华为认证 openEuler 系列教程

HCIA-openEuler

openEuler系统工程师

学员用书

版本:1.0

华为技术有限公司

商标声明


和其他华为商标均为华为技术有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

华为技术有限公司

地址： 深圳市龙岗区坂田华为总部办公楼 邮编：518129

网址： http://e.huawei.com

华为认证体系介绍

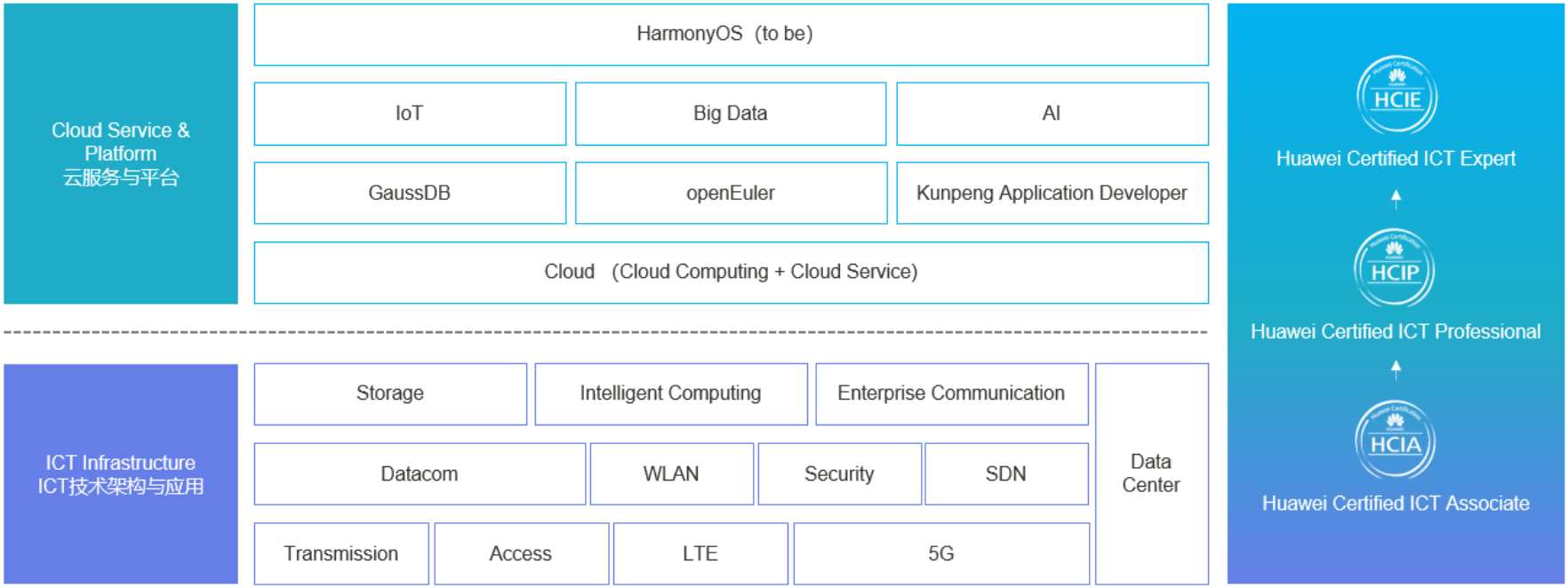
华为认证是华为公司基于“平台+生态”战略，围绕“云-管-端”协同的新ICT技术架构，打造的ICT技术架构认证、平台与服务认证、行业ICT认证三类认证，是业界唯一覆盖ICT（Information and Communications Technology 信息通信技术）全技术领域的认证体系。

根据ICT从业者的学习和进阶需求，华为认证分为工程师级别、高级工程师级别和专家级别三个认证等级。华为认证覆盖ICT全领域，符合ICT融合的技术趋势，致力于提供领先的人才培养体系和认证标准，培养数字化时代新型ICT人才，构建良性ICT人才生态。

HCIA-openEuler（Huawei Certified ICT Associate-openEuler ，华为认证openEuler系统工程师）主要面向华为公司办事处、代表处一线工程师，以及其他希望学习Linux技术人士。HCIA-openEuler认证在内容上涵盖openEuler操作系统概述、openEuler基础操作入门、openEuler文件处理及文本编辑、openEuler用户及权限管理、openEuler逻辑卷管理、openEuler软件安装、openEuler综合实践等内容。

华为认证协助您打开行业之窗，开启改变之门，屹立在Linux世界的潮头浪尖！

Huawei Certification



FOREWORD
序

Linux 作为使用最广泛的操作系统发展了近 30 年，已经成为 IT（Information Technology）产业最基础的平台。openEuler 是一个基于 Linux Kernel（内核）的开源社区，也是一个开放创新的平台，不但承载着对鲲鹏等多种芯片架构的支持，而且承载着对操作系统、体系架构未来的探索任务。openEuler 社区最终会成为一个引领技术创新的开源生态系统。开源是一种产业生态的建设模式，在世界范围内，越来越多的公司利用开源促进产业链生态的建设，甚至引导产业的发展方向，形成了从开源社区到基于开源的企业级产品与服务的生态链。

开源是一种协作创新模式，软件开发的速度大大加快，产业标准的形成时间大大缩短。同时，开放协作的开源环境更容易激发创新的思维和创造的灵感，对于开源社区不断涌现的创新，我们已经喜闻乐见。

开源也是一种文化交流的方式，通过开源，可以集合全世界的智慧，在世界的不同角落，共同协作完成大型软件系统的开发和演进。这在很大程度上也加深了世界人民之间的沟通和了解。我始终相信，沟通与交流是全世界构建美好未来的钥匙。

openEuler 是一个开源、免费的 Linux 发行版平台，将通过开放的社区形式与全球的开发者共同构建一个开放、多元和架构包容的软件生态体系。同时，openEuler 也是一个创新的平台，鼓励任何人在该平台上提出新想法、开拓新思路、实践新方案。

PREFACE
前言

操作系统是最基础的核心软件，也被誉为计算机的“灵魂”。无论对于计算机相关专业的学生或研究人员，还是对于计算机应用开发人员，对操作系统原理的学习和理解都至关重要。

本书以 openEuler 操作系统为例，围绕 HCIA-openEuler 认证涉及的知识点，详细介绍了 openEuler 操作系统的相关概念及操作。HCIA-openEuler V1.0 是华为针对 openEuler 开源操作系统推出的第一个职业认证，也是国内唯一一个针对社区版开源操作系统推出的职业认证。

openEuler 操作系统的职业认证目前定位于 openEuler 操作系统运维工程师的能力认证。分为初级、中级和高级认证。HCIA-openEuler 定位于培养工程师对于 openEuler 基础操作能力的培养，通过 HCIA-openEuler 认证，将能熟练掌握 openEuler 操作系统的基础操作技巧，能够胜任 Linux 基础运维工程师岗位。

本书也将围绕 HCIA-openEuler，分九个章节进行介绍。本书着重于理论与实践的结合，在学习本书时，希望读者们不要吝惜双手，配合 HCIA-openEuler 的 PC 版实验手册，学习并掌握 openEuler 操作系统的基础操作。

千里之行，始于足下。让我们一起开始学习吧！

目录

1 openEuler 操作系统入门 7
1.1 Linux 操作系统介绍 7
1.2 安装 openEuler 操作系统 12
1.3 开始使用 openEuler 操作系统 16
1.4 思考题 18
2 命令行基础操作 19
2.1 Linux 命令行基础知识 19
2.2 Linux 基础命令 20
2.3 思考题 35
3 文本编辑及文本处理 36
3.1 Linux 常用的文本编辑器 36
3.2 文本编辑及文本处理 38
3.3 思考题 49
4 用户和权限管理 50
4.1 用户和用户组管理 50
4.2 文件权限管理 55
4.3 其它权限管理 64
4.3.1 权限临时提升 64
4.4 思考题 66
5 软件安全和服务管理 67
5.1 软件包管理概述 67
5.2 RPM 软件包概述 67
5.3 DNS 软件管理概述 68
5.4 源代码安装概述 76
5.5 服务管理 77
5.6 思考题 81
6 管理文件系统及存储 82
6.1 Linux 存储基础 82
6.2 逻辑卷层管理 86
6.3 思考题 94
7 系统管理 95
7.1 任务管理 95

7.2 网络管理 97
7.3 进程管理 107
7.4 思考题 113
8 使用 shell 脚本 114
8.1 shell 基础介绍 114
8.2 shell 编程基础 117
8.3 思考题 136
9 Samba 文件共享服务器管理 137
9.1 Samba 概述 137
9.2 Samba 文件共享服务器搭建实例 139
9.3 思考题 141
参考文献/博文 142

1 openEuler 操作系统入门

1.1 Linux 操作系统介绍

1.1.1 什么是操作系统

操作系统（Operating System，OS），即操作计算机的系统，是控制和管理整个计算机系统的硬件和软件资源，并合理地组织调度计算机的工作和资源的分配，以提供给用户和其他软件方便的接口和环境的程序集合。操作系统位于硬件和应用程序之间，如图 1-1 所示，它作为计算机硬件之上的第一层软件，为上层的应用程序提供了良好的应用环境，并且让底层的硬件资源高效地协作，完成特定的计算任务。由于它具有良好的交互性，用户能够通过操作界面以非常简单的方式对计算机进行操作。

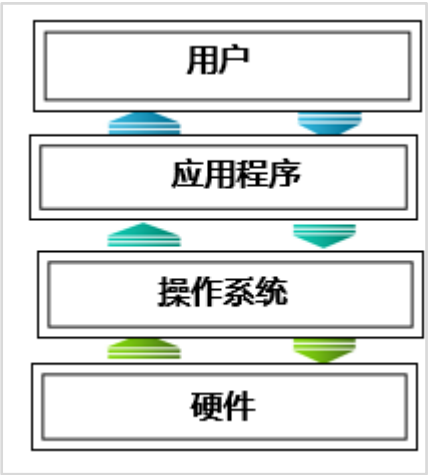


图1-1 操作系统在计算机中的位置

1.1.1.2 操作系统几个核心功能

进程管理

现代计算机系统采用多道程序技术，允许多个程序并发执行，共享系统资源。多道程序系统出现后，为了描述并发执行的程序的动态特性并控制其活动状态，操作系统抽象出了“进程”这一概念。使用进程作为描述程序执行过程且能用来共享资源的基本单位。操作系统为进程分配合理的硬件资源，控制进程状态的转换，完成计算机并发任务的执行。

如何让不同进程合理共享硬件资源呢？首先操作系统需要保持对硬件资源的管理。操作系统通过系统调用向进程提供服务接口，限制进程直接进行硬件资源操作。如果需要执行受限操作，进程只能调用

这些系统调用接口，向操作系统传达服务请求，并将 CPU 控制权移交给操作系统。操作系统接收到请求后，再调用相应的处理程序完成进程所请求的服务。

如何实现多个进程的并发执行呢？各进程需要以时分复用的方式共享 CPU。这意味着操作系统应该支持进程切换：在一个进程占用 CPU 一段时间后，操作系统应该停止它的运行并选择下一个进程来占用 CPU。为了避免恶意进程一直占用 CPU，操作系统利用时钟中断，每隔一个时钟中断周期就中断当前进程的执行，来进行进程切换。

内存管理

系统中的程序和代码在被 CPU 调度执行之前需要先加载到内存中，所以当多个进程并发执行时，所有的并发进程都需要被加载进内存中，所以内存成为了影响操作系统性能的关键因素。操作系统的内存管理主要解决并发进程的内存共享问题，通过虚拟内存、分页机制、利用外存对物理内存进行扩充等技术提高内存利用率和内存寻址效率。

文件系统管理

虽然内存为系统提供了快速的访问能力，但因为内存的容量较为有限，一旦断电，保存在其中的数据就会丢失。所以计算机通常采用磁盘等外存来持久化地存储数据。为了简化外存的使用，操作系统将磁盘等外存抽象成文件（file）和目录（directory），并使用文件系统（file system）管理它们。操作系统可以选择多种物理文件系统，如 Ext4、FAT32、NTFS 等等，用户或应用程序通过文件系统可以方便的完成 I/O 操作，输入输出数据到磁盘当中，实现持久化的数据存储，同时实现对文件存储空间的管理、目录的管理和文件读写的管理和保护。

硬件驱动管理

操作系统作为用户操作底层硬件的接口，管理着计算机各类 I/O 设备，操作系统通过可加载模块功能，将驱动程序编辑成模块以识别底层硬件，以便上层应用程序使用 I/O 设备。所以操作系统会提供开发接口给硬件厂商制作他们的驱动程序，而操作系统获取到硬件资源后完成设备分配、设备控制和 I/O 缓冲区管理的任务。

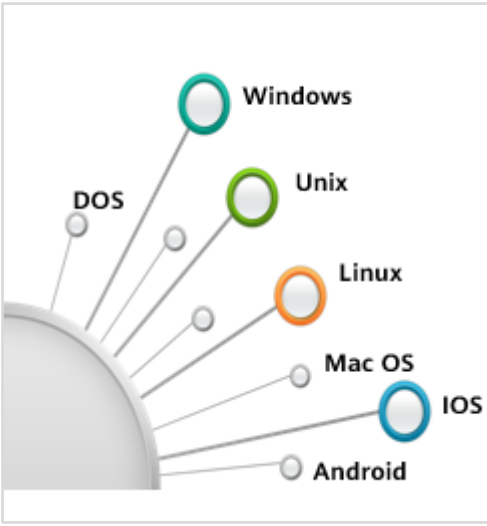
用户交互界面

操作系统为用户提供了可交互性的环境，让用户更加容易地使用计算机。一般来说，用户与操作系统交互的接口分为命令接口和 API 接口两种。

- 命令接口
- 用户通过输入设备或在作业中发出一系列指令，传达给计算机，使计算机按照指令来执行任务。常见的命令接口有两种，一种是命令行界面（Command Line Interface，CLI），用户界面字符化，使用键盘作为输入工具，输入命令、选项、参数执行程序，追求高效，MS-DOS 系统提供的就是字符交互方式。另一种是图形用户界面（Graphical User Interface，GUI），用户界面的所有元素图形化，主要使用鼠标作为输入工具，使用按钮、菜单、对话框等进行交互，追求易用。Windows 系统采用的主要是图形交互方式。
- API 接口
- 主要由系统调用（system call）组成。每一个系统调用都对应着一个在内核中实现、能完成特定功能的子程序。通过这种接口，应用程序可以访问系统中的资源和取得操作系统内核提供的服务。

1.1.2 常见的操作系统

之前章节中提到了操作系统的不同的交互式界面，以 DOS 系统为代表的 CLI 界面和以 Windows 系统为代表的 GUI 界面。由于 Windows 良好的交互性，Windows 系列的操作系统已成为个人计算机中使用度最广的桌面操作系统。除 Windows 以外，还有其他多种主流操作系统，如图 1-2 所示。



操作系统	应用场景
Linux	企业服务器，注重稳定性和性能，类Unix系统，开源免费
Unix	企业服务器，注重稳定性和性能
Windows	PC，注重易用性
Mac OS	PC，注重易用性和个人体验
Android	移动端，注重易用性和个人体验
IOS	移动端，注重易用性和个人体验
DOS	PC，很少单独安装，一般和Windows一起安装

图1-2 常见的操作系统

Linux 操作系统广泛地使用在企业服务中，注重稳定性和性能，受到广大开发者的喜爱与追捧。Linux 是一套免费使用和自由传播的类 Unix 的操作系统，这个系统是由全世界各地的成千上万的程序员设计和实现的。用户不用支付任何费用就可以获得它和它的源代码，并且可以根据自己的需要对它进行必要的修改，无偿地对它使用，无约束地继续传播。

此外计算机不断地向小型化发展，现在，计算机以智能手机、智能手表等移动设备的形式出现在人们的生活当中，其中 iOS 和 Android 是当前最为主流的面向移动设备的操作系统。iOS 是 Apple 公司于 2007 年发布的一款操作系统，属于类 UNIX 的商业操作系统。该操作系统目前没有开源。2008 年 9 月，Google 公司以 Apache 开源许可证的授权方式，发布了 Android 的源代码。Android 基于 Linux 内核，是专门为触屏移动设备设计的操作系统。

1.1.3 Linux 的起源与发展

Unix 发展历程

讲到 Linux 的起源，不得不从 Unix 开始说起。上个世纪六十年代，那个计算机还没有很普及，仅在军事或者学术研究上进行使用，一般人很难接触到，而且当时的计算机系统都是批处理的，就是把一批任务一次性提交给计算机，然后就等待结果。并且中途不能和计算机交互。往往准备作业都需要花费很长时间，导致了计算机资源的浪费。当时的主机仅可以支持少量的终端机接入，进行输入输出的作业，为了强化主机的功能并且能让更多的用户进行使用，在 1965 年前后，贝尔实验室（Bell）、麻省理工学院（MIT）以及通用电气公司（GE）联合起来准备研发一个分时多任务处理系统，简单来说就是实现多人同时使用计算机的梦想，并把计算机取名为 MULTICS（MULTiplexed Information and Computing

System，多路信息计算系统），但是由于项目太复杂，加上其他原因导致了项目进展缓慢，1969 年贝尔实验室觉得这个项目可能不会成功，于是就退出不玩了。

Bell 退出 MULTICS 计划之后，曾参与 MULTICS 项目的贝尔实验室研究人员 Ken Thompson 从 MULTICS 中获得灵感，用汇编语言写了一个小型的操作系统，来运行原本在 MULTICS 系统上的一个叫做太空大战（Space Travel）的游戏，大概就是一个很简单的打飞机的游戏，当完成之后，Thompson 怀着激动的心情把身边同事叫过来，让他们来玩他的游戏，大家玩过之后纷纷表示对他的游戏不感兴趣，但是对他的系统很感兴趣。由于这个系统是 MULTICS 的删减版，于是把它称为：“UNiplexed Information and Computing Service”，缩写为“UNICS”。后来大家取其谐音，就称其为“UNIX”了。这个时候已经是 1970 年了，于是就将 1970 年定为 Unix 元年，因此计算机上的时间就是从这一年开始计算的。1971 年，Thompson 和 Ritchie 共同发明了 C 语言，之后他们用 C 语言重写了 UNIX，并于 1974 年正式对外发布。

Unix 是一种多任务、多用户的操作系统，于 1969 年由美国 AT&T 贝尔实验室开发。Unix 起初是免费的，其安全高效、可移植的特点使其在服务器领域得到了广泛的应用。1982 年 AT&T 分解后变为商业应用，很多大型数据中心的高端应用都用 Unix 系统。各大型硬件公司，配合自己的计算机系统，也纷纷开发出许多不同的 Unix 版本，主要包括早期的 System V、UNIX 4.x BSD（Berkeley Software Distribution）版本、FreeBSD、OpenBSD、SUN 公司的 Solaris、IBM 的 AIX、主要用于教学的 MINIX 和现在苹果公司专用的 MAC OS X 等。

GNU 与开源

为了打破 UNIX 封闭生态的限制，Richard M.Stallman 在 1983 年发起一项名为 GNU 的国际性的源代码开放计划，并创立了自由软件基金（Free Software Foundation，FSF）。自由软件基金会规定了四个自由：第一，处于任何目的的运行程序的自由；第二，学习和修改源代码的自由；第三，重新分发程序的自由；第四，创建衍生程序的自由。由于 GNU 强调“Free”一词，大部分人对它的理解是“免费”的，实际上这是不确切的，这里的“Free”指的是自由的软件，可以自由获取并修改，虽然确实是可以免费获得源码，但对于软件的咨询、售后服务、软件升级等增值服务是需要进行付费的，这就是自由软件的商业行为。GNU 的成立对推动 UNIX 操作系统以及后面的 Linux 操作系统发展起到了非常积极的作用。

GNU 项目的目标是建立完全自由（Free）、开放源码（Open Source）的操作系统。当时并没有这样的操作系统，Stallman 就先开发了适用于 Unix 上运行的小程序，如 Emacs、gcc（GNU C Compiler）和 Bash shell 等等。

到了 1985 年，为了避免 GNU 所开发的自由软件被其他人所利用而成为专利软件，Stallman 发布了通用公共许可证（General Public License，GPL）。GPL 协议采取两种措施来保护程序员的权利：（1）给软件以版权保护；（2）给程序员提供许可证。它给程序员复制、发布和修改这些软件的法律许可。在复制和发布方面，GPL 协议规定“只要你在每一副本上明显和恰当地出版版权声明和不承担担保声明，保持此许可证的声明和没有担保的声明完整无损，并和程序一起给每个其他的程序接受者一份许可证的副本，你就可以用任何媒体复制和发布你收到的原始的程序的源代码。你可以为转让副本的实际行动收取一定费用。你也有权选择提供担保以换取一定的费用。但是只要在一个软件中使用（“使用”

指 类库 引用，修改后的 代码 或者 衍生代码）GPL 协议的产品，则该软件产品必须也采用 GPL 协议，既必须也是开源和免费。”

目前，除了 GPL 开源协议，常见的开源协议还有木兰协议、LGPL 协议、BSD 协议等。

木兰协议是我国首个开源协议，这一开源协议共有五个主要方面，涉及授予版权许可、授予专利许可、无商标许可、分发限制和免责申明与责任限制。在版权许可方面，木兰协议允许“每个‘贡献者’根据‘本许可证’授予您永久性的、全球性的、免费的、非独占的、不可撤销的版权许可，您可以复制、使用、修改、分发其‘贡献’，不论修改与否。”

LGPL 是一个为主要为类库使用设计的开源协议。和 GPL 要求任何使用、修改、衍生自 GPL 类库的的软件必须采用 GPL 协议不同。LGPL 允许商业软件通过类库引用（link）方式使用 LGPL 类库而不需要开源商业软件的代码。这使得采用 LGPL 协议的开源代码可以被商业软件 作为类库引用并发布和销售。但是如果修改 LGPL 协议的代码或者衍生，则所有修改的代码，涉及修改部分的额外代码和衍生的代码都必须采用 LGPL 协议。

BSD 开源协议是一个给于使用者很大自由的协议。可以自由的使用，修改源代码，也可以将修改后的代码作为开源或者专有软件再发布。当你发布使用了 BSD 协议的代码，或者以 BSD 协议代码为基础做二次开发自己的产品时，需要满足三个条件：

- 如果再发布的产品中包含源代码，则在源代码中必须带有原来代码中的 BSD 协议。
- 如果再发布的只是二进制类库/软件，则需要在类库/软件的文档和版权声明中包含原来码中的 BSD 协议。
- 不可以用开源代码的作者/机构名字和原来产品的名字做市场推广。BSD 代码鼓励代码共享，但需要尊重代码作者的著作权。BSD 由于允许使用者修改和重新发布代码，也允许使用或在 BSD 代码上开发商业软件发布和销售，因此是对商业集成很友好的协议。

正因为自由软件的产生和源代码的开放，促进了 IT 技术的迅速发展。

Linux 的诞生

1991 年，芬兰赫尔辛基大学的学生 Linus Torvalds 在 MINIX 操作系统的基础上开发了一个新的操作系统内核。由于 GNU 计划提供的 bash 和 gcc 等自由软件，使得 Linus 顺利地开发了这个新的操作系统内核，为其取名为 Linux 并将其开源，同时呼吁广大开发者一起完善 Linux 操作系统。之后，全球各地的程序员们与 Torvalds 一起加入到了开发 Linux 的行列，为 Linux 添加了许多新特性。1994 年 3 月，在开发者的共同努力下，Linux 1.0 版本正式发布。

Linux 虽然起初并不是 GNU 计划的一部分，但它的历史与 GNU 密不可分。由于共同坚持的开源精神，Linux 与 GNU 计划走到了一起。目前，绝大多数基于 Linux 内核的操作系统使用了部分 GNU 软件。因此，严格地说，这些系统应该被称为 GNU/Linux。

如今，Linux 已经有很多个衍生版本。Linux 发行版是指打包了 Linux 内核和一些系统软件以及实用程序的套件。当前 Linux 发行版众多，这些发行版的主要不同之处在于：所支持的硬件设备以及软件包配置。较为主流的 Linux 发行版如 Redhat、openSUSE、Ubuntu、deepin 等。

Linux 的版本

Linux 的内核版本可以访问 https://www.kernel.org 进行查看和下载。Linux 内核版本号由 3 个数字组成：第一个数字代表目前发布的内核主版本；第二个数字是偶数的话表示稳定版本，奇数表示开发中的版本；第三个数字代表错误修补的次数。如后面章节中要介绍的 openEuler 操作系统，openEuler20.03 LTS 内核版本为 4.19.90，这是一个开发中的内核版本，主版本号为 4，修补次数为 90 次，相比于内核的稳定版会加入很多新的功能。

Linux 的发行版本分为商业发行版和社区发行版，商业发行版以 Redbat 为代表，由商业公司维护，提供收费的服务，如升级补丁等。社区发行版由社区组织维护，一般免费，如 CentOS、Debian、openEuler 等。

1.2 安装 openEuler 操作系统

1.2.1 openEuler 操作系统介绍

openEuler 是一款开源、免费的操作系统，由 openEuler 社区运作。当前 openEuler 内核源于 Linux，支持鲲鹏及其它多种处理器，能够充分释放计算芯片的潜能，是由全球开源贡献者构建的高效、稳定、安全的开源操作系统，适用于数据库、大数据、云计算、人工智能等应用场景。

openEuler 的前身是运行在华为公司通用服务器上的操作系统 EulerOS。EulerOS 是一款基于 Linux 内核（目前是基于 Linux 4.19 版本的内核）的开源操作系统，支持 X86 和 ARM 等多种处理器架构，伴随着华为公司鲲鹏芯片的研发，EulerOS 理所当然地成为与鲲鹏芯片配套的软件基础设施。2019 年底，EulerOS 被正式推送至开源社区，更名为 openEuler。openEuler 也是一个创新的平台，鼓励任何人在该平台上提出新想法、开拓新思路、实践新方案。所有个人开发者、企业和商业组织都可以使用 openEuler 社区版本，也可以基于 openEuler 社区版本发布自己二次开发的操作系统版本。

openEuler 通常有两种版本，一种是创新版本，支撑 Linux 爱好者技术创新，内容较新，如 openEuler 20.09，通常半年发布一个新的版本。另一种是 LTS 版本，是 openEuler 操作系统发行版的稳定版本，如 openEuler LTS 20.03，通常两年发布一个新的版本。本课程所涉及的操作均以 openEuler 20.03 LTS 版本为准。

1.2.2 openEuler 操作系统安装说明

openEuler 操作系统安装流程如下图 1-3 所示。

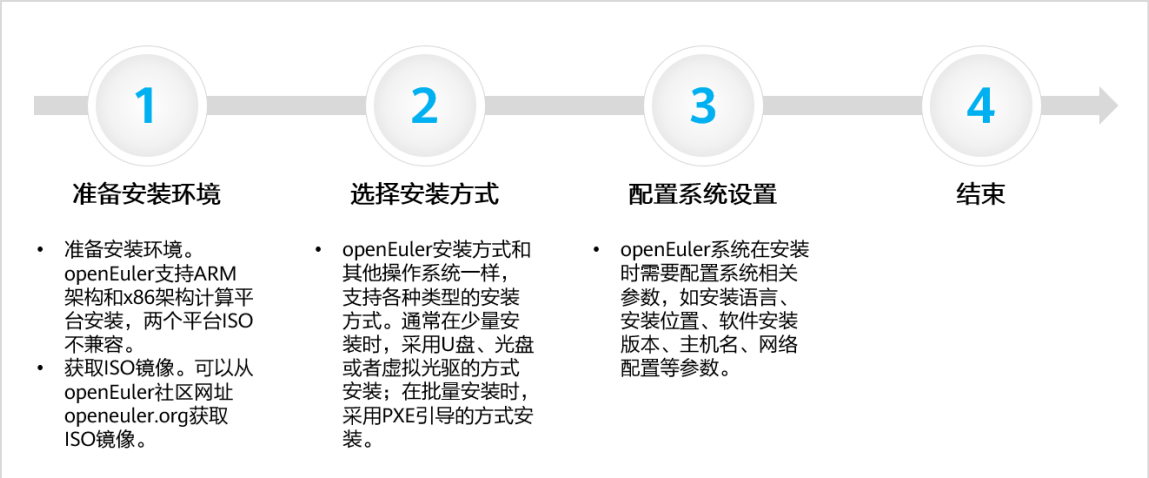


图1-3 openEuler 操作系统安装流程

准备安装环境

openEuler 支持 ARM 架构和 x86 计算平台的安装，由于 x86 和 ARM 指令集的区别，两个平台的 ISO 是不兼容的，在官网 https://openeuler.org/获取 ISO 镜像的时候一定要注意区分所下载的文件路径，选择适用的 ISO 镜像进行安装。

本课程提供以下两种环境供大家学习参考：

- PC 环境：基于 VirtualBox 虚拟化环境安装 openEuler 操作系统，一般 PC 都是基于 x86 的系统，所以请选择安装 x86 版本（文件路径为 x86_64）的 openEuler 操作系统。
- 服务器环境：基于 TaiShan 200 服务器上 FusionCompute 的虚拟化环境安装 openEuler 操作系统，由于 TaiShan 200 服务器所匹配的是 Kunpeng920 芯片，鲲鹏处理器是华为基于 ARMv8 架构开发的通用处理器，所以请选择 ARM 版本（文件路径为 aarch64）的 openEuler 操作系统。

以上两种环境准备都是在虚拟化环境中进行， 都需要先安装虚拟化软件，如 VirtualBox 或 FusionCompute，并创建裸虚拟机，并为此虚拟机分配合适的 cpu、内存或者硬盘空间，最后在此基础上进行下一步的安装。openEuler 所需的最小虚拟化空间要求如下表 1-1 所示。

表1-1 最小虚拟化空间要求

部件名称	最小虚拟化空间要求	说明
架构	• AArch64 • x86_64	• 支持Arm的64位架构。 • 支持Intel的x86 64位架构。
CPU	2个CPU	《HCIA-openEuler实验手册-PC版》实验环境的PC建议为CPU至少为4核
内存	不小于4GB	《HCIA-openEuler实验手册-PC版》实验环境的PC建议为内存至少为16GB
硬盘	不小于32GB	《HCIA-openEuler实验手册-PC版》实验环

		境的PC建议为硬盘空闲空间大于100GB

本课程中所涉及的两种版本的内容是一样的，本书采用 x86_64 版本 openEuler 为例进行介绍。具体环境准备步骤请参考《HCIA-openEuler 实验手册-PC 版》1.2 配置虚拟化环境。

选择安装方式

openEuler 安装方式和其他操作系统一样，支持各种类型的安装方式。通常在少量安装时，采用 U 盘、光盘或者虚拟光驱的方式安装；在批量安装时，采用 PXE 引导的方式安装。本课程以虚拟光驱引导方式安装系统为例。挂载 ISO 后重启虚拟机，即可进入安装引导界面，进入默认选项“Test this media & install openEuler 20.03 LTS”的图形化安装界面，如图 1-4 所示。

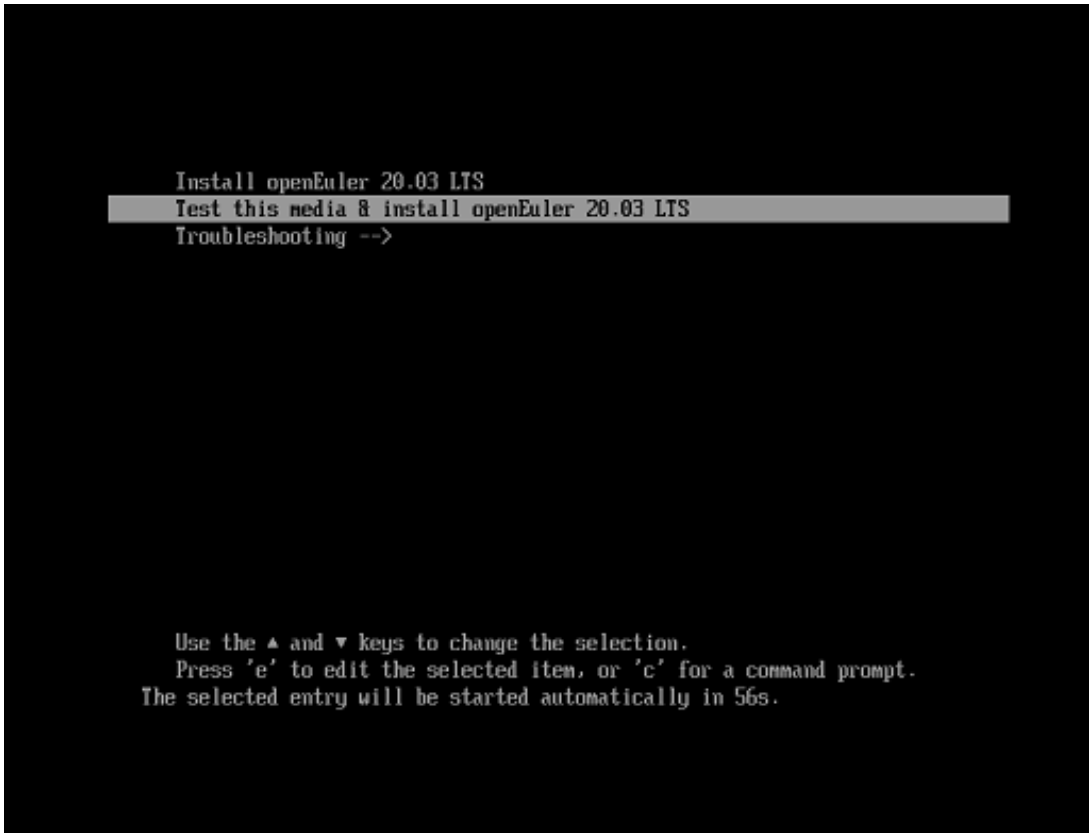


图1-4 openEuler 安装引导界面

系统安装参数配置

openEuler 系统在安装时需要配置系统相关参数，如安装语言、安装位置、软件安装版本、主机名、网络配置等参数，配置项有告警符号的，表示用户必须完成该选项配置后，告警符号消失，才能进行下一步操作。如图 1-5 所示。

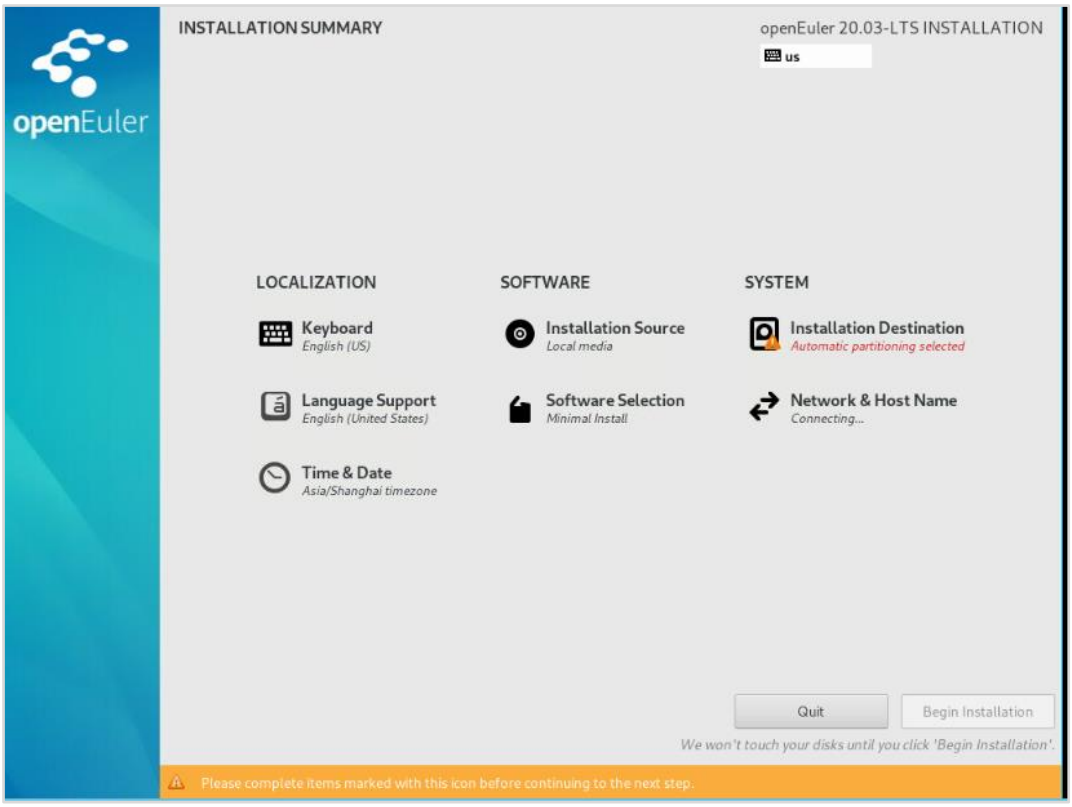


图1-5 openEuler 安装设置界面

以下是对于安装位置设置、选择安装软件和创建用户的重点介绍：

- **安装位置设置**。安装位置设置用于设置系统安装位置以及系统安装分区设置。可以自动也可以手动设置分区。openEuler 系统启动建议设置如下分区：
  - “swap”交换分区，在内存空间不足时，用于置换内存中的脏数据，小内存情况下建议设置为内存大小的两倍，内存较大时，可以据情况减少分配。
  - “/boot”目录保存用于引导操作系统的文件。当计算机打开电源后，首先是 BIOS 开机自检，按照 BIOS 中设置的启动设备（通常是硬盘）来启动。操作系统接管硬件以后，首先读入 /boot 目录下的内核文件。
  - “/boot/efi”是 UEFI 固件要启动的引导器和应用程序的目录。当安装 openEuler for ARM 的版本的时候，启动方式为 UEFI，需要创建/boot/efi 分区才可以启动。
  - “/”是根分区，Linux 中一切从根开始。文件目录的根源，一切文件都存放在根目录下。
- **选择安装软件**。openEuler 20.03 LTS 安装时支持 3 种软件场景选择：
  - 在最小安装的环境下，并非安装源中所有的包都会安装。如果用户需要使用的包未安装，可将安装源挂载到本地制作 repo 源，通过 DNF 工具单独安装。
  - 选择“服务器”的基本环境，则系统已集成了易于管理的服务器组件。
  - 选择“虚拟化主机”时会默认安装虚拟化组件 qemu、libvirt、edk2，且可在附件选项处选择是否安装 ovs 等组件。
- **设置 root 密码及创建用户**。openEuler 在安装过程中需要设置 root 用户密码，root 用户为系统超级管理员，具有最高权限，通常 Linux 管理员是不能使用该用户对系统进行管理。可以根据需求选择性创建普通用户，如创建一个名为 openEuler 的普通用户，并为其设置用户名。openEuler 系统在安装时对用户设置的密码都需要高复杂度。

结束

完成系统安装的配置后，重启系统，使用 root 用户名及密码即可登录到 openEuler 的环境中，如图 1-6 所示。

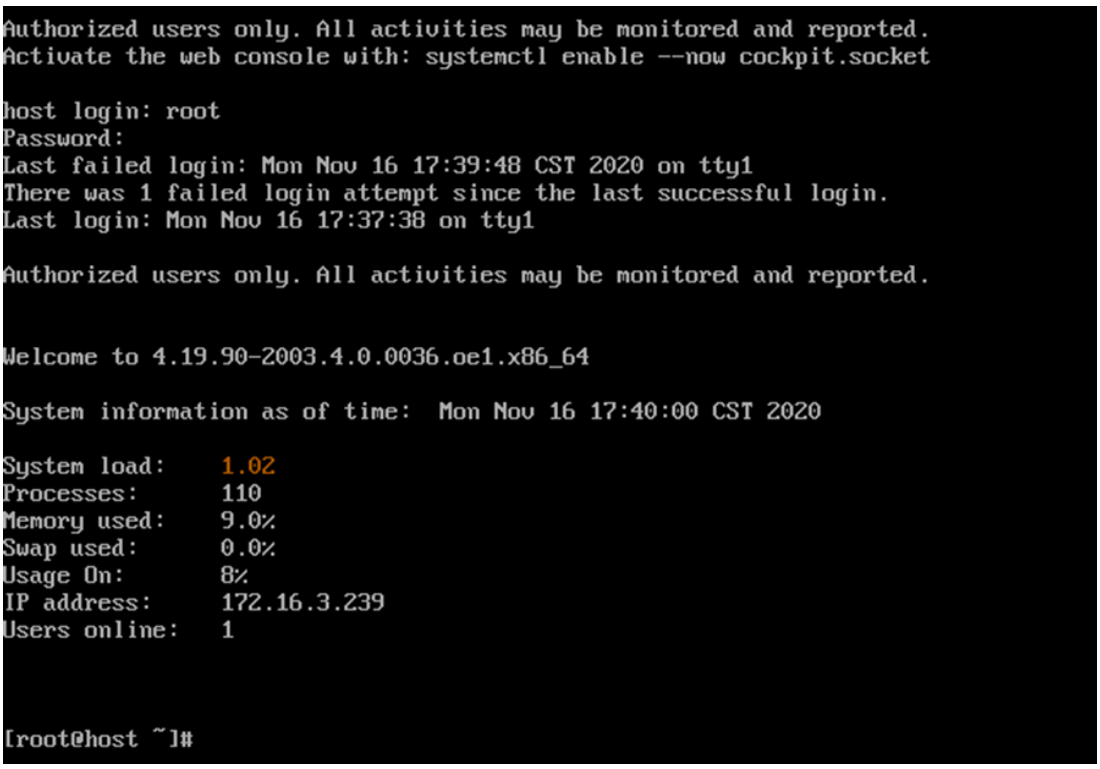


图1-6 openEuler 系统登录界面

具体安装步骤请参考《HCIA-openEuler 实验手册-PC 版》1.3 安装 openEuler 操作系统。

1.3 开始使用 openEuler 操作系统

之前的操作系统功能中已介绍了操作系统为用户提供了可交互性的环境，Linux 有 GUI 的图形用户界面，跟人们经常使用的 Windows 的图形用户界面较为类似，可以通过鼠标进行一些可视化的操作。还有 CLI 的命令行界面，主要使用键盘作为输入工具。Linux 命令行是由 shell 程序提供。Shell 实际是一个用 C 语言编写的程序，它是用户使用 Linux 的桥梁，用户通过 shell 来控制 Linux 系统，Linux 系统通过 shell 展现系统信息。Bash 是 shell 的一种，与 Windows 中的命令行解释程序 cmd.exe 较为类似。

openEuler20.03 LTS 版本暂无图形界面，用户的默认登录 shell 是 bash，所以本课程将通过在 bash shell 中操作学习 openEuler。

Linux 主要有两种登录方式，本地登录和远程登录。本地登录类似类似于打开自己电脑或者服务器直接接显示器的方式，一个典型的 Linux 系统将运行六个虚拟控制台，可以通过 Ctrl+Alt+F[1-6]在六个虚拟控制台之间进行切换。默认情况下 openEuler 支持远程登录，可以通过 putty、xshell 等工具远程登录到 openEuler。

开启 openEuler 的主机后，需要输入用户名和密码登录到操作系统的环境中，root 是 Linux 系统中超级管理员，具有最高权限。在安装过程中已设置了 root 用户的密码，可以通过 root 用户名和密码进行登录。如图 1-6 所示。

通过命令提示符可以清楚地了解当前是 root 用户还是普通用户。在 Unix 或者 Linux 系统中，root 用户命令提示符最后一般是#，普通用户一般是$。若在安装过程中已创建了普通用户，也可用普通用户和用户密码，登录系统。如图 1-7 所示。系统提示符默认表示的是：[当前登录用户@主机名 当前所在位置]$ 。图中当前用户为普通用户 openEuler，主机名称为 host。“~”表示当前位置是在/root 目录下。

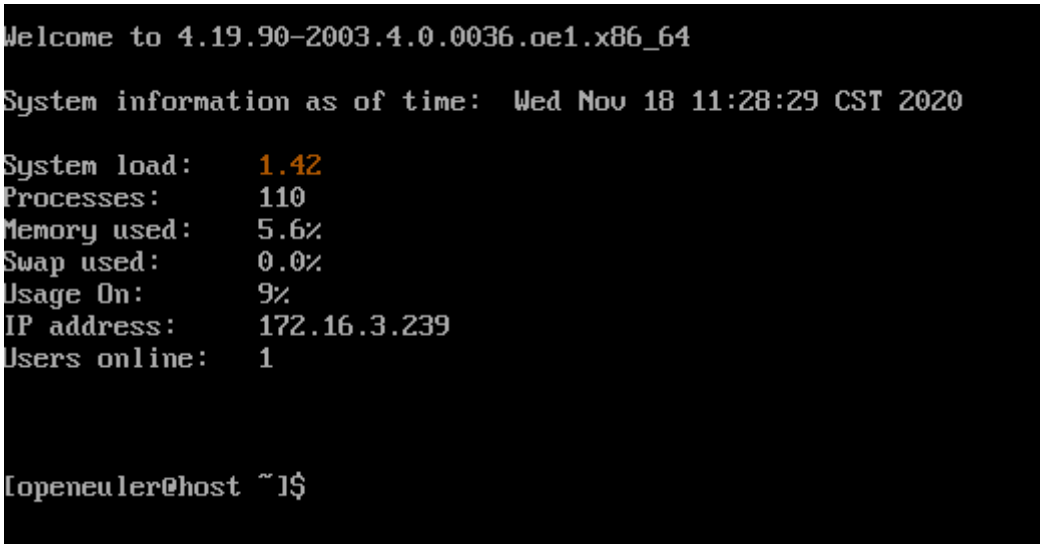


图1-7 openEuler 普通用户登录界面

现在可以开始使用 openEuler 操作系统了。

通过以下命令查看系统信息。对于命令的解释会在后续内容中进行介绍，此处先做一下简单的练习。

- 查看系统信息。

```
[root@host ~]#  cat  /etc/os-release
```

- 查看 CPU 信息。

```
[root@host ~]# lscpu
```

- 查看内存信息。

```
[root@host ~]# free
```

- 查看磁盘信息。

```
[root@host ~]# fdisk  -l
```

- 查看 IP 地址。

```
[root@host ~]# ip addr
```

1.4 思考题

- Windows 操作系统和 Linux 操作系统有什么不同?
- 什么是开源软件?
- x86 架构和 ARM 架构有什么不同?
- root 用户与普通用户的区别是什么?
- 除了 bash 之外，还有哪些 shell 工具？它们之间有什么区别?

2 命令行基础操作

2.1 Linux 命令行基础知识

从上一章的学习中，已了解到 shell 程序提供了一个界面，通过这个界面就可以访问操作系统内核。shell 相当于 Linux 内核外的一层壳，如下图 2-1 所示。

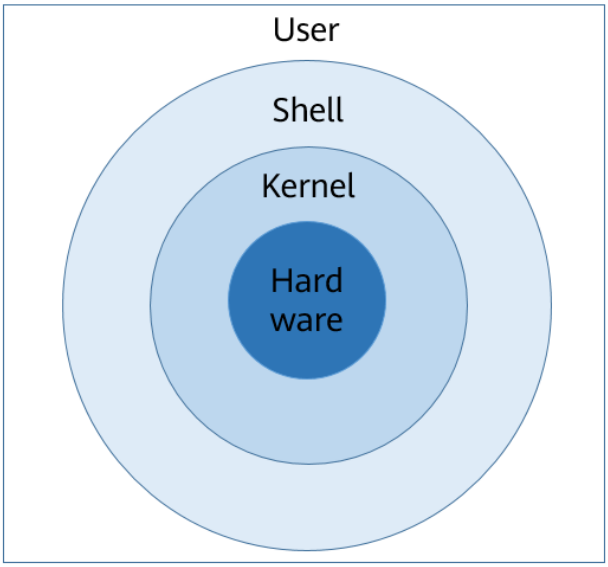


图2-1 Shell 在 Linux 系统中的位置

openEuler 用户默认登录 bash shell 命令行执行操作。为什么要使用命令行进行操作呢？有以下几个特点：

- 命令行更高效。Linux 系统中使用键盘操作速度比鼠标更快，并且命令行可以通过编写的脚本完成所有过程，例如删除过期日志文件，而图形化界面不可重复运行。
- 图形化界面开销大。运行图形化界面会占用很多的系统资源，运行命令行可以节约开销，让系统释放更多资源给它更应该做的事情。
- 命令行有时候是唯一的选择。大部分服务器操作系统不会安装图形界面（GUI），并且联网设备的维护管理工具本来就没有图像化界面供用户使用。

本课程将基于命令行的方式学习 openEuler 操作系统。Linux 系统中一般遵循的命令格式为：**命令【 -选项 】【 参数 】**，其中【 】表示非必选项。由此可见，该命令语句由三部分组成：

- 命令，即要运行的命令，以 ls -la 为例，其中 ls 命令即为显示文件列表，在 linux 命令行界面使用的命令字唯一确定一个命令
- 【 -选项 】，根据命令的不同，选项的个数和内容也不同。

- 当有多个选项时，可以写在一起，如 ls -la 实际包含两个选项，-l 和-a。-l 表示显示文件列表时同时列出文件的详细信息；-a 表示显示当前目录下所有文件或目录的信息。
- 【-选项】分为简化选项和完整选项。简化选项只有一个半角减号符，如 ls -a；完整选项的选项前用灵感半角减号符引导开始，如 ls --all。

- 【参数】，即命令的操作对象，通常情况可以是文件名、目录或用户名。

在操作 Linux 命令时先了解相关的 Linux 命令行操作技巧对于熟练掌握 Linux 操作非常有帮助。首先是 tab 键命令，在 Bash 环境里，使用 tab 键可以自动补全命令或文件名，可以通过多按几次的方式帮助用户准确快速的输入命令。未输入命令状态下，连按两次 tab 键列出所有可用命令。已输入部分命令名或文件名，按 tab 键自动补全。

除了 tab 键以外，下表中列出了常用的快捷键技巧。

表2-1 常用 Linux 命令行操作技巧

快捷键	作用
up 方向键上	可以调出输入历史执行记录，快速执行命令
down 方向键下	配合 up 选择历史执行记录
Home	移动光标到本行开头
Ctrl + A	移动光标到行首
Ctrl + E	移动光标到行尾
Ctrl + C	终止当前程序
Ctrl + L	清理屏幕显示

2.2 Linux 基础命令

Linux 命令可以分为以下几类，接下来的章节内容中会对不同类型中常用的命令进行介绍：

表2-2 Linux 命令分类

分类	命令	对应章节
登录和电源管理	login、shutdown、halt、reboot、install、exit、last等。	《02 命令行操作基础》
文件处理	file、mkdir、grep、dd、find、mv、ls、diff、cat、ln等。	《02 命令行操作基础》 《03 文本编辑器及文本处理》
系统管理	df、top、free、quota、at、ip、kill、crontab等。	《07 系统管理》

网络操作	ifconfig、ip、ping、netstat、telnet、ftp、route、rlogin、rcp、finger、mail、 nslookup等。	《07 系统管理》
文件系统及存储操作	fdisk、df、parted、mkfs、pvcreate、vgcreate、lvcreate、vgs、lvextend、mount、format等	《06 管理文件系统及存储》
系统安全	passwd、su、umask、chgrp、chmod、chown、chattr、sudo ps、who等。	《04 用户和权限管理》
其它	tar、unzip、gunzip、unarj、mtools、man。	《02 命令行操作基础》

- 注：Linux 命令远远不止表中这些，后续章节仅会基于常用的命令进行讲解及使用。

2.2.2 登录命令

2.2.2.1 登录命令介绍

login

当打开安装好 openEuler 的主机后，首先会要求登录系统，如果选择用命令行模式登录 Linux 的话，看到的第一个 Linux 命令就是 login。输入 root 用户名或者安装时已定义好的用户名后，按“Enter”键在 Password 后输入对应的账户密码，即可登录系统。出于安全考虑，输入账户密码时字符不会在屏幕上回显，光标也不移动。

Linux 是一个真正的多用户操作系统，可以同时接受多个用户登录，还允许一个用户进行多次登录。这是因为 Linux 和许多版本的 Unix 一样，提供了虚拟控制台的访问方式，允许用户在同一时间从控制台（系统的控制台是与系统直接相连的监视器和键盘）进行多次登录。每个虚拟控制台可以看作是一个独立的工作站，工作台之间可以切换。虚拟控制台的切换可以通过按下 Alt 键和一个功能键来实现，通常使用 F1-F6 。例如，用户登录后，按一下“Alt+F2”键，用户就可以看到上面出现的“login:”提示符，说明用户看到了第二个虚拟控制台。然后只需按“Alt+F1”键，就可以回到第一个虚拟控制台。一个新安装的 Linux 系统允许用户使用“Alt+F1”到“Alt+F6”键来访问前六个虚拟控制台。虚拟控制台最有用的是，当一个程序出错造成系统死锁时，可以切换到其它虚拟控制台工作，关闭这个程序。

last

last 命令的作用是显示近期用户或终端的登录情况，使用权限是所有用户。通过 last 命令查看该程序的 log，管理员可以获知谁曾经或企图连接系统。主要参数如下：

- -n：指定输出记录的条数。
- -t tty：只显示指定的虚拟控制台上登录情况。
- -y：显示记录的年、月、日。
- -ID：知道查询的用户名。
- -x：显示系统关闭、用户登录和退出的历史。

exit

exit 命令的作用是退出当前 shell，它的使用权限是所有用户。

logout

logout 命令作用是登出系统，相当于注销。它的权限是所有用户。使用 logout 的前提是当前 shell 是登录 shell 才可以。

2.2.2.2 登录命令练习

通过以下操作练习使用登录命令。

- root 用户登录

```
login as: root
root@121.36.14.101's password:

Authorized users only. All activities may be monitored and reported.
Activate the web console with: systemctl enable --now cockpit.socket

Last login: Mon Dec 7 16:47:53 2020



Welcome to 4.19.90-2003.4.0.0036.oe1.x86_64

System information as of time:  2020 年 12 月 7 日 星期一 16:49:27 CST

System load:    0.09
Processes:      129
Memory used:    2.8%
Swap used:      0.0%
Usage On:       9%
IP address:     172.16.3.132
Users online:   1




[root@host ~]#
```

- ALT+F2 可以打开新的虚拟控制台
- 切换成 openEuler 用户

```
[root@host ~]# su openEuler
[openEuler@host ~]$ exit        #退出
```

- 查看近期用户登录情况

```
[root@host ~]# last root
```

- 查看近期终端的登录情况

```
[root@host ~]# last tty2
```

2.2.3 电源命令

shutdown

shutdown 命令的作用是关闭计算机，使用权限是超级用户。对于计算机系统来说，超级用户(Superuser)是一种用于进行系统管理的特殊用户，相比其他普通用户来说，它拥有最高权限，能够进行全系统的配置、维护等工作，做很多普通用户没有权限做的事情；而普通用户的权限一般是超级用户的子集，只具备其部分权限。主要参数如下：

- -h：关机后关闭电源。
- -r：关机后打开电源（相当于重启）。
- -t：在改变到其它运行级别之前，告诉 init 程序多久以后关机。
- -k：并不真正关机，只是送警告信号给每位登录者。
- -F：在重启计算机时强迫 fsck。
- -time：设定关机前的时间。

shutdown 命令可以安全地将系统关机，使用直接断掉电源的方式来关闭 Linux 系统十分危险。Linux 与 Windows 不同，其后台运行着许多进程，所以强制关机可能会导致进程的数据丢失，使系统处于不稳定的状态，甚至在有的系统中会损坏硬件设备。在系统关机前使用 shutdown 命令，系统管理员会通知所有登录的用户系统将要关闭，并且 login 指令会被冻结，即新的用户不能再登录。

halt

halt 命令的作用是关闭系统，使用权限是超级用户。halt 执行时，杀死应用进程，执行 sync（将存于 buffer 中的信息强制写入硬盘中）系统调用，文件系统写操作完成后就会停止内核。若系统的运行级别为 0 或 6，则关闭系统；否则以 shutdown 指令（加上-h 参数）来取代。主要参数如下：

- -n：防止 sync 系统调用，它用在用 fsck 修补根分区之后，以阻止内核用老版本的超级块覆盖修补过的超级块。sync 命令可用来强制将内存缓冲区中的数据立即写入磁盘中。fsck 命令用于检查并且试图修复文件系统中的错误。超级块位于块组的最前面，描述文件系统整体信息的数据结构，主要描述文件系统的目录和文件的静态分布情况，以及描述文件系统的各种组成结构的尺寸、数量等。
- -w：并不是真正的重启或关机，只是写 wtmp（/var/log/wtmp）纪录。/var/log/wtmp 是一个二进制文件，记录每个用户的登录次数和持续时间等信息。
- -f：没有调用 shutdown，而强制关机或重启。
- -i：关机（或重启）前，关掉所有的网络接口。
- -f：强迫关机，不呼叫 shutdown 这个指令。
- -d：关闭系统，但不留下纪录。

reboot

reboot 命令的作用是重新启动计算机，使用权限是系统管理者。主要参数如下：

- -n: 保存数据后再重新启动系统。
- -w: 并不会真的重开机，只是把记录写到/var/log/wtmp 文件里。
- -d: 不把记录写到/var/log/wtmp 文件里（-n 这个参数包含了-d）。
- -i：关闭网络设置之后再重新启动系统。

2.2.4 文件管理基本操作命令

2.2.4.1 文件目录

在 Linux 操作系统中，一切皆是文件，Linux 中的所有内容都通过树形的文件目录结构进行管理，并且以最顶端根目录（ / ）为起点逐层向下延伸，延伸出目录和子目录的分支，如下图 2-2。根目录是整个系统中最重要的一个目录，同时根目录也与开机、还原、系统修复等动作有关。目录层级之间也是通过“/”符号进行划分，如/home/openEuler 是指 home 目录下的 openEuler 目录。

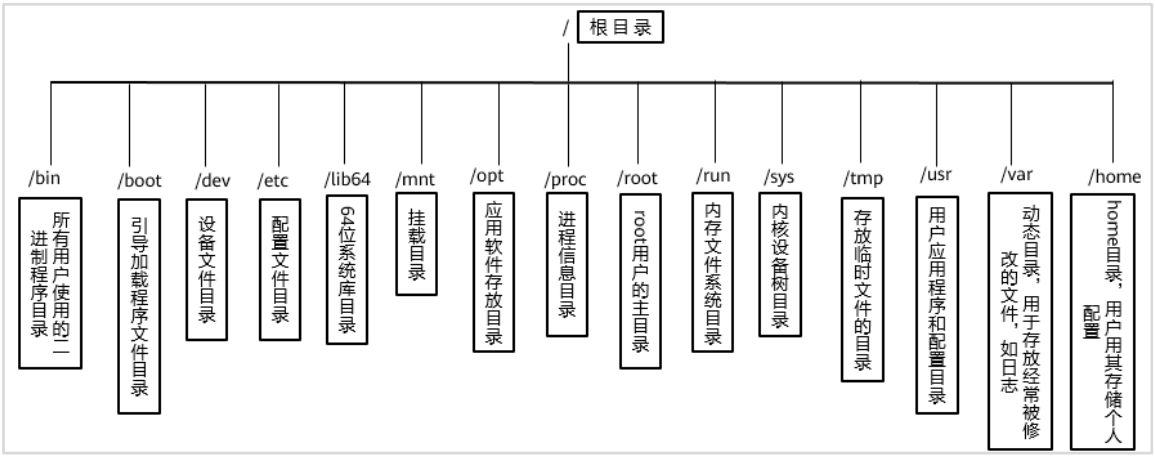


图2-2 文件系统的目录结构与根目录

登录系统后，使用 ls /命令即可查看根目录下的文件或文件目录，命令与回显如下：

```
[root@host ~]# ls /
bin   dev   home   lib64        media   opt    root   sbin   sys   usr
boot  etc   lib    lost+found   mnt     proc   run    srv    tmp   var
```

根目录下的这些目录的用途参考以下表格：

表2-3 Linux 主要目录及用途

目录名	主要存放的文件及其用途
/bin	bin是Binary的缩写，这个目录存放着最经常使用的命令。
/boot	这里存放的是启动Linux时使用的一些核心文件，包括一些连接文件以及镜像文件。
/dev	dev是Device(设备)的缩写，该目录下存放的是Linux的外部设备，在Linux中访问设备的方式和访问文件的方式是相同的。
/etc	这个目录用来存放所有的系统管理所需要的配置文件和子目录。
/home	用户的主目录，在Linux中，每个用户都有一个自己的目录，一般该目录名是以用户的账号命名的。

/lib	这个目录里存放着系统最基本的动态连接共享库，其作用类似于Windows里的DLL文件。几乎所有的应用程序都需要用到这些共享库。/lib64存放的是64位系统的共享库。
/mnt	系统提供该目录是为了让用户临时挂载别的文件系统的。
/opt	这是给主机额外安装软件所摆放的目录。
/proc	系统内存映射的虚拟目录，可以通过直接访问这个目录来获取系统信息。
/root	该目录为系统管理员，也称作超级权限者的用户主目录。
/sbin	s就是Super User的意思，这里存放的是系统管理员使用的系统管理程序。
/srv	该目录存放一些服务启动之后需要提取的数据。
/tmp	这个目录是用来存放一些临时文件的。
/usr	这是一个非常重要的目录，用户的很多应用程序和文件都放在这个目录下，类似于windows下的program files目录，其中/usr/bin是系统用户使用的应用程序；/usr/sbin是超级用户使用的比较高级的管理程序和系统守护程序；/usr/src是内核源代码默认的放置目录。
/var	习惯将那些经常被修改、不断扩充的目录放在这个目录下，包括各种日志文件。
/run	是一个临时文件系统，存储系统启动以来的信息，当系统重启时被清理或删除。

2.2.4.2 文件路径

在用 shell 或调用应用程序的时，都要写明被调用的程序路径。路径分为绝对路径和相对路径。

- 绝对路径：在 Linux 中，绝对路径是由根目录（ / ）开始写起的文件名或者目录名称，不依赖于当前在目录结构中的位置，如果一个路径是从/开始的，那一定是绝对路径。
- 相对路径：相对于当前路径的文件名或者目录名称，由目录结构中的当前的位置开始。

比如有/home/smc/config 和/home/smc/bin 两个目录，若当前用户正处在/home/smc/config 目录下，现在需要进入/home/smc/bin 目录，使用两种路径的方法分别为：

- 绝对路径：cd /home/smc/bin
- 相对路径：cd ../bin，这里的“..”表示上一级目录，若仅有一个点，“.” 表示当前目录。

2.2.4.3 文件基本操作命令

ls

ls 是 list 的意思，是 Linux 命令中使用最频繁的命令之一，用于列出目录的内容，或者文件的信息，该命令的输出结果默认按照文件名排序，如果不指定目标，则列出当前目录的内容。语法如下：

ls [OPTIONS] [FILE]

- -a：显示所有文件及目录包含隐藏的文件和目录（ls 内定将文件名或目录名称开头为“.”的视为隐藏文档）。
- -l：除文件名称外，将文件型态、权限、拥有者、文件大小等信息详细列出。
- -t：文件将根据建立时间的先后次序依次列出。
- -R：若目录下有文件，则以下之文件也都按顺序列出。

cd

cd 命令用于改变当前工作目录。语法如下：

cd [dir]

- cd /usr 表示进入目录 /usr 中。
- cd . 表示进入当前目录。
- cd .. 表示进入（退到）上一层目录，两个点代表父目录。
- cd – 表示进入前一个目录，适用于在两个目录之间快速切换。
- cd ~ 表示进入主目录，若是 root 用户则回到/root 目录下，若是普通用户则回到、home 目录下。
- cd 不带参数，则默认回到主目录。

pwd

pwd 是 print working directory 的意思，该命令用于打印出当前的工作目录。pwd 命令有两个选项，-L 和 -P，-L 目录连接链接时，输出连接路径。-P 输出物理路径。

ls、cd、pwd 文件命令练习：

- 查看根目录下的文件或文件目录：

```
[root@host ~]# ls /
```

- 进入根目录：

```
[root@host ~]# cd /
```

- 查看根目录下所有文件的详细信息并按时间顺序排列：

```
[root@host /]# ls –alt
```

- 使用绝对路径进入到 home 目录下：

```
[root@host /]# cd /home
```

- 使用绝对路径查看家目录下文件：

```
[root@host home]# ls /home
```

- 使用相对路径进入家目录下查看到的文件目录，如 openEuler：

```
[root@host home]# cd openEuler
```

- 查看当前所在的文件目录位置：

```
[root@host openEuler]# pwd
#由于已经进入到 home 目录下的 openEuler 文件目录，则此时使用 pwd 回显如下：
/home/openEuler
```

- 返回到上一级目录：

```
[root@host openEuler]# cd ..
```

- 返回到主目录下：

```
[root@host home]# cd ~
```

- 查看当前所在的文件目录位置：

```
[root@host ~]# pwd
#若使用的是 root 用户，则此时使用 pwd 回显如下：
/root
```

mkdir

mkdir 是 make directory 的意思，该命令用于创建目录（文件夹），可以一次性创建多个目录，如果目录已经存在，默认会报错，-p 选项可以使 mkdir 命令在这种情况下不报错，-p 选项还可以用于自动创建不存在的父目录。语法如下：

```
mkdir [OPTIONS] DIRECTORY
```

常见用法如下：

- 创建一个名叫 dir1 的目录：

```
[root@host ~]# mkdir dir1
```

- 创建多个目录：

```
[root@host ~]# mkdir dir1 dir2
```

- 如果 dir1, dir2 不存在，则一并创建 dir1 父目录，dir1/dir2 子目录和 dir1/dir2/dir3 子目录：

```
[root@host ~]# mkdir -p dir1/dir2/dir3
```

- 创建目录同时显示创建过程：

```
[root@host ~]# mkdir -pv dir1/dir2
```

touch

touch 命令可用于创建空文件，也可用于修改文件的时间戳。语法如下：

```
touch [OPTIONS] FILE
```

常见用法如下：

- 创建一个名为 file 的空白文件，并且时间戳为当前时间：

```
[root@host ~]# touch file
```

- 仅修改文件的访问时间：

```
[root@host ~]# touch -a file
```

- 仅修改文件的内容改变时间：

```
[root@host ~]# touch -m file
```

- 把文件的时间戳设定为指定的时间：

```
[root@host ~]# touch -d "2020-12-07 17:14:10" file
```

cp

cp 是 copy 的意思，该命令用于复制文件或者目录，可以一次复制单个文件，也可以一次复制多个文件，但需要注意的是 cp 命令属于高危命令，使用不慎就会有丢失数据的危险。语法如下：

```
cp [OPTIONS] SOURCE DIRECTORY_or_FILE
```

- -a：此选项通常在复制目录时使用，它保留链接、文件属性，并复制目录下的所有内容。
- -p：除复制文件的内容外，还把修改时间和访问权限也复制到新文件中。
- -r：若给出的源文件是一个目录文件，此时将复制该目录下所有的子目录和文件。

- -l：不复制文件，只是生成链接文件。

常见用法如下：

- 把文件 f1 复制一份，新文件名为 f2。

```
[root@host ~]# cp f1 f2
```

- 复制 f1 到目录 d1 下，新文件名字不变。

```
[root@host ~]# cp f1 d1/
```

- 复制多个文件到同一个目录中。复制多个文件时，目的位置必须是一个目录。

```
[root@host ~]# cp f1 f2 f3 d1/
```

- 如果 f2 已经存在，则覆盖之前等用户确认。若输入"y"则表示确认覆盖。

```
[root@host ~]# cp -i f1 f2
```

- 复制目录 d1 下所有的子目录和文件，新目录名为 d2，复制目录时需要-r 参数。

```
[root@host ~]# cp -r d1 d2
```

- 复制目录 d1 下所有的子目录和文件为 d2 的同时显示复制的过程，加入-v 参数可以显示复制的过程。

```
[root@host ~]# cp -rv d1 d2
```

- 使用-a 参数，在复制 f1 文件时保留原文件的属性，可用于复制块设备，字符设备，管道文件等。

```
[root@host ~]# cp -a f1 f2
```

mv

mv 意思是 move，该命令用于移动件或者目录，同 cp 命令一样，这是一个高危命令，使用不慎就会有丢失数据的危险，使用时需要注意。如果原文件和目标文件在同一个父目录里面，则 mv 命令的效果就相当于给文件改名。语法如下：

mv [OPTIONS] SOURCE DIRECTORY_or_FILE

- -b ： 若需覆盖文件，则覆盖前先行备份。
- -f ： force 强制的意思，如果目标文件已经存在，不会询问而直接覆盖。
- -i ： 若目标文件（destination）已经存在时，就会询问是否覆盖。
- -u ： 若目标文件已经存在，且 source 比较新，才会更新(update)。

mv 和 cp 的完全用法相同，常见用法如下：

- 把文件 f1 移动为 f2。

```
[root@host ~]# mv f1 f2
```

- 移动 f1 到目录 d1 下，新文件名字不变。

```
[root@host ~]# mv f1 d1/
```

- 移动多个文件到同一个目录中。

```
[root@host ~]# mv f1 f2 f3 d1/
```

- 如果 f2 已经存在，则覆盖之前等用户确认。若输入”y”则表示确认覆盖。

```
[root@host ~]# mv -i f1 f2
```

- 移动 f1 到已存在的 f2，在覆盖 f2 前先对 f2 进行备份。

```
[root@host ~]# mv –b f1 f2
```

- 移动目录 d1 下所有的子目录和文件，新目录名为 d2，移动目录时需要-r 参数。

```
[root@host ~]# mv -r d1 d2
```

- 移动目录 d1 下所有的子目录和文件为 d2 的同时显示移动的过程。

```
[root@host ~]# mv -rv d1 d2
```

- 移动 f1 到 f2，若 f2 已存在，使用-f 不需要询问直接进行覆盖，与-i 选项的作用相反。如果命令同时包含-i 和-f，则写在右边的生效。

```
[root@host ~]# mv -f f1 f2
```

rm

rm 是 remove 的意思，该命令用于删除文件或者目录。rm 命令也属于高危命令，没有一个工具能够 100%恢复 rm 命令删除的文件，rm 命令删除文件时并不是把文件放到类似图形界面的“回收站”里，所以没有“撤销删除”操作可用。使用的时候需要谨慎。语法如下：

rm [OPTIONS] DIRECTORY_or_FILE

- -i：进行交互式删除，在删除前进行询问，确认是否需要删除。
- -f：强制删除文件，不会询问而直接删除。
- -r：指示 rm 将参数中列出的全部目录和子目录均递归地删除。
- -v：详细显示进行的步骤。

常见用法如下：

- 删除 f1 文件。

```
[root@host ~]# rm f1
```

- 删除 f1、f2 多个文件。

```
[root@host ~]# rm f1 f2
```

- 交互式删除 f1 文件，删除之前等用户确认。若输入”y”则表示确认删除。

```
[root@host ~]# rm -i f1
```

- 强制删除文件 f1。如果命令同时包含-i 和-f，则写在右边的生效。

```
[root@host ~]# rm -f f1
```

- 递归删除 d1 目录与目录下的所有子目录和文件。

```
[root@host ~]# rm -r d1
```

- 递归删除 d1 目录与目录下的所有子目录和文件，并显示删除过程。

```
[root@host ~]# rm –rv d1
```

find

find 命令用来在指定目录下查找文件。可以指定一些匹配条件，如按文件名、文件类型、用户甚至是时间戳查找文件。语法如下：

find [PATH] [EXPRESSION]

- -name 按照文件名查找文件。
- -perm 按照文件权限来查找文件 。
- -user 按照文件属主来查找文件 。
- -mtime -n +n 按照文件的更改时间来查找文件。
- 查找名字中包含了 book 的文件。

```
[root@host ~]# find -name "*book*"
```

- 查找大小为 0 的文件。

```
[root@host ~]# find -size 0
```

- 查找文件类型为软链接的文件。

```
[root@host ~]# find -type l
```

- 在/etc 下面查找包含 passwd 的文件。

```
[root@host ~]# find /etc -name "*passwd"
```

- 查找空文件或目录并将其删除。

```
[root@host ~]# find -empty -delete
```

locate

locate 可以快速的查找文件系统内是否有指定的文件。locate 查找时，先建立一个文件名及路径的数据库，查找时去这个数据库内查询。语法如下：

```
locate [OPTIONS] PATTERN
```

- -e：将排除在寻找的范围之外。
- -f：将特定的文件排除在外。
- -r：使用正规运算式做查找条件。
- -o：指定文件的名称。
- -d：指定文件的路径。

当用户在执行 loacte 命令查找文件时，它会直接在索引数据库里查找，若该数据库太久没更新或不存在，在查找文件时就提示：“locate: can not open `/var/lib/mlocate/mlocate.db‘: No such file or directory”，此时执行“updatedb”更新下数据库即可。

which

which 命令在 PATH 所指定的目录中查找可执行文件。使用 which 命令，就可以看到某个系统命令是否存在，以及执行的到底是哪一个位置的命令。语法如下：

```
which [OPTIONS] PROGRAMNAME
```

常见用法如下：

- 查找 ls 命令的绝对路径。

```
[root@host ~]# which ls
```

- 查找 ls 命令的绝对路径，如果多个目录中都有匹配的文件，则全部显示。

```
[root@host ~]# which –a ls
```

- 查找多个文件。

```
[root@host ~]# which cp mv rm
```

ln

ln 是 link 的意思，该命令用于创建链接文件。ln 的功能是为某一个文件在另外一个位置建立一个同步的链接。当用户需要在不同的目录，用到相同的文件时，用户不需要在每一个需要的目录下都放一个必须相同的文件，用户只要在某个固定的目录，放上该文件，然后在 其它的目录下用 ln 命令链接（link）它就可以，不必重复的占用磁盘空间。

在 Linux 中有软链接（symbolic link）和硬链接（hard link）两类，如下表所示。

表2-4 Linux 里的链接文件类型

软链接（symbolic link）	硬链接（hard link）
以路径形式存在，类似于Windows的快捷方式	以文件副本形式存在，但不占用实际空间
删除源文件后链接失效	删除源文件后影响
可以对目录进行链接	不可以对目录进行链接
可以跨文件系统	不可以跨文件系统

ln 命令在不带参数的情况下，默认创建的是硬链接。语法如下：

ln [OPTIONS] SOURCE [DIRECTORY_or_FILE]

- -b：删除，覆盖以前建立的链接。
- -d：允许超级用户制作目录的硬链接。
- -f：强制执行。
- -i：交互模式，若文件存在则提示用户是否覆盖。
- -n 把符号链接视为一般目录。
- -s 软链接。

常见用法如下：

- 对 f1 文件建立硬链接，并命名为 f2。

```
[root@host ~]# ln f1 f2
```

- 对 d1 目录建立软链接，并命名为 d2。

```
[root@host ~]# ln -s d1 d2
```

- 交互式建立 f1 文件链接，并命名为 f2，当 f2 已存在时，会询问用户是否继续创建。若输入”y”则表示确认创建并覆盖原有文件。

```
[root@host ~]# ln –i f1 f2
```

- 强制创建 f1 与 f2 的文件链接。

```
[root@host ~]# ln –f f1 f2
```

- 对 f1 创建链接文件 f2 时，若已存在 f2 则覆盖。

```
[root@host ~]# ln –b f1 f2
```

更多关于文件基本操作的命令练习请参考《HCIA-openEuler 实验手册-PC 版》2.2 bash 命令基本操作。

2.2.4.4 文件打包和压缩命令

gzip

gzip 是在 Linux 系统中经常使用的一个对文件进行压缩和解压缩的命令，gzip 不仅可以用来压缩大的、较少使用的文件以节省磁盘空间。据统计，gzip 命令对文本文件有 60%～70%的压缩率，文件经过 gzip 压缩过后，其名称后面会多出".gz"的扩展名。语法如下：

gzip [OPTIONS] DIRECTORY_or_FILE

- -d：解开压缩文件。
- -f：强行压缩文件，不理会文件名是否存在以及该文件是否为符号连接。
- -l：列出压缩文件的相关信息。
- -r：将指定目录下的所有文件及子目录一并递归压缩处理。
- -v：显示指令执行过程。

常见用法如下：

- 对文件 f1 进行压缩。

```
[root@host ~]# gzip f1
```

- 对目录 d1 下的所有文件及子目录进行压缩。

```
[root@host ~]# gzip -r d1
```

- 解压文件 f1。

```
[root@host ~]# gzip -d f1
```

- 解压目录 d1 进行解压并显示执行过程。

```
[root@host ~]# gzip –drv d1
```

tar

tar 命令可用于打包文件，把多个文件打到一个包中，方便数据的移动。tar 命令通常和压缩命令配合起来使用，-z, -j, -J 选项分别对应着 gzip, bzip2, xz 这三个压缩工具，当指定了压缩选项后，tar 就会启动相应的压缩工具来做压缩或者解压工作，并通过管道与压缩工具传输数据。语法如下：

tar [OPTIONS] [FILE]

- -c：建立新的压缩文件。
- -x：从压缩的文件中提取文件。
- -t：显示压缩文件的内容。
- -z：支持 gzip 解压文件。
- -j：支持 bzip2 解压文件。
- -v：显示操作过程。

常见用法如下：

- 把目录 dir1 及其下所有内容打包。

```
[root@host ~]# tar -cf ball.tar dir1
```

- 列出包中的内容。

```
[root@host ~]# tar -tf ball.tar
```

- 把包中的内容解压到当前目录。

```
[root@host ~]# tar -xf ball.tar
```

- 打包然后用 gzip 压缩。

```
[root@host ~]# tar -czf ball.tar.gz dir1
```

- 打包然后用 bzip2 压缩。

```
[root@host ~]# tar -cjf ball.tar.bz2 dir1
```

- 打包然后用 xz 压缩。

```
[root@host ~]# tar -cJf ball.tar.xz dir1
```

- 解到/tmp 目录下（默认在当前目录）。

```
[root@host ~]# tar -xf ball.tar -C /tmp
```

- 显示解压过程过程。

```
[root@host ~]# tar -xvf ball.tar
```

更多关于文件基本操作的命令练习请参考《HCIA-openEuler 实验手册-PC 版》2.4 打包和压缩命令。

2.2.5 帮助命令

man

Linux 系统提供了丰富的文档，常见的有 man 文档，info 文档，txt 分档等，man 文档用 man 工具查看，info 文档用 info 工具查看，txt 文档可以用各种文本阅读器查看。man 命令用于查看文档手册（manual），分为以下 9 类，如下表所示。

表2-5 man 命令的类别

序号	代表内容
1	使用者在shell中可以操作的指令或程序
2	系统核心可调用的函数与工具等
3	一些常用的函数(function)与函数库(library)
4	设备文档的说明，通常是在/dev下的文件
5	文件格式和约定
6	游戏(games)
7	杂项（包括宏和惯例）
8	系统管理命令（通常仅适用于root用户）
9	内核例程（非标准）

man 文档常用的类型为上表中 1,4,5,8 这四类。man 是按照手册的章节号的顺序进行搜索的，比如：man sleep。默认只显示命令的手册，如果想查看库函数就要输入 man 3 sleep，类型 3 即代表一些常用的函数(function)与函数库(library)。可以通过关键字来查找 man 文档，输入命令 man –k KEYWORD，如此处输入关键字为 sleep，可以查看到关键字对应的命令所在的类型。

```
[root@host ~]# man -k sleep
```

回显如下图 2-3 所示：

```
[root@host-172-16-3-132 ~]# man -k sleep
sleep.conf.d (5)     - Suspend and hibernation configuration file
systemd-hibernate.service (8) - System sleep state logic
systemd-hybrid-sleep.service (8) - System sleep state logic
systemd-sleep (8)    - System sleep state logic
systemd-sleep.conf (5) - Suspend and hibernation configuration file
systemd-suspend-then-hibernate.service (8) - System sleep state logic
systemd-suspend.service (8) - System sleep state logic
usleep (1)           - sleep some number of microseconds
[root@host-172-16-3-132 ~]# man 3 sleep.conf.d
No manual entry for sleep.conf.d in section 3
[root@host-172-16-3-132 ~]# man -k sleep
sleep.conf.d (5)     - Suspend and hibernation configuration file
systemd-hibernate.service (8) - System sleep state logic
systemd-hybrid-sleep.service (8) - System sleep state logic
systemd-sleep (8)    - System sleep state logic
systemd-sleep.conf (5) - Suspend and hibernation configuration file
systemd-suspend-then-hibernate.service (8) - System sleep state logic
systemd-suspend.service (8) - System sleep state logic
usleep (1)           - sleep some number of microseconds
[root@host-172-16-3-132 ~]#
```

图2-3 通过关键字查找 man 文档

help

在 linux 系统中，命令太多，命令的选项和参数也纷繁多样，要全部记住几乎是不可能的，因此用户可以通过 help 命令获取帮助。语法如下：

help [OPTIONS] [COMMAND]

- -d：显示命令简短主题描述
- -s：显示命令简短语法描述

常见用法如下：

```
[root@host ~]# help pwd
[root@host ~]# help –d pwd
[root@host ~]# help –s pwd
```

如图 2-4 所示，以下是对 pwd 使用 help 命令的回显：

```
[root@host-172-16-3-132 ~]# help pwd
pwd: pwd [-LP]
    Print the name of the current working directory.

    Options:
      -L        print the value of $PWD if it names the current working
                directory
      -P        print the physical directory, without any symbolic links

    By default, `pwd' behaves as if `-L' were specified.

    Exit Status:
    Returns 0 unless an invalid option is given or the current directory
    cannot be read.
[root@host-172-16-3-132 ~]# help -d pwd
pwd - Print the name of the current working directory.
[root@host-172-16-3-132 ~]# help -s pwd
pwd: pwd [-LP]
[root@host-172-16-3-132 ~]#
```

图2-4 使用 help 获取 pwd 命令的帮助

更多关于帮助命令的练习请参考《HCIA-openEuler 实验手册-PC 版》2.5 帮助命令。

2.3 思考题

- 对不同用户使用 cd ~命令时，所进入的主目录是什么？
- shutdown 和 halt 命令有什么区别？
- 绝对路径和相对路径有什么区别，如何使用？
- 软链接和硬链接有什么区别？
- 删除源文件后对软链接和硬链接分别有什么影响？

3 文本编辑及文本处理

3.1 Linux 常用的文本编辑器

文本处理是操作系统对文件管理的基础操作，文本编辑器是操作系统基础的功能软件之一，主要用来编写和查看文本文件。不同的文件编辑器有不同的辅助功能。根据使用环境的不同，Linux 的文本编辑器有很多类型。常见的 Linux 文本编辑器有：emacs，nano，gedit，kedit，vi，vim。

emacs

emacs 是一款功能强大的编辑器，与其说是一款编辑器，它更像一个操作系统。emacs 带有内置的网络浏览器、IRC 客户端、计算器，甚至是俄罗斯方块。当然，emacs 需要在图形化界面的 Linux 中使用。它的功能强大并且可以进行扩展，能够与许多自由软件编程工具集成，但是对普通用户来说，有一定的入门门槛。

nano

nano 是命令行界面下一个相对简单的文本编辑器，它是为了代替闭源的 Pico 文本编辑器而开发的，1999 年以 GPL 协议发布第一个版本，是一个自由软件，同时也是 GNU 计划的一个组成部分。nano 有很多人性化的功能设计，如语法高亮、正则表达式搜索和替换、平滑滚动、多个缓冲区、自定义快捷键、撤销或重复编辑。nano 的操作较为简单，适用于简单的文本编辑，但对于复杂的文本编辑比较耗时，无强大的命令功能进行复杂操作，不支持如宏、一次编辑多个文件、窗口分割、垂直块/矩形选择/编辑、自动完成等高级功能。

gedit

gedit 是一个 GNOME 桌面环境下兼容 UTF-8 的文本编辑器。它简单易用，有良好的语法高亮，对中文支持很好，支持包括 GB2312、GBK 在内的多种字符编码。gedit 是一款自由软件。gedit 包含语法高亮和标签编辑多个文件的功能。利用 GNOME VFS 库，它还可以编辑远程文件。它支持完整的恢复和重做系统以及查找和替换。gedit 的操作习惯与 Windows 类似，包括常用的快捷键如复制粘贴等，因此操作起来较为容易。gedit 是通过图形化界面进行操作的，所以需要安装图形化桌面才能使用。

kedit

kedit 与 gedit 类似，kedit 是 KDE 图形化桌面中常用的一种文本编辑器。kedit 是一个非常小的编辑器，特别适用于浏览文本和各种配置文件。同样需要安装图形化桌面才能使用，操作习惯与 Windows 类似。

vi

vi 是标准的 Unix 文本编辑器，也是最古老的文本编辑器、最通用的文本编辑器。所有的 Linux、Unix 都默认带有 vi 文本编辑器。虽然 vi 的操作方式与其他常用的文本编辑器（如 gedit）很不相同 ，但是由于其运行于字符界面，并可用于所有 unix/linux 环境，仍被经常使用。vi 有三种命令模式:

- 命令模式：用于输入命令
- 插入模式：用于插入文本
- 可视模式：用于浏览文本

vim

vim 是从 vi 发展出来的一个文本编辑器。其代码补完、编译及错误跳转等方便编程的功能特别丰富，在程序员中被广泛使用。和 Emacs 并列成为类 Unix 系统用户最喜欢的编辑器。vim 的第一个版本由布莱姆·米勒在 1991 年发布。最初的简称是 Vi IMitation，随着功能的不断增加，正式名称改成了 Vi IMproved。现在是在开放源代码方式下发行的自由软件。从 vi 派生出来的 vim 具有多种模式:

- 基本模式:
  - 普通模式：在普通模式中，用的编辑器命令，比如移动光标，删除文本等等。这也是 vim 启动后的默认模式。这正好和许多新用户期待的操作方式相反（大多数编辑器默认模式为插入模式）。vim 强大的编辑能力来自于其普通模式命令。普通模式命令往往需要一个操作符结尾。例如普通模式命令"dd"删除当前行，但是第一个"d"的后面可以跟另外的移动命令来代替第二个"d"，比如用移动到下一行的"j"键就可以删除当前行和下一行。另外还可以指定命令重复次数，"2dd"（重复"dd"两次），和"dj"的效果是一样的。用户学习了各种各样的文本间移动 / 跳转的命令和其他的普通模式的编辑命令，并且能够灵活组合使用的话，能够比那些没有模式的编辑器更加高效的进行文本编辑。在普通模式中，有很多方法可以进入插入模式。比较普通的方式是按"a"（append / 追加）键或者"i"（insert / 插入）键。
  - 插入模式：在这个模式中，大多数按键都会向文本缓冲区中插入文本。大多数新用户希望文本编辑器编辑过程中一直保持这个模式。在插入模式中，可以按 ESC 键回到普通模式。
  - 可视模式：这个模式与普通模式比较相似。但是移动命令会扩大高亮的文本区域。高亮区域可以是字符、行或者是一块文本。当执行一个非移动命令时，命令会被执行到这块高亮的区域上。vim 的"文本对象"也能和移动命令一样用在这个模式中。
  - 选择模式：这个模式和无模式编辑器的行为比较相似（Windows 标准文本控件的方式）。这个模式中，可以用鼠标或者光标键高亮选择文本，不过输入任何字符的话，vim 会用这个字符替换选择的高亮文本块，并且自动进入插入模式。
  - 命令行模式：在命令行模式中可以输入会被解释成并执行的文本。例如执行命令（":"键），搜索（"/"和"?"键）或者过滤命令（"!"键）。在命令执行之后，vim 返回到命令行模式之前的模式，通常是普通模式。
  - Ex 模式：这和命令行模式比较相似，在使用":visual"命令离开 Ex 模式前，可以一次执行多条命令。
- 派生模式
  - 操作符等待模式：这个派生模式指普通模式中，执行一个操作命令后 Vim 等待一个"动作"来完成这个命令。Vim 也支持在操作符等待模式中使用"文本对象"作为动作，包括"aw"一个单词（a word）、"as"一个句子（a sentence）、"ap"一个段落（a paragraph）等等。
  - 插入普通模式：这个模式是在插入模式下按下 ctrl-o 键的时候进入。这个时候暂时进入普通模式，执行完一个命令之后，Vim 返回插入模式
  - 插入可视模式：这个模式是在插入模式下按下 ctrl-o 键并且开始一个可视选择的时候开始。在可视区域选择取消的时候，Vim 返回插入模式。
  - 插入选择模式：通常这个模式由插入模式下鼠标拖拽或者 shift 方向键来进入。当选择区域取消的时候，Vim 返回插入模式。

- 替换模式：这是一个特殊的插入模式，在这个模式中可以做和插入模式一样的操作，但是每个输入的字符都会覆盖文本缓冲中已经存在的字符。在普通模式下按"R"键进入。

- Evim
  - Evim（Easy Vim）是一个特殊的 GUI 模式用来尽量的表现的和"无模式"编辑器一样。编辑器自动进入并且停留在插入模式，用户只能通过菜单、鼠标和键盘控制键来对文本进行操作。可以在命令行下输入"evim"或者"vim -y"进入。

3.2 文本编辑及文本处理

3.2.1 使用 vim 编辑器

通过使用 vim 可以对文本进行编辑、修改、保存等操作，同时提供的多种快捷键也方便用户对文本文件进行操作，以下讲从打开文件、移动光标、数据操作、行号显示与取消、查找与替换、设置搜索高亮、修改文件、撤销或重做、保存文件并退出几个方面对 vim 编辑器的使用进行介绍。

vim 打开文件

使用 vim 命令打开文件，语法如下：

```
vim [OPTIONS] [FILE]
```

- -c ： 打开文件前先执行指定的命令
- -R ： 以只读方式打开，但是可以强制保存
- -M ： 以只读方式打开，不可以强制保存
- -r ： 回复崩溃的会话
- +*num* ： 从第 *num* 行开始

常见用法如下：

- 打开一个名为 filename 的文件，若该文件已存在，则会打开文件并显示文件内容，若该文件不存在，vim 会在下面提示 [New File]，并且会在第一次保存时创建该文件。

```
[root@host ~]$ vim filename
~
~
~
~
~
“filename” [New File]
```

- 以只读的方式打开 filename 文件，vim 会在下面提示[readonly]，并可以强制保存

```
[root@openEuler ~]$ vim filename
~
~
~
~
~
“filename” [readonly]
```

- 当执行编辑命令 i 时会出现如下显示：

-- INSERT – W10: Warning: Changing a readonly file

- 当执行保存命令:wq 时会提示需要加入！符号，进行强制覆盖保存，回显如下：

E45: ‘readonly’ option is set (add ! to override)

- 以只读方式打开 filename 文件，但无法强制保存

[root@host ~]# vim –M filename

- 当执行编辑命令 i 时会出现如下显示：

E21: Cannot make changes, ‘modifiable’ is off

- 当执行保存命令:wq 时会出现如下显示：

E142: File not written: Writing is disabled by ‘write’ option

移动光标

在 vim 普通模式下可以配合快捷键，快速移动光标，简化操作，常见快捷键如下表所示：

表3-1 常用的移动光标快捷键

移动光标快捷键	效果
h或向左箭头键(←)	光标向左移动一个字符
j或向下箭头键(↓)	光标向下移动一个字符
k或向上箭头键(↑)	光标向上移动一个字符
l或向右箭头键(→)	光标向右移动一个字符
0	光标移动到本行行首
$	光标移动到本行行尾
g0	光标移动到所在屏幕行行首
g$	光标移动到所在屏幕行行尾
:n	光标移动到第n行
gg	光标移动到文件头部
G	光标移动到文件尾部

数据操作

在 vim 普通模式下，可以使用快捷键对 vim 中的内容进行复制、粘贴、删除等数据操作，常见用法见下表：

表3-2 常用的数据操作快捷键

类型	快捷键	效果
复制	yy or y	复制整行文本

	y[n]w	复制n个词
粘贴	p在当前行下面	面向当前行下面进行粘贴
	p在当前行上面	面向当前行上面进行粘贴
	p在光标后面	光标字符后面进行粘贴
	p在光标前面	光标字符前面进行粘贴
删除/剪切	[n]dd	删除（剪切）n个单词
	d[n]w	删除（剪切）n行

行号显示与取消

在 vim 命令行模式下，可以通过命令显示文本行号，以便用户快速获取对应行内容，具体操作方法如下：

- 输入:set nu，显示行号

```
  1 hello
  2 openEuler
~
~
~
~
~
:set nu
```

- 输入:set nonu，取消显示行号

查找与替换

在 vim 命令行模式下，可以通过命令快速查找或替换文本内容，常见用法如下：

- 在光标之后查找一个字符串 word。

```
:/word
#按 n 向后继续搜索，shift+n 向上搜索。
```

- 在光标之前查找一个字符串 word。

```
:?word
#按 n 向后继续搜索。
```

- 将文档中 1-5 行的 word1 替换为 word2。

```
:1,5s/word1/word2/g
#若不加 g 则只替换每行的第一个 word1。
```

- 将文档所有的 word1 替换为 word2，不区分大小写。

```
:%s/word1/word2/gi
```

设置搜索高亮

在 vim 命令行模式下，可以临时设置搜索高亮，方便用户阅读，若需要进行永久设置，需要在 /etc/vimrc 中配置，增加一行 set hlsearch，然后更新变量即可。用法如下：

- 在文本中搜索“hello”时显示高亮：
  - 首先在命令行模式下设置高亮：

```
:set hlsearch
```

  - 然后在命令行模式下搜索“hello”：

```
:/hello
#回显如下：
hello
openEuler
hello
world
~
~
:/hello
```

由此可见，重复出现的 hello 都被标为黄色。

- 需要取消高亮则输入：

```
:set nohlsearch
```

修改文件

使用 vim filename 打开文件后，进入的是普通模式。当想要修改文件时，可以按 i 键进入插入模式。进入插入模式时，会在最下面提示当前模式是 Insert。按 ecs 可以退出插入模式，回到普通模式。用法如下：

- 按如下命令执行，即可在 filename 文件中输入“hello openEuler”，回显如下：

```
[root@host ~]# vim filename
hello openEuler
~
~
~
-- INSERT --
```

撤销或重做

若需要对插入的文本内容进行撤销或重做，常见用法如下：

- 在普通模式下输入 u，撤销最近的改变。例如撤销已经输入的“hello openEuler”，回显如下：

```
[root@openEuler ~]$ vim filename
~
~
~
~
1 line less; before #1 08:02:30
```

- 在普通模式下输入 U，撤销当前行自从光标定位在上面开始的所有改变。回显与输入 u 的显示类似。
- ctrl+r 重做最后一次“撤销”改变。例如需要恢复刚撤销的“hello openEuler”，回显如下：

```
[root@openEuler ~]$ vim filename
hello openEuler
~
~
~
1 more line; after #1  8 seconds ago
```

保存文件并退出

完成编写后需要对文本进行保存退出，首先需要退出插入模式，在插入模式下按 esc 键即可回到普通模式下，在普通模式下输入：进入命令行模式，通过命令行模式输入的指令执行保存或退出命令，常见用法如下：

- 输入:w，保存文档
- 输入:wq，保存并退出文档
- 输入:q!，强制退出但不保存，因此会造成文本丢失，需谨慎操作。
- 输入:q，仅退出，在未编辑过文本的情况下可以执行，但文本有修改会出现提示，修改过的文档需要通过其他指令如:wq 或:q！进行退出
- 输入:wq!，强制保存并退出

对于常用的 vim 基本模式：普通模式、插入模式和命令行模式可以进行互相切换，如下图 3-1 所示：

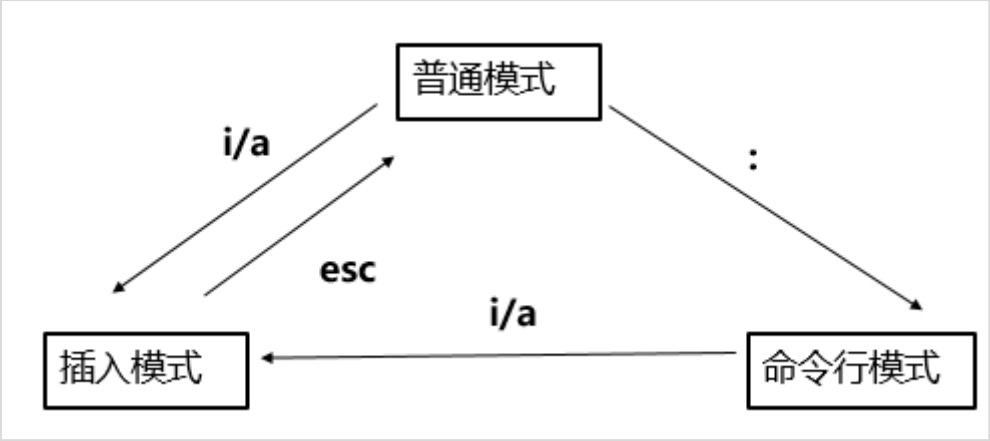


图3-1 vim 常用的三种基本模式间的切换

更多关于帮助命令的练习请参考《HCIA-openEuler 实验手册-PC 版》3 openEuler 文本编辑器。

3.2.2 文本处理

Linux 中对文本处理有以下几种不同类型的常见命令，如下表所示：

表3-3 常见的文本处理命令

类型	命令
查看文件	cat、more、less
文件摘选	head、tail
提取文件内容	cut、awk、grep
文本排序与比较	wc、sort、diff
文本操作工具	sed、tr

3.2.2.1 查看文件

查看文件 cat

cat 是一个文本文件查看和连接工具。语法如下：

```
cat [OPTIONS] [FILE]
```

- -n：从 1 开始对所有行编号并显示在每行开头
- -b：从 1 开始对非空行编号并显示在每行开头
- -s：当有多个空行在一起时只输出一个空行
- -E：在每行结尾增加$

常见用法如下：

- 显示 file1 文件的文件内容。

```
[root@host ~]# cat file1
```

- 编辑 file1 文件。

```
[root@host ~]# cat > file1
```

- 将 file1 和 file2 合并为一个文件 file3。

```
[root@host ~]# cat file1 file2 > file3
```

- 查看/etc/profile 文件内容，并对非空白行进行编号，编号从 1 开始。

```
[root@host ~]# cat -b /etc/profile
```

- 查看/etc/profile 文件内容，并在每行前面显示行号。

```
[root@host ~]# cat -n /etc/profile
```

- 查看/etc/profile 文件内容，但是不输出多行空行，当有多个空行在一起时，只输出一个空行。

```
[root@host ~]# cat -s /etc/profile
```

查看文件 more

more 可以一次查看文件或者标准输入的一页，与 cat 不同的是 more 可以按页来查看文件的内容，还支持直接跳转行等功能。语法如下：

```
more [OPTIONS] [FILE]
```

- +n：从第 n 行开始显示
- -n：定义屏幕大小为 n 行
- -c：从顶部清屏，然后显示
- -s：把连续的多个空行显示为一行

常用操作如下表所示：

表3-4 more 的常用操作

快捷键	用途
Enter	向下n行，需要定义。默认为1行。
Ctrl+F	向下滚动一屏
空格键	向下滚动一屏
Ctrl+B	向上滚动一屏
b	向上滚动一屏
=	输出当前行号
:f	输出文件名和当前行号
q	退出more

查看文件 less

less 命令读取内容，分屏显示，less 与 more 类似，但使用 less 可以随意浏览文件，而 more 仅能向前移动，却不能向后移动，而且 less 在查看之前不会加载整个文件。语法如下：

less [OPTIONS] [FILE]

- -f：强制打开特殊文件，例如外围设备代号、目录和二进制文件
- -g：只标志最后搜索到的关键字
- -i：忽略搜索时的大小写
- -N：显示每行的行号
- -s：当有多个空行在一起时只输出一个空行
- -o <文件名> ： 将 less 输出的内容保存到指定文件

常用操作如下表所示：

表3-5 less 的常用操作

快捷键	用途
b	向上翻一页
d	向下翻半页

h	显示帮助界面
q	退出less
u	向上翻半页
y	向上翻一行
空格键	向下翻一行
Enter	向下翻一页
上下键	向上/下翻一行

3.2.2.2 文件摘选

文件摘选 head

head 用来显示文件的开头至标准输出中，默认 head 命令可以显示文件的前 10 行。语法如下：

```
head [OPTIONS] [FILE]
```

- -q：输出时隐藏文件名，head 默认不显示文件名
- -v：输出时显示文件名
- -c *num*：显示前 *num* 个字节
- -n *num*：显示前 *num* 行，如果是负数

常见用法如下：

- 显示/etc/passwd 文件的前 20 行。

```
[root@host ~]# head –n 20 /etc/passwd
```

- 显示/etc/passwd 文件的除了最后 20 行的前面所有行，此时数值变量可以用负值表示，若该文件一共是 37 行，则会显示前 17 行内容。

```
[root@host ~]# head –n -20 /etc/passwd
```

文件摘选 tail

tail 用来显示文件的末尾至标准输出中，默认 tail 命令可以显示文件的后 10 行。语法如下：

```
tail [OPTIONS] [FILE]
```

- -f：循环读取，对于日志文件的监控非常有用
- -q：不显示文件名，tail 默认不显示文件名
- -v：输出时显示文件名
- -c *num*：显示文件最后 *num* 个字节
- -n *num*：显示文件最后 *num* 行

常见用法如下：

- 显示/etc/passwd 文件的后 20 行。

```
[root@host ~]# tail  –n 20 /etc/passwd
```

- 显示/etc/passwd 文件的 20 行之后的所有行，此时在变量数值前加入“+”号表示，若该文件一共是 37 行，则会显示 21 行到 37 行所有内容。

```
[root@host ~]# tail  -n +20 /etc/passwd
```

- 显示/etc/passwd 文件的第 11 到第 20 行。

```
[root@host ~]# head –n 20 /etc/passwd | tail –n 10
```

3.2.2.3 提取文件内容

提取列或字段 cut

cut 用于显示文件或者标准输入的特定列，语法格式如下：

```
cut [OPTIONS] [FILE]
```

- -b [*范围*]：仅显示行中指定直接范围的内容
- -c [*范围*]：仅显示行中指定范围的字符
- -d：指定字段的分隔符，默认的字段分隔符为“TAB”
- -f [*范围*]：显示指定第 *num* 个字段的内容，可以用逗号隔开显示多个字段

其中指定的字符或者范围有如下方式表示：

- N-：从第 N 个字节、字符、字段到结尾。
- N-M：从第 N 个字节、字符、字段开始到第 M 个（包括 M 在内）字节、字符、字段结束。
- -M：从第一个字节、字符、字段开始到第 M 个（包括 M 在内）字节、字符、字段结束。

常见用法如下：

- 显示/etc/passwd 文件以：间隔的第一列。

```
[root@host ~]# cut -d: -f1 /etc/passwd
```

- 显示/etc/passwd 文件中每行第 3 到第 8 个字符。

```
[root@host ~]# cut -c 3-8 /etc/passwd
```

提取列或字段 awk

awk 是一个强大的文本分析工具，简单来说 awk 就是把文件或者标准输入逐行读入，以空格为默认分隔符将每行切片，切开的部分再进行各种分析处理。语法如下：

```
awk ACTION [FILE]
```

常见用法如下：

- 显示最近登录系统的 5 个账号。

```
[root@host ~]# last -n 5 | awk ‘{print $1}’
#其中$1 是变量名称，表示的是打印对象为第一栏的内容，$2、$3 为第二栏、第三栏内容，以此类推。若变量为$0，则表示的是打印一整行数据。
```

提取关键字 grep

grep 命令是一种强大的文本搜索工具，它能使用正则表达式搜索文本，并把匹配的行打印出来。grep 在一个或多个文件中搜索字符串模板。如果模板包括空格，则必须被引用，模板后的所有字符串被看作文件名。搜索的结果被送到标准输出，不影响原文件内容。语法如下：

grep [OPTIONS] [FILE]

- -c：统计有符合字符样式的行数
- -i：忽略大小写
- -w：只显示全字符合的行
- -x：只显示全行符合的行

常见用法如下：

- 显示/etc/passwd 文件中带有 openEuler 的行内容。

```
[root@host ~]# cat /etc/passwd | grep openEuler
```

3.2.2.4 文本排序与比较

文本统计 wc

wc 命令用于计算字数。利用 wc 指令我们可以计算文件的字节数、字数、或是列数，若不指定文件名称、或是所给予的文件名为"-"，则 wc 指令会从标准输入设备读取数据。语法如下：

wc [OPTIONS] [FILE]

- -c：显示统计的字节数
- -l：显示统计的行数
- -w：显示统计的字数（英文字母）

常见用法如下：

- 获取/etc/passwd 文件的字数、行数和字节数。

```
[root@host ~]# cat /etc/passwd | wc -wlc
```

文本排序 sort

sort 命令可以将文件进行排序，并将排序结果标准输出。sort 命令既可以从特定的文件，也可以从 stdin 中获取输入。语法如下：

sort [OPTIONS] [FILE]

- -b：忽略每行前面开始的空格字符
- -c：检查文件是否已经按照顺序排序
- -d：排序时，处理英文字母、数字及空格字符外，忽略其他字符
- -f：排序时，将小写字母视为大写字母
- -n：依照数值的大小排序
- -r：以相反的顺序排序
- -o <文件>：将排序后的结果存入指定的文件
- -u：忽略相同行

文本比较 diff

diff 以逐行的方式，比较文本文件的异同处。如果指定要比较目录，则 diff 会比较目录中相同文件名的文件，但不会比较其中子目录。语法如下：

```
diff [OPTIONS] FILE1_or_DIRECTORY1 FILE2_or_DIRECTORY2
```

- -B：忽略空白行的差异
- -b：忽略一行中多个空白的差异
- -c：显示全部内容，并标出不同之处
- -i：忽略大小写的不同
- -r：比较子目录中的文件
- -w：忽略全部的空格字符

常见用法如下：

- 比较文件 f1 与文件 f2 的区别，仅列出不同的文本。

```
[root@host ~]# diff f1 f2
#会列出 f1 与 f2 不同的文本，并且有“---”符号分隔，之上是 f1 中不同的内容，之下是 f2 中不同的内容。
```

- 比较文件 f1 与文件 f2 的区别，显示全部内容，并标出不同之处。

```
[root@host ~]# diff -c f1 f2
#会列出 f1 与 f2 的全部内容，并且有“---”符号上方是 f1 的所有内容，下方是 f2 的所有内容，用“！”表示不同的行。
```

- 比较子目录 d1 与子目录 d2 的中文件的区别。

```
[root@host ~]# diff -r d1 d2
```

3.2.2.5 文本操作工具

tr

tr 指令从标准输入设备读取数据，经过字符串转译后，将结果输出到标准输出设备，常用于转换或删除文件中的字符。语法如下：

```
tr [OPTIONS] SET1 [SET2]
```

- -c：反选设定字符，也就是符合 set1 的部分不做处理，不符合的剩余部分才进行转换
- -d：删除字符
- -s 缩减连续重复的字符成指定的单个字符
- -t：削减 set1 指定范围，使之与 set2 设定长度相等

常见用法如下：

- 将 text.txt 文件中小写转换为大写输出。

```
[root@host ~]# cat text.txt | tr a-z A-Z
```

sed

相比较 tr，sed 可以修改字符串。sed 是一种在线编辑器，可以对来自文件、以及标准输入的文本进行编辑。执行时，sed 会从文件或者标准输入中读取一行，将其复制到缓冲区，对文本编辑完成之后，读

取下一行直到所有的文本行都编辑完毕。所以 sed 命令处理时只会改变缓冲区中文本的副本，如果想要直接编辑原文件，可以使用-i 选项或者将结果重定向到新的文件中。语法如下：

```
sed [OPTIONS] {ACTION} [INPUT_FILE]
```

- -n：取消默认输出
- -e：多点编辑，可以执行多个子命令
- -f：从脚本文件中读取命令
- -i：直接编辑原文件
- -l：指定行的长度
- -r：在脚本中使用扩展表达式

常见用法如下：

- 显示/etc/passwd 文件的内容，但不包括第 2 到第 5 行。

```
[root@host ~]# cat /etc/passwd | sed '2,5d'
#这里的'2,5d'就是删除 2 到 5 行的意思，d 表示删除。
```

- 显示/etc/passwd 文件的内容，但不包括第 10 行以后的内容。

```
[root@host ~]# cat /etc/passwd | sed '11,$d'
#这里的'11,$d'就是删除 11 到最后一行的意思。
```

- 显示/etc/passwd 文件的内容，但在第 2 行后，也就是第 3 行加入 openEuler。

```
[root@host ~]# cat /etc/passwd | sed '2a openEuler'
#这里的 a 就是 add，在后面添加的意思。
```

- 显示/etc/passwd 文件的内容，但在第 2 行加入 openEuler，并同时编辑原文件。

```
[root@host ~]# cat /etc/passwd | sed -i '2i openEuler'
#这里的 i 是 insert，也就是插入的意思，第一个 i 即是选项参数，意思是插入内容到原文件，第二个 i 是指在第二行插入 openEuler。
```

- 显示/etc/passwd 文件的第 10 到 20 行内容。

```
[root@host ~]# cat /etc/passwd | sed -n '10,20p'
```

- 显示/etc/passwd 文件的内容，替换 10 到 20 行内容为 openEuler。

```
[root@host ~]# cat /etc/passwd | sed '10,20c openEuler'
```

更多关于文件查看的练习请参考《HCIA-openEuler 实验手册-PC 版》2.3 文件查看。

3.3 思考题

- vim 的普通模式、插入模式和命令行模式都有哪些作用？如何进行切换？
- :wq 和:wq!的区别是什么？会在什么情况下使用？
- cut 和 grep 有哪些异同点？
- 通过 diff 找出差异时如何修补或升级文件？
- sed 与 head/tail、cut 效果相同的命令是什么？

4 用户和权限管理

在 Linux 中，每个用户都有一个账户，包括用户名、密码和主目录等信息。有些用户是用户管理员创建的，有些用户是系统创建的特殊用户，它们具有特殊的意义，其中最重要的是管理员账户，默认用户名是 root。为了方便多账号权限的管理，Linux 设计了另外一个概念：用户组，每一个用户至少属于一个组，从而实现相同权限用户的快速管理。

用户和用户组管理是系统安全管理的重要组成部分，本章主要介绍 openEuler 提供的用户管理和组管理命令，以及为普通用户分配特权的方法。

4.1 用户和用户组管理

4.1.1 用户管理

openEuler 系统中用户账号的管理，主要涉及到用户账号的添加、修改和删除。

添加用户账号就是在系统中创建一个新账号，然后为新账号分配用户号、用户组、主目录和登录 Shell 等资源。

4.1.1.1 增加用户

useradd 命令

在 root 权限下，通过 useradd 命令可以为系统添加新用户信息，其中 options 为相关参数，username 为用户名称。

```
useradd [options] username
```

例如新建一个用户名为 openEuler 的用户，在 root 权限下执行如下命令：

```
[root@host ~] # useradd openEuler
```

没有任何提示，表明用户建立成功。这时并没有设置用户的口令，请使用 passwd 命令修改用户的密码，没有设置密码的新账号不能登录系统。

使用 id 命令查看新建的用户信息，命令如下：

```
[root@host ~]# id openEuler
uid=1000(openEuler) gid=1000(openEuler) groups=1000(openEuler)
```

修改用户 openEuler 的密码：

```
[root@host ~]# passwd openEuler
```

修改用户密码时需要满足密码复杂度要求，密码的复杂度的要求如下：

- 口令长度至少 8 个字符。
- 口令至少包含大写字母、小写字母、数字和特殊字符中的任意 3 种。
- 口令不能和账号一样。
- 口令不能使用字典词汇。

根据提示两次输入新用户的密码，完成密码更改。过程如下：

```
[root@host ~]# passwd openEuler
Changing password for user openEuler.
New password:
Retype new password:
passwd: all authentication tokens updated successfully.
```

- 若打印信息中出现“BAD PASSWORD: The password fails the dictionary check - it is too simplistic/sytematic”，表示设置的密码过于简单，建议设置复杂度较高的密码。
- 说明：查询字典 在已装好的 openEuler 环境中，可以通过如下命令导出字典库文件 dictionary.txt，用户可以查询密码是否在该字典中。

```
[root@host ~]#cracklib-unpacker /usr/share/cracklib/pw_dict > dictionary.txt
```

4.1.1.2 修改用户账号

修改用户账号主要涉及用户账号密码、用户 shell、主目录、用户 UID、账号的有效期等属性的修改。

修改密码

普通用户可以用 passwd 修改自己的密码，只有管理员才能用 passwd username 为其他用户修改密码。

修改用户 shell 设置

使用 chsh 命令可以修改自己的 shell，只有管理员才能用 chsh username 为其他用户修改 shell 设置。

用户也可以使用 usermod 命令修改 shell 信息，在 root 权限下执行如下命令，其中 new_shell_path 为目标 shell 路径，username 为要修改用户的用户名，请根据实际情况修改：

```
usermod -s new_shell_path username
```

例如，将用户 openEuler 的 shell 改为 csh，命令如下：

```
[root@host ~]# usermod -s /bin/csh openEuler
```

修改主目录

修改主目录，可以在 root 权限下执行如下命令，其中 new_home_directory 为已创建的目标主目录的路径，username 为要修改用户的用户名，请根据实际情况修改：

```
usermod -d new_home_directory username
```

如果想将现有主目录的内容转移到新的目录，应该使用-m 选项，命令如下：

```
usermod -d new_home_directory -m username
```

修改 UID

修改用户 ID，在 root 权限下执行如下命令，其中 UID 代表目标用户 ID，username 代表用户名，请根据实际情况修改：

```
usermod -u UID username
```

该用户主目录中所拥有的文件和目录都将自动修改 UID 设置。但是，对于主目录外所拥有的文件，只能使用 chown 命令手动修改所有权。

修改账号的有效期

如果使用了影子口令，则可以在 root 权限下，执行如下命令来修改一个账号的有效期，其中 MM 代表月份，DD 代表某天，YY 代表年份，username 代表用户名，请根据实际情况修改：

```
usermod -e MM/DD/YY username
```

4.1.1.3 删除用户

在 root 权限下，使用 userdel 命令可删除现有用户。

例如，删除用户 Test，命令如下：

```
[root@host ~]# userdel Test
```

如果想同时删除该用户的主目录以及其中所有内容，要使用-r 参数递归删除。

- **说明**：不建议直接删除已经进入系统的用户，如果需要强制删除，请使用 userdel -f Test 命令。

4.1.1.4 管理员账户授权

使用 sudo 命令可以允许普通用户执行管理员账户才能执行的命令。

sudo 命令允许已经在/etc/sudoers 文件中指定的用户运行管理员账户命令。例如，一个已经获得许可的普通用户可以运行如下命令：

```
[root@host ~]#sudo /usr/sbin/useradd newuserl
```

实际上，sudo 的配置完全可以指定某个已经列入/etc/sudoers 文件的普通用户可以做什么，不可以做什么。

/etc/sudoers 的配置行如下所示。

```
[root@host ~]# cat /etc/sudoers
...
```

```
Defaults    !visiblepw
Defaults    env_reset
Defaults    env_keep =  "COLORS DISPLAY HOSTNAME HISTSIZE KDEDIR LS_COLORS"
Defaults    env_keep += "MAIL PS1 PS2 QTDIR USERNAME LANG LC_ADDRESS LC_CTYPE"
Defaults    env_keep += "LC_COLLATE LC_IDENTIFICATION LC_MEASUREMENT LC_MESSAGES"
Defaults    env_keep += "LC_MONETARY LC_NAME LC_NUMERIC LC_PAPER LC_TELEPHONE"
Defaults    env_keep += "LC_TIME LC_ALL LANGUAGE LINGUAS _XKB_CHARSET XAUTHORITY"
Defaults    secure_path = /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
root    ALL=(ALL)       ALL
## Allows people in group wheel to run all commands
%wheel  ALL=(ALL)       ALL
```

- 说明：空行或注释行（以#字符打头），无具体功能的行。

4.1.1.5 用户信息文件

与用户账号信息有关的文件如下：

- /etc/passwd：用户账号信息文件。
- /etc/shadow：用户账号信息加密文件。
- /etc/group：组信息文件。
- /etc/default/useradd：定义默认设置文件。
- /etc/login.defs：系统广义设置文件。
- /etc/skel：默认的初始配置文件目录。

4.1.2 用户组管理

openEuler 系统中每个用户都有一个用户组，管理员可以对一个用户组中的所有用户进行集中管理。openEuler 系统中用户组的管理，主要涉及到用户组的添加、修改和删除。

4.1.2.1 管理用户组

增加用户组

在 root 权限下，通过 groupadd 命令可以为系统添加新用户组信息，其中 options 为相关参数，groupname 为用户组名称。

```
groupadd [options] groupname
```

创建用户组实例

例如新建一个用户组名为 groupexample 的用户，在 root 权限下执行如下命令：

```
[root@host ~] # groupadd groupexample
```

修改 GID

修改用户组 ID，在 root 权限下执行如下命令，其中 GID 代表目标用户组 ID， groupname 代表用户组，请根据实际情况修改：

```
groupmod -g GID groupname
```

修改用户组名

修改用户组名，在 root 权限下执行如下命令，其中 newgroupname 代表新用户组名，oldgroupname 代表已经存在的待修改的用户组名，请根据实际情况修改：

```
[root@host ~] groupmod -n newgroupname oldgroupname
```

删除用户组

在 root 权限下，使用 groupdel 命令可删除用户组。

例如，删除用户组 Test，命令如下：

```
[root@host ~] # groupdel Test
```

- 说明：groupdel 不能直接删除用户的主组，如果需要强制删除用户主组，请使用 groupdel -f Test 命令。

将用户加入用户组或从用户组中移除

在 root 权限下，使用 gpasswd 命令将用户加入用户组或从用户组中移除。

例如，将用户 openEuler 加入用户组 Test ，命令如下：

```
 [root@host ~]# gpasswd -a openEuler Test
Adding user openEuler to group Test
```

例如，将用户 openEuler 从 Test 用户组中移除，命令如下：

```
[root@host ~]# gpasswd -d openEuler Test
Removing user openEuler from group Test
```

切换用户组

一个用户同时属于多个用户组时，则在用户登录后，使用 newgrp 命令可以切换到其他用户组，以便具有其他用户组的权限。

例如，将用户 openEuler 切换到 Test 用户组，命令如下：

```
[root@host ~]# newgrp Test
Welcome to 4.19.90-2003.4.0.0036.oe1.x86_64
System information as of time:   Mon Dec 14 14:36:45 CST 2020
System load:      0.00
Processes:        85
Memory used:      4.5%
Swap used:        0.0%
Usage On:         7%
IP address:       192.168.0.90
Users online:     1
```

4.1.2.2 用户组信息文件

与用户组信息有关的文件如下：

- /etc/gshadow：用户组信息加密文件。
- /etc/group：组信息文件。
- /etc/login.defs：系统广义设置文件。

4.2 文件权限管理

4.2.1 文件权限的基本概念

在 openEuler 操作系统中我们可以通过 ll 或者 ls －l 命令来显示一个文件或文件夹的属性以及文件所属的用户和组，如：

```
[root@host bin]# ls -l
total 97224
-rwxr-xr-x. 1 root root       55624 Mar 24   2020 '['
-rwxr-xr-x. 1 root root       39248 Mar 24   2020   addftinfo
-rwxr-xr-x. 1 root root       35488 Mar 24   2020   addr2line
```

上述操作实例中，addftinfo 文件的第一个属性使用-表示。-在 Linux 中代表该文件是一个普通文件。

在 openEuler 操作系统中第一个字符代表这个文件是目录、文件或链接文件等。

表4-1 文件类似说明

文件类型	解释说明
-	普通文件-除去其他六种类型文件
D	目录
B	块设备文件-可随机存取装置
C	字符设备文件-键盘、鼠标等一次性读取装置
L	符号链接文件-指向另一文件（link file）
P	命名管道文件（piep）
S	套接字文件（socket）

接下来的字符中，以三个为一组，且均为 rwx 的三个参数的组合。其中， r 代表可读(read)、 w 代表可写(write)、 x 代表可执行(execute)。 要注意的是，这三个权限的位置不会改变，如果没有权限，就会出现减号-。

- 读权限（Read）
  - 针对于文件，具有读取文件实际内容的权限
  - 针对于目录，具有读取目录结构列表的权限
- 写权限（Write）
  - 针对于文件，具有编辑、增加或修改文件内容的权限
  - 针对于目录，具有修改、删除或移动目录内文件的权限
- 执行权限（Execute）
  - 针对于文件，具有执行文件的权限
  - 针对于目录，具有进入目录的权限

很多时候为了操作方便会使用二进制代替对应的权限，例如：7 代表 rwx。

图4-1 文件权限

例如：属主权限：7、属组权限：5、其他用户权限：5

最终此目录文件的权限：755。

4.2.2 文件权限管理

4.2.2.1 文件权限概述

Linux 的文件调用权限分为三级：文件所有者、群组及其他，通过 chmod 命令可以控制文件被何人调用

使用权限：文件所有者

chmod：更改文件 9 个属性

openEuler 操作系统有两种设置文件属性的方法，一种是数字，一种是符号。

4.2.2.2 数字修改

openEuler 文件的基本权限就有九个，分别是属主/属组/其它用户，三种身份各有自己的读、写、可执行权限。

每种身份的三个权限可以根据二进制进行累加的，例如当权限为： -rwxrwx--- 分数则是：

- 属主：rwx = 4+2+1 = 7
- 属组： rwx = 4+2+1 = 7
- 其它用户：--- = 0+0+0 = 0

所以等一下我们设定权限的变更时，该文件的权限数字就是 770。变更权限的指令 chmod 的语法是这样的：

```
chmod [-R] xyz 文件或目录
```

选项与参数：

- xyz：就是刚刚提到的数字类型的权限属性，为 rwx 属性数值的相加。
- -R：进行递归(recursive)的持续变更，亦即连同次目录下的所有文件都会变更

举例来说，如果要将 .bashrc 这个文件所有的权限都设定启用，那么命令如下：

```
[root@host ~]# ls -al .bashrc
-rw-r--r--. 1 root root 176 Oct 29  2019 .bashrc
[root@host ~]# chmod 777 .bashrc
 [root@host ~]# ls -al .bashrc
-rwxrwxrwx. 1 root root 176 Oct 29  2019 .bashrc
```

那如果要将权限变成 -rwxr-xr-- 呢？那么权限的分数就成为 ：

```
[4+2+1][4+0+1][4+0+0]=754。
```

4.2.2.3 符号修改

之前的介绍中我们说到，文件有九个权限，分别属于不同用户或用户组是：

- user：属主
- group：属组
- others：其它用户

那么我们就可以使用 u、g、o 来代表三种身份的权限。

此外，a 则代表 all，即全部的身份。读写的权限可以写成 r,w,x，也就是可以使用下表的方式来看：

表4-2 chmod 参数说明

命令	用户/用户组	动作	权限	文件
chmod	u	+(加入)	r	文件或目录

	g o a	-(除去) =(设定)	w x	

现在我们需要将文件权限设置为 -rwxr-xr-- ，则可以使用命令：chmod u=rwx,g=rx,o=r 完成:

```
[root@host ~]# touch test1
[root@host ~]# ls -al test1
-rw------- 1 root root 0 Dec 14 14:43 test1
[root@host ~]# chmod u=rwx,g=rx,o=r   test1
[root@host ~]# ls -al test1
-rwxr-xr-- 1 root root 0 Dec 14 14:43 test1
```

而如果是要将某个权限去掉而不改变其他已存在的权限呢？例如去掉全部人的可执行权限：

```
[root@host ~]# chmod   a-x test1
[root@host ~]# ls -al test1
-rw-r--r-- 1 root root 0 Dec 14 14:43 test1
```

chown 命令：修改文件属主属组（只允许管理员）

- Linux 做为多用户多任务系统，所有文件都有其所有者，通过 chown 可以将特定文件的所有者更改为指定用户或组
- 使用权限：管理员（root 用户）

基本语法说明：

```
chown [–R] 属主名 文件名
chown [-R] 属主名：属组名 文件名
```

举例：

- 进入 /目录（ ）将 test.txt 的拥有者改为 bin 这个账号：

```
[root@host /]# cd /
[root@host /]# touch test.txt
[root@host /]# chown openEuler test.txt
[root@host /]# ls -l
total 64
-rw-------   1 openEuler root       0 Dec 14 14:47 test.txt
```

- 将 test.txt 的拥有者与群组改回为 root：

```
[root@host /]# chown root:root test.txt
[root@host /]# ls -l
total 64
dr-xr-xr-x  13 root root       0 Dec 14 14:47 sys
-rw-------   1 root root       0 Dec 14 14:49 test.txt
```

chgrp 命令：修改文件属组

通过 chgrp 命令可以对文件或目录的所属群组进行更改

使用权限：管理员（root 用户）

Linux chgrp 命令用于变更文件或目录的所属群组。

基本语法说明：

```
chgrp [-cfhRv][--help][--version][所属群组][文件或目录...]
```

或

```
chgrp [-cfhRv][--help][--reference=<参考文件或目录>][--version][文件或目录...]
```

参数说明：

- -c 或--changes 效果类似"-v"参数，但仅回报更改的部分。
- -f 或--quiet 或--silent 不显示错误信息。
- -h 或--no-dereference 只对符号连接的文件作修改，而不更动其他任何相关文件。
- -R 或--recursive 递归处理，将指定目录下的所有文件及子目录一并处理。
- -v 或--verbose 显示指令执行过程。
- --help 在线帮助。
- --reference=<参考文件或目录> 把指定文件或目录的所属群组全部设成和参考文件或目录的所属群组相同。
- --version 显示版本信息。

举例说明：

实例 1：

- 改变文件的群组属性：

```
[root@host /]# chgrp -v openEuler test.txt
changed group of 'test.txt' from root to openEuler
```

- "test.txt" 的所属组已更改为 bin

```
[root@host /]# ll
total 64K
-rw-------  1 root openEuler    0 Dec 14 14:49 test.txt
```

umask 命令：遮罩码

通过 umask 命令可以指定在建立文件时进行权限掩码的预设

使用权限：管理员和普通用户

语法

```
umask [-S][权限掩码]
```

参数说明：

- -S 以文字的方式来表示权限掩码。

举例说明：

- 使用指令"umask"查看当前权限掩码，则输入下面的命令：

```
[root@host /]# umask                                    #获取当前权限掩码
```

- 执行上面的指令后，输出信息如下：

```
0077
```

- 接下来，使用指令"mkdir"创建一个目录，并使用指令"ls"获取该目录的详细信息，输入命令如下：

```
[root@host /]# umask
0077
[root@host /]# mkdir test1
[root@host /]# ls -l -d /test1
```

- 执行上面的命令后，将显示新创建目录的详细信息，如下所示：

```
drwx------ 2 root root 4096 Dec 14 14:53 /test1
```

- 注意：在上面的输出信息中，"drwxr-xr-x"="777-022=755"。

4.2.3 文件的特殊权限

Linux 中，除了常见的读权限（read）、写权限（write）及执行权限（ execute ）以外，还具有三个特殊权限，即 Set UID、Set GID 以及 Sticky BIT

- Set UID：当 s 出现在文件所有者的 x 权限上时，就称之为 Set UID，简称为 SUID
- Set GID：当 s 出现在用户组的 x 权限上时，就称之为 Set UID，简称 SGID
- Sticky BIT：当 t 出现在目录的 x 权限上时，就称之为 Stcky BIT，简称 SBIT

set_uid ： 让普通用户临时拥有文件所有者的身份（前提是该文件为可执行的二进制文件）

Linux 下默认只有/usr/bin/passwd 才有“s”权限

```
[root@host /]#   ll /usr/bin/passwd
-rwsr-xr-x. 1 root root 31K Mar 24   2020 /usr/bin/passwd
```

实例应用 chmod u+s 可执行的二进制文件

```
[openEuler@host ~]$ ls /root
ls: 无法打开目录/root: 权限不够
[openEuler @host ~]# ls -ld /root
```

因为普通用户没有/root 的权限，所以无法查看

```
[root@host ~]# which ls
alias ls='ls --color=auto'
/usr/bin/ls
[root@host ~]# chmod u+s /usr/bin/ls
[openEuler@host ~]$ ls -l /root
```

当加上 s 权限后，openEuler 用户就临时拥有了 root 的权限

chmod u-s filename：去掉普通用户的“s”权限

chmod u=rws filename，同样可以设置，但是得到的会是 u=rwS "S"是没有“x”权限的

set_gid ： 作用于文件时，让普通用户临时拥有该文件所属组下用户的身份

```
[openEuler@host ~]$ ls -l /root
ls: 无法打开目录/root: 权限不够
[root@host ~]# chmod g+s /usr/bin/ls
[root@host ~]$ ls -l /root
总用量 4
```

set_gid 作用于目录时，任何用户在该目录下新建的目录或文件所属组跟该目录是一样的,如下例：

```
[root@host ~]# chmod g+s abc
[root@host ~]# chown :testx abc
[root@host ~]# mkdir abc/a
[root@host ~]# touch abc/a.txt
[root@host ~]# ls -l abc
总用量 0
 [root@host ~]# chmod g-s abc
[root@host ~]# touch abc/b.txt
[root@host ~]# mkdir abc/b
[root@host ~]# ls -l abc
总用量 0
```

特殊权限 stick_bit （防止其他用户删除自己的文件，root 用户除外）

语法：chmod o+t 文件/目录

```
[root@host /]# ls -ld /tmp
drwxrwxrwt 10 root root 240 Dec 14 14:28 /tmp
其中其他用户权限为“rwt”，其中“t” 表示防删除位.
```

4.2.4 文件 ACL 权限

4.2.4.1 文件 ACL 概述

常用权限的操作命令 chmod、chown、chgrp 及 umask 已经可以对文件权限进行修改，那么为什么还会出现访问控制列表 ACL(Access Control List)?

- 在没有 ACL 技术之前，Linux 系统对文件的权限控制仅可划分文件的属主、用户组、其他用户三类，随着技术的发展，传统的文件权限控制已经无法适应复杂场景下的权限控制需求，比如说一个部门（即一个用户组 group）存在有多名员工（即用户 user01、user02…），针对于部门内不同职责的员工，会为其赋予不同的权限，如为 user01 赋予可读写权限，为 user02 赋予只读权限，不为 user03 赋予任何权限，此时由于这些员工属于同一部门，就无法为这些不同的员工进行权限的细化。为此 ACL(Access Control List)访问控制列表技术应运而生，使用 ACL 权限控制可以提供常见权限（如 rwx、ugo）权限之外的权限设置，可以针对单一用户或组来设置特定的权限

常见类型：

- 针对文件所有者(owner)分配权限
- 针对文件所属用户组分配权限
- 针对其他用户分配
- Etc
- ...

4.2.4.2 文件 ACL 操作

在 linux 里我们可以通过 ACL 来管理某个文件及其特定的用户和用户组权限，简单来说 ACL 只需掌握三个命令即可：setfacl,getfacl,chacl

- getfacl：获取文件的 ACL

ACL 规则

setfacl 命令可以识别以下的规则格式：

- - [d[efault]:] [u[ser]:]uid [:perms] 指定用户的权限，文件所有者的权限（如果 uid 没有指定）。
- - [d[efault]:] g[roup]:gid [:perms] 指定群组的权限，文件所有群组的权限（如果 gid 未指定）
- - [d[efault]:] m[ask][:] [:perms] 有效权限掩码
- - [d[efault]:] o[ther] [:perms] 其他的权限

恰当的 acl 规则被用在修改和设定的操作中，对于 uid 和 gid，可以指定一个。 数字，也可指定一个名字。perms 域是一个代表各种权限的字母的组合：读-r 写-w 执行-x，执行只适合目录和一些可执行的文件。pers 域也可设置为八进制格式。

自动创建的规则：

最初的，文件目录仅包含 3 个基本的 acl 规则。为了使规则能正常执行，需要满足以下规则。

- - 3 个基本规则不能被删除。
- - 任何一条包含指定的用户名或群组名的规则必须包含有效的权限组合。
- - 任何一条包含缺省规则的规则在使用时，缺省规则必须存在。

ACL 的名词定义

ACL 是由一系列的 Access Entry 所组成的，每一条 Access Entry 定义了特定的类别可以对文件拥有的操作权限。

Access Entry 有三个组成部分：

- - Entry tag type
- - qualifier (optional)
- - permission。

Entry tag type 它有以下几个类型：

类型说明

- ACL_USER_OBJ： 相当于 Linux 里 file_owner 的 permission
- ACL_USER： 定义了额外的用户可以对此文件拥有的 permission
- ACL_GROUP_OBJ： 相当于 Linux 里 group 的 permission
- ACL_GROUP： 定义了额外的组可以对此文件拥有的 permission
- ACL_MASK： 定义了 ACL_USER,ACL_GROUP_OBJ 和 ACL_GROUP 的最大权限
- ACL_OTHER： 相当于 Linux 里 other 的 permission
- chacl：更改文件或目录的 ACL

与 chmod 相似，但是更为强大精细，通过 chmod 可以控制文件被何人调用，但若是某一用户的文件只想给特定的用户看时，则需要 chacl 出场完成用户的需求

操作示例：

```
[root@host ~]# touch /test
[root@host ~]# chmod 777 /test
[root@host ~]# getfacl /test                    //获得文件的 ACL 权限
getfacl: Removing leading '/' from absolute path names
# file: test                                    //文件名
# owner: root                                   //文件所属者
# group: root                                   //文件所属组
user::rwx                                       //文件所属者权限
group::rwx                                      //同组用户权限
other::rwx                                      //其它者权限
```

可以看到其它者的权限也是可读可写可执行，可以自行测试，现在我们修改其 ACL 策略，使用用户 code 只有读取的权限

```
[root@host ~]# setfacl -m u:code:r /test
[root@host ~]# ll /test
-rwxrwxrwx+ 1 root root 1 Nov 22 09:10 /test
[root@host ~]#
```

现在再次查看一下此文件的 ACL 属性

```
[root@host ~]# getfacl /test
getfacl: Removing leading '/' from absolute path names
# file: test
# owner: root
# group: root
user::rwx
user:code:r--
group::rwx
mask::rwx
other::rwx
```

code 的权限并不是只根据 ACL 配置来决定的，它是由 code 用户基本权限与配置的 ACL 权限的“与”运算决定的，即 other:rwx 与 code:r-- = code:r--

现在使用 code 用户，测试是否可写。

在写文件时，会出现

```
-- INSERT -- W10: Warning: Changing a readonly file
```

除了对单个用户进行设置外，还可以对用户组、有效权限(mask)进行设置如对用户组设置: g:[用户组]:[rwx]

如上面的/test 文件，已经有了可读权限，如果我们把它的有效权限修改为只有写权限，则设置的 acl 权限不在有效权限之内，则用户 code 就不可能再查看/test 文件中的内容了

```
[root@host ~]# setfacl -m m:w /test                    //设置有效权限为只写
```

- 可以查看/test acl 属性

```
[root@host ~]# getfacl /test
getfacl: Removing leading '/' from absolute path names
# file: test
# owner: root
# group: root
user::rwx
user:code:r-- #effective:---
group::rwx #effective:-w-
mask::-w-
other::rwx
[root@host ~]#
```

- 使用 code 用户查看文件内容，首先使用 root 用户写入一些内容，会使测试更加直观

```
[root@host ~]# echo "this is a test getfacl " >/test
[code@ localhost ~]$ vim /test
 "/test" [Permission Denied]
```

- 取消 acl 权限

```
[root@host ~]# setfacl -x u:code /test
[root@host ~]# setfacl -x m /test
[root@host ~]# getfacl /test
getfacl: Removing leading '/' from absolute path names
# file: test
# owner: root
# group: root
user::rwx
group::rwx
other::rwx
[root@host ~]# ll /test
-rwxrwxrwx 1 root root 24 Nov 22 11:13 /test                    //已经可以正常使用
```

4.3 其它权限管理

4.3.1 权限临时提升

Linux 中默认账户为普通用户，但是在更改系统文件或者执行某些命令时，都需要以 root 用户的权限才能进行，此时就需要将默认的普通用户更改为 root 用户

在切换用户身份时，常常用到的命令有三种：

- su：此命令在切换用户时，仅切换 root 用户身份，但 shell 环境仍为普通用户
- su -：此命令在切换用户时，用户身份和 shell 环境都会切换为 root 用户
- sudo：此命令可以允许普通用户执行管理员账户才能执行的命令

Linux su（英文全拼：swith user）命令用于变更为其他使用者的身份，除 root 外，需要键入该使用者的密码。

- 注意：openEuler 操作系统中，普通用户不支持切换至 root 用户。

使用权限：所有使用者。

语法说明：

```
su [-fmp] [-c command] [-s shell] [--help] [--version] [-] [USER [ARG]]
```

参数说明：

- -f 或 --fast 不必读启动档（如 csh.cshrc 等），仅用于 csh 或 tcsh
- -m -p 或 --preserve-environment 执行 su 时不改变环境变数
- -c command 或 --command=command 变更为帐号为 USER 的使用者并执行指令（command）后再变回原来使用者
- -s shell 或 --shell=shell 指定要执行的 shell （bash csh tcsh 等），预设值为 /etc/passwd 内的该使用者（USER） shell
- --help 显示说明文件
- --version 显示版本资讯
- --l 或 --login 这个参数加了之后，就好像是重新 login 为该使用者一样，大部份环境变数（HOME SHELL USER 等等）都是以该使用者（USER）为主，并且工作目录也会改变，如果没有指定 USER ，内定是 root
- USER 欲变更的使用者帐号
- ARG 传入新的 shell 参数

Linux sudo 命令以系统管理者的身份执行指令，也就是说，经由 sudo 所执行的指令就好像是 root 亲自执行。

使用权限：在 /etc/sudoers 中有出现的使用者。

语法说明：

```
sudo -V
sudo -h
sudo -l
sudo -v
sudo -k
sudo -s
sudo -H
sudo [ -b ] [ -p prompt ] [ -u username/#uid] -s
sudo command
```

参数说明：

- -V 显示版本编号
- -h 会显示版本编号及指令的使用方式说明
- -l 显示出自己（执行 sudo 的使用者）的权限
- -v 因为 sudo 在第一次执行时或是在 N 分钟内没有执行（N 预设为五）会问密码，这个参数是重新做一次确认，如果超过 N 分钟，也会问密码
- -k 将会强迫使用者在下一次执行 sudo 时问密码（不论有没有超过 N 分钟）
- -b 将要执行的指令放在背景执行

- -p prompt 可以更改问密码的提示语，其中 %u 会代换为使用者的帐号名称， %h 会显示主机名称
- -u username/#uid 不加此参数，代表要以 root 的身份执行指令，而加了此参数，可以以 username 的身份执行指令（ #uid 为该 username 的使用者号码）
- -s 执行环境变数中的 SHELL 所指定的 shell ，或是 /etc/passwd 里所指定的 shell
- -H 将环境变数中的 HOME （家目录）指定为要变更身份的使用者家目录（如不加 -u 参数就是系统管理者 root ）
- command 要以系统管理者身份（或以 -u 更改为其他人）执行的指令

举例说明：

- sudo 命令使用

```
$ sudo ls
[sudo] password for openEuler:
openEuler is not in the sudoers file. This incident will be reported.
```

- 指定用户执行命令

```
# sudo -u userb ls -l
```

- 显示 sudo 设置

```
$ sudo -L //显示 sudo 设置
Available options in a sudoers ``Defaults'' line:
syslog: Syslog facility if syslog is being used for logging
```

- 以 root 权限执行上一条命令

```
$ sudo
以特定用户身份进行编辑文本

$ sudo -u uggc vi ~www/index.html
//以 uggc 用户身份编辑 home 目录下 www 目录中的 index.html 文件
```

- 列出目前的权限

```
sudo -l
```

- 列出 sudo 的版本资讯

```
sudo –V
```

4.4 思考题

- 请创建一个用户组 it，要求 GID 为 1010；再创建一个组 mg，要求 GID 为 1020。
- 创建一个用户 user1，主组为其私有组，附属组为 it；创建一个用户 user2，主组为 mg，附属组为 it。
- 设置 user1 和 user2 的密码过期时间为 30 天，过期前 3 天提醒。
- 新建一个/tmp/test.txt 文件，要求 test 文件拥有人为 root，拥有组为 it。user1 能拥有该文件的读权限，user2 拥有该文件的读写权限。

5 软件安全和服务管理

5.1 软件包管理概述

Linux 软件包可分为两类：源码包、二进制包，不同的软件包有不同的提供方式，常用的方式有 rpm 包和 tgz 包两种。为此，常用应用软件的安装方式也有两种，一种为使用 rpm 工具安装，一种为编译安装。

rpm 作为标准的软件包管理工具，具有便捷的安装方式，是安装软件的首选方式。

openEuler、RedHat、SUSE、CentOS 等不同的分发版本同样使用 rpm 来对软件包进行管理。

不同的平台使得软件包的打包格式及工具不尽相同，其中 Debin 和 Ubuntu 采用的是 Deb 包安装以及 apt-get 源安装的方式来对软件包进行管理， FreeBSD 则采用的是 Ports，.txz 的打包格式以及 make, pkg 工具。

5.2 RPM 软件包概述

RPM 是一种用于互联网下载包的打包和自动安装工具，会生成具有.RPM 扩展名的文件，可以用来管理应用程序的安装、卸载和维护

RPM 软件包命名格式：

- name-version-release.arch.rpm
- 软件名称-版本号-发行版号-处理器架构

RPM 软件包管理优点：

- 简单便捷，兼容版本
- 参数信息记录在数据库中，便于查询、升级或卸载软件时使用

RPM 软件包管理缺点：

- 安装环境需与打包环境一致
- 具有很强的依赖关系，卸载软件时需要对依赖性软件优先处理，否则会导致其他软件无法正常使用

5.2.1 RPM 命令常用参数

RPM 命令参数较为繁多，本书仅列出常见参数：

命令语法

```
rpm [OPTION...]
```

其中的命令选项说明如下：

- -i：指定安装的软件包
- -h：使用“#（hash）”符显示 rpm 详细的安装过程及进度
- -v：显示安装的详细过程
- -U：升级指定的软件包
- -q：查询系统是否已安装指定的软件包或查询指定 rpm 包内容信息
- -a：查看系统已安装的所有软件包
- -V：查询已安装的软件包的版本信息
- -c：显示所有配置文件
- -p：查询/校验一个软件包的文件
- –vh：显示安装进度；
- –qpl：列出 RPM 软件包内的文件信息；
- –qpi：列出 RPM 软件包的描述信息；
- –qf：查找指定文件属于哪个 RPM 软件包；
- –Va：校验所有的 RPM 软件包，查找丢失的文件；
- –qa: 查找相应文件，如 rpm -qa mysql

举例说明：

- 安装软件

```
[root@host]# rpm -hvi dejagnu-1.4.2-10.noarch.rpm
警告：dejagnu-1.4.2-10.noarch.rpm: V3 DSA 签名：NOKEY, key ID db42a60e
准备...
########################################### [100%]
```

- 显示软件安装信息

```
[root@host]# rpm -qi dejagnu-1.4.2-10.noarch.rpm
```

5.3 DNS 软件管理概述

很多 Linux 发行版本都是基于 Linux 系统的软件管理工具 yum 完成 rpm 包的管理，可以从指定的服务器自动下载 RPM 服务器并进行安装，yum 可以作为软件仓库对软件包进行管理，相当于一个“管家”，同时能够解决软件包间的依赖关系，提升了效率。

但是 yum 工具性能差、内存占用过多、依赖解析速度变慢等问题长期得不到解决，同时 yum 工具过度依赖 yum 源文件，若是源文件出现问题，yum 相关操作可能会失败，针对这种情况，DNF 工具应运而生，DNF 工具克服了 YUM 软件管理工具的一些瓶颈，提升了用户体验、内存占用、依赖分析及运行速度等方面的内容。

DNF 是一款 Linux 软件包管理工具，用于管理 RPM 软件包。DNF 可以查询软件包信息，从指定软件库获取软件包，自动处理依赖关系以安装或卸载软件包，以及更新系统到最新可用版本。

DNF 与 YUM 完全兼容，提供了 YUM 兼容的命令行以及为扩展和插件提供的 API。

使用 DNF 需要管理员权限，本章所有命令需要在管理员权限下执行。

5.3.1 DNF 配置文件

DNF 的主要配置文件是 /etc/dnf/dnf.conf，该文件包含两部分：

- “main”部分保存着 DNF 的全局设置。
- “repository”部分保存着软件源的设置，可以有一个或多个“repository”。

另外，在/etc/yum.repos.d 目录中保存着一个或多个 repo 源相关文件，它们也可以定义不同的“repository”。

所以 openEuler 软件源的配置一般有两种方式，一种是直接配置/etc/dnf/dnf.conf 文件中的“repository”部分，另外一种是在/etc/yum.repos.d 目录下增加.repo 文件。

配置 main 部分

/etc/dnf/dnf.conf 文件包含的“main”部分，配置示例如下：

```
[main]
gpgcheck=1
installonly_limit=3
clean_requirements_on_remove=True
best=True
```

表5-1 dnf.conf 配置项

参数	说明
cachedir	缓存目录，该目录用于存储 RPM 包和数据库文件。
keepcache	可选值是 1 和 0，表示是否要缓存已安装成功的那些 RPM 包及头文件，默认值为 0，即不缓存。
debuglevel	设置 dnf 生成的 debug 信息。取值范围：[0-10]，数值越大会输出越详细的 debug 信息。默认值为 2，设置为 0 表示不输出 debug 信息。
clean_requirements_on_remove	删除在 dnf remove 期间不再使用的依赖项，如果软件包是通过 DNF 安装的，而不是通过显式用户请求安装的，则只能通过 clean_requirements_on_remove 删除软件包，即它是作为依赖项引入的。 默认值为 True。
Best	升级包时，总是尝试安装其最高版本，如果最高版本无法安装，则提示无法安装的原因并停止安装。默认值为 True。
obsoletes	可选值 1 和 0，设置是否允许更新陈旧的 RPM 包。默认值为 1，表示允许更新。

gpgcheck	可选值 1 和 0，设置是否进行 gpg 校验。默认值为 1，表示需要进行校验。
plugins	可选值 1 和 0，表示启用或禁用 dnf 插件。默认值为 1，表示启用 dnf 插件。
installonly_limit	设置可以同时安装“installonlypkgs”指令列出包的数量。默认值为 3，不建议降低此值。

配置 repository 部分

repository 部分允许您定义定制化的 openEuler 软件源仓库，各个仓库的名称不能相同，否则会引起冲突。配置 repository 部分有两种方式，一种是直接配置/etc/dnf/dnf.conf 文件中的“repository”部分，另外一种是配置/etc/yum.repos.d 目录下的.repo 文件。

直接配置/etc/dnf/dnf.conf 文件中的“repository”部分

- 下面是[repository]部分的一个最小配置示例:

```
[repository]
name=repository_name
baseurl=repository_url
```

openEuler 提供在线的镜像源，地址：https://repo.openEuler.org/。以 openEuler 20.03 的 aarch64 版本为例。

表5-2 repository 参数说明

参数	说明
name=repository_name	软件仓库（ repository ）描述的字符串。
baseurl=repository_url	软件仓库（ repository ）的地址。 ● 使用 http 协议的网络位置：例如 http://path/to/repo ● 使用 ftp 协议的网络位置：例如 ftp://path/to/repo ● 本地位置：例如 file:///path/to/local/repo

配置实例:

- openEuler 提供了多种 repo 源供用户在线使用，各 repo 源含义可参考系统安装，以 AArch64 架构的 OS repo 源为例。使用 root 权限在 openEuler_aarch64.repo 文件中添加 openEuler repo 源，示例如下:

```
# vi /etc/yum.repos.d/openEuler_aarch64.repo
```

```
[OS]
name=openEuler-$releasever - OS
baseurl=https://repo.openEuler.org/openEuler-20.03-LTS/OS/$basearch/
enabled=1
gpgcheck=1
gpgkey=https://repo.openEuler.org/openEuler-20.03-LTS/OS/$basearch/RPM-GPG-KEY-openEuler

[update]
name=openEuler-$releasever - Update
baseurl=http://repo.openEuler.org/openEuler-20.03-LTS/update/$basearch/
enabled=1
gpgcheck=1
gpgkey=http://repo.openEuler.org/openEuler-20.03-LTS/update/$basearch/RPM-GPG-KEY-openEuler

[extras]
name=openEuler-$releasever - Extras
baseurl=http://repo.openEuler.org/openEuler-20.03-LTS/extras/$basearch/
enabled=0
gpgcheck=1
gpgkey=http://repo.openEuler.org/openEuler-20.03-LTS/extras/$basearch/RPM-GPG-KEY-openEuler
```

- 说明：
  - enabled 为是否启用该软件源仓库，可选值为 1 和 0。默认值为 1，表示启用该软件源仓库。
  - gpgkey 为验证签名用的公钥。

5.3.2 显示当前配置

要显示当前的配置信息：

```
dnf config-manager --dump
```

要显示相应软件源的配置，首先查询 repo id：

```
dnf repolist
```

然后执行如下命令，显示对应 id 的软件源配置，其中 repository 为查询得到的 repo id：

```
dnf config-manager --dump repository
```

您也可以使用一个全局正则表达式，来显示所有匹配部分的配置：

```
dnf config-manager --dump glob_expression
```

5.3.3 管理软件包

使用 dnf 能够让您方便的进行查询、安装、删除软件包等操作。

5.3.3.1 搜索软件包

您可以使用 rpm 包名称、缩写或者描述搜索需要的 RPM 包，使用命令如下：

```
dnf search term
```

示例如下：

```
[root@host /] $dnf search httpd
======================================== N/S matched: httpd
========================================
httpd.aarch64 : Apache HTTP Server
httpd-devel.aarch64 : Development interfaces for the Apache HTTP server
httpd-manual.noarch : Documentation for the Apache HTTP server
httpd-tools.aarch64  : Tools for use with the Apache HTTP Server
libmicrohttpd.aarch64  : Lightweight library for embedding a webserver in applications
mod_auth_mellon.aarch64  : A SAML 2.0 authentication module for the Apache Httpd Server
mod_dav_svn.aarch64  : Apache httpd module for Subversion server
```

5.3.3.2 列出软件包清单

要列出系统中所有已安装的以及可用的 RPM 包信息，使用命令如下：

```
dnf list all
```

要列出系统中特定的 RPM 包信息，使用命令如下：

```
dnf list glob_expression...
```

示例如下：

```
[root@host /]$ dnf list httpd
Available Packages
httpd.aarch64                2.4.34-8.h5.oe1              Local
```

5.3.3.3 显示 RPM 包信息

要显示一个或者多个 RPM 包信息，使用命令如下：

```
dnf info package_name...
```

例如搜索，命令如下：

```
[root@host /]$ dnf info httpd
Available Packages
Name         : httpd
Version      : 2.4.34
Release      : 8.h5.oe1
Arch         : aarch64
Size         : 1.2 M
Repo         : Local
Summary      : Apache HTTP Server
URL          : http://httpd.apache.org/
License      : ASL 2.0
Description : The Apache HTTP Server is a powerful, efficient, and extensible
            : web server.
```

5.3.3.4 安装 RPM 包

要安装一个软件包及其所有未安装的依赖，请在 root 权限下执行如下命令：

```
dnf install package_name
```

您也可以通过添加软件包名字同时安装多个软件包。配置文件/etc/dnf/dnf.conf 添加参数 strict=False，运行 dnf 命令参数添加--setopt=strict=0。请在 root 权限下执行如下命令：

```
dnf install package_name package_name... --setopt=strict=0
```

示例如下：

```
[root@host /]# dnf install httpd
```

- 说明：安装 RPM 包过程中，若出现安装失败，可参考安装时出现软件包冲突、文件冲突或缺少软件包导致安装失败。

5.3.3.5 下载软件包

使用 dnf 下载软件包，请在 root 权限下输入如下命令：

```
dnf download package_name
```

如果需要同时下载未安装的依赖，则加上--resolve，使用命令如下：

```
dnf download --resolve package_name
```

示例如下：

```
[root@host /]# dnf download --resolve httpd
```

5.3.3.6 删除软件包

要卸载软件包以及相关的依赖软件包，请在 root 权限下执行如下命令：

```
dnf remove package_name...
```

示例如下：

```
[root@host /]# dnf remove totem
```

5.3.3.7 管理软件包组

软件包集合是服务于一个共同的目的一组软件包，例如系统工具集等。使用 dnf 可以对软件包组进行安装/删除等操作，使相关操作更高效。

列出软件包组清单

使用 summary 参数，可以列出系统中所有已安装软件包组、可用的组，可用的环境组的数量，命令如下：

```
dnf groups summary
```

使用示例如下：

```
[root@host /]# dnf groups summary
```

```
Last metadata expiration check: 0:11:56 ago on Sat 17 Aug 2019 07:45:14 PM CST.
Available Groups: 8
```

要列出所有软件包组和它们的组 ID ，命令如下：

```
dnf group list
```

使用示例如下：

```
[root@host /]# dnf group list
Last metadata expiration check: 0:10:32 ago on Sat 17 Aug 2019 07:45:14 PM CST.
Available Environment Groups:
   Minimal Install
   Custom Operating System
   Server
Available Groups:
   Development Tools
   Graphical Administration Tools
   Headless Management
   Legacy UNIX Compatibility
   Network Servers
   Scientific Support
   Security Tools
   System Tools
```

显示软件包组信息

要列出包含在一个软件包组中必须安装的包和可选包，使用命令如下：

```
dnf group info glob_expression...
```

例如显示 Development Tools 信息，示例如下：

```
[root@host /]# dnf group info "Development Tools"
Last metadata expiration check: 0:14:54 ago on Wed 05 Jun 2019 08:38:02 PM CST.

Group: Development Tools
 Description: A basic development environment.
 Mandatory Packages:
   binutils
   glibc-devel
   make
   pkgconf
   pkgconf-m4
   pkgconf-pkg-config
   rpm-sign
 Optional Packages:
   expect
```

安装软件包组

每一个软件包组都有自己的名称以及相应的 ID（groupid），您可以使用软件包组名称或它的 ID 进行安装。

要安装一个软件包组，请在 root 权限下执行如下命令：

```
dnf group install group_name
dnf group install groupid
```

例如安装 Development Tools 相应的软件包组，命令如下：

```
[root@host /]# dnf group install "Development Tools"
[root@host /]# dnf group install development
```

删除软件包组

要卸载软件包组，您可以使用软件包组名称或它的 ID，在 root 权限下执行如下命令：

```
dnf group remove group_name
dnf group remove groupid
```

例如删除 Development Tools 相应的软件包组，命令如下：

```
[root@host /]# dnf group remove "Development Tools"
[root@host /]# dnf group remove development
```

检查并更新

dnf 可以检查您的系统中是否有软件包需要更新。您可以通过 dnf 列出需要更新的软件包，并可以选择一次性全部更新或者只对指定包进行更新。

检查更新

如果您需要显示当前系统可用的更新，使用命令如下：

```
dnf check-update
```

使用实例如下：

```
[root@host /]# dnf check-update
Last metadata expiration check: 0:02:10 ago on Sun 01 Sep 2019 11:28:07 PM   CST.

anaconda-core.aarch64          19.31.123-1.14                updates
anaconda-gui.aarch64           19.31.123-1.14                updates
anaconda-tui.aarch64           19.31.123-1.14                updates
anaconda-user-help.aarch64     19.31.123-1.14                updates
anaconda-widgets.aarch64       19.31.123-1.14                updates
bind-libs.aarch64              32:9.9.4-29.3                 updates
bind-libs-lite.aarch64         32:9.9.4-29.3                 updates
bind-license.noarch            32:9.9.4-29.3                 updates
bind-utils.aarch64             32:9.9.4-29.3                 updates
...
```

升级

如果您需要升级单个软件包，在 root 权限下执行如下命令：

```
dnf update package_name
```

例如升级 rpm 包，示例如下：

```
[root@host /]# dnf update anaconda-gui.aarch64
Last metadata expiration check: 0:02:10 ago on Sun 01 Sep 2019 11:30:27 PM   CST.
Dependencies Resolved
================================================================================
 Package                 Arch          Version                 Repository      Size
================================================================================
Updating:
 anaconda-gui            aarch64       19.31.123-1.14          updates         461 k
 anaconda-core           aarch64       19.31.123-1.14          updates         1.4 M
 anaconda-tui            aarch64       19.31.123-1.14          updates         274 k
 anaconda-user-help      aarch64       19.31.123-1.14          updates         315 k
 anaconda-widgets        aarch64       19.31.123-1.14          updates         748 k

Transaction Summary
================================================================================
Upgrade   5 Package

Total download size: 3.1 M
Is this ok [y/N]:
```

类似的，如果您需要升级软件包组，在 root 权限下执行如下命令：

```
dnf group update group_name
```

更新所有的包和它们的依赖

要更新所有的包和它们的依赖，在 root 权限下执行如下命令：

```
dnf update
```

5.4 源代码安装概述

通过源代码安装软件是除了 rpm 软件包安装外的又一种安装软件的方式。Linux 下许多软件是通过源码包方式发行的，相对于二进制软件包来说，源码包的移植性较好，仅需发布一份源码包，不同用户经过编译即可正常运行，但是其配置和编译过程较为繁琐

openEuler 中会优先选择 rpm 来进行软件安装，但也会存在需要使用源码安装的场景：

- rpm 软件包版本太旧，编译参数不适用于当前业务
- 欲安装的软件无现成 rpm 软件包可用
- rpm 软件包缺乏某些特性

- 优化编译参数，提升性能

源码安装优点：

- 编译过程可以根据自身需求设置参数进行软件安装，灵活性好
- 经过本机编译，使得源码安装的软件与本机兼容性最好

源码安装缺点：

- 配置及编译过程较为繁琐
- 可能由于安装软件过新或其他问题，导致没有对应的依赖包，软件升级较为复杂，得不偿失

5.4.1 源码安装流程

源码安装软件步骤

- 下载源码包并解压（校验包完整性）
- 查看 README 和 INSTALL 文件（记录了软件的安装方法及注意事项）
- 创建 Makefile 文件-通过执行 ./configure 脚本命令生成
- 编译-通过 make 命令将源码自动编译成二进制文件
- 安装软件-通过 make install 安装命令来将上步编译出来的二进制文件安装到对应的目录中去，默认的安装路径为/usr/local/，相应的配置文件位置为/usr/local/etc 或/usr/local/***/etc

使用实例：

- 下载源码包并解压（校验包完整性）：

```
wget https://www.python.org/ftp/python/3.7.7/Python-3.7.7.tgz
tar -zxvf Python-3.7.7.tgz
```

- 进入源码目录，查看 README 文件：

```
cat Python-3.7.7/README.rst
```

- 执行./configure 命令，生成 Makefile 文件：

```
./configure --prefix=/usr/local/Python
```

- 执行 Make 命令进行编译：

```
make/make clean
```

- 执行 make install 命令进行软件的安装：

```
make install
```

5.5 服务管理

5.5.1 服务管理概述

systemd 是在 Linux 下，与 SysV 和 LSB 初始化脚本兼容的系统和服务管理器。systemd 使用 socket 和 D-Bus 来开启服务，提供基于守护进程的按需启动策略，支持快照和系统状态恢复，维护挂载和自挂载点，实现了各服务间基于从属关系的一个更为精细的逻辑控制，拥有更高的并行性能。

systemd 开启和监督整个系统是基于 unit 的概念。unit 是由一个与配置文件对应的名字和类型组成的（例如：avahi.service unit 有一个具有相同名字的配置文件，是守护进程 Avahi 的一个封装单元）。

5.5.2 常见命令介绍

systemd 提供 systemctl 命令来运行、关闭、重启、显示、启用/禁用系统服务。systemd 提供 systemctl 命令与 sysvinit 命令的功能类似。当前版本中依然兼容 service 和 chkconfig 命令，但建议用 systemctl 进行系统服务管理。

表5-3 systemd 与 sysvinit 命令

sysvinit 命令	systemd 命令	备注
service network start	systemctl start network.service	用来启动一个服务（并不会重启现有的）。
service network stop	systemctl stop network.service	用来停止一个服务（并不会重启现有的）。
service network restart	systemctl restart network.service	用来停止并启动一个服务。
service network reload	systemctl reload network.service	当支持时，重新装载配置文件而不中断等待操作。
service network condrestart	systemctl condrestart network.service	如果服务正在运行那么重启它。
service network status	systemctl status network.service	检查服务的运行状态。
chkconfig network on	systemctl enable network.service	在下次启动时或满足其他触发条件时设置服务为启用。
chkconfig network off	systemctl disable network.service	在下次启动时或满足其他触发条件时设置服务为禁用。
chkconfig network	systemctl is-enabled network.service	用来检查一个服务在当前环境下被配置为启用还是禁用。
chkconfig --list	systemctl list-unit-files --type=service	输出在各个运行级别下服务的启用和禁用情况。
chkconfig network –list	ls /etc/systemd/system/*.wants/network.service	用来列出该服务在哪些运行级别下启用和禁用。
chkconfig network –add	systemctl daemon-reload	当您创建新服务文件或者变更设置时使用。

显示服务状态

如果您需要显示某个服务的状态，可执行如下命令：

```
systemctl status name.service
```

同样，如果您需要判断某个服务是否被启用，可执行如下命令：

```
systemctl is-enabled name.service
```

例如查看 gdm.service 服务状态，命令如下：

```
[root@host /]# systemctl status gdm.service
gdm.service - GNOME Display Manager     Loaded: loaded (/usr/lib/systemd/system/gdm.service;
enabled)     Active: active (running) since Thu 2013-10-17 17:31:23 CEST; 5min ago
 Main PID: 1029 (gdm)
    CGroup: /system.slice/gdm.service
                ├─1029 /usr/sbin/gdm
                ├─1037 /usr/libexec/gdm-simple-slave --display-id /org/gno...
                └─1047 /usr/bin/Xorg :0 -background none -verbose -auth /r...Oct 17 17:31:23 localhost
systemd[1]: Started GNOME Display Manager.
```

运行服务

如果您需要运行某个服务，请在 root 权限下执行如下命令：

```
systemctl start name.service
```

例如运行 httpd 服务，命令如下：

```
[root@host /]# systemctl start httpd.service
```

关闭服务

如果您需要关闭某个服务，请在 root 权限下执行如下命令：

```
systemctl stop name.service
```

例如关闭蓝牙服务，命令如下：

```
[root@host /]# systemctl stop bluetooth.service
```

重启服务

如果您需要重启某个服务，请在 root 权限下执行如下命令：

```
systemctl restart name.service
```

执行命令后，当前服务会被关闭，但马上重新启动。如果您指定的服务，当前处于关闭状态，执行命令后，服务也会被启动。

例如重启蓝牙服务，命令如下：

```
[root@host /]# systemctl restart bluetooth.service
```

启用服务

如果您需要在开机时启用某个服务，请在 root 权限下执行如下命令：

```
systemctl enable name.service
```

例如设置 httpd 服务开机时启动，命令如下：

```
[root@host /]# systemctl enable httpd.service
ln -s '/usr/lib/systemd/system/httpd.service' '/etc/systemd/system/multi-user.target.wants/httpd.service'
```

禁用服务

如果您需要在开机时禁用某个服务，请在 root 权限下执行如下命令：

```
systemctl disable name.service
```

例如在开机时禁用蓝牙服务启动，命令如下：

```
[root@host /]# systemctl disable bluetooth.service
Removed /etc/systemd/system/bluetooth.target.wants/bluetooth.service.
Removed /etc/systemd/system/dbus-org.bluez.service.
```

5.5.3 系统基本服务管理

systemd 通过 systemctl 命令可以对系统进行关机、重启、休眠等一系列操作。当前仍兼容部分 Linux 常用管理命令。建议用户使用 systemctl 命令进行操作。

表5-4 systemctl 与 Linux 系统命令

Linux 常用管理命令	systemctl 命令	描述
Halt	systemctl halt	关闭系统
Poweroff	systemctl poweroff	关闭电源
Reboot	systemctl reboot	重启

关闭系统

关闭系统并下电，在 root 权限下执行如下命令：

```
systemctl poweroff
```

关闭系统但不下电机器，在 root 权限下执行如下命令：

```
systemctl halt
```

执行上述命令会给当前所有的登录用户发送一条提示消息。如果不想让 systemd 发送该消息，您可以添加“--no-wall”参数。具体命令如下：

```
systemctl --no-wall poweroff
```

重启系统

重启系统，在 root 权限下执行如下命令：

```
systemctl reboot
```

执行上述命令会给当前所有的登录用户发送一条提示消息。如果不想让 systemd 发送该消息，您可以添加“--no-wall”参数。具体命令如下：

```
systemctl --no-wall reboot
```

使系统待机

使系统待机，在 root 权限下执行如下命令：

```
systemctl suspend
```

使系统休眠

使系统休眠，在 root 权限下执行如下命令：

```
systemctl hibernate
```

使系统待机且处于休眠状态，在 root 权限下执行如下命令：

```
systemctl hybrid-sleep
```

5.6 思考题

- 从华为云镜像站下载 nginx 源码，并编译安装 nginx。
- 配置主机防火墙服务，要求计算机重启后仍能通过网页访问 nginx 首页。

6 管理文件系统及存储

6.1 Linux 存储基础

在 LINUX 系统中有一个重要的概念：一切都是文件。 其实这是 UNIX 哲学的一个体现，而 Linux 是重写 UNIX 而来，所以这个概念也就传承了下来。在 UNIX 系统中，把一切资源都看作是文件，包括硬件设备。UNIX 系统把每个硬件都看成是一个文件，通常称为设备文件，这样用户就可以用读写文件的方式实现对硬件的访问。

LVM 是逻辑卷管理（Logical Volume Manager）的简称，它是 Linux 环境下对磁盘分区进行管理的一种机制。LVM 通过在硬盘和文件系统之间添加一个逻辑层，来为文件系统屏蔽下层硬盘分区布局，提高硬盘分区管理的灵活性。

下图为 Linux 中磁盘空间管理或者划分的全流程。

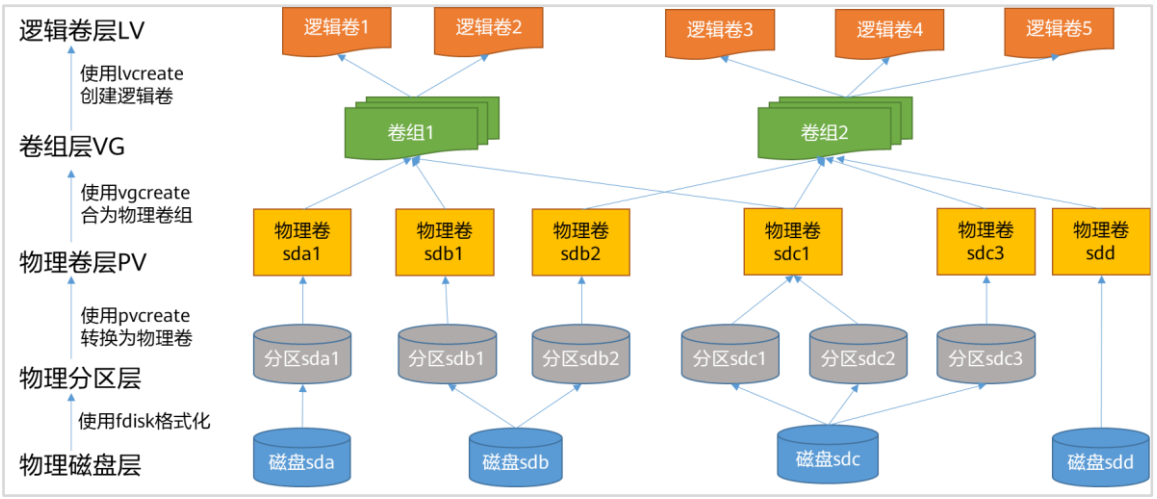


图6-1 磁盘空间管理流程

使用 LVM 管理硬盘的基本过程如下：

- 创建分区
- 创建为物理卷
- 将多个物理卷组合成卷组
- 在卷组中创建逻辑卷
- 在逻辑卷之上创建文件系统

通过 LVM 管理硬盘之后，文件系统不再受限于硬盘的大小，可以分布在多个硬盘上，也可以动态扩容。

基本概念

物理存储介质（The physical media）：指系统的物理存储设备，如硬盘，系统中为/dev/hda、/dev/sda 等等，是存储系统最低层的存储单元。

分区（Logical Volume）：linux 分区不同于 windows，linux 下硬盘设备名为（IDE 硬盘为 hdx（x 为从 a—d）因为 IDE 硬盘最多四个，SCSI，SATA，USB 硬盘为 sdx（x 为 a—z）），硬盘主分区最多为 4 个，所以主分区从 sdb1 开始到 sdb4，逻辑分区从 sdb5 开始。

物理卷（Physical Volume，PV）：指硬盘分区或从逻辑上与磁盘分区具有同样功能的设备(如 RAID)，是 LVM 的基本存储逻辑块。物理卷包括一个特殊的标签，该标签默认存放在第二个 512 字节扇区，但也可以将标签放在最开始的四个扇区之一。该标签包含物理卷的随机唯一识别符（UUID），记录块设备的大小和 LVM 元数据在设备中的存储位置。

卷组（Volume Group，VG）：由物理卷组成，屏蔽了底层物理卷细节。可在卷组上创建一个或多个逻辑卷且不用考虑具体的物理卷信息。

逻辑卷（Logical Volume，LV）：卷组不能直接用，需要划分成逻辑卷才能使用。逻辑卷可以格式化成不同的文件系统，挂载后直接使用。

物理块（Physical Extent，PE）：物理卷以大小相等的“块”为单位存储，块的大小与卷组中逻辑卷块的大小相同。

逻辑块（Logical Extent，LE）：逻辑卷以“块”为单位存储，在一卷组中的所有逻辑卷的块大小是相同的。

6.1.2 磁盘和分区概述

物理磁盘一般按照硬盘材质可以分为机械硬盘和固态硬盘两种类型，按照接口类型又可以分为 SCSI 接口的硬盘、SATA 接口的硬盘、SAS 接口的硬盘等类型。

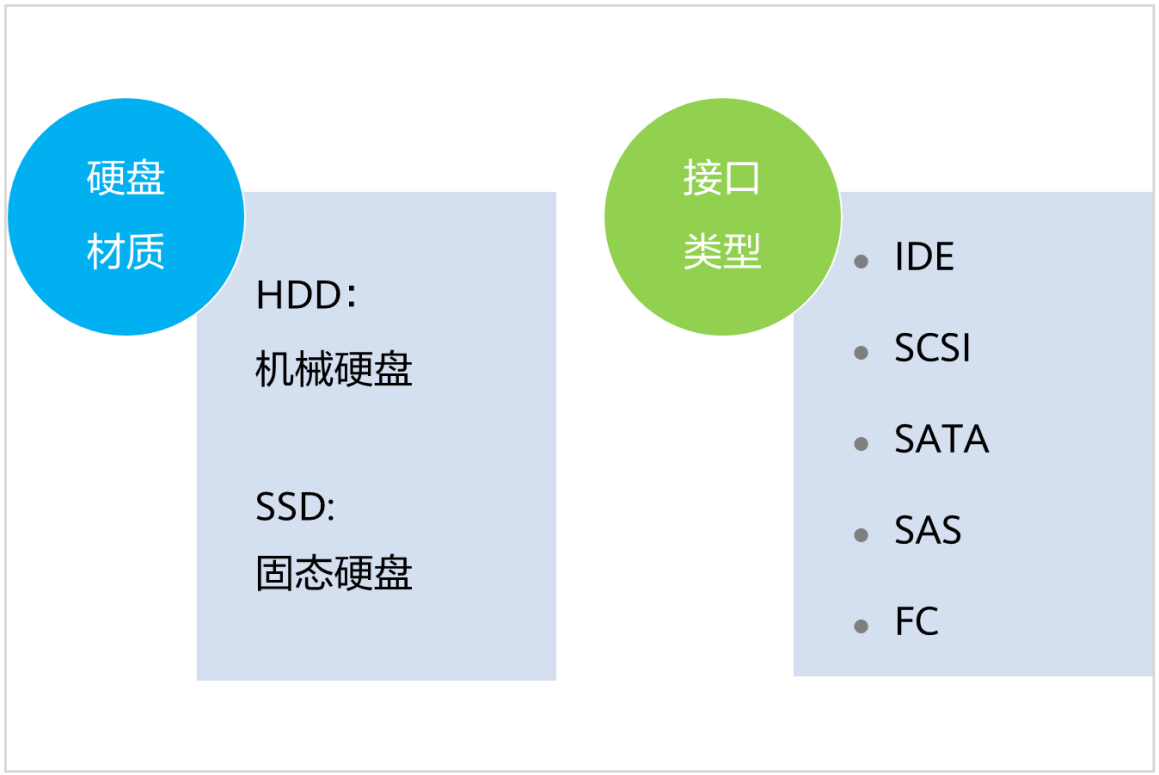


图6-2 常见的磁盘与磁盘接口

磁盘分区可以将硬盘驱动器划分为多个逻辑存储单元，这些单元称为分区。通过将磁盘划分为多个分区，系统管理员可以使用不同的分区执行不同功能。

磁盘分区的好处：

- 限制应用或用户的可用空间。
- 允许从同一磁盘进行不同操作系统的多重启动。
- 将操作系统和程序文件与用户文件分隔。
- 创建用于操作系统虚拟内存交换的单独区域。
- 限制磁盘空间使用情况,以提高诊断工具和备份映像的性能。

6.1.3 物理分区

在 Linux 中，一切设备皆文件，因此需要通过访问文件或文件目录实现对设备的访问，Linux 中磁盘设备名存放在/dev 目录中。

命名规则如下：

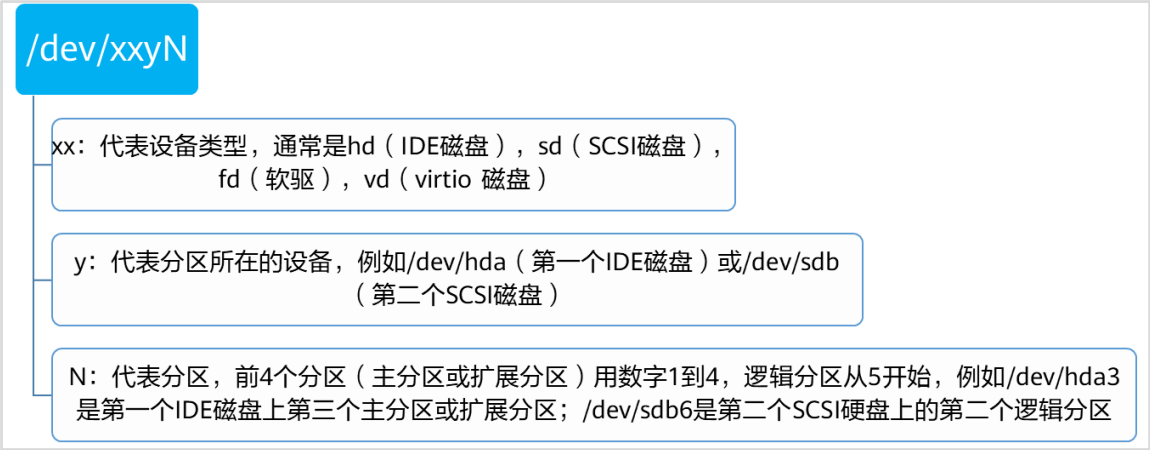


图6-3 磁盘命名

Linux 中，SSD、SAS、SATA 类型的硬盘，都用 sd 来标识，IDE 硬盘属于 IDE 接口类型的硬盘，用 hd 来标识。

通常所说的“硬盘分区”就是指修改磁盘分区表，注意以下情况：

- 考虑到磁盘的连续性，一般建议将扩展分区放在最后面的柱面内。
- 一个硬盘只有一个扩展分区，除去主分区，其它空间都分配给扩展分区。
- 硬盘容量=主分区+扩展分区；扩展分区容量=各个逻辑分区容量之和。

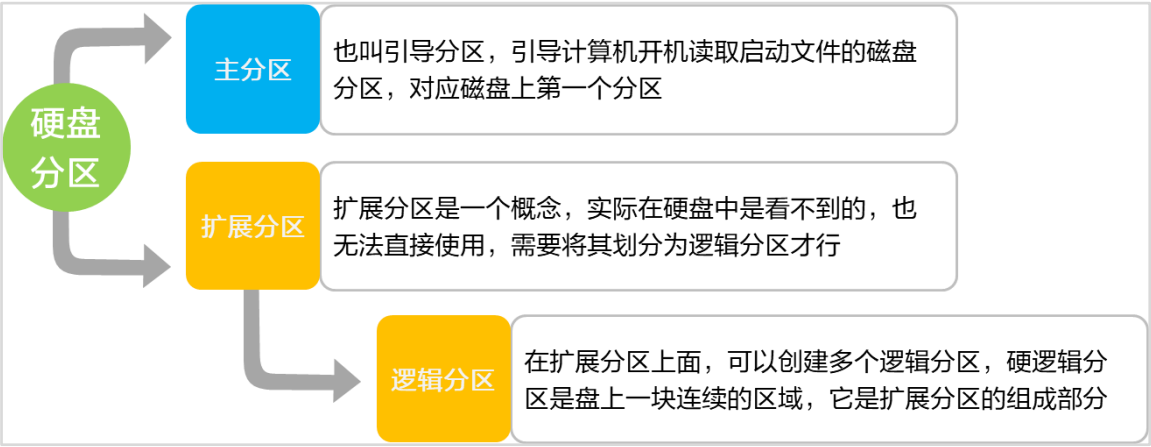


图6-4 MBR 分区类型

基本分区（primary partion）

基本分区也称主分区，引导分区、每块磁盘分区主分区与扩展分区加起来不能大于四个；

基本分区创建后可以立即使用，但是有分区数量上限。

扩充分区(extension partion)

每块磁盘内只能划分一块扩展分区；

扩展分区内可划分任意块逻辑分区；

扩展分区创建后不能直接使用，需要在扩展分区内创建逻辑分区。

逻辑分区（logical partion）

逻辑分区实在扩展分区内创建的分区；

逻辑分区相当与一块存储介质，和其他逻辑分区主分区完全独立。

6.2 逻辑卷层管理

6.2.1 LVM 的安装和确认

openEuler 操作系统默认已安装 LVM。可通过 rpm -qa | grep lvm2 命令查询，若打印信息中包含“lvm2”信息，则表示已安装 LVM；若无任何打印信息，则表示未安装，可参考以下内容进行安装。

- 在 root 权限下安装 LVM。

```
[root@host /]# dnf install lvm2
```

- 查看安装后的 rpm 包。

```
[root@host /]# rpm -qa | grep lvm2
```

6.2.2 创建分区

```
[root@host ~]# fdisk /dev/sdb
Command (m for help): n       # add a new partition
Partition type
   p    primary (0 primary, 0 extended, 4 free)
   e    extended (container for logical partitions)
Select (default p): p       # 主分区
Partition number (1-4, default 1):        # 回车
First sector (2048-10485759, default 2048):        # 回车
Last sector, +sectors or +size{K,M,G,T,P} (2048-10485759, default 10485759):        # 回车

Created a new partition 1 of type 'Linux' and of size 5 GiB.

Command (m for help): t       # 修改分区类型，在 CentOS8 中不用设置为 8e 也可以
Selected partition 1
Hex code (type L to list all codes): 8e       # LVM 类型
Changed type of partition 'Linux' to 'Linux LVM'.

Command (m for help): w       # 保存配置
The partition table has been altered.
```

```
Calling ioctl() to re-read partition table.
Syncing disks.

[root@host ~]# fdisk -l
Disk /dev/vda: 10 GiB, 10737418240 bytes, 20971520 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0x16a1e057

Device     Boot Start      End  Sectors Size Id Type
/dev/sda1  *     2048 20971486 20969439  10G 83 Linux


Disk /dev/sdb: 5 GiB, 5368709120 bytes, 10485760 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0x03864530

Device     Boot Start      End  Sectors Size Id Type
/dev/sdb1        2048 10485759 10483712   5G 8e Linux LVM
```

6.2.3 管理物理卷

6.2.3.1 创建物理卷

可在 root 权限下通过 pvcreate 命令创建物理卷。

```
pvcreate [option] devname ...
```

其中：

- option：命令参数选项。常用的参数选项有：
  - -f：强制创建物理卷，不需要用户确认。
  - -u：指定设备的 UUID。
  - -y：所有的问题都回答“yes”。
- devname：指定要创建的物理卷对应的设备名称，如果需要批量创建，可以填写多个设备名称，中间以空格间隔。

示例 1：将/dev/sdb、/dev/sdc 创建为物理卷。

```
[root@host /]# pvcreate /dev/sdb /dev/sdc
```

示例 2：将/dev/sdb1、/dev/sdb2 创建为物理卷。

```
[root@host /]# pvcreate /dev/sdb1 /dev/sdb2
```

6.2.3.2 查看物理卷

可在 root 权限通过 pvdisplay 命令查看物理卷的信息，包括：物理卷名称、所属的卷组、物理卷大小、PE 大小、总 PE 数、可用 PE 数、已分配的 PE 数和 UUID。

```
pvdisplay [option] devname
```

其中：

- option：命令参数选项。常用的参数选项有：
  - -s：以短格式输出。
  - -m：显示 PE 到 LE 的映射。
- devname：指定要查看的物理卷对应的设备名称。如果不指定物理卷名称，则显示所有物理卷的信息。

示例：显示物理卷/dev/sdb 的基本信息。

```
# pvdisplay /dev/sdb
```

6.2.3.3 修改物理卷属性

可在 root 权限下通过 pvchange 命令修改物理卷的属性。

```
pvchange [option] pvname ...
```

其中：

- option：命令参数选项。常用的参数选项有：
  - -u：生成新的 UUID。
  - -x：是否允许分配 PE”。
- pvname：指定要要修改属性的物理卷对应的设备名称，如果需要批量修改，可以填写多个设备名称，中间以空格间隔。

示例：禁止分配/dev/sdb 物理卷上的 PE。

```
# pvchange -x n /dev/sdb
```

6.2.3.4 删除物理卷

可在 root 权限下通过 pvremove 命令删除物理卷。

```
pvremove [option] pvname ...
```

其中：

- option：命令参数选项。常用的参数选项有：
  - -f：强制删除物理卷，不需要用户确认。
  - -y：所有的问题都回答“yes”。
- pvname：指定要删除的物理卷对应的设备名称，如果需要批量删除，可以填写多个设备名称，中间以空格间隔。

示例：删除物理卷/dev/sdb。

```
[root@host /]# pvremove /dev/sdb
```

6.2.4 管理卷组

6.2.4.1 创建卷组

可在 root 权限下通过 vgcreate 命令创建卷组。

```
vgcreate [option] vgname pvname ...
```

其中：

- option：命令参数选项。常用的参数选项有：
  - -l：卷组上允许创建的最大逻辑卷数。
  - -p：卷组中允许添加的最大物理卷数。
  - -s：卷组上的物理卷的 PE 大小。
- vgname：要创建的卷组名称。
- pvname：要加入到卷组中的物理卷名称。

示例：创建卷组 vg1，并且将物理卷/dev/sdb 和/dev/sdc 添加到卷组中。

```
[root@host /]# vgcreate vg1 /dev/sdb /dev/sdc
```

6.2.4.2 查看卷组

可在 root 权限下通过 vgdisplay 命令查看卷组的信息。

```
vgdisplay [option] [vgname]
```

其中：

- option：命令参数选项。常用的参数选项有：
  - -s：以短格式输出。
  - -A：仅显示活动卷组的属性。
- vgname：指定要查看的卷组名称。如果不指定卷组名称，则显示所有卷组的信息。

示例：显示卷组 vg1 的基本信息。

```
[root@host /]# vgdisplay vg1
```

6.2.4.3 修改卷组属性

可在 root 权限下通过 vgchange 命令修改卷组的属性。

```
vgchange [option] vgname
```

其中：

- option：命令参数选项。常用的参数选项有：
  - -a：设置卷组的活动状态。
- vgname：指定要修改属性的卷组名称。

示例：将卷组 vg1 状态修改为活动。

```
[root@host /]# vgchange -ay vg1
```

6.2.4.4 扩展卷组

可在 root 权限下通过 vgextend 命令动态扩展卷组。它通过向卷组中添加物理卷来增加卷组的容量。

```
vgextend [option] vgname pvname ...
```

其中：

- option：命令参数选项。常用的参数选项有：
  - -d：调试模式。
  - -t：仅测试。
- vgname：要扩展容量的卷组名称。
- pvname：要加入到卷组中的物理卷名称。

示例：将卷组 vg1 中添加物理卷/dev/sdb。

```
[root@host /]# vgextend vg1 /dev/sdb
```

6.2.4.5 收缩卷组

可在 root 权限下通过 vgreduce 命令删除卷组中的物理卷来减少卷组容量。不能删除卷组中剩余的最后一个物理卷。

```
vgreduce [option] vgname pvname ...
```

其中：

- option：命令参数选项。常用的参数选项有：
  - -a：如果命令行中没有指定要删除的物理卷，则删除所有的空物理卷。
  - --removemissing：删除卷组中丢失的物理卷，使卷组恢复正常状态。
- vgname：要收缩容量的卷组名称。
- pvname：要从卷组中删除的物理卷名称。

示例：从卷组 vg1 中移除物理卷/dev/sdb2。

```
[root@host /]# vgreduce vg1 /dev/sdb2
```

6.2.4.6 删除卷组

可在 root 权限下通过 vgremove 命令删除卷组。

```
vgremove [option] vgname
```

其中：

- option：命令参数选项。常用的参数选项有：
  - -f：强制删除卷组，不需要用户确认。
- vgname：指定要删除的卷组名称。

示例：删除卷组 vg1。

```
[root@host /]# vgremove vg1
```

6.2.5 管理逻辑卷

6.2.5.1 创建逻辑卷

可在 root 权限下通过 lvcreate 命令创建逻辑卷。

```
lvcreate [option] vgname
```

其中：

- option：命令参数选项。常用的参数选项有：
  - -L：指定逻辑卷的大小，单位为“kKmMgGtT”字节。
  - -l：指定逻辑卷的大小（LE 数）。
  - -n：指定要创建的逻辑卷名称。
  - -s：创建快照。
- vgname：要创建逻辑卷的卷组名称。

示例 1：在卷组 vg1 中创建 10G 大小的逻辑卷。

```
[root@host /]# lvcreate -L 10G vg1
```

示例 2：在卷组 vg1 中创建 200M 的逻辑卷，并命名为 lv1。

```
[root@host /]# lvcreate -L 200M -n lv1 vg1
```

6.2.5.2 查看逻辑卷

可在 root 权限下通过 lvdisplay 命令查看逻辑卷的信息，包括逻辑卷空间大小、读写状态和快照信息等属性。

```
lvdisplay [option] [lvname]
```

其中：

- option：命令参数选项。常用的参数选项有：
  - -v：显示 LE 到 PE 的映射
- lvname：指定要显示属性的逻辑卷对应的设备文件。如果省略，则显示所有的逻辑卷属性。

说明：逻辑卷对应的设备文件保存在卷组目录下，例如：在卷组 vg1 上创建一个逻辑卷 lv1，则此逻辑卷对应的设备文件为/dev/vg1/lv1。

示例：显示逻辑卷 lv1 的基本信息。

```
# lvdisplay /dev/vg1/lv1
```

6.2.5.3 调整逻辑卷大小

可在 root 权限下通过 lvresize 命令调整 LVM 逻辑卷的空间大小，可以增大空间和缩小空间。使用 lvresize 命令调整逻辑卷空间大小和缩小空间时需要谨慎，因为有可能导致数据丢失。

```
lvresize [option] vgname
```

其中：

- option：命令参数选项。常用的参数选项有：

  - -L：指定逻辑卷的大小，单位为“kKmMgGtT”字节。
  - -l：指定逻辑卷的大小（LE 数）。
  - -f：强制调整逻辑卷大小，不需要用户确认。
- lvname：指定要调整的逻辑卷名称。

示例 1：为逻辑卷/dev/vg1/lv1 增加 200M 空间。

```
# lvresize -L +200 /dev/vg1/lv1
```

示例 2：为逻辑卷/dev/vg1/lv1 减少 200M 空间。

```
# lvresize -L -200 /dev/vg1/lv1
```

6.2.5.4 扩展逻辑卷

可在 root 权限下通过 lvextend 命令动态在线扩展逻辑卷的空间大小，而不中断应用程序对逻辑卷的访问。

```
lvextend [option] lvname
```

其中：

- option：命令参数选项。常用的参数选项有：
  - -L：指定逻辑卷的大小，单位为“kKmMgGtT”字节。
  - -l：指定逻辑卷的大小（LE 数）。
  - -f：强制调整逻辑卷大小，不需要用户确认。
- lvname：指定要扩展空间的逻辑卷的设备文件。

示例：为逻辑卷/dev/vg1/lv1 增加 100M 空间。

```
[root@host /]# lvextend -L +100M /dev/vg1/lv1
```

6.2.5.5 收缩逻辑卷

可在 root 权限下通过 lvreduce 命令减少逻辑卷占用的空间大小。使用 lvreduce 命令收缩逻辑卷的空间大小有可能会删除逻辑卷上已有的数据，所以在操作前必须进行确认。

```
lvreduce [option] lvname
```

其中：

- option：命令参数选项。常用的参数选项有：
  - -L：指定逻辑卷的大小，单位为“kKmMgGtT”字节。
  - -l：指定逻辑卷的大小（LE 数）。
  - -f：强制调整逻辑卷大小，不需要用户确认。
- lvname：指定要扩展空间的逻辑卷的设备文件。

示例：将逻辑卷/dev/vg1/lv1 的空间减少 100M。

```
[root@host /]# lvreduce -L -100M /dev/vg1/lv1
```

6.2.5.6 删除逻辑卷

可在 root 权限下通过 lvremove 命令删除逻辑卷。如果逻辑卷已经使用 mount 命令加载，则不能使用 lvremove 命令删除。必须使用 umount 命令卸载后，逻辑卷方可被删除。

```
lvremove [option] vgname
```

其中：

- option：命令参数选项。常用的参数选项有：
  - -f：强制删除逻辑卷，不需要用户确认。
- vgname：指定要删除的逻辑卷。

示例：删除逻辑卷/dev/vg1/lv1。

```
[root@host /]# lvremove /dev/vg1/lv1
```

6.2.6 创建并挂载文件系统

在创建完逻辑卷之后，需要在逻辑卷之上创建文件系统并挂载文件系统到相应目录下。

6.2.6.1 创建文件系统

可在 root 权限下通过 mkfs 命令创建文件系统。

```
mkfs [option] lvname
```

其中：

- option：命令参数选项。常用的参数选项有：
  - -t：指定创建的 linux 系统类型，如 ext2，ext3，ext4 等等，默认类型为 ext2。
- lvname：指定要创建的文件系统对应的逻辑卷设备文件名。

示例：在逻辑卷/dev/vg1/lv1 上创建 ext4 文件系统。

```
[root@host /]# mkfs -t ext4 /dev/vg1/lv1
```

6.2.6.2 手动挂载文件系统

手动挂载的文件系统仅在当时有效，一旦操作系统重启则会不存在。可在 root 权限下通过 mount 命令挂载文件系统。

```
mount lvname mntpath
```

其中：

- lvname：指定要挂载文件系统的逻辑卷设备文件名。
- mntpath：挂载路径。

示例：将逻辑卷/dev/vg1/lv1 挂载到/mnt/data 目录。

```
[root@host /]# mount /dev/vg1/lv1 /mnt/data
```

6.2.6.3 自动挂载文件系统

手动挂载的文件系统在操作系统重启之后会不存在，需要重新手动挂载文件系统。但若在手动挂载文件系统后在 root 权限下进行如下设置，可以实现操作系统重启后文件系统自动挂载文件系统。

执行 blkid 命令查询逻辑卷的 UUID，逻辑卷以/dev/vg1/lv1 为例。

```
[root@host /]# blkid /dev/vg1/lv1
```

查看打印信息，打印信息中包含如下内容，其中 uuidnumber 是一串数字，为 UUID， fstype 为文件系统。

```
/dev/vg1/lv1: UUID=" uuidnumber " TYPE=" fstype "
```

执行 vi /etc/fstab 命令编辑 fstab 文件，并在最后加上如下内容。

```
UUID=uuidnumber   mntpath                    fstype     defaults          0 0
```

内容说明如下：

- 第一列：UUID，此处填写 1 查询的 uuidnumber 。
- 第二列：文件系统的挂载目录 mntpath 。
- 第三列：文件系统的文件格式，此处填写 1 查询的 fstype 。
- 第四列：挂载选项，此处以“defaults”为例；
- 第五列：备份选项，设置为“1”时，系统自动对该文件系统进行备份；设置为“0”时，不进行备份。此处以“0”为例；
- 第六列：扫描选项，设置为“1”时，系统在启动时自动对该文件系统进行扫描；设置为“0”时，不进行扫描。此处以“0”为例。

验证自动挂载功能。

- 执行 umount 命令卸载文件系统，逻辑卷以/dev/vg1/lv1 为例。

```
[root@host /]# umount /dev/vg1/lv1
```

- 执行如下命令，将/etc/fstab 文件所有内容重新加载。

```
[root@host /]# mount -a
```

- 执行如下命令，查询文件系统挂载信息，挂载目录以/mnt/data 为例。

```
[root@host /]# mount | grep /mnt/data
```

- 查看打印信息，若信息中包含如下信息表示自动挂载功能生效。

```
/dev/vg1/lv1 on /mnt/data
```

6.3 思考题

- 逻辑卷需要考虑安全性，请创建一个 LVM，使得其允许底层一个硬盘损坏的情况下，数据不丢失。
- 请配置一个 LVM，要求该 LVM 包含 100 个 LE，每个 PE 大小为 8M。

7 系统管理

7.1 任务管理

在 Linux 系统运维时，我们经常需要配置一些任务，以便于定时或周期性执行某种动作。这便是本章要讲的任务。

7.1.1 单次任务 at

计划任务 at 介绍

at 允许使用一套相当复杂的指定时间的方法。它能够接受在当天的 hh:mm（小时:分钟）式的时间指定。假如该时间已过去，那么就放在第二天执行。当然也能够使用 midnight（深夜），noon（中午），teatime（下午茶时间，一般是下午 4 点）等比较模糊的 词语来指定时间。用户还能够采用 12 小时计时制，即在时间后面加上 AM（上午）或 PM（下午）来说明是上午还是下午。 也能够指定命令执行的具体日期，指定格式为 month day（月 日）或 mm/dd/yy（月/日/年）或 dd.mm.yy（日.月.年）。指定的日期必须跟在指定时间的后面。

上面介绍的都是绝对计时法，其实还能够使用相对计时法，这对于安排不久就要执行的命令是很有好处的。指定格式为：now + count time-units，now 就是当前时间，time-units 是时间单位，这里能够是 minutes（分钟）、hours（小时）、days（天）、weeks（星期）。count 是时间的数量，究竟是几天，还是几小时，等等。

更有一种计时方法就是直接使用 today（今天）、tomorrow（明天）来指定完成命令的时间。

在系统的 /etc 目录下可能会有两个关于 at 命令使用限制的文件，一个是黑名单(etc/at.deny)一个是白名单（/etc/at.allow），一般只有一个黑名单文件，因为大部分情况下是允许执行 at 命令的，如果使用白名单，有 100 个用户的话，岂不是要写 100 行，但黑名单只需要把被限制的一两个写上就行。

- 如果系统中有/etc/at.allow 文件，那么只有被写入该文件的用户才可以执行 at 命令（将使用者的账号写入即可，一个一行）相当于白名单文件，此时/etc/at.deny 就会被忽略，相当于黑名单。
- 如果系统中没有/etc/at.allow 文件，只有/etc/at.deny，那么被写入该文件的用户就不能使用 at 命令，黑名单对 root 不起作用。
- 如果两个文件都不存在，那么只有 root 用户可以执行。

7.1.1.1 at 用法实例

使用 at 命令创建计划任务，如下示例所示。

- 三天后的下午 5 点执行/bin/ls

```
[root@openEuler ~]# at 5pm+3 days
warning: commands will be executed using /bin/sh
at> /bin/ls
at> <EOT>
job 4 at Fri Dec 18 17:00:00 2020
```

- 明天 17 点二十，输出时间到指定文件内

```
[root@openEuler ~]# at 17:20 tomorrow
warning: commands will be executed using /bin/sh
at> date > /root/2020.log
at> <EOT>
job 3 at Wed Dec 16 17:20:00 2020
```

7.1.2 周期任务 crond

Linux 下的任务调度分为两类：系统任务调度和用户任务调度。Linux 系统任务是由 cron (crond) 这个系统服务来控制的，这个系统服务是默认启动的。用户自己设置的计划任务则使用 crontab 命令。

cron 是 Linux 或者类 Unix 系统中最为实用的工具之一。cron 服务（守护进程）在系统后台运行，并且会持续地检查 /etc/crontab 文件和 /etc/cron.*/目录。它同样也会检查 /var/spool/cron/ 目录。每个用户都可以拥有自己的 crontab 文件，虽然这些文件都位于 /var/spool/cron/crontabs 目录中，但并不意味着你可以直接编辑它们。你需要通过 crontab 命令来编辑或者配置你自己的定时任务。

crontab 命令

cron 服务提供 crontab 命令来设定 cron 服务的，以下是这个命令的一些参数与说明：

- crontab -u //设定某个用户的 cron 服务，一般 root 用户在执行这个命令的时候需要此参数
- crontab -l //列出某个用户 cron 服务的详细内容
- crontab -r //删除某个用户的 cron 服务
- crontab -e //编辑某个用户的 cron 服务(按 X 删除光标所选字符，按 I 在所选位置插入，dd 删除当前行，P 粘贴)
- ：wq！ +enter 保存退出
- ：q！+enter 不保存退出

crontab 编辑语法

输入 crontab -e 后进入编辑模式，可以设置周期任务。其语法为：

- 1 2 3 4 5 /path/to/command arg1 arg2
- 其中：
- 第 1 个字段：分钟 (0-59)
- 第 2 个字段：小时 (0-23)
- 第 3 个字段：日期 (0-31)
- 第 4 个字段：月份 (0-12 [12 代表 December])
- 第 5 个字段：一周当中的某天 (0-7 [7 或 0 代表星期天])
- /path/to/command – 计划执行的脚本或命令的名称

简单的 crontab 示例如下。

- 每隔 5 分钟运行一次 backupscript 脚本 ##

```
*/5 * * * * /root/backupscript.sh
```

- 每天的凌晨 1 点运行 backupscript 脚本 ##

```
0 1 * * * /root/backupscript.sh
```

- 每月的第一个凌晨 3:15 运行 backupscript 脚本 ##

```
15 3 1 * * /root/backupscript.sh
```

如何使用操作符

操作符允许为一个字段指定多个值，这里有三个操作符可供使用：

- 星号（*）：此操作符为字段指定所有可用的值。举个例子，在小时字段中，一个星号等同于每个小时；在月份字段中，一个星号则等同于每月。
- 逗号（,）：这个操作符指定了一个包含多个值的列表，例如：1,5,10,15,20,25.
- 横杠（-）：此操作符指定了一个值的范围，例如：5-15 ，等同于使用逗号操作符键入的 5,6,7,8,9,…,13,14,15。
- 分隔符（/）：此操作符指定了一个步进值，例如： 0-23/ 可以用于小时字段来指定某个命令每小时被执行一次。步进值也可以跟在星号操作符后边，如果你希望命令行每 2 小时执行一次，则可以使用 */2。

7.2 网络管理

7.2.1 OSI 协议栈（扩展）

OSI 是 Open System Interconnection 的缩写，意为开放式系统互联。国际标准化组织（ISO）制定了 OSI 模型，该模型定义了不同计算机互联的标准，是设计和描述计算机网络通信的基本框架。OSI 模型把网络通信的工作分为 7 层，分别是物理层、数据链路层、网络层、传输层、会话层、表示层和应用层。

物理层(Physical Layer)

传输数据格式：比特(bit)流；

主要功能和连接方式：建立、维护和取消物理连接；

典型设备：光纤、同轴电缆、双绞线、网卡、中继器、集线器等；

- 说明：在 OSI 参考模型中，物理层（Physical Layer）是参考模型的最低层，也是 OSI 模型的第一层。物理层的主要功能是，利用传输介质为数据链路层提供物理连接，实现比特流的透明传输。物理层的作用是实现相邻计算机节点之间比特流的透明传送，尽可能屏蔽掉具体传输介质和物理设备的差异。使其上面的数据链路层不必考虑网络的具体传输介质是什么。"透明传送比特流"表示经实际电路传送后的比特流没有发生变化，对传送的比特流来说，这个电路好像是看不见的。

数据链路层(Data Link Layer)

传输数据格式： 将比特信息封装成数据帧(Frame)；

主要功能和连接方式：在物理层上建立、撤销、标识逻辑链接和链路复用 以及差错校验等功能。通过使用接收系统的硬件地址或物理地址来寻址；

典型设备：网桥、交换机；

- 说明：数据链路层（Data Link Layer）是 OSI 模型的第二层，负责建立和管理节点间的链路。该层的主要功能是通过各种控制协议，将有差错的物理信道变为无差错的、能可靠传输数据帧的数据链路。

该层通常又被分为介质访问控制（MAC）和逻辑链路控制（LLC）两个子层。

- MAC 子层的主要任务是解决共享型网络中多用户对信道竞争的问题，完成网络介质的访问控制；
- LLC 子层的主要任务是建立和维护网络连接，执行差错校验、流量控制和链路控制。

数据链路层的具体工作是接收来自物理层的位流形式的数据，并封装成帧，传送到上一层；同样，也将来自上层的数据帧，拆装为位流形式的数据转发到物理层；并且，还负责处理接收端发回的确认帧的信息，以便提供可靠的数据传输。

网络层(Network Layer)

传输数据格式：分割和重新组合数据包(Packet)；

主要功能和连接方式：基于网络层地址（IP 地址）进行不同网络系统间的路径选择

典型设备：网关、路由器；

- 说明：网络层（Network Layer）是 OSI 模型的第三层，它是 OSI 参考模型中最复杂的一层。其主要任务是通过路由选择算法，为报文或分组通过通信子网选择最适当的路径。该层控制数据链路层与传输层之间的信息转发，建立、维持和终止网络的连接。具体地说，数据链路层的数据在这一层被转换为数据包，然后通过路径选择、分段组合、顺序、进/出路由等控制，将信息从一个网络设备传送到另一个网络设备。一般地，数据链路层是解决同一网络内节点之间的通信，而网络层主要解决不同子网间的通信。例如在广域网之间通信时，必然会遇到路由（即两节点间可能有多条路径）选择问题。

在实现网络层功能时，需要解决的主要问题如下：

- 寻址：数据链路层中使用的物理地址（如 MAC 地址）仅解决网络内部的寻址问题。在不同子网之间通信时，为了识别和找到网络中的设备，每一子网中的设备都会被分配一个唯一的地址。由于各子网使用的物理技术可能不同，因此这个地址应当是逻辑地址（如 IP 地址）。
- 交换：规定不同的信息交换方式。常见的交换技术有：线路交换技术和存储转发技术，后者又包括报文交换技术和分组交换技术。
- 路由算法：当源节点和目的节点之间存在多条路径时，本层可以根据路由算法，通过网络为数据分组选择最佳路径，并将信息从最合适的路径由发送端传送到接收端。
- 连接服务：与数据链路层流量控制不同的是，前者控制的是网络相邻节点间的流量，后者控制的是从源节点到目的节点间的流量。其目的在于防止阻塞，并进行差错检测。

传输层(Transport Layer)

传输数据格式：数据组织成数据段(Segment)；

主要功能和连接方式： 用一个寻址机制来标识一个特定的应用程序(端口号)；

典型设备：终端设备(PC、手机、平板等)；

- 说明：OSI 下 3 层的主要任务是数据通信，上 3 层的任务是数据处理。而传输层（Transport Layer）是 OSI 模型的第 4 层。因此该层是通信子网和资源子网的接口和桥梁，起到承上启下的作用。该层的主要任务是：向用户提供可靠的端到端的差错和流量控制，保证报文的正确传输。传输层的作用是向高层屏蔽下层数据通信的细节，即向用户透明地传送报文。该层常见的协议：TCP/IP 中的 TCP 协议、Novell 网络中的 SPX 协议和微软的 NetBIOS/NetBEUI 协议。

传输层提供会话层和网络层之间的传输服务，这种服务从会话层获得数据，并在必要时，对数据进行分割。然后，传输层将数据传递到网络层，并确保数据能正确无误地传送到网络层。因此，传输层负责提供两节点之间数据的可靠传送，当两节点的联系确定之后，传输层则负责监督工作。综上，传输层的主要功能详解：

- 传输连接管理：提供建立、维护和拆除传输连接的功能。传输层在网络层的基础上为高层提供“面向连接”和“面向无接连”的两种服务。
- 处理传输差错：提供可靠的“面向连接”和不太可靠的“面向无连接”的数据传输服务、差错控制和流量控制。在提供“面向连接”服务时，通过这一层传输的数据将由目标设备确认，如果在指定的时间内未收到确认信息，数据将被重发。
- 监控服务质量。

会话层(Session Layer)

传输数据格式：数据 DTPU；

主要功能和连接方式：会话层连接到传输层的映射；会话连接的流量控制；数据传输；会话连接恢复与释放；会话连接管理、差错控制；

典型设备：终端设备(PC、手机、平板等)；

- 说明：会话层（Session Layer）是 OSI 模型的第 5 层，是用户应用程序和网络之间的接口，主要任务是：向两个实体的表示层提供建立和使用连接的方法。将不同实体之间的表示层的连接称为会话。因此会话层的任务就是组织和协调两个会话进程之间的通信，并对数据交换进行管理。用户可以按照半双工、单工和全双工的方式建立会话。当建立会话时，用户必须提供他们想要连接的远程地址。而这些地址与 MAC（介质访问控制子层）地址或网络层的逻辑地址不同，它们是为用户专门设计的，更便于用户记忆。域名（DN）就是一种网络上使用的远程地址。

会话层的具体功能详解：

- 会话管理：允许用户在两个实体设备之间建立、维持和终止会话，并支持它们之间的数据交换。例如提供单方向会话或双向同时会话，并管理会话中的发送顺序，以及会话所占用时间的长短。
- 会话流量控制：提供会话流量控制和交叉会话功能。
- 寻址：使用远程地址建立会话连接。
- 出错控制：从逻辑上讲会话层主要负责数据交换的建立、保持和终止，但实际的工作却是接收来自传输层的数据，并负责纠正错误。会话控制和远程过程调用均属于这一层的功能。但应注意，此层检查的错误不是通信介质的错误，而是磁盘空间、打印机缺纸等类型的高级错误。

表示层(Presentation Layer)

传输数据格式：数据 PTPU；

主要功能和连接方式：数据表示、数据安全、数据压缩；

典型设备：终端设备(PC、手机、平板等)；

- 说明：表示层（Presentation Layer）是 OSI 模型的第六层，它对来自应用层的命令和数据进行解释，对各种语法赋予相应的含义，并按照一定的格式传送给会话层。其主要功能是“处理用户信息的表示问题，如编码、数据格式转换和加密解密”等。

表示层的具体功能如下：

- 数据格式处理：协商和建立数据交换的格式，解决各应用程序之间在数据格式表示上的差异。
- 数据的编码：处理字符集和数字的转换。例如由于用户程序中的数据类型（整型或实型、有符号或无符号等）、用户标识等都可以有不同的表示方式，因此，在设备之间需要具有在不同字符集或格式之间转换的功能。
- 压缩和解压缩：为了减少数据的传输量，这一层还负责数据的压缩与恢复。
- 数据的加密和解密：可以提高网络的安全性。

应用层(Application Layer)

传输数据格式：数据 ATPU；

主要功能和连接方式：网络服务与使用者应用程序间的一个接口；

典型设备：终端设备(PC、手机、平板等)；

- 说明：应用层（Application Layer）是 OSI 参考模型的最高层，它是计算机用户，以及各种应用程序和网络之间的接口，其功能是直接向用户提供服务，完成用户希望在网络上完成的各种工作。它在其他 6 层工作的基础上，负责完成网络中应用程序与网络操作系统之间的联系，建立与结束使用者之间的联系，并完成网络用户提出的各种网络服务及应用所需的监督、管理和服务等各种协议。此外，该层还负责协调各个应用程序间的工作。

应用层为用户提供的服务和协议有：文件服务、目录服务、文件传输服务（FTP）、远程登录服务（Telnet）、电子邮件服务（E-mail）、打印服务、安全服务、网络管理服务、数据库服务等。上述的各种网络服务由该层的不同应用协议和程序完成，不同的网络操作系统之间在功能、界面、实现技术、对硬件的支持、安全可靠性以及具有的各种应用程序接口等各个方面的差异是很大的。应用层的主要功能如下：

- 用户接口：应用层是用户与网络，以及应用程序与网络间的直接接口，使得用户能够与网络进行交互式联系。
- 实现各种服务：该层具有的各种应用程序可以完成和实现用户请求的各种服务。

OSI 七层模型太过于理想化，现实的生产环境下比较少用上，大部分实际应用都是按照 TCP/IP 协议栈来设计实现。在 7 层模型中，每一层都提供一个特殊的网络功能。从网络功能的角度观察：下面 4 层（物理层、数据链路层、网络层和传输层）主要提供数据传输和交换功能，即以节点到节点之间的通信为主；第 4 层作为上下两部分的桥梁，是整个网络体系结构中最关键的部分；而上 3 层（会话层、表示层和应用层）则以提供用户与应用程序之间的信息和数据处理功能为主。简言之，下 4 层主要完成通信子网的功能，上 3 层主要完成资源子网的功能。

7.2.2 IPv4 地址

IP 地址是一种逻辑地址，它与 MAC 地址有所不同。它的构成：

- IP(Internet protocol)地址：网络号+主机号

Ipv4 由 32 位组成，而且每 8 位为一段，构成 4 段 8 位，每段通常用对应二进制的十进制数字表示，所以通常 ipv4 地址的格式也叫"点分十进制格式"。如果 8 位二进制，其取值范围为 00000000 ~ 11111111，其对应十进制表示为 0~255。

所以电分十进制范围为：

- 0.0.0.0~255.255.255.255

7.2.3 ifcfg 和 iproute

7.2.3.1 网络接口命名方式

传统命名

以太网：ethX, [0,oo)，例如 eth0, eth1, ...

PPP 网络：pppX, [0,...]，例如，ppp0, ppp1, ...

可预测命名方案（openEuler）

支持多种不同的命名机制：Fireware，拓扑结构

- 如果 Firmware 或 BIOS 为主板上集成的设备提供的索引信息可用，则根据此索引进行命名，如 eno1, eno2, ...
- 如果 Firmware 或 BIOS 为 PCI-E 扩展槽所提供的索引信息可用，且可预测，则根据此索引进行命名，如 ens1, ens2, ...
- 如果硬件接口的物理位置信息可用，则根据此信息命名，如 enp2s0, ...
- 如果用户显式定义，也可根据 MAC 地址命名，例如 enx122161ab2e10, ...

上述均不可用，则仍使用传统方式命名；

命名格式的组成

en：ethernet

wl：wlan

ww：wwan

名称类型

o<index>：集成设备的设备索引号；

s<slot>：扩展槽的索引号；

x<MAC>：基于 MAC 地址的命名；

p<bus>s<slot>：基于总线及槽的拓扑结构进行命名；

7.2.4 networkmanager 简介

在 openEuler 20.03 LTS 中，NetworkManager 提供的默认联网服务是一个动态网络控制和配置守护进程，它尝试在其可用时保持网络设备和连接处于活动状态。仍支持传统 ifcfg 类型配置文件。

表7-1 联网共计及应用程序概述

应用程序或工具	描述
NetworkManager	默认联网守护进程
nmcli	允许用户及脚本与 NetworkManager 互动的命令行工具

NetworkManager 可用于以下连接类型：以太网、VLAN、网桥、绑定、成组、Wi-Fi、移动宽带（比如移动网络 3G）及 IP-over-InfiniBand。在这些连接类型中，NetworkManager 可配置网络别名、IP 地址、静态路由器、DNS 信息及 VPN 连接以及很多具体连接参数。最后，NetworkManager 通过 D-bus 提供 API，D-Bus 允许应用程序查询并控制网络配置及状态。

用户和脚本都可使用命令行工具 nmcli 控制 NetworkManager。该命令的基本格式为：

- nmcli OPTIONS OBJECT { COMMAND | help }

其中 OBJECT 可为 general、networking、radio、connection 或 device 之一。最常用的选项为：-t, --terse（用于脚本）、-p, --pretty 选项（用于用户）及 -h, --help 选项。在 nmcli 中采用命令完成功能，无论何时您不确定可用的命令选项时，都可以按 Tab 查看。

nmcli 工具有一些内置上下文相关的帮助信息。例如：运行以下两个命令，并注意不同之处：

```
[root@openEuler ~]# nmcli help
Usage: nmcli [OPTIONS] OBJECT { COMMAND | help }

OPTIONS
  -a, --ask                                ask for missing parameters
  -c, --colors auto|yes|no                 whether to use colors in output
  -e, --escape yes|no                      escape columns separators in values
  -f, --fields <field,...>|all|common      specify fields to output
  -g, --get-values <field,...>|all|common  shortcut for -m tabular -t -f
  -h, --help                               print this help
  -m, --mode tabular|multiline             output mode
  -o, --overview                           overview mode
  -p, --pretty                             pretty output
  -s, --show-secrets                       allow displaying passwords
```

```
  -t, --terse                              terse output
  -v, --version                            show program version
  -w, --wait <seconds>                     set timeout waiting for finishing operations

OBJECT
  g[eneral]         NetworkManager's general status and operations
  n[etworking]      overall networking control
  r[adio]           NetworkManager radio switches
  c[onnection]      NetworkManager's connections
  d[evice]          devices managed by NetworkManager
  a[gent]           NetworkManager secret agent or polkit agent
  m[onitor]         monitor NetworkManager changes

[root@openEuler ~]# nmcli general help
Usage: nmcli general { COMMAND | help }

COMMAND := { status | hostname | permissions | logging }

  status

  hostname [<hostname>]

  permissions

  logging [level <log level>] [domains <log domains>]
```

在上面的第二个示例中，这个帮助信息与对象 general 有关。

nmcli 用法参考如下示例。

```
#显示 NetworkManager 总体状态:
nmcli general status
#要控制 NetworkManager 日志记录:
nmcli general logging
#要显示所有链接:
nmcli connection show
#要只显示当前活动链接，如下所示添加 -a, --active:
nmcli connection show --active
#显示由 NetworkManager 识别到设备及其状态:
nmcli device status
#可简化命令并省略一些选项。例如：可将命令
nmcli connection modify id 'MyCafe' 802-11-wireless.mtu 1350
#简化为
nmcli con mod MyCafe 802-11-wireless.mtu 1350
#可省略 id 选项，因为在这种情况下对于 nmcli 来说连接 ID（名称）是明确的。您熟悉这些命令后可做进一步简化。例如：可将
nmcli connection add type Ethernet
```

```
#使用 nmcli 启动和停止接口
#可使用 nmcli 工具启动和停止任意网络接口，其中包括主接口。例如：
nmcli con up id bond0
nmcli con up id port0
nmcli dev disconnect iface bond0
nmcli dev disconnect iface ens3
```

很多 nmcli 命令是可以顾名思义的，但有几个选项需要进一步了解：

type — 连接类型

- 允许值为：adsl, bond, bond-slave, bridge, bridge-slave, bluetooth, cdma, ethernet, gsm, infiniband, olpc-mesh, team, team-slave, vlan, wifi, wimax。
- 每个连接了类型都有具体类型的命令选项。按 Tab 键查看该列表，或查看 nmcli(1) man page 中的 TYPE_SPECIFIC_OPTIONS 列表。type 选项可用于如下命令：nmcli connection add 和 nmcli connection edit。
- con-name — 为连接配置分配的名称
- 如果未指定连接名称，则会以如下格式生成名称：

```
type-ifname[-number]
```

- 连接名称是 connection profile 的名称，不应与代表某个设备的名称混淆（比如 wlan0、ens3、em1 等等）。虽然用户可为根据接口为链接命名，但这是两回事。一个设备可以有多个连接配置文件。这对移动设备，或者在不同设备间反复切换网线时很有帮助。与其编辑该配置，不如创建不同的配置文件，并根据需要将其应用到接口中。id 选项也是指连接配置文件名称。

id — 用户为连接配置文件分配的身份字符串

- 可在 nmcli connection 命令中用来识别某个连接的 ID。输出结果中的 NAME 字段永远代表连接 ID（名称）。它指的是 con-name 给出的同一连接配置文件名称。
- uuid — 系统为连接配置文件分配的独有身份字符串
- 可在 nmcli connection 命令中用来识别某个连接的 UUID。

7.2.5 使用 nmcli 连接到网络

添加以太网连接意味着生成一个配置文件，并将其分配给某个设备。生成新配置文件前，请检查可用设备，方法如下：：

```
nmcli dev status
DEVICE   TYPE       STATE          CONNECTION
ens3     ethernet   disconnected   --
ens9     ethernet   disconnected   --
lo       loopback   unmanaged      --
```

添加动态以太网连接

要使用动态 IP 配置添加以太网配置文件，以便 DHCP 分配网络配置，可使用如下格式的命令：

```
nmcli connection add type ethernet con-name connection-name ifname interface-name
```

NetworkManager 会将内部参数 connection.autoconnect 设定为 yes。NetworkManager 还会将设置保存到 /etc/sysconfig/network-scripts/ifcfg-my-office 文件中，在该文件中会将 ONBOOT 指令设定为 yes。

使用以下命令激活以太网连接：

```
nmcli con up interface-name
```

7.2.6 使用 nmcli 配置静态路由

要使用 nmcli 工具配置静态路由，则必须使用命令后或者互动式编辑器。

使用命令行为现有以太网连接配置静态路由，输入如下命令：

```
nmcli connection modify eth0 +ipv4.routes "192.168.122.0/24 10.10.10.1"
```

这样会将 192.168.122.0/24 子网的流量指向位于 10.10.10.1 的网关

7.2.7 网络接口配置文件

在 openEuler 系统中，每个 interface 都有一个文件保存着该网络接口的配置，该文件位于 /etc/sysconfig/network-scripts/目录下，命名格式为 ifcfg-*interfacename*。我们可以通过修改该文件的方式修改网络接口配置，但是需要重新加载网络配置或重启网络接口才能生效。

以 enp4s0 网络接口进行静态网络设置为例，通过在 root 权限下修改 ifcfg 文件实现，在 /etc/sysconfig/network-scripts/目录中生成名为 ifcfg-enp4s0 的文件中，修改参数配置，示例如下：

```
TYPE=Ethernet
PROXY_METHOD=none
BROWSER_ONLY=no
BOOTPROTO=none
IPADDR=192.168.0.10
PREFIX=24
DEFROUTE=yes
IPV4_FAILURE_FATAL=no
IPV6INIT=yes
IPV6_AUTOCONF=yes
IPV6_DEFROUTE=yes
IPV6_FAILURE_FATAL=no
IPV6_ADDR_GEN_MODE=stable-privacy
NAME=enp4s0static
UUID=08c3a30e-c5e2-4d7b-831f-26c3cdc29293
DEVICE=enp4s0
ONBOOT=yes
```

要通过 ifcfg 文件为 em1 接口配置动态网络，请按照如下操作在/etc/sysconfig/network-scripts/目录中生成名为 ifcfg-em1 的文件，示例如下：

```
DEVICE=em1
```

```
BOOTPROTO=dhcp
ONBOOT=yes
```

要配置一个向 DHCP 服务器发送不同的主机名的接口，请在 ifcfg 文件中新增一行内容，如下所示：

```
DHCP_HOSTNAME=hostname
```

要配置忽略由 DHCP 服务器发送的路由，防止网络服务使用从 DHCP 服务器接收的 DNS 服务器更新/etc/resolv.conf。请在 ifcfg 文件中新增一行内容，如下所示：

```
PEERDNS=no
```

要配置一个接口使用具体 DNS 服务器，请将参数 PEERDNS=no，并在 ifcfg 文件中添加以下行：

```
DNS1=ip-address
DNS2=ip-address
```

其中 ip-address 是 DNS 服务器的地址。这样就会让网络服务使用指定的 DNS 服务器更新/etc/resolv.conf。

7.2.8 主机名管理

每个 Linux 系统都有一个主机名，hostname 有三种类型：static、transient 和 pretty。

- static：静态主机名，可由用户自行设置，并保存在/etc/hostname 文件中。
- transient：动态主机名，由内核维护，初始是 static 主机名，默认值为“localhost”。可由 DHCP 或 mDNS 在运行时更改。
- pretty：灵活主机名，允许使用自由形式（包括特殊/空白字符）进行设置。静态/动态主机名遵从域名的通用限制

7.2.8.1 使用 hostnamectl 配置主机名

查看当前的主机名，使用如下命令：

```
hostnamectl status
```

在 root 权限下，设定系统中的所有主机名，使用如下命令：

```
hostnamectl set-hostname name
```

在 root 权限下，通过不同的参数来设定特定主机名，使用如下命令：

```
hostnamectl set-hostname name [option...]
```

- 其中 option 可以是--pretty、--static、--transient 中的一个或多个选项。
- 如果--static 或--transient 与--pretty 选项一同使用时，则会将 static 和 transient 主机名简化为 pretty 主机名格式，使用“-”替换空格，并删除特殊字符。
- 当设定 pretty 主机名时，如果主机名中包含空格或单引号，需要使用引号。命令示例如下：

```
hostnamectl set-hostname "Stephen's notebook" --pretty
```

7.2.8.2 使用 nmcli 配置主机名

查询 static 主机名，使用如下命令：

```
nmcli general hostname
```

在 root 权限下，将 static 主机名设定为 host-server，使用如下命令：

```
nmcli general hostname host-server
```

要让系统 hostnamectl 感知到 static 主机名的更改，在 root 权限下，重启 hostnamed 服务，使用如下命令：

```
systemctl restart systemd-hostnamed
```

7.3 进程管理

7.3.1 进程概述

通俗的讲程序是一个包含可以执行代码的静态的文件。进程是一个开始执行但是还没有结束的程序的实例。

当程序被系统调用到内存以后，系统会给程序分配一定的资源（内存，设备等等）然后进行一系列的复杂操作，使程序变成进程以供系统调用。

7.3.1.1 进程分类

按照进程的功能和运行的程序分类，进程可划分为两大类：

系统进程

可以执行内存资源分配和进程切换等管理工作，而且该进程的运行不受用户的干预，即使是 root 用户也不能干预系统进程的运行。

用户进程

通过执行用户程序、应用程序或内核之外的系统程序而产生的进程，此类进程可以在用户的控制下运行或关闭。

针对用户进程，又可以分为如下 3 类：

- 交互进程：由一个 Shell 终端其他的进程，在执行过程中，需要与用户进行交互操作，可以运行于前台，也可以运行于后台。
- 批处理进程：该进程是一个进程集合，负责按顺序启动其他的进程。
- 守护进程：守护进程是一直运行的一种进程，经常在 Linux 系统时启动，在系统关闭时终止。它们独立于控制终端且周期性地质学某种任务或等待处理某些发生的时间。例，httpd 进程，crond 进程等。

7.3.1.2 进程状态

为了充分的利用资源，系统还对进程区分了不同的状态。一般操作系统将进程分为五个状态：

- 新建：新建表示进程正在被创建。
- 运行：运行是进程正在运行。
- 阻塞：阻塞是进程正在等待某一个事件发生。

- 就绪：就绪是表示系统正在等待 CPU 来执行命令。
- 完成：完成表示进程已经结束了系统正在回收资源。

Linux 上进程有 5 种状态，这 5 中状态可以与一般操作系统的状态对应起来：

- 运行：正在运行或在运行队列中等待。
- 中断：休眠中， 受阻， 在等待某个条件的形成或接受到信号。
- 不可中断：收到信号不唤醒和不可运行， 进程必须等待直到有中断发生。
- 僵死：进程已终止， 但进程描述符存在， 直到父进程调用 wait4()系统调用后释放。
- 停止：进程收到 SIGSTOP， SIGSTP， SIGTIN， SIGTOU 信号后停止运行运行。

7.3.1.3 进程 ID 与父子进程

一个程序可能有许多进程，而每一个进程又可以有许多子进程。依次循环下去，而产生子孙进程。

为了区分各个不同的进程，系统给每一个进程分配了一个 ID 以便识别。Linux 系统中，进程 ID（PID）是区分不同进程的唯一标识。PPID 表示父进程。所有的进程都是 PID 为 1 的 init 进程的后代。内核在系统启动的最后阶段启动 init 进程。

一般每个进程都会有父进程，父进程与子进程之间是管理与被管理的关系，当父进程停止时，子进程也随之消失，但子进程关闭，父进程不一定终止。

7.3.1.4 僵尸进程

每个进程在结束后都会处于僵死状态，等待父进程将其释放资源，处于该状态的进程已经结束，但父进程还没有释放其系统资源。

由于某种原因，父进程在子进程退出前退出，则所有子进程就变成一个孤儿进程，拖没有相应处理机制，则孤儿进程会一直处于僵死状态，资源无法释放。这种僵死的孤儿进程即僵尸进程。

此时解决方法是在启动进程内找一个进程作为这些孤儿进程的父进程，或者直接让 init 进程作为它们的父进程，进而释放孤儿进程占用的资源。

7.3.1.5 线程

线程在 Linux 中被称为轻量级的进程。进程有独立的内存地址空间，线程没有。线程不能独立存在，线程由进程创建的。

7.3.2 进程监控

7.3.2.1 使用 ps 命令监控系统进程

ps 命令（Process Status）用于显示当前进程的状态。ps 命令有两种不同风格的语法规则：

BSD 形式，BSD 形式的语法的选项前没有破折号，如：ps aux

UNIX/LINUX 形式，Linux 形式的语法的选项前有破折号，如：ps -ef

在 Linux 系统上混合这两种语法是可以的。比如 "ps ax -f"。这里主要讨论 UNIX 形式语法。

- 注意："ps aux"不等同于"ps -aux"。比如"-u"用于显示用户的进程，但是"u"意味着显示具体信息。

7.3.2.2 ps 常用参数及显示

- -a：显示同一终端下的所有程序。
- -A：显示所有进程。
- -u：有效用户相关的进程。
- -N：反向选择。
- -f：详细显示程序执行的路径群。
- -e ： 所有进程。等同于-A。
- -l ： 显示长格式。
- -F ： 附加全格式。
- -H ： 显示进程的树状结构。
- -L ： 显示线程，可能出现 LWP 和 NLWP 栏位。
- -m ： 在进程后显示线程。
- -h 不显示标题。
- -w 宽输出。
- --lines<行数> 每页显示的行数。
- --width<字符数> 每页显示的字符数。
- --help 显示帮助信息。
- --version 显示版本显示。
- a：显示终端中包括其它用户的所有进程。
- x：显示无控制终端的进程。
- r：显示当前终端的进程。
- c：显示进程的真实名称。
- e：显示环境变量。
- f： 显示程序间的关系。
- T：显示当前终端的所有程序
- u: 指定用户的所有进程

结果的列显示

- F ： 进程标志。
- S ： 进程状态。同 STAT。
- SID 会话 ID（Session id）
- USER：该进程属于那个使用者账号的。
- PID ： 进程 ID。
- PPID ： 父进程的进程 ID（Parent Process id） 。
- %CPU：进程占 CPU 的百分比。
- %MEM ： 该进程所占用的物理内存百分比
- VSZ ： 进程使用掉的虚拟内存量 (Kbytes)（Virtual Size） 。
- RSS ： 进程占用的固定的内存量 (Kbytes)。
- TTY ： 与进程关联的终端（tty）。进程是在那个终端机上面运作，若与终端机无关，则显示 ?，另外，tty1-tty6 是本机上面的登入者程序，若为 pts/0 等等的，则表示为由网络连接进主机的程序。
- STAT：程序目前的状态，主要的状态有：
- D ： 不可中断
- R ： 运行
- S ： 中断
- T ： 停止

- Z ： 僵死
- START：进程被触发启动的时间。
- TIME ： 进程实际使用 CPU 运作的时间。
- COMMAND：进程的运启动的实际指令。
- LWP ： LWP(轻量级进程【 light weight process 】，也称作线程)ID。
- C ： 处理器使用率百分比
- NLWP ： 进程中 lwp（线程）的数量。（别名 thcount）
- PRI ： 进程的优先级。（参看 1.1.4“进程优先级和 Nice 值”）
- NI ： Nice 值（whether the process tries to be nice by adjusting the priority by the number given; see below for details）
- ADDR ： 进程地址空间（不显示）
- SZ ： 进程所有内存（code+data+stack）总数，单位为 KB。
- WCHAN ： 内核功能名称，如果进程正在运行中
- RSS ： 常驻集大小，任务所使用的非交换物理内存（KB）
- PSR ： 当前执行进程的处理器
- STIME ： 开始时间。
- CMD ： 启动任务的命令行（包括参数）
- WCHAN ： 进程正在睡眠的内核函数名称；该函数的名称是从/root/system.map 文件中获得的。
- FLAGS ： 与进程相关的数字标识。

7.3.2.3 使用 pstree 命令监控系统进程

该命令显示当前运行的所有进程及其相关的子进程，以树的格式输出。

基本格式： pstree 参数

- 说明：pestree 命令对程序名称相同的会自动合并，所有“|-httpd---8*[httpd]”即表示系统中有 8 个 httpd 进程产生的子进程。

参数：

- -a ： 显示出该命令的参数，假如这个命令进程被其他进程替换掉，那么进程将显示在括号中 -a 选项包含有压实进程树的选项，对于相同的进程，会使用 n*(process)的形式展显出来。
- -c ： 关闭禁用显示结果进程树，在默认情况下，进程子树是会被压缩的。不管有多少进程名相同的进程，都会逐个显示出来。
- -G ： 使用 vt100 线性描述树
- -h ： 突空出显示当前进程的父进程并高亮显示出来，如果没有父进程那么什么都不会显示。
- -H ： 突出显示出指定进程的父进程信息并高亮显示出来，使用方法为 pstree -H PID
- -l ： 显示长格式命令选项，在默认的情况下，命令行最多显示宽度为 132bit ，超过将不能正常显示。
- -n ： 基于进程相同的祖先来进行排序，可以命名 pid 来代替进程名称 。
- -p ： 显示所有的时程，显示结果包含进程名和时进程 ID
- -u ： 显示出用户的 UID，无论何时，这个 UID 和进程比较 UID 参数，这个新的 UID 将在进程名后显示不同的参数。
- -U ： 使用 utf-8 字符集以十进制表示，
- -v ： 显示版本号。

7.3.2.4 使用 top 命令监控系统进程

top 命令可以动态管理监控 linux 进程，非常类似于 Windows 任务管理器。top 命令是一个功能十分强大的监控系统的工具，对于系统管理员而言尤其重要。但是，它的缺点是会消耗一定的系统资源。

top 界面详解

第一行：任务队列信息

- 当前时间
- 系统运行时间。
- 当前登录用户数。
- 系统负载，即任务队列的平均长度。 三个数值分别为 1 分钟、5 分钟、15 分钟前到现在的平均值。

第二行：进程数信息

- 进程总数
- 在运行的进程数
- 睡眠的进程数
- 停止的进程数
- 僵尸进程数

第三行：CPU 状态

- 用户进程占用 CPU 百分比。
- 系统进程占用 CPU 百分比。
- 用户进程空间内改变过优先级的进程占用 CPU 百分比。
- 空闲 CPU 百分比。
- IO 等待占用 CPU 的百分比
- 硬中断（Hardware IRQ）占用 CPU 的百分比
- 软中断（Software Interrupts）占用 CPU 的百分比

第四行：内存状态

- 物理内存总量
- 使用的物理内存总量
- 空闲内存总量
- 用作内核缓存的内存量

第五行：swap 交换状态

- 交换区总量。
- 使用的交换区总量。
- 空闲交换区总量。
- 缓冲的交换区总量。

第六行：空行

第七行以下：各进程的状态监控

- PID 进程 id
- PPID 父进程 id
- RUSER Real user name
- UID 进程所有者的用户 id
- USER 进程所有者的用户名
- GROUP 进程所有者的组名
- TTY 启动进程的终端名。不是从终端启动的进程则显示为 ？
- PR 优先级

- NI nice 值。负值表示高优先级，正值表示低优先级。
- P 最后使用的 CPU，仅在多 CPU 环境下有意义
- %CPU 上次更新到现在的 CPU 时间占用百分比
- TIME 进程使用的 CPU 时间总计，单位秒
- TIME+ 进程使用的 CPU 时间总计，单位 1/100 秒
- %MEM 进程使用的物理内存百分比
- VIRT 进程使用的虚拟内存总量，单位 kb。VIRT=SWAP+RES
- SWAP 进程使用的虚拟内存中，被换出的大小，单位 kb。
- RES 进程使用的、未被换出的物理内存大小，单位 kb。RES=CODE+DATA
- CODE 可执行代码占用的物理内存大小，单位 kb
- DATA 可执行代码以外的部分(数据段+栈)占用的物理内存大小，单位 kb
- SHR 共享内存大小，单位 kb
- nFLT 页面错误次数
- nDRT 最后一次写入到现在，被修改过的页面数。
- S 进程状态。
  - D=不可中断的睡眠状态
  - R=运行
  - S=睡眠
  - T=跟踪/停止
  - Z=僵尸进程
- COMMAND 命令名/命令行
- WCHAN 若该进程在睡眠，则显示睡眠中的系统函数名
- Flags 任务标志，参考 sched.h

7.3.2.5 结束进程

kill 命令

kill 语法：

- kill [OPTION] [PID]

命令参数：

- -l ： 信号，若果不加信号的编号参数，则使用“-l”参数会列出全部的信号名称。
- -a ： 当处理当前进程时，不限制命令名和进程号的对应关系。
- -p ： 指定 kill 命令只打印相关进程的进程号，而不发送任何信号。
- -s ： 指定发送信号。
- -u ： 指定用户。

只有第 9 种信号(SIGKILL)才可以无条件终止进程，其他信号进程都有权利忽略。 下面是常用的信号：

- HUP 1 终端断线
- INT 2 中断（同 Ctrl + C）
- QUIT 3 退出（同 Ctrl + \）
- TERM 15 终止
- KILL 9 强制终止
- CONT 18 继续（与 STOP 相反， fg/bg 命令）
- STOP 19 暂停（同 Ctrl + Z）

kill 结束进程有些进程是无法杀死的。关键进程是无法结束的。比如 bash 的进程。

init 进程是无法被终止，或者说是不允。init 是 Linux 系统操作中不可缺少的程序之一。所谓的 init 进程，它是一个由内核启动的用户级进程。内核自行启动（已经被载入内存，开始运行，并已初始化 所有的设备驱动程序和数据结构等）之后，就通过启动一个用户级程序 init 的方式，完成引导进程。所以,init 始终是第一个进程（其进程编号始终为 1）。 其它所有进程都是 init 进程的子孙。init 进程是不可杀的!

killall 命令

该命令用于结束指定名字的进程及其所有子进程。例如结束 svn 服务器的进程：

```
killall svnserver
```

若需要强制结束进程，与 kill 命令相似，可以使用 -9 ，例：killall -9 cpusd。

7.3.2.6 进程优先级

Priority 值与 Nice 值

PRI 值越低，表示进程运行的优先级越高，PRI 值有 Linux 内核调整的，用户无法直接调整 PRI 值。若用户需要调整进程的优先级，可以通过 Nice 值进行。一般来说，PRI 与 NI 的关系如下：

- PRI(new) = PRI(old) + NI
  - NI 的范围为-20 ~ 19 ，40 个等级。
  - NI 为负数时，PRI 的值会减小，进程的优先级就会提高。

7.4 思考题

- 请将主机名修改为 openEuler。
- 给虚拟机新增一个网卡，设置网卡的 IP 地址为 192.168.101.100/24，网关为 192.168.101.254；192.168.0.0/16 网段的默认路由走该网卡。
- 新建一个定时任务，在下午 17 点半输出当前时间。
- 新建一个周期任务，每天晚上 2 点备份/etc 目录到/backup 目录，备份过去的目录名称需要修改为当前日期-etc，如 20201202-etc。
- 新建一个周期任务，每隔一小时检测一次系统根分区使用率，若根分区使用率超过 50%，则输出一个告警信息，并删除/tmp 目录内所有数据。

8 使用 shell 脚本

8.1 shell 基础介绍

8.1.1 shell 简介

Shell 本身是一个用 C 语言编写的程序，它是用户使用 Unix/Linux 的桥梁，用户的大部分工作都是通过 Shell 完成的。Shell 既是一种命令语言，又是一种程序设计语言。作为命令语言，它交互式地解释和执行用户输入的命令；作为程序设计语言，它定义了各种变量和参数，并提供了许多在高级语言中才具有的控制结构，包括循环和分支。

它虽然不是 Unix/Linux 系统内核的一部分，但它调用了系统核心的大部分功能来执行程序、建立文件并以并行的方式协调各个程序的运行。因此，对于用户来说，shell 是最重要的实用程序，深入了解和熟练掌握 shell 的特性极其使用方法，是用好 Unix/Linux 系统的关键。

Shell 有两种执行命令的方式：

- 交互式（Interactive）：解释执行用户的命令，用户输入一条命令，Shell 就解释执行一条。
- 批处理（Batch）：用户事先写一个 Shell 脚本(Script)，其中有很多条命令，让 Shell 一次把这些命令执行完，而不必一条一条地敲命令。

Shell 脚本和编程语言很相似，也有变量和流程控制语句，但 Shell 脚本是解释执行的，不需要编译，Shell 程序从脚本中一行一行读取并执行这些命令，相当于一个用户把脚本中的命令一行一行敲到 Shell 提示符下执行。

Shell 初学者请注意，在平常应用中，建议不要用 root 帐号运行 Shell 。作为普通用户，不管您有意还是无意，都无法破坏系统；但如果是 root，那就不同了，只要敲几个字母，就可能导致灾难性后果。

8.1.2 几种常见的 shell

Unix/Linux 上常见的 Shell 脚本解释器有 bash、sh、csh、ksh 等，习惯上把它们称作一种 Shell。我们常说有多少种 Shell，其实说的是 Shell 脚本解释器。openEuler 默认的登录 shell 是 bash。

bash

bash 是 Linux 标准默认的 shell，本教程也基于 bash 讲解。bash 由 Brian Fox 和 Chet Ramey 共同完成，是 BourneAgain Shell 的缩写，内部命令一共有 40 个。

Linux 使用它作为默认的 shell 是因为它有诸如以下的特色：

- 可以使用类似 DOS 下面的 doskey 的功能，用方向键查阅和快速输入并修改命令；
- 自动通过查找匹配的方式给出以某字符串开头的命令；
- 包含了自身的帮助功能，你只要在提示符下面键入 help 就可以得到相关的帮助。

sh

sh 由 Steve Bourne 开发，是 Bourne Shell 的缩写，sh 是 Unix 标准默认的 shell。

ash

ash shell 是由 Kenneth Almquist 编写的，Linux 中占用系统资源最少的一个小 shell，它只包含 24 个内部命令，因而使用起来很不方便。

csh

csh 是 Linux 比较大的内核，它由以 William Joy 为代表的共计 47 位作者编成，共有 52 个内部命令。该 shell 其实是指向 /bin/tcsh 这样的一个 shell，也就是说，csh 其实就是 tcsh。

ksh

ksh 是 Korn shell 的缩写，由 Eric Gisin 编写，共有 42 条内部命令。该 shell 最大的优点是几乎和商业发行版的 ksh 完全兼容，这样就可以在不用花钱购买商业版本的情况下尝试商业版本的性能了。

- 注意：bash 是 Bourne Again Shell 的缩写，是 linux 标准的默认 shell ，它基于 Bourne shell，吸收了 C shell 和 Korn shell 的一些特性。bash 完全兼容 sh，也就是说，用 sh 写的脚本可以不加修改的在 bash 中执行。

8.1.3 shell 与编译语言的差异

大体上，可以将程序设计语言可以分为两类：编译型语言和解释型语言。

编译型语言

很多传统的程序设计语言，例如 Fortran、Ada、Pascal、C、C++ 和 Java，都是编译型语言。这类语言需要预先将我们写好的源代码(source code)转换成目标代码(object code)，这个过程被称作“编译”。

运行程序时，直接读取目标代码(object code)。由于编译后的目标代码(object code)非常接近计算机底层，因此执行效率很高，这是编译型语言的优点。

但是，由于编译型语言多半运作于底层，所处理的是字节、整数、浮点数或是其他机器层级的对象，往往实现一个简单的功能需要大量复杂的代码。例如，在 C++ 里，就很难进行“将一个目录里所有的文件复制到另一个目录中”之类的简单操作。

解释型语言

解释型语言也被称作“脚本语言”。执行这类程序时，解释器(interpreter)需要读取我们编写的源代码(source code)，并将其转换成目标代码(object code)，再由计算机运行。因为每次执行程序都多了编译的过程，因此效率有所下降。

使用脚本编程语言的好处是，它们多半运行在比编译型语言还高的层级，能够轻易处理文件与目录之类的对象；缺点是它们的效率通常不如编译型语言。不过权衡之下，通常使用脚本编程还是值得的：花一个小时写成的简单脚本，同样的功能用 C 或 C++ 来编写实现，可能需要两天，而且一般来说，脚本执行的速度已经够快了，快到足以让人忽略它性能上的问题。脚本编程语言的例子有 awk、Perl、Python、Ruby 与 Shell。

8.1.4 什么时候用 shell

因为 Shell 似乎是各 UNIX 系统之间通用的功能，并且经过了 POSIX 的标准化。因此，Shell 脚本只要“用心写”一次，即可应用到很多系统上。因此，之所以要使用 Shell 脚本是基于：

- 简单性：Shell 是一个高级语言；通过它，你可以简洁地表达复杂的操作。
- 可移植性：使用 POSIX 所定义的功能，可以做到脚本无须修改就可在不同的系统上执行。
- 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。

但是，考虑到 Shell 脚本的命令限制和效率问题，下列情况一般不使用 Shell：

- 资源密集型的任务，尤其在需要考虑效率时（比如，排序，hash 等等）。
- 需要处理大任务的数学操作，尤其是浮点运算，精确运算，或者复杂的算术运算（这种情况一般使用 C++ 或 FORTRAN 来处理）。
- 有跨平台（操作系统）移植需求（一般使用 C 或 Java）。
- 复杂的应用，在必须使用结构化编程的时候（需要变量的类型检查，函数原型，等等）。
- 对于影响系统全局性的关键任务应用。
- 对于安全有很高要求的任务，比如你需要一个健壮的系统来防止入侵、破解、恶意破坏等等。
- 项目由连串的依赖的各个部分组成。
- 需要大规模的文件操作。
- 需要多维数组的支持。
- 需要数据结构的支持，比如链表或数等数据结构。
- 需要产生或操作图形化界面 GUI。
- 需要直接操作系统硬件。
- 需要 I/O 或 socket 接口。
- 需要使用库或者遗留下来的老代码的接口。
- 私人的、闭源的应用（shell 脚本把代码就放在文本文件中，全世界都能看到）。

如果你的应用符合上边的任意一条，那么就考虑一下更强大的语言吧——或许是 Perl、Tcl、Python、Ruby ——或者是更高层次的编译语言比如 C/C++，或者是 Java。即使如此，你会发现，使用 shell 来原型开发你的应用，在开发步骤中也是非常有用的。

8.2 shell 编程基础

8.2.1 shell 脚本规范与运行

通常，shell 脚本为了便于区分，会以.sh 结尾，如 test.sh。在 shell 脚本的第一行，会写入以“#!”开头的标记，它会高速系统这个脚本需要用什么解释器来执行，如“#!/bin/bash”，即使用 bash 解释该脚本。

运行脚本需要给脚本添加执行权限，可以使用 chmod 命令添加执行权限。具有执行权限的脚本，可以通过输入绝对路径如/root/test.sh 执行，也可以进入/root 目录，输入./test.sh 执行。

在 shell 脚本中，可以在行前添加#号，注释单行。

8.2.2 shell 变量

Shell 支持自定义变量。

定义变量

定义变量时，变量名不加美元符号（$），如：

```
variableName="value"
```

注意，变量名和等号之间不能有空格，这可能和你熟悉的所有编程语言都不一样。同时，变量名的命名须遵循如下规则：

- 首个字符必须为字母（a-z，A-Z）；
- 中间不能有空格，可以使用下划线（_）；
- 不能使用标点符号；
- 不能使用 bash 里的关键字（可用 help 命令查看保留关键字）。

使用变量

使用一个定义过的变量，只要在变量名前面加美元符号（$）即可，如：

```
your_name="mozhiyan"
echo $your_name
echo ${your_name}
```

变量名外面的花括号是可选的，加不加都行，加花括号是为了帮助解释器识别变量的边界，比如下面这种情况：

```
for skill in Ada Coffe Action Java
do
    echo "I am good at ${skill}Script"
done
```

如果不给 skill 变量加花括号，写成 echo "I am good at $skillScript"，解释器就会把 $skillScript 当成一个变量（其值为空），代码执行结果就不是我们期望的样子了。

推荐给所有变量加上花括号，这是个好的编程习惯。

重新定义变量

已定义的变量，可以被重新定义，如：

```
myUrl="http://see.xidian.edu.cn/cpp/linux/"
echo ${myUrl}
myUrl="http://see.xidian.edu.cn/cpp/shell/"
echo ${myUrl}
```

这样写是合法的，但注意，第二次赋值的时候不能写 $myUrl="http://see.xidian.edu.cn/cpp/shell/"，使用变量的时候才加美元符（$）。

只读变量

使用 readonly 命令可以将变量定义为只读变量，只读变量的值不能被改变。下面的例子尝试更改只读变量，结果报错：

```
#!/bin/bash

myUrl="http://see.xidian.edu.cn/cpp/shell/"
readonly myUrl
myUrl="http://see.xidian.edu.cn/cpp/danpianji/"
```

运行脚本，结果如下：

```
/bin/sh: NAME: This variable is read only.
```

删除变量

使用 unset 命令可以删除变量。语法：

```
unset variable_name
```

变量被删除后不能再次使用；unset 命令不能删除只读变量。

举个例子：

```
#!/bin/sh
myUrl="http://see.xidian.edu.cn/cpp/u/xitong/"
unset myUrl
echo $myUrl
```

上面的脚本没有任何输出。

变量类型

运行 shell 时，会同时存在三种变量：

- 局部变量：局部变量在脚本或命令中定义，仅在当前 shell 实例中有效，其他 shell 启动的程序不能访问局部变量。
- 环境变量：所有的程序，包括 shell 启动的程序，都能访问环境变量，有些程序需要环境变量来保证其正常运行。必要的时候 shell 脚本也可以定义环境变量。
- shell 变量：shell 变量是由 shell 程序设置的特殊变量。shell 变量中有一部分是环境变量，有一部分是局部变量，这些变量保证了 shell 的正常运行

8.2.3 shell 特殊变量与位置化参数

特殊变量

在运行脚本时，还会涉及到一些特殊变量，如表 8-1。

表8-1 特殊变量

变量	含义
$0	当前脚本的文件名
$n	传递给脚本或函数的参数。n 是一个数字，表示第几个参数。例如，第一个参数是$1，第二个参数是$2。
$#	传递给脚本或函数的参数个数。
$*	传递给脚本或函数的所有参数。
$@	传递给脚本或函数的所有参数。被双引号(" ")包含时，与 $* 稍有不同，下面将会讲到。
$?	上个命令的退出状态，或函数的返回值。
$$	当前 Shell 进程 ID。对于 Shell 脚本，就是这些脚本所在的进程 ID。

位置化参数

运行脚本时传递给脚本的参数称为命令行参数。命令行参数用 $n 表示，例如，$1 表示第一个参数，$2 表示第二个参数，依次类推。例如：

```
cat /root/test.sh
#!/bin/bash

echo "File Name: $0"
```

```
echo "First Parameter : $1"
echo "First Parameter : $2"
echo "Quoted Values: $@"
echo "Quoted Values: $*"
echo "Total Number of Parameters : $#"
```

运行结果：

```
$./test.sh huawei openEuler
File Name : ./test.sh
First Parameter : huawei
Second Parameter : openEuler
Quoted Values: huawei openEuler
Quoted Values: huawei openEuler
Total Number of Parameters : 2
```

8.2.4 shell 替换

字符替换

如果表达式中包含特殊字符，Shell 将会进行替换。例如，在双引号中使用变量就是一种替换，转义字符也是一种替换。

举个例子：

```
#!/bin/bash
a=10
echo -e "Value of a is $a \n"
```

运行结果：

```
Value of a is 10
```

这里 -e 表示对转义字符进行替换。如果不使用 -e 选项，将会原样输出：

```
Value of a is 10\n
```

下面的转义字符都可以用在 echo 中。

表8-2 转义字符

转义字符	含义
\	转义
\a	警报，响铃
\b	退格（删除键）

\f	换页(FF)，将当前位置移到下页开头
\n	换行
\r	回车
\t	水平制表符（tab 键）
\v	垂直制表符

命令替换

命令替换是指 Shell 可以先执行命令，将输出结果暂时保存，在适当的地方输出。

命令替换的语法： command 注意是反引号，不是单引号，这个键位于 Esc 键下方。

下面的例子中，将命令执行结果保存在变量中：

```
#!/bin/bash
DATE=`date`
echo "Date is $DATE"
USERS=`who | wc -l`
echo "Logged in user are $USERS"
UP=`date ; uptime`
echo "Uptime is $UP"
```

运行结果：

```
Date is Thu Jul  2 03:59:57 MST 2009
Logged in user are 1
Uptime is Thu Jul  2 03:59:57 MST 2009
03:59:57 up 20 days, 14:03,  1 user,  load avg: 0.13, 0.07, 0.15
```

变量替换

变量替换可以根据变量的状态（是否为空、是否定义等）来改变它的值

可以使用的变量替换形式。

表8-3 变量替换形式

形式	说明
${var}	变量本来的值

${var:-word}	如果变量 var 为空或已被删除(unset)，那么返回 word，但不改变 var 的值
${var:=word}	如果变量 var 为空或已被删除(unset)，那么返回 word，并将 var 的值设置为 word。
${var:?message}	如果变量 var 为空或已被删除(unset)，那么将消息 message 送到标准错误输出，可以用来检测变量 var 是否可以被正常赋值。若此替换出现在 Shell 脚本中，那么脚本将停止运行。
${var:+word}	如果变量 var 被定义，那么返回 word，但不改变 var 的值。

例如：

```
#!/bin/bash

echo ${var:-"Variable is not set"}
echo "1 - Value of var is ${var}"

echo ${var:="Variable is not set"}
echo "2 - Value of var is ${var}"

unset var
echo ${var:+"This is default value"}
echo "3 - Value of var is $var"

var="Prefix"
echo ${var:+"This is default value"}
echo "4 - Value of var is $var"

echo ${var:?"Print this message"}
echo "5 - Value of var is ${var}"
```

运行结果：

```
Variable is not set
1 - Value of var is
Variable is not set
2 - Value of var is Variable is not set

3 - Value of var is
This is default value
4 - Value of var is Prefix
Prefix
5 - Value of var is Prefix
```

8.2.5 shell 运算符

Bash 支持很多运算符，包括算数运算符、关系运算符、布尔运算符、字符串运算符和文件测试运算符。原生 bash 不支持简单的数学运算，但是可以通过其他命令来实现，例如 awk 和 expr，expr 最常用。

算数运算符

先来看一个示例：

```
#!/bin/sh

a=10
b=20
val=`expr $a + $b`
echo "a + b : $val"

val=`expr $a - $b`
echo "a - b : $val"

val=`expr $a \* $b`
echo "a * b : $val"

val=`expr $b / $a`
echo "b / a : $val"

val=`expr $b % $a`
echo "b % a : $val"

if [ $a == $b ]
then
   echo "a is equal to b"
fi

if [ $a != $b ]
then
   echo "a is not equal to b"
fi
```

运行结果：

```
a + b : 30
a - b : -10
a * b : 200
b / a : 2
b % a : 0
a is not equal to b
```

表8-4 算术运算符列表

运算符	说明	举例
+	加法	`expr $a + $b` 结果为 30。
-	减法	`expr $a - $b` 结果为 10。
*	乘法	`expr $a \* $b` 结果为 200。
/	除法	`expr $b / $a` 结果为 2。
%	取余	`expr $b % $a` 结果为 0。
=	赋值	a=$b 将把变量 b 的值赋给 a。
==	相等。用于比较两个数字，相同则返回 true。	[$a == $b] 返回 false。
!=	不相等。用于比较两个数字，不相同则返回 true。	[$a != $b] 返回 true。

注意：条件表达式要放在方括号之间，并且要有空格，例如 [$a==$b] 是错误的，必须写成 [$a == $b]。

关系运算符

关系运算符只支持数字，不支持字符串，除非字符串的值是数字。

表8-5 关系运算符列表

运算符	说明
-eq	检测两个数是否相等，相等返回 true。
-ne	检测两个数是否相等，不相等返回 true。
-gt	检测左边的数是否大于右边的，如果是，则返回 true。
-lt	检测左边的数是否小于右边的，如果是，则返回 true。
-ge	检测左边的数是否大等于右边的，如果是，则返回 true。

-le	检测左边的数是否小于等于右边的，如果是，则返回 true。

示例

```
#!/bin/sh

a=10
b=20
if [ $a -eq $b ]
then
   echo "$a -eq $b : a is equal to b"
else
   echo "$a -eq $b: a is not equal to b"
fi

if [ $a -ne $b ]
then
   echo "$a -ne $b: a is not equal to b"
else
   echo "$a -ne $b : a is equal to b"
fi

if [ $a -gt $b ]
then
   echo "$a -gt $b: a is greater than b"
else
   echo "$a -gt $b: a is not greater than b"
fi

if [ $a -lt $b ]
then
   echo "$a -lt $b: a is less than b"
else
   echo "$a -lt $b: a is not less than b"
fi

if [ $a -ge $b ]
then
   echo "$a -ge $b: a is greater or  equal to b"
else
   echo "$a -ge $b: a is not greater or equal to b"
fi

if [ $a -le $b ]
then
   echo "$a -le $b: a is less or  equal to b"
else
```

```
    echo "$a -le $b: a is not less or equal to b"
fi
```

运行结果：

```
10 -eq 20: a is not equal to b
10 -ne 20: a is not equal to b
10 -gt 20: a is not greater than b
10 -lt 20: a is less than b
10 -ge 20: a is not greater or equal to b
10 -le 20: a is less or    equal to b
```

布尔运算符

示例

```
#!/bin/sh

a=10
b=20

if [ $a != $b ]
then
    echo "$a != $b : a is not equal to b"
else
    echo "$a != $b: a is equal to b"
fi

if [ $a -lt 100 -a $b -gt 15 ]
then
    echo "$a -lt 100 -a $b -gt 15 : returns true"
else
    echo "$a -lt 100 -a $b -gt 15 : returns false"
fi

if [ $a -lt 100 -o $b -gt 100 ]
then
    echo "$a -lt 100 -o $b -gt 100 : returns true"
else
    echo "$a -lt 100 -o $b -gt 100 : returns false"
fi

if [ $a -lt 5 -o $b -gt 100 ]
then
    echo "$a -lt 100 -o $b -gt 100 : returns true"
else
    echo "$a -lt 100 -o $b -gt 100 : returns false"
fi
```

运行结果：

```
10 != 20 : a is not equal to b
10 -lt 100 -a 20 -gt 15 : returns true
10 -lt 100 -o 20 -gt 100 : returns true
10 -lt 5 -o 20 -gt 100 : returns false
```

表8-6 布尔运算符列表

运算符	说明	举例
!	非运算，表达式为 true 则返回 false，否则返回 true。	[! false] 返回 true。
-o	或运算，有一个表达式为 true 则返回 true。	[$a -lt 20 -o $b -gt 100] 返回 true。
-a	与运算，两个表达式都为 true 才返回 true。	[$a -lt 20 -a $b -gt 100] 返回 false。

字符串运算符

示例

```
#!/bin/sh

a="abc"
b="efg"

if [ $a = $b ]
then
   echo "$a = $b : a is equal to b"
else
   echo "$a = $b: a is not equal to b"
fi

if [ $a != $b ]
then
   echo "$a != $b : a is not equal to b"
else
   echo "$a != $b: a is equal to b"
fi

if [ -z $a ]
then
   echo "-z $a : string length is zero"
else
```

```
    echo "-z $a : string length is not zero"
fi

if [ -n $a ]
then
    echo "-n $a : string length is not zero"
else
    echo "-n $a : string length is zero"
fi

if [ $a ]
then
    echo "$a : string is not empty"
else
    echo "$a : string is empty"
fi
```

运行结果：

```
abc = efg: a is not equal to b
abc != efg : a is not equal to b
-z abc : string length is not zero
-n abc : string length is not zero
abc : string is not empty
```

表8-7 字符运算符列表

运算符	说明	举例
=	检测两个字符串是否相等，相等返回 true。	[$a = $b] 返回 false。
!=	检测两个字符串是否相等，不相等返回 true。	[$a != $b] 返回 true。
-z	检测字符串长度是否为 0，为 0 返回 true。	[-z $a] 返回 false。
-n	检测字符串长度是否为 0，不为 0 返回 true。	[-z $a] 返回 true。
str	检测字符串是否为空，不为空返回 true。	[$a] 返回 true。

文件测试运算符

文件测试运算符用于检测 Unix 文件的各种属性。

例如，变量 file 表示文件“/var/www/tutorialspoint/unix/test.sh”，它的大小为 100 字节，具有 rwx 权限。下面的代码，将检测该文件的各种属性：

```
#!/bin/sh
```

```
file="/var/www/tutorialspoint/unix/test.sh"

if [ -r $file ]
then
   echo "File has read access"
else
   echo "File does not have read access"
fi

if [ -w $file ]
then
   echo "File has write permission"
else
   echo "File does not have write permission"
fi

if [ -x $file ]
then
   echo "File has execute permission"
else
   echo "File does not have execute permission"
fi

if [ -f $file ]
then
   echo "File is an ordinary file"
else
   echo "This is sepcial file"
fi

if [ -d $file ]
then
   echo "File is a directory"
else
   echo "This is not a directory"
fi

if [ -s $file ]
then
   echo "File size is zero"
else
   echo "File size is not zero"
fi

if [ -e $file ]
then
   echo "File exists"
```

```
else
    echo "File does not exist"
fi
```

运行结果：

```
File has read access
File has write permission
File has execute permission
File is an ordinary file
This is not a directory
File size is zero
File exists
```

表8-8 文件测试运算符列表

操作符	说明	举例
-b file	检测文件是否是块设备文件，如果是，则返回 true。	[-b $file] 返回 false。
-c file	检测文件是否是字符设备文件，如果是，则返回 true。	[-b $file] 返回 false。
-d file	检测文件是否是目录，如果是，则返回 true。	[-d $file] 返回 false。
-f file	检测文件是否是普通文件（既不是目录，也不是设备文件），如果是，则返回 true。	[-f $file] 返回 true。
-g file	检测文件是否设置了 SGID 位，如果是，则返回 true。	[-g $file] 返回 false。
-k file	检测文件是否设置了粘着位(Sticky Bit)，如果是，则返回 true。	[-k $file] 返回 false。
-p file	检测文件是否是具名管道，如果是，则返回 true。	[-p $file] 返回 false。
-u file	检测文件是否设置了 SUID 位，如果是，则返回 true。	[-u $file] 返回 false。
-r file	检测文件是否可读，如果是，则返回 true。	[-r $file] 返回 true。
-w file	检测文件是否可写，如果是，则返回 true。	[-w $file] 返回 true。
-x file	检测文件是否可执行，如果是，则返回 true。	[-x $file] 返回 true。
-s file	检测文件是否为空（文件大小是否大于 0），不为空返回 true。	[-s $file] 返回 true。

-e file	检测文件（包括目录）是否存在，如果是，则返回 true。	[-e $file] 返回 true。

8.2.6 shell 字符串

字符串是 shell 编程中最常用最有用的数据类型（除了数字和字符串，也没啥其它类型好用了），字符串可以用单引号，也可以用双引号，也可以不用引号。

单引号

```
str='this is a string'
```

单引号字符串的限制： 单引号里的任何字符都会原样输出，单引号字符串中的变量是无效的； 单引号字串中不能出现单引号（对单引号使用转义符后也不行）。

双引号

```
your_name='openEuler'
str="Hello, I know your are \"$your_name\"! \n"
```

双引号的优点： 双引号里可以有变量 双引号里可以出现转义字符

拼接字符串

```
your_name="openEuler"
greeting="hello, "$your_name" !"
greeting_1="hello, ${your_name} !"

echo $greeting $greeting_1
```

获取字符串长度

```
string="abcd"
echo ${#string} #输出 4
```

提取子字符串

```
string="huawei is a great company"
echo ${string:1:4}
```

查找子字符串

```
string="huawei is a great company"
echo `expr index "$string" is`
```

8.2.7 if else 语句

if 语句通过关系运算符判断表达式的真假来决定执行哪个分支。Shell 有三种 if ... else 语句：

- if ... fi 语句；

- if … else … fi 语句；
- if … elif … else … fi 语句。

if … else 语句

语法

```
if [ expression ]
then
   Statement(s) to be executed if expression is true
fi
```

if … else … fi 语句

语法

```
if [ expression ]
then
   Statement(s) to be executed if expression is true
else
   Statement(s) to be executed if expression is not true
fi
```

if … elif … fi 语句

语法

```
if [ expression 1 ]
then
   Statement(s) to be executed if expression 1 is true
elif [ expression 2 ]
then
   Statement(s) to be executed if expression 2 is true
elif [ expression 3 ]
then
   Statement(s) to be executed if expression 3 is true
else
   Statement(s) to be executed if no expression is true
fi
```

8.2.8 case 语句

case … esac 与其他语言中的 switch … case 语句类似，是一种多分枝选择结构。

case 语句匹配一个值或一个模式，如果匹配成功，执行相匹配的命令。case 语句格式如下：

```
case 值 in
模式 1)
```

```
    command1
    command2
    command3
    ;;
模式 2 ）
    command1
    command2
    command3
    ;;
*)
    command1
    command2
    command3
    ;;
esac
```

case 工作方式如上所示。取值后面必须为关键字 in，每一模式必须以右括号结束。取值可以为变量或常数。匹配发现取值符合某一模式后，其间所有命令开始执行直至 ;;。;; 与其他语言中的 break 类似，意思是跳到整个 case 语句的最后。

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，再执行后面的命令。

8.2.9 for 循环

与其他编程语言类似，Shell 支持 for 循环。

for 循环一般格式为：

```
for 变量 in 列表
do
    command1
    command2
    ...
    commandN
done
```

列表是一组值（数字、字符串等）组成的序列，每个值通过空格分隔。每循环一次，就将列表中的下一个值赋给变量。in 列表是可选的，如果不用它，for 循环使用命令行的位置参数。

例如，顺序输出当前列表中的数字：

```
for loop in 1 2 3 4 5
do
    echo "The value is: $loop"
done
```

运行结果：

```
The value is: 1
The value is: 2
The value is: 3
The value is: 4
The value is: 5
```

顺序输出字符串中的字符：

```
for str in 'This is a string'
do
    echo $str
done
```

运行结果：

```
This is a string
```

显示主目录下以 .bash 开头的文件：

```
#!/bin/bash

for FILE in $HOME/.bash*
do
   echo $FILE
done
```

运行结果：

```
/root/.bash_history
/root/.bash_logout
/root/.bash_profile
/root/.bashrc
```

8.2.10 while 循环

while 循环用于不断执行一系列命令，也用于从输入文件中读取数据；命令通常为测试条件。其格式为：

```
while command
do
   Statement(s) to be executed if command is true
done
```

命令执行完毕，控制返回循环顶部，从头开始直至测试条件为假。

8.2.11 脚本实例

if 语句

- 猜数字游戏

```
#!/bin/bash
a=3
```

```
b=$1
if [ $a == $b ]
then
   echo "You win!"
else
   echo "Please guess again."
fi
```

while + if 语句

- 判断位置参数是否是文件，是则输出文件内容，否则提示。

```
#!/bin/bash
while [ $1 ]
do
        if [ -f $1 ]
        then echo "display:$1"
                cat $1
        else echo "$1 is not a file name"
        fi
        shift
done
```

for 语句

- 从文件中读取每行信息，并输出

```
#!/bin/bash
for Name in $(cat ./namefile)
do
        echo $Name
done
```

case 语句

- 从键盘输入数据，并根据输入的数据选择性输出数据。

```
#!/bin/bash
echo 'Input a number between 1 to 4'
printf 'Your number is:\n'
read aNum
case $aNum in
    1)  echo 'You select 1'
    ;;
    2)  echo 'You select 2'
    ;;
    3)  echo 'You select 3'
    ;;
    4)  echo 'You select 4'
    ;;
    *)  echo 'You do not select a number between 1 to 4'
    ;;
```

```
esac
```

8.3 思考题

- 新建用户列表文件 **userlist**，在文件中按行写入 **10** 个用户名。新建一个 **useradd.sh** 脚本，要求该脚本能自动创建用户，用户名为 **userlist** 列表中的用户，密码为 **openEuler12#$**，新创建的用户家目录内包含一个 **hello.txt** 文件，**hello.txt** 文件的归属用户和私有组为该用户。

9 Samba 文件共享服务器管理

9.1 Samba 概述

1987 年，微软公司和英特尔公司共同制定了 SMB（Server Messages Block，服务器消息块）协议，旨在解决局域网内的文件或打印机等资源的共享问题，这也使得在多个主机之间共享文件变得越来越简单。到了 1991 年，当时还在读大学的 Tridgwell 为了解决 Linux 系统与 Windows 系统之间的文件共享问题，基于 SMB 协议开发出了 SMBServer 服务程序。这是一款开源的文件共享软件，经过简单配置就能够实现 Linux 系统与 Windows 系统之间的文件共享工作。当时，Tridgwell 想把这款软件的名字 SMBServer 注册成为商标，但却被商标局以 SMB 是没有意义的字符而拒绝了申请。后来 Tridgwell 不断翻看词典，突然看到一个拉丁舞蹈的名字—Samba，而且这个热情洋溢的舞蹈名字中又恰好包含了“SMB”，于是 Samba 服务程序的名字由此诞生。Samba 服务程序现在已经成为在 Linux 系统与 Windows 系统之间共享文件的最佳选择。

SAMBA 最初发展的主要目就是要用来沟通 Windows 与 Unix Like 这两个不同的作业平台，那么 SAMBA 可以进行哪些动作呢?

- 共享档案与打印机服务;
- 可以提供用户登入 SAMBA 主机时的身份认证，以提供不同身份者的个别数据;
- 可以进行 Windows 网络上的主机名解析 (NetBIOS name)
- 可以进行装置的分享 (例如 Zip, CDROM...)

9.1.1 Samba 服务器的基础设定

9.1.1.1 Samba 软件介绍

Samba 服务程序的配置方法与之前讲解的很多服务的配置方法类似，首先需要先通过 dnf 软件仓库来安装 Samba 服务程序（Samba 服务程序的名字也恰巧是软件包的名字）。Samba 包括的主要软件有:

- samba: 这个软件主要提供了 SMB 服务器所需的各项服务程序 (smbd 及 nmbd)、 的文件档、以及其他与 SAMBA 相关的 logrotate 配置文件及开机默认选项档案等;
- samba-client: 这个软件则提供了当 Linux 做为 SAMBA Client 端时，所需要的工具指令，例如挂载 SAMBA 文件格式的 mount.cifs、 取得类似网芳相关树形图的 smbtree 等等;
- samba-common: 这个软件提供的则是服务器与客户端都会使用到的数据，包括 SAMBA 的主要配置文件 (smb.conf)、语法检验指令 (testparm) 等等;

9.1.1.2 Samba 配置文件

Samba 相关的配置文件有：

- /etc/samba/smb.conf： 这是 Samba 的主要配置文件，基本上，咱们的 Samba 就仅有这个配置文件而已，且这个配置文件本身就是很详细的说明文件了，请用 vim 去查阅它吧！主要的设定项目分为服务器的相关设定(global)，如工作组、NetBIOS 名称与密码等级等， 以及分享的目录等相关设定，如实际目录、分享资源名称与权限等等两大部分。
- /etc/samba/lmhosts： 早期的 NetBIOS name 需额外设定，因此需要这个 lmhosts 的 NetBIOS name 对应的 IP 檔。 事实上它有点像是 /etc/hosts 的功能！只不过这个 lmhosts 对应的主机名是 NetBIOS name 喔！不要跟 /etc/hosts 搞混了！目前 Samba 预设会去使用你的本机名称 (hostname) 作为你的 NetBIOS name，因此这个档案不设定也无所谓。
- /etc/sysconfig/samba： 提供启动 smbd, nmbd 时，你还想要加入的相关服务参数。
- /etc/samba/smbusers： 由于 Windows 与 Linux 在管理员与访客的账号名称不一致，例如: administrator (windows) 及 root(linux)， 为了对应这两者之间的账号关系，可使用这个档案来设定
- /var/lib/samba/private/{passdb.tdb,secrets.tdb}： 管理 Samba 的用户账号/密码时，会用到的数据库档案;
- /usr/share/doc/samba-<版本>： 这个目录包含了 SAMBA 的所有相关的技术手册喔！也就是说，当安装好了 SAMBA 之后，系统里面就已经含有相当丰富而完整的 SAMBA 使用手册了。

9.1.2 Samba 服务端配置

9.1.2.1 smb.conf 文件配置

[global]项目

在 [global] 当中的就是一些服务器的整体参数了，包括工作组、主机的 NetBIOS 名称、字符编码的显示、登录文件的设定、 是否使用密码以及使用密码验证的机制等等，都是在这个 [global] 项目中设定的。至于 [分享资源名称] 则是针对你开放的目录来进权限方面的设定，包括谁可以浏览该目录、是否可以读写等等参数。 在 [global] 部分关于主机名信息方面的参数主要有：

- workgroup = 工作组的名称：注意，主机群要相同;
- netbios name = 主机的 NetBIOS 名称啊，每部主机均不同;
- server string = 主机的简易说明，这个随便写即可。

除此之外，还有登录文件方面的信息，包括这些参数：

- log file = 登录档放置的档案，文件名可能会使用变量处理;
- max log size = 登录档最大仅能到多少 Kbytes ，若大于该数字，则会被 rotate 掉。

还有安全性程度有关的密码参数，包括这几个：

- security = share, user, domain：三选一，这三个设定值分别代表：
- share：分享的数据不需要密码，大家均可使用 (没有安全性);
- user ： 使用 SAMBA 服务器本身的密码数据库，密码数据库与底下的 passdb backend 有关;
- domain：使用外部服务器的密码，亦即 SAMBA 是客户端之意，如果设定这个项目， 你还得要提供『password server = IP』的设定值才行;

分享资源的相关参数设定

分享名称内常见的参数有：

- [分享名称] ： 这个分享名称很重要，它是一个『代号』而已。
- comment ： 这个目录的说明。
- path ： 这个分享名称实际会进入的 Linux 文件系统 (目录)。而实际操作的文件系统则是在 path 里头所设定的。
- browseable ： 是否让所有的用户看到这个项目。
- writable ： 是否可以写入。
- writelist = 使用者, @群组，这个项目可以指定能够进入到此资源的特定使用者。 如果是 @group 的格式，则加入该群组的使用者均可取得使用的权限，设定上会比较简单！

因为分享的资源主要与 Linux 系统的档案权限有关，因此里头的设定参数多与权限有关。

smb.conf 内的可用变量功能

为了简化设定值，Samba 提供很多不同的变量给我们来使用，主要有底下这几个变量：

- %S：取代目前的设定项目值；
- %m：代表 Client 端的 NetBIOS 主机名喔！
- %M：代表 Client 端的 Internet 主机名喔！就是 HOSTNAME；
- %L：代表 SAMBA 主机的 NetBIOS 主机名；
- %H：代表用户的家目录；
- %U：代表目前登入的使用者的使用者名称；
- %g：代表登入的使用者的组名；
- %h：代表目前这部 SAMBA 主机的 HOSTNAME ；
- %I：代表 Client 的 IP ；
- %T：代表目前的日期与时间

9.2 Samba 文件共享服务器搭建实例

Samba 服务器搭建实例如下：

- 安装 samba 服务，及其相关组件。

```
[root@openEuler ~]# dnf -y install samba samba-client samba-common
```

- 启动 samba 服务，并设置为开机启动。

```
[root@openEuler ~]# systemctl start smb;systemctl enable smb
Created symlink /etc/systemd/system/multi-user.target.wants/smb.service →
/usr/lib/systemd/system/smb.service.
```

- 查看服务器监听状态，在 tcp 139,445 端口上监听。

```
[root@openEuler ~]# netstat -lantp
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:111             0.0.0.0:*               LISTEN      1276/rpcbind
tcp        0      0 192.168.122.1:53        0.0.0.0:*               LISTEN      3674/dnsmasq
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      5632/sshd
tcp        0      0 0.0.0.0:445             0.0.0.0:*               LISTEN      755432/smbd
tcp        0      0 0.0.0.0:44321           0.0.0.0:*               LISTEN      2585/pmcd
tcp        0      0 0.0.0.0:4330            0.0.0.0:*               LISTEN      739788/pmlogger
```

```
tcp        0      0 0.0.0.0:139             0.0.0.0:*               LISTEN      755432/smbd
tcp        0      0 192.168.110.246:22      172.19.130.180:60842    ESTABLISHED 753195/sshd: root [
tcp        0     64 192.168.110.246:22      172.19.130.180:56950    ESTABLISHED 686088/sshd: root [
tcp6       0      0 :::111                  :::*                    LISTEN      1276/rpcbind
tcp6       0      0 :::22                   :::*                    LISTEN      5632/sshd
tcp6       0      0 :::445                  :::*                    LISTEN      755432/smbd
tcp6       0      0 :::44321                :::*                    LISTEN      2585/pmcd
tcp6       0      0 :::4330                 :::*                    LISTEN      739788/pmlogger
tcp6       0      0 :::139                  :::*                    LISTEN      755432/smbd
```

- 查看防火墙状态是否开放，如果开放关闭防火墙。

```
[root@openEuler ~]# systemctl stop firewalld
[root@openEuler ~]# setenforce 0     #临时关闭 seLinux
```

- 配置共享访问用户，如 smb。

```
[root@openEuler ~]# useradd -s /sbin/nologin -M smb
```

- 设置用户 smb 的 samba 服务器密码，如 Huawei12#$。

```
[root@openEuler02 samba]# smbpasswd -a smb
New SMB password:
Retype new SMB password:
Added user smb.
```

- 创建共享文件目录

```
[root@openEuler ~]# mkdir /var/share /var/smb
[root@openEuler ~]# chmod 777 /var/share /var/smb
[root@openEuler ~]# chown smb:smb /var/smb
```

- 配置共享目录

```
[root@openEuler ~]# vim /etc/samba/smb.conf
#在 global 添加如下内容:
[global]
        workgroup = SAMBA
        security = user
        map to guest = Bad User                 #新增此行
        passdb backend = tdbsam

        printing = cups
        printcap name = cups
        load printers = yes
        cups options = raw
```

- 添加公共 share 目录，允许匿名访问目录。

```
[share]
        comment = share
        path = /var/share
        guest ok = yes
        writeable = yes
        browseable = yes
```

- 添加 smb 目录，配置共享用户访问目录。

```
[smb]
        comment = smb
        path = /var/smb
```

```
        write list = smb
        browseable = yes
        writeable = yes
        read list = smb
        valid users = smb
        create mask = 0777
        directory mask = 0777
```

- 配置完成后保存退出，然后重启 Samba 服务。

```
[root@openEuler ~]# systemctl restart smb
```

- 接下来就可以用 Windows 或其他安装了 cifs 访问客户端的 Linux 系统访问该共享了。

9.3 思考题

- 若在访问共享时需要私密性，又该如何配置?

  提示：可以配置 homedir。

参考文献/博文

《openEuler 操作系统》.任炬，张尧学，彭许红编著. ——北京：清华大学出版社，2020.9

《Linux 实用教程》.於岳编著. ——3 版. ——北京：人民邮电出版社，2017.1

https://openEuler.org/zh/

https://www.runoob.com/linux/linux-tutorial.html

https://www.cnblogs.com/xiangsikai/p/10683209.html

http://cn.linux.vbird.org/linux_server/0370samba.php

https://wiki.jikexueyuan.com/project/shell-tutorial/

https://juejin.cn/post/6844903938823553031