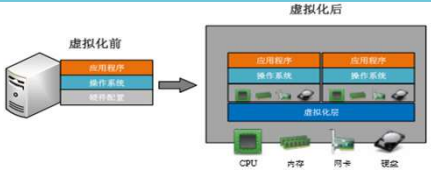
虚拟机是一个抽象的计算机，和现实世界的计算机一样具有一个指令集，并使用不同的存储区域。利用虚拟机技术，可以从原有系统中分割出一部分硬盘空间和内存容量，虚拟出“新机器”，这些新“新机器”拥有独立的 **BIOS**、硬盘等硬件资源，可以像对待实际的机器那样进行分区、格式化、安装操作系统和软件等操作，而不会对原有主机产生任何影响。同时虚拟机的使用可以更合理地利用资源，充分发挥计算机的效率。

虚拟机的实现依赖虚拟化技术，虚拟化是指在物理服务器上部署特定的虚拟化软件，通过该软件将物理资源进行逻辑化，实现了逻辑上的隔离，同时，在虚拟化层面部署相应的虚拟机，每个虚拟机类似于一个物理服务器，它们会通过虚拟化层（虚拟化层是由 **VMware** 设计用来运行虚拟机的内核，它控制 **ESXServer 3i** 主机使用的硬件，并调度虚拟机之间的硬件资源分配）得到相应的虚拟化硬件资源，如 **CPU**、内存、网卡、磁盘等资源。虚拟化过程如下图所示。

2021/12/2

35

2.1.1 虚拟机内安装Linux



常用的虚拟机有以下 **3** 种。

（1）**VMware Workstation**

VMware Workstation 是 **VMware** 公司开发的一款功能强大的桌面虚拟计算机软件， 提供了可在单一桌面上同时运行不同操作系统的解决方案，并可开发、测试、部署新的应用程序。**VMware Workstation** 可在一部实体机器上模拟完整的网络环境，以及可便于携带的虚拟机器，其更好的灵活性与先进的技术胜过了市面上其他的虚拟计算机软件。

2021/12/2

36

2.1.1 虚拟机内安装Linux

（2）Virtual Box

Virtual Box 是一款开源虚拟机软件。**Virtual Box** 由德国 **Innotek** 公司开发，由 **Sun Microsystems** 公司出品。**Virtual Box** 不仅具有丰富的特色，而且性能很卓越。可虚拟的操作系统包括所有的 **Windows** 版本、**Mac OS X**、**Linux**、**OpenBSD**、**Solaris**、**IBM OS2** 甚至 **Android** 等。

（3）Virtual PC

Virtual PC 是 **Microsoft** 公司新的虚拟化技术。**Virtual PC** 允许在一个工作站上同时运行多个 **PC** 操作系统。当用户转向一个新操作系统时，可以为运行传统应用提供安全的环境以保持兼容性。

2．安装镜像 CentOS 7

以下将在 **Windows** 环境下使用 **VMware Workstation 12** 演示 **CentOS 7** 的安装。

（1）打开 **VMware Workstation 12**，单击菜单栏“文件”→“新建虚拟机”，弹出向导对话框，为了简单起见，不对虚拟机的高级选项进行配置，即在弹出窗口中选择“典型”，然后单击“下一步”按钮，如下图所示。

37

2.1.1 虚拟机内安装Linux

（2）在弹出的对话框中选择事先准备好的 **CentOS 7** 系统镜像文件，单击“下一步” 按钮，如图所示。

38

2.1.1 虚拟机内安装Linux

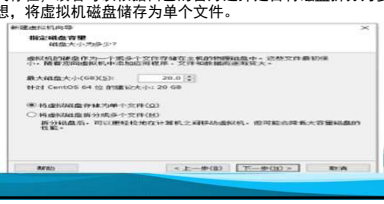
（3）在弹出的对话框中设置虚拟机的基本信息，如图所示。其中“全名”是为 **Linux** 系统起的别名，相当于 **Windows** 系统中的计算机名，如 **My CentOS**；“用户名” 为登录 **Linux** 系统的用户唯一标识，如 **user**。要注意的是，此时设置的密码与超级管理 员的登录密码相同，必须牢记。

（4）设置好以上信息后，单击“下一步”按钮，在弹出的对话框中设置 **CentOS 7** 在 **VMware** 中的名称和安装在物理机上的位置，单击“下一步”按钮，如下图所示。

39

2.1.1 虚拟机内安装Linux

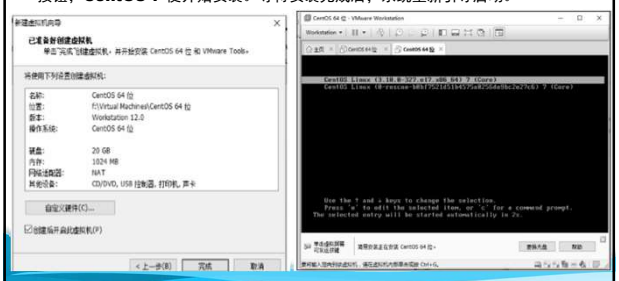
（5）在弹出的对话框中设置虚拟机磁盘，如图所示。虚拟机的磁盘将在物理机中以文件的形式存在，读者可以根据自己的喜好选择是否将磁盘拆分为多个文件。本书为了性能着想，将虚拟机磁盘储存为单个文件。



40

2.1.1 虚拟机内安装Linux

（6）单击“下一步”按钮后，弹出对话框提示虚拟机已准备好安装，并显示虚拟机信息。读者可以单击“自定义硬件”，根据自己机器的性能详细设置硬件。这里使用默认设置，关于 **Linux** 的配置会在 **2.4** 节中详细介绍。一切就绪后，单击“完成”按钮，**CentOS 7** 便开始安装。等待安装完成后，系统重新引导启动。



41

2.1.2 生产实践安装Linux

（1）将刻录了 **CentOS 7** 系统镜像文件的存储介质插入光驱或 **USB** 接口中，启动计算机并进入 **BIOS** 设置界面，将第一启动设备设置为刚才的存储设备。

（2）保存并重启计算机后，进入 **CentOS 7** 的安装提示界面。

（3）选择“**Install CentOS 7**”，按回车键后，进入安装前的配置界面，首先需要选择语言与键盘类型。

（4）下一步进入配置安装的汇总界面，在这个界面中可以设置大部分与安装有关的信息。下面介绍一些重要的设置。

42

2.1.2 生产实践安装Linux

1. 日期和时间设置

在 **LOCALIZATION** 栏下，语言（**LANGUAGE SUPPORT**）和键盘类型（**KEYBOARD**）已经设置过了，现在设置日期和时间。单击 **DATE&TIME**，选择所在的时区、日期和时间。若此时已经连接网络，系统将会使用 **NTP** 服务自动设置日期与时间，否则需要仔细手动设置，因为时间同步在服务器通信中十分重要。

2. 安全策略设置

在新的 **CentOS 7**（**1511**）版本中，新增了一个可供用户选择的安全策略设置。单击“**SECURITY POLICY**”，可以看到系统提供了一系列服务器安全的应用场景。这里只需选择默认策略，更详细的 **Linux** 安全配置见 **2.4** 节。

3. 选择需要安装的软件

在 **SOFTWARE** 栏下，单击 **SOFTWARE SELECTION** 可以选择随 **CentOS** 操作系统一起安装的软件和工具。默认情况下是小化安装（**Minimal Install**），而实际上我们需要一些能提高效率的工具和桌面环境，因此这里选择安装 **GNOME** 或 **KDE** 当中的一种桌面环境。

4. 磁盘划分

在 **SYSTEM** 栏下，选择 **INSTALLATION DESTINATION** 可以划分磁盘。

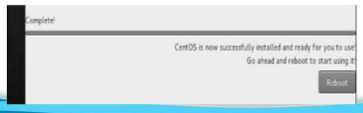
43

2.1.2 生产实践安装Linux

5. 网络设置

在 **SYSTEM** 栏下，选择 **NETWORK&HOST NAME** 可以设置网络。这里可以设置网络连接和主机名，详细的网络配置见 **2.4** 节。

当所有安装配置完成后，单击**Begin Installation**，**CentOS 7** 便开始安装。此时可以进行 **root** 密码设置和新增用户的操作。等待安装完成后，系统会做出相应提示，单击 **Reboot** 按钮系统重新引导启动，至此 **CentOS 7** 生产实践的安装便完成了。



44

2.2.1 GUI与X Window

如今，几乎所有计算机用户都在知情或不知情的情况下使用了 **GUI**。**Windows** 或 **Mac OS** 系统一启动就会进入一个图形界面，人们大部分的办公和娱乐都是在这个图形界面上进行的，这个图形界面就是 **GUI**。那么 **GUI** 究竟是什么呢？**GUI** 或者图形用户界面是一个将计算机的输出直接以图形形式显示在屏幕上，并可以使用键盘、鼠标等设备直接与计算机进行交互的程序。这里需要注意的是，**GUI** 是一种程序，在实现图形化交互时，必须与计算机硬件（如屏幕、键盘、鼠标等）通信，因此 **GUI** 依赖于各种设备驱动程序和底层系统，**X Window** 便是其中重要的一项。

X Window 是麻省理工学院于 **1984** 年提出的一个为程序提供图像数据服务的系统。**X Window** 提出了一个独立于硬件的图形界面标准，可以将大量异构的计算机硬件连接到同一个网络中。目前，**X Window** 几乎是所有操作系统 **GUI** 的基础。就**X Window**本身来说，它提供了 **GUI** 和硬件之间通信的协议，而图形界面终是什么样，用户如何与之交互，**X Window** 并没有参与，而是由另一个程序实现—窗口管理器（**Window Manager**）。窗口管理器控制窗口以及其他所有图形元素的外观和特征。当 **GUI** 需要显示图形界面时，窗口管理器会自动定义好如图标、按钮、窗体等各种图形的各项特征（颜色、形状、大小等），然后 **X Window** 与实际绘制图像的硬件进行通信，后将图形界面输出在屏幕上。这样就形成了一种层次调用的关系。

45

2.2.2 KDE桌面和GNOME桌面

1. KDE桌面环境

KDE 是 **1996** 年一位名叫 **Matthias Ettrich** 的德国学生启动的 **Kool Desktop Environment** 项目的缩写，如今已更名为 **K Desktop Environment**。**KDE** 项目创建时是为了在当时混乱的 **UNIX GUI** 环境下，提出一个完整统一的应用程序界面。**1997** 年，**KDE** 项目吸引了全世界大量程序员的关注，**Ettrich** 在开发 **KDE** 桌面环境中使用了 **Qt** 程序库（**Qt** 是由 **Trolltech** 公司开发的编程工具套件）。在 **KDE** 中包含的应用程序多以 **K** 开头，如文本编辑器 **Kate**、即时通信软件 **Kopete**、计算器 **KCalc**、媒体播放器 **Kaffeine** 等，甚至终端模拟器 **Console** 在 **KDE** 中都变成了 **Konsole**。

2. GNOME 桌面环境

GNOME 是 **1997** 年由 **Miguel de Icaza** 和 **Federico Mena** 两人发起的项目 **GNU Network Object Model Environment**，用于替代 **KDE** 桌面环境。由于 **KDE** 使用的 **Qt** 程序库的软件许可方式不允许用户用于商业用途，因此 **GNOME** 的意义在于独立于 **Qt** 程序库并可以自由发行。

与 **KDE** 的情况如出一辙，**GNOME** 中的应用程序多以 **G** 开头，如图像编辑器 **Gimp**、即时通信软件 **Gaim**、计算器 **Gcalctool**、电子表格软件 **Gnumeric** 等。

46

2.2.3 图形界面的基本操作

由于人们的文化与习惯问题，计算机操作系统的界面风格都很相似，熟悉 **Windows** 系统的读者很容易掌握 **Linux** 系统图形界面的操作。这里以 **GNOME** 为例，介绍 **Linux** 图形界面与 **Windows** 界面不同的地方，方便读者使用。

默认情况下，**GNOME** 有两个图标 **home** 和 **Trash**，类似于 **Windows** 的“我的电脑”和“回收站”，双击图标即可进入相应的目录或应用。

GNOME 的任务栏分为上下两部分，屏幕上方的任务栏显示当前激活的应用程序，屏幕下方的任务栏显示已经被打开的窗口。任务栏中的**Application** 下拉菜单类似于 **Windows** 的开始菜单，包含了许多应用程序，并以分类的方式显示。**Places** 下拉菜单则包含了当前用户的相关目录，如用户根目录（**Home**）、用户文档（**Document**）、用户下载目录（**Download**）等。

在**GNOME**中，可以在下方任务栏右侧的标签中切换工作空间。如在 **Workspace1** 中进行文件目录操作、在 **Workspace2** 中进行命令行操作、在 **Workspace3** 中浏览网页，使用 **win** 键可以查看各工作空间的情况。

47

2.3.1 进入Linux CLI

如果 **Linux** 系统本来就以命令行模式引导，则系统启动完成后将自动进入命令行模式。如果以图形桌面的方式启动系统，需要从桌面环境中进入终端仿真器，或按 **Ctrl+Alt+F2**（**F2**～**F7**）组合键切换到命令行模式。通过这两种方式都可以进入 **Linux CLI**。

在命令行模式或终端仿真器中可以看到如下字符。

[user@localhost ~]$

这表示系统已经准备好接收用户的命令，其中 **user** 为当前用户名，**localhost** 为主机名。～符号为当前工作目录，**$**符号为命令提示符。关于它们的知识将在后面的章节中介绍，这里只需明白**$**符号后是用户输入命令的地方。

当输入**date** 并按下回车时，**date** 这条命令交由 **Shell** 处理，经过一系列程序调用后，屏幕上输出了当前系统的时间。可以说 **CLI** 为用户提供了使用命令与 **Shell** 进行交流的环境。

48

2.3.2 Linux命令格式

在 **Linux** 中关于命令的格式有明确的定义，使用 **Linux** 命令时，必须严格按照命令的格式输入。通常命令由命令名（**command**）、选项（**options**）和参数（**arguments**）三部分组成。依次从左往右排列并以空格分隔，格式如下。

command options arguments

命令名是命令的标识，表示命令的基本功能。事实上 **Linux** 命令都是一个个程序， 命令名是程序所在的脚本名。用户输入命令时，**Shell** 会根据命令名到相应的位置搜索并执行程序。选项是命令执行的方式，参数是命令作用的对象。下面分别介绍它们， 后给出帮助文档的获取方式。

1. 选项

通常情况下，选项直接位于命令名之后，用连字符号“-”后跟一个字母表示。顾名思义，选项是可选的，并且不一定需要设置。不设置选项时，命令将采用默认的方式执行。一旦设置了选项，命令将按照选项的设置执行。

2. 参数

某些时候需要使用参数指定命令的作用对象，或为命令提供数据。仍然以 **date** 命令为例，使用**-d** 选项可以显示用户指定的时间，指定的时间以参数的形式给出。

3. 获取帮助

man 命令可以查询某个命令的帮助信息。**man** 命令的格式为：**man [option] filename**

49

2.3.3 命令行技巧

在使用命令行时，可能会遇到一些复杂的参数，或者需要输入多次较长的命令。许多时候，我们需要一些技巧来提升命令行的使用效率。这些技巧可能是 **Linux** 系统内置的或是 **bash** 专门提供的功能，以下将介绍这些常用的技巧。

1. **Tab**键自动补全

bash 提供了自动补全的功能，在输入命令时，按下 **Tab** 键，可以自动补全未输入的命令字符。

2. 命令历史记录

bash 会自动保存使用过的命令的历史记录。按下向上方向键时，会发现上一次输入过的命令再次出现在命令提示符后，这些就是命令的历史记录，使用向上或向下方向键可以在命令历史记录中来回翻阅。

3. 命令历史记录的扩展

使用叹号“！”后跟数字的方式，就可以将历史记录中的命令扩展到命令行中。

扩展用法	说明
!!	重复最后一个输入的命令，效果与向上方向键相同
!<a>	重复最后一个以字符 a 开头的命令
!?<string>	重复最后一个包含字符串 string 的命令
!<number>	重复历史记录中第 number 行的命令
!-< number >	重复之前的第 number 个命令

50

2.4.1 配置文件

大多数 **Linux** 程序或软件都会有配置文件，配置文件中包含了程序运行时所需的信息，通过编辑配置文件可以定制程序。在 **Linux** 系统中同样存在一些特定的配置文件，这些配置文件在系统引导时被调用，用来构建系统工作的基础环境。

从我们按下计算机电源到 **Linux** 准备好为用户工作的这一段时间，系统发生了以下一系列事情。

（**1**）当按下电源时，启动 **BIOS**（**Basic Input Output System**），**BIOS** 检测计算机各硬件。

（**2**）检测完成后，执行一个叫作 **Bootloader** 的程序，该程序读取了包括 **/boot/grub/menu.lst** 在内的各种配置文件，用来加载 **Linux** 内核。

（**3**）**Linux** 内核加载完成后，第一个运行的进程是**/sbin/init**，我们将它称为 **1** 号进程。**1** 号进程读取了**/etc/inittab** 中的内容，确定系统运行级别。通常 **Linux** 的运行级别为 **3** 或 **5**，其他运行级别是管理员有特定维护需要时使用的。

（**4**）为了完成系统的启动，**1** 号进程创建了许多子进程。最后执行**/bin/login** 程序等待用户登录。用户登录后，首先读取**/etc/profile** 和**/etc/bashrc** 文件，建立一个所有用户共享的初始环境；然后读取用户目录下**~/.bash_profile** 和**~/.bashrc** 文件，用于建立用户自定义的个人环境。

51

2.4.2 Linux网络配置

网络连接是操作系统中十分重要的一环。如今互联网高速发展，网络的使用和配置方面的知识显得尤为重要。由于计算机网络涉及的领域很广，完全可以成为一个单独的专业领域，其中的内容足以再写一本书。本节只着重讲解在 **Linux** 环境下网络配置的一 些常用命令。在学习本节的内容前，读者需具备基本的网络知识包括 **IP** 地址、**DNS**、 路由等概念。

默认情况下，在 **VMware** 中安装 **Linux** 时使用 **NAT** 网络链接模式。在这种模式下，不需要进行任何配置就可以通过物理机的网络访问公网，然而用户也可以进行具体的网络配置。在 **Linux** 桌面环境中单击任务栏右侧的下拉菜单，选择 **PCI Ethernet** 下的 **Wired Settings** 选项，弹出网络信息窗口。

在网络信息窗口中可以查看当前网络的基本信息，单击右下角的齿轮按钮进入网络配置窗口，选择左侧的“**IPv4**”选项卡，默认使用 **DHCP** 服务自动分配 **IP** 地址，将其改为 **Manual** 后就可以自由配置 **IP** 地址、**DNS**、网关等内容了。

配置好 **IP** 等信息后，使用 **service network restart** 命令重启网络服务使配置生效，期间会要求输入 **root** 用户密码，然后使用 **ifconfig** 命令查看当前网络配置情况。

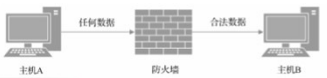
52

2.4.3 Linux防火墙设置

介绍了网络配置的相关知识后，不得不提及网络安全的问题。实际上，在使用任何计算机系统时，都应该考虑在安全的环境下建立网络连接。

关于防火墙的作用，一个例子就可以解释。假设网络中有两台相连主机 **A** 和 **B**，并且没有使用防火墙。当主机 **A** 向主机 **B** 发送数据时，无论发送的是什么数据，主机 **B** 都将全部接收。若这些数据中包含了大量的垃圾信息，或是某种具有攻击性的数据包，那么主机 **B** 将受到严重的安全威胁，如图所示。

使用防火墙后，上述情况将会有效避免。当主机 **A** 向主机 **B** 发送数据时，只有那些符合防火墙规则的数据才能通过并达到主机 **B**，不符合规则的数据将被过滤掉，如图所示。



53

2.4.3 Linux防火墙设置

在 **Linux** 中，常见的防火墙是 **Netfilter**，而防火墙的规则由 **iptables** 设置。**Netfilter** 是早期防火墙 **ipfwadm** 和 **ipchains** 的替代品，**Netfilter** 集成了它们的优点并添加了自己的特性，现在作为 **Linux** 的默认防火墙。**Netfilter** 和 **Linux** 是由两个不同的组织开发的，但 **Netfilter** 运行在 **Linux** 内核中。得益于 **Netfilter** 的开源，每一个 **Linux** 内核版本都可以将 **Netfilter** 编译到内核中。由于用户在使用防火墙时，常用的命令是 **iptables**，所以导致人们一度认为 **Linux** 的防火墙就是 **iptables**。其实 **iptables** 只是一个规则编辑工具，用户通过 **iptables** 将过滤规则写入 **Netfilter** 的规则数据库中。因此，**Linux** 防火墙应该称为 **Netfilter/iptables**。

从 **CentOS 7** 开始，默认不安装 **iptables**，使用一个基于 **iptables** 核心的新组件 **firewalld**，因此对防火墙的操作命令会与以往的 **CentOS** 版本有所不同，如果读者偏爱旧版本的命令，可以自行安装 **iptables** 组件。

几个常用的防火墙命令：

查看防火墙状态：**firewall-cmd --state**。

开启/关闭防火墙：**systemctl start/stop firewalld.service**。

54

2.4.4 系统日志

Linux 系统以及运行在其上的应用程序都会产生日志，这些日志记录了程序的运行状态，包括各种错误信息、警告信息和其他的提示信息。当系统发生故障时，可以通过系统日志快速定位故障发生的位置和原因。另外查看日志还可以发现一些潜在的威胁，如试图破解登录口令的动作。 通常 **Linux** 的日志文件存放在**/var/log** 目录下，**/var/log/messages** 日志文件是整体系统信息的汇总。

以**/var/log/secure** 日志文件为例，介绍如何通过查看日志文件排查系统问题。**/var/log/secure** 日志文件记录了所有登录验证信息，如果有人试图通过 **SSH** 远程登录系统，就会在日志中留下记录。使用 **grep "sshd" /var/log/secure | grep "Failed"**命令可以输出**/var/log/secure** 文件中所有 **SSH** 登录失败的记录，关于 **grep** 和管道的知识会在后续的章节中详细介绍。

55

Linux操作系统基础教程

任课教师 刘昊霖
联系方式： liuhaolin@xtu.edu.cn

56

目 录 CONTENTS

57

第3章 Linux文件系统与磁盘管理

《Linux操作系统基础教程》

58

第3章 Linux文件系统与磁盘管理

59

第3章 Linux文件系统与磁盘管理

60

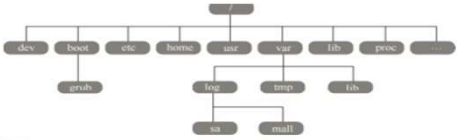
第3章　Linux文件系统与磁盘管理

61

3.1.1　Linux目录结构

Linux 系统以文件目录的方式组织和管理系统中的所有文件。所谓文件目录，就是将所有文件的说明信息采用树形结构组织起来。整个文件系统有一个“根（**root**）”，然后在根上分“杈（**directory**）”，任何一个分杈上都可以再分杈，杈上也可以长出“叶子”。“根”和“杈”在 **Linux** 中被称为“目录”或者“文件夹”。而“叶子”则是文件。这种结构的文件系统效率高，现代操作系统基本都采用这种结构方式。

通常 **Linux** 系统在安装后都会默认创建一些系统目录，以存放和整个操作系统相关的文件。**Linux** 系统树状目录结构如图所示。



62

3.1.1　Linux目录结构

系统目录及其说明如下。

1. /

根目录 **root**

即超级用户的主目录是/**root**。位于 **linux** 文件系统目录结构的顶层，它是整个系统最重要的目录，因为所有的目录都是由根目录衍生出来，它是 **Linux** 文件系统的入口，是最高一级的目录。

2. /dev

/**dev** 是 **device** 的缩写，这个目录下保存所有的设备文件，用户可以通过这些文件访问外部设备，如 **sda** 文件表示硬盘设备。并且该目录下有一些由 **Linux** 内核创建的用来控制硬件设备的特殊文件。

3. /boot

/**boot** 叫作引导目录，主要放置开机时会使用到的文档，即该目录下存放系统的内核文件和引导装载程序文件，例如，系统中非常重要的 **Linux** 内核 **vmlinux** 就放在该目录下。

4. /etc

/**etc** 保存绝大部分的系统配置文件，基本都是纯文本的，一般以扩展名**.conf** 或**.cnf**结尾，如 **passwd**、**inittab**、**group** 等。

63

3.1.1　Linux目录结构

5. /home

家目录，即用户的主目录，每一个用户都有一个文件夹，保存该用户的私有数据。默认情况下，除 **root** 外的用户，主目录都会放在这个目录下。在 **Linux** 下，可以通过**#cd~**来切换至自己的主目录。

6. /usr

该目录是系统存放程序的目录，其空间比较大。例如，/**usr/src** 中存放着**Linux** 内核的源代码，/**usr/include** 中存放着 **Linux** 下开发和编译应用程序需要的头文件。这个目录下有很多文件和目录，当我们安装一个 **Linux** 官方提供的发行版软件包时，大多文件都安装在这里。

7. /var

存放系统产生的文件，该目录的内容经常变动。例如，/**var/tmp** 就是用来存储临时文件的。还有很多其他的进程和模块把它们的记录文件也放在这个地方，包括如下一些重要的子目录。

8. /lib

/**lib** 是 **library** 的缩写，启动时需要用到的库文件都放在该目录下，相当于**Windows**下的 **.dll** 文件。而非启动用的库文件都会放在 /**usr/lib** 目录下。内核模块是放在/**lib/modules**（内核版本）下的。

64

3.1.1　Linux目录结构

9. /proc

这个目录在磁盘中是不存在的，它是存放在内存中的一个虚拟的文件夹，是启动 **Linux** 系统时创建的，里面的文件都是关于当前系统的实时状态信息，包括正在运行的进程、硬件状态、内存使用信息等。

10. /tmp

临时文件目录，有时用户运行程序时，会产生临时文件。因为/**tmp** 会自动删除文件，所以有用的文件不要放在该目录下。/**var/tmp** 目录和这个目录相似。

11. /mnt

该目录一般用于存放挂载储存设备的挂载目录（一个分区挂载在一个已存在的目录上，这个目录可以不为空，但挂载后，这个目录下以前的内容将不可用），它是安装软盘、光盘、**U** 盘的挂载点（挂载点实际上就是 **Linux** 中的磁盘文件系统的入口目录，类似于 **Windows** 中的用来访问不同分区的 **C**、**D**、**E** 等盘符）。/**media** 是自动挂载，与/**mnt**相同，但有些 **Linux** 系统没有 /**media**，而所有 **Linux** 系统都有 /**mnt**。

65

3.1.1　Linux目录结构

12. /bin

/**bin** 是 **binary** 的缩写，二进制文件，即可执行程序。里面保存的是基础系统所需的最基础的、最常用的命令，如 **ls**、**cp**、**mkdir** 等命令，功能和/**usr/bin** 类似。这个目录中的文件都是可执行的，并且是普通用户都可以使用的命令。

13. /sbin

/**sbin** 是 **super binary** 的缩写，存放的大多是涉及系统管理的命令，存储的也是二进制文件，但只有超级用户 **root** 才可以使用，普通用户无权执行这个目录下的命令，这个目录和/**usr/sbin**、/**usr/lib/debug/sbin** 或/**usr/local/sbin** 目录相似。目录 **sbin** 中包含的命令只有具有 **root** 权限才能执行的。

66

3.1.2 Linux文件类型

Linux 中常用的文件类型有 **5** 种：普通文件、目录文件、链接文件、设备文件和管道文件。

1．普通文件

一般来说，**Linux** 的普通文件是指以字节为单位的数据流类型文件，它是最常用的一类文件，其特点是不包含文件系统的结构信息。通常用户接触到的文件，如图形文件、数据文件、文档文件、声音文件等都属于普通文件。这种类型的文件按其内部结构又可细分为文本文件和二进制文件。

2．目录文件

目录文件不存放常规数据，它是用来组织、访问其他文件的。它是内核组织文件系统的基本节点。目录文件可以包含下一级目录文件或普通文件。在 **Linux** 中，目录文件是一种文件，与其他操作系统中“目录”的概念不同，它是 **Linux** 文件中的一种。

3．链接文件

链接文件是一种特殊的文件，实际上是指向一个真实存在的文件链接，类似于 **Windows** 下的快捷方式。根据链接文件的不同，又可以细分为硬链接（**Hard Link**）文件和符号链接（**Symbolic Link**，又称为软链接）文件。

67

3.1.2 Linux文件类型

4．设备文件

设备文件是 **Linux** 中最特殊的文件。正是由于它的存在，**Linux** 系统可以十分方便地访问外部设备。**Linux** 系统为外部设备提供一种标准接口，将外部设备视为一种特殊的文件。用户可以像访问普通文件一样访问任何外部设备，使 **Linux** 系统可以很方便地适应不断变化的外部设备。通常 **Linux** 系统将设备文件放在**/dev** 目录下，设备文件使用设备的主设备号和次设备号来指定某外部设备。根据访问数据方式的不同，设备文件又可以分为块设备和字符设备文件。

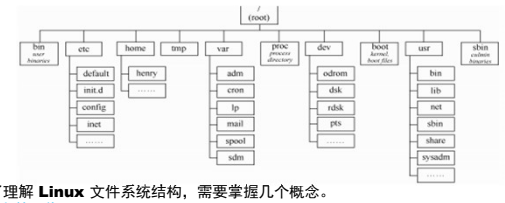
5．管道文件

管道文件是一种很特殊的文件，主要用于不同进程的信息传递。当两个进程间需要传递数据或信息时，可以使用管道文件。一个进程将需传递的数据或信息写入管道的一端，另一进程则从管道的另一端取得所需的数据或信息。

68

3.1.3 Linux文件系统结构

Linux 文件系统是一个倒转的单根树状结构。在 **Linux** 系统中，任何软件和 **I/O** 设备都被视为文件，而所有的文件及文件夹都是存在于一个根目录 **root** 下，如图所示。



为了理解 **Linux** 文件系统结构，需要掌握几个概念。

1．当前工作目录

在 **Linux** 文件系统中，每一个 **Shell** 或系统进程都有一个当前工作目录，使用 **pwd**命令可以显示当前的工作目录。每当在终端进行操作时，都会有一个当前工作目录。

69

3.1.3 Linux文件系统结构

2．文件名称

Linux 文件名称最多可使用 **255** 个字符，除了正斜线“/”外，都是有效字符，如可用 **A~Z**、**a~z**、**0~9** 等字符来命名。建议文件名称最好能体现文件的功能。和 **Windows**系统不同，**Linux** 文件系统严格区分大小写。以“**.**”开头的文件是隐藏文件。注意：在**Linux** 文件系统中，文件和文件夹是没有区别的，都统称为文件。

3．绝对路径与相对路径

如何到达一个文件或者目录有两种方式：绝对路径和相对路径，这是 **Linux** 文件系统管理中一个很重要的概念。绝对路径是以根目录“/”开始，递归每级目录直到目标路径；相对路径是以当前目录为起点，到达目标的路径。从以上定义可以看出，绝对路径不受当前所在目录限制，而相对路径受当前所在目录的限制。

70

3.2.1 工作目录与目录的切换

Linux 系统使用 **cd**（**change directory**）命令来切换工作目录，作用是改变当前工作目录。

cd 的命令格式为：**cd [directory]**

该命令将当前目录改变为 **directory** 指定的目录。若没有制定 **directory**，则回到用户的主目录，“**~**”是 **home** 目录的意思。主目录是当前用户的 **home** 目录，是添加用户时指定的。一般用户默认的 **home** 目录是**/home/xxx**（**xxx** 是用户名），**root** 的默认 **home**目录是**/root**。

要改变到指定目录，用户必须拥有对指定目录的执行和读权限。该命令可以使用通配符。

例如，假设用户当前的目录是**/root/working**，要更换到**/user/src** 目录下，可使用如下命令。

[root@localhost working]# cd /user/src

若在 **usr** 目录下有子目录 **abc**，要更换到**/usr/abc** 目录中，可采用更改相对路径的方法，命令操作如下。

[root@localhost working]# cd ../abc

跳到自己的 **home** 目录：

[root@localhost working]# cd ~

71

3.2.2 ls命令

ls（**list**）命令是用户最常用的命令之一。对于目录，**ls** 命令将输出该目录下的所有子目录与文件；对于文件，**ls** 命令将输出其文件名以及要求的其他信息。该命令类似于**DOS** 下的 **dir** 命令。默认情况下，输出条目按字母顺序排序。

ls 的命令格式为：**ls [option] [names]**

其中，**option** 选项可以省略，常用的参数如表所示。

选项	功能描述
-a	显示指定目录下的所有子目录与文件，包括隐藏文件
-A	显示指定目录下的所有子目录与文件，包括隐藏文件，但不列出“.”和“..”
-d	列出目录文件本身的状态，而不是列出目录下包括的文件内容。常与-l 选项联用，以得到目录的详细信息
-l	以长格式显示文件的详细信息。这个选项最为常用，每行列出的信息：文件类型与权限、链接数、文件属主（属主就是所属的主人，即 owner）、文件属组（属组就是 owner 所在的 group）、文件大小、建立或最近修改的时间名字
-L	若指定的名称为一个符号链接文件，则显示链接指向的文件
-n	输出格式与-l 选项相同，只不过在输出中文件属主和属组是用相应的 UID 号和 GID 号来表示，而不是实际的名称
-R	递归地列出其中包含的子目录中的文件信息及内容

72

3.2.3 目录的创建和删除

下面介绍 **Linux** 系统中的目录创建与删除命令。

1. mkdir 命令

创建目录需要使用 **mkdir** 命令。

mkdir 的命令格式为： **mkdir [option] [dirname]**

其中， **option** 选项可以省略，**dirname** 是要创建的目录名称。

2. rmdir 命令

rmdir 命令只能用来删除空目录，若目录中存在文件，就要使用 **rm** 命令删除文件后再删除目录，后面会详细介绍 **rm** 命令。

rmdir 的命令格式为： **rmdir [option] [dirname]**

其中， **option** 选项可以省略。 **dirname** 表示目录名。

73

3.2.4 文件的创建、复制、移动和删除命令

文件的创建、复制、移动和删除操作在 **Linux** 系统中使用得相当频繁，下面详细介绍这些操作命令。

1. touch 命令

touch 命令有两个功能：一是用于把已存在文件的时间标签更新为系统当前的时间（默认方式），它们的数据将原封不动地保留下来；二是用来创建新的空文件。

touch 的命令格式为： **touch [option] filename**

其中， **option** 选项可以省略。 **filename** 是将要创建的文件的名称。

2. cp 命令

该命令的功能是将给出的文件或目录复制到另一文件或目录中，就像 **DOS** 下的 **copy** 命令一样，功能十分强大。

cp 的命令格式为： **cp [option] source dest**

其中， **option** 选项可以省略。 **source** 表示需要复制的文件，**dest** 表示需要复制到的目录。

3. mv 命令

用户可以使用 **mv** 命令为文件或目录改名或者将文件由一个目录移动到另一个目录中。该命令类似于 **DOS** 下的 **ren** 和 **move** 的组合。

mv 的命令格式为： **mv [option] source dest**

74

3.2.4 文件的创建、复制、移动和删除命令

视 **mv** 命令中第二个参数类型的不同（是目标文件还是目标目录）， **mv** 命令将文件重命名或将其移至一个新的目录中。当第二个参数类型是文件时， **mv** 命令完成文件重命名，此时，源文件只能有一个（也可以是源目录名），它将所给的源文件或目录重命名为给定的目标文件名。当第二个参数是已存在的目录名称时，源文件或目录参数可以有多个， **mv** 命令将各参数指定的源文件均移至目标目录中。在跨文件系统移动文件时，**mv** 先拷贝，再将原有文件删除，而链接该文件的链接也将丢失。

例如，将文件 **test.txt** 重命名为 **mv.doc**。

[user@localhost ~]$ # mv test.txt mv.doc

4. rm 命令

在 **Linux** 系统中，可以使用 **rm** 命令将无用文件删除。该命令的功能是删除一个目录中的一个或者多个文件，也可以将某个目录及其下的所有文件及子目录均删除。对于链接文件，只是删除了链接，原有文件均保持不变。

rm 的命令格式为： **rm [option] filename**

75

3.2.5 其他操作

下面介绍 **Linux** 系统中常用的其他操作命令。

1. sort 命令

该命令的功能是对文件中的各行进行排序。**sort** 将文件的每一行作为一个单位，相互比较，比较原则是从首字符向后，依次按 **ASCII** 码值进行比较，最后将它们按升序输出。

sort 的命令格式为： **sort [option] filename**

其中，**option** 选项可以省略，**filename** 是操作对象的文件名称。

2. cat 命令

该命令的主要功能是用来显示文件内容， 依次读取其后所指文件的内容并将其输出到标准输出设备上。另外，还能够用来连接两个或者多个文件，形成新文件。

cat 的命令格式为： **cat [option] filename**

其中， **option** 选项可以省略，**filename** 是操作对象的文件名称。

3. more 命令

在查看文件的过程中,可以使用 **more** 命令一次只显示一屏文本，并在终端底部打出“**--more--**”，系统还将同时显示已显示文本占全部文本的百分比。如果要继续显示，可以按回车键或空格键。

more 的命令格式为： **more [option] filename**

76

3.2.5 其他操作

4. info 命令

info 是一种文档格式，也是阅读此格式文档的阅读器，常用它来查看 **Linux** 命令的 **info** 文档。它以主题的形式把几个命令组织在一起，以便于阅读。在主题内以 **node**（节点）的形式把本主题的几个命令串联在一起。

info 的命令格式为： **info [option] filename**

5. file 命令

file 命令用于辨识文件类型。

file 的命令格式为： **file [option] filename**

其中，**option** 选项可以省略，**filename** 是操作对象的文件名称。

77

3.3.1 文件内容查找命令

文件内容查询命令主要是指 **grep**、 **egrep** 与 **fgrep** 命令。这组命令以指定的查找模式搜索文件，通知用户在什么文件中搜索到与指定模式匹配的字符串，并且打印出所有包含该字符的文本行，该文本行的最前面是该行所在的文件名。这 **3** 个命令的含义分别如下。

（**1**）**grep** 命令：是最早的文本匹配程序，使用 **POSIX** 定义的基本正则表达式（**BRE**）来匹配文本。该命令一次只能搜索一个指定的模式。

（**2**）**egrep** 命令：扩展式 **grep**，其使用扩展式表达式（**ERE**）匹配文本。

（**3**）**fgrep** 命令：快速 **grep**，这个版本匹配固定字符串而非正则表达式。并且是唯一可以并行匹配多个字符串的版本。

78

3.3.2 find命令

Linux 下 **find** 命令在目录结构中搜索文件，并执行指定的操作。该命令的功能是从指定的目录开始，递归地搜索其各个子目录，查找满足寻找条件的文件并对其采取相关的操作。因为此命令提供了相当多的查找条件，功能很强大，所以它的选项也很多。

find 的命令格式为： **find [option] filename**

find 命令提供的寻找条件可以使一个用逻辑运算符 **not**、 **and** 和 **or** 组成的复合条件。逻辑运算符 **not**、 **and** 和 **or** 的含义如下。

and：逻辑与，在命令中用“**-a**”表示，是系统默认的选项，表示只有当所给的条件都满足时，寻找条件才算满足。

or：逻辑或，在命令中用“**-o**”表示。该运算符表示只要所给的条件中有一个满足，寻找条件就算满足。

not：逻辑非，在命令中用“！”表示。该运算符表示查找不满足所给条件的文件。

79

3.3.3 locate命令

该命令的功能也是查找文件，比 **find** 命令的搜索速度快，原因在于它不搜索具体目录，而是搜索一个数据库（**/var/lib/located**），这个数据库中含有本地所有文件信息。**Linux** 系统自动创建这个数据库，并且每天自动更新一次，所以使用 **locate** 命令查不到最新变动过的文件。为了避免这种情况，可以在使用 **locate** 之前，先使用 **updatedb** 命令，手动更新数据库。

locate 的命令格式为： **locate [option] filename**

例如，搜索 **etc** 目录下所有以 **sh** 开头的文件，忽略大小写区别。

```
[user@localhost ~]$ locate  -i  /etc/sh
/etc/shadow
/etc/shadow-
/etc/shells
```


80

3.3.4 whereis命令

whereis 命令用于查找文件。该指令会在特定目录中查找符合条件的文件。这些文件应属于原始代码、二进制文件，或帮助文件。该指令只能用于查找二进制文件、源代码文件和 **man** 手册页，一般文件的定位需使用 **locate** 命令。

whereis 的命令格式为： **whereis [option] filename**

例如，使用命令 **whereis** 查看 **bash** 命令的位置，输入如下命令。

```
[user@localhost ~]$ whereis bash
```

上面的指令执行后，输出信息如下。

```
bash:/bin/bash/etc/bash.bashrc/usr/share/man/man1/bash.1.gz
```


81

3.4.1 tar命令

tar 是一个归档程序，也就是说， **tar** 命令可以将许多文件打包成为一个归档文件或者把它们写入备份设备，如一个磁带驱动器。所以通常 **Linux** 下，保存文件都是先用 **tar**命令将目录或者文件打成 **tar** 归档文件（也称 **tar** 包），然后进行压缩。

tar 的命令格式为： **tar [option] filename**

例如，使用 **touch** 命令创建一个文件名为**a.c**的文件。

```
[user@localhost ~]$ touch a.c
```

压缩 **a.c** 文件为 **test.tar.gz**。

```
[user@localhost ~]$ tar -czvf test.tar.gz a.c
```

列出压缩文件内容。

```
[user@localhost ~]$ tar -tzvf test.tar.gz
-rw-r--r-- root/root 0 2017-02-15 16:51:59 a.c
```


82

3.4.2 zip命令

zip 命令可以用来解压缩文件，或者对文件进行打包操作。 **zip** 是个使用广泛的压缩程序，文件经它压缩后会另外产生具有 “**.zip**”扩展名的压缩文件。

zip 的命令格式为： **zip [option] filename**

例 如 ， 将 **/home/Blinux/html/**目录下的所有文件和文件夹打包为当前目录下的**html.zip**。

```
[user@localhost ~]$ zip -q -r html.zip /home/Blinux/html
```


83

3.4.3 常用压缩格式

1. 文件压缩——**gzip** 命令

gzip 命令用于压缩一个或更多文件。执行命令后，原文件会被其压缩文件取代。

gzip的命令格式为：**gzip [option] filename**

例如，压缩 **hello.c**，压缩后，文件以 **gz** 结尾，原始文件已删除。

```
[user@localhost ~]$ gzip hello.c
[user@localhost ~]$ ls
hello.c.gz
```

2. 文件压缩——**bzip2** 命令

bzip2 命令由 **Julian Seward** 开发，与 **gzip** 命令功能相仿，但是使用不同的压缩算法。该算法具有高质量的数据压缩能力，但降低了压缩速度。多数情况下，其用法与 **gzip**类似，只是用 **bzip2** 压缩后，文件的后缀为**.bz2**。

bzip2 的命令格式为： **bzip2 [option] filename**

例如，解压**.bz2** 文件。

```
[user@localhost ~]$ bzip2 -v temp.bz2
```


84

3.5.1 硬链接

硬链接是最初 **UNIX** 用来创建链接的方式，符号链接较之更为先进。默认情况下，每个文件有一个硬链接，该硬链接会给文件起名字。创建一个硬链接时，也为这个文件创建了一个额外的目录条目。硬链接有以下两个重要的局限性。

（**1**）硬链接不能引用自身文件系统之外的文件。也就是说，链接不能引用与该链接不在同一磁盘分区的文件。

（**2**）硬链接无法引用目录。硬链接和文件本身没有什么区别。与包含符号链接的目录列表不同，包含硬链接的目录列表没有特别的链接指示说明。当硬链接被删除时，只是删除了这个链接，但是文件本身的内容依然存在（也就是说，该空间没有释放），除非该文件的所有链接都被删除了。

85

3.5.2 符号链接

符号链接是为了克服硬链接的局限性而创建的。 符号链接是通过创建一个特殊类型的文件来起作用的，该文件包含了指向引用文件或目录的文本指针。就这点来看，符号链接与 **Windows** 系统下的快捷方式非常相似，但是，符号链接要早与 **Windows** 的快捷方式很多年出现。

符号链接指向的文件与符号链接自身几乎没有区别。例如，将一些东西写进符号链接里，这些东西同样也写进了引用文件。而当删除一个符号链接时，删除的只是符号链接而没有删除文件本身。如果先于符号链接之前删除文件，那么这个链接依然存在，但不指向任何文件。此时，这个链接就称为坏链接。在很多实现中， **ls** 命令会用不同的颜色来显示坏链接，如红色。

86

3.6.1 文件系统

随着 **Linux** 的不断发展，其支持的文件格式系统也在迅速扩展。特别是 **Linux 2.6**内核正式推出后，出现了大量新的文件系统，其中包括日志文件系统 **Ext4**、**Ext3**、**ReiserFS**、 **XFS**、 **JFS** 和其他文件系统。 **Linux** 系统核心可以支持十多种文件系统类型：**JFS**、 **ReiserFS**、 **Ext**、 **Ext2**、 **Ext3**、 **ISO9660**、 **XFS**、 **Minx**、 **MSDOS**、 **UMSDOS**、 **VFAT**、**NTFS**、 **HPFS**、 **NFS**、 **SMB**、 **SysV**、 **PROC** 等。其中，使用较为普遍的有如下几种。

（**1**） **Minix**：是 **Linux** 支持的第一个文件系统，对用户有很多限制，性能低下，有些没有时间标记，文件名最长为 **14** 个字符。

（**2**） **Xia**：是 **Minix** 文件系统修正后的版本，在一定程度上解决了文件名和文件系统大小的局限。

（**3**） **NFS**（**Network File System**）：是 **Sun** 公司推出的网络文件系统，允许在多台计算机之间共享同一文件系统，易于从所有这些计算机上存取文件。

（**4**）扩展文件系统（**Ext File System**）：是随着 **Linux** 的不断成熟而引入的，它包含了几个重要的扩展，但提供的性能令人不满意。 **1994** 年人们引入了第二扩展文件系统（**second Extended Filesystem**， **Ext2**）以代替过时的 **Ext** 文件系统。

87

3.6.1 文件系统

（**5**） **Ext3**（**third Extended Filesystem**）：是由开放资源社区开发的日志文件系统，被设计成 **Ext2** 的升级版本，尽可能地方便用户从 **Ext2** 向 **Ext3** 迁移。

（**6**） **Ext4**（**The fourth extended file system**）：是一种针对 **Ext3** 系统的扩展日志式文件系统，是专门为 **Linux** 开发的原始扩展文件系统（**ext** 或 **extfs**）的第 **4** 版。

（**7**）**Reiser**：是另一套专为 **Linux** 设计的日志文件系统，目前最新的版本是 **Reiser4**。**Reiser** 文件系统在处理小文件上比 **Ext3** 文件系统更有优势，效率更高，碎片也更少。

（**8**） **XFS**：是一种高级日志文件系统， **XFS** 具备较强的伸缩性，非常健壮。其数据完整性、传输特性、可扩展性等诸多指标都非常突出。

（**9**） **ISO9660** 标准 **CDROM** 文件系统，通用的 **Rock Ridge** 增强系统，允许长文件名。

除了上述这些 **Linux** 支持的文件系统外， **Linux** 还可以支持基于 **Windows** 和 **Netware** 的文件系统，如 **UMSDOS**、 **MSDOS**、 **VFAT**、 **HPFS**、 **SMB** 和 **NCPFS** 等。

88

3.6.2 磁盘分区

1．磁盘分区命名方式

在 **Linux** 中，每一个硬件设备都映射到一个系统的文件，包括硬盘、光驱等 **IDE** 或**SCSI**（**Small Computer System Interface**，设备小型计算机系统接口，一种用于计算机和智能设备之间（硬盘、软驱、光驱、打印机、扫描仪等）系统级接口的独立处理器标准，**SCSI** 是一种智能的通用接口标准）设备。 **Linux** 为各种 **IDE** 设备分配了一个由 **hd** 前缀组成的文件。各种 **SCSI** 设备，则被分配了一个由 **sd** 前缀组成的文件，编号方法为拉丁字母表顺序。如第一个 **IDE** 设备（如 **IDE** 硬盘或 **IDE** 光驱）， **Linux** 定义为 **hda**；第二个 **IDE** 设备就定义为 **hdb**；下面以此类推。而 **SCSI** 设备就应该是 **sda**、 **sdb**、 **sdc** 等。**USB** 磁盘通常会被识别为 **SCSI** 设备，因此其设备名可能是 **sda**。

常见的 **Linux** 磁盘命名规则为 **hdXY**（或者 **sdXY**），其中 **X** 为小写拉丁字母， **Y** 为阿拉伯数字。个别系统可能命名略有差异。

2．磁盘分区方法

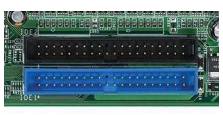
对于一个新硬盘，首先需要对其进行分区。和 **Windows** 一样，在 **Linux** 下用于磁盘分区的工具也是 **fdisk** 命令。除此之外，还可以通过 **parted**、 **cfdisk** 等可视化工具进行分区。

89

3.6.2 磁盘分区








90

3.6.2 磁盘分区

3．分区的格式化

分区完成后，需要格式化文件系统才能正常使用。格式化磁盘的主要命令是**mkfs**。

mkfs 的命令格式为： **mkfs -t type device [block_size]**

其中，选项**-t** 的参数 **type** 为文件系统格式，如 **ext4**、 **vfat**、 **ntfs** 等；参数 **device** 为设备名称，如**/dev/hda1**、 **/dev/sdb1** 等；参数**[block_size]**为 **block** 大小，可选。

如果需要把**/dev/sda1** 格式化为 **FAT32** 格式，则可以使用如下命令。

mkfs -t vfat /dev/sda1

格式化交换分区的命令略有不同，不是 **mkfs**，而是 **mkswap**。例如，将**/dev/hda8**格式化为 **swap** 分区，可以使用如下命令。

mkswap /dev/hda8

91

3.6.3 磁盘检验

对于没有正常卸载的磁盘，如遇到断电等突发情况，可能损坏文件系统目录结构或其中的文件。因此，遇到这种情况需要检查和修复磁盘分区。检查和修复磁盘分区的命令为 **fsck**。

fsck 的命令格式为： **fsck [option] device**

其中， **option** 选项可以省略，参数 **device** 为设备名称，如**/dev/hda1**、**/dev/sdb1** 等。

和 **mkfs** 一样， **fsck** 也有很多别名，如 **fsck.ext4**、 **fsck.reiserfs**、**fsck.vfat** 等。 **fsck.fstype**形式的别名还有 **e2fsck**、 **reiserfsck** 等类型。例如，检测 **Reiserfs** 格式的分区**/dev/hda5**，以下 **3** 个命令均可。

fsck -t reiserfs /dev/hda5
fsck.reiserfs /dev/hda5
reiserfsck /dev/hda5

92

3.6.4 磁盘挂载和卸载

1．挂载磁盘分区

要使用磁盘分区，就需要挂载该分区。挂载时指定需要挂载的设备和挂载目录（该挂载目录即是挂载点）。挂载磁盘分区的命令为 **mount**。

mount 的命令格式为： **mount -t type device dir**

其中，选项**-t** 的参数 **type** 为文件系统格式，如 **ext4**、**vfat**、 **ntfs** 等；参数 **device** 为设备名称，如**/dev/hda1**、**/dev/sdb1** 等；参数 **dir** 为挂载目录，成功挂载后，就可以通过访问该目录访问该分区内的文件，如**/mnt/windows_c**、**/mnt/cdrom** 等。凡是未被使用的空目录，都可用于挂载分区。

2．卸载磁盘分区

移除磁盘，如卸载 **USB** 磁盘、光盘或者某一硬盘分区，需要首先卸载该分区。卸载磁盘分区的命令为 **umount**。

umount 的命令格式为： **umount [device|dir]**

卸载时只需要一个参数，可以是设备名称，也可以是挂载点（目录名称）。例如，卸载一个光驱设备**/dev/hdc**，该设备挂载于**/mnt/cdrom**。那么既可以直接卸载该设备，也可以通过其挂载的目录卸载。

93

3.6.5 交换空间

当系统的物理内存不够用时，就需要将物理内存中的一部分空间释放出来，以供当前运行的程序使用。那些被释放的空间可能来自一些很长时间都没有什么操作的程序，这些释放的空间被临时保存到 **Swap** 空间中，等到那些程序要运行时，再从 **Swap** 中恢复保存的数据到内存中。这样，系统总是在物理内存不够时，才进行 **Swap** 交换。其实，**Swap** 的调整对 **Linux** 服务器，特别是 **Web** 服务器的性能至关重要。调整 **Swap**，有时可以越过系统性能瓶颈，节省系统升级费用。

Swap 空间有两种形式：交换分区和交换文件。总之对 **Swap** 的读写都是磁盘操作。

增加交换空间有以下两种方法（严格来说，在系统安装完后，只有一种方法可以增加 **Swap**，那就是下面介绍的第二种方法，至于第一种方法，应该是安装系统时设置交换区）。

方法一：使用分区：在安装 **OS** 时划分出专门的交换分区，空间大小要事先规划好，启动系统时自动进行 **mount**。这种方法只能在安装 **OS** 时设定，一旦设定好，就不容易改变，除非重装系统。

94

3.6.5 交换空间

方法二：使用 **swapfile**（或者是整个空闲分区）：新建临时 **swapfile** 或者空闲分区，在需要时设定为交换空间，最多可以增加 **8** 个 **swapfile**。交换空间的大小与 **CPU** 密切相关，如在 **i386** 系中，最多可以使用 **2GB** 的空间。在系统启动后，根据需要在 **2GB** 的总容量下增减。这种方法比较灵活，也比较方便，缺点是启动系统后需要手工设置。

运用 **swapfile** 增加交换空间涉及命令如下。

（**1**） **free**：查看内存状态命令，可以显示 **memory**、 **swap**、 **buffer cache** 等的大小及使用状况。
（**2**） **dd**：读取，转换并输出数据命令。
（**3**） **mkswap**：设置交换区。
（**4**） **swapon**：启用交换区，相当于 **mount**。
（**5**） **swapoff**：关闭交换区，相当于 **umount**。

95

Linux操作系统基础教程

任课教师 刘昊霖
联系方式： liuhaolin@xtu.edu.cn

96

目 录 CONTENTS

97

第4章 Linux用户及权限机制

《Linux操作系统基础教程》

98

第4章 Linux用户及权限机制

99

4.1.1 用户的管理

Linux 系统是一个多用户多任务的分时操作系统，任何一个要使用系统资源的用户，都必须首先向系统管理员申请一个账号，然后以该账号的身份进入系统。每个用户账号都拥有一个唯一的用户名和口令，同时系统会为每个用户账号分配一个用户 **ID**（**uid**）来标识用户。用户在登录时键入正确的用户名和口令后，就能够进入系统和自己的主目录。

根据用户 **ID** 的不同，在 **Linux** 系统中，用户可分为以下 **3** 种类型。

（**1**） **root** 用户：又称为超级用户， **ID** 为 **0**，拥有最高权限。

（**2**）系统用户：又称为虚拟用户、伪用户或假用户，不具有登录**Linux** 系统的能力，但却是系统运行不可缺少的用户，一般 **ID** 为 **1**～**499**，本书中使用的**Centos7** 为 **1**～**999**。

（**3**）普通用户： **ID** 为 **500** 以上， **Centos7** 为 **1000** 以上。可以登录**Linux** 系统，但是使用的权限有限，由管理员创建。

用户管理的常用命令包括： **useradd**、 **passwd**、 **usermod**、 **userdel**。 **useradd** 用来添加用户， **passwd** 修改用户口令， **usermod** 修改用户信息， **userdel** 删除用户。

100

4.1.1 用户的管理

1．添加用户

添加用户就是在系统中创建一个新账号，并为新账号分配用户 **ID**、用户组、主目录和登录 **Shell** 等资源。通过 **useradd** 命令添加用户。

useradd 的命令格式为： **useradd [option] username**

username 表示新账号的登录名。

2．修改用户口令

用户账号刚创建时没有口令，被系统锁定无法使用，必须为其指定口令后才可以使用，即使是空口令。使用 **passwd** 命令指定和修改用户口令。超级用户可以为自己和其他用户指定口令，普通用户只能用它修改自己的口令。

passwd 的命令格式为： **passwd [option] [username]**

其中， **option** 选项可以缺省，主要对**/etc/shadow** 文件的字段产生影响，**username** 参数也可以缺省，没有指定该参数时，表示修改当前用户的口令；如果指定了该参数，则表示修改指定用户的口令，只有 **root** 用户才有修改指定用户口令的权限。

101

4.1.1 用户的管理

3．修改用户信息

修改用户信息就是更改用户的属性，如用户 **ID**、主目录、用户所在组、登录 **Shell**等。通过 **usermod** 命令修改用户信息。

usermod 的命令格式为： **usermod [option] username**

username 表示用户名。

4．删除用户

如果一个用户账号不再使用，可以从系统中删除。删除用户就是删除与用户有关的系统配置文件中的记录（如**/etc/passwd**）。通过 **userdel** 命令删除用户。

userdel 的命令格式为： **userdel [option] username**

其中， **option** 选项最常用的参数是**-r**，表示同时删除用户的主目录。

username 表示要删除的用户。例如，删除用户 **user2**，执行如下命令。

[root@localhost Desktop]# userdel -r user2

此时再查看**/etc/passwd** 文件中的信息，将不会找到与 **user2** 有关的信息行。

102

4.1.2 用户组的管理

用户组是具有相同特征用户的集合，每个用户都有一个用户组，方便系统集中管理一个用户组中的所有用户。用户组的管理主要包括用户组的添加、修改和删除。常用命令有： **groupadd**、 **groupmod**、**groupdel**。

1．添加用户组

使用 **groupadd** 命令增加新的用户组。

groupadd 的命令格式为： **groupadd [option] group**

group 表示用户组的组名。

2．修改用户组信息

使用 **groupmod** 命令修改用户组的属性。

groupmod 的命令格式为： **groupmod [option] group**

group 表示需要修改属性的用户组的名称。

3．删除用户组

使用 **groupdel** 命令删除用户组。

groupdel 命令的格式为： **groupdel group**

group 表示要删除的用户组的名称。

103

4.1.3 用户配置文件

与用户相关的系统配置文件主要有 **/etc/passwd**、**/etc/shadow**、**/etc/group** 。 **/etc/passwd** 文件保存用户信息， **/etc/shadow** 文件保存加密的用户密码，**/etc/group** 文件保存用户组信息。

1．**/etc/passwd** 文件

系统中所有的用户信息都会记录到**/etc/passwd** 文件中，是系统识别用户的一个文件。当用户登录时，系统首先查阅 **/etc/passwd** 文件。假设用户名为 **user**， 则会在**/etc/passwd** 文件中查看是否有该账号，然后确定 **user** 的 **uid**，通过 **uid** 确认用户和身份。在**/etc/passwd** 文件中，每一行都表示一个用户的信息。每行有 **7** 个字段，字段之间通过“：”分隔。例如：

[root@localhost Desktop]# cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
user:x:1001:1001::/home/user:/bin/bash

104

4.1.3 用户配置文件

2．**/etc/shadow** 文件

/etc/shadow 与**/etc/passwd** 文件是互补的，由于**/etc/passwd** 文件所有用户都可以访问，为保证安全，将密码和其他**/etc/passwd** 文件不能包括的信息（如有效期限）单独保存在**/etc/shadow** 中，此文件只有 **root** 用户有权查看。例如，用 **user** 用户查看该文件时，会出现提示“**Permission denied**（权限不足）”。

[user@localhost Desktop]$ cat /etc/shadow
cat: /etc/shadow: Permission denied

3．**/etc/group** 文件

/etc/group 文件是用户组的配置文件，可以直观看出用户组中包括哪些用户。每个用户组是一条记录，每个记录包含 **4** 个字段，字段之间通过“：”分隔，具体内容如下。

[root@localhost Desktop]# cat /etc/group
root:x:0:myu
user:x:1001:
user1:x:1002:
myuser:x:1003:
myGroup:x:1004:

105

4.2.1 所有者、所在组和其他用户

有 **3** 种类型的用户可以访问文件或目录：文件所有者、同组用户、其他用户。所有者一般是文件的创建者，对该文件的访问权限拥有控制权。所有者可以允许同组用户有权访问文件，还可以将文件的访问权限赋予系统中的其他用户。通过 **chown** 和 **chgrp** 命令可以改变文件的所属用户和所属组。

1．改变文件所属用户

通过 **chown** 命令将文件的所有者修改为指定的用户，普通用户不能将自己的文件改变成其他的拥有者，超级用户才拥有此权限。

chown 的命令格式为： **chown [option] [owner][:[group]] file**。

owner 表示文件的所有者，可以是用户名，也可以是用户 **ID**。 **group** 表示文件的所在组，组名或者组 **ID** 均可。 **file** 是文件的名称。

2．改变文件所在组

通过 **chgrp** 命令变更目录和文件的所属组，只有超级用户才拥有此权限。

chgrp 的命令格式为： **chgrp [option] group file**。

group 表示目录或文件的所在组，可以是组名或者组 **ID**。 **file** 是目录或文件的名称。

106

4.2.2 读、写和执行操作

每一文件或目录的访问权限都有三组，每组用三位表示，分别为文件属主的读、写、执行权限，与属主同组用户的读、写和执行权限，系统中其他用户的读、写和执行权限。文件被创建时，文件的所有者自动拥有对该文件的读、写、执行权限，以便于阅读和修改文件。用户也可根据需要把访问权限设置为需要的任何组合，目录必须拥有执行权限，否则无法查看其内容。 **r** 表示读权限， **w** 表示写权限， **x** 表示执行权限。三种权限模式对文件和目录的影响如表所示。

权限	对文件的影响	对目录的影响
r（读）	可读取文件内容	可列出目录内容
w（写）	可修改文件内容	可在目录中创建、删除文件
x（执行）	可作为命令执行	可访问目录内容

当用 **ls –l** 命令显示文件或目录的详细信息时，最左边一列为文件的访问权限。**/etc/passwd** 文件的详细信息如下。

[root@localhost Desktop]# ls -l /etc/passwd
-rw-r--r--. 1 root root 2418 Mar 22 10:48 /etc/passwd

107

4.2.2 读、写和执行操作

输出结果的前十个字符“**-rw-r--r--**”表示文件属性，第一个字符表示文件类型，剩下的 **9** 个字符（三个一组）分别表示文件所有者、文件所在组以及其他用户对该文件的读、写和执行权限，具体含义如图所示。

通过 **chmod** 命令改变不同用户对文件或目录的访问权限，文件或目录的所有者和超级用户拥有修改权限。该命令有以下两种使用方法：表达式法和数字法。

108

4.2.2 读、写和执行操作

1. 表达式法

表达式法的**chmod**的命令格式为：**chmod [who] [operator] [mode] file**。

其中， **who** 指定用户身份，若此参数省略，则表示对所有用户进行操作。**operator** 表示添加或取消某个权限，取值为“**+**”或“**-**”。**mode** 指定读、写、执行权限，取值为 **r**、 **w**、 **x** 的任意组合。

2. 数字法

chmod 支持以数字方式修改权限，读、写、执行权限分别由 **3** 个数字表示： **r**（读）**=4**；**w**（写）**=2**； **x**（执行）**=1**。每组权限分别为对应数字之和，如“**rw-rw-r--**”表示为“**664**”（其中的**-**表示没有某一个权限，取值为 **0**）。当前 **a** 文件的访问权限为“**rwxrwxrw-**”，将其恢复到原来的“**rw-rw-r--**”，只需要执行命令“**chmod 664 a**”，结果如下。

```
[root@localhost four]# chmod 664 a
[root@localhost four]# ls -ld a
-rw-rw-r--. 1 myUser myGroup 0 Mar 28 15:20 a
```


109

4.2.3 umask属性和特殊权限

文件和目录的默认访问权限是不同的。文件默认没有执行权限，对于所有者、同组用户、其他用户都只有 **rw** 两个权限，即默认属性为“**-rw-rw-rw-**（数字表示为：**666**）”。对于目录而言，所有权限均开放，即默认属性为“**drwxrwxrwx**（数字表示为： **777**）”。但新建文件目录的访问权限是由默认权限和 **umask** 属性的差值决定的，每个终端都拥有一个 **umask** 属性。一般普通用户的默认 **umask** 是 **002**，**root** 用户的默认 **umask** 是 **022**，使用 **umask** 命令查看当前终端的 **umask** 值。

```
[user@localhost Desktop]$ umask //普通用户
0002
[user@localhost Desktop]$ su root //切换到 root 用 户
Password:
[root@localhost Desktop]# umask //root 用户
0022
```

使用 **umask** 命令查看默认权限时，有 **4** 位数字（如 **0002**），而所有者、所在组、其他用户的读、写、执行权限只占后面三位。这是因为除读、写、执行 **3** 个普通权限外，系统中还存在 **3** 个特殊权限： **suid**、 **sgid**、 **sbit**，最开头的一位保存特殊权限。

110

4.2.3 umask属性和特殊权限

1. suid

设置了 **suid** 权限的文件，在执行时以文件的所属用户身份执行，而非执行文件的用户。例如，查看文件**/usr/bin/passwd**，所有者的 **x** 权限被 **s** 替代，命令如下。

```
[root@localhost Desktop]# which passwd
/usr/bin/passwd
[root@localhost Desktop]# ls -ld /usr/bin/passwd
-rwsr-xr-x. 1 root root 27832 Jun 10 2014 /usr/bin/passwd
```

当 **s** 标志出现在文件所有者的 **x** 权限上时，如上述结果出现的“**-rwsr-xr-x**”，该文件就拥有了 **suid** 权限。 **suid** 权限的限制与功能为： **suid** 只能用于二进制可执行文件（即需对该文件拥有可执行权限），对目录无效；执行者将具有该文件所有者的权限；本权限只在文件执行时有效，执行完毕不再拥有所有者权限。

2. sgid

sgid 权限出现在文件所属组权限的执行位上面。与 **suid** 不同的是，对普通二进制文件和目录都有效。当它作用于普通文件时和 **suid** 类似，执行者若具有该文件的 **x** 权限，执行者将获得该文件所属组的权限。当 **sgid** 作用于目录时，若执行者对某一目录具有 **x**、 **w** 权限，该执行者就可以在该目录下建立文件，而且该执行者在这个目录下建立的文件都是属于这个目录所属的组。

111

4.2.3 umask属性和特殊权限

3. sbit

sbit 权限出现在其他用户的 **x** 权限上，只对目录有效，若执行者对此目录具有 **w**、**x** 权限，在该目录下创建文件或目录时，仅自己与 **root** 才有权删除新建的目录或文件。例如， **/tmp** 文件的访问权限是“**drwxrwxrwt**”。

```
[root@localhost Desktop]# ls -ld /tmp
drwxrwxrwt. 16 root root 4096 Mar 29 18:55 /tmp
```

特殊权限的修改和设置同样使用 **chmod** 命令，表达式法和数字法均可。表达式法与修改普通权限一样，针对 **u**（所有者）、 **g**（所在组）、 **o**（其他用户）进行设置，由于**suid**、 **sgid**、 **sbit** 权限是代替特定用户的 **x** 权限位，使用方法一共有 **3** 种： **chmod u+s file**、**chmod g+s file**、 **chmod o+t file**， **file** 表示文件或目录的名称。数字法则是在原来三位数字的前面添加一位， **4** 表示 **suid** 权限， **2** 表示 **sgid** 权限， **1** 表示 **sbit** 权限，如 **chmod 4777 file**（**file** 表示文件或目录的名称）。

112

4.2.4 文件属性控制

在平时使用计算机时，我们会遇到这样的需求：在操作文件时，不能让用户修改文件本身的内容，但允许用户添加新的内容到文件中。虽然从需求的角度上来看这没有问题，但是从实现的角度上来看好像是有冲突的，即文件既不允许写又允许写。这在其他操作系统与文件系统中很难实现，而 **Linux** 则提供了很容易实现的机制——文件属性控制。在文件系统 **ext4** 中，只需要 **chattr +a file** 就可以实现以上功能。当然，并不是所有文件系统都支持这样的特性，当前支持的文件系统有： **ext2**、**ext3**、 **ext4**、 **btrfs** 等。

1. lsattr 查看文件属性

lsattr 列出当前文件的属性信息。

lsattr 的命令格式为：**lsattr [-RVadv] [files ...]**，参数选项及含义如表所示。

选项	功能描述
-R	递归显示出所有文件与目录
-V	查看软件版本信息
-a	显示目录中所有文件与目录的属性，包括隐藏文件与隐藏目录
-d	显示出目录本身的属性

113

4.2.4 文件属性控制

2. chattr 修改文件属性

修改文件属性方式有 **3** 种不同的方法，如表所示。

选项	功能描述
+[acdeijstuACDST]	在原有文件属性基础上添加一个或多个属性，如：chattr +a file
-[acdeijstuACDST]	在原有文件属性基础上移除一个或多个属性，如：chattr -a file
=[acdeijstuACDST]	设置文件属性为新的属性，如 chattr =ae file

114

Linux操作系统基础教程

任课教师 刘昊霖
联系方式： liuhaolin@xtu.edu.cn

115

目 录 CONTENTS

2021/12/2

116

第5章 Linux文本处理

《Linux操作系统基础教程》

117

第5章 Linux文本处理

2021/12/2

118

第5章 Linux文本处理

2021/12/2

119

5.1.1 Vim简介

在介绍 **Vim** 之前，先介绍 **vi**。 **vi** 是一个命令行界面下的文本编辑工具，由加州大学和伯克利大学等机构，以原来的 **UNIX** 行编辑器 **ed** 等为基础开发出来的，取 **visual**（可视化）单词的前两个字母进行命名。在 **Linux** 诞生时， **vi** 与基本**UNIX** 应用程序一样被保留下来，成为管理系统的好帮手，为用户提供了一个全屏幕的窗口编辑平台。同时 **vi**也融合了强大的行编辑器 **ex** 的功能，用户在使用 **vi** 的同时，也可以使用行编辑器的命令。

1991 年， **Bram Moolenaar** 基于 **vi** 进行了改进和优化，发布了 **Vim**（**vi improved** 的缩写），其最大的特点是加入了对 **GUI** 的支持。

大多数 **Linux** 的发行版配备的都是 **Vim**，即在使用**vi** 命令时，指向的也是 **Linux** 系统中的 **Vim** 程序。

但有些 **Linux** 版本默认安装的是 **Vim** 的最小版本（**Vim-tiny**），功能不全面，只支持有限的 **Vim** 特性。

2021/12/2

120

5.1.2 Vim模式

Vim 有 **3** 种工作模式：命令模式（或称常规模式）、插入模式、末行模式（或称**ex**模式）。

1. 命令模式

Vim 启动后，默认进入命令模式，在任何模式下，都可以按 **Esc** 键返回到命令模式，可以多按几次 **Esc** 键，保证顺利返回到命令模式。在命令模式下，可以键入不同的命令完成选择、复制、粘贴、删除等操作。

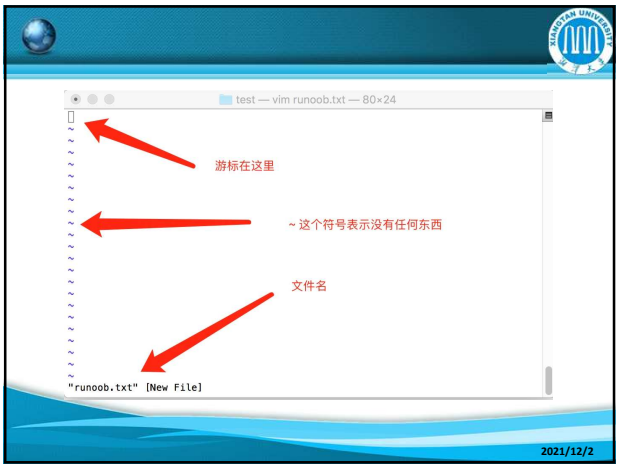
2. 插入模式

在插入模式下可以编辑文本内容。在命令行模式下按 **i**、 **a** 等键可以进入插入模式，在此模式下可以输入文本，但命令执行后的字符插入位置不同。

3. 末行模式

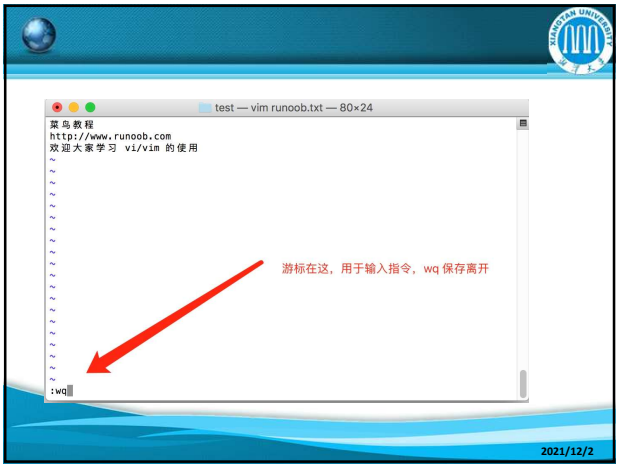
在命令模式下按：键进入末行模式。这时光标会移到屏幕底部，在这里可以保存修改或退出 **Vim**，也可以设置编辑环境、寻找字符串、列出行号等。

121



122

123



124

5.1.3 Vim环境下的查找和替换

Vim 具有一个十分强大的功能：查找和替换，在 **Vim** 环境下，根据搜索条件将光标移动到指定的位置，并用颜色标记查找出的内容。需要注意的是，在执行文本的替换工作时，可以用命令控制用户是否确认才可进行替换。

1. 行内搜索

命令 **f** 表示在光标所在的行内进行搜索。例如， **ft** 表示在光标所在行查找字母 **t**，光标会定位到第一个出现字母 **t** 的位置，此时输入分号 **";"**表示继续往下找，输入逗号 "，" 表示反方向查找。

2. 搜索整个文件

在命令行模式或末行模式下，输入 **"/"**，在屏幕的底部会出现一个 **"/"**符号，在 **"/"**后面输入想要查找的内容，按 **Enter** 键结束，当然在 **"/"**后面也可输入正则表达式进行搜索。使用 **n** 命令可以重复查找，如按 **"/"**查找，输入关键字，找出来后，按 **n** 键在查找到的关键字之间切换。查找到的内容会有颜色标识，方便观察。

3. 替换

替换文本内容是在末行模式下进行的，即如果需要替换则输入 **":"**进入末行模式。

125

5.2.1 剪切命令cut

在 **Linux** 中， **cut** 命令是剪切的意思，用于在数据中提取需要的部分，注意**cut** 命令是以行为对象来进行操作的。

cut 命令的格式为： **cut option [file]**

其中 **option** 选项指定 **cut** 以何种方式进行剪切，并给出剪切的具体位置， **file** 是 **cut** 命令操作对象的文件名，如果不指定 **file** 参数， **cut** 命令将读取标准输入。**option** 选项参数及含义如表所示,在执行 **cut** 命令时，必须指定**-b**、 **-c**、 **-f** 标志之一。

选项	功能描述
-b	以字节为单位进行剪切
-c	以字符为单位进行剪切
-f	以域为单位进行剪切
-d	自定义分割符（默认为制表符 Tab），与-f 一起使用，指定提取区域
-n	取消分割多字节字符。仅和-b 标志一起使用。如果字符的最后一个字节落在由-b 标志的 List 参数指示的 范围之内，该字符将被写出，否则该字符将被排除

cut 命令有 **3** 种剪切方式：以字节为单位截取、以字符为单位截取和以域为单位截取。

126

5.2.2 粘贴命令paste

paste 命令的作用与 **cut** 命令正好相反，不是从文本中提取信息，而是向文本中添加信息。

paste 命令的格式为： **paste [option] [file1] [file2]**

其中，**option** 选项可以省略， **option** 选项常用参数及含义如表所示。 **file1**、**file2**表示进行合并的文件的名称。

选项	功能描述
-s	将每个文件中的内容占用一行输出
-d	指定分隔符，若未使用该参数，则默认为制表符（TAB）分隔

通过示例了解 **paste** 命令的用法。

（**1**）将 **cut_bc**、 **cut_z** 文件中的内容合并， **paste** 命令后不跟任何参数。

```
[user@localhost five]$ paste cut_bc cut_z
Tom:Jones:4404 今天天气很好呢
Mary:Adams:2980 明天是雾霾
Sally:Chang:9999 周六又可以放假啦
Billy:Black:6666 好开心呢
```


127

5.2.2 粘贴命令paste

查看 **cut_bc**、 **cut_z** 文件内容，结果如下。

```
[user@localhost five]$ cat cut_bc cut_z
Tom:Jones:4404
Mary:Adams:2980
Sally:Chang:9999
Billy:Black:6666
今天天气很好呢
明天是雾霾
周六又可以放假啦
好开心呢
```

（**2**） **-s** 参数的作用。

-s 参数会将读取到的每一个文件中的内容作为一行输出。

（**3**） **–d** 参数的作用。

-d 参数与 **cut** 命令的**-d** 参数的作用一样，指定分隔符，将 **paste** 读到的文件内容平行输出，每个文件之间使用**-d** 指定的分隔符连接。

128

5.2.3 连接命令join

join 命令的作用与 **paste** 类似，简单来说也就是向文本中添加信息。

join 命令的格式为： **join [option] file1 file2**

其中，**option** 选项可以省略，**file1** 和 **file2**是要操作的文件名，**file1**和**file2** 必须是有序的，且包含相同的列。如果文件不是有序的，则会出现 “**join: filename is not sorted:**”的提示，若两个文件不包含相同的列，则 **join** 执行后的结果为空。

下面通过示例加深对 **join** 命令的理解，为了使文件是 **join** 命令可以操作的对象，做以下两步准备工作。

（**1**）分别对 **cut_bc**、 **cut_z** 文件进行排序，并将排好序的文本分别保存在 **join_file1**、**join_file2** 文件中。**sort** 命令可以对文本内容进行排序。

（**2**）通过 **vi** 命令，在插入模式下对 **join_file1**、 **join_file2** 文件中的文本添加行号并保存，使两个文件有共同的列。

到此准备工作就完成了，通过 **join** 命令对文件 **join_file1**、 **join_file2** 进行操作。

（**1**） **join** 后直接添加文件，两个文件中的文本逐行进行连接。

129

5.2.3 连接命令join

（**2**） **–a**、 **-v** 参数作用。

如果想将 **join_file1** 文件中的第 **5** 行输出，可以使用**-a** 参数，会按照**-a** 后面的数字，将指定文件中的内容完全输出，数字 **1** 表示第一个文件，数字 **2** 表示第二个文件。

-a 后面必须跟数字，只使用**-a** 系统会报错。例如：

```
[user@localhost five]$ join -a join_file1 join_file2
join: invalid field number: ‘join_file1’
```

-v 参数与**-a** 正好相反，输出指定文件中的特有行，即只在指定文件中存在的文本。例如：

```
[user@localhost five]$ join -v1 join_file1 join_file2
5 Tom:Jones:4404
```

（**3**） **-1**、 **-2** 参数作用。

默认情况下，判断两个文件能否连接，是看两个文件的第一列是否相同。如果需要改变连接字段，可以使用**-1**、 **-2** 选项，在**-1**、 **-2** 后面跟上希望使用连接字段的列号。 **-1** 指定第一个文件， **-2** 指定第二个文件。

130

5.3.1 有序文件逐行比较命令comm

comm 将逐行比较已经排好序的文本文件，如果文本是杂乱的，则可以通过**sort** 命令对文件进行排序。

comm 命令的格式为： **comm [option] file1 file2**

其中，**option** 选项可以省略，选项参数及含义如表所示，其中**-1**、 **-2**、 **-3** 分别针对 **comm** 输出结果中的第 **1**、第 **2**、第 **3** 列。 **file1**、 **file2** 表示要操作文件的名称。

选项	功能描述
-1	不显示输出结果的第一列
-2	不显示输出结果的第二列
-3	不显示输出结果的第三列

注意：因为 **comm** 只能对已经排好序的文件进行操作，所以要先对文件进行排序。

131

5.3.2 逐行比较命令diff

diff 同样也是比较文件的区别，对文件进行逐行比较，支持多种输出形式。相对于**comm** 命令来说， **diff** 有两个优点：文件可以是无序的，可以是比较大的文件集，尤其是在程序开发过程中，修改过后，利用 **diff** 可以很方便地查找版本之间的不同之处。

diff 命令的格式为： **diff [option] file**

其中，**option** 选项可以省略，**file** 表示要操作文件的名称。

diff 支持多种输出方式，有提示符 **c**、 **d**、 **a** 的是默认格式。通过 **option** 选项参数控制输出方式，常用的有上下文格式、统一格式、并排格式。

（**1**）上下文格式

使用**-c** 选项，输出结果的格式为上下文格式。文件内容全部输出，并分为上下两部分。

132

5.3.2 逐行比较命令diff

（2）统一格式
上下文格式输出内容比较齐全，也容易理解。但是因为文件中的文本内容都要输出，导致输出的部分有重复，在文本内容很多时，会使得输出很繁琐。使用**-u** 选项，正好解决了这一问题。 **-u** 选项输出格式为统一格式。
这样输出的内容就不会出现重复，文本相同的部分只输出一次，这使得输出相对来说比较精简。同样也是先输出两个文件的信息， @@部分表示文件各自的行范围，此时文本内容每行前面只会出现-、 +和“无符号”三种形式，表示的含义与上下文格式中一样。

（3）并排格式
使用**-y** 选项，输出格式为并排格式，文件中的对应行并排显示。

133

5.3.3 原文件比较命令patch

patch 命令用于更新文本文件，主要操作对象是 **diff** 生成的补丁文件，将旧版本文件更新成新文件。首先利用 **diff** 命令查找文件的不同，生成 **patch** 命令可操作的 **diff** 文件。操作过程如下。

```
[user@localhost five]$ diff cut_file1 cut_file2 > patch_file1
[user@localhost five]$ patch cut_file1 < patch_file1
patching file cut_file1
```

查看 **cut_file1** 文件中的内容，与 **cut_file2** 的文本内容一样，表示更新成功。

```
[user@localhost five]$ cat cut_file1
Tom
Mary
Sally
Billy
Adson
Jack
```

当在**diff**后面添加参数**-c**或**-u**选项时，生成的**diff**文件在页眉中会包含文件名，此时**patch**后面可以不用添加文件名。

134

5.4.1 格式化输出命令printf

printf 格式化并输出结果到标准输出。
printf 的命令格式为： **printf format [argument]**
其中，**format** 表示格式说明，此参数不能省略，并将该格式应用于 **argument** 代表的的输入内容。例如，输出字符串。

```
[user@localhost five]$ printf "I am working at %s\n" company
I am working at company
```

各类型数据输出格式如下。

```
[user@localhost five]$ printf "%d %f %s %o %x %X\n" 100 100 100 100 100 100
100 100.000000 100 144 64 64
```


135

5.4.2 简单文本格式化命令fmt

fmt 命令的作用是格式化段落，使文本看上去更加整齐。默认情况下（**fmt** 命令后不跟选项参数），在读取文件时，将所有的制表符换成空格，同时保持单词以及空行之间的所有缩进、空格。
fmt 的命令格式为： **fmt [option] [file]**。 **option** 选项常用参数如表所示。

选项	功能描述
-c	保留段落前两行缩进
-p string	只格式化以字符串 string 开头的行
-s	根据指定列宽截断长行，短行不会与其他行合并
-t	除每个段落的第一行外，都缩进
-u	统一间隔符，字符之间间隔一个空格，句子之间间隔两个空格
-w width	每行文本不超过 width 个字符，默认值是 75

136

5.4.3 行标命令nl

nl 命令的功能很简单却很实用，为文本创建行号，如果不保存， **nl** 只会在输出中加入行号，阅读起来更加方便，不会影响原文件的文本内容。
nl 的命令格式为： **nl [option] [file]**，其中 **option** 选项的常用参数及含义如表所示。

选项	功能描述		
-v	设置起始编号，默认为 1		
-i	改变增量，默认值为 1		
-w	设置行号字段宽度，默认值为 6		
-b	正文编号	a	对所有行编号（包括空白行）
		t	仅对非空白行编号（默认项）
		n	不对任何行编号
		pregexp	只对与基本正则表达式匹配的行编号
-n	控制编号格式	ln	左对齐，无前导 0
		rn	右对齐，无前导 0
		rz	右对齐，有前导 0

137

5.4.4 指定行长度命令fold

fold 命令是对行进行操作，将文本行进行折叠，长行分解成短行。
fold 的命令格式为： **fold [option] [file]**
其中， **option** 选项常用参数及含义如表所示。

选项	功能描述
-b	按照字节计算宽度
-c	按照字符计算宽度
-s	不分割单词
-w	设置行宽（默认是 80）

注意：**fold** 命令的默认行宽是 **80**，如果想调整宽度就使用**-w** 选项。

138

5.5.1 awk文本分析工具

awk 是一种优良的文本处理工具，其名称来源于 **Bell** 实验室的三名开发者 **Aho**、**Wenberger** 和 **Kernighan** 姓氏的首字母组合，主要完成字符串查找、替换、加工等操作，还包含可以进行模式装入、流控制、数学运算、进程控制等语句。尽管 **awk** 具有完全属于其本身的语法，但在很多方面类似于 **shell** 编程语言。它的设计思想来源于**SNOBOL4**、**sed**、**Marc Rochkind** 设计的有效性语言、语言工具 **yacc** 和 **lex**，当然还从 **C** 语言中获取了一些优秀的思想。在最初创造 **awk** 时，其目的是用于文本处理，并且这种语言的基础是，只要在输入数据中有模式匹配，就执行一系列指令。该实用工具扫描文件中的每一行，查找与命令行中给定内容相匹配的模式，如果发现匹配内容，则进行下一个编程步骤；如果找不到匹配内容，则继续处理下一行。

awk 的语法格式为： **awk 'pattern {action}'file**

awk 扫描 **file** 中的每一行，对符合模式 **pattern** 的行执行操作 **action**。也可以只有 **pattern** 或者 **action**，在 **action** 操作中可能会用到一些特殊字符，常用的特殊字符及含义如下表所示。

139

5.5.1 awk文本分析工具

特殊字符	含义
$0	所有列
$1	第一列（$2 表示第二列，第 10 列用${10}表示）
NF	所有列数
NR	所有行数
-F	指定间隔符

通过 **awk** 操作 **cut_bc** 文件，为使结果更加清晰，在 **cut_bc** 文件中的文本添加行号，修改后的 **cut_bc** 文件内容如下。

```
[user@localhost five]$ cat cut_bc
1 Tom:Jones:4404
2 Mary:Adams:2980
3 Sally:Chang:9999
4 Billy:Black:6666
```


140

5.5.1 awk文本分析工具

（**1**）匹配有“**Sally**”的行。

```
[user@localhost five]$ awk '/Sally/' cut_bc
3 Sally:Chang:9999
```

（**2**）输出 **cut_bc** 文件中的第一列。

```
[user@localhost five]$ awk '{print $1}' cut_bc
1
2
3
4
```

（**3**）匹配有“**Sally**”的行，并输出此行的第二列。

```
[user@localhost five]$ awk '/Sally/ {print $2}' cut_bc
Sally:Chang:9999
```

（**4**） **-F** 指定间隔符。

```
[user@localhost five]$ awk -F: '{print $1}' cut_bc
1 Tom
2 Mary
3 Sally
4 Billy
```


141

5.5.2 sed编辑器

sed 是一个精简的、非交互式的编辑器，功能与 **vi** 编辑器相同，但不能进入文件进行编辑，只能在命令行下输入编辑命令，擅长对文本进行编辑，如替换文本。 **sed** 命令后跟 **s** 参数执行替换操作。例如，在 **cut_bc** 文件中搜索 **Sally** 字符串，并替换为 **sally**。

```
[user@localhost five]$ sed s/Sally/sally/g cut_bc
1 Tom:Jones:4404
2 Mary:Adams:2980
3 sally:Chang:9999
4 Billy:Black:6666
5 Adson:Blue:7809
```

默认情况下， **sed** 将输出写入标准输出，对原文件没有影响。例如，执行完上述替换操作后，查看 **cut_bc** 文件中的内容，会发现文件内容并没有变化，结果如下。

```
[user@localhost five]$ cat cut_bc
1 Tom:Jones:4404
2 Mary:Adams:2980
3 Sally:Chang:9999
4 Billy:Black:6666
5 Adson:Blue:7809
```


142

5.5.2 sed编辑器

如希望在进行替换操作时，改变原始文件，可以在 **sed** 命令后跟“**-i**”选项，但这种改变是永久性的，文本内容将发生变化，没有撤销的操作。例如，仍然将 **Sally** 字符串替换为 **sally**，但是命令添加**-i** 选项，执行命令及 **cut_bc** 文本变化如下。

```
[user@localhost five]$ sed -i s/Sally/sally/g cut_bc
[user@localhost five]$ cat cut_bc
1 Tom:Jones:4404
2 Mary:Adams:2980
3 sally:Chang:9999
4 Billy:Black:6666
5 Adson:Blue:7809
```

sed还可以只对指定的行进行操作，在**s**参数前添加行号如“**3s/sally/Sally/g**”表示只对第**3**行进行替换操作。指定行的范围，用逗号将两个行号隔开，如“**3,5 s/sally/Sally/g**”表示对第 **3** 行至第 **5** 行进行替换操作。最后一行一般用“**$**”表示。为使下面示例结果更加明显，在 **cut_bc** 文件中的“**5 Adson:Blue:7809**”行后添加“**3 sally:Chang:9999**”，修改后的文本如下。

143

5.5.2 sed编辑器

```
1 Tom:Jones:4404
2 Mary:Adams:2980
3 sally:Chang:9999
4 Billy:Black:6666
5 Adson:Blue:7809
6 sally:Chang:9999
```

然后只替换第 **3** 行的 **sally**，第 **6** 行中的 **sally** 并不会发生变化。执行命令及结果如下。

```
[user@localhost five]$ sed 3s/sally/Sally/g cut_bc
1 Tom:Jones:4404
2 Mary:Adams:2980
3 Sally:Chang:9999
4 Billy:Black:6666
5 Adson:Blue:7809
6 sally:Chang:9999
```


144

Linux操作系统基础教程

任课教师 刘昊霖
联系方式： liuhaolin@xtu.edu.cn

145

目 录 CONTENTS

146

第6章 Linux多命令协作

《Linux操作系统基础教程》

147

第6章 Linux多命令协作

148

6.1 命令行（CLI）数据流

在 **Linux** 系统中，命令行 **Shell** 数据流的定义如下表所示。在这里每个命令行定义了 **3** 个基本概念：标准输入（**standard input**，表示为**STDIN**）、标准输出（**standard out**，表示为 **STDOUT**）和标准错误（**standard error**，表示为 **STDERR**）。之所以这样分类，是为了方便管理命令行 **Shell** 中的数据流，并且可以通过管道和重定向机制控制 **CLI** 数据流。

名称	说明	编号	默认
STDIN	标准输入	0	键盘
STDOUT	标准输出	1	终端（屏幕）
STDERR	标准错误	2	终端（屏幕）

149

6.1 命令行（CLI）数据流

STDIN 是用来采集信息的，命令通过 **STDIN** 接收参数或数据，默认情况下，标准输入就是从键盘读入数据。**STDOUT** 和 **STDERR** 都是用来输出信息的，通过 **STDOUT**输出结果，通过 **STDERR** 输出状态或错误信息等。默认情况下，标准输出和标准错误都是直接在终端（即屏幕）显示，而且不会被保存到磁盘文件中。

一个命令可以把生成的输出内容发送到任意的数据流中，这些数据流一般分为三类：标准输入数据流、标准输出数据流、标准错误数据流。 **Shell** 在内部有特定的符号描述数据流，标准输入数据流编号为 **0**，标准输出数据流编号为 **1**，标准错误数据流编号为 **2**。

150

6.2.1 重定向标准输入

默认情况下，标准输入就是从键盘读入数据，每次一行，同时按 **Ctrl+D** 组合键结束输入数据，重新回到 **Shell** 命令环境。而重定向标准输入可以重新定义从文件中读入数据。通过重定向符“**<**”，可以把标准输入重定向到文件，即从文件中读入数据作为某条命令的输入数据。

例如，创建 **six** 目录，并将 **five** 目录中的 **cut_bc** 文件复制到 **six** 目录下，操作流程如下。

```
[user@localhost ~]$ mkdir six
[user@localhost ~]$ ls
Desktop Downloads Music Public Templates Videos
Documents five Pictures six test
[user@localhost ~]$ cd six
[user@localhost six]$ cp /home/user/five/cut_bc . (末尾有一个点，表示当前路径)
[user@localhost six]$ ls
cut_bc
```


151

6.2.1 重定向标准输入

利用 **cat** 命令接收标准输入数据（从键盘输入），只在终端输入 **cat** 命令，不带任何参数。

```
[user@localhost six]$ cat
```

此时光标会跳到下一行一直闪动，而没有其他任何反应，这是在等待从键盘读入数据。从键盘敲入想输入的内容，同时按 **Ctrl+D** 组合键结束输入。由于 **cat** 会把标准输入的内容显示在屏幕上，所以当输入一行，按下 **Enter** 键时，会显示刚刚输入的内容，在结束输入时，会有种重复显示的感觉。

```
[user@localhost six]$ cat
111111111
111111111
222222222
222222222
333333333
333333333
[user@localhost Desktop]$
```


152

6.2.1 重定向标准输入

当数据文件已经存在，不需要从键盘输入时，通过重定向符“**<**”，可以将输入源定向为已经存在的文件，文件中的内容会直接在屏幕上显示。例如，将读取的内容定向到 **cut_bc** 文件，在终端输入“**cat < cut_bc**”命令。

```
[user@localhost six]$ cat < cut_bc
1 Tom:Jones:4404
2 Mary:Adams:2980
3 Sally:Chang:9999
4 Billy:Black:6666
5 Adson:Blue:7809
```


153

6.2.2 重定向标准输出

默认情况下，标准输出在屏幕上显示，而重定向标准输出可以重新定义输出内容到文件。重定向标准输出有两种格式，一种是通过重定向符“**>**”把标准输出重定向到文件，即将标准输出内容保存到文件中，是覆盖操作。也就是说，如果目标文件不存在，则创建文件并将标准输出内容保存进去；如果目标文件存在，则覆盖其中的内容。另一种重定向符号“**>>**（中间没有空格）”是追加操作，实现连续保存文件中的内容，即原来的文本内容不会被覆盖，而是在文件尾部添加标准输出的内容。当然，如果文件不存在，也会自动创建。

例如，通过 **echo** 命令输出字符串“**I am happy**”，默认情况下，输出结果在终端显示。

```
[user@localhost six]$ echo "I am happy"
I am happy
```

但现在需要将输出结果“**I am happy**”保存到文件 **stdout** 中，此时就需要重定向改变输出流的方向，将输出内容保存到目标文件中。先用 **ls** 命令查看 **six** 目录下存在的文件，只有 **cut_bc**，目标文件 **stdout** 并不存在。执行重定向标准输出操作后，原来在命令行输出的字符串“**I am happy**”并未在终端显示。

154

6.2.2 重定向标准输出

```
[user@localhost six]$ ls
cut_bc
[user@localhost six]$ echo "I am happy" > stdout
[user@localhost six]$
```

再次使用 **ls** 命令查看，原来不存在的 **stdout** 文件此时已经存在，说明重定向标准输出符号后跟的目标文件如果不存在，就自动创建。通过 **cat** 命令查看 **stdout** 文件中的内容，字符串“**I am happy**”成功输出到 **stdout** 目标文件中。

```
[user@localhost six]$ ls
cut_bc stdout
[user@localhost six]$ cat stdout
I am happy
```


155

6.2.2 重定向标准输出

如果再执行一次重定向命令，将“**I am very happy**”字符串输出到 **stdout** 文件中，**stdout** 文件中的文本变化如下。

```
[user@localhost six]$ echo "I am very happy" > stdout
[user@localhost six]$ cat stdout
I am very happy
```

原来的文本“**I am happy**”已经替换为“**I am very happy**”，说明目标文件存在，原来的文本内容会被覆盖。如果不希望原文本被覆盖，则可以采用重定向符“**>>**”将新文本添加到目标文件中。例如，执行重定向操作，将“**I am fine**”输出到 **stdout** 文件中，原来的文本“**I am very happy**”则不会被覆盖。

```
[user@localhost six]$ echo "I am fine" >> stdout
[user@localhost six]$ cat stdout
I am very happy
I am fine
```


156

6.2.3 重定向标准错误

默认情况下，标准错误在屏幕上显示，而重定向标准错误可以重新定义输出错误内容到文件。重定向标准错误有两种格式，一种是通过重定向符“**2>**”把标准错误输出重定向到文件，即将标准错误内容保存到文件中，是覆盖操作。也就是说，如果目标文件不存在，则创建文件并将标准错误内容保存进去；如果目标文件存在，则覆盖其中的内容。此命令更多地用于日志中，执行一条指令可能有很多步操作，可是如果只想保存报错信息，就可用此命令。注意：如果指令正常执行了，即没有错误，就会发现它的标准输出在终端显示，因为没有错误，所以目标文件中不会保存任何内容。另一种重定向符号“**2>&1**（中间没有空格）”是将标准输出和标准错误结合在一起输出到文件，即将正确结果及错误全部输出到文件。从这里可以看到，标准输出和标准错误的数据流是分开输出的。

例如，查看 **stdout** 文件中的内容，却不小心写成了 **cat stdouu**，但是 **stdouu** 文件是不存在的，此时终端会显示错误信息提示命令有误。

```
[user@localhost six]$ cat stdouu
cat: stdouu: No such file or directory
```


157

6.2.3 重定向标准错误

也可以通过重定向将错误信息保存到文件中，以便后期查阅。将错误信息保存到文件中，通过重定向符“**2>**”实现。例如，将上述终端输出的错误信息“**cat: stdouu: No such file or directory**”保存到 **stderr** 文件中。

```
[user@localhost six]$ cat stdouu 2> stderr
[user@localhost six]$ cat stderr
cat: stdouu: No such file or directory
```

在终端执行“**cat cut_bc stdouu**”命令， **cut_bc** 文件存在，文件中的文本会在终端输出； **stdouu** 文件不存在，则会在终端输出错误信息。

```
[user@localhost six]$ cat cut_bc stdouu
1 Tom:Jones:4404
2 Mary:Adams:2980
3 Sally:Chang:9999
4 Billy:Black:6666
5 Adson:Blue:7809
cat: stdouu: No such file or directory
```


158

6.2.3 重定向标准错误

也可以通过重定向将错误信息保存到文件中，以便后期查阅。将错误信息保存到文件中，通过重定向符“**2>**”实现。例如，将上述终端输出的错误信息“**cat: stdouu: No such file or directory**”保存到 **stderr** 文件中。

```
[user@localhost six]$ cat stdouu 2> stderr
[user@localhost six]$ cat stderr
cat: stdouu: No such file or directory
```

在终端执行“**cat cut_bc stdouu**”命令， **cut_bc** 文件存在，文件中的文本会在终端输出； **stdouu** 文件不存在，则会在终端输出错误信息。

```
[user@localhost six]$ cat cut_bc stdouu
1 Tom:Jones:4404
2 Mary:Adams:2980
3 Sally:Chang:9999
4 Billy:Black:6666
5 Adson:Blue:7809
cat: stdouu: No such file or directory
```


159

6.2.3 重定向标准错误

因为重定向标准错误只能将错误信息保存到文件中，所以如果命令可以正确执行，结果作为标准输出在终端显示，不会被保存到目标文件中。例如，执行命令“**cat cut_bc stdouu 2> stderr**”，会发现屏幕上没有显示“**cat: stdouu: No such file or directory**”这一行。因为 **cut_bc** 文件存在，可以正确执行，虽然使用了重定向符号，但结果仍在终端显示，不会被保存到 **stderr** 文件中；而 **stdouu** 文件不存在，错误信息会被重定向保存到**stderr** 文件中。

```
[user@localhost six]$ cat cut_bc stdouu 2> stderr
1 Tom:Jones:4404
2 Mary:Adams:2980
3 Sally:Chang:9999
4 Billy:Black:6666
5 Adson:Blue:7809
[user@localhost six]$ cat stderr
cat: stdouu: No such file or directory
```


160

6.2.3 重定向标准错误

为实现标准输出与标准错误同时被重定向到目标文件中，命令格式为：**command>file 2>&1**，表示先将标准输出重定向到文件 **file** 中，然后“**2>&1**”将标准错误重定向到标准输出（数字 **1** 表示标准输出），由于标准输出已经重定向到 **file** 文件中，所以标准错误也会被重定向到目标文件 **file** 中，因而实现了标准输出与标准错误同时被保存到目标文件 **file** 中。

```
[user@localhost six]$ cat cut_bc stdouu > stderr 2>&1
[user@localhost six]$ cat stderr
1 Tom:Jones:4404
2 Mary:Adams:2980
3 Sally:Chang:9999
4 Billy:Black:6666
5 Adson:Blue:7809
cat: stdouu: No such file or directory
```


161

6.3.1 统计字数命令wc

wc（**word count**，单词统计）命令可以统计行、单词和字符的数量。很多时候，统计的数据来自于一个或多个文件，或者其他命令的标准输入，因此 **wc** 命令与管道息息相关，在分析文本内容时有很大的作用。

wc 的命令格式为： **wc [option] [file]**

其中， **option** 选项可以省略，控制 **wc** 命令的输出结果，常用的 **option** 选项参数如下表所示。 **file** 是 **wc** 命令要操作的文件的名称，也是可以省略的。

选项	功能描述
-l（小写 L）	统计行
-w	统计单词数
-c	统计字符数
-L	统计最长行的长度

下面通过具体示例理解 **wc** 命令的用法以及如何与管道结合使用。

162

6.3.1 统计字数命令wc

1．统计 **cut_bc** 文件的文本信息，**wc** 命令后不跟任何参数，输出结果如下。

```
[user@localhost six]$ wc cut_bc
5 10 91 cut_bc
```

此时 **wc** 命令没有带任何参数，输出结果即为默认输出。结果中包含了 **3** 个数字，这 **3** 个数字分别代表行数、单词数、字符数，即 **cut_bc** 文件中有 **5** 行、 **10** 个单词（注意： **Tom:Jones:4404** 算是一个单词）、 **91** 个字符。行、单词、字符的含义如下。

（**1**）行：以新行字符（如 **Enter** 键）结尾的一串字符。
（**2**）单词：是一串连续的字符，用空格、制表符或新行字符分隔。
（**3**）字符：字母、数字、标点符合、空格、制表符或新行字符。

2．同时统计多个文件

wc 命令同时统计多个文件时，每个文件的统计结果作为一行输出，并且在最后一行会显示所有文件总的行数、单词数、字符数。例如，同时统计 **cut_bc**、 **stdout** 中的文本信息。

```
[user@localhost six]$ wc cut_bc stdout
5 10 91 cut_bc
2 7 26 stdout
7 17 117 total
```


163

6.3.1 统计字数命令wc

3．只输出行数、单词数、字符数的一个或两个。

有时候需要的结果可能只是行数或者字符数，而不需要把 **3** 个数字都输出，此时可以在 **wc** 命令后添加选项参数来控制输出结果。例如，只输出 **cut_bc** 文件中的行数和单词数。

```
[user@localhost six]$ wc -lw cut_bc
5 10 cut_bc
```

4．**wc** 命令与管道结合。

因为 **wc** 的统计功能，所以 **wc** 在管道中也有非常重要的作用，可以统计其他命令的输出结果。例如， **Linux** 允许多用户同时登陆，先通过 **who** 命令查看系统当前登录的用户，一个用户的详细信息作为一行输入，所以统计 **who** 命令的输出结果就可以知道用户数量。因此通过“**who | wc -l**”将 **who** 命令的输出结果作为 **wc** 的输入，统计当前登录用户的数量。

```
[user@localhost six]$ who
user :0 2017-02-19 13:36 (:0)
user pts/0 2017-02-19 14:04 (:0)
[user@localhost six]$ who | wc -l
2
```


164

6.3.2 管道线分流命令tee

tee 命令的作用是从标准输入读取数据，并向标准输出和一个或更多的文件发送数据。即将读到的数据在终端显示的同时，还可以同时被保存到一个或多个文件中。也就是说，当需要把获得的数据在同一时刻发送到两个地方时（同时完成两个任务），就可以使用 **tee** 命令。

tee 的命令格式为： **tee [option] [file]**

其中， **option** 选项可以省略，控制文本添加方式。 **file** 是 **tee** 命令要操作的文件的名称，也是可以省略的。

例如，将 **cut_bc** 文件中的文本输出到屏幕，并保存到 **file1** 文件中。在不借助 **tee** 命令的情况下，要通过 **cat cut_bc**、 **cat cut_bc > file1** 两条命令来完成。

```
[user@localhost six]$ cat cut_bc
1 Tom:Jones:4404
2 Mary:Adams:2980
3 Sally:Chang:9999
4 Billy:Black:6666
5 Adson:Blue:7809
```


165

6.3.2 管道线分流命令tee

```
[user@localhost six]$ cat cut_bc > file1
[user@localhost six]$ cat file1
1 Tom:Jones:4404
2 Mary:Adams:2980
3 Sally:Chang:9999
4 Billy:Black:6666
5 Adson:Blue:7809
```

通过 **tee** 将两条命令连接起来，同样实现上述功能， **cut_bc** 文件中的文本在屏幕输出的同时被成功保存到 **file1** 文件中，因为是两个命令的组合，所以需要借助管道来完成。

到此为止，已将 **cut_bc** 文件中的文本两次保存到 **file1** 文件中，但是目前 **file1** 文件中只包含一次的文本信息。这是因为 **tee** 命令的默认输出就是标准输出，也就是直接在屏幕上显示。如果 **tee** 后面跟的文件不存在，就自动创建；如果存在，则将原来的内容覆盖。如果希望在目标文件的文本末尾添加内容，就要在 **tee** 命令后添加**-a** 选项参数。例如，在屏幕输出 **stdout** 文件中的文本，并将其文本添加到 **file1** 文件中。

166

6.3.2 管道线分流命令tee

```
[user@localhost six]$ cat stdout | tee -a file1
I am very happy
I am fine
[user@localhost six]$ cat file1
1 Tom:Jones:4404
2 Mary:Adams:2980
3 Sally:Chang:9999
4 Billy:Black:6666
5 Adson:Blue:7809
I am very happy
I am fine
```

因为管道允许同时使用多个命令，所以可以同时使用 **cat** 命令、 **tee** 命令、 **wc** 命令。例如，将 **cut_bc** 中的文本保存到 **file1** 中，并统计 **file1** 文本的行数。

```
[user@localhost six]$ cat cut_bc | tee file1 | wc -l
5
[user@localhost six]$ cat file1
1 Tom:Jones:4404
2 Mary:Adams:2980
3 Sally:Chang:9999
```


167

6.3.2 管道线分流命令tee

```
4 Billy:Black:6666
5 Adson:Blue:7809
```

tee 命令后可以添加多个文件名，例如，把 **cut_bc** 中的内容保存到 **file1**、**file2** 中，并统计行数，执行命令和结果如下。

```
[user@localhost six]$ cat cut_bc | tee file1 file2 | wc -l
5
[user@localhost six]$ cat file1
1 Tom:Jones:4404
2 Mary:Adams:2980
3 Sally:Chang:9999
4 Billy:Black:6666
5 Adson:Blue:7809
[user@localhost six]$ cat file2
1 Tom:Jones:4404
2 Mary:Adams:2980
3 Sally:Chang:9999
4 Billy:Black:6666
5 Adson:Blue:7809
```


168

6.3.3 查找重复行命令uniq

uniq 命令会一行一行地检查数据，查找出连续重复的行。注意 **uniq** 只能查找有序的文件，所以重复行一定是连续的。 **uniq** 命令可以执行 **4** 项不同的任务：消除重复行、选取重复行、选取唯一行和统计重复行的数量。

uniq 的命令格式为： **uniq [option] [input [output]]**

其中， **option** 选项可以省略，常用参数如下表所示。 **input** 是输入文件，可以省略，若指定了该参数， **uniq** 命令从该文件读入数据。 **output** 是输出文件，同样可以省略，若指定了该参数，则 **uniq** 命令将输出结果保存到该文件中。

选项	功能描述
-c	在输出行前面加上每行输入文件中出现的次数
-d	仅显示重复行
-u	仅显示不重复的行

为了使 **uniq** 命令结果更加明显，对 **cut_bc** 文件做以下修改，修改后的内容如下。

```
[user@localhost six]$ cat cut_bc
1 Tom:Jones:4404
```

2021/12/2

169

6.3.3 查找重复行命令uniq

```
2 Mary:Adams:2980
1 Tom:Jones:4404
3 Sally:Chang:9999
4 Billy:Black:6666
5 Adson:Blue:7809
4 Billy:Black:6666
```

以下示例为保证文件是有序的，通过管道，将 **sort** 的排序结果传入 **uniq** 命令。

（**1**）消除重复行， **uniq** 命令后不跟任何参数，输出结果会屏蔽重复行。

```
[user@localhost six]$ sort cut_bc | uniq
1 Tom:Jones:4404
2 Mary:Adams:2980
3 Sally:Chang:9999
4 Billy:Black:6666
5 Adson:Blue:7809
```

（**2**）只输出重复行。

```
[user@localhost six]$ sort cut_bc | uniq -d
1 Tom:Jones:4404
4 Billy:Black:6666
```

2021/12/2

170

6.3.3 查找重复行命令uniq

（**3**）仅显示不重复的行。

```
[user@localhost six]$ sort cut_bc | uniq -u
2 Mary:Adams:2980
3 Sally:Chang:9999
5 Adson:Blue:7809
```

（**4**）在输出行前面添加出现的次数（注意：第一个数字是结果中的次数，第二个数字是文本中原来就存在的，不要混淆）。

```
[user@localhost six]$ sort cut_bc | uniq -c
2 1 Tom:Jones:4404
1 2 Mary:Adams:2980
1 3 Sally:Chang:9999
2 4 Billy:Black:6666
1 5 Adson:Blue:7809
```

2021/12/2

171

Linux操作系统基础教程

任课教师 刘灵霖
联系方式： liuhaolin@xtu.edu.cn

172

目 录 CONTENTS

2021/12/2

173

第7章 Shell编程

《Linux操作系统基础教程》

174

第7章 Shell编程

- 7.1 Linux编程基础
  - 7.1.1 使用gcc编译C程序
  - 7.1.2 使用make编译C程序
  - 7.1.3 通过编译源代码安装程序
- 7.2 Shell脚本
  - 7.2.1 什么是Shell脚本
  - 7.2.2 开始编写Shell脚本
- 7.3 变量及其使用方法
  - 7.3.1 Shell变量和环境变量
  - 7.3.2 变量的操作

2021/12/2

175

第7章 Shell编程

- 7.4 输入、输出和引用
  - 7.4.1 输入与输出
  - 7.4.2 引用
- 7.5 分支控制语句
  - 7.5.1 if语句
  - 7.5.2 case语句
- 7.6 循环控制语句
  - 7.6.1 while和until循环
  - 7.6.2 for循环
  - 7.6.3 跳出循环

2021/12/2

176

第7章 Shell编程

- 7.7 位置参数
  - 7.7.1 获取位置参数
  - 7.7.2 位置参数使用案例
- 7.8 数组
  - 7.8.1 为什么使用数组
  - 7.8.2 数组的创建、赋值和删除
  - 7.8.3 遍历访问数组元素

2021/12/2

177

第7章 Shell编程

- 7.9 函数
  - 7.9.1 函数的定义与调用
  - 7.9.2 在函数中使用位置参数
  - 7.9.3 使用函数返回值
  - 7.9.4 将函数保存到文件

2021/12/2

178

7.1.1 使用gcc编译C程序

我们使用的计算机在与低层硬件交互时使用一种称为机器语言的程序。机器语言是由一系列二进制指令组成的，这些指令描述了一些非常基本的操作，如“指向内存中某个位置”“写入一字节”“删除一字节”等。如果程序员以这样的方式操作计算机将极其低效并难以理解，因此 **Linux** 提供了编译器将高级语言或汇编语言转化为机器语言。**gcc** 便是 **Linux** 环境中最常用的编译器。

gcc（**GNU Compiler Collection**）是 **GNU** 推出的多平台编译器，支持编译 **C**、**C++**、**Java**、**Objective C**、**Fortran** 等多种语言。**CentOS** 默认已经安装了 **gcc** 编译器，读者也可以使用以下命令自行安装。

yum install gcc

下面以编译 **C** 程序为例，介绍 **gcc** 的用法。在学习 **C** 语言时，我们都知道使用编译器编译 **C** 语言源代码经历了两个步骤：先将源代码编译成后缀名为**.o** 的目标文件，也就是机器语言；然后链接**.o** 文件，生成可执行文件。在 **Linux** 下使用 **gcc** 命令可以一次性完成这些工作。

gcc 的命令格式为：**gcc [options] [file]**

假设待编译的程序为当前目录下的 **hello.c** 文件，代码内容如下。

2021/12/2

179

7.1.1 使用gcc编译C程序

```
#include<stdio.h>
main()
{
    printf("Hello World!\n")
}
```

使用 **gcc** 命令编译此文件：

```
[user@localhost ~]$ gcc -o hello hello.c
[user@localhost ~]$ ./hello
Hello World!
[user@localhost ~]$
```

可以发现 **hello.c** 已被编译成可执行文件，位置由**-o** 选项设置，如果未设置**-o** 选项，编译结果为当前目录下的 **a.out** 文件。

当程序依赖一个以上的文件时，可以先将每个文件编译成目标文件，再把所有目标文件链接成可执行文件。例如，**hello.c** 的 **main** 函数调用 **greeting.c** 中的 **func** 函数，代码如下。

2021/12/2

180

7.1.1 使用gcc编译C程序

hello.c 文件内容：

```
#include<stdio.h>
#include"greeting.h"
main()
{
    func("Tom");
}
```

greeting.c 文件内容：

```
void func(char *str)
{
    printf("Hello %s!\n",str);
}
```

greeting.h 头文件内容如下。

```
#ifndef _H_GREETING
#define _H_GREETING
void greeting(char *str);
#endif
```


181

7.1.1 使用gcc编译C程序

使用 **gcc** 命令的**-c** 选项，将**.c** 文件编译成**.o** 文件，然后将所有**.o** 文件链接成可执行文件。

```
[user@localhost ~]$ gcc -c greeting.c
[user@localhost ~]$ gcc -c hello.c
[user@localhost ~]$ gcc -o hello hello.o greeting.o
[user@localhost ~]$ ./hello
Hello Tom!
[user@localhost ~]$
```

这样就完成了多文件依赖程序的编译。

182

7.1.2 使用make编译C程序

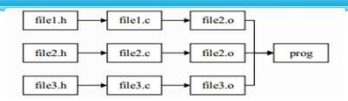
make 可以获知所管理项目中源文件的修改情况，根据程序员设定的规则，自动编译被修改过的部分，而那些没有修改的部分将不会重新编译。这样既保证了程序的正确性，又大大提高了项目开发的效率。

那么 **make** 是如何知晓哪些文件被修改了？需要执行什么指令才能保证程序的正确？这涉及一个重要的文件——**makefile**，**make** 通过 **makefile** 文件描述的内容自动维护编译工作。**makefile** 文件需要程序员按照某种格式编写，并说明项目中各个源文件之间的依赖情况。在 **Linux** 系统中，**makefile** 文件通常以 **MakeFile** 为文件名。为了说明 **make** 和 **makefile** 的工作原理，下面用一个简单的例子加以说明。

假设程序 **prog** 由 **3** 个源文件 **file1.c**、**file2.c** 和 **file3.c** 编译生成，这 **3** 个源文件有各自的头文件 **file1.h**、**file2.h** 和 **file3.h**。通常情况下，编译器会生成 **3** 个目标文件 **file1.o**、**file2.o** 和 **file3.o**，然后用这 **3** 个目标文件链接成 **prog** 程序，其过程如下图所示。

183

7.1.2 使用make编译C程序



要使用 **make** 对 **prog** 程序进行管理，则 **makefile** 文件应按如下内容编写。

```
prog:file1.o file2.o file3.o
gcc -o prog file1.o file2.o file3.o
file1.o:file1.c file1.h
gcc -c file1.c
file2.o:file2.c file2.h
gcc -c file2.c
file3.o:file3.c file3.h
gcc -c file3.c
```

在该 **makefile** 文件中，第一行说明了程序 **prog** 由 **3** 个目标文件 **file1.o**、**file2.o** 和 **file3.o** 链接生成，第 **3**、第 **5**、第 **7** 行又说明了这三个目标文件依赖的 **.c** 文件及 **.h**文件。而第 **2**、第 **4**、第 **6**、第 **8** 行则是根据这些依赖关系，编译目标文件或可执行文件。

184

7.1.2 使用make编译C程序

make 的命令格式为：**make [flags] [macro definitions] [targets]**

其中，**flags** 为标志位，常用的标志位选项如下表所示；**macro definition** 为宏命令，在这里指定的宏命令将覆盖 **makefile** 文件中的宏命令；**targets** 为要编译的文件，允许定义多个目标文件，按从左到右的顺序依次编译，如果此项缺省，则默认指向 **makefile** 文件中第一个目标文件。

选项	说明
-f file	指定 file 文件为 makefile 文件。若此项缺省，则系统默认指定当前目录下名为 makefile 的文件
-i	忽略命令执行返回的错误信息
-s	沉默模式
-r	禁止使用 build-in 规则
-n	非执行模式，输出所有执行命令，但不执行
-q	make 操作将根据目标文件是否已更新返回 0 或非 0 的状态信息
-p	输出所有宏定义和目标文件描述
-d	Debug 模式，输出有关文件和检测时间的详细信息
-c dir	在指定目录 dir 下读取 makefile 文件
-I dir	当包含多个 makefile 文件时，利用 dir 指定搜索目录
-w	在处理 makefile 之前和之后都显示工作目录

185

7.1.3 通过编译源代码安装程序

许多发行商将自己开发的软件预编译成二进制库，用户下载解压后就能马上使用。尽管这样十分方便，但很多时候也需要通过编译源代码安装软件，这是由于：

（**1**）软件开发商在更新版本时，会开发一些全新的功能，但为了程序的稳定性并不会将其加入当前的发现版本。因此想要获取最新的功能，必须通过编译源代码。

（**2**）有时候软件并不能满足用户的全部需求，用户希望在程序中加入自定义的部分。这种情况也需要编译源代码。

在 **Linux** 系统中，许多程序都是直接提供源代码的，这样就可以利用 **7.1.2** 节介绍的 **make** 和 **makefile** 文件编译源代码并完成程序的安装。

pcre（**Perl Compatible Regular Expressions**）是一个 **Perl** 库，用于代替庞大的 **Boost** 来解决 **C** 语言中使用正则表达式的问题。

下面以安装 **pcre** 为例，介绍通过编译源代码安装程序的步骤。

186

7.1.3 通过编译源代码安装程序

（1）登录 **pcre** 官方网站 **http://www.pcre.org/**下载最新版本的 **pcre** 源代码，这里以 **pcre2-10.23** 为例。其中 **wget** 命令用来从指定的 **URL** 下载文件。
（2）解压 **tar** 文件。
（3）在目录中可以发现一个名为 **configure** 的脚本程序，它随着源代码一起发布。**configure** 脚本的作用是分析当前系统的环境，并且检查系统是否已经安装了必要的外部工具和组件，然后生成合适的 **makefile** 文件以便下一步编译。目前许多软件都是设计成可移植的，程序的源代码可以在各种 **UNIX** 系统上编译，在编译时，各系统之间会有细微的不同，因此需要 **configure** 进行调整。另外， **configure** 还可以使用选项 “**--prefix**” 指定程序的安装路径，默认路径为**/usr/local**。
（4）运行 **configure**。
（5）如果在检查过程中，发现了某些导致安装无法进行的问题，如缺少开发用的某些软件或开发库， **configure** 会以失败告终。若没有发现此类问题，则可以使用 **make** 命令编译程序。
（6）若编译顺利完成，则可以使用命令 **make install** 进行安装。该命令会在安装目录下生成可执行程序。

187

7.2.1 什么是Shell脚本

到目前为止，我们都是以用户交互接口的方式使用 **Shell** 的，即人工通过输入设备在 **CLI** 命令行中输入命令，等待系统执行并将结果打印在屏幕上。设想一下，如果需要重复完成一个需要输入多条命令的任务，采用人工在命令行中一条一条输入的方法十分繁琐且容易出错，如果让 **Shell** 记住这些命令并自动完成输入将会大大提升效率（可以联想到上一节提到的 **make**）。因此将命令通过设计与组合后，记录到一个特定的文件中， **Shell** 就作为命令解析器执行文件中的一系列命令， 这里的文件就是 **Shell** 脚本。
简单来说， **Shell** 脚本是一个包含一系列命令的文件。 **Shell** 读取这个文件，然后执行这些命令，就好像这些命令是直接输入命令行中一样。从这个角度看，作为用户交互接口的 **Shell** 和作为命令解析器的 **Shell** 所做的工作是完全一样的，大多数能在命令行中完成的工作都可以在 **Shell** 脚本中完成，反之亦然。而使用 **Shell** 脚本的原因除了刚才提到的效率问题外，还基于以下 **3** 点考虑。

188

7.2.1 什么是Shell脚本

（1）简单性： **Shell** 是一个高级语言；通过它，可以简洁地表达复杂的操作。
（2）可移植性：使用 **POSIX** 定义的功能，可以做到脚本无须修改就可在不同的系统上执行。
（3）开发容易：可以在短时间内完成一个功能强大又好用的脚本。
作为用户交互接口的 **Shell** 称为交互式 **Shell**，而作为命令解析器的 **Shell** 称为非交互式 **Shell**。因为它只需要通过 **Shell** 脚本就可以完成工作，不需要人为干预。需要特别注意的是，交互式 **Shell** 和非交互式 **Shell** 指的都是同一个 **Shell**。换句话说， **Shell** 既是交互式的又是非交互式的，这取决于用户如何使用它。

189

7.2.2 开始编写Shell脚本

Shell 脚本本质上是 **Linux** 系统下的文本文件，通过第 **5** 章的学习我们已经掌握文本进行处理的方法。运用 **vim** 文本编辑器提供的“语法高亮”功能，可以很方便地编写 **Shell** 脚本。仍然以经典的 **HelloWorld** 程序为例，启动 **vim** 文本编辑器并输入以下内容。

```
#!/bin/bash
#My first shell script.
echo Hello World!
```

脚本第一行开头的“**#!**”是一个约定的标记，称为 **shebang**，用来告知操作系统需要用什么解析器来执行此脚本，这里表示使用 **bash** 来执行 **Shell** 脚本；第二行为注释，**Shell** 的注释以“**#**”开头，与所有编程语言一样注释的内容不会执行，在命令行中也是如此。例如：

```
[user@localhost ~]$ echo 'Hello World!' #This is a annotation
Hello World!
```


190

7.2.2 开始编写Shell脚本

编写完后将脚本保存为 **HelloWorld.sh**，这里的以 **sh** 为脚本扩展名并没有什么特殊含义，仅为了表明这是一个 **Shell** 脚本，达到见名知义的目的，类似于 **python** 脚本的**.py** 或 **php** 脚本的**.php**。
然后给文件增加可执行权限，并运行。

```
[user@localhost ~]$ chmod +x HelloWorld.sh
[user@localhost ~]$ ./HelloWorld.sh
Hello World!
[user@localhost ~]$
```

可以看到， **Shell** 脚本的执行结果和直接在命令行中输入命令的结果完全相同。在上面的例子中，使用三行内容的 **Shell** 脚本完成了命令行中一行命令的工作，这是为了向读者介绍编写 **Shell** 脚本最基本的方法，在本章后面的内容中将展示 **Shell** 编程强大的功能。

191

7.3.1 Shell变量和环境变量

1．Shell 变量
首先回顾一下在学习各种编程语言时对变量的定义。变量是一个用来存储数据的实体。每个变量都有一个变量名和一个值，其中变量名是引用变量的标识符，值是存储在变量中的数据。
与许多编程语言的变量一样， **Shell** 变量在命名时需要遵守一些规则：变量名必须由大写字母（**A**～**Z**）、小写字母（**a**～**z**）、数字（**0**～**9**）或下画线（_）构成；变量名的第一个字符不能是数字。
对于变量的值，大多数编程语言都可以包含多种不同类型的数据，而 **Shell** 变量只有字符串一种类型，即无论给 **Shell** 变量赋予什么值，在存储时都会转换为字符串。**Shell** 变量只能在创建它的 **Shell** 中使用，对于其他 **Shell** 是不可见的，并且 **Shell** 变量也不会从父进程传递给子进程，这一点与局部变量非常相似。因此在编写 **Shell** 程序过程中，当需要临时存储时，可以使用 **Shell** 变量。

192

7.3.1 Shell变量和环境变量

2. 环境变量

在学习各种编程语言的变量时，往往会提及全局变量的概念。全局变量可以在程序的任何地方使用，通常设计成可供函数在任何时候进行获取和修改。在 **Shell** 编程中，这一功能由环境变量实现。环境变量是一种特殊的变量类型，系统中的所有进程都可以使用， 这与全局变量十分相似， 但环境变量并不完全等同于全局变量，因为子进程对环境变量的修改不会传递到父进程中，关于进程的内容将在第 **8** 章详细介绍。

环境变量的命名和值类型与 **Shell** 变量相同，但需要注意的是，在 **bash** 中，系统定义的环境变量全部使用大写字母命名，如 **PATH**。下表列出了 **bash** 中常用的环境变量以及它们的说明，其他类型的 **Shell** 环境变量的命名会有区别。

环境变量	说明
PATH	指定命令的搜索路径
PWD	当前工作目录
HOME	当前用户的主工作目录
SHELL	当前用户使用的 Shell
TERM	当前正在使用的终端类型
USER	当前用户标识
HOSTNAME	当前主机的名称
LOGNAME	当前用户标识
ENV	环境文件的名称

193

7.3.1 Shell变量和环境变量

下面使用 **PATH** 介绍环境变量具有的全局性， **PATH** 表示命令执行时系统的搜索路径，输出 **PATH** 的值如下。

```
[user@localhost ~]$ echo $PATH
/usr/lib64/qt-3.3/bin:/usr/local/bin:/usr/local/sbin:/usr/bin:/usr/sbin:/bin:/sbin:/home/user/.local/bin:/home/user/bin
```

命令执行时，**Shell** 会在以上目录中寻找命令。回顾上面的 **HelloWorld.sh** 脚本，在执行时是通过 **“./”**的方式显式指定脚本的路径。而对于一些常用的脚本，通常会把它们放在 **PATH** 指定的目录里，运行时 **Shell** 自动搜索，而不需要用户指定路径。因此将 **HelloWorld.sh** 脚本复制到 **PATH** 中的一个名为 **“/home/user/bin”** 的目录，这个目录是存放个人脚本的理想位置，然后再执行脚本。

```
[user@localhost ~]$ cp HelloWorld.sh /home/user/bin/
[user@localhost ~]$ HelloWorld.sh
Hello World!
[user@localhost ~]$
```

此时就不必显式指定脚本的路径了，这就是环境变量发挥的作用。

194

7.3.2 变量的操作

无论是 **Shell** 变量还是环境变量，对于它们的操作可以归纳为 **4** 种：创建变量、获取变量的值、修改变量的值和删除变量。

1. 创建变量

变量的创建十分简单，只需要指定变量名称和变量值，它们之间用等号（**=**）连接，等号两边不能有空格。创建变量的语法为：

NAME=value

变量创建好后，可以使用**$**符号后面接变量名的方式获取变量的值。例如，定义一个名为 **os** 的变量，它的值为 **CentOS**，然后用 **echo** 命令输出变量的值。

```
[user@localhost ~]$ os=CentOS
[user@localhost ~]$ echo $os
CentOS
```

变量的值并不是必须的，如果变量在创建时没有赋值，则系统默认此变量的值为 **null**。另外，当使用**$**符号获取一个并没有被创建过的变量时，系统会自动创建变量，并为变量赋值为 **null**。例如，获取一个没有被创建的变量 **os1**。

```
[user@localhost ~]$ echo $os1

[user@localhost ~]$
```


195

7.3.2 变量的操作

2. 获取变量的值

前面已经提到过，可以使用**$**符号获取变量的值。然而考虑一种情况，如果变量后面需要紧跟其他字符将怎么办呢？如需要将一个名为 **file** 的文件改名为 **file1**， 用户可以通过 **mv** 命令快速实现。而在实际编程当中，往往不会显式地给出文件名，而是用变量的方式存储文件名，就目前学到的知识可以按下面的方式执行 **mv** 命令。

```
[user@localhost ~]$ fname=file
[user@localhost ~]$ mv $fname $fname1
mv: missing destination file operand after ‘file’
Try 'mv --help' for more information.
[user@localhost ~]$
```

在系统的错误提示中可以看到，在 **file** 后面缺少了参数。这是因为系统把 **fname1** 当成了一个变量，因为此变量没有被创建，所以取值时为空值。为了区分变量名和变量名后面紧跟的字符，可以使用花括号**{}**将变量名括起来。

```
[user@localhost ~]$ mv $fname ${fname}1
[user@localhost ~]$
```

这样 **Shell** 就不会把 **1** 当成变量名的一部分了。

196

7.3.2 变量的操作

3. 变量的修改与删除

修改变量与创建变量使用相同的语法，如果变量已经存在，重复对它进行赋值就可以改变变量的值。例如，将 **os** 变量的值由 **CentOS** 改为 **ubuntu**。

```
[user@localhost ~]$ os=CentOS
[user@localhost ~]$ os=ubuntu
[user@localhost ~]$ echo $os
ubuntu
```

环境变量的修改与 **Shell** 变量有所不同，因为子进程对环境变量的修改不会传递到父进程中。例如，在下面的例子中，编写脚本修改 **PATH** 环境变量，在当前 **Shell** 中执行此脚本，再查看当前 **PATH** 环境变量的情况。

编写脚本 **test** 如下。

```
PATH="PATH has been removed"
echo $PATH
```

运行脚本，并查看 **PATH** 环境变量。

```
[user@localhost ~]$ ./test
PATH has been removed
```


197

7.3.2 变量的操作

```
[user@localhost ~]$ echo $PATH
/usr/lib64/qt-3.3/bin:/usr/local/bin:/usr/local/sbin:/usr/bin:/usr/sbin:/bin:/sbin:/home/user/.l ocal/bin:/home/user/bin
[user@localhost ~]$ HelloWorld.sh
Hello World!
```

对于 **Shell** 变量来说，通常不需要主动删除变量，因为 **Shell** 变量的生存周期只在当前 **Shell**。但是如果需要的话，可以使用 **unset** 命令。

unset 命令的格式为： **unset NAME**

unset 命令同样可以删除环境变量。值得注意的是，用户可以通过 **export** 命令将 **Shell** 变量导出为环境变量，但没有办法将环境变量再恢复成 **Shell** 变量。换句话说，移除一个环境变量唯一的办法就是删除它。

198

7.4.1 输入与输出

1. 输出

在输出操作中使用得最多的是 **echo** 命令， **echo** 命令的功能是将字符串输出到屏幕。

echo 命令的格式为： **echo [-ne] [string]**

其中，**string** 表示要输出的字符串。选项 **n** 表示输出不换行，例如：

```
[user@localhost ~]$ echo -n Hello;echo ' World!'
Hello World!
```

选项 **e** 表示处理特殊字符，例如，在下面的例子中，**echo** 命令识别并输出制表符“**\t**”。

```
[user@localhost ~]$ echo -e 'Hello\tWorld!'
Hello World!
```

除了 **echo** 外，还有一种功能更强大的输出命令——**printf**。**Shell** 中的 **printf** 命令与 **C** 语言中的 **printf** 函数非常相似，功能都是格式化输出数据。

printf 命令的格式为： **printf format [arguments...]**

199

7.4.1 输入与输出

其中， **format** 为输出的格式，**arguments** 为要输出的数据。使用 **printf** 输出时必须指定数据的格式，例如：

```
[user@localhost ~]$ printf "%s\n" 'Hello World!'
Hello World!
```

其中格式部分用引号包围，单引号或双引号都可以。“**%s**”为格式符，表示输出的格式为字符串，类似的还有“**%d**”、“**%c**”、“**%f**”等，代表的格式与 **C** 语言中的相同。与 **echo** 命令不同的是， **printf** 不会自动换行，所以要达到换行的效果，就需要在格式最后加上换行符“**\n**”。

200

7.4.1 输入与输出

2. 输入

有时候编写程序需要考虑到程序的交互性，更准确的话应该是程序与用户的互动能力。增加互动性最直接的办法就是从用户获取输入。在 **Shell** 中获取用户输入可以使用 **read** 命令。

read 命令的作用是从标注输入读取一行数据。 此命令可以用于读取键盘输入或应用重定向读取文件中的一行。

read 命令的格式为： **read [options] [variable...]**

先通过一个简单的例子来感受一下 **read** 的用法。修改 **HelloWorld.sh** 脚本如下。

```
#!/bin/bash
#My first shell script.
echo Hello World!
read os
echo My Linux distribution is $os
```

然后运行脚本。

```
[user@localhost ~]$ ./HelloWorld.sh
Hello World!
```


201

7.4.1 输入与输出

此时程序执行完 **echo** 的输出，等待 **read** 命令的输入。输入 **CentOS** 并按 **Enter** 键。

```
[user@localhost ~]$ ./HelloWorld.sh
Hello World!
CentOS
My Linux distribution is CentOS
```

read常用命令选项如表所示。

特殊字符	说明
-a array	将输入值从索引为 0 的位置开始赋值给 array
-d str	将字符串 str 的第一个字符作为输入的结束
-e	使用 readline 处理输入
-n num	从输入中读取 num 个字符
-p str	使用字符串 str 作为输入提示
-r	不将“\”当作转义符号
-s	输入时不显示输入的字符
-t sec	在等待 sec 秒后结束输入，并返回非 0 退出状态
-u fd	从文件说明符 fd 读取输入

202

7.4.2 引用

在 **Linux** 的日常使用中，需要输入各种命令。命令由字母、数字和一些其他字符组成，某些字符有其特殊的含义，如表示单条命令结束的分号“；”，重定向符**<**和**>**，以及刚提到的变量取值符**$**等，我们将这些具有特殊含义的字符称为元字符。如果在输出时包含了元字符，但并不想使用它们的特殊功能该怎么办呢？

例如，想输出的内容是：“**This pen is $1**”，可能读者会按照下面的格式输入

```
[user@localhost ~]$ echo This pen is $1
This pen is
```

这并不是我们预期的输出，因为 **Shell** 使用了**$**的特殊功能，把 **1** 当成了变量。在这种情况中，我们可以采用转义字符“****”加元字符的方式告诉 **Shell** 不使用特殊功能，并原样输出。

```
[user@localhost ~]$ echo This pen is \$1
This pen is $1
```

在上面的例子中，使用转义字符取消了元字符的特殊含义，这样的操作称为引用。理论上，转义字符可以满足所有需要引用的情况。但是如果需要引用的字符过多，插入转义字符时会降低命令的可读性。

```
[user@localhost ~]$ echo I am using Linux\(CentOS\)\;My username is \<$USER\>.
I am using Linux(CentOS);My username is <user>.
```


203

7.4.2 引用

这样使用多个转义字符，不仅麻烦而且命令不易阅读。为此 **Linux** 还提供了两种引用的方式：单引号引用和双引号引用。

以下是使用单引号的例子。

```
[user@localhost ~]$ echo 'I am using Linux(CentOS);My username is <$USER>.'
I am using Linux(CentOS);My username is <$USER>.
```

单引号之间的所有内容都被引用了，这样做比使用转义字符高效许多。

3 种引用方式，读者可以根据实际需求选择使用任意一种或几种。

（**1**）转义字符：用于引用任意的单个字符。

（**2**）单引号引用：也称为强引用，用于引用包含的字符串。

（**3**）双引号引用：也称为弱引用，用于引用包含的字符串，但保留**$**、\和`的特殊含义。

204

7.5.1 if语句

if 语句是最常见的分支语句，语法格式如下。

```
if expression; then
   command...
[elif expression; then
   command...]
[else
   command...]
fi
```

通过命令行中的一个简单例子说明 **if** 语句的用法。

```
[user@localhost ~]$ if true; then echo "it's true"; else echo "it's false";
fi
it's true
[user@localhost ~]$ if false; then echo "it's true"; else echo "it's false";
fi
it's false
```


205

7.5.1 if语句

执行哪一个分支由 **if** 后的表达式结果决定。需要创建多分支时，可以使用 **elif** 指定其余分支。

从 **if** 语句的结构可以看出，关键字 **if**、 **else**、 **elif** 等决定了语句的分支情况，而表达式决定了执行哪一条分支。表达式根据实际需求变化，在复杂的判断条件中，需要使用 **test** 命令或组合表达式的方式完成。

1. test 命令

test 命令经常出现在 **if** 语句表达式的编写中，用于执行各种检查和比较。

test 命令的格式为：**[expression]**

先用一个例子说明 **test** 命令的使用方法。

```
if read -t 5 -p 'Enter the result of 4+4 in 5 seconds:' result; then
      if [ $result -eq 8 ] ; then
        echo "The result is correct!"
      else
        echo "The result is wrong!"
      fi
 else
      echo -e "\nTime out!"
 fi
```


206

7.5.1 if语句

现在分析这个脚本。最外层的 **if** 语句后的表达式是一个 **read** 命令， **-t** 选项的存在使得 **read** 可以根据输入延迟返回不同的状态值，如果在 **5** 秒内输入，则返回 **0** 状态值，程序进入内层嵌套的 **if** 语句，否则进入外层 **else** 语句。在内层 **if** 语句中判断 **result** 变量的值，方括号之间的内容为 **test** 命令，其中“**-eq**”是判断两个整数是否相等的表达式。整个 **test** 命令表示如果变量 **result** 的值等于 **8**，就返回 **TRUE**，否则返回 **FALSE**。通过 **if** 语句的分支控制，实现了程序判断对错和限制回答时间的功能。

bash 还提供了一个增强的 **test** 命令，命令格式如下。

[[expression]]

使用这种格式的 **test** 命令支持所有 **test** 命令表达式，并且增加了一个重要的功能——支持正则表达式匹配。匹配符号为“**=~**”，若“**=~**”左边的字符串匹配了“**=~**”右边的正则表达式，则返回 **TRUE**，反之返回 **FALSE**。

2. 组合表达式

有些时候分支的判定条件比较复杂， **if** 语句可以使用逻辑运算符将表达式组合起来，实现更复杂的计算。可用的逻辑运算符有 **3** 种，分别为逻辑与、逻辑或和逻辑非。这 **3** 种逻辑运算符的操作符与说明如下表所示。

207

7.5.1 if语句

名称	操作符	说明
逻辑与	&&	若操作符两边的表达式都为真，返回 TRUE，否则返回 FALSE
逻辑或	||	若操作符两边的表达式都为假，返回 FALSE，否则返回 TRUE
逻辑非	!	若操作符后的表达式为真，返回 FALSE，否则返回 TRUE

下面是一个逻辑与的使用例子。这个脚本可用来检查一个数是否属于某个范围。

```
#!/bin/bash
num=50
if [ $num -ge 40 ]&&[ $num -le 60 ];then
   echo "The number between 40 and 60."
fi
```

在这个脚本中，检查变量 **num** 的值是否在 **40** 到 **60** 之间。 **if** 后的两个表达式被逻辑与运算符**&&**分隔，判断顺序从左往右，只有前一个表达式为真时，才会判断下一个表达式，如果所有表达式为真，则返回 **TRUE**，否则返回 **FALSE**。

下面是一个逻辑或的使用例子。这个脚本可用来检查一个数是否在某个范围之外

```
#!/bin/bash
num=20
if [ $num -lt 40 ]||[ $num -gt 60 ];then
   echo "The number is outside 40 to 60."
fi
```


208

7.5.1 if语句

在这个脚本中，检查变量 **num** 的值是否在 **40** 到 **60** 之外。 **if** 后的两个表达式被逻辑或运算符**||**分隔，判断顺序仍然从左往右，但是只要一个表达式为真，就返回 **TRUE**，只有当表达式为假时，才会判断下一个表达式，如果所有表达式为假，则返回 **FALSE**。

逻辑非运算符！会对表达式的运算结果取反。如果表达式为 **FALSE**，则返回 **TRUE**；反之，如果表达式为 **TRUE**，则返回 **FALSE**。因此为第一个例子中的表达式加上逻辑非运算符，将与第二个例子完全等价。

```
#!/bin/bash
num=20
if !([ $num -ge 40 ]&&[ $num -le 60 ]);then
   echo "The number is out of range!"
fi
```

需要注意的是，因为！运算符仅对后面的第一个表达式有效，所以需要将整个表达式用圆括号括起来。

209

7.5.2 case语句

在分支控制中， **if** 可以满足很多情况下的需求。但是当分支条件非常多时， **if** 语句也随之变长。

case 语句为多选择语句，可以用 **case** 语句匹配一个值与一个模式，如果匹配成功，则执行相匹配的命令。 **case** 语句格式如下。

```
case value in
      [ expression) command...
      ;;
      ...
      ]
      esac
```

符号 **)** 前面的表达式称为待匹配的模式，取值将检测匹配的每一个模式。一旦模式匹配，则执行完匹配模式相应命令后不再继续其他模式。如果无一匹配模式，使用星号*捕获该值。

另外 **case** 语句还支持多语句组合，使用 **|** 符号连接多个模式，可以达到在模式间“或”的功能。

210

```
[root@localhost ~]# vi sh/case.sh
#!/bin/bash
#判断用户输入
read -p "Please choose yes/no: " -t 30 cho
#在屏幕上输出"请选择yes/no"，然后把用户选择赋予变量cho
case $cho in
#判断变量cho的值
  "yes")
  #如果是yes
     echo "Your choose is yes!"
     #则执行程序1
     ;;
  "no")
  #如果是no
     echo "Your choose is no!"
     #则执行程序2
     ;;
  *)
  #如果既不是yes,也不是no
  echo "Your choose is error!"
  #则执行此程序
  ;;
esac
```


211

7.6.1 while和until循环

在编写一段需要重复运行的程序时，要明确哪些工作需要重复，什么时候结束。使用 **while** 命令或 **until** 命令可以达到这样的目的。

while 命令的语法格式如下。

```
while expression; do
      command...
done
```

until 命令的语法格式如下。

```
until expression; do
      command...
done
```

与 **if** 命令一样，**while** 和 **until** 会判断表达式的返回值，从而控制循环条件，这里的表达式也经常使用 **test** 命令。而 **do** 和 **done** 之间的为重复执行的命令。**while** 命令的表达式返回值为真时，循环一直进行，直到返回值为假。而 **until** 命令正好相反，当表达式返回为假时，循环一直进行，直到返回值为真。

212

7.6.2 for循环

Shell 中还提供了另一种循环结构—**for** 循环。在处理数值序列的循环时，**for** 循环比 **while** 循环和 **until** 循环有效。 **for** 循环使用 **for** 命令，其语法格式如下。

```
for (( expression1; expression2; expression3));do
     command...
done
```

for循环执行过程如下：先执行**expression1**，再判断**expression2**，若返回**TRUE**，则进入循环执行**command**命令，完成每次循环后都需要执行**expression3**。因此 **for** 循环等同于如下结构的 **while** 循环。

```
expression1
while expression2;do
     command...
     expression3
 done
```

for 循环还提供了另外一种命令格式，如下所示。

```
for variable [in sequence]; do
    command...
done
```


213

7.6.3 跳出循环

在循环语句中，并不是只有循环条件能够控制循环。**bash** 还提供了两种在循环体内控制循环的命令： **break** 和 **continue**。其中 **break** 命令为跳出当前循环，**continue** 为开始下一次循环。下面的例子说明了这两个命令的用法。

编写脚本 **bkANDctn.sh**，在相同的循环中使用两种跳出命令。

```
#!/bin/bash
echo "Example of break:"
for (( i=1; i<6; i=i+1 ));do
     if [ $i -eq 3 ];then
          break
     fi
     echo $i
done
echo "Example of continue:"
for (( i=1; i<6; i=i+1 ));do
     if [ $i -eq 3 ];then
          continue
     fi
     echo $i
done
```


214

7.6.3 跳出循环

脚本 **bkANDctn.sh** 执行结果如下。

```
[user@localhost ~]$ ./bkANDctn.sh
Example of break:
1
2
Example of continue:
1
2
4
5
```

通过这个例子可以理解两种跳出命令的区别。在第一个循环中， 当变量 **i** 等于 **3** 时，执行 **break** 命令，然后程序就跳出了这个循环；而在下一个循环中，当变量 **i** 等于 **3** 时，跳过当前循环剩下的命令，继续下一次循环。

还需要注意的是，如果在嵌套的循环中使用跳出命令，只对 **break** 或 **continue** 所在的循环起作用，而不会影响外层循环。

215

7.7.1 获取位置参数

大部分 **Linux** 提供的命令都允许用户使用选项和参数来完成各种任务。例如，解压缩命令“**tar -xzf xxx.tar**”，其中“**-xzf**”和“**xxx.tar**”是以参数的形式传递给程序的。**Shell** 使用位置参数存储这些命令行中传入的内容，并可以使用**$**加上特殊的变量名获取位置参数。比如使用从 **1** 开始的整数依次存储每一个实参，如下面的例子所示。

编写脚本 **parameter1.sh**，输出 **1**～**9** 的位置参数。

```
#!/bin/bash
echo "\$1 = $1"
echo "\$2 = $2"
echo "\$3 = $3"
echo "\$4 = $4"
echo "\$5 = $5"
echo "\$6 = $6"
echo "\$7 = $7"
echo "\$8 = $8"
echo "\$9 = $9"
```


216

7.7.1 获取位置参数

按下面的方式执行脚本 **parameter1.sh**。

```
[user@localhost ~]$ ./parameter1.sh first second third fourth
$1 = first
$2 = second
$3 = third
$4 = fourth
$5 =
$6 =
$7 =
$8 =
$9 =
```

通过这个简单的例子介绍了如何获取位置参数。如果实参大于 **9** 个，可以用大括号将大于一位数的变量名括起来，如“**{10}**”。

$# 表示参数个数

217

7.7.2 位置参数使用案例

1. 处理选项

在学习 **while** 循环时，介绍了一个显示系统信息的菜单脚本。在该脚本中，使用循环输入指令的方式与用户交互。然而在 **Shell** 中很少用这种方式，更多的是使用选项指定程序的运作。下面编写一个全新的脚本 **show_info.sh**，使用位置参数来代替菜单。

```
#!/bin/bash
# check the argument number
if [ $# -ne 1 ];then
    echo "Invalid argument number, expected 1 received $#"
    exit 1
fi
case $1 in
        -n|-N) echo "Your user name is $USER."
        ;;
        -t|-T) echo -n "It's ";date +%H:%M
        ;;
        -d|-D) df -h
        ;;
```


218

7.7.2 位置参数使用案例

```
        -h|-H) echo "Available option:"
            echo "-n or -N:Show user name."
            echo "-t or -T:Show current time."
            echo "-d or -D:Show disk info."
            echo "-h or -H:Get help."
        ;;
        *) echo -e "Invalid option:'$1'\nTry 'show_info.sh -h' for help."
        ;;
esac
```

2. 处理字符串

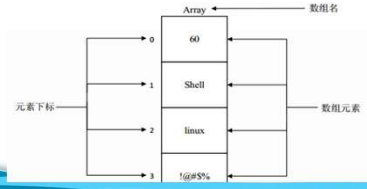
上面的例子展示了使用位置参数接收用户选项的方法。然而在很多程序中，不仅需要选项，还需要用户提供一些参数以完成特定的工作。例如，前面已经说过的 **tar –xzf xxx.tar** 命令便是提供了 **xxx.tar** 参数以供命令处理。

219

7.8.1 为什么使用数组

目前我们使用的变量都是由变量名和变量值组成的，并且一个变量只能包含一个值，这种数据结构称为标量变量。如果在编程中需要使用多个数据，使用标量变量的方法将会产生多个变量，这样会影响计算机处理的效率，并且使用多个变量名也会给编程造成极大的麻烦。为了解决这些问题，在许多计算机语言中使用数组来存放数据类型相同的数据。在 **Shell** 中，因为没有数据类型的约束，数组的使用将更加灵活。

数组是可以在内存中连续存储多个元素的结构，也可以理解为是一次存放多个值的变量。数组由数组名、数组元素和元素下标组成，下图是一个数组在内存中的存储情况。



220

7.8.2 数组的创建、赋值和删除

与 **bash** 中的其他变量一样，数组的创建和赋值可以同时进行，但是需要注明数组元素的下标。数组创建与赋值的命令格式如下。

array[subscript]=value

其中 **array** 为数组名，是数组的唯一标识；**subscript** 为从 **0** 开始的数组元素下标；**value** 为对应元素的值。下面是数组创建和赋值的例子。

```
[user@localhost ~]$ array[2]=10
[user@localhost ~]$ echo $array[2]
[2]
[user@localhost ~]$ echo ${array[2]}
10
```

在这个例子中，第一条命令将 **10** 赋值给数组 **array** 中下标为 **2** 的元素。在后面的两条命令中演示了访问数组元素的方法，在取数组元素的值时，需要用花括号指明数组元素的完整名称，以免 **Shell** 将元素名扩展成路径名。可以看出数组除需要注意元素下标外，使用方法和 **Shell** 变量非常相似。与其他编程语言不同的是，**Shell** 数组只会给赋值的元素分配内存，如这个例子中只为下标为 **2** 的元素分配了内存，而在其他编程语言中，下标为 **0** 和 **1** 的元素会被初始化为空值并占用内存。

221

7.8.2 数组的创建、赋值和删除

作为能够存储多个数值的数据结构，数组还支持一次赋值多个数据，命令格式如下。

array=(value1 value2...)

其中 **value1**、**value2** 等值依次赋予从下标为 **0** 开始的数组元素。例如：

```
[user@localhost ~]$ array=(one two three four)
[user@localhost ~]$ echo ${array[0]} ${array[1]} ${array[2]} ${array[3]}
one two three four
```

对于数组的删除操作，可分为删除数组和删除数组中的指定元素。删除操作需要使用 **unset** 命令，删除数组的命令格式如下。

unset array

删除数组中指定元素的命令格式如下。

unset array[subscript]

222

7.8.3 遍历访问数组元素

在前面的例子中使用了下标的方式对数组元素进行创建和赋值等操作。 在处理多个值时，这么做显然没有发挥数组作用。特别是在一些数组长度不确定的情况下，就需要遍历访问数组中的每一个元素。循环是一种非常适合遍历数组的方法，如下面的例子使用循环对数组进行赋值和访问。

编写脚本 **array1.sh**，使用 **for** 循环对数组进行赋值，例如：

```
#!/bin/bash
array1[0]=1
for (( i=1; i<10; i=i+1 ));do
    array1[$i]=$[${array1[$i-1]}*2]
done
```

该脚本用一个等比数列的前 **10** 项为数组 **array1** 的前 **10** 个元素赋值。

223

7.9.1 函数的定义与调用

函数可以看作是对特定代码的封装，用户定义函数后，可以在任何位置调用。**Shell** 中函数的定义有以下两种格式。

```
function name{
    command...
    [return]
}
```

或者是：

```
name(){
    command...
    [return]
}
```

这两种格式是等价的，其中 **name** 为函数名， **return** 是可选项，表示函数的返回值。下面的脚本 **func1.sh** 演示了函数的调用方法。

224

7.9.1 函数的定义与调用

```
#!/bin/bash
myfunc(){
    echo "Calling the function!"
}

echo "Before calling“
myfunc
echo "After calling“
```

脚本 **func1.sh** 执行结果如下。

```
[user@localhost ~]$ ./func1.sh
Before calling
Calling the function!
After calling
```

脚本执行时会跳过函数的定义部分，直接执行第一个 **echo** 命令，输出 **Before calling**。然后调用函数 **myfunc**，执行函数体中的命令，也就是第二个 **echo** 命令，输出 “**Callingthe function!”**。函数执行完后，继续执行函数调用后的代码，即第三个 **echo** 命令，输出 **After calling**。

225

7.9.2 在函数中使用位置参数

向函数传递参数与向脚本传递参数的情况非常相似， 使用的参数符号也与脚本中的位置参数相同。下面的例子演示了向函数传递参数的情况。

脚本 **func2.sh** 中定义了函数 **add**，实现两个整数的加法运算，代码如下。

```
#!/bin/bash
add(){
    if [ $# -ne 2 ];then
        echo "Invalid argument number,expected 2 received $#"
        return
    fi
    echo "$1+$2=$[$1+$2]"
}
add 2 2
```

脚本 **func2.sh** 执行结果如下。

```
[user@localhost ~]$ ./func2.sh
2+2=4
```


226

7.9.2 在函数中使用位置参数

在这个例子中，调用 **add** 函数时传递了两个参数，在 **add** 函数体中使用**$1** 和 **$2** 接收了这两个参数，并且还使用 **$#** 判断参数数量的合法性。这些都与脚本的位置参数如出一辙，所以 **Shell** 函数可以被视为位于脚本中的迷你脚本。需要注意的是，函数体内部的 **$1**、 **$#**等符号代表传递给函数的相关参数，不能与脚本的位置参数相混淆。在本例中，**add** 函数的**$1**、 **$2** 和 **$#** 都为 **2**，而脚本位置参数 **$1** 和 **$2** 为空， **$#**为 **0**。

227

7.9.3 使用函数返回值

函数中的关键字 **return** 用于指定函数的返回值，返回值作为函数执行的结果返回给函数调用命令。前面提到过，**Shell** 函数是迷你的脚本，所以函数返回值类似于程序的退出状态，可以使用变量 “**$?”**获取。例如，下面的例子在 **add** 函数中使用了返回值。

修改脚本 **func2.sh** 如下。

```
#!/bin/bash
add(){
    if [ $# -ne 2 ];then
        echo "Invalid argument number,expected 2 received $#"
        return
    fi
    return $[$1+$2]
}
add 2 2
echo "The result is $?"
```


228

7.9.3 使用函数返回值

脚本 **func2.sh** 执行结果如下。

```
[user@localhost ~]$ ./func2.sh
The result is 4
```

在上面的例子中， **add** 函数返回了参数的运算结果，在函数调用完成后，使用变量 **$?** 接收到了返回值。尽管函数返回值与程序退出状态如此相似，但它们之间也存在区别。**return** 关键字只能用于函数中，表示将返回值提供给函数的调用命令；**exit** 关键字可以用于程序的任意位置，表示退出当前程序并将退出状态返回到父进程。因此，如果把脚本 **func2.sh** 返回运算结果的 **return** 改为 **exit**，就只能在运行脚本的 **Shell** 中取得返回值。代码如下。

```
[user@localhost ~]$ ./func2.sh
[user@localhost ~]$ echo $?
4
```


229

7.9.4 将函数保存到文件

利用函数可以将程序功能模块化，如前面将加法功能模块化到了 **add** 函数中。模块化最大的优势之一就是方便代码重用，现在只能在当前程序中使用 **add** 函数，如何让其他程序也使用 **add** 函数呢？可以将函数保存到文件中，程序在需要时可以读取文件中的函数，达到代码重用的目的。

读取文件可以使用 **source** 命令， **source** 命令是 **bash** 的内部命令。功能是使 **Shell** 读入指定的 **Shell** 程序文件并依次执行文件中的所有语句。

source 命令格式如下：

```
source filename
```

删除 **func2.sh** 中 **add** 函数以外的内容，并编写脚本 **func3.sh** 调用 **func2.sh** 中的 **add**函数，代码如下。

```
#!/bin/bash
source ./func2.sh
echo "func3.sh calling add function"
add 14 17
echo "The result is $?"
```


230

7.9.4 将函数保存到文件

脚本 **func3.sh** 执行结果如下所示：

```
[user@localhost ~]$ ./func3.sh
func3.sh calling add function
The result is 31
```

脚本 **func3.sh** 使用 **source** 命令读取脚本 **func2.sh**，并调用其中的 **add** 函数。甚至可以将函数写入到 **.bashrc** 文件中，如下所示。这样函数在系统启动时将一直有效。

```
# .bashrc
# Source global definitions
if [ -f /etc/bashrc ]; then
        . /etc/bashrc
fi
# Uncomment the following line if you don't like systemctl's auto-paging feature:
# export SYSTEMD_PAGER=
# User specific aliases and functions
```


231

7.9.4 将函数保存到文件

```
add(){
        if [ $# -ne 2 ];then
                echo "Invalid argument number,expected 2 received $#"
                return
        fi
        return $[$1+$2]
}
```

从以上例子可以看出，将函数保存到文件中，可以使其他程序重用一些常用代码，这在大型项目中是非常常见的。

232

Linux操作系统基础教程

任课教师 刘昊霖

233

目 录 CONTENTS

234

第8章　进程与设备管理

《Linux操作系统基础教程》

235

第8章　进程与设备管理

2021/12/2

236

第8章　进程与设备管理

2021/12/2

237

8.1.1　进程的概念

Linux 是一个多用户多任务的操作系统，可以同时执行多个用户的多个程序。为了保证程序能够并发执行，需要对程序的执行过程进行动态控制。进程就是用来描述这一控制过程的，组织安排不同程序等待 **CPU** 的调度。进程是一个具有一定独立功能的程序或程序段在一组数据集合上的一次动态执行过程，同时也是程序能够并发执行的基础机制。各种资源的分配和管理都是以进程为单位。

进程的基本组成结构为：进程控制块（**Processing Control Block**，**PCB**）、程序段和数据结构集。**PCB** 是进程动态特征的反映，主要包括进程的描述信息、控制信息、进程使用资源情况等。程序段是该进程需要完成功能的程序代码。数据结构集是进程执行时需要访问的工作区和数据对象，即执行进程时需要的系统资源。总地来说，一个进程包括要运行的程序，程序所需的数据以及跟踪管理程序状态所需的各种信息。

进程在创建时，会被内核赋予一个 **ID** 作为进程唯一标识号（也称为 **PID**），它是一个非负整数。进程 **ID** 可以重用，当进程终止后，**Linux**一般通过延迟重用算法，使得赋予新进程的 **ID** 不同于最近终止进程的 **ID**。在 **Linux** 系统中可以同时运行多个进程，为了方便跟踪管理系统中的所有进程，内核中存在一个进程表，以 **PID** 为索引，每个进程在进程表中都会存在一条记录，描述及管理进程所需的信息。

2021/12/2

238

8.1.1　进程的概念

在 **Linux** 系统中根据进程的特点和属性，将进程分为 **3** 类：交互进程、批处理进程和守护进程。

（**1**）交互进程：由 **Shell** 启动的进程，既可以在前台运行，也可以在后台运行，且必须由用户给出某些参数或者信息，进程才能继续执行。

（**2**）批处理进程：与终端没有联系，是一个进程序列，负责按照顺序启动其他进程。

（**3**）守护进程：执行系统特定功能或者执行系统相关任务的进程，并在后台运行。守护进程是一个特殊的进程，不是内核的组成部分。大部分守护进程是在系统启动时启动，直到系统关闭时才停止运行。

2021/12/2

239

8.1.2　进程上下文

在 **Linux** 中，用户程序装入系统形成一个进程的实质是系统为用户程序提供一个完整的运行环境。进程的运行环境是由它的程序代码、程序运行所需的数据结构和硬件环境组成的。进程的运行环境主要包括以下几种。

（**1**）进程空间中的代码和数据、各种数据结构、进程堆栈和共享内存区等。

（**2**）环境变量：提供进程运行所需的环境信息。

（**3**）系统数据：进程空间中的对进程进行管理和控制所需的信息，包括进程任务结构体以及内核堆栈等。

（**4**）进程访问设备或者文件时的权限。

（**5**）各种硬件寄存器。

（**6**）地址转换信息。

从以上组成情况可以看出，进程的运行环境是动态变化的，尤其是硬件寄存器的值以及进程控制信息是随着进程的运行而不断变化的。在 **Linux** 中把系统提供给进程的处于动态变化的运行环境总和称为进程上下文。

2021/12/2

240

8.1.2 进程上下文

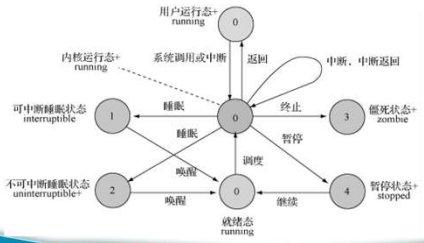
系统中的每一个进程都有自己的上下文。一个正在使用处理器运行的进程称为当前进程。当前进程因时间片用完或者因等待某个事件而阻塞时，进程调度需要把处理器的使用权从当前进程交给另一个进程，这个过程叫作进程切换。此时，被调用进程成为当前进程。在进程切换时，系统要把当前进程的上下文保存在指定的内存区域，然后把下一个使用处理器运行的进程的上下文设置成当前进程的上下文。 当一个进程经过调度再次使用 **CPU** 运行时，系统要恢复该进程保存的上下文。

所以，进程的切换也就是上下文切换。当内核需要切换到另一个进程时，它需要保存当前进程的所有状态，即保存当前进程的上下文，以便再次执行该进程时，能够得到切换时的状态执行下去。

241

8.2.1 进程状态及状态转换

Linux 进程总体来说有 **5** 种状态：运行态、就绪态、睡眠状态、暂停状态、僵死状态。进程之间相互独立，一个进程不能改变另一个进程的状态，但是进程自己在通过事件触发被调度的过程中，可以在各状态之间切换，进程状态之间的转换如下图所示。



242

8.2.1 进程状态及状态转换

（1）运行态。它是在 **run_queue** 队列里的状态，占有 **CPU** 处理进程任务，一个进程只能出现在一个 **CPU** 的可执行队列里。同一时刻允许有多个进程处于运行状态，但运行状态的进程总数应小于或等于处理器的个数。 运行状态分为用户运行态和内核运行态两种，在内核态下运行的进程不能被其他进程抢占。

（2）就绪态。该状态的进程已经拥有除 **CPU** 以外的所有请求资源，只等待被核心程序调度。只要被分配到 **CPU** 就可执行，在队列中按照进程优先级进行排队。

（3）睡眠状态。处于该状态的进程需要被某一事件触发才可继续执行，分为可中断的睡眠状态和不可中断的睡眠状态。处于可中断的睡眠状态的进程，是在等待资源被释放，一旦得到资源，进程就会被唤醒进入就绪态。由于 **CPU** 数量有限，而进程数量众多，所以很多请求无法及时得到响应，因此大部分进程都处于可中断的睡眠状态。处于不可中断睡眠状态的进程，只能通过 **wake_up( )**函数唤醒。

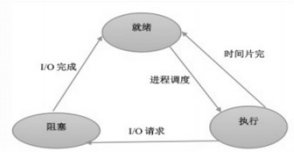
（4）暂停状态。也被称为跟踪状态。是指进程在从内核返回用户时，被核心程序抢先调度了另一个进程，该进程就处于暂停状态。处于暂停状态的进程只有等待下次调度，才能返回用户态。当进程收到信号 **SIGSTOP** 时会进入暂停状态，发送 **SIGCONT** 信号，进程可转换到运行状态。

（5）僵死状态。处于该状态的进程已经终止运行，等待父进程询问其状态，收集它的进程控制块所占资源。

243

8.2.1 进程状态及状态转换

一般情况下，一个进程至少有 **3** 种基本状态：运行态、就绪态、封锁态（阻塞态）。其结构关系如图所示。



（1）运行态：已经分配到 **CPU**，正在处理器上执行。

（2）就绪态：已经具备运行条件，但所需 **CPU** 资源被其他进程占用，需等待分配**CPU**。

（3）阻塞态：尚不具备运行条件，需要等待某种事件的发生，即使 **CPU** 空闲，也无法使用。

244

8.2.1 进程状态及状态转换

在 **Linux** 系统中，存在一些 **Shell** 命令可以查看系统中正在运行的进程的信息。最常用的是 **ps** 命令和 **top** 命令。二者之间的区别为 **top** 命令可以动态地查看进程信息。

1. 查看进程状态：**ps**

ps 命令可以查看系统中正在运行的进程信息以及进程的状态。

ps 的命令格式为： **ps [option]**

（**1**）简单显示系统中当前用户正在运行的进程信息

只需要在终端输入 **ps** 命令本身，就可以在终端显示当前用户正在运行的进程信息，但是此时的输出结果中不显示进程状态。

```
[user@localhost Desktop]$ ps
PID TTY TIME CMD
3927 pts/0 00:00:00 bash
4864 pts/0 00:00:00 ps
```

输出结果中含有 **4** 个数据列，分别为 **PID**（进程**ID**）、 **TTY**（控制终端的名称）、 **TIME**（消耗的**CPU**时间总和）、 **CMD**（正在被执行的命令名称）。

245

8.2.1 进程状态及状态转换

（**2**）显示进程状态

为了使输出结果显示进程状态，可以在 **ps** 命令后添加 **j** 参数，结果如下。

```
[user@localhost Desktop]$ ps j
PPID PID PGID SID TTY TPGID STAT UID TIME COMMAND
3920 3927 3927 3927 pts/0 5400 Ss 1001 0:00 /bin/bash
3927 5400 5400 3927 pts/0 5400 R+ 1001 0:00 ps j
```

输出结果中包含很多数据列信息，最后一列 **COMMAND** 与上述的 **CMD** 含义相同，但是列名不同。

（**3**）按用户名和启动时间的顺序显示进程

当有多个用户同时登录时，通过 **u** 选项参数控制输出结果按照用户名和启动时间的顺序显示进程。

```
[user@localhost Desktop]$ ps u
USER PID %CPU %MEM VSZ RSS TTY STAT START TIME COMMAND
user 3927 0.0 0.3 116676 3324 pts/0 Ss 19:59 0:00 /bin/bash
user 6065 0.0 0.1 139492 1628 pts/0 R+ 21:19 0:00 ps u
```


246

8.2.1 进程状态及状态转换

其中，**USER** 表示该进程的所有者；**%CPU** 表示 **CPU** 使用百分比；**%MEM** 表示内存使用百分比；**VSZ** 表示虚拟耗用内存大小；**RSS** 表示实际使用的内存大小，以 **KB** 为单位；**START** 表示进程开启的时间，如果数值超过 **24** 个小时， 将使用日期来表示。**TIME** 表示消耗的 **CPU** 时间总和；**COMMAND** 表示正在被执行的命令的名称。

2．动态查看进程信息：**top**（监控进程）

top 命令将按照进程的活动顺序，持续更新显示当前系统进程的信息，即屏幕显示的进程信息是不断变化的。在终端输入 **top** 命令。

[user@localhost Desktop]$ top

输出结果中包含上下两个部分，顶部表示系统状态的总体信息，下半部分显示当前系统中正在运行的进程的详细信息。

247

8.2.2 进程控制

每个进程都有自己的生命周期，主要包括进程的创建、进程的执行和进程的终止。在 **Linux** 系统中通过 **fork**、 **exec**、 **exit**、 **wait** 等函数控制进程从创建到终止的过程。**fork** 函数创建进程；**exec** 函数控制进程执行特定任务；**exit** 函数终止进程；**wait** 函数控制进程同步执行。

1．进程创建

当需要在系统中创建一个新的进程时，可以通过系统调用 **fork( )**来完成，引用 **fork**系统调用的进程是父进程，由 **fork** 创建的进程是子进程。

fork 系统调用的格式为：

pid = fork()

其中， **pid** 表示执行 **fork** 系统调用后的返回值， **pid** 的值有 **3** 种情况。

（**1**） **pid=0**，表示此进程是子进程。

（**2**） **pid>0**，表示此进程是父进程。

（**3**） **pid<0**，表示进程创建失败。子进程从 **fork** 调用后的语句开始与父进程并发执行。

2．控制进程执行特定任务

引用 **fork** 系统调用创建的子进程，与父进程执行相同的代码，如果需要子进程执行不同的任务，则需要在子进程中使用 **exec** 系统调用，让子进程执行新的代码段。

248

8.2.2 进程控制

exec 系统调用的一般格式为：

execve(pathname, argv, envp)

pathname 表示要执行文件的路径名；**argv** 是字符指针数组，表示可执行函数的参数； **envp** 也是字符指针数组，表示执行程序的环境。

exec 有 **6** 种调用方式： **execl**、 **execv**、 **execle**、 **execve**、 **execlp**、 **execvp**，这 **6** 种调用方式函数名末尾字母是不相同的（如 **execl**、 **execve**），共存在“**l**”“**v**”“**e**”“**p**”四种情况。**4** 个字母的含义如下。

（**1**）**l**：要求把参数指针数组的每个指针都作为独立的参数传递，并以空指针 **NULL**结束。

（**2**）**v**：同 **C** 语言使用 **argc**、 **argv** 传递参数一样，通过 **argv** 指针数组中的内容传递 **exec** 系统调用需要的参数。

（**3**）**e**：从 **envp** 指针数组传递环境参数，表示要执行的程序需要新的运行环境，否则用现有环境变量复制新程序环境。

（**4**）**p**：在搜索执行程序时，在环境变量 **PATH** 指定的目录中搜索指定的文件，否则在当前目录中进行搜索。

249

8.2.2 进程控制

3．进程终止

通过执行系统调用 **exit** 来终止正在运行的进程，进程终止后进入僵死状态，释放所占用的大部分资源。但是该进程在系统中仍然存在，等待父进程将其回收，父进程通过 **exit** 系统调用完成回收工作。

exit 系统调用的格式为：

exit(status)

其中，**status** 是一个整数，作为进程结束时的状态传递给父进程。

4．进程同步

为了让子进程的终止点和父进程同步，可以在父进程中引用 **wait** 系统调用，等待子进程完成其指定任务。

wait 系统调用的格式为：

pid=wait(stat-addr)

其中，**pid** 表示要终止的子进程的 **ID** 号，参数 **stat-addr** 表示子进程结束时返回的状态信息存放的地址。

wait 系统调用最常用的方式是：**wait(0)**。使父进程暂停执行，处于等待状态，一旦子进程执行完毕，父进程就会重新进入执行状态，从而保证子进程与父进程的执行同步。

250

8.3.1 管道及命名管道

管道（**pipe**）可用于具有亲缘关系进程间的通信，命名管道（**named pipe** 或 **FIFO**）克服了管道没有名字的限制，允许无亲缘关系进程间的通信。

1．管道（无名管道）

管道是一种最基本的进程通信机制，使用内存实现进程通信，只适用于父子进程或父进程安排的各个子进程之间的通信。通过 **pipe** 系统调用在内核中创建管道，函数原型为：

int pipe（int fildes[2]）

调用 **pipe** 函数，会得到两个文件描述符（一个读端，一个写端），然后通过 **fildes** 参数将这两个文件描述符传给用户程序： **fildes[0]**为读端， **filedes[1]**为写端，然后用户程序通过系统调用 **read(filedes[0])**和 **write(filedes[1])**进行管道的读和写。

251

8.3.1 管道及命名管道

2．命名管道

命名管道借助于磁盘在任意进程间通信。命名管道与管道的不同之处在于它提供了一个路径名与之关联，以 **FIFO** 的文件形式存储在系统中，因此即使进程与创建 **FIFO** 的进程不存在亲缘关系，只要可以访问路径，就可以建立通信。遵循先进先出的原则，即第一个被写入的数据首先从管道中读出。通过 **mknod** 系统调用建立命令管道，函数原型为：

int mknod(const char *path, mode_t mod, dev_t dev)

（**1**）参数 **path** 表示创建的命名管道的全路径名。

（**2**）参数 **mod** 表示创建的命名管道的模式，指明其存取权限。

（**3**）参数 **dev** 表示设备值，该值取决于文件创建的种类，只在创建设备文件时，才会用到。

252

8.3.2 信号

信号（**signal**）用于通知接收进程有某种事件发生。除了用于进程间通信外，进程还可以发送信号给进程本身。进程通过 **kill** 系统调用发送信号，通过 **signal** 系统调用接收或捕获信号，进程通过 **signal** 接收信号后，一般有 **3** 种处理方法：指定处理函数，由函数来处理；忽略信号，不做任何处理；对该信号保留系统的默认值。

1. kill 函数

kill 函数用于发送信号，进程通过调用 **kill** 函数可以向自身或其他进程发送信号，函数原型为：

int kill(pid_t pid, int sig)

其中，参数 **pid** 表示进程标识符；参数 **sig** 表示信号标识符。

kill 函数的意义是把信号 **sig** 发送给进程号为 **pid** 的进程。函数调用成功返回值为 **0**，调用失败返回值为**-1**。调用失败一般有三种原因：指定的信号无效；发送权限不够，即目标进程由另一个用户拥有；目标进程不存在。

253

8.3.2 信号

2. signal 函数

signal 函数用于处理信号进程。函数原型为：

void (*signal(int sig, void (*handler)(int))) (int)

这个函数看起来很复杂，实际上该函数只带有 **sig** 和 **handler** 两个参数。

（**1**）参数 **sig** 表示准备接收或捕获的信号。

（**2**）参数 **handler** 是一个类型为 **void (*)(int)**的函数指针，它指向的函数会对 **sig** 信号做出处理操作。参数 **handler** 也可以是两个特殊值：**SIG_IGN**：忽略信号；**SIG_DFL**：回复信号的默认行为。

254

8.3.3 消息队列

消息队列（**message**）是消息的链接表，包括 **Posix** 消息队列和 **system V** 消息队列。消息队列的通信方式是一个进程向另一个进程发送一个数据块，且消息的发送随时可进行，不需要接收方准备好。有足够权限的进程可以向队列中添加消息，被赋予读权限的进程则可以读走队列中的消息，消息队列的读取不一定是先进先出。消息队列克服了信号承载信息量少，管道只能承载无格式字节流以及缓冲区大小受限等缺点。 **Linux** 系统提供了一些系统调用对消息队列进行管理，常用函数有：**msgget**、 **msgsnd**、 **msgrcv**、**msgctl**。

1. msgget 函数

msgget 函数用来创建和访问一个消息队列。函数原型为：

int msgget(key_t key, int msgflg)

其中，参数 **key** 表示消息队列的标识符；参数 **msgflg** 是一个权限标志，表示消息队列的访问权限，它与文件的访问权限一样。

msgget 函数调用成功，返回一个以 **key** 命名的消息队列的标识符（非零整数），失败时返回**-1**。

255

8.3.3 消息队列

2. msgsnd 函数

msgsnd 函数用来把消息添加到消息队列中。函数原型为：

int msgsnd(int msgid, const void *msgp, size_t msgsz, int msgflg)

（**1**）参数 **msgid** 是由 **msgget** 函数返回的消息队列标识符。

（**2**）参数 **msgp** 是一个指向准备发送消息的指针，但是消息的数据结构有一定的要求，指针 **msgp** 指向的消息结构一定要是以一个长整型成员变量开始的结构体。

（**3**）参数 **msgsz** 是 **msgp** 指向的消息的长度，不包括长整型消息类型成员变量的长度。

（**4**）参数 **msgflg** 用于控制当前消息队列满或队列消息到达系统范围的限制时将要发生的事情。

msgsnd 函数调用成功，消息数据的一个副本将被放到消息队列中，并返回 **0**，失败时返回**-1**。

3. msgrcv 函数

msgrcv 函数用来从一个消息队列获取消息。函数原型为：

ssize_t msgrcv(int msgid, void *msgp, size_t msgsz, long msgtyp, int msgflg)

（**1**）参数 **msgid**、 **msgp**、 **msgsz** 的含义同 **msgsnd** 函数的一样。

256

8.3.3 消息队列

（**2**） **msgtyp** 确定接收消息的优先级。如果 **msgtyp** 为 **0**，就获取队列中的第一个消息。如果 **msgtyp>0**，将获取具有相同消息类型的第一个信息。如果 **msgtyp<0**，就获取类型等于或小于 **msgtyp** 绝对值的第一个消息。

（**3**） **msgflg** 用于控制当队列中没有相应类型的消息可以接收时将发生的事情。**msgrcv** 函数调用成功，该函数返回放到接收缓存区中的字节数，消息被复制到由 **msgp** 指向的用户分配的缓存区中，然后删除消息队列中的对应消息。失败时返回**-1**

4. msgctl 函数

msgctl 函数用来控制消息队列，函数原型为：

int msgctl(int msgid, int cmd, struct msgid_ds *buf)

（**1**）参数 **msgid** 是由 **msgget** 函数返回的消息队列标识符。

（**2**）参数 **cmd** 是将要采取的动作，它可以取 **3** 个值。

① **IPC_STAT**：把 **msgid_ds** 结构中的数据设置为消息队列的当前关联值，即用消息队列的当前关联值覆盖 **msgid_ds** 的值。

② **IPC_SET**：如果进程有足够的权限，就把消息列队的当前关联值设置为 **msgid_ds**结构中给出的值。

③ **IPC_RMID**：删除消息队列。

（**3**）**buf**是指向**msgid_ds**结构的指针，它指向消息队列模式和访问权限的结构。

257

8.3.4 信号量

信号量（**semaphore**）的主要作用是协调进程访问共享资源。信号量是一种外部资源标识，本身不具有数据交换的功能，通过控制其他的通信资源实现进程间的通信。当请求一个使用信号量表示的资源时，进程需要先读取信号量的值来判断资源是否可用。信号量的值大于 **0** 时，表示资源可以请求；信号量的值为 **0** 时，表示资源已经被占用，此时进程会进入睡眠状态等待资源可用。**Linux** 提供了一些函数对信号量进行操作，常用函数为： **semget**、 **semop**、 **semctl**。

1. semget 函数

semget 函数用于创建一个新信号量或取得一个已有信号量，函数原型为：

int semget(key_t key, int nsems, int semflg)

semget 函数调用成功时，返回一个相应信号标识符（非零），失败时返回**-1**。

2. semop 函数

semop 函数用于改变信号量的值，函数原型为：

int semop(int semid, struct sembuf *sops, unsigned nsops)

其中，参数 **semid** 是由 **semget** 返回的信号量标识符。

258

8.3.4 信号量

3. semctl 函数

semctl 函数用于直接控制信号量信息，函数原型为：

int semctl(int semid, int semnum, int cmd, ...)

（**1**）参数 **semid** 是由 **semget** 返回的信号量标识符。

（**2**）参数 **semnum** 指定需要的信号量数目。

（**3**）参数 **cmd** 表示要进行的操作，它的常用取值有两个。

① **SETVAL**：用来把信号量初始化为一个已知的值。

② **IPC_RMID**：用于删除一个已经无需继续使用的信号量标识符。

（**4**）如果有第四个参数，它通常是一个 **union semum** 结构，定义如下。

```
union semun
{
    int val;
    struct semid_ds *buf;
    unsigned short *arry;
};
```


259

8.3.5 共享内存

共享内存是 **Linux** 系统中最底层的通信机制，也是最快速的通信机制。共享内存通过两个或多个进程共享同一块内存区域来实现进程间的通信。通常是由一个进程创建一块共享内存区域，然后多个进程可以对其进行访问。发送进程将要传出的数据存放到共享内存中，接收进程（一个或多个进程）则直接从共享内存中读取数据，避免了数据的复制过程，因此这种通信方式是最高效的进程间通信方式。

采用共享内存的进程通信有三个特点。

（**1**）当进程通信时，需要交互的数据或信息不发生存储移动。

（**2**）当需要交互时，通信进程双方通过一个共享内存完成信息交互。

（**3**）对共享内存，可以用虚拟映射方式将其作为交互过程中的一部分存储体使用

当进程采用共享内存与另一进程通信时，常用函数为： **shmget**、 **shmat**、**shmdt** 和 **shmctl** 函数。

1. shmget 函数

shmget 函数用来创建或打开共享内存，函数原型为：

int shmget(key_t key, size_t size, int shmflg)

（**1**）参数 **key**，是一个非 **0** 整数，表示共享内存标识符。若用户提供的 **key** 值不存在，则创建新的共享内存，根据参数 **key** 为共享内存段命名。若 **key** 值存在，则打开现有共享内存。

260

8.3.5 共享内存

（**2**）参数 **size** 表示以字节为单位指定需要共享的内存容量。

（**3**）参数 **shmflg** 是权限标志，共享内存的权限标志与文件的读写权限一样。

shmget 函数调用成功时，返回一个与 **key** 相关的共享内存标识符（非负整数），用于后续的共享内存函数。调用失败返回**-1**。

2. shmat 函数

shmat 函数用于连接共享内存。第一次创建完共享内存时，它还不能被任何进程访问，**shmat** 函数就是用来启动对该共享内存的访问，并把共享内存连接到当前进程的地址空间，父进程已连接的共享内存可被**fork**创建的子进程继承。函数原型为：

void *shmat(int shmid, const void *shmaddr, int shmflg)

（**1**）参数 **shmid** 表示由 **shmget** 函数返回的共享内存标识。

（**2**）参数 **shmaddr** 指定共享内存连接到当前进程中的地址位置，一般为空，表示让系统来选择共享内存的地址。

（**3**）参数 **shmflg** 是一组标志位，通常为 **0**。

shmat 函数调用成功时，返回一个指向共享内存第一字节的指针，即共享内存的首地址。如果调用失败，则返回**-1**。

3. shmdt 函数

shmdt 函数用于将共享内存从当前进程中分离。将共享内存分离并不是删除它，只是使该共享内存对当前进程不再可用，拆除共享内存与本进程地址空间的连接。

261

8.3.5 共享内存

函数原型为：

int shmdt(const void *shmaddr)

参数 **shmaddr** 是 **shmat** 函数返回的地址指针，**shmat** 函数调用成功时返回 **0**，失败时返回**-1**。

4. shmctl 函数

shmctl 函数用来控制共享内存，函数原型为：

int shmctl(int shmid, int cmd, struct shmid_ds *buf)

（**1**）参数 **shmid** 表示 **shmget** 函数返回的共享内存标识符。

（**2**）参数 **cmd** 表示要在 **shmid** 指定的共享内存中采取的操作，它可以取 **5** 个值。

① **IPC_STAT**：把 **shmid_ds** 结构中的数据设置为共享内存的当前关联值，即用共享内存的当前关联值覆盖 **shmid_ds** 的值。

② **IPC_SET**：如果进程有足够的权限， 就把共享内存的当前关联值设置为 **shmid_ds** 结构中给出的值。

③ **IPC_RMID**：删除共享内存段。

④ **SHM_LOCK**：锁定共享内存进程，只有超级用户才有该权限。

⑤ **SHM_UNLOCK**：解锁共享内存，只有超级用户才有该权限。

（**3**）参数 **buf** 是一个结构指针，它指向共享内存模式和访问权限的结构。

262

8.4.1 设备管理的基本概念

设备是指计算机系统中除 **CPU**、内存和系统控制台以外的所有设备。设备管理模块负责控制系统的 **I/O** 部件。在 **Linux** 中，设备管理模块完成的主要功能如下。

（**1**）为用户提供简捷、统一的设备使用接口方式，包括建立用户的命令接口和程序设计接口函数等。

（**2**）完成设备的分配、占用与释放管理。当用户请求使用某一外设时，设备管理模块负责分配设备通道或控制器；当设备使用完成后，负责回收和释放资源。

（**3**）帮助用户访问和控制设备。当有多个进程请求使用外设时，设备管理模块要进行并发访问控制和共享设备分配管理，还要处理出现的错误情况。

（**4**）完成对 **I/O** 缓冲区的管理和控制。通过对 **I/O** 缓冲区的管理，提高 **CPU** 的利用率及 **I/O** 的访问效率。

263

8.4.2 Linux设备类型

在 **Linux** 系统中，为了便于对设备进行管理和控制，按照信息的组织特征，设备可分为三类：字符设备、块设备、网络设备。

（**1**）字符设备（如键盘、打印机），以字符为单位输入输出数据，可直接对设备进行读写，但是一般只允许顺序访问。

（**2**）块设备（如磁盘、光盘），以一定大小的数据块为单位输入输出数据，需要借助缓冲区技术对数据进行处理，允许随机访问。

（**3**）网络设备，通过网络传输数据的设备，一般是指硬件设备，如与通信网络连接的网络适配器（网卡）。

264

8.4.3 设备管理结构

Linux 系统的 **I/O** 管理模块与文件系统紧密相关，设备管理结构如图所示。

按照 **I/O** 设备性能的不同，将设备管理分为两类。

（**1**）无缓冲区 **I/O**：无缓存 **I/O** 访问方式采用将用户访问进程与系统 **I/O** 管理模块直接进行数据交换的形式完成 **I/O** 操作。

（**2**）有缓冲区 **I/O**：使用有缓存 **I/O** 进行数据交换时，要经过系统的缓冲区 **cache**或字符队列管理机构完成对 **I/O** 的访问。

一个设备一般都有 **3** 个标志：设备类型、主设备号、次设备号。通过这 **3** 个标志识别设备。主设备号与驱动程序对应，即使用的驱动程序不同，设备的主设备号也就不同。次设备号用来区分使用同一驱动程序的各个具体设备。

265

8.4.4 设备管理技术

Linux 对 **I/O** 传输的常用控制方式有 **4** 种：查询等待方式、中断控制方式、**DMA** 控制方式、通道控制方式。

1．查询等待方式

查询等待方式又称为轮询方式， 因为对于不支持中断方式的机器只能采用这种方式来控制 **I/O** 过程，所以 **Linux** 中也配备了查询等待方式。例如，并行接口的驱动程序中默认的控制方式就是查询等待方式。

2．中断控制方式

中断处理中的 **I/O** 操作由系统控制程序发起，当 **I/O** 设备完成相应命令时（如读或写操作），外设控制部件向处理器发出中断请求，向系统控制程序通告本次 **I/O** 操作的执行结果，并等待下一次处理器发出的执行命令。在中断控制方式中，数据和信息的每次（或每批）读写操作都需要系统监管，所以采用这种控制方式时，处理器对 **I/O** 设备的管理任务比较繁重。

266

8.4.4 设备管理技术

3．**DMA** 控制方式

直接存储器访问（**direct memory access**， **DMA**）控制方式是一种适合于块设备的批量数据传递方式。在 **DMA** 方式中，控制程序完成 **DMA** 控制字寄存器的设置（如传送内存的起始地址、传送字节数等）。当 **I/O** 操作开始后，处理器可以转去执行其他的处理，直到这一批数据传送结束， **DMA** 控制器向处理器发出中断请求，并告知系统本次的数据传送结束后，处理器再转向对它进行下一步操作的控制。

4．通道控制方式

通道控制方式是一种更加智能化的外设控制方式， 现代通道控制部件中都有自己专用的通道处理器和缓冲存储器，这样在执行系统提出的由通道指令组成的通道程序时，可以用比较高的效率进行较复杂的 **I/O** 控制。采用通道控制时，通常执行一次通道程序可以完成多批的通道数据处理，因此通道方式的 **I/O** 访问效率更高。通道控制方式主要用于连接智能性较高的外部设备，如网卡上的信道访问控制。

267